UNIVERSITY OF NOVI PAZAR
Department of Computer Sciences
Study Program - Informatics

Aybeyan Selimi

ALGORITHM OF COMPUTATIONAL GEOMETRY AND THEIR APPLICATION IN LINEAR OPTIMIZATION AND DYNAMIC PROGRAMMING

(Doctoral Dissertation)

Mentor: Prof. Dr Muzafer Saračević

Novi Pazar, 2019

Document type: Monograph
Language of text: English
The Language of Abstract: English
Country of Publication: Serbia
The locality of Publication: Serbia
Accepted by Scientific Board on:
Defendant on:

Defended Board

- **President:** Prof. Dr. Tatjana Atanasova Pačemska - Associate Professor, Department of Mathematics, Faculty of Computer Sciences, University " Goce Delcev" Stip, Macedonia
- **Member:** Prof. Dr. Šemsudin Plojović - Associate Professor, Department of Computer Sciences, University of Novi Pazar
- **Member, Mentor:** Prof. Dr. Muzafer Saračević - Associate Professor, Department of Computer Sciences, University of Novi Pazar

Scientific field: Computer Science
Scientific Discipline: Computational Geometry, Applied Mathematics

Acknowledgments

Firstly, I would like to thank Allah and all the people who have contributed and has participated in the success of this dissertation.

I have the special honor and pleasure to express my sincere appreciation to my mentor Prof Dr. Muzafer Saračević who supported and guided me during my studies. He has given the greatest contribution to the understanding of computational geometry as a discipline and with the numerous consultations, it was my biggest support in the preparation of this work. That's why my most sincere gratitude goes to him.

I would like to thank Prof Dr. Tatjana Atanasova Pačemska for her valuable comments which helped me improve the quality of the text in the thesis. Great thanks also to Prof Dr. Šemsudin Plojović for his support during my studies.

Let me express my best thanks to Prof Dr. Fadil Hoca rector of International Vision University and vice-rectors Prof Dr. Mensur Nuredin and Prof Dr. Abdulmecit Nuredin for the given support whole time of my studies. They enabled me to continue with my scientific research and encouragement me to finish this work.

I owe special gratitude to my wife Semra and my sons Erdem and Eren who were my biggest inspiration in the preparation of this dissertation. With their support and love, I managed to prepare this work. Finally, I thank my parents Emin and Emriye for their love, help, and support that give me all my life.

Priznanja

Prvo, želim se zahvaliti Allahu i svim ljudima koji su doprinijeli i sudjelovali u uspjehu ove disertacije.

Imam posebnu čast i zadovoljstvo da izrazim svoju najiskreniju zahvalnost mom mentoru prof. dr. Muzaferu Saračeviću koji me je podržao i vodio tokom studija. Dao je najveći doprinos u razumijevanju računarske geometrije kao discipline i uz brojne konsultacije, bio je moja najveća podrška u pripremi ovog rada. Zato i moja najiskrenija zahvalnost ide njemu.

Zahvaljujem se prof. dr Tatjani Atanasovi Pačemskoj na njenim vrijednim komentarima koji su mi pomogli da poboljšam kvalitet teksta u tezi. Veliko hvala i prof. dr. Šemsudinu Plojoviću na podršci tokom studija.

Dozvolite mi da se zahvalim rektoru Međunarodnog Univerziteta Vizion prof. dr. Fadil Hodza i prorektorima prof. dr. Mensura Nuredinija i prof. dr. Abdulmeđita Nuredinija za podršku koju su mi je pružali cijelo vrijeme u toku studija. Omogućili su mi da nastavim sa svojim naučnim istraživanjima i ohrabrili me da završim ovaj rad.

Posebnu zahvalnost dugujem mojoj supruzi Semri i mojim sinovima Erdemu i Erenu koji su bili moja najveća inspiracija u pripremi ove disertacije. Uz njihovu podršku i ljubav, uspio sam pripremiti ovaj rad. Na kraju, zahvaljujem se svojim roditeljima Eminu i Emriji za njihovu ljubav, pomoć i podršku koju mi daju cijeli moj život.

Abstract

The computational geometry as a discipline has a significant place and major importance in the technological development of engineering and is applied in different areas. As a branch of computer science is dedicated to the study of algorithms that can be expressed in terms of geometry. Some of these studies the purely geometric problems, while others are obtained as the corollary of examining computational geometric algorithms. Algorithms of computational geometry today are applied in numerical computation, geometric modeling, computer vision, computer graphics, geodesy, dynamic computing, isothetic computational geometry, and parallel computing. In this research, is given the procedure for the generation of cryptographic keys with algorithm of simple polygon triangulation and Catalan numbers, is constructed an algorithm for polygon triangulation based on the planted trivalent tree, is set up the minimum-weight triangulation algorithm based on matrix chain product and memoization and was analyzed the application of the computational geometry in linear optimization. This research describes in detail, the interaction between Catalan numbers, triangulation of convex polygon and one part of cryptography. The implementations are done in a Java environment and they are designed in the way to be efficient and easy to use.

Keywords: Computational geometry, Linear optimization, Catalan number, Cryptographic key and Polygon triangulation.

Sažetak

Računarska geometrija kao disciplina ima značajno mjesto i veliki značaj u tehnološkom razvoju inženjerstva i primjenjuje se u različitim područjima. Kao grana informatike posvećena je proučavanju algoritama koji se mogu izraziti u smislu geometrije. Neke od ovih studija su čisto geometrijski problemi, dok se drugi dobijaju kao posljedica ispitivanja računarskih geometrijskih algoritama. Algoritmi računarske geometrije danas se primjenjuju u numeričkom računanju, geometrijskom modeliranju, računarskom vidu, kompjuterskoj grafici, geodeziji, dinamičkom računanju, u izotetičkoj računarskoj geometriji i u paralelnom računanju. U ovom istraživanju dat je postupak za generisanje kriptografskih ključeva sa algoritmom jednostavne poligonske triangulacije i Katalonskih brojeva, konstruisan je algoritam za triangulaciju poligona zasnovan na zasađenom trivalentnom stablu, postavljen je algoritam za triangulaciju minimalne težine koji se temelji na proizvodu matričnog lanca i memoizaciji i analizirana je primjena računarske geometrije u linearnoj optimizaciji. Ovo istraživanje detaljno opisuje interakciju između Katalonskih brojeva, triangulacije konveksnog poligona i jednim delom kriptografije. Implementacije su izvršene u Java okruženju i dizajnirane su tako da budu efikasne i jednostavne za upotrebu.

Ključne reči: Računarska geometrija, Linearna optimizacija, Katalonov broj, Kriptografski ključ i Triangulacija poligona.

Contents

1 Resume of Dissertation

1.1 Statement of Results

The main purposes of the doctoral dissertation are to study the triangulation of polygons with Catalan numbers. The polygon triangulation process takes place with computational geometry algorithms. The results of this dissertation can be summarized as follows.

- We represent the procedure for applying computational geometry algorithms in the process of generating cryptological keys from one segment of the 3D image, where the Catalan keys are generated based on the combinatorial problem of the Balanced parenthesis and are applied in encryption of the segment of the image. Keys obtained with this method have a large keyspace, they provide an advantage in cryptanalysis and an effective mechanism for encryption and decryption of text.
- The models of linear optimization can be observed as problems of computational geometry and they have various application in different fields. Computational geometry develops efficient algorithms for optimizing of these models. Computer models can be developed based on objects that really exist or some imaginary object. In practice, experimenting with developed models is made with imaginary objects because experimenting with them is easier than with real objects. The prune and search algorithm represents an example relation between linear programming and computational geometry. It is, to our conclusion, that the linear optimization gives a good basis for further investigation in the low dimensional space treated in computational geometry.
- We developed a method for solving the problem of triangulation of a convex polygon from the aspect of generating graphical representation based on movements in planted a trivalent binary tree. Presented method was constructed on basis of two combinatorial problems: ballot record and lattice path. The movements in constructed method through polygon are derived upon vertices and leaves of the planted trivalent binary tree. For the polygon triangulation, is constructed two algorithms who are reverse to each other and transform the triangulation to ballot record and vice versa.
- We implemented our new method for finding and storing optimal triangulations of convex polygons in a Java environment. The algorithm presents a new method for an optimal triangulation based on the memoization, which preserves all results of triangulations for given n. The main emphasis of the method is on the speed of

generating optimal triangulation and saving memory space during the calculation of a large number of triangulations. The method of data storage is constructed on the idea of not calculating and storing each weight of triangulation particularly, but performing it all at once. It is important to notice that the implementation of our algorithm gives better results for saving and speed compared with the Hurtado-Noy and square matrix algorithm.

1.2 Outline of the Dissertation

The dissertation is subdivided into six chapters. The first chapter gives an introduction to computational geometry as a discipline of computer science and the problem of triangulation. Chapter two, three and four contain the theoretical background for the application and implementation work described in chapter five and six.

We start Chapter 1 with development and application areas of computational geometry as a branch of computer science. We describe the terms of diagonals and triangulation of polygon. We give the theoretical background of the principles of dynamic programming and the principles of optimality. Chapter 2 describes the Catalan numbers and their relationship with polygon triangulation, multiplication ordering and balanced parenthesis, correspondence between Catalan numbers – trees and Catalan numbers – lattice path. In Chapter 3 we present three techniques of dynamic programming. First is the technique of top-down and bottom-up approach of the recursive relation of the Fibonacci sequence. Second is the technique of memoization and third is matrix-chain multiplication algorithm. Chapter 4 describes the relationship between optimization and triangulation.

Chapter 5 presents five applications. The first one is a generation of cryptographic keys with the algorithm of simple polygon triangulation and Catalan numbers. It is used for ironing three-dimensional figures. The second one is an application of computational geometry in linear optimization. The third is convex polygon triangulation based on ballot problem and planted trivalent binary tree. We use the bijection between planted trivalent binary tree and triangulation of convex polygon in this application. The fourth application is memoization method for finding and storing of optimal triangulations in the convex polygons. In this method, we use the memoization and matrix-chain multiplication for data storage. Chapter six presents the implementation of the applications.

1.3 Introduction

The place and importance of mathematics in the development of science are very large. Arithmetic and geometry are two basic areas that form the basis of mathematics.

Geometry with considering some rules examines shapes that can be designed in the plane and in space. The etymological word "*geometry*" means the measurement of the earth. Geometry exists since ancient times. However, the name of the geometry began to be used by the Ancient Greeks, when geometry as a discipline became systematic. The original goal of geometry was to study shapes in a plane and in space. Although the shapes in nature can be derived from specific objects, geometry has left the use of experimental methods very early. On the contrary, she tried to reduce real objects to the ideal form for testing.

Calculating the surface, determining the orbits of celestial bodies, the ratio in geographical maps, machine building, and architecture are some of the areas in which geometry is applied. The first geometries due to their nature were created intuitively. These geometries are mostly visual. The second phase in the development of geometry began with the measurement of the shape that developed in Egypt, Mesopotamia and Ancient Greece [88]. The development of geometry continues with Classical Indian geometry, Chinese geometry, Islamic golden age geometry, Analytic geometry, Non-Euclidean geometry to Modern geometry [89]. In the second half of the 20th century by the appearance of computers, a new discipline is developed, called computational geometry.

Computational Geometry is a discipline of computer science that deals with the research of algorithms from the aspect of geometry. The first use of the term "Computational Geometry" dates back to 1975 and was used by M. I. Shamos [60]. The subject of research of the computational geometry is to solve geometric problems with the help of computers whereas the resultant of these research geometric algorithms that have wide application in various fields are obtained. Although a computational geometry has made significant progress today, it represents one of the early geometric areas that has been used to solve the practical problems of everyday life of people in the earliest times of humanity.

Algorithms of computational geometry contain a large number of points. These algorithms are encountered in very large datasets and are therefore of great importance in practical terms. The difference between these data sets is expressed as the difference $O\ (n^2)$ and $O\ (n\ log\ n)$. The goal of all researchs in computational geometry is to minimize the value of this difference.

At the center of development of the computational geometry as a discipline, are the problems of geometry that were present in computer graphics, computer-aided design, and manufacturing (CAD / CAM). Problems computational geometry basically are classical geometric problems that have been added with mathematical visualization and modeling in the computer science. Other important applications of computational geometry are seen in robotics (motion planning and visibility problems), in geographic information systems (geometric locations and search, motion planning), integrated circuit design, computer-aided engineering (CAE) (Mesh generation), computer vision. Computational geometry as a discipline with large domain today has the following sub-disciplines.

Combinatorial computational geometry, also called algorithmic geometry is the sub-disciple of computational geometry which considers geometric objects as separate objects.

Numerical computational geometry deals with the design of numerically robust algorithms for solving geometric problems [18], [51]. It has several different approaches to solve geometric problems. Edelsbrunner and Husk described the technique they designated as simulations of the simplicity of geometric algorithms [19].

Geometric modeling refers to the process of creating geometric models of real objects or dynamic processes that can be stored in a computer for the purpose of designing (CAD / CAM) or for simulation processes. There are various sub-problems in geometric modeling [53]. One of the most important problems in geometric modeling is to automatically generate a mesh inside a polygon. This is also a basic and necessary tool for solving the system of partial differential equations with the finite element method.

A *computer vision* analyzes the scene in the real world using an input device that is usually a kind of transducer (digital camera) and to arrive at a description of a scene that is useful for accomplishing of some task [90], [48].

Motion planning, linkages, and automated assembly are three problems of *robotics* in which computational geometry is applied. The problem of motion planning involves modeling robots as a polygon in two dimensions or a polyhedron in three dimensions that can move in space between an obstacle collection [3], [4], [91].

Art Gallery Theorems and Algorithms is a discipline that deals with the problems of viewing the walls of a large and complex gallery that is not in the shape of a star. The solution to this problem was given by the Canadian-Czech mathematician Václav Chvátal, who is known as "Chvátal's Art Gallery Theorem", also known as "Watchman Theorem" [10]. Avis and Toussaint in 1981 developed an efficient algorithm for finding locations where cameras should be mounted in the gallery's walls [6].

Computer graphics is an area where computational geometry provides faster algorithms for solving problems that are associated with hidden lines and hidden surface removal problems [15], [35], [90].

Geodesic computational geometry is an area where instead of Euclidean measure is used geodesic measure, i.e., the length of the shortest path (geodesic path) between two points that avoids the obstacles [28], [30].

Dynamic computational geometry is an area that deals with calculations in the convex hull with n points and the additional point p in the plane. The determination of a new convex hull with $n + 1$ points is done by modifying the existing convex hull of n points. First, it is tested whether the new point p lies in a convex hull of n points. If point p belongs to the convex hull of n points, then the obtained convex hull with n + 1 points is again convex and other calculations are not required. But first of all, in this process is need what's possible faster determine whether new point p belongs to the interior of the convex polygon with n sides. Determining whether the point p is in the interior of the convex polygon with n sides is given with algorithm of $O\ (n)$ computational complexity. Is possible to make this procedure much better? The answer is YES. This is done by storing the previous processing of the convex polygon in the corresponding data structure [61]. These data structures are called a dynamic data structure. The latest results on data structures for computational geometry are given in [42].

Parallel computational geometry is an area in which parallel computers are used. Parallel computers must communicate with each other in the parallel machine all time they use to find a solution to the geometric problem [2]. Instead of using one computer where primitive operations are performed one after the other, parallel computational geometry use a set of k computers, all of which work on the same problem at the same time. Each computer solves only part of the problem, and computers communicate with one another so as not to duplicate the work and "stitch" the partial solutions obtained from each computer into a complete solution. Algorithms used by parallel computers are different from those used by sequential

computers. Such algorithms are called parallel algorithms. An algorithm of primitive operations for finding a convex hull of n points have complexity $O\ (n3)$ unit of time. If $O\ (n3)$ computers with simple processors are used as a massively parallel computer then the convex hull of n points can be calculated in one unit of time [1]. The neural networks and optical computer geometry are domains of application of the parallel computational geometry [53]. The analysis of the complexity of optical algorithms is done with the appropriate definition of optical primitives [37].

Isothetic computational geometry or rectilinear calculus geometry is an area of computational geometry that deals with input data such as line segments and polygons in which all edges are vertical or horizontal [93].

1.4 Diagonals and Triangulations

Computational geometry is basically a discrete discipline. Problems of computational geometry are generally the subject of research in other fields of science such as "*geometric modeling*". Computational geometry explores simple and easily approximating surfaces and geometric objects. The basic terms that appear in this discipline are the point and line segment, which then builds more complex structures. Among the most important geometric figures of this discipline are the polygons in the plane, while in their space their generalization of polyhedra. Polygon is a closed geometric figure in a plane that is a finite collection of crossed line segments called the edges of the polygon. The points where the two edges meet are called vertices of the polygon. A set of all edges and vertices of a polygon is called the boundary of the polygon and is labeled with ∂P. In the Figure 1.1(a) is given polygon with nine edges joined at nine vertices. In the diagrams (b)–(d) are given objects that fail to be polygons. From Jordan curve theorem we know that the every simple closed planar curve separates the plane into a bounded interior region and an unbounded exterior. For this reason in this doctoral dissertation, polygons representing a special part of the Jordan curve theorem are analyzed. For a better continuity of the doctoral dissertation text, are given the following theorems and definition from [17 ,pp.2].

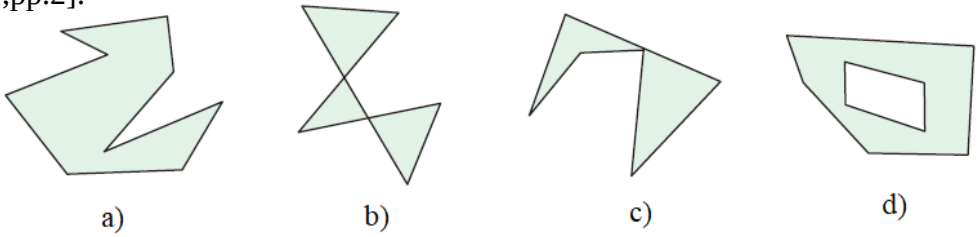


Figure 1.1. (a) A polygon. (b)–(d) Objects that are not polygons [17 ,pp.2].

Theorem 1.1 (**Polygonal Jordan Curve**) [17]. *The boundary ∂P of a polygon P partitions the plane into two parts. In particular, the two components of $\mathbb{R}^2 \setminus \partial P$ are the bounded interior and the unbounded exterior.*

Proof. Let *P* be a polygon in the plane. We first choose a fixed direction in the plane that is not parallel to any edge of *P*. This is always possible because *P* has a finite number of edges. Then any point *x* in the plane not on ∂P falls into one of two sets:

1. The ray through *x* in the fixed direction crosses ∂P an even number of times: *x* is exterior. Here a ray through a vertex is not counted as crossing ∂P.
2. The ray through *x* in the fixed direction crosses ∂P an odd number of times: *x* is interior.

Notice that all points on a line segment that do not intersect ∂P must lie in the same set. Thus the even sets and the odd sets are connected. And moreover, if there is a path between points in different sets, then this path must intersect ∂P.

Definition. A *triangulation* of a polygon *P* is a decomposition of *P* into triangles by a maximal set of noncrossing diagonals [17].

Here the word *maximal* has the mean that there is no other noncrossing diagonal which is in the set of triangulations of a geometric figure. Figure 1.3 shows three different triangulations of the polygon. There are several questions asked for triangulation of the polygon.

- What is the number of different triangulations of a given polygon?
- How many triangles consist each triangulation of a given polygon?
- Does every polygon always have a triangulation?
- Must each polygon have at least one diagonal?
- We start with the last question.

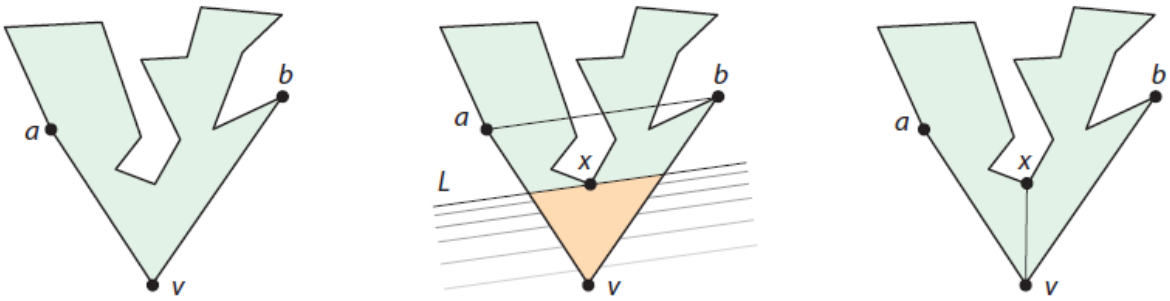


Figure 1.2. Finding a diagonal of a polygon through sweeping [17, pp.4].

Lemma 1.2 [17] *Every polygon with more than three vertices has a diagonal.*

Proof. Let v be the lowest vertex of P; if there are several, let v be the rightmost. Let a and b be the two neighboring vertices to v. If the segment ab lies in P and does not otherwise touch ∂P, it is diagonal. Otherwise, since P has more than three vertices, the closed triangle formed by a, b, and v contains at least one vertex of P. Let L be a line parallel to segment ab passing through v. Sweep this line from v parallel to itself upward toward ab; see Figure 1.4. Let x be the first vertex of the closed triangle abv, different from a, b, or v, that L meets along this sweep. The (shaded) triangular region of the polygon below line L and above v is empty of vertices of P. Because vx cannot intersect ∂P except at v and x, we see that vx is diagonal.

Theorem 1.3 [17] *Every polygon has a triangulation.*

Proof. We prove this by induction on the number of vertices n of the polygon P. If $n = 3$, then P is a triangle and we are finished. Let $n > 3$ and assume the theorem is true for all polygons with fewer than n vertices. Using Lemma 1.2, find a diagonal cutting P into polygons P_1 and P_2. Because both P_1 and P_2 have fewer vertices than n, P_1 and P_2 can be triangulated by the induction hypothesis. By the Jordan curve theorem (Theorem 1.1), the interior of $P1$ is in the exterior of P_2, and so no triangles of P_1 will overlap with those of P_2. A similar statement holds for the triangles of P_2. Thus P has a triangulation as well.

We know that every polygon has at least one triangulation. Next, we show that the number of triangles in any triangulation of a fixed polygon is the same. The proof is essentially the same as that of Theorem 1.4, with more quantitative detail.

Theorem 1.4 [17] *Every triangulation of a polygon P with n vertices has* $n - 2$ *triangles and* $n - 3$ *diagonals.*

Proof. We prove this by induction on n. When $n = 3$, the statement is trivially true. Let $n > 3$ and assume the statement is true for all polygons with fewer than n vertices. Choose a diagonal d joining vertices a and b, cutting P into polygons P_1 and P_2 having n_1 and n_2 vertices, respectively. Because a and b appear in both P_1 and P_2, we know $n_1 + n_2 = n + 2$. The induction hypothesis implies that there are $n_1 - 2$ and $n_2 - 2$ triangles in P_1 and P_2, respectively. Hence P has $(n_1 - 2) + (n_2 - 2) = (n_1 + n_2) - 4 = (n + 2) - 4 = n - 2$ triangles. Similarly, P has $(n_1 - 3) + (n_2 - 3) + 1 = n - 3$ diagonals, with the +1 term counting d.

In many algorithms and proofs, a special triangle should be included which is often used in the start of recursion or initial induction. The place of these special triangles in computational geometry is often used "*ears*". Three consecutive vertices *a*, *b*, *c* form an ear of a polygon if *ac* is a diagonal of the polygon. The vertex *b* is called the ear tip.

Corollary 1.5 [17] *Every polygon with more than three vertices has at least two ears.*
Proof. Consider any triangulation of a polygon P with $n > 3$ vertices, which by Theorem 1.4 partitions P into $n - 2$ triangles. Each triangle covers at most two edges of ∂P. Because there are n edges on the boundary of P but only $n - 2$ triangles, by the pigeonhole principle at least two triangles must contain two edges of P. These are the ears.

1.5 Concrete Applications of Computational Geometry

It is important to note that there is a correlation between the triangulation of the polygon and the decomposition of Catalan numbers see [85].

Block method is the process of generating the triangulation of a polygon using a set of triangles. The general strategy used in the Block method is to break the main problem into fewer problems that are mutually dependent. Each problem is solved only once and is used several times, thus avoiding unnecessary repetition of the same calculations and is suitable for recursion with memoization. This allows convenience in the practical application of this method. Based on the experimental results of paper [28], from examining the Java application we see that the average execution time at one level in the Hurtado-Noy algorithm is 46.16 while for the block method is 10.74. All this significantly affects the load on the memory during execution, and the speed of generating triangulation.

In the paper [84] was processed, a triangulation from the aspect of notation and their storage. These methods have been presented with the aim of saving memory space. A connection has been made between polygon triangulation problems and combinatorial problems, such as the ballot problem and the lattice path. The first notation technique is called *ballot notation*. The second notation is called *alfa-numeric* (AN). The experimental results obtained in this paper show that the use of these notations achieves significant memory savings in the process of storage of triangulations. In this paper, has been introduced the possibility of using a stack structure in the storage of polygon triangulation.
Polygon triangulation also is a complex problem that requires complex classes for efficient object-oriented implementation. Object-Oriented Analysis and Design (OOAD) provides a

comprehensive insight into the implementation of triangulation problems. The paper [73] gives approaches to the OOAD methodology for the polygon triangulation problem. The problem of triangulation in this paper is analyzed from three aspects, where each approach is defined by the appropriate type of engineering of this methodology: forward engineering, reverse engineering, and round-trip engineering. An important result of this paper is an improvement achieved by applying the reverse engineering and synchronization of the UML model and Java source code. There are also some advantages for all three approaches, with particular emphasis on improving implementation by applying synchronization in forwarding engineering and reverse engineering approaches.

Triangulation as an important aspect of computational geometry is also a concept that is applied in computer graphics. Basically, the polygon triangulation in computer graphics provides a three-dimensional view of images from a set of points. One of the basic goals of applying triangulation is what is used as a replacement for the storage of nodes and the distribution of internal polygon diagonals. Efficiency is achieved at the rate of triangulation recording in the Java buffer (cache) used to store interim results [68].

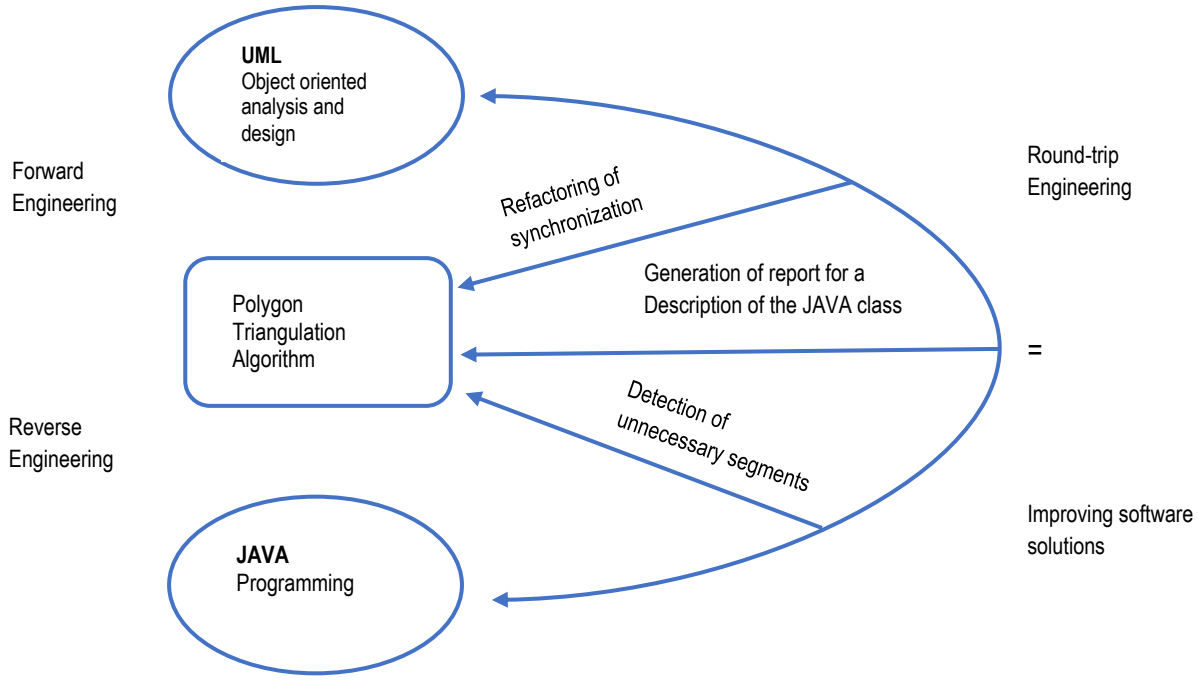


Figure 1.3 Three object oriented analysis and design approaches [source author]

The process of recording convex polygonal triangulations is related to binary trees that in the computer sciences represent the structure for data storage. In the paper [68], a Java implementation of the Lukasiewicz algorithm for triangulation of polygon based on binary trees is shown. This application enables efficient graphic representation of triangulation. The result of the application is a binary notation of a triangulation of a convex polygon that can be used to detect some regularities in generating triangulations of larger polygons.

Triangulation as a procedure can be used in the presentation of a three-dimensional object from a set of points and provides a mechanism for the so-called "glazing" of these three-dimensional objects. As a procedure it is very important for speed in computer graphics, it provides quality and good resolution for the object. Triangulation of convex polygons is a question that arises in two-dimensional computational geometry [69]. One of the good software solutions for triangulation is the optional Final Surface plug-in triangulation that offers the possibility of generating network triangles from cloud points whose application in the paper [69] has been shown in the reconstruction of the human brain. The alignment of the brain surface is determined by the correspondence of 3D points between pairs in the surface. This algorithm is based on geodetic distance combinations and curvature of the surface see Figure 1.4.

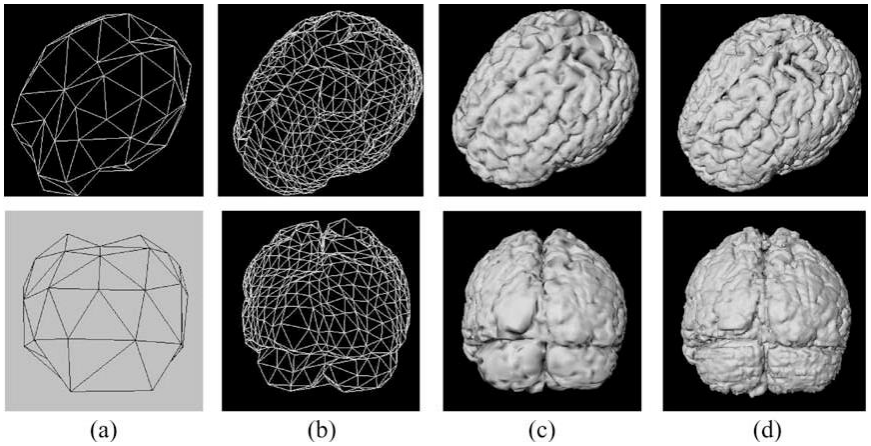


Figure 1.4 Brain surface reconstruction

Triangulation as a procedure also is applied in the duplex ultrasonic scanner for quantifying the speed of blood flow to two dimensions. In this procedure is applied in obtaining of surface triangulation from the three-dimensional image and in the triangular networks that are used to share the image in several non-overlapping regions and have similar characteristics.

Most CAD / CAM objects today are made using some form of automatic production. Computers as modern multimedia appliances play an important role in the design and construction of these moving and non-moving objects. The process of constructing a particular object depends on the construction material, the shape and the quantity that is produced. The

application of computational geometry in linear optimization is seen in the aspects of the production of plastic and metal objects. The process of producing metal objects is called casting. In Figure 1.5, the process of casting liquid metal into the mold is given.

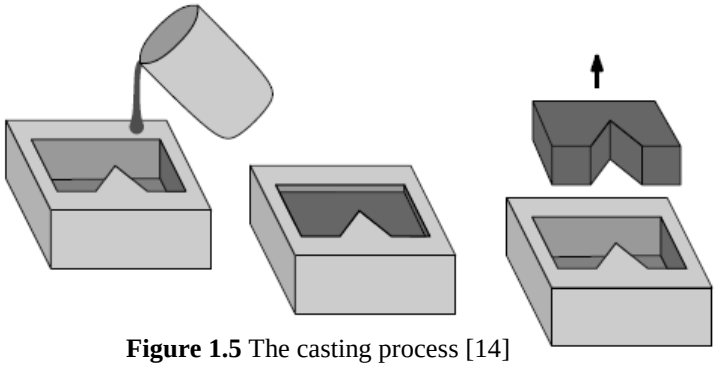


Figure 1.5 The casting process [14]

The casting process consists of three parts, the casting of liquid metal into the mold, the solidifies of liquid metal and the removal of the mold from the mold. In the process of manufacturing, removing the object from the mold is not always easy because the object can be stuck in the mold and may be required the breaking of mold. There are also objects for which it is very difficult to make a mold. These are generally objects that have winkled surfaces, such as spheres, ellipsoids, and so on. In order to avoid such complications in the manufacturing process, are making certain constraints. Due to this, in most of the cases in manufacturing are taken the objects which are polyhedral.

A greater number of casting uses a mold that consists of only one part and from which object can be removed only with one movement. For the production of objects by casting, one must first determine whether its shape is suitable for production by casting and how to make the appropriate mold to it. The shape of the object determines the cavity in the mold, from which it is understood that different molds must be made for the production of different objects.

Orientation is one of the most important traits in the process of removing objects from the mold. It is not possible to remove the object from the mold with the wrong orientation. Each object must have a horizontal upper side in the orientation, which is the only side that is not in contact with the mold. There are various potential orientations of objects, depending how many facets the object has and for this reason, several different molds are being built in the process of production for these objects. The process of removing a particular object from the mold is a process that works with linear optimization. This is the process where we must find the solution to the problem where the object is castable. Today in the manufacturing are used the algorithms of incremental linear programming and randomized linear programming.

Point Location

One other application of computational geometry can be seen in finding a point location in a map. Point location problem occurs in various situations of today. It is used in traffic on the water into determining the current position of a particular floating object. Point location problems also occur in interactive geographic systems that show the map on the screen [14]. These systems provide information about the location that is selected with the mouse. In the process, the point location terrain is divided into subdivision *S* with *n* edges and are looking for the answer to the following question is asked: Is the query point *q* in the face *f* of *S*. This can be arrived withdrawing the vertical lines through all vertices of subdivision where the plane is divided to vertical slabs.

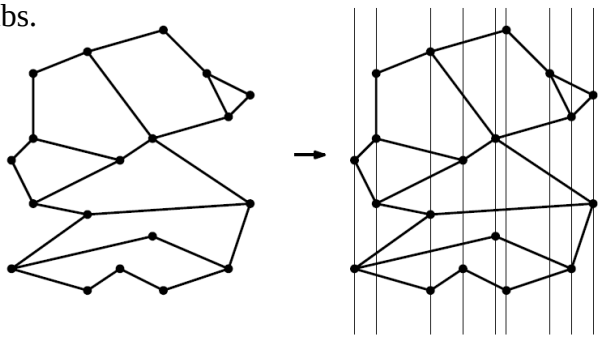


Figure 1.6 Partition into slabs [14]

By sorting in the array the *x*-coordinates of vertices is determined a slab that contains the query point *q*. This is achieved with binary search and shows which segment is below *q*. If there is no segment below *q*, then *q* lies in the unbounded face of *S*.

A trapezoidal map is another decomposition that facilitates point location query. The time complexity of this decomposition is not much greater than the complexity of the original subdivision and fulfills several desirable properties in the point location problem. Trapezoidal maps are defined as sets of n non-crossing segments in the plane.

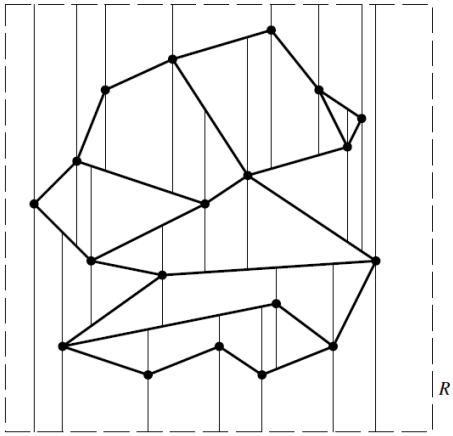


Figure 1.7 Trapezoidal maps [14]

The application of trapezoidal maps in a point location problem is realized by releasing of unbounded trapezoid faces which are at the boundary. This is done by introducing a large axis-parallel rectangle R that contains whole scenes and with the assumption that there are no two different points that have the same x-coordinates, i.e. there are no two points lies on the same vertical line. A set of these properties is a set of lines in a general position. Trapezoidal maps of planar division S is simply a subdivision of S, rectangle R, and lower and upper vertical extensions. For the construction of trapezoidal maps, is used the randomized incremental algorithm [14].

Robot Motion Planning

One of the main goals in robotics is the design of autonomous robots, which can plan their movements. For planning the movement, we must have information about the obstacles and the environment in which the robot will move. The construction of robots with such characteristics is very rare and in practice more parts are used which are called robot arms. Each robot arms consists of several connected joints and is generally used to manipulate and assemble part of the items that are being manufactured.

In order to achieve this goal, robots in the process of moving a certain movement must move from one position to another. This robot movement is known as the motion planning problem. To define this problem, it is necessary to make several assumptions, for example, the environment in which the robot is moving, are the other objects static, can be represented the planar region with polygonal obstacles, on the road in which the robot is moving no people. Different movements that the robot can do depend on its mechanics. Some robots can move in any direction, while others are limited in their movements [14]. The movement of a robot into a 2-dimensional environment is done with translation and rotation in relation to the reference point $R(0,0)$.

