B.Sc. in Computer Science and Engineering Thesis

A Critical Assessment of Graceful Graphs and Trees: Study and Development of New Approaches

Submitted by

Mohammad Abdul Kader
ID: 201114002

Sharmin Manjur
ID: 201114035

Md. Tasnim Manzur Ankon
ID: 201114048

Supervised by

Group Captain Md. Afzal Hossain, psc

Head of the Department

Department of Computer Science and Engineering (CSE)

Military Institute of Science and Technology (MIST)

Department of Computer Science and Engineering
Military Institute of Science and Technology

December 2014

CERTIFICATION

This thesis paper titled **"A Critical Assessment of Graceful Graphs and Trees: Study and Development of New Approaches"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in December 2014.

Group Members:

Mohammad Abdul Kader

Sharmin Manjur

Md. Tasnim Manzur Ankon

Supervisor:

Group Captain Md. Afzal Hossain, psc
Head of the Department
Department of Computer Science and Engineering (CSE)
Military Institute of Science and Technology (MIST)

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis paper, titled, "A Critical Assessment of Graceful Graphs and Trees: Study and Development of New Approaches", is the outcome of the investigation and research carried out by the following students under the supervision of Group Captain Md. Afzal Hossain, psc, Head of the Department, Department of Computer Science and Engineering (CSE), Military Institute of Science and Technology (MIST).

It is also declared that neither this thesis paper nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Mohammad Abdul Kader
ID: 201114002

Sharmin Manjur
ID: 201114035

Md. Tasnim Manzur Ankon
ID: 201114048

ACKNOWLEDGEMENT

We are thankful to Almighty Allah for His blessings for the successful completion of our thesis. Our heartiest gratitude, profound indebtedness and deep respect go to our supervisor, Group Captain Md. Afzal Hossain, psc, Head of the Department, Department of Computer Science and Engineering (CSE), Military Institute of Science and Technology (MIST), for his constant supervision, affectionate guidance and great encouragement and motivation. His keen interest on the topic and valuable advices throughout the study was of great help in completing this thesis. We also thank Lecturer Jannatul Maowa, Department of Computer Science and Engineering (CSE), Military Institute of Science and Technology (MIST), for her relentless guidance and timely advice in completing this thesis work. We are specially grateful to her for introducing us with the topic, as well as for sharing her analytical knowledge and research expertise through out this thesis work.

We are especially grateful to the Department of Computer Science and Engineering (CSE) of Military Institute of Science and Technology (MIST) for providing their all out support during the thesis work.

Finally, we would like to thank our families and our course mates for their appreciable assistance, patience and suggestions during the course of our thesis.

MIST, Dhaka
December 2014

Mohammad Abdul Kader

Sharmin Manjur

Md. Tasnim Manzur Ankon

ABSTRACT

A graph G is graceful if its set of vertices is denoted by the set $\{0, 1, \ldots, m\}$ and the set of edges is denoted by $\{1, 2, \ldots, m\}$, such that all edges are labeled uniquely and according to the difference of the labels of the vertices connecting it. A simple notation to be mentioned that all the vertices will have unique labels, and so will the edges. The idea of graceful labeling came to being from Ringel's conjecture(see chapter 1). The conjecture introduced in 1963, established the open problem of graceful labeling. Any graph that can be labeled gracefully suggests that the class of the graph implies Ringel's conjecture. Over the years many people worked on this conjecture and found out many classes of graceful trees and graph. Such graceful classes of trees and graphs are Path or Chain, Caterpillar, Extended caterpillar, super caterpillar, Star, Olive tree, Banana tree, Lobstar, product tree Cyle wheel, Crown graph etc. A C_4 graph is a cycle consisting of four vertices. The C_4 graph is graceful. However, graphs formed by multiple number of C_4 cycles in them were yet to be studied. Another type of graph is a star. A star S_1, n is a center vertex connected with n leaves. A star is also classified as a tree, according to the definition. It has been proved earlier that all stars are graceful. The purpose of this paper was to study and develop procedures to gracefully labeled graphs formed by the combination of multiple C_4 cycles and the combination of multiple stars. A general procedure was developed for each of the three classes, proved to be graceful, through this paper. Further research was conducted on finding more varieties of such graphs and the results were discussed. The result of the research proposes an open problem about a class of graphs which partially supports Ringel's conjecture. The paper highlights the classes of graphs and trees that were previously proved to be graceful.