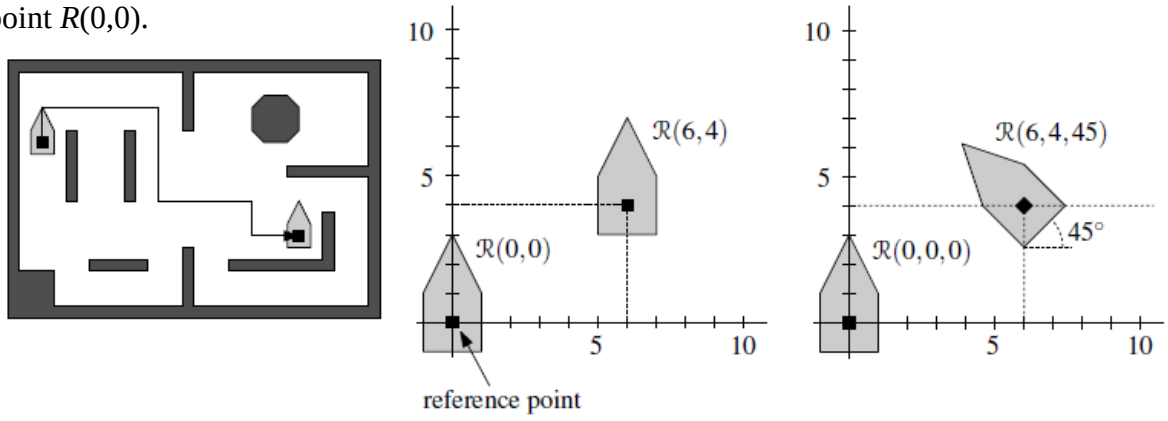


Figure 1.8 Robot motion planning [14]

For the robot movement also are used the trapezoidal maps. These maps are used to calculate the representation of the free space which finds the path for each starting point and goal position. The problem of robot motion planning can also be planned with the rotations defined by Minkowski sums [14].

1.6 Principles of Dynamic Programming

Dynamic programming DP is a discipline that encompasses a set of elaborated and adopted optimization procedures in which complex and/or difficult problems are divided into a number of small easy solved problems that are system components. The basic principle of dynamic programming is the "principle of optimality", a principle that is not influenced by the past processes and enables the most optimal and the most appropriate decision making. One problem of dynamic programming can be solved in two ways. The first way is the Forward recursion procedure, which solves the problem starting from the smallest units at the beginning of the problem and continuing towards the end. Backward recursion is a second way of solving problems. With this procedure, the problem is solved starting from the last point of the problem and continuing toward the beginning of the problem.
This method was developed by Richard Bellman and is called as a Bellman's Principle of Optimality [44].

The optimality principles also in the literature are known as the property of "optimal substructure". For DP to be computationally efficient (especially relative to evaluating all possible sequences of decisions), there should be common subproblems such that subproblems of one are subproblems of another. In this case, the solution of subproblem is needed only be found once and reused as often as necessary [44, pp.5].

Keeping in mind then that the main objective of this investigation is to examine dynamic programming and its role as a mathematical programming method, the problem definition that will suit to the purposes is to find an expression way to the various types of optimization problems that will concern the field of computational geometry. This expression way, without assuming an unduly complex format, it will have to be broad enough to encompass the largest variety of problems possible. Suppose that is considered the problem of DP using the following formulation:

$$\textit{Problem } P^{\circ}: \quad p^{*} := \operatorname*{opt}_{x \in X} q(x)$$

where q is a real-valued function on some set X and $:=$ denotes definition. With q is denoted the objective function, X is the solution set or decision space, and x is the decision variable.

It is important to note that, if the constituent elements of a DP problem (solution set X and the objective function q) are not well defined the problem becomes too abstract for any analysis or treatment. The implication, therefore, is to give to DP problem structure description properties for X and q. And furthermore, given our subject of interest, these properties must be defined so as to render *Problem P° a dynamic programming problem*. But this implies, in turn, that quest for properties for X and q would have to be guided by a pre-existing conception of what a dynamic programming problem is [84, p .14].

When developing a dynamic programming algorithm, we follow a sequence of four steps [12, pp. 359]:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Steps 1–3 form the basis of a dynamic-programming problem solution. If is needed only the value of an optimal solution, and not the solution itself, then can be omit step 4. When we do perform step 4, we sometimes maintain additional information during step 3 so that we can easily construct an optimal solution [12, pp. 359].

2 Catalan Numbers

Catalan numbers are a special sequence of numbers used to solve many problems in the combinatorics. These interesting numbers were first encountered by Leonhard Euler (1703-1783) and Johann Andreas von Segner (1704-1777), by studying the problem of triangulation of the convex polygon. The problem was first resolved by Euler in 1760, who by using combinatorial tools obtain the number of triangulations in the convex polygon. Recursive relations of this numbers are introduced from Segner, while many of the properties and identities of these numbers by studying well-formed sequences of parentheses in 1838 are discovered from Eugene Charles Catalan (1814-1894). In his honor, these numbers today are called Catalan numbers. These numbers were completely independent discovered from Chinese mathematician Ming An-Tu (1692-1763) in 1730. Ming's work was published in Chinese, so it was not known in the West long time.

Definition 2.1 *Catalan numbers* C_n *are integer sequence defined by*

$$C_n = \frac{(2n)!}{(n+1)!\,n!} = \frac{1}{n+1}\binom{2n}{n}, \quad n \geq 0 \tag{2.1}$$

Table 2.1 The first 30 Catalan numbers

n	C_n	n	C_n	n	C_n
1	1	**11**	58,786	**21**	24,466,267,020
2	2	**12**	208,012	**22**	91,482,563,640
3	5	**13**	742,900	**23**	343,059,613,650
4	14	**14**	2,674,440	**24**	1,289,904,147,324
5	42	**15**	9,694,845	**25**	4,861,946,401,452
6	132	**16**	35,357,670	**26**	18,367,353,072,152
7	429	**17**	129,644,790	**27**	69,533,550,916,004
8	1,430	**18**	477,638,700	**28**	263,747,951,750,360
9	4,862	**19**	1,767,263,190	**29**	1,002,242,216,651,360
10	16,796	**20**	6,564,120,420	**30**	3,814,986,502,092,300

There is an alternative way of defining C_n:

$$\binom{2n}{n} - \binom{2n}{n-1} = \frac{(2n)!}{(n!)^2} - \frac{(2n)!}{(n-1)!\,(n+1)!} = \frac{(2n)!}{n!\,(n+1)!} = C_n \tag{2.2}$$

2.1 Triangulation of convex $n-$gon

We will denote the convex polygon with *n* sides with P_n. It is described by a series of vertices $v_1, v_2, \dots, v_n$. The inner diagonal connecting the strings v_i and v_j is marked with δ_{ij}. Also, and the polygon sides are considered as diagonals, so also $\delta_{i,i+1}$ represents the side $v_i v_{i+1}$. Set of triangles of the maximal way of decomposition of a convex polygon P_n to the $n-2$ triangles is denoted with T_n. For triangulation is necessary to draw $n-3$ non-intersect diagonals. It is not hard to see, one diagonal convex polygon divides it into two parts, two to three, and so on by induction.

The problem of triangulation considered with induction starts with the triangle. Since the triangle is already triangulated, there is only one way of triangulation and therefore $T_3 = 1$. For a square ($n = 4$) we can draw one diagonal. This can be done in two ways (because the square has two diagonals) so it is $T_4 = 2$.

For the pentagon ($n = 5$) solution is less obvious, there are 5 ways of triangulation. Let us now look at the general solution for the number T_n of triangulation of $n-$gon. Note that each $n-$gon side is a part of the triangle in a triangulation. For counting of different triangulations of a convex polygon, we will use recursion and the following way. Randomly select and fix one particular side and call him the base. We count the triangulations in which each of the triangles has the base as a side. For the $k-$point as the tip of the triangle, the left is $(n-k+1)-$gon, which we can triangulate in T_{n-k+1} ways, and the left $k-$gon that we can triangulate in T_k ways (see Figure 1). We define $T_2 = 1$.

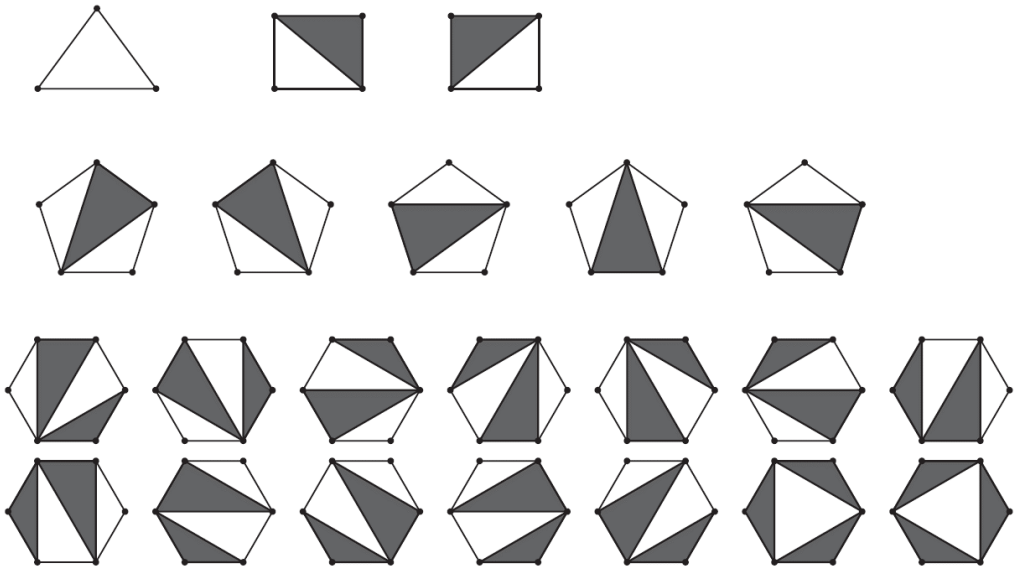


Figure 2.1 Triangulations of an *n*-gon, where $3 \leq n \leq 6$ [40, pp.107]

In 1761, Segner, using the addition and multiplication principles, published a second order recurrence formula for T_n, where $n \geq 3$ and present it to the St. Petersburg Academy [40, pp. 114]. Since the triangulation choices of the isolated polygons are independent of each other, the combinatorial principle of the product is valid and for the selected point to this number is equal to $T_k T_{n-k+1}$ see Figure 2.2.

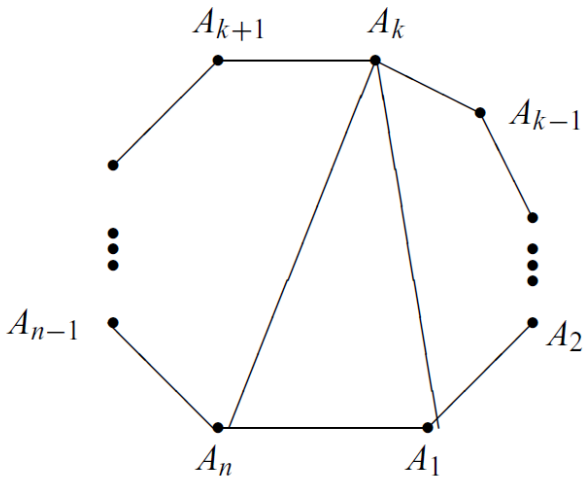


Figure 2.2 Polygonal Dissection [40, pp.116]

The choice of this point can also be done in several (independent) ways, so we still have to go through all the possible values of k. The final form of Segner's Recursive Formula T_n is:

$$T_n = \sum_{k=2}^{n-1} T_k T_{n-k+1}$$
$$= T_2 T_{n-1} + T_3 T_{n-2} + \cdots + T_{n-1} T_2 , \quad n \geq 3 \tag{2.3}$$

Euler established a relationship between the triangulation of polygons and Catalan numbers with the following formula:

$$T_n = \frac{2 \cdot 6 \cdot 10 \cdot \cdots \cdot (4n-10)}{(n-1)!} , \quad n \geq 3 \tag{2.4}$$

If we take $k = n - 3$ then the formula has the form

$$T_{k+3} = \frac{2 \cdot 6 \cdot 10 \cdot \cdots \cdot (4k+2)}{(k+2)!} , \quad k \geq 0 \tag{2.5}$$

Then $T_3 = 1, T_4 = 2$, and $T_5 = 5$. These are the Catalan numbers C_1, C_2, and C_3, respectively.

Thus, for the problem of polygon triangulation we see that holds following equality:

$$T_{n+2} = C_n \ , \qquad n \geq 1 \tag{2.6}$$

where *n* is the number of polygon vertices. From this equation can be written

$$C_n = \frac{2 \cdot 6 \cdot 10 \cdot \cdots \cdot (4n-2)}{(n+1)!} \ , \qquad n \geq 1 \tag{2.7}$$

and rewritten as

$$\begin{aligned} C_n &= \frac{4n-2}{n+1} \cdot \frac{2 \cdot 6 \cdot 10 \cdot \cdots \cdot (4n-2)}{(n+1)!} \\ &= \frac{4n-2}{n+1} \cdot C_{n-1} \end{aligned} \tag{2.8}$$

Since $C_n = T_{n+2}$, this yields *Segner's recurrence relation* for C_n:

$$C_n = C_0 C_{n-1} + C_1 C_{n-2} + \cdots + C_{n-1} C_0 \tag{2.9}$$

Therefore, the value of the Catalan number C_{n-2} determines the number of triangulations corresponding to the polygon P_n [76, pp.5]. Based on the formula for obtaining the values of Catalan numbers (1.1) we can define value T_n, for $n \geq 3$:

$$\begin{aligned} T_n &= \frac{1}{n-1} \cdot \binom{2n-4}{n-2} \\ &= \frac{(2n-4)!}{(n-1)!\,(n-2)!} \cdot \end{aligned} \tag{2.10}$$

The value of T_{n+2} also can be expressed equivalent form to (2.9):

$$\begin{aligned} T_{n+2} &= \frac{2}{n+1} \cdot \binom{2n-1}{n} \\ &= \frac{1}{n+1} \binom{2n}{n} = C_n \end{aligned} \tag{2.11}$$

2.2 Basic Properties of the Catalan Numbers

In this section are given others approximations, recursive relations and properties of the Catalan numbers. An approximate value of C_n can be calculated using *Stirling's approximation for n*! , $n! \approx (n/e)^n\sqrt{2\pi n}$:

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2}$$

$$\approx \frac{(2n/e)^{2n}\sqrt{2\pi \cdot 2n}}{(n/e)^{2n} \cdot 2\pi n}$$

$$= \frac{2^{2n}}{\sqrt{n\pi}}$$

So

$$C_n = \frac{2^{2n}}{(n+1)\sqrt{n\pi}}$$

$$\approx \frac{2^{2n}}{n\sqrt{n\pi}}$$

For $n = 5$ we obtain

$$\frac{2^{2n}}{n\sqrt{n\pi}} = \frac{2^{10}}{5\sqrt{5\pi}}$$

$$\approx 52$$

whereas $C_5 = 42$. When $n = 10$:

$$\frac{2^{2n}}{n\sqrt{n\pi}} = \frac{2^{20}}{10\sqrt{10\pi}}$$

$$\approx 18\,708$$

whereas $C_{10} = 16\,796$ [40, pp.111]. From (2.8) follows

$$\lim_{n\to\infty} \frac{C_{n+1}}{C_n} = 4$$

Thus, when *n* is sufficiently large, $C_{n+1} \approx 4C_n$.

The following formulas are a direct consequence of Formula (2.1).

Corollary 2.1 *For the Catalan numbers C_n, we have the following relations:*

1) $C_n = \frac{1}{n+1}\sum_{k=0}^{n}\binom{n}{k}^2$

2) *Nonlinear recurrence relation* $C_{n+1} = \frac{2(2n+1)}{n+2} = C_n\,,\ C_0 = 1$

3) *Asymptotic expression* $\frac{C_n}{\frac{4^n}{\sqrt{\pi}n^{3/2}}} \longrightarrow 1 \quad as \quad n \longrightarrow \infty.$

Proof. 1) From (2.1) we know that

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

and the formula $\binom{m+n}{r} = \sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}$ gives

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{1}{n+1}\sum_{k=0}^{n}\binom{n}{k}^2.$$

2)
$$\begin{aligned}\frac{C_{n+1}}{C_n} &= \frac{(2n+2)!}{(n+2)[(n+1)!]}\cdot\frac{(n+1)(n!)^2}{(2n)!}\\ &= \frac{(2n+2)(2n+1)(n+1)}{(n+2)(n+1)^2}\\ &= \frac{2(2n+1)(n+1)^2}{(n+1)^2(n+2)}\end{aligned}$$

$$= \frac{2(2n+1)}{(n+2)}$$

$$C_{n+1} = \frac{2(2n+1)}{(n+2)} C_n.$$

3) Stirling's approximation to *n*! is

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

and is obtained

$$C_n = \frac{1}{n+1}\frac{(2n)!}{n!\,n!} \sim \frac{1}{n+1}\frac{2\sqrt{\pi n}\left(\frac{2n}{e}\right)^{2n}}{2\pi n\left(\frac{n}{e}\right)^{2n}} \sim \frac{1}{n+1}\frac{1}{\sqrt{\pi n}}\ 4^n \sim \frac{1}{n}\frac{1}{\sqrt{\pi n}}\ 4^n = \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}}$$

Definition 2.2 *The gamma function for complex variable z with a positive real part is defined by*

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

Using integration parts, we can show that

$$\Gamma(z) = (z-1)\Gamma(z-1) \qquad (2.12)$$

If z is a positive integer, $\Gamma(n) = (n-1)!$, so $\Gamma(n+2) = (n+1)!$.

Euler's reflection formula for gamma function Γ and $\forall z \in Z$ is

$$\Gamma(1-z) = \frac{\pi}{sin\pi z}$$

where it follows $\Gamma(1/2) = \sqrt{\pi}$.

By solving recursion (2.11), we get the following expression for C_n:

$$\Gamma\left(n+\frac{1}{2}\right) = \left(n-\frac{1}{2}\right)\Gamma\left(n-\frac{1}{2}\right)$$

$$= \left(n-\frac{1}{2}\right)\left(n-\frac{3}{2}\right)\Gamma\left(n-\frac{3}{2}\right)$$

$$\vdots$$

$$= \left(n-\frac{1}{2}\right)\left(n-\frac{3}{2}\right)\cdots\left(n-\frac{2n-1}{2}\right)\Gamma\left(\frac{1}{2}\right)$$

$$= \frac{1\cdot 3\cdot 5\cdots(2n-1)}{2^n}\sqrt{\pi}$$

$$= \frac{(2n)!}{2^n(2\cdot 4\cdot 6\cdots n)}\sqrt{\pi}$$

$$= \frac{(2n)!}{4^n n!}\sqrt{\pi}$$

So

$$\frac{\Gamma\left(n+\frac{1}{2}\right)}{\Gamma(n+2)} = \frac{(2n)!}{4^n n!\,(n+1)!}\sqrt{\pi}$$

$$= \frac{C_n}{4^n}\sqrt{\pi}$$

Thus

$$C_n = \frac{4^n\Gamma\left(n+\frac{1}{2}\right)}{\sqrt{\pi}\Gamma(n+2)} \tag{2.13}$$

Theorem 2.2 *Let* T_n *denote the number of triangulations of an* $n-$*gon, where* $n \geq 4$. *Then*

$$(2n-6)T_n = n\sum_{k=3}^{n-1} T_k T_{n-k+2} \tag{2.14}$$

Proof. Is given in [40, pp:117 -118]

Corollary 2.3 *By substitution* $C_n = T_{n+2}$ *in the formula (2.14) for* $n \geq 2$ *we have*

$$(2n-2)C_n = (n+2)\sum_{k=1}^{n-1} C_k C_{n-k} \tag{2.15}$$

2.3 Generating Function and Integral Representation of Catalan Number

For the given sequence of numbers $a_0, a_1, a_2, \ldots, a_n, \ldots$ it can be formed the series

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

This series is the generating function for the sequence. The generating function allows to perform calculations with the terms in the sequence, and thus to give easier ways to prove certain identities [91].

The generating function for the Catalan numbers is

$$C(x) = C_0 + C_1 x + C_2 x^2 + C_3 x^3 + \cdots + C_n x^n + \cdots \tag{2.16}$$

where C_n, is the n-th Catalan number. By quadrating this expression

$$[C(x)]^2 = C_0^2 + (C_0 C_1 + C_1 C_0)x + (C_0 C_2 + C_1 C_1 + C_2 C_0)x^2 + \cdots$$

$$+(C_0 C_n + C_1 C_{n-1} + \cdots + C_n C_0\ )x^n + \cdots$$

$$= C_1 + C_2 x + C_3 x^2 + \cdots + C_{n+1} x^n + \cdots$$

$$= \frac{C(x) - C_0}{x}$$

Then

$$x[C(x)]^2 - C(x) + 1 = 0$$

Solving this equation for variable $C(x)$,

$$C(x) = \frac{1 \pm \sqrt{1 - 4x}}{2x}$$

Because C_n is always a positive integer, in the solution of the quadratic equation we must choose the minus sign, otherwise, the coefficients of the powers of x in the generating function $C(x)$ all can be negative. Thus, generating function $C(x)$ is

$$C(x) = \frac{1 - \sqrt{1 - 4x}}{2x}$$

From the binomial formula for the coefficient of x^n (if $n > 0$) in the series we have

$$C_n = -\frac{1}{2}\left\{\frac{\frac{1}{2}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\cdots\left(-\frac{(2n-3)}{2}\right)}{n!}(-4)^n\right\}$$

$$= \frac{1}{2}\left\{\frac{\frac{1}{2}\left(\frac{1}{2}\right)\left(\frac{3}{2}\right)\cdots\left(\frac{(2n-3)}{2}\right)(n-1)!}{n!\,(n-1)!}(2^2)^n\right\}$$

$$= \frac{1}{2}\left\{\frac{\frac{1}{2}\left[\left(\frac{1}{2}\right)\cdot 1\left(\frac{3}{2}\right)\cdot 2\cdots(n-2)\left(\frac{(2n-3)}{2}\right)(n-1)\right]}{n!\,(n-1)!}(2^2)^n\right\}$$

$$= \frac{1}{2}\left\{\frac{\frac{1}{2}[1\cdot 2\cdot 3\cdot\cdots\cdot(2n-4)(2n-3)(2n-2)]}{n!\,(n-1)!}2^2\right\}$$

$$= \frac{(2n-2)!}{n!\,(n-1)!}$$

$$= \frac{1}{n}\binom{2n-2}{n-1}$$

In order to demonstrate the relation of Catalan numbers with the continued fractions, we will use a generating function equation

$$x[C(x)]^2 - C(x) + 1 = 0$$

$$C(x)\begin{bmatrix}1 - xC\\(x)\end{bmatrix} = 1$$

With pre-grouping of the expression and exclude *C*(*x*)

$$C(x) = \frac{1}{1 - xC(x)}$$

Thus, with the iterative enumeration of the above expression for *C* (*x*) in the denominator is obtained by the continued fraction

$$C(x) = \cfrac{1}{1 - \cfrac{x}{1 - \cfrac{x}{1 - \cfrac{x}{1 - \cfrac{x}{1 - \cfrac{x}{1 - \frac{x}{\cdots}}}}}}}$$

For example, let's check this by developing the continued fraction for *n* = 6

$$1 + x + 2x^2 + 5x^3 + 14x^4 + 42x^5 + 132x^6 + O[x]^7.$$

We see that the coefficients in this development are the Catalan numbers.

2.4 Catalan Numbers and Multiplication Ordering/ Balanced Parenthesis

Suppose we have n numbers to multiply together, meaning that there are $n - 1$ multiplications to perform. By changing only the multiplication order, without changing the order of the numbers themselves this numbers can be multiplied in many orders. Multiplication is a binary operation and therefore in the multiplication of the n numbers first we have to multiply the two, then multiply their multiplicity by the next element and so on. Multiplication is ended when all numbers are multiplied. To show the multiplication ordering explicitly, we will use parenthesis. In Table 2.2 are given the possible multiplication orderings for $0 \leq n \leq 4$ multiplications. The multiplications are indicated by dots and multiplication groups with parentheses in Table 2.2.

n	Multiplications	*Ways*
0	(a)	1
1	$(a \cdot b)$	1
2	$((a \cdot b) \cdot c),\ (a \cdot (b \cdot c))$	2
3	$(((a \cdot b) \cdot c) \cdot d),\ ((a \cdot b) \cdot (c \cdot d)),\ ((a \cdot (b \cdot c)) \cdot d),$ $(a \cdot ((b \cdot c) \cdot d)),\ (a \cdot (b \cdot (c \cdot d)))$	5
4	$((((a \cdot b) \cdot c) \cdot d) \cdot e),\ (((a \cdot b) \cdot c) \cdot (d \cdot e)),\ (((a \cdot b) \cdot (c \cdot d)) \cdot e),$ $((a \cdot b) \cdot ((c \cdot d) \cdot e)),\ ((a \cdot b) \cdot (c \cdot (d \cdot e))),\ (((a \cdot (b \cdot c)) \cdot d) \cdot e),$ $((a \cdot (b \cdot c)) \cdot (d \cdot e)),\ ((a \cdot ((b \cdot c) \cdot d)) \cdot e),\ ((a \cdot (b \cdot (c \cdot d))) \cdot e),$ $(a \cdot (((b \cdot c) \cdot d) \cdot e)),\ (a \cdot ((b \cdot c) \cdot (d \cdot e))),\ (a \cdot ((b \cdot (c \cdot d)) \cdot e)),$ $(a \cdot (b \cdot ((c \cdot d) \cdot e))),\ (a \cdot (b \cdot (c \cdot (d \cdot e))))$	14

Table 2. 2 Multiplication Arrangements

The problem of balanced parenthesis is very similar to the problem of the multiplication order. Let have a string of parentheses which are counted from left to right. The question is - How many parentheses we can set up, such that we don't set more closed than open parenthesis at the same time? We have to note that we can not set up parenthesis arbitrarily. Suppose we have *n* pairs of parentheses and we want to form valid groupings of them, where "***valid***" means that each open parenthesis has a matching closed parenthesis. "**()(())**" is example for valid grouping. How many groups can be formed for each value of n? How can we multiply n numbers in different multiplication order? Let the number of multiplications is marked with P_n.

Let S denote the set of well-formed sequences with *n* pairs of parentheses. Then S can be defined recursively, where the terminal clause is omitted for convenience:

- $\lambda \in S$.
- If $x, y \in S$, then $(x), xy \in S$, where xy denotes the concatenation of the symbols x and y.

n	Correctly Parenthesized Expressions P_n					P_n
0	λ					1
1	()					1
2	()()	(())				2
3	()()()	(()())	(())()	()(())	((()))	5
4	()()()() ()(())() ((()))()	((()))() ()()(() (()())()	(())(()) (()())() ((())())	()((())) ()(()()) ((()()))	(())()() (()()())	14

Table 2. 3 Well-Formed Sequences with *n* Pairs

Let λ is selected such that $x\lambda = x = \lambda x$ for all $x \in S$. Now we can develop the recursive formula for P_n. Suppose $n \geq 2$ and let $0 \leq i \leq n-1$. By definition is $P_0 = 1 = P_1$. The first i pairs can be correctly grouped in P_i ways and the remaining $n-1-i = n-i-1$ pairs in P_{n-i-1} ways. Using the multiplication principle, these two events can take place together in $P_i\, P_{n-i-1}$ different ways [40, pp. 134]. Because this is true for each value of *i*, by the addition principle,

$$P_n = \sum_{i=0}^{n-1} P_i P_{n-i+1}$$

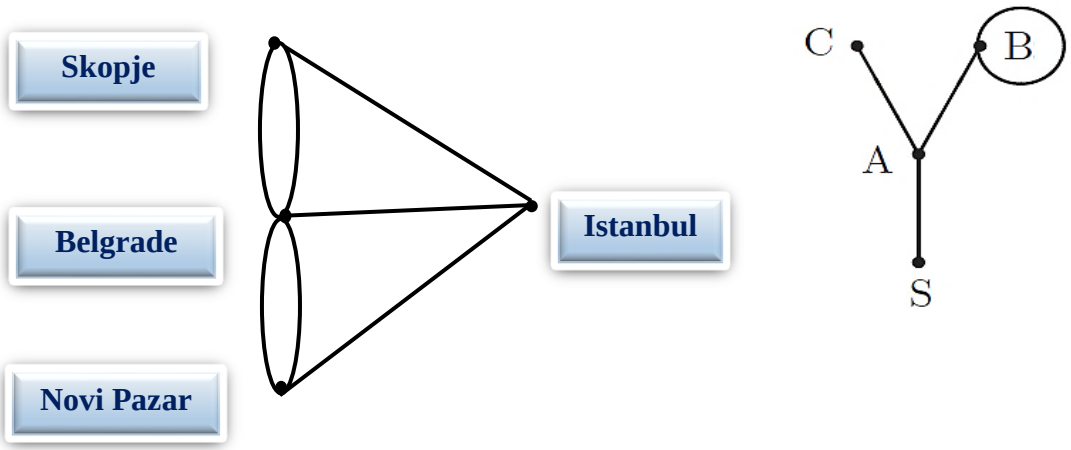
From this equation, we see that P_n satisfies the same recurrence as a C_n , thus $P_n . = C_n$.

2.5 Definitions and theorems that are used in research

In this section are presented the relations between the Catalan numbers and the trees. Graph theory is a mathematical discipline that analyzes graphs. The graph is a kind of network structure consisting of vertices and edges connecting these vertices. Graph models today are used in the solution of problems in various scientific disciplines. They have application in computer networks, social networks, communication networks, information networks, transportation networks, biological networks, tournaments etc. For more clarity and precision, let's start with the definition of some basic terms from graph theory.

Definition 2.3 *A graph G = (V, E) consists of V, a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints* [65, pp. 641].

Let V denote the set of vertices and E the set of edges. An edge between vertices v and w is denoted by $v - w$ or $\{v, w\}$. For example, let suppose that a network is made up of data centers and communication links between computers. The location of each data center can be represented by a point and each communications link by a line segment, (Figure 2.7).



a) ***b)***

Figure 2.7 Graph Examples: a) Computer Network b) Social network

The graph in Figure 2.7 – a) has four vertices *Skopje*, *Belgrade*, *Novi Pazar*, and *Istanbul* and seven edges. The edges with the same vertices are called *parallel* edges. A loop is the edge that starts from and ends at the same vertex. The loop in Figure 2.7– *b*) is on to B.

The non-simple graphs in which are permitted both graph loops and multiple edges are called *pseudographs*. A graphs in the Figure 2.7 are undirected graphs. Many times in practice is needed construction of a graph model with directions. For example, in a computer network, single duplex lines operate in only one direction. This may be the case if there is a large amount of traffic sent to some data centers, with little or no traffic going in the opposite direction [65, pp. 643]. In the modeling of such computer network are used directed graphs, where each edge is associated with an ordered pair.

Definition 2.4 *A directed graph* (V, E) *consists of a nonempty set of vertices* V *and a set of directed edges (or arcs)* E. *Each directed edge is associated with an ordered pair of vertices.*

The directed edge associated with the ordered pair (u, v) is said to start at u and end at v. A directed graph also is called digraph [65, pp. 643].

For analyzing of graph problem that can be drawn in the plane so that no two of its edges cross we will give the following definitions.

Definition 2.5 *Two vertices u and v in an undirected graph G are called adjacent (or neighbors) in G if u and v are endpoints of an edge e of G. Such an edge e is called incident with the vertices u and v and e is said to connect u and v* [65, pp. 651].

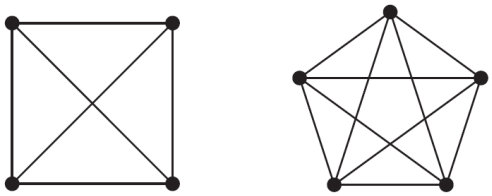
Definition 2. 7 *The set of all neighbors of a vertex v of $G = (V, E)$, denoted by $N(v)$, is called the neighborhood of v. If A is a subset of V, we denote by $N(A)$ the set of all vertices in G that are adjacent to at least one vertex in A. So, $N(A) = \bigcup_{v \in A} N(v)$* [65, pp. 652].

Definition 2.8 *The degree of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex v is denoted by $deg(v)$* [65, pp. 652].

Definition 2. 9 *When (u, v) is an edge of the graph G with directed edges, u is said to be adjacent to v and v is said to be adjacent from u. The vertex u is called the initial vertex of (u, v), and v is called the terminal or end vertex of (u, v). The initial vertex and terminal vertex of a loop are the same* [65, pp. 654].

Definition 2.10 *In a graph with directed edges the in-degree of a vertex v, denoted by $deg-(v)$, is the number of edges with v as their terminal vertex. The out-degree of v, denoted by $deg+(v)$, is the number of edges with v as their initial vertex* [65, pp. 654].

A graph in which each edge connects two different vertices and where no two edges connect the same pair of vertices is called a *simple graph* [65, pp. 642]. A *complete graph on n vertices*, is a simple graph that contains exactly one edge between each pair of distinct vertices and is denoted by K_n. The Figure 2.8 shows the complete graphs K_4 and K_5.



K_4 K_5

Figure 2.8 Complete graphs [65, pp. 655]

A *cycle* C_n, $n \geq 3$, is a path with the same endpoints consists of n vertices $v_1, v_2, \ldots, v_n$ and edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$ and $\{v_n, v_1\}$. A graph is *acyclic* if it contains no cycles.

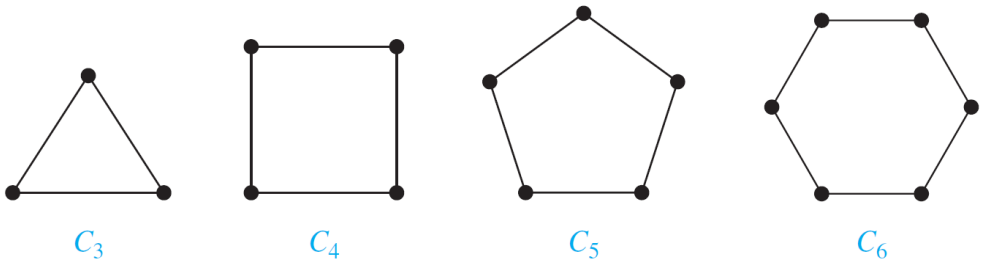
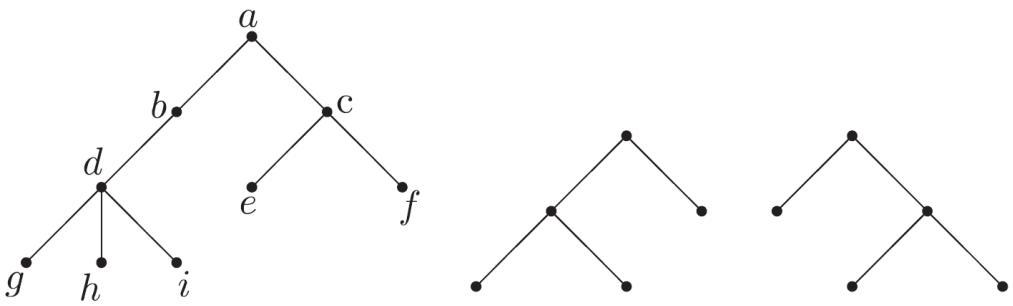


Figure 2.9 The cycles C_3, C_4, C_5 and C_6 Complete graphs [65, pp. 655]

Definition 2. 11 *Let n be a nonnegative integer and G an undirected graph. A path of length n from u to v in G is a sequence of n edges* $e_1, \ldots, e_n$ *of G for which there exists a sequence* $x_0 = u, x_1, \ldots, x_{n-1}, x_n = v$ *of vertices such that* e_i *has, for* $i = 1, \ldots, n$, *the endpoints* x_{i-1} *and* x_i *. When the graph is simple, we denote this path by its vertex sequence* $x_0, x_1, \ldots, x_n$ *(because listing these vertices uniquely determines the path). The path is a circuit if it begins and ends at the same vertex, that is, if* $u = v$, *and has length greater than zero. The path or circuit is said to pass through the vertices* $x_0, x_1, \ldots, x_{n-1}$ *or traverse the edges* $e_1, \ldots, e_n$. *A path or circuit is simple if it does not contain the same edge more than once* [65, pp. 679].

Definition 2. 12 *A tree is a connected undirected graph with no simple circuits* [65, pp. 746].

An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices [65, pp. 746]. The graph in Figure 2.10 - a) is tree, but the one in Figure 2.10 - b) is not.



a) ***b)***

Figure 2.10 Examples of the trees [40, pp. 229].

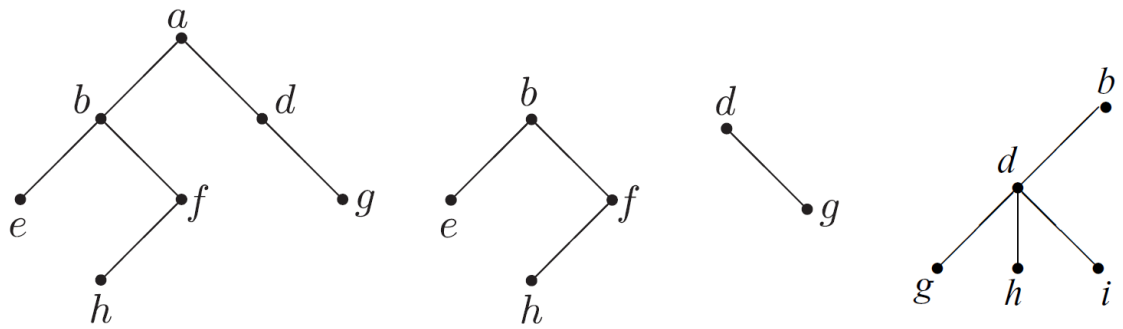
The tree in Figure 2.10 – a) contains a particular vertex, called the *root*. A tree together with its root produces a directed graph and called a *rooted tree*.

Definition 2. 13 *A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root. A set of trees is forest* [65, pp. 747].

A rooted tree in which the vertices at each level are ordered as the first, second, third, and so on is called ordered rooted tree. Different ordered trees are the two trees in Figure 2.10 - *b*).

Rooted trees with the property that all of their internal vertices have the same number of children have applications in the triangulation of the polygons.

Definition 2.14 *A rooted tree is called an m-ary tree if every internal vertex has no more than m children. The tree is called a full m-ary tree if every internal vertex has exactly m children. An m-ary tree with* $m = 2$ *is called a binary tree* [65, pp. 747].



a) ***b)*** ***c)***

Figure 2.11 Binary tree [40, pp. 230].

The tree in Figure 2.11 – a) is a binary tree; its *left subtree* is the binary tree rooted at b and its *right subtree* is the binary tree rooted at d .The tree in Figure 2.11 – c is not a binary tree.

Theorem 2.4 *A tree with n vertices has* $n - 1$ *edges.*

Proof. Is given in [65, pp. 752].

Theorem 2.4 *A full m-ary tree with i internal vertices contains* $n = mi + 1$ *vertices.*

Proof. Is given in [65, pp. 752].

Theorem 2.5 *The number of binary trees with n vertices is* C_n.

Proof. Is given in [40, pp. 230-231].

A *full binary tree* is a binary tree in which each internal vertex has exactly two children. The next theorem gives the correspondence between full binary trees and Catalan numbers [40, pp. 232].

Theorem 2.6 *The number of full binary trees with n vertices is* $C_{\frac{n-1}{2}}$ *, where n is odd.*

Proof. Is given in [40, pp. 233].

Theorem 2.6 establishes a bijection between the set of full binary trees with n vertices and the set of binary trees with $\frac{n-1}{2}$ vertices and satisfy the following equation:

Number of full binary trees with n vertices	=	**Number of binary trees with $\frac{n-1}{2}$ vertices.** $C_{\frac{n-1}{2}} = \frac{2}{n+1}\binom{n-1}{\frac{n-1}{2}}$

Definition 2. 15 *Let* T *be an ordered rooted tree with root* r. *If* T *consists only of* r, *then* r *is the preorder traversal of* T. *Otherwise, suppose that* $T_1, T_2, \ldots, T_n$ *are the subtrees at* r *from left to right in* T. *The preorder traversal begins by visiting* r. *It continues by traversing* T_1 *in preorder, then* T_2 *in preorder, and so on, until* T_n *is traversed in preorder* [65, pp. 773].

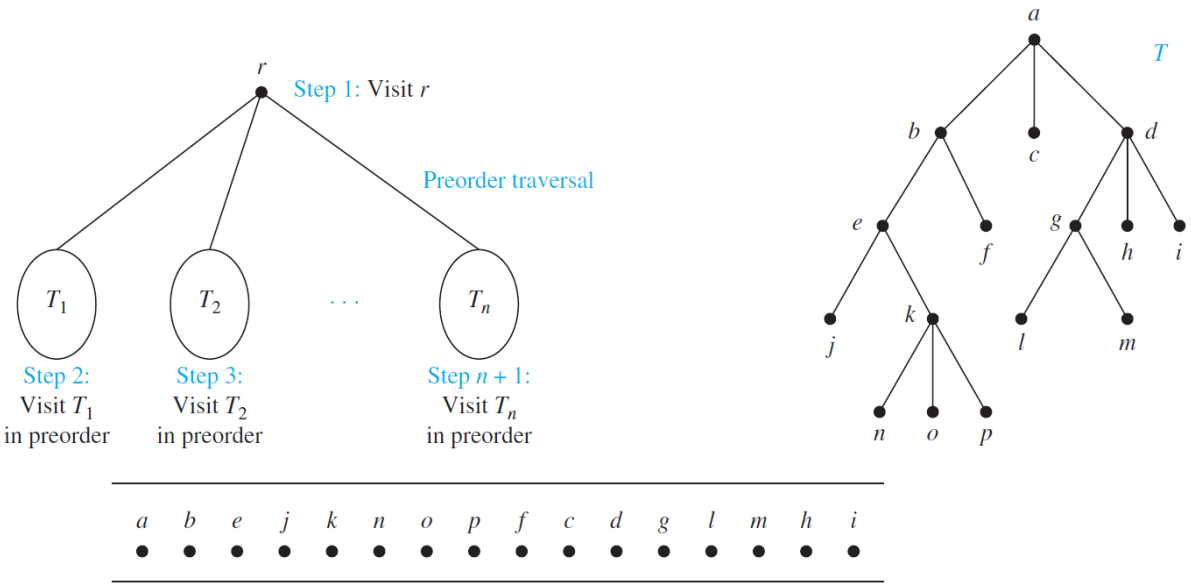


Figure 2.14 The preorder traversal of T [65, pp. 774]

ALGORITHM 2.1 Preorder Traversal.

procedure *preorder*(*T* : ordered rooted tree)
r := root of *T*
list *r*
for each child *c* of *r* from left to right
    T (c) := subtree with *c* as its root
  preorder(T (c))

Using the recursive *preorder tree traversal algorithm* of a binary tree can be constructed a bijection between set of full binary trees with $2n + 1$ vertices and the set of sequences of n $1s$ and n $-1s$ such that every partial sum (from left to right) is nonnegative; that is, the number of $2n$-tuples $a_1 a_2 \dots a_{2n}$ of $1s$ and $-1s$ such that $a_1 + a_2 + \dots + a_k \geq 0$ and $a_1 + a_2 + \cdots + a_{2n} = 0$, where $1 \leq k < 2n$ and $n \geq 0$. The values of the possible sequences for $0 \leq n \leq 4$ are given in the table below.

Table 2.3 Values of sequence for $0 \leq n \leq 4$

n	Possible Sequences	Count
0	λ	1
1	1 -1	1
2	1-11-1 11-1-1	2
3	1-11-11-1 11-11-1-1 11-1-11-1 1-111-1-1 111-1-1-1	5
4	1-11-11-11-1 111-1-1-11-1 11-1-111-1-1 1-1111-1-1-1 11-1-11-11-1 1-111-1-11-1 1-11-111-1-1 11-11-1-11-1 1-111-11-1-1 1-1111-1-1-1 111-1-1-11-1 11-111-1-1-1 111-1-11-1-1 1111-1-1-1-1	14

Definition 2.16 *Let T be an ordered rooted tree with root r. If T consists only of r, then r is the inorder traversal of T. Otherwise, suppose that $T_1, T_2, \dots, T_n$ are the subtrees at r from left to right. The inorder traversal begins by traversing T_1 in inorder, then visiting r. It continues by traversing T_2 in inorder, then T_3 in inorder, . . . , and finally T_n in inorder* [65, pp. 775].

ALGORITHM 2.2 Inorder Traversal

procedure *inorder* (*T* : ordered rooted tree)
r := root of *T*
if *r* is a leaf then list *r*
else
l := first child of *r* from left to right
T (l) := subtree with *l* as its root
inorder(T (l))
list *r*
for each child *c* of *r* except for *l* from left to right
T (c) := subtree with *c* as its root
inorder(T (c))

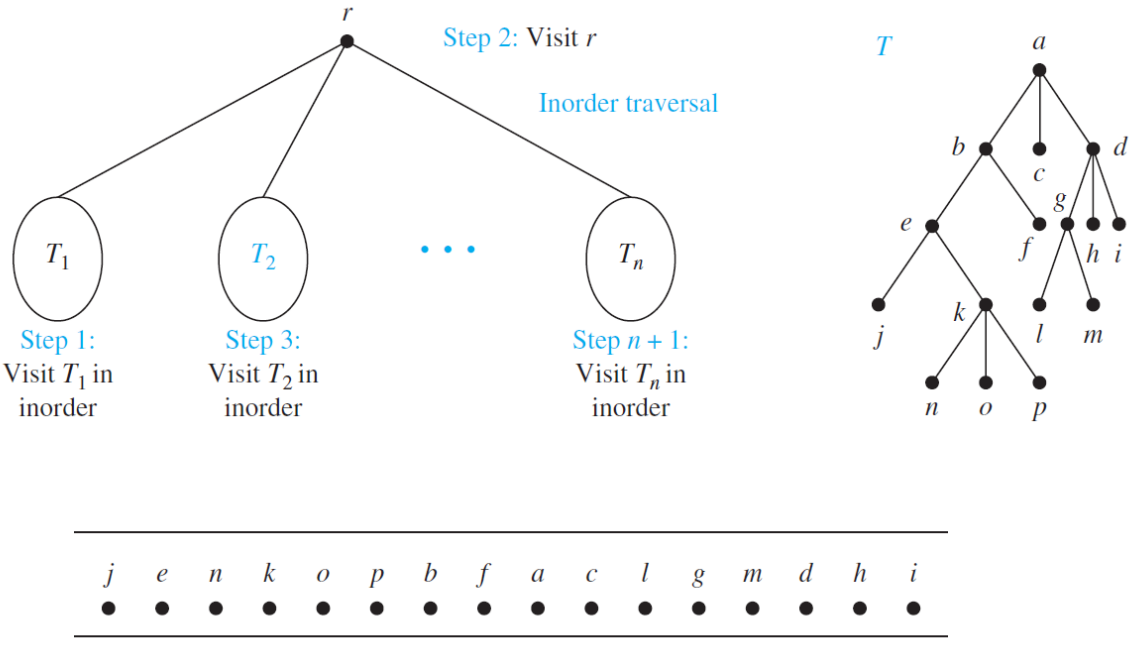
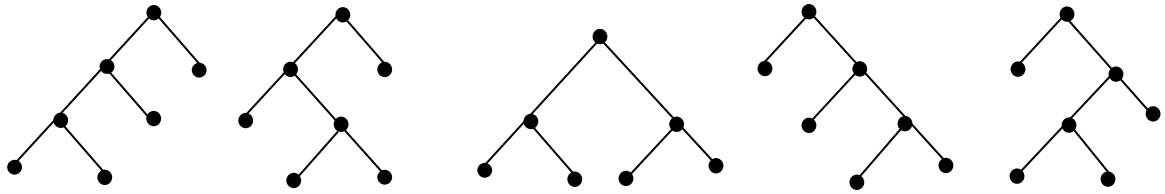


Figure 2.15 The inorder traversal of *T* [65, pp. 776]

We record 1 every time when is traversed the left edge, and -1 when is traversed the right edge. Let's look the sequences in the 3rd row of Table 2.2. Using the preorder traversal, they yield the full binary trees on Figure 2.15, respectively.



111-1-1-1 11-11-1-1 11-1-11-1 1-11-11-1 1-111-1-1

Figure 2.16 The binary trees of 3rd rows

2.6 Triangulation and Binary Trees

A binary tree with the degree of root one and each internal vertex three is called the planted trivalent binary tree [40, pp. 235]. There is a bijection between the set of binary trees and the set of planted binary trees. By attaching a new root at the existing root of a binary tree, we get a planted binary tree, by deleting the root of a planted trivalent binary tree, we get an ordinary binary tree. It means every planted binary tree with n vertices correspondent to the binary tree with $n-1$ vertices. Thus, from (2.1) formula we have:

Number of planted trivalent binary trees with n vertices. = **Number of binary trees with n - 1 vertices.**

= C_{n-1}

= $\frac{1}{n}\binom{2n-2}{n-1}$

In section 2.4 we see that there is a bijection between the set of triangulations of an $n-$gon and the set of balanced parentheses with n pairs, where each containing C_n elements. From the equation between the number of planted trivalent binary trees with *n* vertices and the number of binary trees with *n* - 1 vertex, we can establish a one-to-one correspondence between polygonal triangulations and planted trivalent binary trees. This relationship can be demonstrated using the hexagonal triangulation in Figure 2. 16, by placing a dot next to each label, below the base, and inside each triangle.

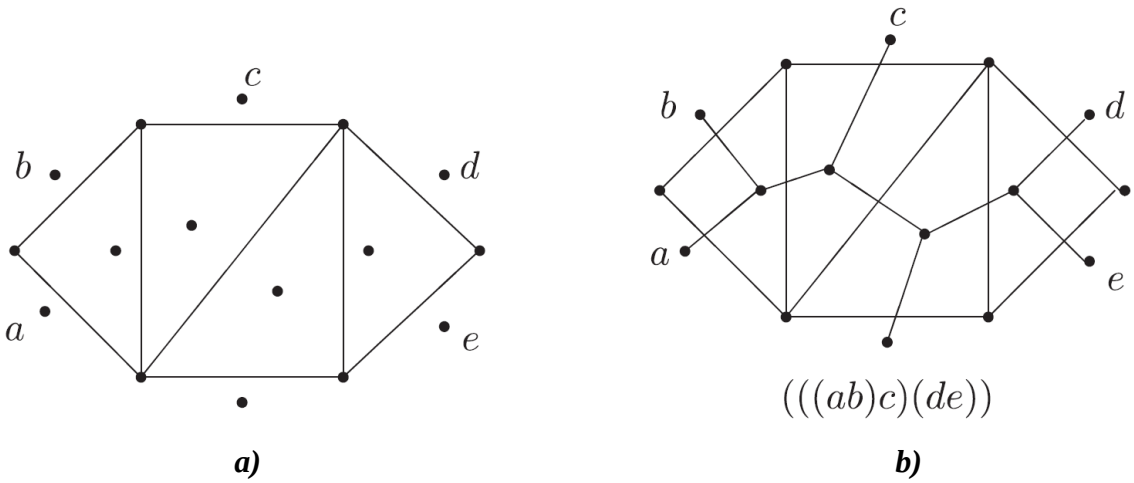


Figure 2.17 Triangulation of polygon and Planted Trivalent Binary Tree [40, pp. 237]

By connection of every two separated dots from exactly one side of a triangle; is obtained the graph of the planted trivalent binary tree with five leaves see Figure 2.16 - b) and Figure 2.17 - a). The resulting graph of the planted trivalent binary tree correspondent to the building up of the multiplication order $(((ab)c)(de))$. The planted trivalent binary tree for a better overview can be redrawn to a normal-looking as in Figure 2.17 - b).

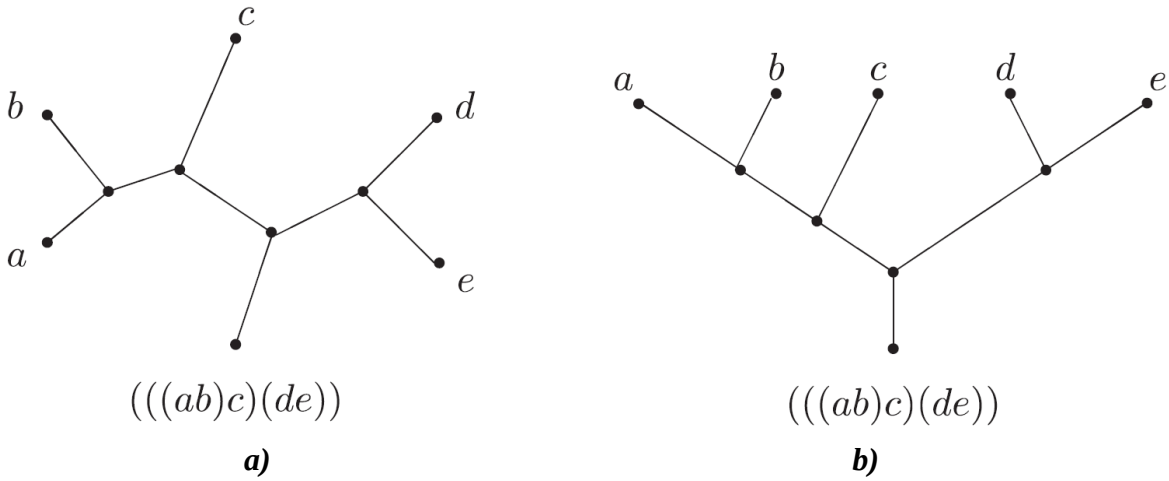
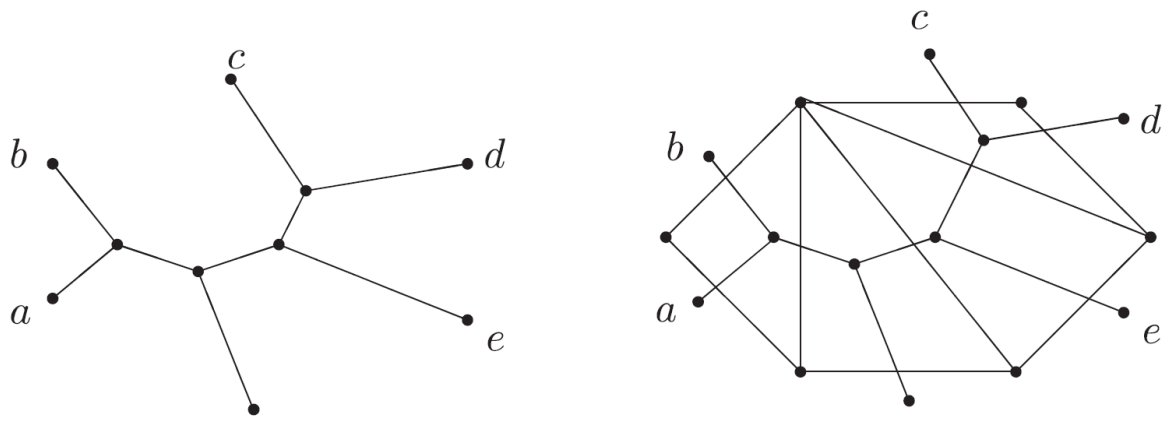


Figure 2.18 Planted Trivalent Binary Tree of hexagon [40, pp. 237]

This algorithm constructs a planted trivalent binary tree with $n-1$ leaves corresponding to every triangulation of a convex n −gon, where $n \geq 3$. Can this process be reversed? In other words, does there exist a polygonal triangulation corresponding to each planted trivalent binary tree? The answer is YES, and the reverse algorithm can be illustrated as follow.

Let's consider the planted trivalent binary tree in Figure 2.18 – *a*); it has five leaves, labeled a through e. Place a dot between every two leaves. Join the dots by line segments in such a way that each line segment crosses an edge of the planted trivalent binary tree exactly once. This results in Figure 2.18 – *b*). Deleting the original planted trivalent binary tree yields a unique triangulation of a hexagon.

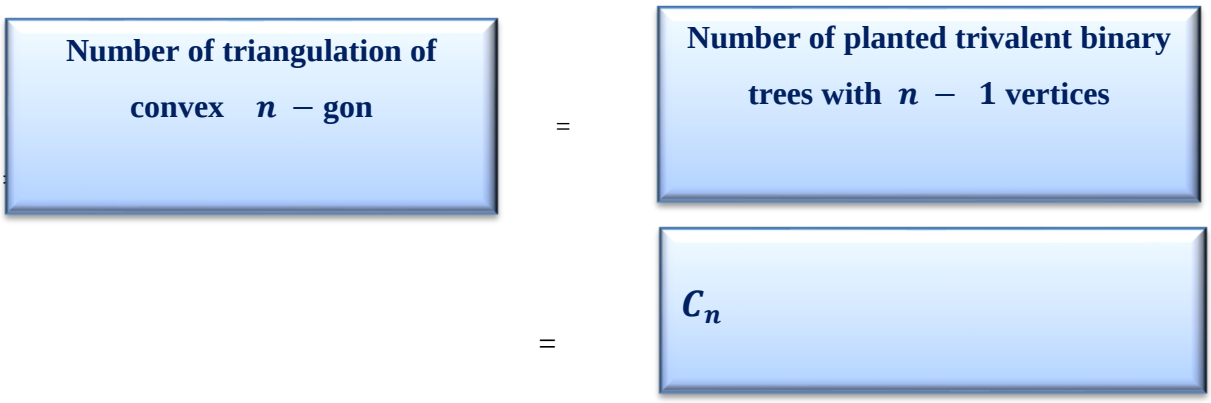


a) ***b)***

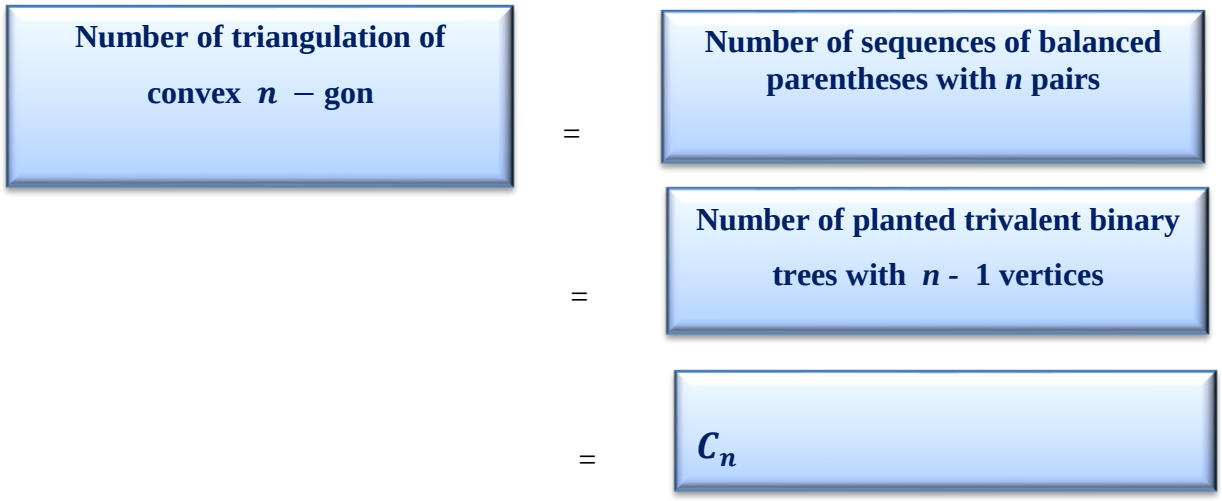
Figure 2.19 A Hexagonal Triangulation [40, pp. 238]

This technique can be applied to any planted trivalent binary tree with $n-1$ leaves yields a unique triangulation of a convex n −gon, where $n \geq 3$ [40, pp 238].

From both algorithms, we see that there is a bijection between the set of triangulations of a convex n −gon and the set of planted trivalent binary trees with $n-1$ leaves. Consequently, for $n \geq 3$ we have:



Thus, from the bijection given above it follows that:



2.7 Catalan Numbers and Lattice Paths

The integer lines parallel to coordinate axes in the Cartesian coordinate system form the lattice of integer lines or discrete lattice.

The set $L = \{(m,n) : m,n = 0,1,2,\dots\}$ are the points of the lattice and the lines joining these points are called the edges of the lattice. Let's fix two points (m_1,n_1) and (m_2,n_2) in this lattice, such that $m_2 \geq m_1$ and $n_2 \geq n_1$. A **increasing/lattice** path from (m_1,n_1) to (m_2,n_2) is a subset $\{e_1,e_2,\dots,e_k\}$ of L such that:

- either $e_1 = (m_1, n_1 + 1)$ or $e_1 = (m_1 + 1, n_1)$;
- either $e_k = (m_2, n_2 - 1)$ or $e_k = (m_2 - 1, n_2)$; and
- if we represent the tuple $e_i = (a_i, b_i)$, for $1 \leq i \leq k$, then for $2 \leq j \leq k$,
  - either $a_j = a_j - 1$ and $b_j = b_j - 1 + 1$
  - or $b_j = b_j - 1$ and $a_j = a_j - 1 + 1$.

The movement on the lattice is either to the **right** or **up**, see Figure 2.19.

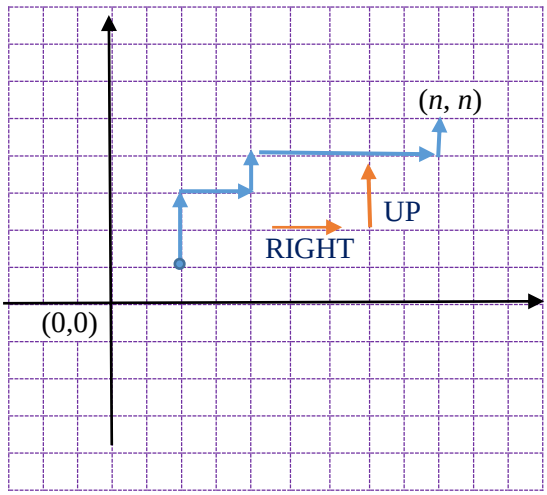


Figure 2.20 A lattice with a path

Let's look the $n \times n$ size discrete lattice. How many different paths can be drawn in this discrete lattice? That is, the number of possible lattice paths from $(0,0)$ to the lattice point (n, n) on the discrete lattice such that from any lattice point (x, y), we can walk one block right $(\boldsymbol{R})$ or one block up $(\boldsymbol{U})$, where $x, y \geq 0$. Every path can be represented by a word made up of exactly n $\boldsymbol{R}$'s and n $\boldsymbol{U}$'s. So, the total number of paths through the discrete lattice to the point (n, n) is $\binom{2n}{n}$.

Notice that the shortest paths in the lattice are the paths that do not go above the diagonal $(y = x)$.

The number of shortest lattice paths or valid paths from $(0, 0)$ to (n, n) we will denote with P_n. In Figure 2.19 are given the various possible valid lattice paths on an $n \times n$ discrete lattice, where $0 \leq n \leq 3$.

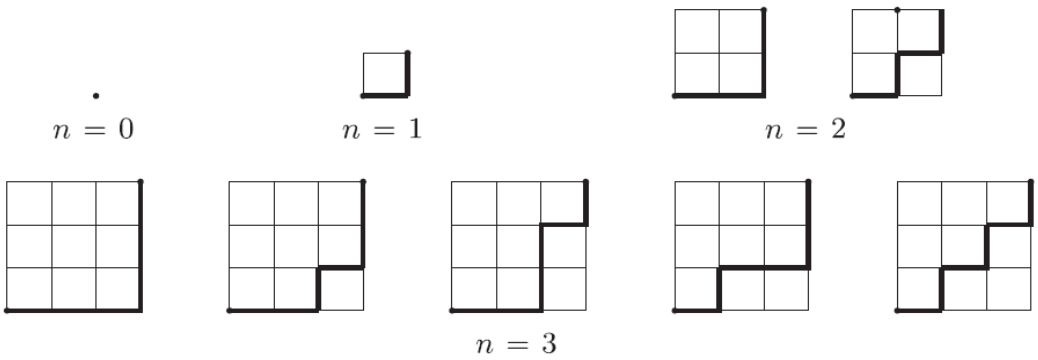


Figure 2.21 Valid Paths [40, pp. 260]

We make three important observations:

- Every valid path must begin with a R and end in an U.
- Every valid string (that is, path) has the property that the number of R's is greater than or equal to the number of U's in each substring.
- Replacing a R with a 1 and an U with a 0 results in a ten-bit word such that the number of $1s$ is greater than or equal to the number of $0s$ in each substring, when read from left to right [40, pp. 261].

Now we count the paths that go above the diagonal. We look at the first point of the invalid path over the diagonal. After that point, we are moving by replacing each right move with move upward and vice versa. Since we have come to a single field above the diagonal, so far we have k movement to the right and $k + 1$ upwards. To arrive in (n, n) we must take $n - k$ movement on right and $n - k - 1$ upwards. In the remaining paths the numbers of the other movements are replaced as above and in modified path will have $k + (n - k - 1) = n - 1$ movements to the right and $(k + 1) + (n - k) = n + 1$ upward, and in this way we arrive at the point $(n - 1, n + 1)$. Notice that any invalid path can be modified uniquely. Let's also observe that any valid path in the discrete lattice from $(0, 0)$ to $(n - 1, n + 1)$ can be passed to an invalid path from $(0,0)$ to (n, n), by moving above the diagonal. In this way, we have established a bijection between the set of all invalid path and all valid paths in the discrete network to the point $(n - 1, n + 1)$, which they are $\binom{2n}{n-1}$. By subtracting the number of invalid paths from the number of all possible paths, we obtain the number of valid paths. Thus, the number of legal paths on an $n \times n$ grid is given by

$$\binom{2n}{n} - \binom{2n}{n-1} = \frac{(2n)!}{n!\,n!} - \frac{(2n)!}{(n+1)!\,(n-1)!} = \frac{(2n)!}{(n+1)n!} = C_n$$

3 Dynamic Programming

Dynamic programming as a technique is used in the operational research in the mathematical optimization and as in the algorithms in computer programming. In this research is presented a practical view of the dynamic programming, specifically in the context of application to finding optimal solutions for polygonal triangulation problems.

In the application of dynamic programming, similar to the divide and conquer technique, the resulting instances of the problem are systematically divided into simpler sub-problems of the same problem. Solving the simpler sub-problems is then used to solve the initial problem. However, while aiming to divide and conquer an initial problem into independent sub-problems, dynamic programming is applicable when simple sub-problems overlap.

By dynamic programming, the naive algorithms for many combinatorial problems can be significantly accelerated with increasing of storage space. Although formally incorrect, intuitively can be said that the algorithms of dynamic programming with exponential time complexity can be decreased in polynomial time complexity with increasing spatial complexity for the polynomial factor. Even when high spatial complexity is acceptable, dynamic programming is often not appropriate for solving a problem, ie it does not bring the acceleration in the output.

In the broadest sense, layout problems require optimum problems organization on some set, with different conditions and different objective functions. Different versions of scheduling problems naturally occur in the organization of production, but also in computer systems, for example in the schedule of problems in the operating system.

A significant part of practically interesting variants of the scheduling problem is *NP-complete*[1] or *NP-hard*[2], so it is typically solved by heuristic algorithms that find "good enough" problem-solving. However, it is possible to design a dynamic programming algorithm that finds an optimum order in pseudo-polynomial time and may be applicable to smaller instances of problems whose optimum is of particular importance.

[1] A problem is *NP-complete* if answers can be verified quickly and a quick algorithm to solve this problem can be used to solve all other NP problems quickly.
[2] A combinatorial optimization problem which is proved that belong to the class of *NP-complete* problems is the *NP-hard* problem.

Dynamic programming is closely related to recursion and mathematical induction. Any problem solved by dynamic programming can be reduced to the calculation of a particular member of a sequence of numbers, the sequence definition being dependent on the problem being solved. In a truncated case, the sequence will be multi-parameter, where each parameter corresponds to a problem dimension, but the multi-parameter sequence can also be converted to a single parameter the sequence of somewhat more complex definitions. The definition of the sequence describes the sub-problems that are used in solving the problem, and how the solutions of these smaller sub-problems are linked to the solution of the problem. Consequently, as a useful introduction to dynamic programming can be a simple problem of computing an n-member of a single-parameter sequence of numbers.

3.1 Memoization

The simplest way to apply dynamic programming to an existing recursive solution is the memoization. Memoization implies memorizing the calculated result for certain function arguments in order to avoid re-calculation when the function invokes with the same arguments.

Many sequences of numbers can be described recursively, through the finite number of initial members and the definition of a *n*-member as the functions of the previous members. For example, the Fibonacci sequence is defined by the first two initial values zero and one, and *n*-th members equal to the sum of the previous two. Mathematically, the definition of the of the Fibonacci sequence can be written as follows:

$$F_0 = 0, \qquad F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}, \quad \forall n > 1$$

ALGORITHM 3.1 Calculation of Fibonacci Numbers with Memoization (naive algorithm).

memo = { }
$fib(n)$:
if n in memo: return memo$[n]$
else if $n = 0$: return 0
else if $n = 1$: return 1
else: $f = fib(n - 1) + fib(n - 2)$
memo$[n] = f$
return f

Figure 3.2 shows the tree of function f for $n = 5$. It is obvious from the picture that the invocation of the fib $(n - 2)$ is executed in a constant time, because during the execution of the call the fib $(n - 2)$.

The time complexity of the memoization algorithm is $\Theta(n)$. In this case, by using dynamic programming, there was no asymptotic increase in time complexity, since the naive algorithm has the same spatial complexity due to the software used for recursive calls.

Memoization is sometimes referred to as top-down dynamic programming because a smaller instance of the problem is solved only when its solution is necessary to solve the larger instance. This property represents a fundamental difference in relation to the "right" dynamic programming, where systematically solving instances from smaller to larger, ie from bottom-up.

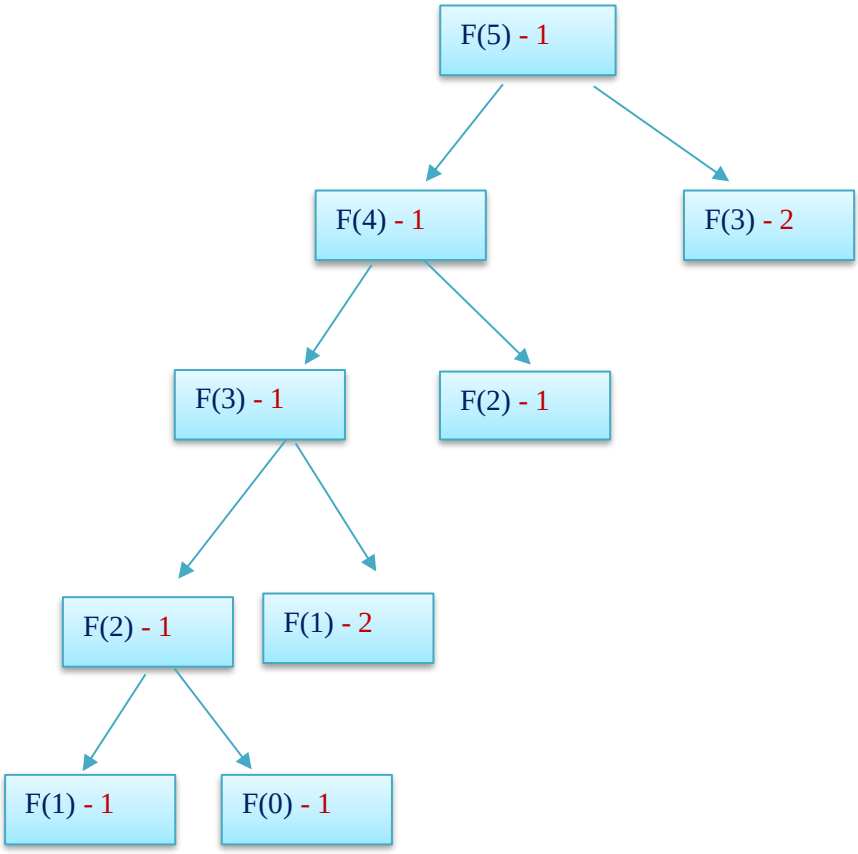


Figure 3.2 Application of Algorithm 3.1, $n \leq 5$.

3.2 Matrix-chain multiplication

In this section, is given an algorithm of dynamic programming that solves the problem of matrix-chain multiplication. Let $(A_1, A_2, \dots, A_n)$ is a sequence (chain) of n matrices which are multiplied, and for which is required to compute the product

$$A_1 \cdot A_2 \cdots A_n \tag{3.1}$$

The product (3.1) can be calculated using the standard algorithm for multiplying matrix pairs and parenthesization. Because the matrix multiplication is associative, all parenthesizations give the same product. A product of matrices is ***fully parenthesized*** if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses [12]. For matrix chain (A_1, A_2, A_3, A_4) , the product $A_1 \cdot A_2 \cdot A_3 \cdot A_4$ can be fully parenthesized in five different ways:

$$(A_1(A_2(A_3A_4))),$$
$$(A_1((A_2A_3)A_4)),$$
$$((A_1A_2)(A_3A_4)),$$
$$((A_1(A_2A_3))A_4),$$
$$(((A_1A_2)A_3)A_4).$$

How we parenthesize a chain of matrices can have a big number of operations in the evaluation of the product. The standard procedure of *square-matrix-multiplication* algorithm is given with the following algorithm, which generalizes the matrix chain multiplication [12]. The attributes columns and rows in the algorithm are the numbers of columns and rows in a matrix.

ALGORITHM 3.4 Matrix-Multiply (A, B)

```
if   A:columns ≠ B:rows
     error "incompatible dimensions"
else let C be a new A:rows × B:columns matrix
      for i = 1 to A:rows
          for j = 1 to B:columns
              c_ij = 0
              for k = 1 to A:columns
                  c_ij = c_ij + a_ik · b_kj
 return C
```

Two matrices X and Y can be multiplied only if the number of columns of X is equal the number of rows of Y. If X is a $m \times n$ matrix and Y is a $n \times k$ matrix, the resulting matrix Z is a $m \times k$ matrix. The time to compute Z is depend on the number of scalar multiplications in step 8 of algorithm, which is mnk. In what follows, are given the costs of evaluation of matrix chain in terms of the number of scalar multiplications.

For the illustration of the different products that are obtained by different parenthesizations, see the problem of a chain (A_1, A_2, A_3) of three matrices [12].

The ***matrix-chain multiplication problem*** is the problem that fully parenthesizes the product $A_1 \cdot A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications in the matrices chain $(A_1, A_2, \ldots, A_n)$, where matrix A_i has dimension $p_{i-1} \times p_i$, $i = 1, 2, \ldots, n$. The main goal in this procedure is a determination of the order in the multiplications of matrices that have the lowest cost in the matrix-chain, not the multiplication of matrices.

The Number of Different Parenthesizations

The number of different parenthesizations of a sequence of n matrices is denoted with P_n. One matrix has only one-way of the parenthesization in the matrix product. When $n \geq 2$, a full parenthesization of the matrix product is the product of two parenthesized of the matrix subproducts, and the split between the two subproducts may occur between the k –th and $k + 1$ – st matrices for any $k = 1, 2, \ldots, n - 1$ [12]. Thus, is obtained the recurrence

$$P(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2. \end{cases} \tag{3.2}$$

and it is equal to the Catalan number C_{n-1} .