TABLE OF CONTENT

LIST OF FIGURES

List of Algorithms

LIST OF SYMBOLS

$x \in A$: Element x is a member of set A

$\mid A \mid$: The cardinality of set A

$\lfloor x \rfloor$: The greatest integer less than or equal to the number x

$a \equiv b(\ mod\ n)$: a is congruent to b modulo n

$G = (V, E)$: G is a graph with vertex set V and edge set E

$T = (V, E)$: T is a tree with vertex set V and edge set E

O_G : A labeling of graph G

K_n : K is a complete graph of length n

C_n : C is a cycle of length n

W_n : W is a wheel obtained from the cycle C_n

R_n : R is a crown with $2n$ edges

H_n : H is a helm with $3n$ edges

P_n : P is a path or snake of length n

$D_n(m)$: D is a dragon obtained by joining the end point of path P_m to the cycle C_n

$\Delta_n - snake$: A triangular snake with n blocks

CHAPTER 1
INTRODUCTION

1.1 Overview

The theory of graphs has a definite starting point, which is an exception in the field of mathematics. It started when the Swiss mathematician Leonard Euler (1707-1783) considered the problem of the seven Konigsberg bridges. In the early 18th century the city of Konigsberg (in Prussia) was divided into four sections by the Pregel river. Seven bridges connected these regions as shown in Figure 1.1. Regions are shown by A, B, C, D respectively. It is said that the townsfolk of Konigsberg amused themselves by trying to find a route that crossed each bridge just once (It was all right to come to the same island any number of times). Euler discussed whether or not it is possible to have such a route by using the graph shown in Figure 1.1. He published the first paper in graph theory in 1736 to show the impossibility of such a route and give the conditions which are necessary to permit such a stroll. Graph

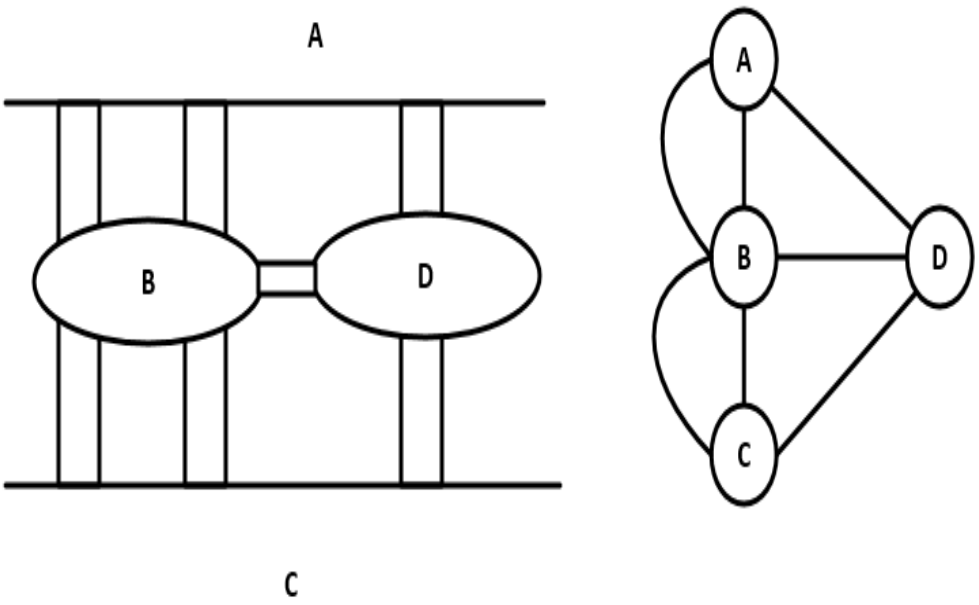

Figure 1.1: A map of Konigsberg and a graph representing the bridges of Konigsberg.

theory was born to study problems of this type. Graph theory is one of the topics in an area of mathematics described as Discrete Mathematics.