Detail for construction of the structure of an optimal parenthesization and recursive solution of the matrix chain problem is given on [12].

The minimum number of scalar multiplications $m[i, j]$ needed to compute the matrix chain is equal with the computing of the subproducts $A_{i\ldots k}$ and $A_{k+1\ldots j}$ plus the cost of multiplying these two matrices together. If $i = j$, the problem is trivial $m[i, i] = 0$, $i = 1, 2, \ldots, n$; if $i < j$ the matrix product $A_{i\ldots k}\ A_{k+1\ldots j}$ takes $p_{i-1}\, p_k\, p_j$ scalar and is obtained that the recursive relation

$$m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1}\, p_k\, p_j \tag{3.3}$$

In the recursive equation (3.3) it is assumed that we know the value of k, in fact, we do not know. For this reason, the optimal parenthesization must use one of the values for $k = i, i + 1, \ldots, j - 1$ and check them all until it is found the best solution. Therefore, the recursive definition of the minimum cost of the product. Thus, the recursive definition of the minimum cost of parenthesizing the product $A_i\, A_{i+1} \ldots A_j$ becomes

$$m[i,j] = \begin{cases} 0 & if\ i = j \\ \min\limits_{i \le k < j} m[i,k] + m[k+1,j] + p_{i-1}\, p_k\, p_j & if\ \ i < j \end{cases} \qquad (3.4)$$

We notice here that the values $m[i,j]$ give only the costs of optimal solutions to subproblems, but they do not provide all the information we need to construct an optimal solution. For this reason is defined $s[i, j]$ the value of k at which the product $A_i\, A_{i+1} \ldots A_j$ in is split in an optimal parenthesization. Furthermore, $s[i,j] = k$ such that $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}\, p_k\, p_j$ .Computing the optimal cost for matrix chain multiplication $A_1 \cdot A_2 \cdots A_n$ is done with the following recursive algorithm with a bottom-up approach.

ALGORITHM 3.5 Matrix-Chain-Order (p)

```
n = p.length - 1
let m[1 ... n, 1 ... n] and s[1 ... n, 1 ... n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                    // l is the chain length
    for l = 1 to n - l + 1
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1
            q = m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

4 Optimization and Triangulations

Triangulations of the polyhedron and the point configuration are two concepts that appear in problems in mathematical programming (also called optimization). An optimization is a mathematical approach to solving a particular problem for select the best of the offered / possible alternatives. These alternatives belong to a particular set called feasible set. If a particular problem that is subject to analysis is presented as a mathematical model, that is, a certain production real function, solving this problem is limited to determining the optimal (maximum or minimum) value of this function.

Linear programming – LP (also called linear optimization) as a special part of optimization is an effective method that finds an application in optimization problems solution in fields of industry, theoretical computer science, and combinatorics. The application of LP involves prior modeling of the problem that is subject of analysis. These optimization problems are part of the inputs/outputs system and are represented by variables with its own characteristics, with certain specific linear constraints. The objective function which is actually the subject of optimization has a linear form and is represented with the constraints which mathematically are given in the form of linear equations. The importance in the direction of correct application of LP as a mathematical model for optimizing of the objective function is in the generation of a clear and precise model that will reflect the reality. In the linear optimization problem the objective function and the constraints of the system are given with linear relations, ie, the problem has a mathematical representation with a finite number of linear equalities and inequalities. In computer science, the application of linear programming we see in inefficient algorithms for maximum flows on networks, matchings of graphs, etc.

Parametric linear optimization problems are an important class of linear optimization problems. In these problems, the inputs instead of fixed values are in the form of parameters. Is important to note that in the parametric linear programs data can have uncertainty or can analyze the effects of deviations from the initial values. In this section is explained the relationship between triangulations and parametric linear optimization problems given below.

$$\begin{aligned} & min\ c \cdot x \\ \text{Subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{4.1}$$

In the section, the linear optimization is reviewed in terms of point configurations. Let linear program is given in the matrix form presented in (4.1), where A is $d \times n$ matrix of coefficients and b is $d-$ column vector. With $cone(A)$ is denoted the *cone* generated by the nonnegative linear combinations of the columns of the matrix *A*. For each subset $B \subset [n]$ of columns with A_B is denoted the column sub-matrix. A *cone subdivision* of $cone(A)$ is a finite collection *S* of subcones $cone(A_B)$, such that the intersection of any pair of subcones in *S* is a face of both and the union of all the subcones is $cone(A)$ [16]. A finite collection *S* of subcones $cone(A_B)$, such that the intersection of any pair of subcones in *S* is a face of both and the union of all the subcones is $cone(A)$ is subdivision of $cone(A)$. The cone subdivision where all parts of division are simplicial cones (all submatrices are square matrices) is cone triangulation. We consider the linear optimization problem

$$LP_{\boldsymbol{A,c}}(b) := min\{c \cdot x : Ax = b, x \geq 0\} \tag{4.2}$$

where A is $d \times n$ matrix, $b \in cone(A) \subset \mathbb{R}^d$ right-handside vector and $c \in \mathbb{R}^n$ is cost vector. The constraints $x \geq 0$ means that all the entries of x are non-negative.

By definition of a $cone(A)$, linear optimization problem (4.2) is has a solution x (or in LP terminology is feasible) if and only if the right-hand side b lies in $cone(A)$. If $cone(A_B)$ corresponds to a full-rank matrix[1] we say A_B is a *basis*. We must notice that for a basis A_B the $cone(A_B)$ is a *simplicial cone*. From a basis with indices $B = \{j_1, j_2, \ldots, j_d\}$ and $j_1 < j_2 < \ldots < j_d$ can be immediately constructed a tentative feasible solution of the linear optimization problem. We set to zero any variable x_l where $l \notin B$ and for x_{jk} with $jk \in B$ set the variable x_{jk} to the value of the $k-$th component of $(A_B)^{-1}b$. This is just a tentative solution because some of the x_{jk} 's may be negative, but if all of of them are non-negative then the vector *x* is a *basic feasible solution* of the linear optimization problem. There are only finitely basic feasible solutions (is at most $\binom{n}{d}$), each of them are determined by *d*-subset of the columns of matrix *A*.

[1] The matrix with *d* linearly independent columns

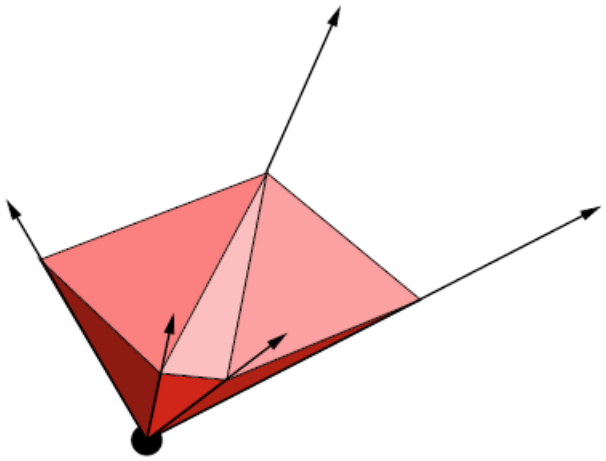


Figure 4.1 A triangulation of a pointed cone, cut off by an affine hyperplane; this section of the cone looks like a triangulation of a point configuration [16, pp.14].

It is well-known that if there are at least one feasible solution and the objective function $c \cdot x$ is bounded from above on the set of all feasible solutions, then there exists an optimal solution. If an optimal solution exists, then there is a basic feasible solution that is optimal [16]. Thus, an optimal solution of the linear optimization problem (4.2) of is achieved in the vector x with least $n - d$ zero coordinates. If coefficient matrix b and $c \in \mathbb{R}^n$ are sufficiently generic, then *exactly* $n-d$ coordinates are zero. We can then consider b as selecting the maximal rank square $d \times d$ submatrix of A, the *basis*, within the set of columns corresponding to non-zero entries [16] . It is important to note that in selection of the basic feasible solution finite linear optimization problem for the right-handside vector b we also select a simplex of the triangulation.

The simplex method is one of the linear optimization procedures that come to an optimal solution. Basically, in the simplex method, the constraints imposed by the inequality are replaced with the corresponding equation, into which is introduced the additional negative variable. The algorithm of the simplex method starts from an arbitrary basic feasible solution and ideally finds a sequence of cheaper and cheaper alternatives. The other method for solving linear optimization problems today is the ellipsoid method and the interior-point algorithms.

Following lemma and theorem gives a relationship between linear optimization problem and triangulations.

Lemma 4.1 (Complementary slackness)**.** *Let A be a matrix, b and c the right-hand-side and cost vectors of* $LP_{A,c}(b) := min\{c\,x : Ax = b, x \geq 0\}$. *There is an associated dual problem, and the following duality equation holds:*

$$max\{yb : yA \leq c = min\{c \cdot x : Ax = b, x \geq 0\} \quad (4.3)$$

If both optima are finite and x^* and y^* are feasible solutions, then the following conditions are equivalent:

(i) x^* and y^* are optimum solutions of their problems.

(ii) If a component of x^* is positive, the corresponding inequality in $yA \leq c$ is satisfied by y^* with equality, i.e., $x^*(c - y^*A) = 0$.

In other words, the minimum value of $LP_{A,c}(b))$ is attained at a vector x^* if and only if there exists a y such that $yA_j \leq c_j$, $\forall\ j = 1, \ldots, n$ and $\forall$ indices either $x_j^* = 0$ or $yA_j = c_j$.

Using the principle of complementary slackness, one can find the primal-dual optimal solution of the linear optimization problem. From the idea that $cone(A)$ will be triangulated by each choice of cost fixed vector c and varying b is needed to study the parametric family of linear programs which are given as below

$$LP_{A,c} = \{LP_{A,c}(b) : b \in cone(A)\}.$$

For simplicity, usually can be assumed that $ker(A) \cap \mathbb{R}_+^n = \{0\}$, where $ker(A) = \{x \in \mathbb{R}^n : Ax = 0\}$ and $\mathbb{R}_+^n = \{x \in \mathbb{R}^n : x \geq 0\}$. This assumption on A builds $LP_{A,c}$ a family of bounded linear programs (so the minimum exists in all cases, no matter what c and b). The division of $cone(A)$ into regions consisting of "equivalent" linear optimization problems first time was observed by Walkup and Wets. If in division c is generic they are turn out that the subdivision of cone is a triangulation.

Theorem 4.2 (Walkup-Wets)**.** *Let $LP_{A,c}(b)$ denote the linear optimization problem*

$$min\{c \cdot x : Ax = b, x \geq 0\}$$

for each c and A. $LP_{A,c}(b)$ is bounded, then for each generic cost vector c there exists a triangulation $T(c)$ of $cone(A)$ such that, for each $b \in cone(A)$, the extreme rays of any d-dimensional cone of $T(c)$ containing b are an optimal basis for $LP_{A,c}(b)$.

The proof is given in [16, pp. 16].

5 Applications

In this section we describe the application of the computational geometry algorithms in cryptography, linear optimization, triangulation of polygon based to planted trivalent binary trees and square matrix method for finding and storing of optimal triangulations.

5.1 Generation Of Cryptographic Keys With Algorithm Of Simple Polygon Triangulation And Catalan Numbers

This section presents a procedure for the application of a computational geometry algorithm in the process of generating cryptological keys from one segment of the *3D* image. The presented procedure consists of three phases. In the first phase, is done the separation of one segment from the *3D* image and determination of triangulation of the separated polygon. In the second phase, is done a conversion from the obtained triangulation of the polygon in the record which represent the Catalan key. In the third phase, the Catalan-key is applied in encryption based on the Balanced parentheses combinatorial problem.

Visual cryptography is a special encryption technique that allows to hide the information (secret messages) in a image in such a way that a person can decode only if he owns and uses the correct key. Hiding information is a great area and a modern way of communication for successfully avoiding attacks and decipher confidential information. In this paper is represented a way of application of simple polygon triangulation algorithm in the process of generating cryptographic keys from one segment of the 3D image.

Computational geometry is an integral part of mathematics that deals with the algorithmic solving of the geometric problems. This discipline also is considered a branch of computer science, which was created as a result of an attempt to solve geometric problems with a computer. From the very beginning, computational geometry is connect different areas of science and technology such as theory of algorithms, combinatorial and Euclidean geometry, but also is include the data structures, optimization, etc. Today, computational geometry has a great application in computer graphics, visualization, GIS, CAD programs, etc. In the background of a view (images, animations, etc.), complex geometric calculations and theories have been hiding, which are long time ago confirmed and developed, but which from the aspect of application in modern information technologies are still at the beginning.

Triangulation of the simple polygon is one of the more important problems applied in computational geometry. This problem is applied in the process of obtaining three-dimensional representations of objects from a set of points.

This section determines the importance of polygon triangulation and Catalan numbers whic has in the cryptology, primarily in the development of algorithms for generating pseudorandom numbers that are necessary for generating keys. In the papers [76, 87], can be seen the concrete applications for solving of some combinatorial problems in computer geometry. The research [87] presents the methods and techniques for solving of some problems in the field of computer geometry, based on Catalan numbers and combinatorial problems, which in second paper of authors [78] are applied in the field of cryptography. So, in this paper is made a combination of computer geometry, cryptography and combinatorics. Generally, number theory in asymmetric systems has an important place not only in generating keys, but also in the design of the cryptologic algorithm, but also in cryptanalysis [5, 11, 31].

5.1.1 Catalan numbers and polygon triangulation algorithm

Catalan numbers are widely used in solving many combinatorial problems. In the monograph [5] are listed the concrete applications of these numbers with possible solutions when it comes to the representations of the Catalan numbers. The author in his practicum [86] lists a set of problems that describe over 60 different interpretations of Catalan numbers. We can enumerate some of interpretations: *binary trees, polygon triangulations, stack permutations, problem of paired parenthesis, Ballot problem, lattice path problem*, etc. It is generally known that all of these combinatorial problems can be solved on the basis of values that possess the properties of Catalan numbers, and more precisely the solution of these combinatorial problems are covered with the application of these numbers.

The procedure presented in the paper is focused to the proposal of generation of the Catalan keys based on the isolated triangulation from one part of the 3D image. The resulting Catalan number, will represent the key for encryption and decryption of confidential information (hereinafter referred to as Catalan-key). Encryption can be implemented in combination with the aforementioned combinatorial problems that are based and whose solution are given with these numbers. For a concrete example of encryption in paper, is taken an example with Balanced parentheses.

5.1.2 Connectedness of convex polygon triangulation method and Catalan numbers

Triangulation of the polygon is a historically very old problem which led to the discovery of Catalan numbers. The triangulation of the *n*-angled polygon requires the division of triangles with *(n-3)* -internal non-intersect diagonals. For convex polygons, all diagonals are internal diagonals. In this case, the number of triangulations of a convex *n*-angle polygon is independent of the form and can be uniquely characterized by the number of vertices *n*. Algorithm 5.1 is a procedure for finding the number of possible polygon decomposition into triangles using its non-intersecting diagonals.

ALGORITHM 5. 1 Triangulation of simple polygon [5]

INPUT: *n*, number of vertices of polygon.

1: Set the counter to $i = 1$

2: *i*-th vertice joins with $(i + 2)$ −th vertice

3: Has the obtained diagonal internal?

Yes: It is added to the list and the $(i + 1)$ vertice of the polygon is ejected.

No: $i = i + 1$

4: Return to Step 2

OUTPUT: $n - 3$ diagonals

In the research [76, 87], some ways of solving this type of problem have been presented. Now we will associate the concept of polygon triangulation and Catalan numbers. If with T_n we denote the number of triangles of the n-angle, then the following relation is hold:

$$T_n = C_{n-2}, n \geq 3 \tag{5.1}$$

where *n* is the number of vertices of the polygon. On basis to (3) the T_n is represent in the form:

$$T_n = \frac{1}{n-1}\binom{2n-4}{n-2} = \frac{(2n-4)!}{(n-1)!(n-2)!} \tag{5.2}$$

Triangulation allows the display of three-dimensional objects from a set of points and provides a mechanism for the so-called. *ironing of three-dimensional figures* (Figure 1). Triangulation of convex polygons is an actual problem that arises in two-dimensional

computational geometry. The triangulation technique in computational geometry is the most common paneling method. The easiest way of segmentation and ironing with double-curved surfaces is via a triangle network. The advantage of the triangle as a geometric figure is that, the area between three points is always straight.

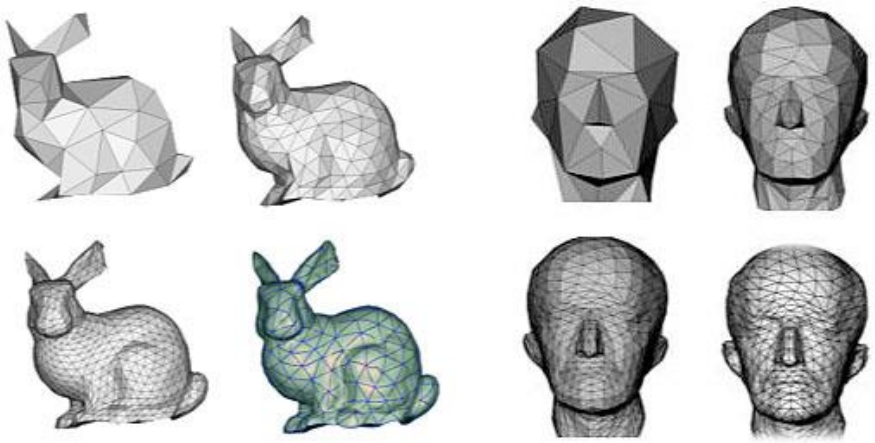


Figure 5.1. The method of ironing three-dimensional figures

In the procedure developed on this research, we will apply the polygon triangulation method that has wide application in modeling of the 3D objects (Figure 2). The advantages of this method are: small deviations from the original shape, good structural properties and the possibility of cladding of the complex free forms. Based on this method, in the 3D image we will extract a segment, which will serve as a material for generating a *Catalan-key*.

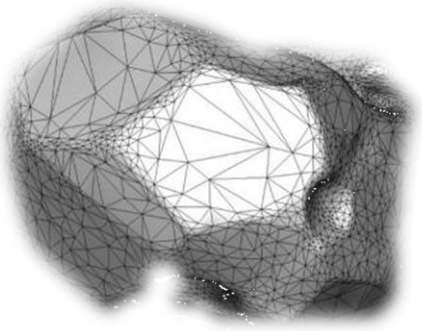


Figure 5.2 Modeling of the 3D objects, respectively separating one segment from a 3D image

5.1.3 Extraction of the keys from one segment of 3D image

This section explains the work methods of generating cryptographic keys based on triangulation. This process consists of three phases:

1. Separation of one segment from a 3D image and definition the triangulation in the separated polygon
2. Converting triangulation of a polygon into a binary or some other record that corresponds to the property of Catalan number. From this phase we get the record of the Catalan-key.
3. Application of the Catalan-key in the encryption on the basis of some combinatorian problem, which is based on Catalan numbers.

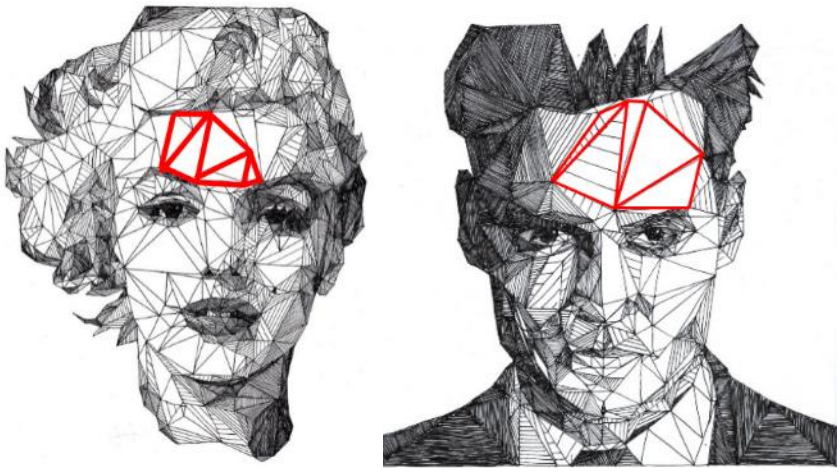


Figure 5.3 First phase: Separation of one segment from 3D image and determination of the triangulation

In the second phase, it is necessary to connect the obtained triangulation with binary trees. The binary tree is a well-known concept in the field of computer science and represents a structure for data storage. In order to obtain an appropriate binary record for each generated triangulation of a convex polygon, we apply the Lukasiewicz's algorithm [40]. The procedure for visiting binary trees according to this algorithm is realized, by labelling leaves with 0 and each internal vertex with 1. In this way, is obtained a corresponding record (notation), which is unique and corresponds to exactly one triangulation for which this binary tree is valid [87].

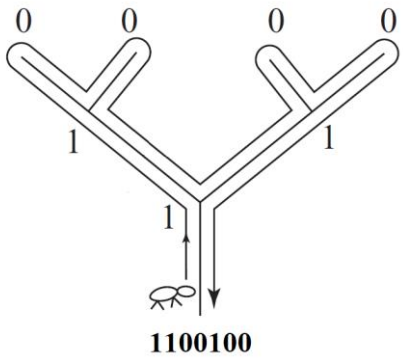


Figure 5.4 Recording of binary trees using Lukasiewicz's algorithm

What is important in this case is that binary trees represent the Catalan numbers. Each binary tree corresponds to one binary record. Figure 5 shows the process for generating binary records for Catalan number C_3, which is a total of 5 combinations of binary trees and 5 binary records of Catalan-keys.

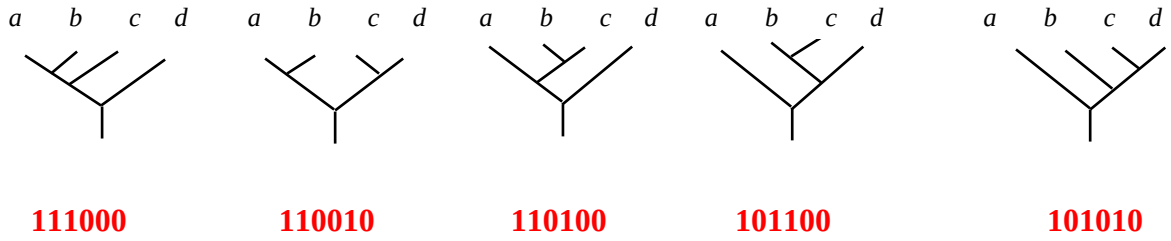
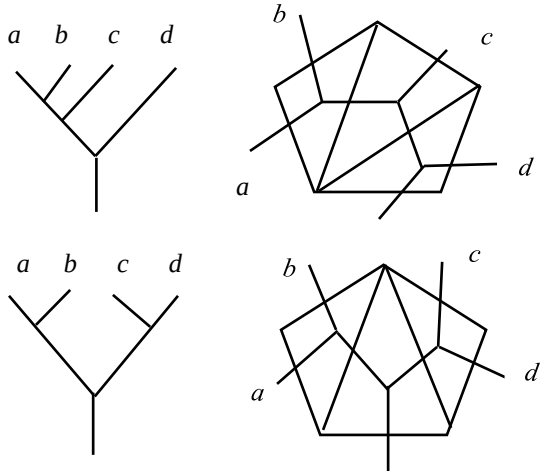


Figure 5. 5 Binary record of trees

Hence, it is now very easy to connect the binary tree and triangulation of the polygon, and thus, it is easy to represent each triangulation as a binary record or in the form of the paired parenthesis (Figure 6).



Binary notation of Triangulation: **111000**
Balanced Parentheses notation: **((()))**

Binary notation of Triangulation: **110010**
Balanced Parentheses notation: **(())()**

Figure 5. 6 Second Phase: Triangulation and corresponding binary tree for the first two cases from the previous image

5.1.4 Method for Alpha-Numeric notation of keys

The method for generating Alpha-numeric records (hereinafter AN record or AN notation) was created with the aim to save the memory space in the process of generating, distributing and storage of large key records. More precisely, the AN notation represents a shortened record for the Catalan-key originally presented as Balanced Parentheses.

The method is based on the problem of balanced parentheses, but also can be applied to other records (eg binary records of trees). It consists of 4 phases:

1. **Elimination** - the first phase where the first open and last closed parenthese is removed. In this way we didn't affect to the uniqueness of each record, because each record have same begin and end.
2. **Replacement** - the second phase in which we convert characters to binary record, so the opened parentheses after this phase were replaced with 1 and closed with 0 (bit-string rule).
3. **Selection** - the third phase in which the first and last group with two or three binary numbers is taken. The selected groups of binary numbers are replaced with the Alpha character (described in detail in our paper [79]), and this part makes Alpha a part of the record.
4. **Conversion** - the fourth phase in which the remaining central part of the binary record is converted into a decimal number. The result obtained from the conversion phase is the numerical portion of the record that is added to the alpha record.

The figure shows the process of converting BP notation into an AN notation through four phases:

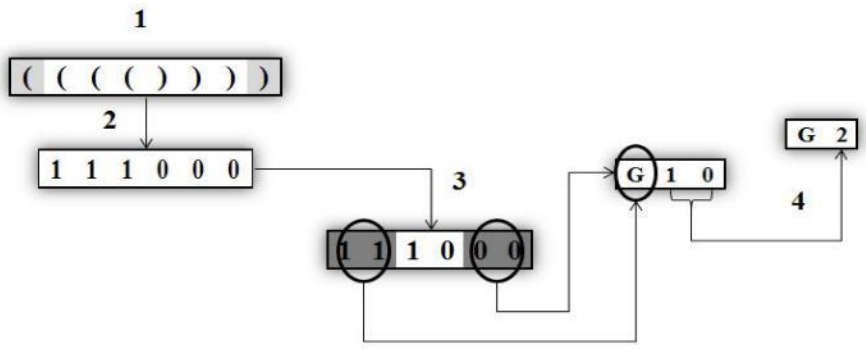


Figure 5.7 The phase of elimination, replacement, selection and conversion

5.1.5 Application of Catalan keys for encryption of the text

In the references [32, 39, 43], are liseted the concrete applications of combinatorial problems in cryptography. In this paper, we present the Catalan numbers as a keys for text encryption over a particular combinatorial problem. All of these combinatorial problems in [86], can be solved on the basis of values that possess the properties of Catalan numbers. Therefore, the number of combinations and the method for generation of Catalan numbers is a solution of the certain combinatorial problems.

Basic properties of Catalan keys and their scope (key space)

In the Table 1 are given the values of first 30 Catalan numbers. From this table, we can see that n is the basis for generation of keys, and C_n determines the number of valid keys in that basis, i.e. those values that satisfy the property of Catalan number (space of keys). For example, for the base $n = 30$ we have the space of the keys $C_{30} = 3\ 814\ 986\ 502\ 092\ 304$, that is, the values that satisfy the property of Catalan number. As the base n increases, thus the key space is drastically increased. To provide as possible as strong and more resistant encryption mechanism on cryptanalysis, the keys must be chosen mainly for values with bases greater than 30. With the application of the software solution in the Java programming language, we will show the number of values for bases n from 140 to 150.

Catalan(140)= 6563763990246161693492536077533454353889420384665868119527796560671706463922728 40
Catalan(141)= 2597771382055171036438595264488592497806939617029730903644099765561619037129981240
Catalan(142)= 1028208812757501263373597845944435911719390086180998385638154172942570891619279 2880
Catalan(143)= 4069993217165109167520491473530058817222585757799785276484360267897676445992980 5150
Catalan(144)= 1611155935622601835970180762625002593852251189639363274966912271567769848275841 94180
Catalan(145)= 6378411854725094939662770416419530816757542380901040910485447212097061454133127 68740
Catalan(146)= 2525330407789119221009341756704875466226455554887350891090156651320061065513932 186440
Catalan(147)= 9998943371381242321023474793439574481139884832189105555262377011307809353994353 116580
Catalan(148)= 3959313147057001992888490018878757680451363792611793474902551970920541958964206 9387800
Catalan(149)= 1567888006234572789183842047475988041458740061874270216061410580484534615749825 94775688
Catalan(150)= 6209251839260096211469785062189674495313420907290156219898831305495044372307257 72687824

Basic property of the Catalan-key: The number can be labelled with Catalan number if its binary form consists of an equal number of bits "1" and "0" and begins with the bit "1". If the binary record of Catalan number is associated with some other way of recording, usually with a record of balanced parenthesis, then "1" is an open parenthesis and "0" represents a closed parenthesis, so it can be said that each open parenthesis is closes, that is, each bit 1 has its pair, that is the bit 0. Also, the binary record of Catalan number can be

represented in the form of a *stack* permutations or *Ballot* records. Representation using Stack permutation bit 1 treats as a *PUSH* command and bit 0 as a *POP* command [76]. In the continuation of the paper, encryption mode based on paired parentheses will be displayed.

5.1.6 Method of paired (balanced) parenthesis in the encryption process of text

The binary notation of the Catalan number we can represent in the form of paired (balanced) parenthesis. The problem relates to calculation of the number of combinations of the possible parenthesis pairs. This number of possible valid combinations is directly determined by the formula of calculation of the set of Catalan numbers C_n. If we want to present the binary record of Catalan number in the Balanced Parentheses notation, then bit 1 we present with open parenthesis "(", and bit 0 with a closed parenthesis ")".

The encryption procedure can be applied to the text message (*String variant*), but can also be applied to the binary notation of message (*ASCII Text to Binary*). Below, we will show the text encryption process where open text values are taken as binary record (is obtained the substitution cipher). Based on the character layout "(" and ")" in the key record, the elements can have 2 states:

1. *Free element* – this is a character from a message that is not encrypted, more accurately not transfered in the cipher. A free element is conditioned by the occurrence of an open parenthesis (in a key), which is waiting for its pair, ie, a closed parenthesis.
2. *Busy element* – this is a character from a message that is encrypted and passed to the cipher. That is an element that is conditioned by the occurrence of a closed parenthesis (in a key). In this way, the element is "closed", i.e. transferred to the cipher, because is appeared the character ")", which closes the corresponding character "(".

Below is an example of encryption of one character based on a key in the form of BP record.

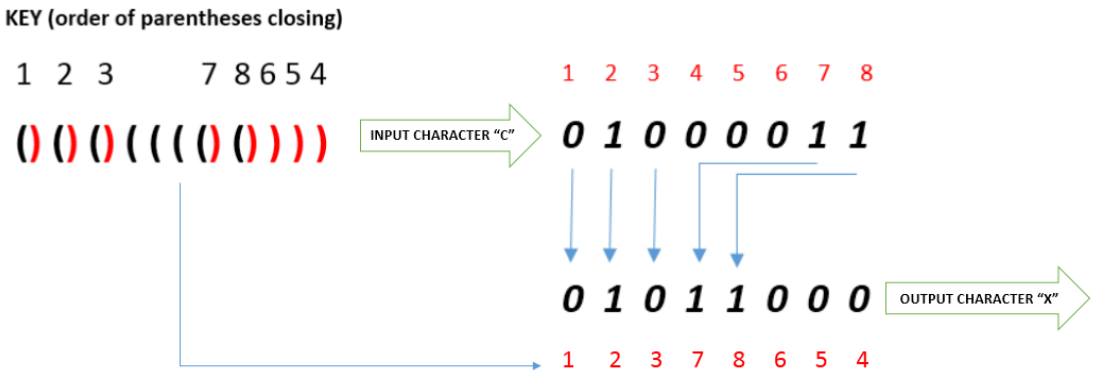


Figure 5. 8 Encryption of character C in character X, based on Catalan-key

Example: Let P_{txt}=*"CRYPTOLOGY"* is open text. If *ASCII Text to Binary* is applied to *P*, then is obtained a sequence of bits P_{bin}= *01000011 01010010 01011001 01010000 01010100 01001111 01001100 01001111 01000111 01011001*.

Using the key *K* = ((())) (()) () (() () ()) on the basis of which we will perform the permutation of the bits from the message, we obtain the following sequence of bits, ie the binary cipher: C_{bin}=*01000010 01010100 01100010 01110000 00111000 01010110 01110010 01001110 01101001 01110011.*

By applying *Binary to ASCII Text* to cipher *C*, is obtained a cipher C_{txt}***="BTbp8VrNis".***

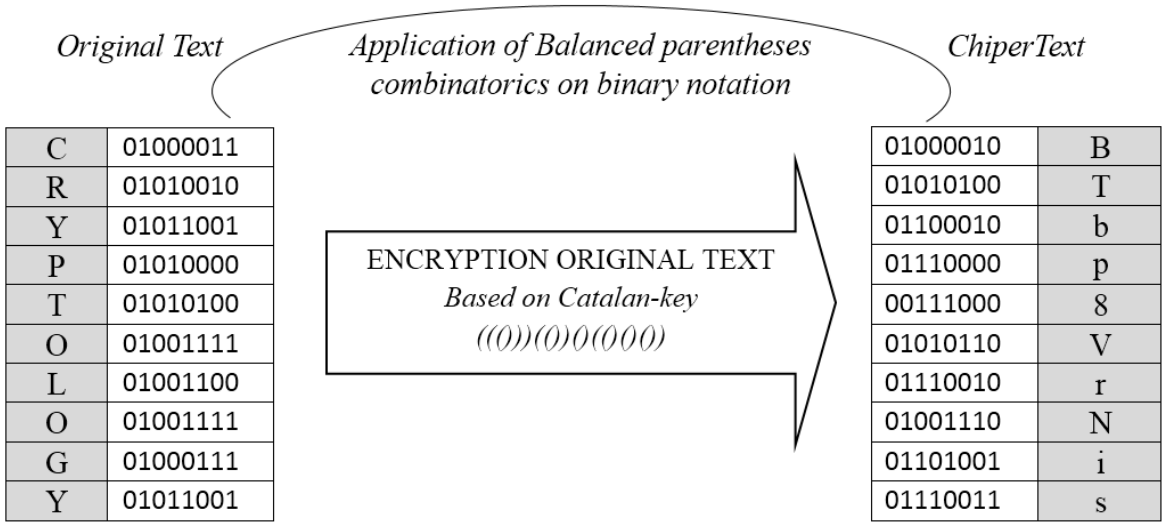


Figure 5. 9 Encryption of string CRYPTOLOGY to string BTbp8VrNis, based to Catalan-key

If we compare the open text P_{txt}***="CRYPTOLOGY"*** and the cipher C_{txt}***="BTbp8VrNis",*** we can see that the first character "Y" is replaced by „b" and the other character "Y" with „s". The same case is with the characters „O", where the first is replaced with the "V" and the other with „N".

In this way, we provide a stronger encryption mechanism, that is, one character is replaced by some completely different character, depending on the received bit permutation. In this case, we do not have the classic transposition cipher, as in the previous examples, but the needed substitution cipher. It is important to note that, here isn't realized the classic substitution, because one character from the message is not always replaced with the same character in the cipher. The substitution mode depends on the key itself and its length, as well

as from the bit key schedules. In addition, the process depends on the length of the message and the size of the segment in message that are taken in the encryption process.

CONCLUSION

Bearing in mind the fact that cryptography is a very dynamic discipline, that it is current and is very widespread, with this paper are covered only some of basic mathematical concepts and is given contribution to the application of combinatorics and computational geometry in the field of cryptography.

In this paper are considered the possibilities of applying of algorithm of simple polygon triangulation and the Catalan numbers, as well as combinatorial problems in cryptography. Theoretical bases of research are listed where the basic properties of Catalan numbers are examined, first of all, the emphasis is placed on the bit balance property in the binary record of Catalan number, which is related to the combinatorial problem - balanced parenthesis.

From the aspect of the achieved results, we can say that we have proposed the application of one polygon triangulation algorithm in the process of generating cryptological keys from one segment of the 3D image. The key thus obtained has the property of Catalan numbers that has a large keys space, which gives priority when cryptanalysis is concerned. In this case, these numbers serve as pseudo-generators, which in combination with the combinatorual problem Balanced parenthesis, can provide an effective mechanism for encrypting and decrypting text.

5.2 Application of the Computational Geometry in Linear Optimization

Computers have an important role in the automated construction and production of various items and objects today. The production process is a mathematical model that develops methods for the best outcome. These models are formulated as the maximization or minimization of some target function along with given constraints and can also be observed as problems of computational geometry. Computational geometry develops efficient algorithms for optimizing these models. Computer models can be created based on objects that really exist or some imaginary object. In practice, experimenting with created models is make with imaginary objects because experimenting with them is easier than with a real object. In this section, is given prune and search algorithm which is represents an example relation between linear programming and computational geometry.

Linear programming is a branch of mathematics that deals with the technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. The problem of linear programming is introduced from Leonid Kantorovich in 1939 as a method of solving the problem expenditures and returns to the army and increase losses incurred by the enemy. Also, in the United States, linear programming was developed during the II. World War primarily for problems of military logistics, such as optimizing the transportation of military and equipment to convoys. Linear programming as mathematical model is used in field of computational geometry. Computational geometry is a part of the field of algorithms and deals with the development and analysis of efficient algorithms and the structure of data suitable for geometric problems. As synthesis of geometry and computer sciences, computational geometry develops thanks to problems and applications, first of all in computer graphics, computer vision, robotics, databases, geographic information systems, Computer Aided Design / Computer Aided Manufacturing (CAD/CAM) systems, molecular biology, etc. Some of the concrete applications are applications in virtual reality, planning of movement, drug design, fluid dynamics, etc. The field of computer geometry usually deals with problems in the Euclidean plane or space and implies the availability of elementary operations such as: checking whether the point belong to line or circle, checking intersection of the lines or line segments [14]. In the introductory section we will give an overview of the optimization, focusing especially on convex optimization.

The mathematical problem of optimization, or just the problem of optimization, is problem from the following form

$$\begin{aligned} & min\, f\,(x) \\ \text{Subject to} \quad & f_i(x) \le b_i \quad i = 1, \dots m \end{aligned} \tag{5.1}$$

The vector $x = (x_1, \dots, x_n)^T$ is a optimization variable of problem, the function $f: \mathbb{R}^n \to \mathbb{R}$ is the objective function, the functions $f_i: \mathbb{R}^n \to \mathbb{R}$, $i = 1, \dots, m$, are the (inequality) constraint functions, and the constants $b_1, \dots, b_m$ are the limits, or bounds, for the constraints. A vector x^* is called optimal, or a solution of the problem (5.1), if it has the smallest objective value among all vectors that satisfy the constraints: $\forall z \in \mathbb{R}^n$ with $f_1(z) \le b_1, \dots, f_m(z) \le b_m$ we have $f(z) \ge f(x^*)$ [9].

A set $X = \{z | f_i(z) \le b_i,\ i = 1,2, \dots, m\}$ is called feasible region of problem (5.1), and $z \in X$ is feasible point. If $X = \emptyset$ than the problem (5.1) is called infeasible optimization problem. If the objective function of problem (5.1) is unbounded than the (5.1) is unconstrained optimization problem. Usually the families or classes of the optimization problems are characterized with the certain forms of objective function and the functions of constraints. As a special case, the optimization problem (5.1) is a problem of linear programming if the objective function f and the functions of constraints $f_1, \dots, f_m$ are linear, that is, the equations

$$\begin{aligned} & f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \\ & f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y) \end{aligned} \tag{5.2}$$

for any $x, y \in X \subseteq \mathbb{R}^n$ and any $\alpha, \beta \in \mathbb{R}$. The convex programming problem is the one in which the objective function and the functions of constraints are convex functions, which means that the inequalities hold

$$\begin{aligned} & f(\alpha x + \beta y) \le \alpha f(x) + \beta f(y) \\ & f_i(\alpha x + \beta y) \le \alpha f_i(x) + \beta f_i(y) \end{aligned} \tag{5.3}$$

for any $x, y \in X \subseteq \mathbb{R}^n$ and any $\alpha, \beta \in \mathbb{R}$ such that $\alpha + \beta = 1$, $\alpha \ge 0$, $\beta \ge 0$. If we compare (1.3) and (1.2) we can see that convexity is more general than linearity, equality is replaced

by inequality, and the inequality must apply only to certain values of α and β. Since the problem of linear programming is simultaneously the problem of convex programming, we can consider convex programming as a generalization of linear programming [9].

The solving method of optimization problem is an algorithm that calculates the solution of the problem to a certain accuracy. The efficiency of these algorithms, that is, the ability to solve the optimization problem (5.1), varies greatly and depends on factors such as certain types of objective function. The solving of problem (5.1) means that one of the following four conditions is fulfilled:

- The optimal solution of (1.1) is found
- It's shown that (1.1) is unbounded from the down on X
- It's proved that $x^* = \underset{x \in X}{inf} f(x)$ doesn't exist
- It's proved that (5.1) is infeasible problem.

5.2.1 Conceptual Determination

Linear programming is an optimization problem which maximizes a linear objective function under linear inequality constraints:

$$\max f\,(x_1, x_2, \dots, x_n) = c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{5.4}$$

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\ \text{Subject to}\quad &a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\ &\qquad\vdots \quad \vdots \quad \vdots \\ &a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \le b_m \end{aligned} \tag{5.5}$$

$$x_1 \ge 0, x_2 \ge 0, \dots, x_n \ge 0 \tag{5.6}$$

The objective function (5.4) is linear, where $c_j, j = 1,2, \dots, n$ are coefficients $x_j, j = 1,2, \dots, n$ are structural variables of objective function. The optimization problem (5.4) – (5.6) can be written on matrix form as a

$$\begin{aligned} &max\, f = c^T x \\ \text{Subject to}\quad &Ax \le b \\ &x \ge 0 \end{aligned} \tag{5.7}$$

$S = \{x | x \in \mathbb{R}^n, Ax \geq b, x \geq 0\}$ is set of possible solutions (feasible region) of problem (5.4). Feasible solution $x* \in S$ for which $f(x) \leq f(x*), \forall x \in S$ is optimal solution of problem (2.4), while $f(x*)$ is the optimal value of objective function.

Definition 5.1 The set $C \subseteq \mathbb{R}^n$ is convex if the line segment between any two points from *C* completely lies in *C*, for any $x_1, x_2 \in C$ and any $\theta, (0 \leq \theta \leq 1)$ we have $\theta x_1 + (1 - \theta)x_2 \in C$ [9].

Definition 5.2 A hyper-plane is a set of the form

$$\{x | a^T x = b\},$$

where $a \in \mathbb{R}^n$, $a \neq 0$ and $b \in \mathbb{R}$ [9].

Definition 5.3 A closed halfspace is a set of the form

$$\{x | a^T x \leq b\},$$

where where $a \neq 0$, *i.e.*, the solution set of one (nontrivial) linear inequality [9].

Definition 5.4 A polyhedron is the solution set of a finite number of linear equalities and inequalities:

$$\{x | a^T x \leq b, c^T x = d\}$$

where $a, c \in \mathbb{R}^n$, $a, c \neq 0$ and $b, d \in \mathbb{R}$ [9].

From definition we obtain that the polyhedron is intersection of a finite number of half-spaces and hyper-planes. Hyper-plane, halfspace and polyhedron are convex sets. A bounded polyhedron is called a polytope. Optimization problem which minimized the linear objective function is the dual problem of (5.7) and can be written in the form

$$\begin{aligned} & min\, f = b^T \lambda \\ \text{Subject to} \quad & A^T \lambda \geq c \\ & \lambda \geq 0 \end{aligned} \tag{5.8}$$

5.2.2 Linear Programming in Computational Geometry

Often in the application of computational geometry, the problems of linear programming appear [45]. In computational geometry, randomized algorithms are used that

give the possibility of treating geometric problems in the general case. Let $f(x_1, x_2, \dots, x_n)$ is objective function of LP problem an let half-space of LP problem is defined by non-zero vector $a_i = \{a_{i1}, a_{i2}, \dots, a_{in}\}$ and real number b_i such that $s_i = \{x | a_i^T x \leq b_i\}$ for $i = 1,2, \dots, n$. We partition the set of half-spaces into three sets S^-, S^0 and S^+ such that $s_i \in S^-$ if $a_i^T x < 0$, $s_i \in S^0$ if $a_i^T x = 0$ and $s_i \in S^+$ if $a_i^T x > 0$.

Definition 5.5 $P(L) = \bigcap_{i=1}^{n} s_i$ is the feasible domain of LP problem, and the range of LP problem is

$$\Sigma(L) = \bigcap_{s \in S^0} s$$

If $P(L) \neq \emptyset$ we said LP problem is feasible, and if $P(L) = \emptyset$ than LP problem is infeasible. A linear programming problem in two dimensions is one which involves only two variables, x_1 and x_2 where each constraint is a half-plane in E^2 [13]. In this paper is given the prune-and-search paradigm algorithm who solves a linear program defined by n half-planes in time O(n). The global structure of the algorithm follows the search step who decreases the range of possible solutions, and a prune step eliminates data which is irrelevant in this range [34]. After a search and a prune step, we simply recur with the smaller set of data until the problem becomes trivial. A prune step is typically straightforward, while search steps require the sophistication for design. The main purpose in a search step is to decrease the range of possible solutions in a way that allows us to eliminate a proportional amount of the data. Thus, a search step consists of two steps which may be iterated a constant number of times: first, we find a suitable test, and second, we answer this test. In our case, a test comes as a vertical line, and we decide on which side of this line we are going to continue the search for a solution. Let with *D* we denote the set of data (*n* elements) and with *E* the range which contain all solutions [19]. Bellow is given the algorithm for LP problem in two dimension.

ALGORITHM 5. 2 Prune-and-search

if the size of *D* is at most some constant the
  Use a trivial procedure to solve the problem.
else
  SEARCH: Iterate the following two steps some constant number of
    times:
    FIND_TEST: Find an appropriate test *t.*
    BISECT: Decrease the range *E* which contains all solutions by
    answering the test *t.*
  PRUNE: Eliminate some subset of *D* which is irrelevant in *E.*
  RECUR: Repeat the computation for the new sets *D* and *E.*
endif.

In this section we make the analysis for steps in Algorithm 5.2 and we give a solution of the numerical example with algorithm. To this end, we will look first **the selection problem:** Given a set $S = \{a_1, a_2, \dots, a_n\}$ of n elements on which a linear ordering is defined, and an integer $k, 1 \leq k \leq n$, find the k −th smallest element in the set. The concept of the prune-and-search paradigm in selection problem consist the following steps:

- $S = \{a_1, a_2, \dots, a_n\}$ is a set of n elements
- With $p \in S$, the set S is partitioned into 3 subsets S_1, S_2, S_3:
  - $S_1 = \{ a_i \,|a_i < p, 1 \leq i \leq n\}$
  - $S_2 = \{ a_i \,|a_i = p, 1 \leq i \leq n \}$
  - $S_3 = \{ a_i \,|a_i > p, 1 \leq i \leq n \}$
- For partitioned subsets we have this three cases:
  - If $|S_1| \geq k$, then the k−th smallest element of S is in S_1, prune away S_2 and S_3.
  - Else, if $|S_1| + |S_2| \geq k$, then p is the k–th smallest element of S.
  - Else, the k−th smallest element of S is the $(k - |S_1| - |S_2|)$ –th smallest element in S_3, prune away S_1 and S_2.

Let see how the prune-and-search paradigm can be used for develop a linear-time algorithm for the two-dimensional linear programming problem [55, 56]. A general two-dimensional *LP* problem with inequality constraints is given as follows

$$\begin{aligned} & min\, c_1 x_1 + c_2 x_2 \\ \text{Subject to} \quad & a_{i1}x_1 + a_{i2}x_2 + a_{i0} \leq 0 i = 1,2,\dots,n \end{aligned} \tag{5.9}$$

The problem is solved if one can illustrate the feasible region satisfying the inequality constraints in the (x_1, x_2)–plane, who represent a convex polygon (see Figure 2.1). Here, instead of considering the problem in this general form, we restrict our attention to the following problem

$$\begin{aligned} & min\, y \\ \text{Subject to} \quad & a_i x + y + b_i \leq 0 i = 1,2,\dots,n \end{aligned} \tag{5.10}$$

This is a special problem for linear-time algorithm for the general two-dimensional problem, which is simpler structure for better exhibit the essence of the prune-and-search technique. Figure 5.10 depicts this restricted problem for $n = 8$. Let the problem is defined as above, we define a function $f(x)$ by $f(x) = max\{a_i x + b_i | i = 1,2,\dots,n\}$. Minimizing of $f(x)$ is equivalent to LP problem. The graph of $y = f(x)$ is drawn in red lines in Figure 2.1 an represent a convex function.

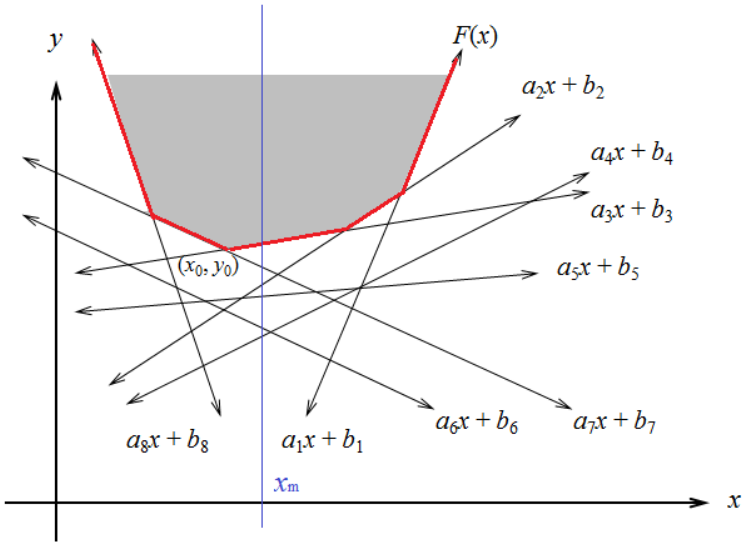


Figure 5.10 A two dimensional LP problem

Let we choose the point x_m on x – axis. If $x_0 < x_m$ and the intersection of $a_3 x + b_3$ and $a_2 x + b_2$ is greater than x_m, then one of these two constraints is always smaller than the other for $x < x_m$. Thus, this constraint can be deleted. It is similar for $x_0 > x_m$. Let

$y_m = f(x_m) = \max\limits_{1 \le i \le n}\{a_i x_m + b_i\}$ and suppose an x_m is known. How do we know whether $x_0 < x_m$ or $x_0 > x_m$? To get answer this question we must look two cases.

- **Case 1:** y_m is on only one constraint and let g denote the slope of this constraint.
  - If $g > 0$, then $x_0 < x_m$.
  - If $g < 0$, then $x_0 > x_m$.
- **Case 2:** y_m is the intersection of several constraints and

  $g_{max} = \max\limits_{1 \le i \le n}\{a_i | a_i x_m + b_i = f(x_m)\}$ is maximal slope and

  $g_{min} = \min\limits_{1 \le i \le n}\{a_i | a_i x_m + b_i = f(x_m)\}$ is minimal slop of constraints.
  - If $g_{min} > 0, g_{max} > 0,$ then $x_0 < x_m$
  - If $g_{min} < 0, g_{max} < 0,$ then $x_0 > x_m$
  - If $g_{min} < 0, g_{max} > 0$, then (x_m, y_m) is the optimal solution.

Now the question arises as to how to choose x_m? Arbitrarily must be grouped the n constraints into n / 2 pairs. For each pair, must be found their intersection. Among these n /2 intersections, must be choose the median of their x-coordinates as x_m.

The prune – and – search approach with Input: $S: a_i x + b_i, i = 1, 2, \dots, n$ (constraints) and Output: the value x_0 such that y is minimized at x_0 subject to the above constraints, consist the following steps.

- **Step 1**: If S contains no more than two constraints, solve this problem by a brute force method.
- **Step 2**: Divide S into $\frac{n}{2}$ pairs of constraints randomly. For each pair of constraints $a_i x + b_i$ and $a_j x + b_j$, find the intersection p_{ij} of them and denote its $x-$value as x_{ij}.
- **Step 3**: Among the x_{ij}'s, find the median x_m.
- **Step 4**: Determine $y_m = f(x_m) = \max\limits_{1 \le i \le n}\{a_i x_m + b_i\}$

  $$g_{min} = \min_{1 \le i \le n}\{a_i | a_i x_m + b_i = f(x_m)\}$$

  $g_{max} = \max\limits_{1 \le i \le n}\{a_i | a_i x_m + b_i = f(x_m)\}$
- **Step 5**:

  Case 5a: If $g_{\min}$ and g_{max} are not of the same sign, y_m is the solution and exit.

  Case 5b: otherwise, $x_0 < x_m$, if $g_{\min} > 0$, and $x_0 > x_m$, if $g_{\min} < 0$.
- **Step 6**:

Case 6a: If $x_0 < x_m$, for each pair of constraints whose x −coordinate intersection is larger than x_m, prune away the constraint which is always smaller than the other for $x \leq x_m$.

Case 6b: If $x_0 > x_m$, do similarly.

Let *S* denote the set of remaining constraints. Go to Step 2.

There are totally $\left\lfloor \frac{n}{2} \right\rfloor$ intersections. Thus, $\left\lfloor \frac{n}{4} \right\rfloor$ constraints are pruned away for each iteration. Time complexity: $T(n) = T\left(\frac{3n}{4}\right) + O(n) = O(n)$ [61].

The general two-variable linear programming problem solution is a procedure of finding piece - wise linear convex function of the x − axis. The problem (5.9) can be transformed by setting $Y = c_1x_1 + c_2x_2$ and $X = x$ as follows

$$\min Y \tag{5.11}$$

$$\text{Subject to} \quad \alpha_i X + \beta_i Y + a_{i0} \leq 0 i = 1,2, \dots, n$$

where

$$\alpha_i = \left(a_{i1} - \frac{c_1}{c_2}\right) a_{i2} \qquad \text{and} \qquad \beta_i = \frac{a_{i2}}{c_2}.$$

Depending upon whether is zero, negative or positive we partition the index set on three subsets I_0 , I_- , I_+ respectively. All constraints whose index is in I_0 are vertical lines and determines the feasible intervals for X as follows

$$u_1 \leq X \leq u_2$$
$$u_1 = max\{-a_{i0}/\alpha_i | i \in I_0, \alpha_i < 0\}$$
$$u_2 = min\{-a_{i0}/\alpha_i | i \in I_0, \alpha_i > 0\}$$

In the other hand letting $-\left(\frac{\alpha_i}{\beta_i}\right) \triangleq \delta_i$ and $-\left(\frac{a_{i0}}{\beta_i}\right) \triangleq \gamma_i$ all constraints on I_+ are of the form

$$Y \leq \delta_i X + \gamma_i \qquad i \in I_+$$

so that collectively define a piece – wise linear upward – convex function $F_+(x)$ of the form

$$F_+(x) \triangleq \min_{i \in I_+}(\delta_i X + \gamma_i).$$

Similarly the constraints in I_- defines piece-wise linear downward – convex functions $F_-(x)$ of the form

$$F_-(x) \triangleq \max_{i \in I_-}(\delta_i X + \gamma_i).$$

In this way we obtain the transformed constraint $F_-(x) \leq Y \leq F_+(x)$, and since we have minimizing LP problem $F_-(x)$ is our objective function. The problem is

$$\begin{aligned} & \min F_-(x) \\ \text{Subject to} \quad & F_-(X) \leq F_+(X) \\ & u_1 \leq X \leq u_2 \end{aligned}$$

Let $H(x) = F_-(x) - F_+(x)$.

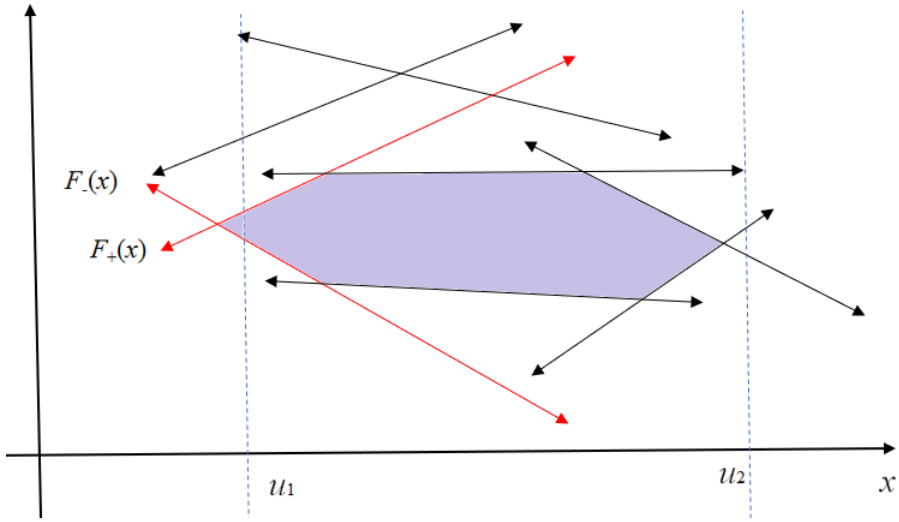


Figure 5. 11 Ilustration $F_-(x)$, $F_+(x)$, u_1 and u_2 in the reformulation of the LP problem

If we know $x_0 < x_m$, then $a_1x + b_1$ can be deleted because $a_1x + b_1 < a_2x + b_2$ for $x < x_m$.

Define:

$g_{\min} = \min\{a_i \mid i \in I_-,\ a_i x_m + b_i = F_-(x_m)\}$, minimal slope

$g_{\max} = \max\{a_i \mid i \in I_-,\ a_i x_m + b_i = F_-(x_m)\}$, maximal slope

$h_{\min} = min\{a_i \mid i \in I_+,\ a_i x_m + b_i = F_+(x_m)\}$, minimal slope

$h_{\max} = max\{a_i \mid i \in I_+,\ a_i x_m + b_i = F_+(x_m)\}$, maximal slope

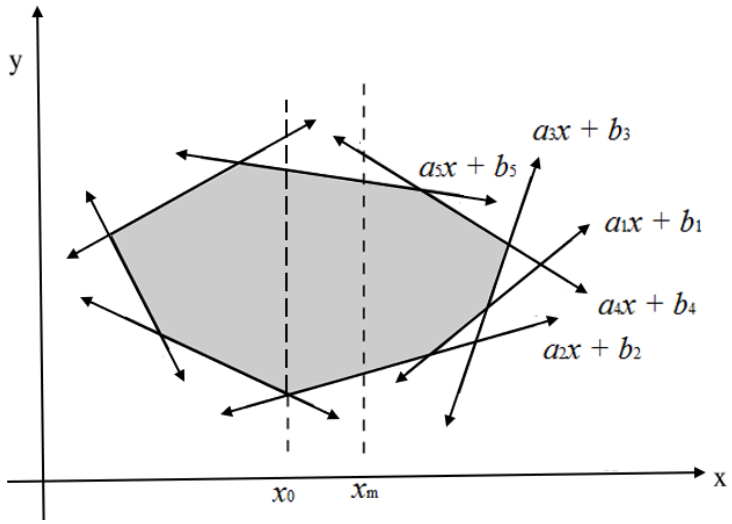


Figure 5. 12 Ilustration of possible cases for $F_-(x) > F_+(x)$

- **Case 1:** If $F(x_m) \leq 0$, then x_m is feasible.
  - If $g_{\min} > 0, g_{max} > 0$, then $x_0 < x_m$.
  - If $g_{\min} < 0, g_{max} < 0$, then $x_0 > x_m$.
  - If $g_{\min} < 0, g_{max} > 0$, then x_m is the optimum solution.
- **Case 2:** If $F(x_m) > 0, x_m$ is infeasible.
  - If $g_{\min} > h_{max}$, then $x_0 < x_m$.
  - If $g_{\min} < h_{max}$, then $x_0 > x_m$.
  - If $g_{\min} \leq h_{max}$, and $g_{max} \geq h_{min}$, then no feasible solution exists.

The prune – and – search approach for general two variable LP problem with Input:

$I_- : y \geq a_i x + b_i, i = 1, 2, \ldots, n_1 \quad I_+ : y \leq a_i x + bi, i = n_1 + 1, n_1 + 2, \ldots, n, a \leq x \leq b$

(constraints) and Output: the value x_0 such that y is minimized at x_0 subject to the above constraints, consist the following steps.

- **Step 1**: Arrange the constraints in I_1 and I_2 into arbitrary disjoint pairs respectively. For each pair, if $a_i x + b_i$ is parallel to $a_j x + b_j$, eliminate $a_i x + b_i$ if $b_i < b_j$ for

$i, j \in I_-$ or $b_i > b_j$ for $i, j \in I_+$. Otherwise, find the intersection p_{ij} of $y = a_i x + b_i$ and $y = a_j x + b_j$. Let the $x-$coordinate of p_{ij} be x_{ij}.

- **Step 2**: Find the median x_m of x_{ij}'s (at most $\left\lfloor \frac{n}{2} \right\rfloor$ points).
- **Step 3**:
    - If x_m is optimal, report this and exit.
    - f no feasible solution exists, report this and exit.
    - Otherwise, determine whether the optimum solution lies to the left, or right, of x_m.
- **Step 4**: Discard at least 1/4 of the constraints. Go to Step 1.

The general approach given on this section by Megiddo is applied for minimum enclosing circle of n point set.

CONCLUSION

Linear programming as a central problem in the discrete-algorithm study plays a very important role in solving numerous combinatorial optimization problems. Because of its various applications in many areas, the problem of linear programming is gaining great attention in the field of computational geometry. Linear programming can also be viewed as computational geometry problems in which the feasible region is the cross-section of the half-spaces determined by their constraints. For these problems, the target function is minimized or maximized in the convex polyhedron field. There are several known problems in computational geometry such as the smallest circle, extreme point, farthest point that are closely connected from the *n* points in the plane. These problems are considered as problems of linear programming with n variables and in the end dimension $\mathbb{R}^2$ consists of finding a point P which is a convex combination of other *n* points from $\mathbb{R}^2$. Another problem of computational geometry that is serious for O (*n*) is the problem of finding the smallest circle enclosing *n* given points in the plane. In the end we can conclude that the linear programming give good basis for further investigation in the low dimensional space treated in computational geometry.

5.3 Convex Polygon Triangulation based on Ballot problem and Planted Trivalent Binary Tree

In this section is presented a new technique of generation of convex polygon triangulation based on planted trivalent binary tree and ballot notation. The properties of Catalan number were examined, are given their decomposition and application in developing of the hierarchy and triangulation trees. The method presented in the section was constructed on basis of ballot combinatorial problem. The movements in constructed method through polygon are derived upon vertices and leaves of the planted trivalent binary tree. In the section are given two algorithms who are reverse to each other and transform the triangulation to ballot record and vice versa. The research subject of the section is analysis, testing, and comparison of a constructed method for solving of convex polygon triangulation problem with other methods and generating of graphical representation. The application code of the algorithms is done in the Java programming language.

The polygon triangulation is a bigger important problem of the polygon partition which is applied in a computational geometry. The polygon triangulation algorithms are developed based on a ballot record and the lattice path problem. In algorithms, the ballot records are constructed with the planted trivalent binary trees (*PTBT*). These records are obtained with the selection of the particular edge of the polygon as a base and through which is entered in the tree from the starting position.

These records are obtained with the selection of the particular edge of the polygon as a base and through which is entered in the tree from the starting position. The implementation of our method is realized through the following three phases:

1) Generation of a complete triangulation tree from the initial basic triangle ($n = 3$) to the given n-gon. The resulting triangulation hierarchy is based on the decomposition of Catalan numbers (this procedure is described in detail in Section 5.2).
2) Generation of individual triangles within each level of the triangulation tree based on the Planted trivalent binary tree (this procedure is described in detail in Section 5.4).
3) Storage of obtained triangulation based on Ballot record i.e. the notation (this procedure is described in detail in Section 5.3).

Two new algorithms are presented. The algorithms "*Ballot notation for PTBT to triangulation*" and "*Triangulation to ballot notation for PTBT*" are inverse to each other. The

Ballot notation for *PTBT* to triangulation algorithm generates convex polygon triangulation based on ballot records obtained from movements through vertices and leaves 1 of the planted trivalent binary tree. The inverse algorithm with movements through vertices and leaves of the planted trivalent binary tree generate the ballot record for convex polygon triangulation.
Other sections of the section are organized in the following order: Section 5.3.1, consists of some preliminary exposures related to polygonal triangulation, binary trees, and Catalan numbers decomposition. Section 5.3.2, presents the problem of ballot records, lattice paths and the correspondence of a lattice path with the well-formed sequences of parentheses. Section 5.3.3, contains the method for construction of convex polygon triangulations based on a ballot records and a planted trivalent binary. The comparative analysis of experimental results for Ballot Lattice, Ballot Trivalent, and Hurtado trees are given in Section 5.3.4. Also in this section is the complexity of algorithms from the aspect of the data storage amount for triangulations and number of operations for generating triangulations. The final section provides concluding observations and possible directions for further research in this domain.

5.3.1 Preliminaries about trees and hierarchy of triangulations

The trees as an important class of graph theory [21] also have an important use in the triangulation of polygon and they are defined as an acyclic connected graph [27, 58, 61]. The trees with root are rooted trees. In rooted trees, the roots are drawn at the top and they grow downward. A rooted tree in which the vertices at each level are ordered as the first, second, third and so on is an ordered tree. An ordered rooted tree is a binary tree if each vertex has degree less than two (each vertex has most two children; the left and the right child) [41]. The number of non–isomorphic binary trees with n vertices that can be drawn has a direct connection with the Catalan numbers and their relationship is given with the following theorem. The number of binary trees with n vertices is C_n.

A binary tree is planted trivalent if the degree of its root is one and that of each internal vertex is three. By deleting the root of a planted trivalent binary tree, we get an ordinary binary tree, by attaching a new root at the existing root of a binary tree, we get a planted trivalent binary tree. Thus there is a bijection between set of planted trivalent binary trees with n vertices and set of binary trees with $n-1$ vertices. The number of triangulation of convex n-gon is equal of number of planted trivalent binary tree with $n-1$ leaves [40].

Let P_n stand for convex polygon with n vertices and $\mathbf{T}_n$ for set of all triangulations of P_n and τ_n denotes a particular triangulation from $\mathbf{T}_n$. Also $deg(i)$ denotes the degree of a

vertex i in a triangulation. The vertex i satisfying $deg(i) = 2$ is called an ear. It is well-known that the number of triangulations $\mathbf{T}_n$ of polygon $\mathrm{P_n}$ is equal to $(n-2)$-th Catalan number, denoted by $\mathrm{C_{n-2}}$:

$$T_n = C_{n-2} = \frac{1}{n-1}\binom{2n-4}{n-2} = \frac{(2n-4)!}{(n-1)!\,(n-2)!}, \quad n \geq 3 \tag{5.9}$$

In [33] Hurtado and Noy have suggested an algorithm for graph of triangulations of a convex polygon and tree of triangulations, where triangulations of P_n are derived from triangulations of P_{n-1}. Their procedure consists of "*splitting*" polygon diagonals (both internal and external which are polygon edges) incident to the highest mark vertex ($n-1$ for P_{n-1}). If we perform splitting of these diagonals $\delta_{i,n-1}, i \in \{1,2,\dots,n-2\}$ in increasing order of i we get an ordering of triangulations of P_n. Moreover, Hurtado and Noy define the infinite tree of triangulations for all convex polygons where at tree level n we have all triangulations of P_n.

This algorithm has inspired us to consider a specific decomposition of Catalan number which could be used to faster triangulations generation. As a result of that decomposition, we have an expression containing terms of the form $(2+i), 0 \leq i \leq n-4$, that can be used in generating the trees of triangulation of a convex polygon P_n based on the triangulations of P_{n-1}. Obtained expressions indicate how many P_{n-1} triangulations appear as the parts of P_n triangulations. Also, they indicate triangulations of P_n which do not contain some of P_{n-1} triangulations. It should be noted that we get the triangulations of P_n in the same ordering as Hurtado in [33]. Moreover, every triangulation in $\mathbf{T}_n$ has a parent in $\mathbf{T}_{n-1}$ and a specific number of exactly defined descendants in $\mathbf{T}_{n+1}$.

Hurtardo and Noy have proposed an algorithm to generate the triangulations of P_n based on the triangulations of P_{n-1}. Moreover, they defines the tree of triangulation where all triangulations of P_n, i.e. the triangulations from $\mathbf{T}_n$, are arranged at the level n of this tree. Each triangulation at the level n has a "*father*" in $\mathbf{T}_{n-1}$ and two or more "*sons*" in $\mathbf{T}_{n+1}$. The sons of the same father are "*brothers*". There is an ordering among the children of a triangulation, and consequently among all triangulations. Implementation of this algorithm in three programming languages (*Java*, *Python* and *C++*) is analyzed in this dissertation [73].

Let us denote a triangulation $\tau_{n-1} \in \mathbf{T}_{n-1}$ satisfying $\deg(n-1) = l$ by τ_{n-1}^{l}. Assume that diagonals incident to $n-1$ are sorted from the left by $\delta_{i_k,n-1}, k = 1,\dots,l$. Then

the number of diagonals incident to $n-1$ which are located left from $\delta_{i_k,n-1}$ is equal to $k-1$. The sons of τ_{n-1}^{l} are derived by "*splitting*" the diagonals incident to vertex $n-1$

$$\delta_{i_k,n-1},\ k=1,\dots,l,\ i_k\in\{1,\dots,n-2\},\ 1=i_1<i_2<\cdots<i_l=n-2$$

in increasing order with respect to i_k. If we split the diagonal $\delta_{i_k,n-1}$ we get the son $S^{i_k}(\tau_{n-1}^{l})$. Then the sons of triangulation τ_{n-1}^{l} are

$$S^{i_1}(\tau_{n-1}^{l}), S^{i_2}(\tau_{n-1}^{l}),\dots,S^{i_l}(\tau_{n-1}^{l}). \tag{5.10}$$

The splitting of the diagonal $\delta_{i_k,n-1}$ produces the son $S^{i_k}(\tau_{n-1}^{l})$ with $deg(n)=2+k-1$. We are assigning the weights of the form $(2+i)$ to the edges in the tree of triangulations. When numbering the edges in the tree of triangulations we will be guided by the basic principle that each weight of an edge represents the number of descendants for a triangulation at end of this edge. The number of τ_{n-1}^{l} descendants is between 2 and $n-2$. So, from one particular triangulation of P_{n-1} we can derive $2+i$ triangulations of P_n where $i\in\{0,1,\dots,n-4\}$ in the general case. As the number of descendants is greater than or equal to 2, the usage of the weights $(2+i)$, $i\in\{0,1,\dots,n-4\}$ is obvious.

By $\left(\tau_{n-1}^{l},S^{i_k}(\tau_{n-1}^{l})\right)$ we denote the edge in the tree of triangulation connecting the nodes $\tau_{n-1}^{l}\in\mathbf{T}_{n-1}$ and $S^{i_k}(\tau_{n-1}^{l})\in\mathbf{T}_n$between the levels $n-1$ and n. The triangulation $S^{i_k}(\tau_{n-1}^{l})$ is derived by splitting the diagonal $\delta_{i_k,n-1}$. Since the diagonal $\delta_{i_k,n-1}$ has $k-1$ diagonals left to it and incident to the vertex $n-1$, by splitting this diagonal we get $2+k-1$ descendants of the triangulation $S^{i_k}(\tau_{n-1}^{l})$ (i.e. $2+k-1$ diagonals incident to n). Then, we assign the expressions of the form $(2+i)$ as the weights to outgoing edges of τ_{n-1}^{l}, namely

$$\left(\tau_{n-1}^{l},S^{i_1}(\tau_{n-1}^{l})\right),\dots,\left(\tau_{n-1}^{l},S^{i_l}(\tau_{n-1}^{l})\right) \tag{5.11}$$

The weight $(2+k-1)$ of the edge $\left(\tau_{n-1}^{l}, S^{i_k}\left(\tau_{n-1}^{l}\right)\right)$ means that the triangulation $\tau_{\mathrm{n}-1}^{l}$ has $k-1$ diagonals incident to vertex $n-1$ left to $\delta_{i_k,n-1}$, and that $\mathrm{S}^{\mathrm{i_k}}\left(\tau_{\mathrm{n}-1}^{\mathrm{l}}\right) \in \mathrm{T_n}$ has $2+k-1$ descendants.

5.3.2 Tree of triangulations based on expression of Catalan numbers

According to equation (5.9), the sons of τ_{n-1}^{l} are derived by splitting the diagonals $\delta_{i_1,n-1}, \dots, \delta_{i_l,n-1}$. The edges (5.10) have the following weights

$$(2+0), (2+1), \dots, (2+l-1),$$

respectively.

The triangulation $\tau_{\mathrm{n}-1}^{l}$ has l descendants. According to the adopted principle of assigning weights to the tree of triangulations, incoming edge to node $\tau_{\mathrm{n}-1}^{l}$ has the weight $2+l-2$. In the origin (before the tree root), there is the expression $(2+0)$, which indicates that the triangle in the tree root has an ear in vertex 3, i.e. has 2 descendants (these are two possible triangulations of a quadrilateral). The sum of all weights of the edges connecting tree levels $\mathrm{n}-1$ and n is equal to C_{n-1}, and the number of summands, i.e. the number of these edges is equal to C_{n-2} [85].

By $\Delta(\mathrm{C_{n-1}})$ we denote the decomposition of $\mathrm{C_{n-1}}$ defined by summing all weights of edges connecting tree levels $n-1$ and n. The cardinal number of basic terms of the form $(2+\mathrm{i})$ in $\Delta(\mathrm{C_{n-1}})$ is equal to $\mathrm{C_{n-2}}$. The sum of terms included in $\Delta(C_{n-1})$ is equal to C_{n-1}.

Can we enlist some more properties of our decomposition $\Delta(\mathrm{C_n})$? The incoming edge to node $\tau_{\mathrm{n}+1}^{l}$ with the weight $2+i$ produces the edges outgoing to the node $\tau_{\mathrm{n}+1}^{l}$ with weights $(2+0), (2+1), \dots, (2+i+1)$. Therefore, our decomposition of Catalan number $\mathrm{C_n}$ can be derived from the already made decomposition of $\mathrm{C_{n-1}}$ and the transformation f defined by

$$(2+i) \to (2+0)+(2+1)+\cdots+(2+i+1) = f(2+i),\ i \geq 0. \qquad (5.12)$$

Further, assume the operator f is distributive with respect to+:

$$f\left(\sum(2+i)\right) = \sum f(2+i) = \sum\sum_{l=0}^{i+1}(2+l).$$

The first useful property is discovered in the following lemma.

Lemma 5.1 *For each* $n \geq 4$ *the recurrent relation* $\Delta(C_n) = f\big(\Delta(C_{n-1})\big)$ *is valid.*

There is C_{n-2} edges between tree levels $n-1$ and n. For each l, the edge weights

$$(\tau_n^l, S^{i_1}(\tau_n^l)), \dots, (\tau_n^l, S^{i_l}(\tau_n^l)) \tag{5.13}$$

are defined applying the function f on weight of some of the edges

$$\left(\tau_{n-1}^k, S^{i_p}\left(\tau_{n-1}^k\right)\right),\ k, p \leq l.$$

As the sum of weights, belonging to edges connecting tree levels $n-2$ and $n-1$, is equal to C_{n-2} and the sum of weights assigned to edges connecting tree levels $n-1$ and n, is equal to C_{n-1}, it is clear that the following holds $\Delta(C_{n-1}) = f\big(\Delta(C_{n-2})\big)$, or $\Delta(C_n) = f\big(\Delta(C_{n-1})\big)$ in general case.

In the paper [11] is presented a way of decomposition of Catalan numbers that can be used to generate tree of triangulation (or hierarchy of triangulation). For the simplicity we use the notation

$$\sigma_i = (2+0) + (2+1) + \cdots + (2+i) \tag{5.14}$$

Then (5.12) should be simplified to $f(\sigma_0) = \sigma_1,\ f(2+i) = \sigma_{i+1},\ i \geq 1$.

Further, using $f(\sigma_i) = \sigma_1 + \cdots + \sigma_{i+1} = \sum_{k=1}^{i+1} \sigma_k$, $i \geq 0$ decompositions in the paper (11) should be further simplified:

$$\begin{aligned}
&\Delta(\mathrm{C}_2) = (2+0) = \sigma_0\\
&\Delta(\mathrm{C}_3) = \sigma_1\\
&\Delta(\mathrm{C}_4) = \sum_{l=1}^{2} \sigma_l\\
&\Delta(\mathrm{C}_5) = \sum_{l=1}^{2} \sigma_l + \sum_{l=1}^{3} \sigma_l\\
&\Delta(C_6) = \left(\sum_{l=1}^{2} \sigma_l + \sum_{l=1}^{3} \sigma_l\right) + \left(\sum_{l=1}^{2} \sigma_l + \sum_{l=1}^{3} \sigma_l + \sum_{l=1}^{4} \sigma_l\right)
\end{aligned} \tag{5.15}$$

Now, we replace

$$\mathrm{f}\left(\sum_{l=1}^{i}\sigma_l\right)=\sum_{l=1}^{i}f(\sigma_l)=\sum_{l=1}^{i}\sum_{l_1=1}^{l+1}\sigma_{l_1}\text{ , i}\geq 1$$

By this we simplify (5.15) and get:

$$\Delta(\mathrm{C}_2)=\sigma_0$$

$$\Delta(\mathrm{C}_3)=f(\sigma_0)=\sigma_1$$

$$\Delta(\mathrm{C}_4)=f(\sigma_1)=\sum_{l=1}^{2}\sigma_l$$

$$\Delta(\mathrm{C}_5)=\sum_{l=1}^{2}f(\sigma_l)=\sum_{l=1}^{i}\sum_{l_1=1}^{l+1}\sigma_{l_1}$$

$$\Delta(C_6)=\sum_{l=1}^{2}f^2(\sigma_l)=\sum_{l=1}^{i}\sum_{l_1=1}^{l+1}f(\sigma_{l_1})=\sum_{l=1}^{i}\sum_{l_1=1}^{l+1}\sum_{l_2=1}^{l_1+1}\sigma_{l_2}.$$

Theorem 5.2 *Decomposition of* C_n *defined by summation of all weights of edges connecting tree levels* n *and* $n+1$ *is defined by*

$$\Delta(C_n)=f^{n-2}(\sigma_0)=f^{n-3}(\sigma_1)=\sum_{k=1}^{2}f^{n-4}(\sigma_k)=\sum_{l=1}^{i}\sum_{l_1=1}^{l+1}\cdots\sum_{l_{n-4}=1}^{l_{n-5}+1}\sigma_{l_{n-4}}\text{ , }n\geq 3 \qquad (5.16)$$

where σ_i is defined in (5.11).

It is sufficient to verify the inductive step. From the inductive hypothesis and Lemma 5.1 we get

$$\Delta(C_n)=f\big(\Delta(C_{n-1})\big)=f\left(\sum_{l=1}^{i}\sum_{l_1=1}^{l+1}\cdots\sum_{l_{n-5}=1}^{l_{n-6}+1}\sigma_{l_{n-5}}\right)$$

$$=\sum_{l=1}^{i}\sum_{l_1=1}^{l+1}\cdots\sum_{l_{n-5}=1}^{l_{n-6}+1}f(\sigma_{l_{n-5}})=\sum_{l=1}^{i}\sum_{l_1=1}^{l+1}\cdots\sum_{l_{n-4}=1}^{l_{n-5}+1}\sigma_{l_{n-4}}$$

which completes the proof.

Besides $\Delta(C_n) = f\big(\Delta(C_{n-1})\big)$ the following recurrent relation is also satisfied. For each n ≥ 4 the general recurrent relation holds $\Delta(C_n) = f^k\big(\Delta(C_{n-k})\big),\ k \geq 1$.

We have presented a Catalan number decomposition into the sum of terms of the form $(2+i), i \in \{0, \dots, n-4\}$. This kind of expression guide us in generation of $\boldsymbol{T}_n$ using already known $\boldsymbol{T}_{n-1}$. Our decomposition is unique and suitable for generation of tree of triangulations (see Figure below).

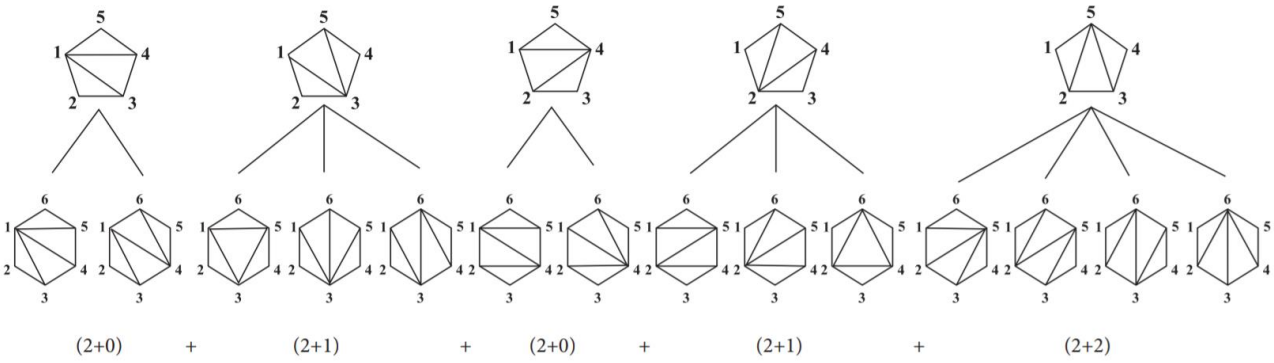


Figure 5.13 Levels five and six of the tree of triangulations based on decomposition of Catalan numbers

This idea with Catalan numbers decomposition was used in the construction of an algorithm for the Ballot Trivalent Trees method that is presented in Section 5.4. In this way, we perform triangulation of the convex polygon based on graph of triangulations of a convex polygon and tree of triangulations (similar principles as Hurtado in [3] achieving significant speedup of the algorithm.

5.3.3 Relationship of Catalan numbers and Ballot problem

The ballot problem can be illustrated graphically by a lattice paths under certain constraints in the Cartesian coordinate system [40]. In the two-dimensional space, by paths from the lattice point $(0,0)$ to the lattice point (n, n) we mean a directed paths who begin in $(0,0)$ and end in (n, n). The paths that are always underneath diagonal and the passes through lattice only with movements right (*R*) and up (*U*) are called legal paths. In the Cartesian coordinate system lets define two movements starting at the point (0, 0):

$$\boldsymbol{R}\colon (\boldsymbol{x}, \boldsymbol{y}) \longrightarrow (\boldsymbol{x}+\boldsymbol{1}, \boldsymbol{y}) \ \text{ and } \ \boldsymbol{U}\colon (\boldsymbol{x}, \boldsymbol{y}) \longrightarrow (\boldsymbol{x}, \boldsymbol{y}+\boldsymbol{1}).$$

We see that with R is defines the path one unit on the right, and with U one unit up in the lattice. With the application of these movements, we obtain that the problem of different paths from the point $(0,0)$ to the point (n,n) in the lattice correspond to the ballot problem. Every legal path on lattice must begin with an A and end in an B. The number of valid movement through in the $n \times n$ grid is Catalan number C_n (for details on the combination of Ballot notation and Lattice Path or problem of movement in discrete grid, see [80].

Example 5.1 *The following Table can be formed for the voting sequence* ***ABAABB****. This voting sequence corresponds to the movement through the lattice path* [78]. *Note that integers within a row denote numbers of the specific vote occurrence.*

Table 5.1 Tally of the votes in the ballot record ***ABAABB***

Tally for candidate	i_1	i_2	i_3	i_4	i_5	i_6
	A	***B***	***A***	***A***	***B***	***B***
A	1	1	2	3	3	3
B	0	1	1	1	2	3

There is a bijection between the set of well-formed sequences of parentheses with n pairs and the a set of legal lattice paths [40]. On the basis of this bijection, we can present a set of well-formed sequences in parentheses with n pairs with the ballot record. For that purpose, let the each left parenthesis in the sequence of well-formed parentheses we replace with an A and the right with a B. In this way, we get the unique presentation of the corresponding set of well-formed sequence of parentheses with n pairs by the ballot record. This procedure is presented in Example 5.2.

Example 5.2 *Let we consider the five possible parenthesized expressions for pentagon:*

$$()()(); \ (()()); \ (())(); \ ()(()); \ ((())) \ .$$

From correspondence $(\leftrightarrow A \text{ and } B \leftrightarrow)$ we have

$$()()() \leftrightarrow \boldsymbol{ABABAB};$$
$$(()()) \leftrightarrow \boldsymbol{AABABB};$$
$$(())() \leftrightarrow \boldsymbol{AABBAB};$$
$$()(()) \leftrightarrow \boldsymbol{ABAABB};$$
$$((())) \leftrightarrow \boldsymbol{AAABBB}.$$

The problem of well - formed sequences of parenthesis corresponds to the problem number of different ways of multiplying. To prove the mutual bijection, we establish the bounce in the following way: We remove all except right parentheses and signs of multiplying. The multiplying signs we replace with the left parenthesis. Obtained sequence of parenthesis is a sequence of well - formed parenthesis.

Example 5.3 *Well - formed parenthesis:* (((*ab*)*c*)*d*); (*a*(*b*(*cd*))); ((*a*(*bc*))*d*); ((*ab*)(*cd*)); (*a*((*bc*)*d*))) *are five possible multiplication expressions (they also correspond to planted trivalent tree of pentagon).*

$$(((ab)c)d) \leftrightarrow ()()() ;$$
$$(a(b(cd))) \leftrightarrow ((())) ;$$
$$((a(bc))d) \leftrightarrow (())() ;$$
$$((ab)(cd)) \leftrightarrow ()(()) ;$$
$$(a((bc)d)) \leftrightarrow (()()).$$

5.3.4 Algorithms for polygon triangulations based on trivalent binary tree and ballot notation

Let the planted trivalent binary tree (*PTBT*) correspond to the convex polygon with $v = n + 2$ vertices. In Figure 5.2 shows the general form of moving through the polygon where the movement is based on the appearance of signs *A* and *B* in the ballot record. The path between vertices of the tree has defines the appearance of the sign *A*, while from the vertex of the tree to the corresponding leaves defines the appearance of the sign *B*. In this way, it is always possible to determine the path through which moves (see Figure 5.2).

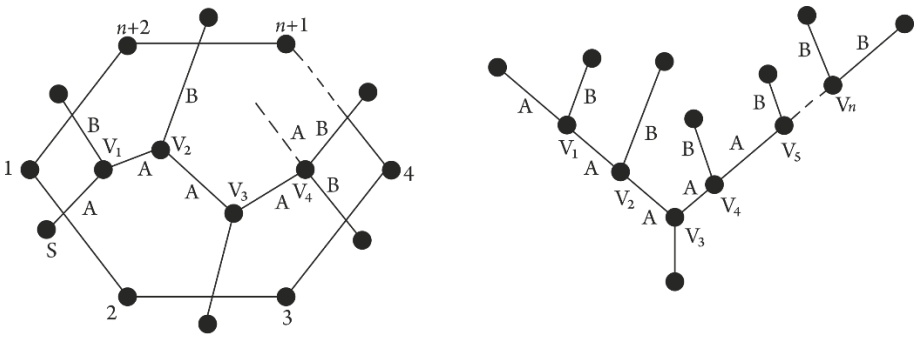


Figure 5.14 General schema of triangulated polygon and their corresponding planted trivalent binary tree.

The algorithms presented later use a planted trivalent binary tree for the triangulation of polygon. Corresponding planted trivalent binary tree with $n-1$ leaves used in Algorithm 5.1 is constructed by choosing one particular edge called base (in our case $(2,3)$). For this algorithm, we choose a point inside every triangle in the triangulated polygon, and one point outside each side of the triangle except the side which is in the interior of the polygon. The resulting graph is a planted trivalent binary tree with $n-1$ leaves.

ALGORITHM 5. 3 From Ballot notation to Triangulation based on PTBT

Require: A ballot record for *PTBT*, denoted by $b = (b_1, \dots., b_{2n})$.

1: Initialization: Create a polygon with $v = n + 2$ vertices. Make the corresponding planted trivalent binary tree. The vertices of the tree (except the root) are marked by V_j, $j = 1, \dots, n$ in the counterclockwise order. The leaves of tree corresponding to the outside points are labelled by *B* except the starting point. Every vertex has two indicators *visited* and *finished*, all initially on *false*. Set *k* = 0. Make an empty list $aux =$.

Set an array $visited_j, j = 1, \dots, n,$ on *false.*

Begin the movement from $(1,2)$ side in the counterclockwise order.

Set $k = 1, j = 1$, the starting point of movement is $P_k = S$ and always begin with $b_1 = A$.

2: for $i = 2$ to $2n$

2.1: If the current character in the ballot record is $b_i = A$, then find the smallest $j_2 \geq j$ where $visited_{j_2} = false$. Set $j = j_2,\ k = k + 1, P_k = S$

2.2: If the current character in the ballot record is $b_i = B$, then find the smallest $j_2 \geq j$ where $visited_{j_2} = false$ and $visited_{j_2} = true$.

2.2.1 If there are two successive characters *B* in ballot record then return to the parent of *B* (V_j) and set $j = j - 1$.

3: Draws the internal diagonals based on Algorithm 5.2

Output: Generated triangulation corresponding to the ballot record from the input.

ALGORITHM 5. 4 Finding Intersections in Polygon

Require: Path P_k, $k = 1,\dots, n$ inside a polygon.

1: for $i = 1$ to m; where m is the number of all polygon diagonals, $m = v(v - 3) / 2$

1.1: Choosing the diagonal from the set of all internal diagonals $(\delta_{1,3}; \dots; \delta_{1,v}), (\delta_{2,4}; \dots; \delta_{2,v})$, etc.)

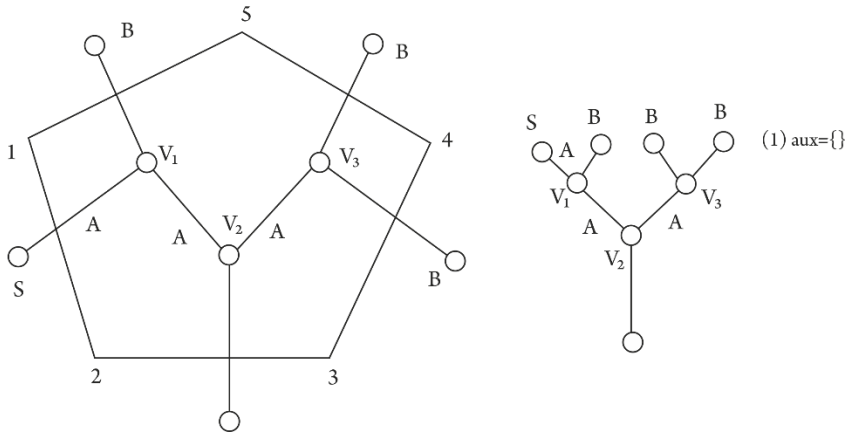
1.1.1: If detect intersection in polygon with only one part of path AND if it not intersects with the previous contained diagonal - THEN draw the diagonal

1.1.2: ELSE - Choose next diagonal (go to step 1.1)

Output: $v - 3$ internal diagonals (without intersection).

Example 5.1 Let us illustrate how Algorithm 5.3 produces a pentagon triangulation corresponding to the ballot record ***ABAABB*** (which corresponds well-formed sequence of parentheses with 3 pairs ()(()) / planted trivalent binary tree given in the first step). The process can be presented by steps as follows.

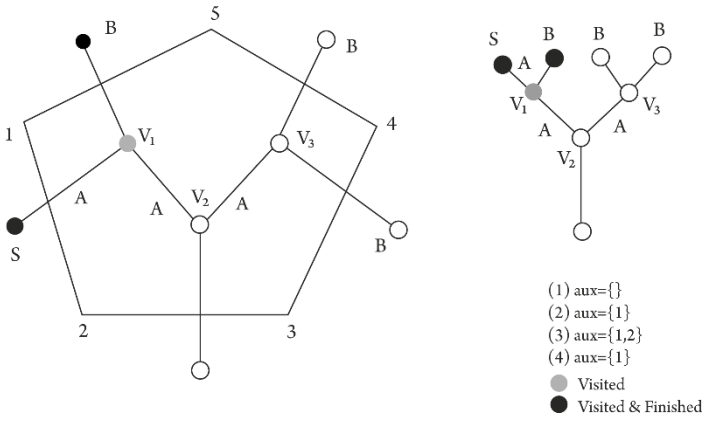
(1) The first step is to create the corresponding planted trivalent binary tree for the pentagon. This tree has depending on n, and can be easily generated, with corresponding vertex labelling and initializing attributes *visited* and *finished* on *false* and integer k on 0. Also, $aux = \{\}$.



(2) Since the first character in ballot record is *A*, move from the starting point *S* to the next *V* labelled descendent (this is V_1); $k = 1$, $aux = aux \cup \{\}$, V_1*:visited* = *true*.

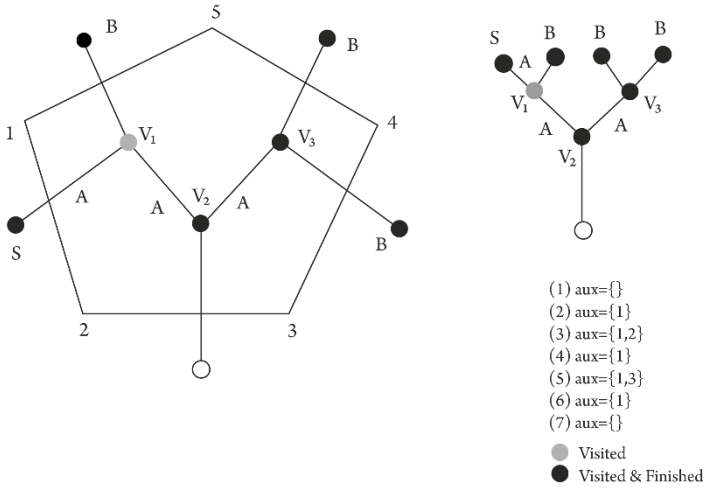
(3) The next character in the record is *B*, set *B:visited* and *B:finished* on *true*, and return to the parent of leaf *B* (V_2).

(4) The next character is *A*, go to the V_2, $k = k + 1$, $aux = aux \cup \{\}$, V_2*:visited = true*



(5) The next character is *A*, go to the V_3, $k = k + 1$, $aux = aux \cup \{\}$, V_3*:visited = true*

(6) The next character is *B*, set *B:visited* and *B:finished* on *true*, and return to the parent of leaf *B* (V_3). (7) The next character is *B*, set *B:visited* and *B:finished* on *true*, set V_3*:visited = finished*, return to vertex V_2 set V_2*:finished = true*, return to vertex V_1 and set V_1*:finished = true*. Thus, there are no more characters in the ballot record, the movements are finished.



(8) Upon the path *P* traversed inside the polygon the corresponding triangulation can be constructed. Draw all possible internal diagonals that cut a path inside the polygon marked with *A* (based on Algorithm 5.4). In this case, the possible non-crossing

internal diagonals that intersect the given path marked with *A*'s (i.e. with V_1; V_2; V_3) are $\delta_{2,5}$ and $\delta_{3,5}$.

Now we present a reverse algorithm for this process. Namely, how to get an appropriate ballot record from the convex polygon triangulation (reverse of the Algorithm 5.3). Algorithm 5.5 illustrate how from convex polygon triangulation based on planted trivalent binary tree, is obtained the corresponding ballot record.

ALGORITHM 5. 5 Triangulation to Ballot

Require: A convex polygon triangulation based on *PTBT*.
1: Make the corresponding planted trivalent binary tree for given polygon triangulation, where the point out of polygon next to the side $(2;\ 3)$ is omitted.
2: Mark the path between vertices in the tree with *A* and tree leaves by *B*. The exception is point next to $(1;\ 2)$ which is a starting point and need to be marked with *A*.
3: Use *inorder* traversal taking vertex marks along the traversal. In this way we get the corresponding ballot record.
Output: The corresponding ballot record.

Example 5.2. *Let us illustrate the application of Algorithm* 5.3 *in the process of movement through the pentagon triangulation defined by internal diagonals* $\delta_{2,5}$ *and* $\delta_{3,5}$.

Create the corresponding planted trivalent binary tree for the given triangulation and use inorder traversal in taking vertex marks. Movement between two vertices marks with *A*, between vertex and leaf with *B*.

(1) The triangulation tour starts from position *S* (see the figure below), more precisely from the edge $(1;\ 2)$ and the first character in the Ballot notation is marked with *A*. Set $k = 1$, $aux = aux \cup \{k\}$ and V_1*:visited = true*.
(2) Go to the leaf 1 of *PTBT* (outer edge $(1;\ 5)$) and mark character in the Ballot notation with *B*.
(3) Go to the vertex V_2 of *PTBT*, set $k = k + 1$, $aux = aux \cup \{k\}$ and V_2*:visited = true*. Mark the third character in Ballot notation with *A*.
(4) Go to the vertex V_3 of *PTBT*, set $k = k + 1$, $aux = aux \cup \{k\}$ and V_3*:visited = true*. Mark the third character in Ballot notation with *A*.

(5) Go to the leaf 1 of *PTBT* (outer edge $(4;\ 5)$) and mark character in the Ballot notation with *B*.

(6) Go to the leaf 1 of *PTBT* (outer edge $(3;\ 4)$) and mark character in the Ballot notation with *B*. Set V_3*:visited = finished*, return to vertex V_2 set V_2*:finished = true*, return to the vertex V_1 set V_1*:finished = true*. We will get the corresponding ballot record ***ABAABB***.

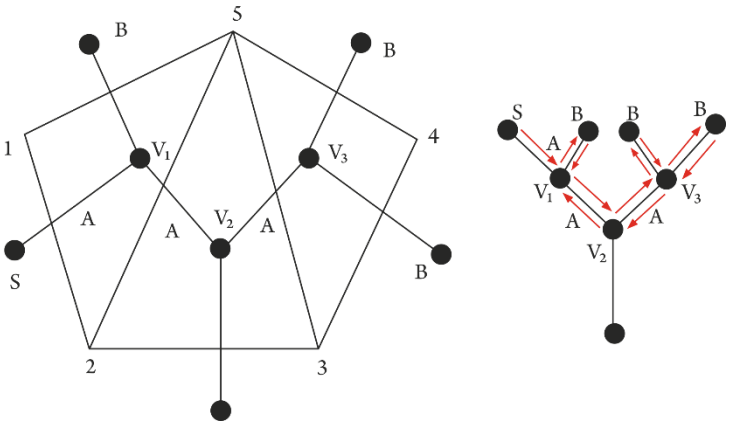


Figure 5. 15 Triangulation based on *PTBT* and corresponding ballot notation *ABAABB*.

5.3.5 Comparative analysis of experimental results

In order to get an evaluation of the presented methods (Ballot Lattice [11] and Ballot Trivalent), we offer a comparative analysis with the one existing method for triangulation of a convex polygon given by Hurtado and Noy [3] (hereinafter Hurtado method). The reason why we chose this method is that the generation of triangulation of a convex polygon is based on hierarchy or trees of triangulations. The number of all combinations in each level is determined by Catalan number.

Generating triangulation in our methods (Ballot Lattice [11] and Ballot Trivalent) is also based on a Catalan number for n-gon, respectively on n-th level like as in the existing Hurtado method. The only difference is in generating the new triangulation, by Hurtado method the process goes by splitting the outer edge of parents in generating new descendants and thus it can generate their hierarchy as shown in the previous figure (about the topics see [3]). However in our method, generating new descendants is based on the basis of movement through the polygon based on a given algorithm. All methods are implemented in Java language (NetBeans environment).

Details of implementation and testing of Hurtado-Noy algorithm are presented in the paper [9]. Tables in this section presents the results of testing for both methods for $n-$gon with vertices $v \in \{7, \dots, 16\}$.Testing results are analyzed from two aspects (see tables below):

- *Speed of execution* (*Table* 1): Time which refers to the phase of generating triangulation without storage and total time for all three phases: input, generating triangulation, output - their storage.
- *Working memory* (*Table* 2): Buffer occupancy (working memory) during generating all triangulations for the given n.

Symbols in the Table 1: BL - Ballot Lattice method (+ with or - without graphic generation of triangulations), BT - Ballot planted trivalent binary tree method (+ with or - without graphic generation of triangulations), HT - Hurtado method (+ with or - without graphic generation of triangulations).

Table 5.2 Experimental results: Ballot Lattice - Ballot Trivalent - Hurtado Tree (Execution CPU times)

n-gon	No. of triang.	Execution CPU times (in sec.)					
		Ballot Trivalent +	Ballot Trivalent -	Ballot Lattice +	Ballot Lattice -	Hurtado Trees +	Hurtado Trees -
5	5	0.23	0.21	0.24	0.22	0.25	0.21
6	14	0.32	0.28	0.33	0.31	0.34	0.29
7	42	0.41	0.34	0.41	0.38	0.43	0.35
8	132	0.49	0.40	0.47	0.44	0.49	0.40
9	429	0.62	0.54	0.63	0.60	0.67	0.55
10	1,430	1.15	0.87	1.03	0.98	1.18	0.89
11	4,862	3.79	2.14	3.51	3.09	3.81	2.19
12	16,796	12.41	5.69	11.29	10.11	12.46	5.81
13	58,786	49.91	15.07	43.55	38.81	50.51	15.24
14	208,012	107.02	42.22	108.07	97.13	119.05	46.34
15	742,900	272.66	124.18	274.19	244.02	318.63	124.18
16	2,674,440	673.23	570.01	677.17	572.87	/	/

Table 1 presents the results of testing for three methods. Number of triangulation of polygons with the number of vertices $v \in \{5,6, \dots, 15,16\}$ is $t \in \{5,14, \dots, 742\,900, 2\,674\,440\}$. We consider the differences in times for generating triangulations for all methods. Implementation and experimental results of this kind of storage methods are given in [11] and a part of them is shown in Table 1.

The graph below shows the CPU in the process of generating triangulation, where it can be noted that the greatest load is in the Hurtado method, then the Ballot Lattice method, and the least load in the Ballot Trivalent method (the graph is generated based on values from the Table 1).

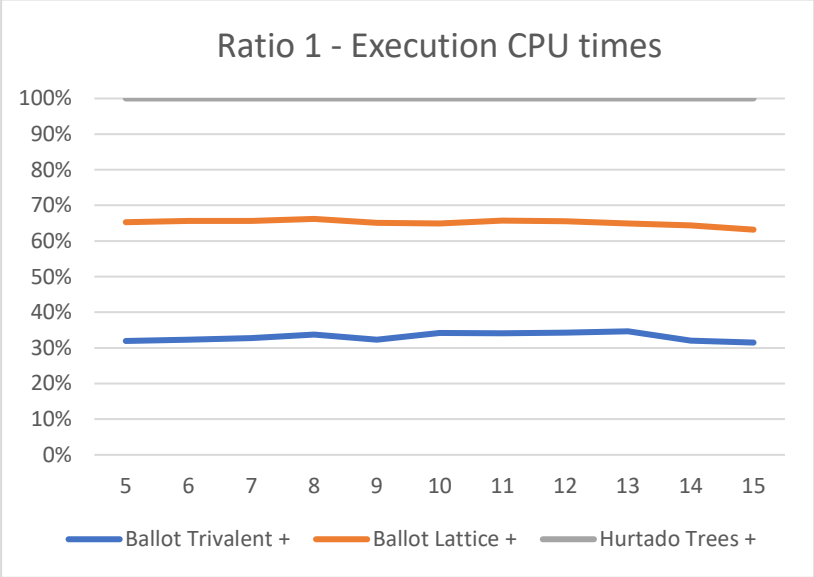


Figure 5.16 Load of CPU

Table 2 presents a comparison in storage requirements for the tested methods (Load memory in KB). We give differences in the size of required memory as well as the percentage share the time needed for storage in the program execution.

Table 5. 3 Experimental results: Ballot Lattice - Ballot Trivalent - Hurtado Tree (Load memory)

n-gon	No. of triang.	Load memory (in Kb)		
		Ballot Trivalent	Ballot Lattice	Hurtado Trees
5	5	0.02	0.02	0.12
6	14	0.08	0.08	0.46
7	42	0.29	0.29	1.76
8	132	1.05	1.05	6.46
9	429	3.86	3.86	24.49
10	1,430	14.31	14.30	93.13
11	4,862	53.42	53.40	355.67
12	16,796	196.12	196.00	1,363.43
13	58,786	502.71	502.00	4,121.13
14	208,012	1,442.00	1,441.00	11,523.62
15	742,900	4,299.90	4,295.00	29,874.29
16	2,674,440	12773.89	12,752.00	/

The graph below shows the workload ratio of the triangulation storage process, where it can be noted that the greatest load is in the Hurtado method, then the Ballot Lattice method, and the least load on the Ballot Trivalent method (the graph is generated based on the values from the Table 2).

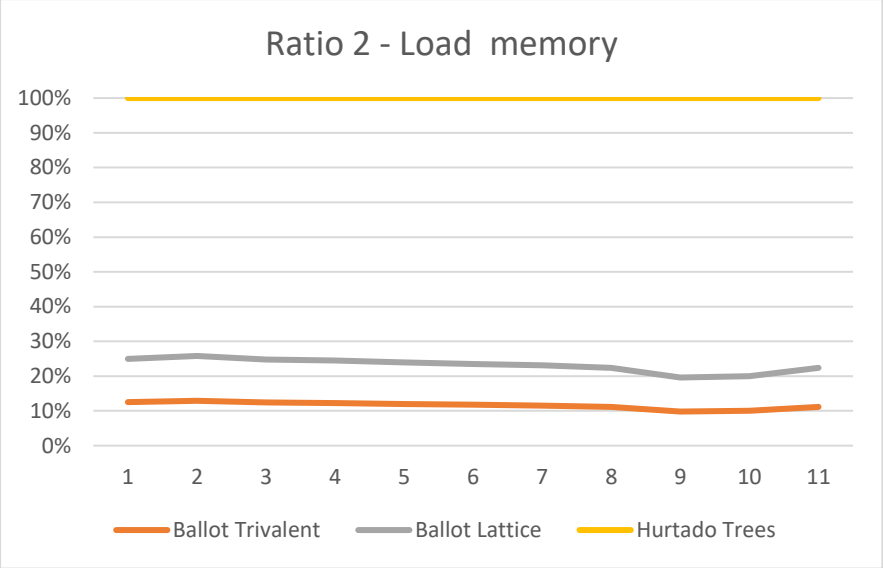


Figure 5.17 Load of memory

Based on the testing results, it can be concluded that Ballot Lattice and Ballot Trivalent method gives better results in the case where both generation and storage are included at the same time. Significant savings are achieved by applying Ballot output. The testing is performed in NetBeans testing module "Profile Main Project / CPU Analyze Performance" in configuration: *CPU - Intel(R) Core(TM)2Duo T7700, 2.40 GHz, L2 Cache 4 MB (On-Die, ATC, Full-Speed), RAM 2 Gb, Graphic card - NVIDIA GeForce 8600M GS*.

5.3.6 Complexity of the algorithms

Now we'll analyze ratio of the data (notation for triangulations) amount for Hurtado algorithm and Ballot Trivalent algorithm. Hurtado method uses all internal and external diagonals, while the Ballot Trivalent method uses only internal diagonals. Hurtado method requires recording of coordinates of n edges, plus two coordinates in storage $(n - 3)$ of internal diagonals: $2n + 2(n - 3)$. From the aspect of the storage complexity Hurtado Algorithm method requires:

$$HT_s = 2(2n - 3).$$

The Ballot Trivalent method requires a Ballot notation that is $2(n-2)$ length. Because the fist character (bit 1 or character A) and the last character (bit 0 or character B) in Ballot Notation are always known, the number of storage data is reduced by two. From the aspect of necessary data storage amount, Ballot Trivalent method requires:

$$BT_s = 2(n-2) - 2 = 2(n-3).$$

Let us estimate the processing complexity of both method. As we generate C_{n-2} triangulations of an n-gon by completing $2(n-2)$ permutation the number of movements in Lattice Path or in the Trivalent binary tree. Total number of operations for the Ballot Trivalent method for all triangulations of n-gon is

$$BT_p = 2C_{n-2}(n-2)$$

On the other hand, in the case of Hurtado algorithm we have the following. For every triangulation at level $n-1$ we need to perform $2n-5$ checks to find the diagonals incident to the vertex $n-1$. Total number of these checks is $(2n-5)C_{n-3}$.

Further we must go through diagonals and copy some without transforming, while some of them should be transformed and the two new diagonals should be inserted, for every incident diagonal which has been found. In a such a way we make $2n-3$ pairs describing one new triangulation.

The total number of incident diagonals is equal to C_{n-2}. Total number of operations for the Hurtado method for all triangulations of n-gon is

$$HT_p = (2n-5)C_{n-3} + (2n-3)C_{n-2}.$$

It is not difficult to verify the inequality, for all values of $n \geq 5$:

$$(2n-5)C_{n-3} + (2n-3)C_{n-2} > 2C_{n-2}(n-1)$$

Table 3 displays numerical data relating to the complexity of algorithms in two aspects: Storage and processing complexity. The number of operations required to generate all triangulations of n −gon is presented. In addition, the required data storage amount of a single triangulation of n-gon is shown.

Table 3. Ballot Trivalent (BT) vs Hurtado Trees (HT): Storage and processing complexity

n-gon	Storage complexity for one triangulation			Processing complexity for all triangulations		
	BT_storage	HT_storage	Savings	BT_processing	HT_processing	Speedup
5	4	14	10	30	45	15
6	6	18	12	112	161	49
7	8	22	14	420	588	168
8	10	26	16	1584	2178	594
9	12	30	18	6006	8151	2145
10	14	34	20	22,880	30,745	7865
11	16	38	22	87,516	116,688	29,172
12	18	42	24	335,920	445,094	109,174
13	20	46	26	1,293,292	1,704,794	411,502
14	22	50	28	4,992,288	6,552,378	1,560,090
15	24	54	30	19,315,400	25,258,600	5,943,200

Conclusion and further works

The proposed method in the paper gives an effective way of construction of the convex polygon triangulations with the ballot record and planted trivalent binary tree. The algorithms are related to the combination of ballot problem and convex polygon triangulations. The movements in constructed method through polygon are derived upon vertices and leaves of the planted trivalent binary tree. The significance of the presented algorithms is reflected in the effective construction of the convex polygon triangulations and their recording and storing in form of Ballot notation.

This research we hope that gives the first step toward further developing this important issue to the concave polygons and polyhedron. By decomposing of the concave polygon in a set of convex polygons algorithms can be extended to the case of a concave polygon. The decomposition will be the task of identifying of the reflex vertex/vertices and creating a new edge/edges and vertex/vertices (if it's necessary) until there is no reflex vertex in the concave polygon. It's important to remark here that the best solution to the problem is the decomposition with the least produced diagonals.

Similarly to the case of the concave polygon, by identifying the notch edges in the polyhedron the algorithms can be extended to solving problems in 3D. The solution of the problem would be looked in the identification of notch edges and plane cuts of the polyhedron until all polyhedrons in decomposition don't become convex.

5.4 MINIMUM-WEIGHT TRIANGULATION ALGORITHM BASED ON MATRIX CHAIN PRODUCT AND MEMOIZATION

Triangulation is a term that appears in various disciplines of mathematics and computer science. The triangulation is defined as decomposition of the particular region into smaller pieces that are also called triangulation elements (usually triangles) and are easy to handle. Application of triangulation is seen in different areas where there is a fixed set of points such as algebra, topology, volume calculations, and meshing. The triangulation problem generally is based on determining the set of all triangulations for a given geometric figure (construction of triangular meshes) or in finding the optimal triangulation for the given set of points in a plane or in a space. Triangular meshes represent the dominant discretization of surfaces in computer graphics, for which there is a significant number of research. Many triangulation applications in geometry rely on the orthogonal primal/dual network structure. The use of such dual structures in triangulation depends on the type of application, using, for example, in physical simulation [20, 7], parameterization [50, 36] and architecture modeling [22, 88]. Most of the obtained research results are based on planar triangular meshes.

In many applications of computational geometry in the problem of triangulation is needed to determine the boundary of the surface occurs by connecting one or more closed geometric figures. Such surfaces are encountered in filling the holes in an incomplete mesh, while in the problem of complex holes this problem is dealt with more identified boundaries such as the outer boundary and several inner islands [46]. In contour interpolation [88], 2D curves from neighboring flat parts must be connected with one or more surfaces. In the case of arbitrary planar points, in the practice, firstly is created an initial mesh for the surface treatment, usually a triangulation involving only the vertices of the polygon, and then refine this starting surface in order to achieve a better smoothness or mesh quality. The initial triangulation of one polygon can be calculated using a simple dynamic programming algorithm [25, 26]. The important property of this algorithm is the production an optimal triangulation that minimizes the sum of a certain "weight", whether it is the size that can be measured for each triangle individually or for each pair of adjacent triangles. Optimization of triangulation is important because the success of the construction of a final adult clearing

network often depends on the quality of the initial mesh. In this paper, we present an algorithm for finding the optimal triangulation of a convex polygon with memoization.

We suggest a solution that avoids the calculation of the same values several times and which does not endanger the optimality of obtained results. More precisely, given the number of calculations required in determining the gravity of the triangles in triangulation, our algorithm finds the optimal triangulation value by minimizing the optimal function that is defined as the sum of all the individual triangulation weights in the triangulation. Practically, in this paper, we present the optimal triangulation algorithm of a convex polygon based on the principles of dynamic programming significantly reduces computational cost in comparison with the algorithm constructed in the paper [70].

5.4.1 Related works in the field

Triangulation of a plane polygon is one of the basic problems of computational geometry. It is applied in the obtaining process of three-dimensional representations of objects from the given set of points. The main purpose in the triangulation process is the speed of finding triangulation of the polygons in all possible variants. The speed in this process is important since, with the increase in the number of polygon vertices, the number of this different triangulations increases drastically.

There are many efficient algorithms that generate triangulation of different geometric figures [8, 62], where most often it does not guarantee the optimality of the constructed triangulation. In the papers [59] and [23] are given the algorithms for calculation of the minimum weighted polygon triangulation into the plane, and are also known as algorithms that minimize the sum of the total lengths of the edges and have the time complexity O (n3). Algorithms developed on the principle of dynamic programming are aimed at constructing optimal triangulation of larger domains and they use the previously calculated optimal triangulation of smaller sub-domains. The paper [25] presents an algorithm that finds the optimal polygon triangulation by minimizing the triangle weight in the triangulation process with the same complexity. There are other approaches for polygon triangulation, but they do not have any guarantee for optimality and are often limited to specific class inputs. In cases where the polygon is enough planar, there are algorithms that use the polygon projection in the best-fitting plane, and they after triangulation of this projected polygon, finally, they get the required triangulation [24]. However, this algorithm can not be applied to the triangulation of the polygon curve; these polygons do not have planar projections in many

cases. In addition to the concept of triangulation, there are also methods that construct quadrangulations of the polygon by interpolating the sketches with flow lines [51]. For triangulation of the 3D polygon in work [38], extracts near-developable surfaces using convex hulls are presented.

Calculation of the optimal triangulation of a convex polygon can be constructed on the basis of the block method of generating triangulations that have been exhibited in [47]. The presented method in this paper reflects on the fact that is used the recorded values in finding the optimal triangulation and directly record the values of vertices and their weight. Another approach of the convex polygon triangulation is presented in the paper [86], where authors developed the algorithm for convex polygon triangulation which uses the already made triangulations from one vertex less in the polygon.

5.4.2 The method for optimal triangulation

Dynamic programming is a discipline that analyzes problem solution methods based on the principles of optimization. The optimal solution of the problem is independent of the initial position and is obtained as a succession of optimal solutions of subproblems. The variables that correspond to the system in dynamic programming are determined consecutively. The method developed in this paper is based on principle storing of best solution from different solutions which give the same result.

In the solving of the optimization problem with the dynamic programming method, optimal substructure and overlapping property of subproblems are two key elements that are examined. Our method for finding and storing of optimal triangulations of a convex polygon is developed taking into account this overlapping property of the subproblems. The first step in solving an optimization problem by dynamic programming is to characterize the structure of an optimal solution [12].

The idea for the construction of the method for obtaining optimal triangulation of convex polygon with memoization starts from possibility which gives the Java as a programming language in the implementation of weights storing results that are determined with the advanced application of packages in the work with table cells. The *DefaultTableCellRendererJava* package takes the main role in storing of triangulations weight. This package enables storage of multiple values within a single table cell (which provide efficient cell division into multiple columns and rows) [70].

The calculations in the constructed method are based on principles of dynamic programming techniques. The memoization technique, except the possibility of fast calculation of optimal triangulations, give an opportunity to save memory space and time during calculation polygonal triangulations.

Memoization is a set of dynamic programming and recursion. The technique is in the top-down direction where all solutions are stored in memory. This technique don't solve the same problem multiple times, already use the calculated solutions in finding the solution of the general problem. Dynamic matrix, i.e. the part where are stored the solved problems, initially are denoted with some impossible solution number (for example, with 0, because the optimal triangulation of polygon is positive).

Let $P = \langle v_0,\ v_1, \dots, v_{n-1} \rangle$ represent a convex polygon by listing its vertices in counterclockwise order. Given two non-adjacent vertices v_i and v_j , δ_{ij} is a diagonal of the polygon. A triangulation of a polygon is a set T of diagonals of the polygon that divide the polygon into triangles with nonintersecting diagonals. In the optimal polygon triangulation problem, the weight function w is defined on triangles formed by edges and diagonals of P. The problem is to find a triangulation with minimal sum of weight of triangles in the decomposition of a convex polygon. The weight w of triangle $v_i v_j v_k$ is calculated as follow:

$$w(i,j,k) = |v_i v_j| + |v_j v_k| + |v_k v_i|$$

where $|v_i v_j|$ is the length of the edge with vertices v_i and v_j of triangle.

We develop the method using the correspondence between polygon triangulation, Catalan numbers, and balanced parenthesis. The edges of a convex polygon with n vertices we denote with matrices $A_i,\ i = 1, 2, \dots, n-1$. The balanced parenthesized product of n matrices correspond to the polygon with $(n+1)$ vertices. Each matrix A_i from this product correspond to the edge $v_i v_{i+1}$ of polygon, and diagonal $\delta_{ij}, (i < j)$ corresponds to a matrix $A_{i+1,\dots,j}$ which is obtained during matrix product calculation. From fact that the matrix chain is special case of optimal triangulation problem for the matrix product $A_1 A_2 \cdots A_n$ we consider the triangulation of convex polygon with $(n+1)$ vertices. For the matrix A_i with dimension $p_{i-1} \times p_i$ the weight function of triangulation is $w(i,j,k) = p_i p_j p_k$.

By replacing the matrix dimensions $p_0, p_1, \dots, p_n$ in the matrix chain with the vertices $v_0, v_1, \dots, v_n$ of a convex polygon, we have a weight function of triangulation

$$w = m[i,k] + m[k+1,j] + w(i,j,k) \tag{5.18}$$

The weight of optimal triangulation is the sum of the weights of sub-polygons $\langle v_0, v_1, \dots, v_k \rangle$ and $\langle v_k, v_{k+1}, \dots, v_n \rangle$ and the triangle $\Delta v_0 v_k v_n$ in the triangulation. Similarly to the computation of the minimum cost $m[i,j]$ in matrix chain, for $1 \leq i < j \leq n$ the $w[i,j]$ is defined to be a weight of optimal triangulation of subpolygon $\langle v_{i-1}, v_i, \dots, v_j \rangle$, and the optimal triangulation of polygon is $w[1,n]$. Then for optimal triangulation has a following recursive formulation:

$$w[i,j] = \begin{cases} 0 & if\ i = j \\ \min\limits_{i \leq k < j} w[i,k] + w[k+1,j] + w(i,j,k) & if\ \ i < j \end{cases} \tag{5.19}$$

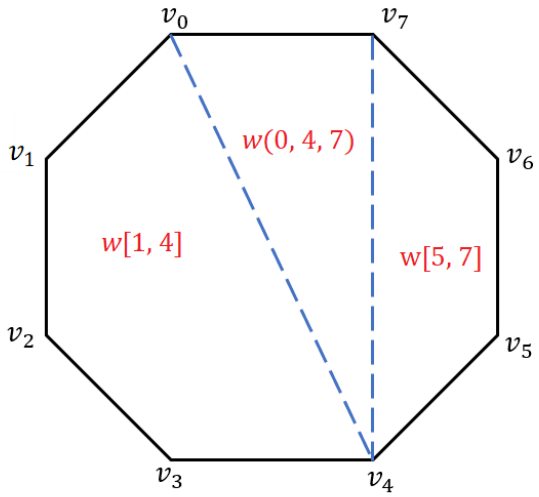


Figure 5. 18 Optimal triangulation of convex octagon with associated weights

Storing of the weights in optimal triangulation

The method for storing triangulation weights is based on a table with $(n \times n)$ dimension $(i = j = n)$, where *i* represents a row, *j* columns, and *n* is the number of vertices of the polygon. The table $(n \times n)$ is used for calculation of weights of triangles in triangulation. The table is diagonally divided into two parts. The diagonal divides $(i = j)$ the table into the left and the right part. In the left side of the table are filled the edges of triangles which length are represent by matrices $A_1, A_2, \dots, A_{n-1}$ in the triangulation. The elements in the diagonal of the table represent the vertices of the polygon and are filled with zero because they have no weights. From correspondence between *n* balanced parenthesized product and

$(n+1)$-vertex polygon the elements an *n*-th column of the table were no needed in matrix chain calculation and are filled *X*.

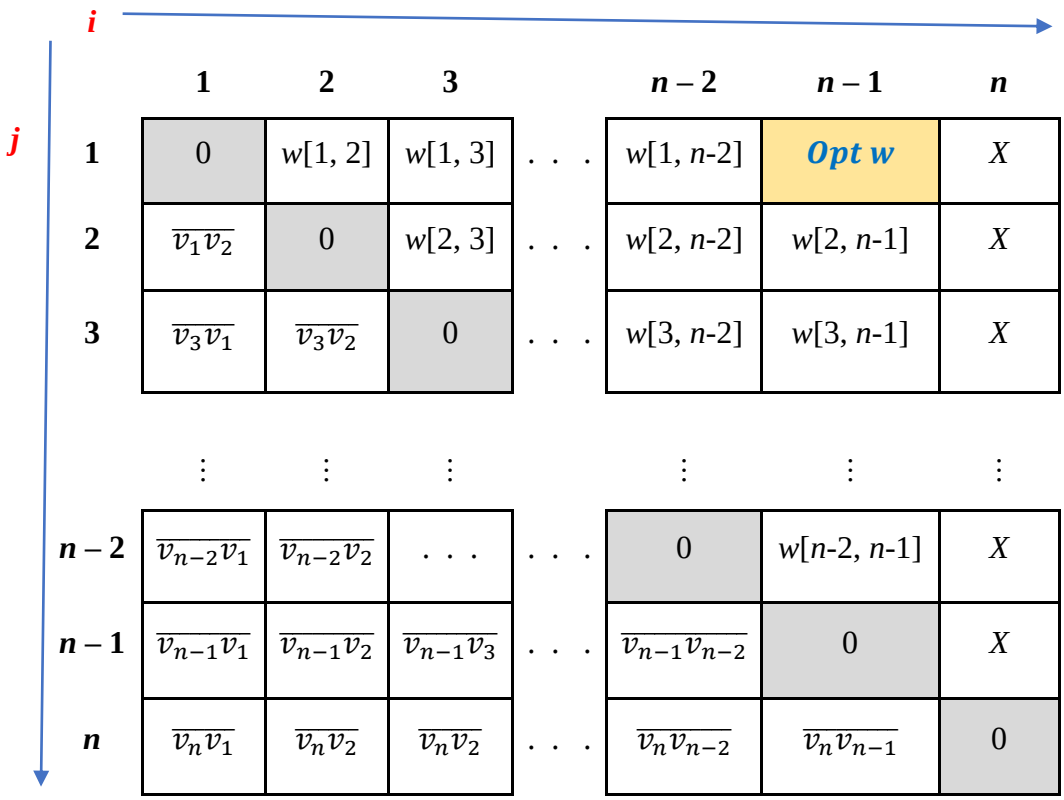


Figure 5. 19 The general scheme of labeling of fields in the table

For $k = 1$ we have the polygon edge a which is determined by adjacent vertices v_1 and v_2, see Figure 5. 19. Similarly, for $k = 2, 3, \dots, n-1$ we get the others edges of a polygon which are determined from the adjacent vertices v_i and v_j for $i = 2, 3, \dots, n-1$ and $j = i + 1$. By increasing the number of rows on the table we obtain the edges of the triangles that are elements of triangulation, i.e the edges which are obtained from non-adjacent vertices of the polygon. For example, for $k = 1,2$ we have a weight of edge between vertices v_1 and v_3, $k = 2,3$ the weight of the edge between vertices v_2 and v_4 and so on. Similarly, in the end for $k = 1,2, \dots, n-1$ we obtain the weight of the edge between vertices v_1 and v_n which represents the total weight of the triangulation. Below, we present a 5×5 table for pentagon triangulation according to the filling general scheme given in Figure 5.19.

ALGORITHM 5. 6 Storing of optimal triangulation with memoization

Require: n, Table $(n \times n)$

1: Create a new Table $(n \times n)$

for $(i = 1; i \leq n; i++)$

Label the fields $(i,j) = 0$, where is $i = j$

Label the field $(i,j) = adj$, where is $j - i = 1$

2: Filling the other fields

For $(i = 1; i \leq n; i++)$

For $(j = 2; j \leq n; j++)$

$j = i + 1$

3. Calculate the Matrix Chain Product for elements of Table

if ($w[i,j] < \infty$) then return $w[i,j]$

if ($i == j$) then $w[i,j] = 0$

else $k = i$;

$Opt\, w = w[i,k] + w[k+1,j] + w(i,j,k)$

For ($k = i+1; k < j; k++$)

if ($Opt\, w < w[i,j]$) then $Optw = w[i,j]$

return $w[i,j]$

We present Algorithm 5.6 for finding and storing of optimal triangulation. In the beginning, the algorithm expects the number of rows and columns of table n which correspond to the $n-$vertex polygon. All vertices of the polygon match to the diagonal elements of the table. The $n-$th column of the table consist the elements which haven't a role in the calculation of the weights in the triangulation process and due to this, they are labeled with X and has the meaning of a cell that didn't have a weight. The algorithm has 3 steps:

1. We form the Table $i \times j$ and we fill the cells along the diagonal with 0 where $(i = j)$. Then we fill the edges of the polygon on the position $(i,j), i = 2,3,\dots,n$ and $j = 1,2,\dots,n-1$. They are adjacent vertices of a polygon which can be used to form (i,j,k) triangle in triangulation process. Their weights are filled in the cells of the Table where $(i,j), i = 1,2,\dots,n-1$, $j = i+1$ and it's mean that there is no k-value for them between (i,j).

2. We calculate all triangulation weights $w[i,j]$ based on memoization bearing in mind calculation do not calculate multiple times the same weight for rows and columns of the table where $(j-i)>1$ and $i=2,3,\ldots,n-1$. According to the formula (5.19), we're going to the next step. To assigned k values on position (i,j) we add their correspondent weights on position (j,i).
3. In the cell $1\times(n-1)$ of the Table we obtain weight of optimal triangulation.

Example 5.3 This example shows the recursion tree of memorized matrix chain for pentagon.

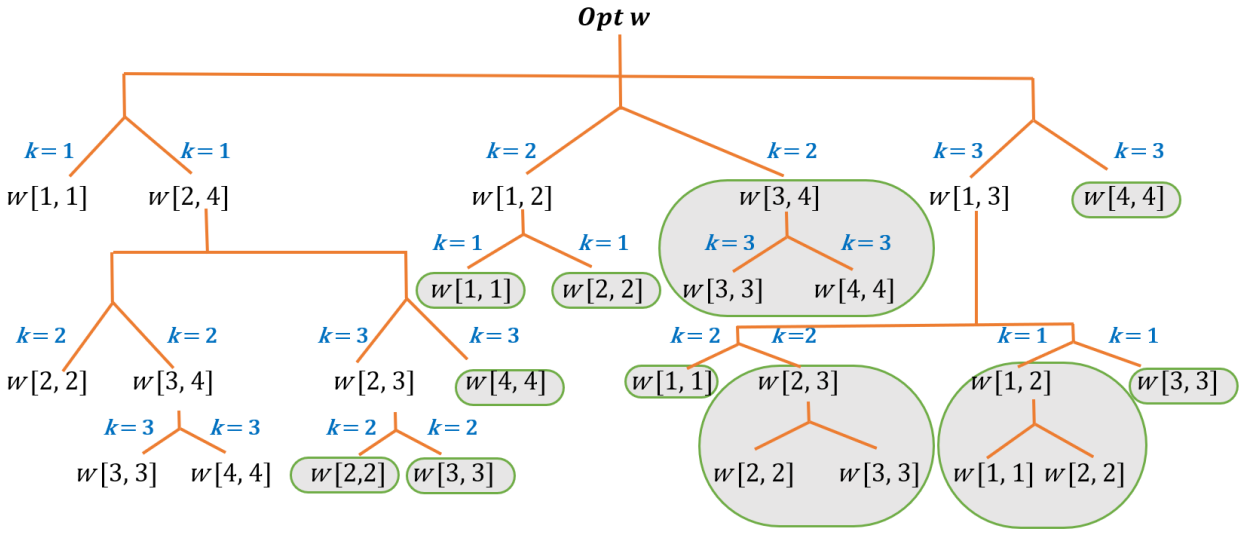


Figure 5.20 Recursion tree of memorized matrix chain of Pentagon

Based on correspondence $P(n+1)=C_n$ for pentagon $(n=5)$ we obtain five different triangulations see Figure 5.21. The edges of pentagon are labeled with matrices $A_i, i=1,2,\ldots,5$. The matrices with even number index have dimension 5×1, matrices with odd number index have dimension 1×5.

	1	2	3	4	5
1	0	25	10	$Opt\ 5=35$	X
2	$\overline{v_1v_2}$	0	5	10	X
3	$\overline{v_3v_1}$	$\overline{v_3v_2}$	0	25	X
4	$\overline{v_4v_1}$	$\overline{v_4v_2}$	$\overline{v_4v_3}$	0	X
5	$\overline{v_5v_1}$	$\overline{v_5v_2}$	$\overline{v_5v_3}$	$\overline{v_5v_4}$	0

There is (5 × 5) Table for all five triangulations contains the optimal triangulation which corresponds to the minimum weight in the triangulation process of the pentagon. The table for pentagon according to the labeling of the general scheme from Figure 5.19 is given in the following example.

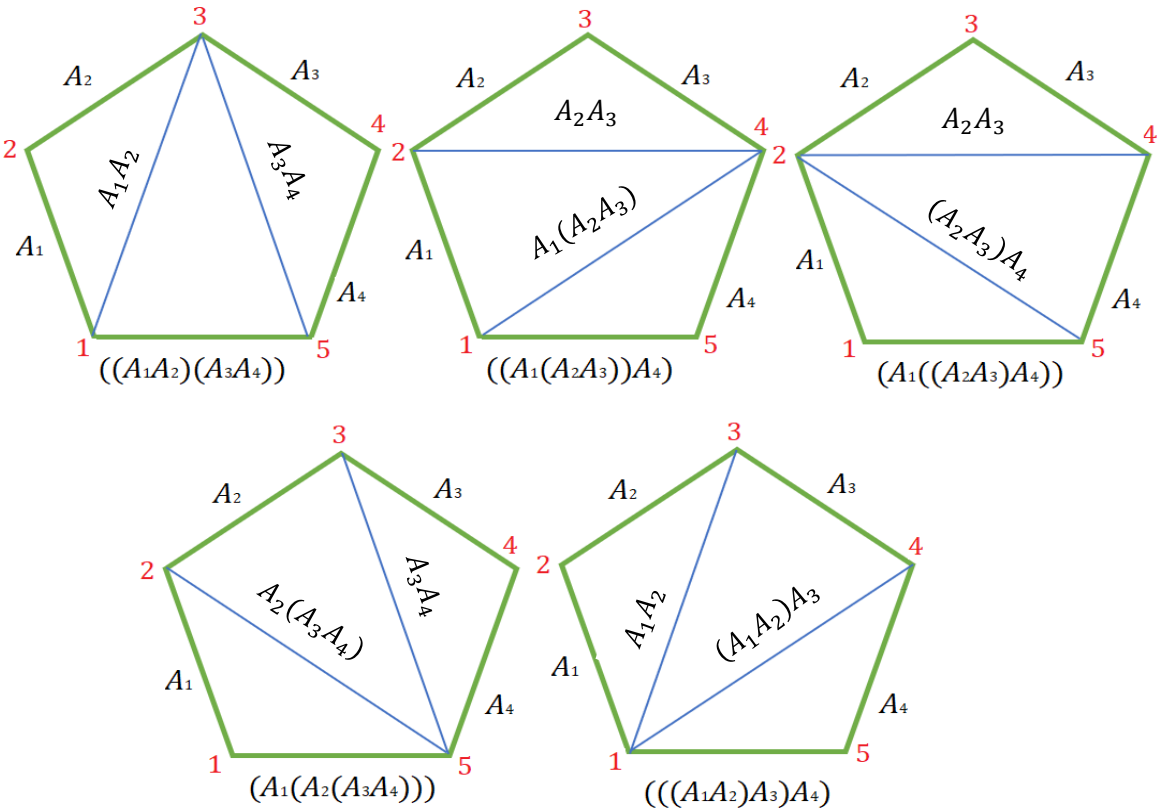


Figure 5.21 Triangulation of pentagon

Example 5.4 Acquiring of the optimal triangulation for the irregular convex pentagon we describe with following steps.

Step 1: In this step is formed the 5 × 5 Table, and the cells in diagonal are filled with zeroes and the cells below of the diagonal on the left side are represented adjacent vertices, that is the vertices of the polygon. Thereafter, is set up the values for all (i, j, k) triangles. The values in the parenthesis {} represent the number of vertices that are part of triangulation, for example, {2} is between vertex 1 and 3 and is in the cell (3, 1) of Table. After these steps, the Table has the following form:

0	25	10	**35**	X
(2, 1)	0	5	10	X
{2}	(3, 2)	0	25	X
{2, 3}	{3}	(4, 3)	0	X
{2, 3, 4}	{3, 4}	{4}	(5, 4)	0

Step 2: We find the weights of the triangles which are part of triangulation procedure for C_3 rows that are given on the 5×5 Table. The table for generated triangulations is created on the bases of the Algorithm 5.6. The sums of all triangle weights are saved in the button-up technique in memoization. Below is given the Table of weights which correspond to the diagonals in the possible triangulation of pentagon see Figure 5.21.

Table 1. Extended table for triangulation of pentagon

Triangulations	Diagonals in triangulation	Optimal triangulation
1	δ_{13}, δ_{35}	**71**
2	δ_{14}, δ_{24}	**61**
3	δ_{24}, δ_{25}	**35**
4	δ_{25}, δ_{35}	**51**
5	δ_{14}, δ_{13}	**76**

Step 3: In this step, is calculated the $\boldsymbol{Opt\ w}$ according to the condition**:**

$$Opt\ w\ <\ w[i,j] \tag{5.20}$$

where is $1\ \leq i\ \leq 5$.

Based on (5.20) is obtained optimal triangulation $Optw\ =\ 35$.

After finding the optimal triangulation from generated 5×5 Table, follows drawing the same triangulation of pentagon based on non-crossing diagonals $\delta_{i,j}$ which corresponds to the upper portion of cells in the table, i.e. pair (i,j): $Optw\ =\ \{\delta_{1,4}, \delta_{2,4}\}$, see Figure 5. 22.

0	25	10	**35**	X
(2, 1)	0			
{2}	(3, 2)			
{2, 3}	{3}			
{2, 3, 4}	{3, 4}			

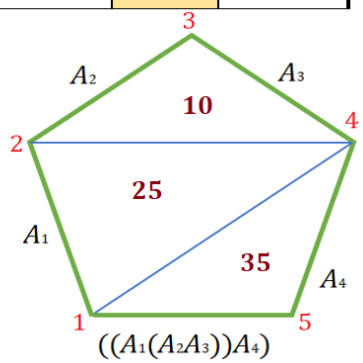


Figure 5. 22 The corresponding values in the table and optimal triangulation of pentagon

Comparative analysis of our method with other methods in this field

In order to get a better overview for the developed method in this paper, we first analyze the calculations in the traditional, Hurtado-Noy and square matrix method, which are needed for finding of the weight of each triangulation separately of the polygon.

In the traditional method the total number of the weights (is included and the case with repetition) in the n −vertex polygon is obtained as a product of the total number of triangulations and the number of diagonals of the polygon:

$$TM_w = C_{n-2} \times (n - 2) \tag{5.21}$$

For the determining of the triangulation weights in the pentagon by using the formula (5.21), there are 15 calculations in total.

In the paper [70] is presented the square matrix method for storing an finding an optimal triangulations of the polygon. In square matrix method in the begin are determined the values of the number k that are in the square matrix. According to the general scheme of this method in the filling of the matrix elements, the number k increases as is going further from the diagonal. In the first row of the matrix which is constructed by this method, there are no elements since they represent the so-called adjacent vertices, the second has one element, the third has two and so on. The last diagonal row contains $(n - 2)$ − vertices since the difference between the first and the last n −vertex is always two because the first and the last vertex are always subtracted from the total [70]. A total number of stored k-vertices n-vertex polygon is labeled with V_n and is given with relation in (5.22)

$$V_n = \sum_{i=1}^{n-2} i(n - i - 1) \tag{5.22}$$

By using of this relationship in this paper are obtained the following results: $V_5 = 10, V_6 = 20$ and $V_7 = 35$. The number of calculated weights in the triangulation process is determined with the following formula:

$$SM_w = \sum_{i=1}^{n-2} i(n - i - 1) \tag{5.23}$$

The number of saved calculations is $TM_w - SM_w$.

For the purposes of analysis we present the Hurtardo-Noy algorithm for generation of polygon triangulation based on predecessor results of triangulation, that is, triangulation P_n is obtained from P_{n-1}. This algorithm define the triangulations tree of in the procedure of the n −vertex polygon triangulation. All triangulations T_n are arranged at the level n of the tree and are used for calculation of the weights in the process of storage.

Every triangulation at the level n has a “father" in T_{n-1} and two or more “sons" in T_{n+1}. The sons of the same father are “brothers". There is an ordering among the children of a triangulation, and consequently among all triangulations [70].

We can restate Hurtado-Noy algorithm as the following.

ALGORITHM 5. 6 Hurtado-Noy algorithm

Require: Positive integer n and the set T_{n-1} of P_{n-1} triangulations. Each triangulation is described as a structure containing $2n - 5$ vertex pairs presenting P_{n-1} diagonals (here diagonals means both internal diagonals and outer face edges).

1: Check the structure containing $2n - 5$ vertex pairs corresponding to a particular τ_{n-1}, looking for pairs $(i_k;\ n - 1)$, $i_k \in \{1;\ 2; \ldots ;\ n - 2\}$ (upper bound for k is between 2 and $n - 2$), i.e. diagonals incident to vertex $n - 1$.

2: For every i_k, generate the son $S^{i_k}\tau_{n-1}$ by performing the transformation $\delta_{i_l,n-1} \longrightarrow \delta_{i_l,n}$, $i_l < i_k$, $0 \le l \le n - 3$ and inserting new pairs $\delta_{i_k,n}$ and $\delta_{n-1,n}$ into the structure.

3: Take next i_k, if any, and go to Step 2.

4: Continue the above procedure with next P_{n-1} triangulation (i.e. structure with $2n - 5$ vertex pairs) if any. Otherwise halt.

From Hurtado-Noy algorithm, it follows that for every triangulation at level $n-1$ is needed to perform $2n-5$ checks to find the diagonals which can be drawn from the vertex $n-1$. From this starting point for all possible triangulations, we see that $(2n-5)C_{n-3}$ is the number of checking that can be taken.

Further, we must go through diagonals and copy some without transforming, while some of them should be transformed and two new diagonals should be inserted, for every incident diagonal which has been found. In such a way we make $2n-3$ pairs describing one new triangulation.

The total number of incident diagonals is equal to C_{n-2} . All together, in the case of Hurtado-Noy algorithm we need

$$HN_w = (2n-5)C_{n-3} + (2n-3)C_{n-2}$$

number of weights calculations.

The developed method of storing weights in this paper is compared with the traditional method, Hurtado-Noy method and square matrix method given in the paper [70].

Our method as a procedure is developed on the bases of the generic dynamic programming concepts which in calculation of optimal triangulation uses the iteration technique through all subproblems. The iteration starts from the "smallest" and continues to the "biggest" subproblems. Using the previously-computed optimal values of the smaller problems for each subproblem is find the optimal value. Recording of the choices in the calculations of the weights allows to obtain the optimal triangulation weight.

At the case of pentagon triangulation we have the value 35 as a optimal solution of the optimal triangulation problem. Calculations are made according to the relation given in (5. 19). From the requirement $1 \le j \le i \le n$ in the relation $w[i,i+1] = w[i,i] + w[i+1,i+1] + p_{i-1}p_ip_{i+1}$, for $i = j$ we have $w[i,i] = 0$ and $w[i,i+1] = p_{i-1}p_ip_{i+1}$. This values are the weight of vertices and edges of the pentagon. The other calculations are made in the following order.

Step 1. $w[1,2] = p_0p_1p_2 = 5\cdot 1\cdot 5 = 25,$ $k=1$

$w[2,3] = p_1p_2p_3 = 1\cdot 5\cdot 1 = 5,$ $k=2$

$w[3,4] = p_0p_1p_2 = 5\cdot 1\cdot 5 = 25,$ $k=3$

Step 2. $w[1,3] = w[1,1] + w[2,3] + p_0p_1p_3 = 0 + 5 + 5\cdot 1\cdot 1 = 10,$ $k=1$

$w[1,3] = w[1,2] + w[3,3] + p_0p_2p_3 = 25 + 0 + 5\cdot 5\cdot 1 = 50,$ $k=2$

Step 3. $w[2,4] = w[2,2] + w[3,4] + p_1p_2p_4 = 0 + 25 + 1\cdot 5\cdot 5 = 50,$ $k=2$

$$w[2,4] = w[2,3] + w[4,4] + p_1p_3p_4 = 5 + 0 + 1 \cdot 1 \cdot 5 = 10, \qquad k = 3$$

Step 4.
$$w[1,4] = w[1,1] + w[2,4] + p_0p_1p_4 = 0 + 10 + 5 \cdot 1 \cdot 5 = 35, \qquad k = 1$$
$$w[1,4] = w[1,2] + w[3,4] + p_0p_2p_4 = 25 + 25 + 5 \cdot 5 \cdot 5 = 175, \qquad k = 2$$
$$w[1,4] = w[1,3] + w[4,4] + p_0p_3p_4 = 10 + 0 + 5 \cdot 1 \cdot 5 = 35, \qquad k = 3$$

In the triangulation of pentagon there are 10 diagonals with repetition from which only 6 are used in the calculation of the optimal triangulation with the memorization algorithm. The rows in red on the step 2, 3 and 4 are skipped with our method. Our algorithm obtains the optimal triangulation from the upper portion of $n \times n$ table. We notice that in the first row of the table we have $n-2$ calculation, in the second $n-3$ calculation, …, in the $n-1$ th we have $n-(n-1)$ calculation. From here in the optimal triangulation of n vertex polygon for the total number of saved calculations we have

$$MM = n-2+n-3+\ldots n-(n-1) = n-2+n-3+\ldots+2+1 = \frac{(n-2)(n-1)}{2}$$

different savings. Below is given the table of comparative analysis for four different methods.

Table 2. Comparative analysis

n-gon	*Traditional method*	*Hurtado-Noy method (HN)*	*Square Matrix method (SM)*	*Method with memoization (MM)*	*Difference (savings) between (HN) and (MM)*	*Difference (savings) between (SM) and (MM)*	*Ratio (speedup)*
7	210	45	35	15	30	20	2.33
8	792	161	56	21	140	35	2.67
9	3003	588	84	28	560	56	3.00
10	11440	2178	120	36	2142	84	3.33
11	43758	8151	165	45	8106	120	3.67
12	167960	30745	220	55	30690	165	4.00
13	646646	116688	286	66	116622	220	4.33
14	2496144	445094	364	78	445016	286	4.67

5.4.3 Analysis, results processing and implementation

In our method, the process of determining the optimal triangulation is based on access to weights storing on filling the upper part of the table with subproblem solutions. Using the memoization technique in the table, the solutions of the subproblem are filled only once and the same results are not calculated several times. Each cell in the table has its own entry which shows that the weight of a particular triangle in the triangulation process is calculated the first time and then stored in the upper part of the table. Every next time when is required the weight of a triangle which is part of triangulation, it is first checked whether this value is stored in some cell of the table. If this value is stored previously in the table, it is not calculated again but is used, if it is not stored this value is computed for the first time and is stored in the adjacent cell in the table.

In the upper part of the table constructed with our method, there are all values of the $w[i,\ j]$ which represents the solutions of optimal triangulation problem of the convex polygon. Each cell in the table is initially assigned with infinity, which indicates that the input has yet to be filled in that cell of the table.

In the calculation of *Optw*, if $w\ [i,j] < infinity$ then we return to the previous $w\ [i,\ j]$, else calculates the $w[i,\ j]$ and store it in the field $(i,\ j)$ of the table and return. In determining the value of the $Optw$ each time return $w[i,\ j]$, it is calculated only if it first appears where the optw is called with the parameters *i* and *j*.

In the *Java* NetBeans environment, was developed the main class OptimalTriangulation. This main class has the method compute that is responsible for calculating all the weights in table. In the working with table cells, was used the DefaultTableCellRenderer from *Swing* package. This class inherits the Table class and allows manipulation over the table cells (in this case, it allows to assign a series of independent values to one cell of the table).

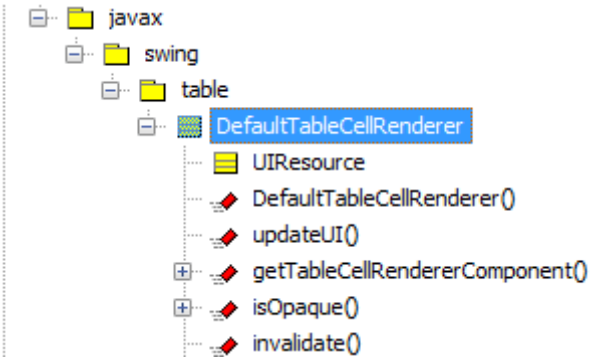


Figure 5. 23 The class `DefaultTableCellRenderer` and some of applications

In addition to that class, they were also used *JTable*, *TableCellRenderer*, *BasicTableUI* from *Swing* package. The obtained values can be recorded in the form of the table in the execution of the application *Java* NetBeans in the service for working with databases. The *defaultTableCellRenderer* class source code part to join the integer string to the table cells is:

```
public Component getTableCellRendererComponent(JTable table, Object obj,
boolean isSelected, boolean hasFocus, int row, int column) {

    Component cell = super.getTableCellRendererComponent(table, obj,
    isSelected, hasFocus, row, column);
    int[] array;
    array = new int[10];
    cell.add(array);
    return cell; }
```

In the NetBeans environment, there is the ability to load a matrix into the JTable component via SQL Script Editor:

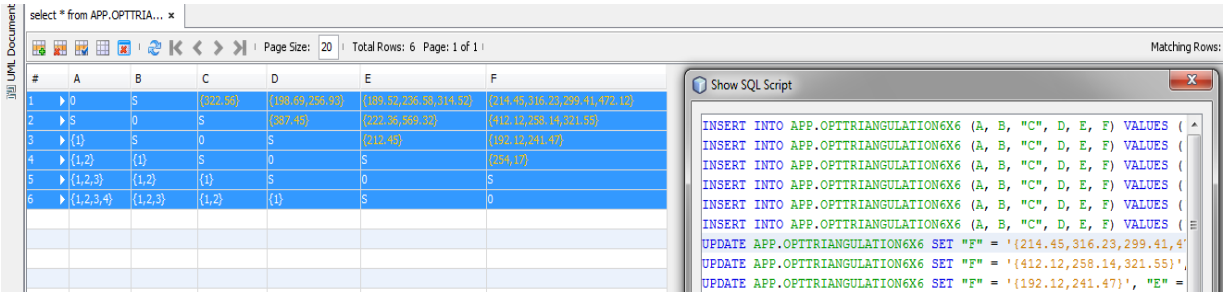


Figure 5.24 Loading of a values in JTable via SQL Script

Part of the source code of the method for dividing cells into columns and rows:

```
CellSpan cellAtt =
(CellSpan)((AttributiveCellTableModel)getModel()).getCellAttribute();
if (! cellAtt.isVisible(row,column)) {
      int temp_row = row; int temp_column = col;
      row += cellAtt.getSpan(temp_row,temp_column)[CellSpan.ROW];
      col += cellAtt.getSpan(temp_row,temp_column)[CellSpan.COLUMN];}
```

Is established *OptimalTriangulation* database in the *Java* service which contains a database system for working with persistent values of triangulation weights on the table. These tables correspond to filling schemes:

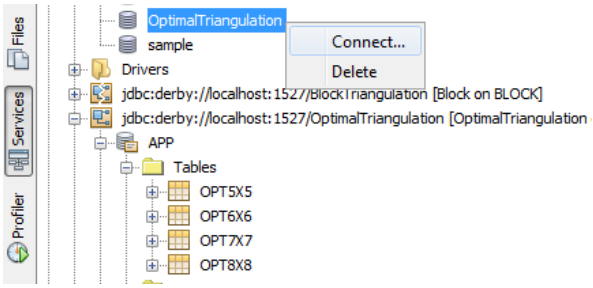


Figure 5.25: Java service for connection to the *OptimalTriangulation* base

Part of the source code for loading the triangulation block with corresponding table after the establishment of the JDBC connection:

```
FileInputStream data = new FileInputStream("\baze\T"+n+".jdb");
DataInputStream in = new DataInputStream(data);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
```

In the third step of the method, you add a new column to the loaded table. Part of the source code of the method that adds a new column is given below:

```
T = new DefaultTableModel(data,col);
T.addColumn("W"); JTable table = new JTable(T);
```

After adding a new column, in the third step of the algorithm, we add the weights for the current row of the table. Below is given a part of the source code of the method that loads the weight for the positions and assigns values to the new column *W*:

```
FileInput in = new FileInput(args[0]);
int[][] matrica = new int[rows][cols];
for(int i=0; i < rows; i++) {
for(int j=0; j < cols; j++) {
matrica[i][j] = Integer.parseInt(line[j]);
DefaultTableModel T = new DefaultTableModel(matrica,col); }}
```

The application contains a central panel and the *Toolbar*. The central panel allows you to enter points (n). Coordinates of the entered vertices are determined by the user by clicking *JPanel*.

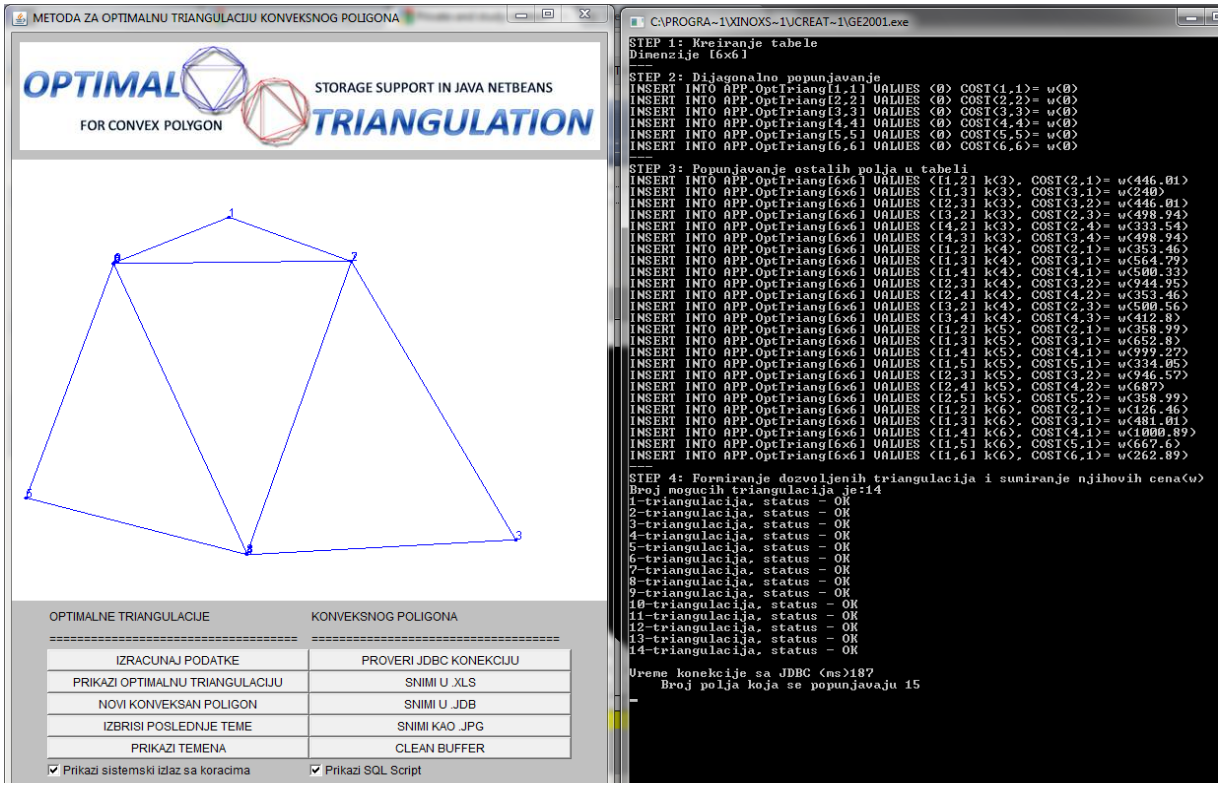


Figure 5.26: Java application for finding an optimal triangulation

Table 3 shows the calculation time for two methods: 1) the MWT method based on the square matrix and 2) the MWT method based on memoization (A- Filling time of the matrix, without the graphical generation of the minimum triangulation and B- total execution time, i.e. Finding optimal triangulation with plotting).

Table 3: Experimental results for the MWT method based on the square matrix and the MWT method based on memoization

n	***Number of Triangulations***	MWT with square matrix [70]		MWT with memoization		**RATIO (A)**	**RATIO (B)**
		(A)	*(B)*	*(A)*	*(B)*		
5	5	0.01	1.7	0.01	0.8	**1.00**	**2.13**
6	14	0.03	2.4	0.02	1.4	**1.50**	**1.71**
7	42	0.09	2.8	0.05	2.1	**1.80**	**1.33**
8	132	0.12	3.7	0.09	3.0	**1.33**	**1.23**
9	429	0.19	5.2	0.10	4.7	**1.90**	**1.11**
10	1430	0.27	14.8	0.13	13.3	**2.08**	**1.11**
11	4,862	0.41	27.4	0.19	24.9	**2.16**	**1.10**
12	16,796	0.62	48.7	0.31	44.1	**2.00**	**1.10**
13	58,786	1.04	64.6	0.63	59.3	**1.65**	**1.09**
14	208,012	1.59	85.5	1.04	78.5	**1.53**	**1.09**

In this case for testing was used the module of the NetBeans environment, the Profiler for CPU testing. Testing was performed on a computer with the following performance: *CPU – Intel Core2 Duo, 2.40GHz, Cache 4MB, RAM: 2Gb, Graphic: NVIDIA GeForce 8600M GS.* Table 3 presents the testing results for $n = \{5,6,\ldots,14\}$. The significance of this method of storage is reflected in the fact that the obtained values from the above method can effectively provide the drawing of the least weight triangulation, provided that there is a method that will ensure the generation of all triangulations. In addition to the speed of searching and plotting optimal triangulation, it is important to emphasize that this saves the memory.

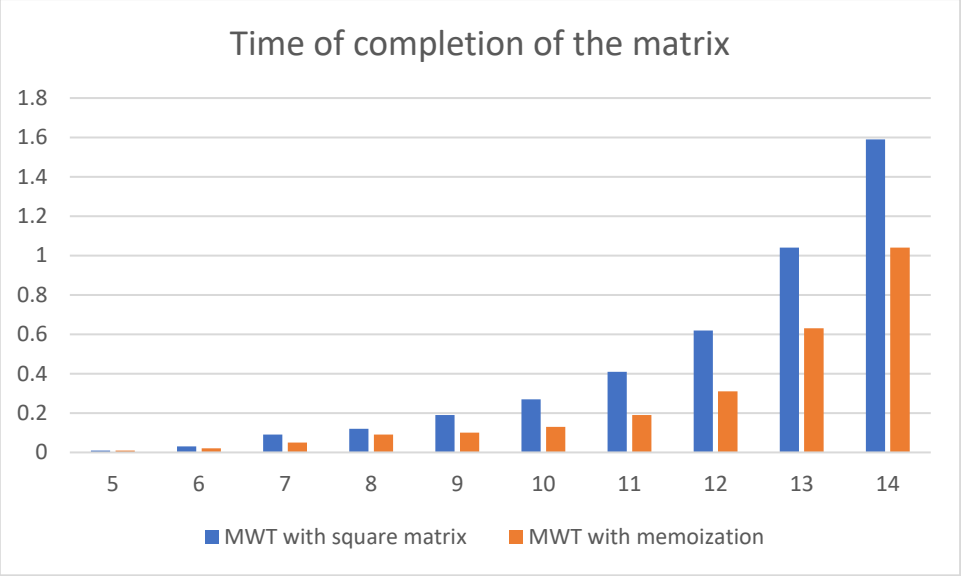


Figure 5.27 Time of completion of the matrix

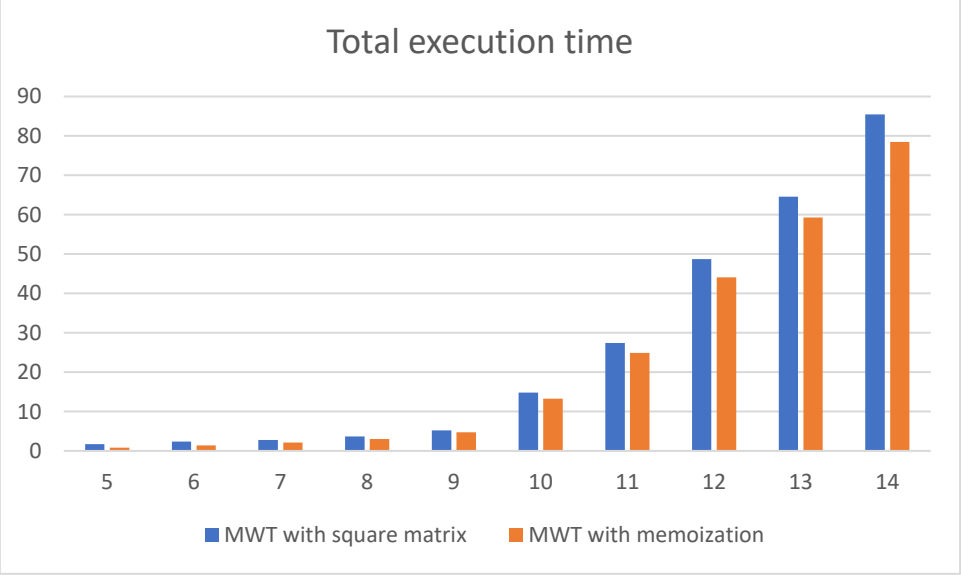


Figure 5.28: Total execution time

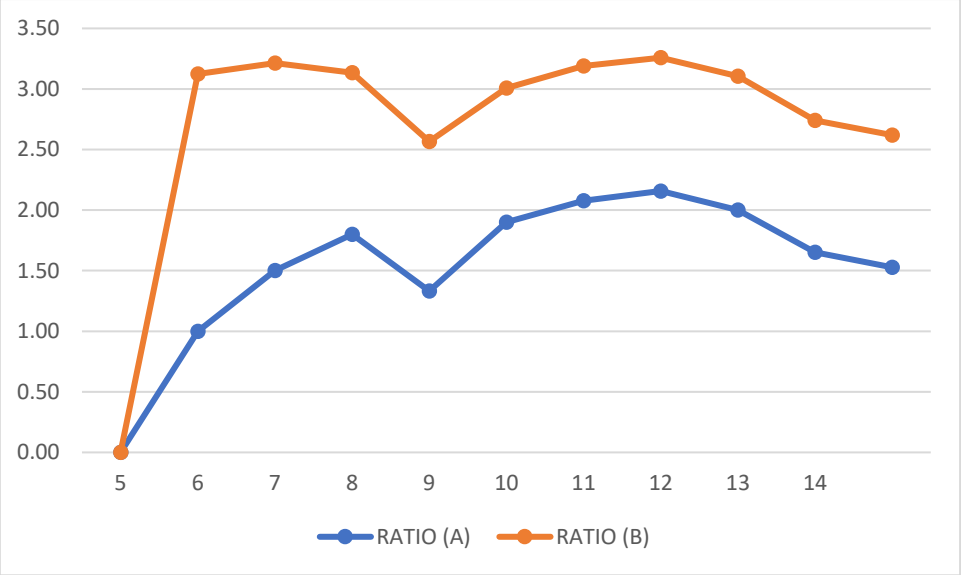


Figure 5.29: Ratio – MWT Square vs MWT memoization

Conclusion

This paper describes an algorithm that finds optimal triangulation of a convex polygon based on the dynamic programming technique. Memoization as part of dynamic programming is a technique that is used in the algorithm. As approach memoization give very effective and suitable for solving certain types of complex problems. The basic idea in the method developed in this research is to avoid multiple calculations of the same value by using additional space in which are stored the values between among results.

The constructed algorithm is formulated using the rows and columns of the $n \, x \, n$ table. The algorithm finds optimal triangulation on the principle of rows/ column pairing on different sides of the table diagonal. The task of optimization is achieved with the memoization technique that corresponds to the values found in the upper and lower part of the table. The complete cell fill strategy in the table is developed based on the calculation of the weights of the triangles that are parts of the polygon triangulation. The filling is done by finding the mutual matching of the values between the cell tables found on different sides of the diagonal.

We plan to expand our algorithm in the case of concave polygon and polyhedron. Optimal triangulation process of the concave polygon will begin by clipping off the ears, which will end up with another closed polygon, which will own two ears to clip off. This procedure is will repeated until has no more ears left to clip off. We want to use memoization

technique in ear clipping strategy. By choosing an interior point and drawing the edges to the three vertices of the triangle that contains this point we want to develop our algorithm in the case of 3D shapes. The algorithm will be ended when all interior points of the polyhedron will be exhausted.

6 Implementation

The implementation of the developed methods in the dissertation was realized through the following classes: Triangulation and GenerateTriangulation. The cl

6.1 Source Code of Triangulation class in Java

```
import java.io.IOException;
import java.util.Vector;

public class Triangulation {
	static int XMARGIN = 50;
	static int XDELTA  = 80;
	static int XLIMIT  = 600;
	static int YDELTA  = 92;
	static int YLIMIT = 832;

	private int sCursorX;
	private int sCursorY;
	private Vector<Point> points;
	private PostScriptWriter writer;
	private Vector<Double> sSine;
	private Vector<Double> sCosine;

	public Triangulation(int edges, PostScriptWriter writer) {
	  this.sCursorX = Triangulation.XLIMIT;
	  this.sCursorY = Triangulation.YLIMIT;
	  this.points = new Vector<Point>();
	  this.writer = writer;

	  this.sSine = new Vector();
	  this.sCosine = new Vector();

	  for (int k=0; k < edges; k++) {
	    double d = (4*k+edges)*Math.PI/(2*edges);
	    this.sSine.add(30*Math.sin(d));
	    this.sCosine.add(30*Math.cos(d));
	  }
	}

	public void clear() {
		this.points.removeAllElements();
	}
```

```
	public void copyFrom(int aOffset, Node t) {
		if (!(t instanceof LeafNode)){
			this.copyFrom(aOffset,t.getLeft());
			this.copyFrom(aOffset+t.getLeft().leaves(),t.getRight());
		}
	   this.points.add(new Point(aOffset, aOffset + t.leaves()));
	}

	public void Draw() throws IOException {
		if (Triangulation.XLIMIT <= this.sCursorX) {
			this.sCursorX = Triangulation.XMARGIN;
			this.sCursorY -= Triangulation.YDELTA;
		}

	   for (int i = 0; i < this.points.size(); i++) {
		Point p = this.points.get(i);
		this.writer.drawLine (
		-this.sCosine.get(p.x)+this.sCursorX, this.sSine.get(p.x)+this.sCursorY,
		-this.sCosine.get(p.y)+this.sCursorX, this.sSine.get(p.y)+this.sCursorY
		);
	   }
	   this.sCursorX += Triangulation.XDELTA;
	}

	public void DrawAll(Vector<Node> trees) throws IOException {
	   this.writer.psHeader();
	   for (int i = 0; i < trees.size(); i++) {
		Node t = trees.get(i);
		this.clear();
		this.copyFrom(0, t);
		this.Draw();
	   if (i != 0 && i % 55 == 0) {
	      this.sCursorX = Triangulation.XLIMIT;
	      this.sCursorY = Triangulation.YLIMIT;
	      this.writer.newPage();
	   }
		}
	   this.writer.Trailer();
	}

}
```

6.2 Source Code of GenerationTriangulation in Java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.Vector;
import javax.swing.*;
import java.awt.event.*;

import java.awt.*;
import javax.swing.event.*;
import java.awt.Toolkit;
import javax.swing.JToolBar;
import java.io.*;

import java.util.*;
import java.io.*;
import java.net.*;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import javax.swing.border.*;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.BufferedWriter;
import java.io.FileWriter;

public class GenerateTriangulations extends JFrame implements ActionListener {

   static int n;
   static int val;
   private PostScriptWriter writer;


   JLabel naslov = new JLabel("TRIANGULATION: HURTADO-NOY HIERARCHY");
   JLabel nove = new JLabel("PANEL ZA UNOS PARAMETARA");
   JLabel l1 = new
JLabel("_____");
   JLabel labela2 = new JLabel("Unesite broj temena poligona (n)");
```

```
    JTextField polje = new JTextField(10);

    JButton dugme1 = new JButton("Kreiraj triangulacije samo za nivo (n)");
    JButton dugme2 = new JButton("Kreiraj triangulacije od n=3 do unetnog (n)");

    JLabel slika1 = new JLabel();

    GridLayout gl = new GridLayout(2, 1);
    Checkbox c1 = new Checkbox("Omoguci automatsko otvaranje izlaznog fajla");
    Checkbox c2 = new Checkbox("Ukljuci sistemski izlaz");


public GenerateTriangulations () {
        super("Triangulation Convex Polygon - Hurtado-Noy Hierarchy");
        setSize(360, 540);
        slika1.setIcon(new ImageIcon("baner.jpg"));

        JPanel pane = new JPanel();
        setLayout(gl);
        pane.add(slika1);
        pane.add(naslov);
        pane.add(l1);
        pane.add(nove);
        pane.add(labela2);
        pane.add(polje);
        pane.add(dugme1);
        pane.add(dugme2);
        pane.add(c1);
        pane.add(c2);

        dugme1.addActionListener(this);
        //dugme2.addActionListener(this);
        setContentPane(pane);
}


public Vector<Node> Hurtado (int limit) throws java.io.IOException {
                    Vector<Vector> bl = new Vector<Vector>();
              //System.out.println("NASTAVI OD NIVOA "+ (limit+1));
              this.openFile("baze/[T"+(limit+1)+" BAZA].jdb");
                    bl.add(new Vector<LeafNode>());
                    bl.get(0).add(new LeafNode());
                    for (int h=0; h <= limit; h++) {
                 Vector<Node> level = new Vector();
```

```
                    for (int q = 0; q < bl.get(h).size(); q++) {
                if(h==limit) System.out.println("T=["+(q+1)+"]");
                        Node t = (Node)bl.get(h).get(q);
                        Node s = new Node(new LeafNode(), t.copy());
                        level.add(s);
                        for (int k = 0; k < t.leftBranch(); k++) {
                    s = t.copy();
                    Node r = s;
                                for (int i=0; i < k; i++)
                       s = s.getLeft();
                       // ZA BAFER ZA PROMENE
                       if(h>=limit) System.out.println("["+(q+1)+"-"+(k+1)+r+"]");
                       s.setLeft(new Node(new LeafNode(), s.getLeft()));
                       level.add(r);

//////////////
            BufferedWriter bufferedWriter = null;
                 BufferedWriter bufferedWriter1 = null;
      try {
         bufferedWriter = new BufferedWriter(new FileWriter("notation.txt",true));
         bufferedWriter.write(""+r+"");
         bufferedWriter.newLine();

      } catch (FileNotFoundException ex) {
         ex.printStackTrace();
      } catch (IOException ex) {
         ex.printStackTrace();
      } finally {
         try {
            if (bufferedWriter != null) {
               bufferedWriter.flush();
               bufferedWriter.close();
            }
         } catch (IOException ex) {
            ex.printStackTrace();
         }
      }
//////////////

                        } System.out.println("");
                    }
                     bl.add(level);

                }
```

```
return bl.get(limit);
}



public void actionPerformed (ActionEvent evt) {
   if (evt.getSource()==dugme1){
         try{
         n = Integer.parseInt(polje.getText());
         FileWriter fstream = new FileWriter(n+"-polygons.ps");
                              BufferedWriter out = new BufferedWriter(fstream);
                              PostScriptWriter writer = new PostScriptWriter(out);
                              GenerateTriangulations gt = new GenerateTriangulations();
         Vector<Node> btbl = gt.Hurtado(n-2);
                              Triangulation mPicture = new Triangulation(n,writer);
         mPicture.DrawAll(btbl);
                              out.close();
                     }
      catch (Exception e){
                     e.printStackTrace();
                     }
      this.openFile(String.valueOf(n)+"-polygons.ps");
   }
   }



public void openFile(String filename) {
try {

                     if ((new File(filename)).exists()) {

                              Process pr = Runtime
                                .getRuntime()
                                .exec("rundll32 url.dll,FileProtocolHandler " + filename);
                              pr.waitFor();
            System.out.println(" ");
            labela2.setText("POSTOJI REZULTAT ZA UNETO N, SLEDI REZULTAT");
                     } else {
                              System.out.println("Ne postoji rezultat za uneto n");
                     labela2.setText("NE POSTOJI REZULTAT ZA UNETO N, UNESITE
PONOVO");
                     }
```

```
            } catch (Exception ex) {
                    ex.printStackTrace();
            }
}



public static void main (String[] args) {
      GenerateTriangulations rb = new GenerateTriangulations();
      rb.show();
         }
}
```

7. Bibliography

[1] Akl, S., "A constant-time parallel algorithm for computing convex hulls," *BIT*, vol. 22, 1982, pp. 130-134.

[2] Akl, S., *The Design and Analysis of Parallel Algorithms*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[3] Alt, H. and Yap, C. K., "Algorithmic aspects of motion planning: a tutorial, part 1," *Algorithms Review*, 1990, vol. 1, No. 1, pp. 43-60.

[4] Alt, H. and Yap, C. K., "Algorithmic aspects of motion planning: a tutorial, part 2," *Algorithms Review*, 1990, vol. 1, No. 2, pp. 61-78.

[5] Amounas, F., El-Kinani, E.H., Hajar, M. Novel Encryption Schemes Based on Catalan Numbers, International Journal of Information & Network Security, 2013, Vol.2, No.4, pp. 339-347.

[6] Avis, D. and Toussaint, G. T., "An efficient algorithm for decomposing a polygon into star-shaped pieces," *Pattern Recognition*, vol. 13, 1981, pp. 295-298.

[7] Batty, C., Xenos, S., and Houston, B. 2010. Tetrahedral embedded boundary methods for accurate & flexible adaptive fluids. Comp. Graph. Forum (Eurographics) 29, 695–704.

[8] Bernard Chazelle. Triangulating a simple polygon in linear time. Discrete Comput. Geom., 6(5): 485–524, August 1991.

[9] Boyd Stephen., Vandenberghe Lieven., *Convex Optimization*, Cambridge University Press, 2004

[10] Chvatal, V., "A combinatorial theorem in plane geometry," *Journal of Combinatorial Theory, Series B*, vol. 18, 1975, pp. 39-41.

[11] Cohen, E., Hansen, T., Itzhaki, N. From entanglement witness to generalized Catalan numbers, Scientific Reports, 2016, Vol.6, No.3.

[12] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L. and Clifford Stein, Introduction to Algorithms, *Third Edition*, MIT Press, 2009

[13] Danny Z. Chen , Jinhui Xu Two-variable linear programming in parallel, Computational Geometry 21 (2002) 155–165

[14] de Berg M., Cheong O., Kreveld van M., Overmars M., Computational Geometry Algorithms and Applications, Springer-Verlag Berlin Heidelberg, 2008

[15] de Berg, Mark, *Efficient Algorithms for Ray Shooting and Hidden Surface Removal*, University of Utrecht, 1992.

[16] De Jesús A. L., Rambau J. and Santos F., Triangulations Structures for Algorithms, Springer-Verlag Berlin Heidelberg, Volume 25, 2010

[17] Devadoss S. L. and Rourke J. Discrete and computational geometry, Princeton University Press, 2011

[18] Dobkin, D. and Silver, D., "Recipes for geometry & numerical analysis - Part I: An empirical study," *Proc. 4th Annual Symposium on Computational Geometry*, Urbana, June 1988, pp. 93-105.

[19] Edelsbrunner, H. and Mucke, E., "Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms," *Proc. 4th Annual Symposium on Computational geometry*, Urbana, Illinois, June 1988, pp. 118-133.

[20] Elcott, S., Tong, Y., Kanso, E., Schröder, P., and Desbrun, M. 2007. Stable, circulation-preserving, simplicial fluids. ACM Trans. Graph. 26, 1 (Jan.).

[21] Even, S. Graph Algorithms. Cambridge University Press, 2nd ed., New York, 2011.

[22] De Goes, F., Breeden, K., Ostromoukhov, V., And Desbrun, M. 2012. Blue noise through optimal transport. ACM Trans. Graph. (SIGGRAPH Asia) 31, 171:1–171:11.

[23] G.T. Klincsek. Minimal triangulations of polygonal domains. In Peter L. Hammer, editor, Combinatorics 79, volume 9 of Annals of Discrete Mathematics, pages 121 –123. Elsevier, 1980.

[24] Gerhard Roth and Eko Wibowoo. An efficient volumetric method for building closed triangular meshes from 3-d image and point data. In Proceedings of the conference on Graphics interface '97, pp. 173–180, 1997.

[25] Gill and Micha Sharir. Filling gaps in the boundary of a polyhedron. Comput. Aided Geom. Des., 12(2):207–229, March 1995.

[26] Gill Barequet, Matthew Dickerson, and David Eppstein. On triangulating three dimensional polygons. In Proceedings of the twelfth annual symposium on Computational geometry, SCG '96, pages 38–47, New York, NY, USA, 1996. ACM.

[27] Goodman, J.E., O'Rourke, J. Handbook of Discrete and Computational Geometry. Chapman and Hall and CRC Press, New York, 2004.

[28] Guibas, L. and Hershberger, J., "Optimal shortest path queries in a simple polygon," *Journal of Computer and System Sciences*, vol. 39, No. 2, 1989, pp. 126-152.

[29] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer, New York, 1987

[30] Hershberger, J., “Optimal parallel algorithms for triangulated simple polygons,” *Proc. 8th ACM Symposium on Computational Geometry,* Berlin, June 10-12, 1992, pp. 33-42

[31] Higgins, P.M. Number Story: From Counting to Cryptography, Springer Science & Business Media, Berlin, Germany, 2008.

[32] Horak, P., Semaev, I., Tuza, I. Z. An application of Combinatorics in Cryptography, Electronic Notes in Discrete Mathematics, 2015, Vol. 49, pp. 31-35.

[33] Hurtado, F., Noy, M. Graph of Triangulations of a Convex Polygon and tree of triangulations. Computational Geometry 13, 179{188, 1999.

[34] Imai H, Computational Geometry and Linear Programming, Algorithms & Architectures: Proceedings of the Second NEC Research Symposium, (1991), 9 – 28

[35] Imai, H. and Iri, M., “Polygonal approximations of a curve - formulations and algorithms,” in *Computational Morphology,* G. T. Toussaint, ed., North-Holland, 1988, pp. 71-86.

[36] Jin, M., Kim, J., Luo, F., and Gu, X. 2008. Discrete surface ricci flow. IEEE Tran. Vis. Comp. Graph. 14, 5 (sept.-oct.), 1030–1043.

[37] Karasik, Y. B. and Sharir, M., “Optical computational geometry,” *Proc. 8th ACM Symposium on Computational Geometry*, Berlin, June 10-12, 1992, pp. 232-241.

[38] Kenneth Rose, Alla Sheffer, Jamie Wither, Marie-Paule Cani, and Boris Thibert. Developable surfaces from arbitrary sketched boundaries. In Proc. Eurographics Symposium on Geometry processing, 2007.

[39] Kościelny, C., Kurkowski, M., Srebrny, M. Modern Cryptography Primer: Theoretical Foundations and Practical Applications, Springer Science & Business Media, Berlin, Germany, 2013.

[40] Koshy, T. Catalan Numbers with Applications, Oxford University Press, New York, 2009.

[41] Koshy, T. Discrete Mathematics with Applications. Elsevier Academic Press, Burlington, 2004.

[42] Kreveld, M. van, *New Results on Data Structures in Computational Geometry*, University of Utrecht, 1992.

[43] Lachaud, G., Ritzenthaler, C., Tsfasman, M.A. Arithmetic, Geometry, Cryptography, and Coding Theory, American Mathematical Society, United States, 2009.

[44] Lew A., Mauch H., Dynamic Programming A Computational Tool, Springer-Verlag Berlin Heidelberg, 2007

[45] M.E. Dyer, A parallel algorithm for linear programming in fixed dimension, in: Proc. of 11th Annual Symp. on Computational Geometry, 1995, pp. 345–349.

[46] Marco Attene, Marcel Campen, and Leif Kobbelt. Polygon mesh repairing: An application perspective. ACM Comput. Surv., 45(2):15:1–15:33, March 2013.

[47] Mašović S., Saračević M., Finding Optimal Triangulation Based On Block Method, Southeast Europe Journal of Soft Computing, VOL.3 NO.2 , 2014

[48] Mašović, S., Saračević, M., Stanimirović, P. Alpha-Numeric notation for one Data Structure in Software Engineering, Acta Polytechnica Hungarica: Journal of Applied Sciences, 2014, Vol.11, No.1, pp.193-204.

[49] Melter, R. A., Rosenfeld, A. and Bhattacharya, P., eds., *Vision Geometry*, American Mathematical Society, 1991.

[50] Mercat, C. 2001. Discrete Riemann surfaces and the Ising model. Comm. Math. Phys. 218, 177–216.

[51] Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh. Design-driven quadrangulation of closed 3d curves. Transactions on Graphics (Proc. SIGGRAPH ASIA 2012), 31(5), 2012.

[52] Milenkovic, V., "Verifiable implementations of geometric algorithms using finite precision arithmetic," Tech. Rept. CMU-CS-88-168, Carnegie Mellon University, July 1988.

[53] Minsky, M. and Papert, S., *Perceptrons: An Introduction to Computational Geometry*, M.I.T. Press, 1969.

[54] Mortenson, M. E., *Geometric Modeling*, John Wiley & Sons, 1985.

[55] Nimrod Megiddo, Linear programming in linear time when the dimension is fixed, J. ACM 31 (1) (1984) 114–127.

[56] Nimrod Megiddo, "On the complexity of linear programming" in: Advances in economic theory Fifth world congress, T. Bewley, ed., Cambridge University Press, 1987, pp. 225-268.

[57] Nimrod Megiddo, Linear time algorithms for linear programming in R^3 and related problems, SIAM Journal of Computation 12 (4) (1983) 759–776.

[58] O'Rourke J., Computational Geometry in C, Second Edition, Cambridge University Press, 1997

[59] P. D. Gilbert. New results in planar triangulations. Technical report, Urbana, Illinois: Coordinated Science Laboratory, University of Illinois, 1979.

[60] Peter Liepa. Filling holes in meshes. In Symposium on Geometry Processing, pp. 200–206, 2003.

[61] Preparata, F. P. and Shamos, M. I., *Computational Geometry,* Springer-Verlag, NewYork, 1985.

[62] R. Seidel, Small-dimensional linear programming and convex hulls made easy, Discrete Computational Geometry. 6 (1991) 423–434.

[63] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Comput. Geom., 1:51–64, 1991

[64] Roman S., An Introduction to Catalan Numbers, Springer Cham Heidelberg New York Dordrecht London, 2015

[65] Rosen Kenneth H., Discrete Mathematics and Its Applications, Seventh Edition, McGraw-Hill Companies, USA, 2012.

[66] Saračević M. and Selimi A., Convex Polygon Triangulation Based on Planted Trivalent Binary Tree 2 and Ballot Problem, Turkish Journal of Electrical Engineering & Computer Sciences , doi:10.3906/elk (accepted)

[67] Saračević M., Mašović S., Kamberović H. (2012), Procedure division of convex polygon triangulation and application in computer graphics, *6th International quality conf., Faculty of Engineering*, ISBN: 978-86-86663-82-5, Center for Quality, Vol.2, pp. 991.

[68] Saračević M., Mašovic S., Miloševic D. (2013), Java implementation for triangulation of convex polygon based on Lukasiewicz's algorithm and binary trees, *Souteast European Journal of Soft Computing*, ISSN: 2223 -1859, Vol.2, No.2.

[69] Saračević M., Mašović S., Milošević D., Kudumović M. (2013), Proposal for applying the optimal triangulation method in 3D medical image processing: Software solution based on Java Net Beans environments, *Balkan Journal of Health Science* (BJHS), ISSN: 2303 -4092, Vol.1, No.1, pp. 27-34.

[70] Saračević M., Mašović S., Stanimirović P., Krtolica P.,., Method for finding and storing optimal triangulations based on square matrix, Applied Sciences (electronic journal), Volume 20 (2018), pp. 167-180.

[71] Saračević M., Selimi A., and Selimovic F., Generation of cryptographic keys with algorithm of polygon triangulation and Catalan numbers, Computer Science, 19(3), 2018

[72] Saračević M., Stanimirović P., Krtolica P., Mašović S. (2014), Construction and Notation of Convex Polygon Triangulation based on ballot problem, *ROMJIST-Journal of Information Science and Technology*, ISSN: 1453–8245, Vol.17, No.3, pp. 237-251.

[73] Saračević M., Stanimirović P., Mašović S. (2013), Object-oriented analysis and design for one algorithm of computational geometry: Forward, reverse and round-trip engineering, *Journal of Information Technology and Applications*, ISSN: 2232-9625, DOI: 10.7251/JIT1302096S, Vol.3, No.2, pp. 96-106.

[74] Saračević M., Stanimirović P., Mašović S., Biševac E, Implementation of the convex polygon trangulation algorithm, Facta Universitatis, series: Mathematics and Informatics Vol.27, No.2, pp. 213–228, 2012.

[75] Saračević M. and Selimi A., Application of the Computational Geometry in Linear Optimization, International Scientific Journal Vision, September, 2017; 2 (2)

[76] Saračević, M. Application of Catalan numbers and some combinatorial problems in cryptography (Bachelor's thesis), Faculty of Informatics and Computing, Singidunum University in Belgrade, 2017.

[77] Saračević, M. Methods for solving the polygon triangulation problem and their implementation (PhD thesis), Faculty of Science and Mathematics, University of Niš, 2013.

[78] Saračević, M., Korićanin, E., Biševac, E. Encryption based on Ballot, Stack permutations and Balanced Parentheses using Catalan-keys, Journal of Information Technology and Applications, 2017, Vol.7, No.2.

[79] Saračević, M., Stanimirović, P., Krtolica, P., Mašović, S. Construction and Notation of Convex Polygon Triangulation Based on Ballot Problem. Romanian Journal of Information Science and Technology 17(3), 237{251, 2014.

[80] Saračević, M., Stanimirović, P., Mašović, S., Biševac, E. Implementation of the convex polygon trangulation algorithm. Facta Universitatis, Series Mathematics and Informatics 27(2), 213{228, 2012.

[81] Saračević, Mašović, S, Milošević, D. JAVA Implementation for Triangulation of Convex Polygon Based on Lukasiewicz Algorithm and Binary Tree. Southest Europe Journal of Soft Computing 2(2), pp. 40-45, 2013.

[82] Selimi A. and Saračević M., Computational Geometry Applications, Southeast Europe Journal of Soft Computing., Vol.7, No.2, 2018

[83] Sen-Gupta, S., Mukhopadhyaya, K., Bhattacharya, B. B., Sinha, B. P. Geometric Classification of Triangulations and Their Enumeration in a Convex Polygon. Computers and Mathematics with Applications 27, 99{115, 1994.

[84] Sniedovich M., Dynamic Programming Foundations and Principles, Second Edition, Taylor and Francis Group, 2011.

[85] Stanimirovic P, Krtolica P, Saracevic M, Masovic S (2012) “Block Method for Convex Polygon Triangulation”, *Romanian Journal of Information Science and Technology - ROMJIST*, Vol.15, No.4: pp. 344-354

[86] Stanimirović, P., Krtolica, P., Saračević, M., Mašović, S. Decomposition of Catalan numbers and Convex Polygon Triangulations, International Journal of Computer Mathematics, 2014, Vol. 91, No. 6, pp. 1315-1328.

[87] Stanley, R. P. Catalan addendum to Enumerative Combinatorics, [on-line], 2012, http://www-math.mit.edu/~rstan/ec/catadd.pdf. [Available 24.05.2017.]

[88] Liu, Y., Hao, P., Snyder, J., Wang, W., And Guo, B. 2013. Computing self-supporting surfaces by regular triangulation. ACM Trans. Graph. (SIGGRAPH) 32.

[89] Tabak J., *The History of Mathematics and the Laws of Nature: Developing the Language of Science*, Facts On File, Inc. NewYork, 2004.

[90] Tabak J., *The History of Mathematics Geometry: The Language of Space and Form*, Facts On File, Inc. NewYork, 2004.

[91] Toussaint, G. T., “Efficient triangulation of simple polygons,” *The Visual Computer*, vol. 7, No. 5-5, September 1991, pp. 280-295.

[92] Toussaint, G. T., “What is Computational Geometry,” *Computational Geometry Laboratory*” School of Computer Science McGill University. Montreal, Quebec, Canada

[93] Wilf Herbert S., Generatingfunctionology, Third Edition, A K Peters, Ltd, 2006.

[94] Wood, D., “An isothetic view of computational geometry,” in *Computational Geometry*, G. T. Toussaint, ed., North-Holland, 1985, pp. 429-459.

[95] Zou M., Tao J. and Carr N., An algorithm for triangulating multiple 3D polygons, Eurographics Symposium on Geometry Processing, *Volume 32* (*2013*), *Number 5.*

Aybeyan Selimi, M. Sc.
International Vision University, Faculty of Informatics
Gostivar, Republic of Macedonia
aybeyan@vizyon.edu.mk

8. Biography of the author

Born 6 July 1980 in Tetovo, Macedonia. In 1999, he enrolled as a regular student in the Institute of Mathematics at the Faculty of Natural Sciences and Mathematics at the University "St. Cyril and Methodius University in Skopje. Graduate in 2004 with a diploma thesis titled "Binomial and Multinomial Coefficients" under the mentoring of the Academic Doncho Dimovski, with average success 8.48. The same year he is employed as a teacher of a mathematics at the high school "Pance Popovski" in Gostivar. Since 2004 he is a member of the Association of Mathematicians of the Republic of Macedonia and actively participates in the organization and realization of the mathematics competitions for primary and secondary education. In 2015 he defended his master's thesis entitled “Quadratic programming and its application to the analysis of the mean-variance model in portfolio optimization” at the Institute of Mathematics, the Faculty of Natural Sciences and Mathematics in Skopje with average success 9.75. He registered his doctoral studies in 2015 in the Faculty of Computer Science at the University of Novi Pazar and passed all the exams expiring right to the defense of the doctoral dissertation. Currently, he is employed as a lecturer at the International University Vision, Faculty of Informatics in Gostivar. Research interests are on computational geometry and mathematical programming.