The problems as well as the methods of solution in discrete mathematics differ fundamentally from those in continuous mathematics. In discrete mathematics we "count" the number of objects while in continuous mathematics we "measure" their sizes. Although discrete mathematics began as early as man learned to count, it is continuous mathematics which has long dominated the history of mathematics. This picture began to change in twentieth century. The first important development was the change that took place in the conception of mathematics. Its central point changed from the concept of a number to the concept of a set which was more suitable to the methods of discrete mathematics than to those of continuous mathematics.

The second dramatic point was the increasing use of computers in society. Much of the theory of computer science uses concepts of discrete mathematics. Graph theory as a member of the discrete mathematics family has a surprising number of applications, not just to computer science but to many other sciences (physical, biological and social), engineering and commerce. Graceful graphs have a range of practical application domains, including in radio astronomy, X-ray crystallography, cryptography, and experimental design,coding theory and communication network addressing such as Multiprotocol Label Switching (MPLS) multicasting using Caterpillars and a Graceful. Graph theory has turned out to be a vast area with innumerable applications in the field of social networks, data organization, communication network, discrete mathematics and so on.

The purpose of our paper is to provide some results in a class of problems categorized as Graph labeling. Let G be an undirected graph without loops or double connections between vertices. In labeling (valuation or numbering) of a graph G, we associate distinct nonnegative integers to the vertices of G as vertex labels (vertex values or vertex numbers) in such a way that each edge receives a distinct positive integer as an edge label (edge value or edge number) depending on the vertex labels of vertices which are incident with this edge. Interest in graph labeling began in mid-1960s with a conjecture by Kotzig-Ringel and a paper by Rosa [1].

In 1963, ringel conjectured the following:

Conjecture 1. Let T be a given tree with n vertices and $n-1$ edges, then the edges of K_{2n-1} can be partitioned int $2n-1$ trees isomorphic to T.

Figure 1.2 shows the conjecture for T, a tree of 3 edges, and K_7. Tree T has 4 vertices or $n=4$. As per Conjecture 1 there are 7 distinct copies of T in K_7. It is interesting to consider the number of edges in K_{2n-1} relative to the number of edges in T. The number of edges in K_{2n-1} is equivalent to the number of ways one can choose 2 from the group $2n-1$ or

$$\binom{2n-1}{2} = (2n-1)(2n-2)/2 = (2n-1)(n-1)$$

Therefore, the number of distinct edges in K_{2n-1} is always $(2n-1)$ times $\mid E(T) \mid$. It remains to be proven that the partitioning results in trees isomorphic to T, but we know there exist the correct number of edges.

In 1967, Rosa published a pioneering paper on graph labeling problems. He called a function f a β-labeling of a graph G with n edges (Golomb [2]) subsequently called such labeling graceful and this term is now the popular one) if f is an injection from the vertices of G to the set $\{0, 1, \ldots, n\}$ such that, when each edge is labeled with the absolute value of the difference between the labels of the two end vertices, the resulting edge labels are distinct. This labeling provides a sequential labeling of the edges from 1 to the number of edges. Any graph that can be gracefully labeled is a graceful graph. Although numerous families of graceful graphs are known, a general necessary or sufficient condition for gracefulness has not yet been found. Also It is not known if all tree graphs are graceful. Another important labeling is an α-labeling or α-valuation which was also introduced by Rosa [1]. An α-valuation of a graph G is a graceful valuation of G which also satisfies the following condition:

There exists a number $\gamma(0 \leq \gamma < E(G))$ such that, for any edge $e \in E(G)$ with the end vertices $u, v \in V(G)$, min $\{$ vertex label (v), vertex label $(u)\} \leq \gamma <$ max $\{$ vertex label (v), vertex label $(u)\}$ It is clear that if there exists an α-valuation of graph G, then G is a bipartite graph. The first graph in Figure 3.1 is a path with five edges and it has an α-labeling with $\gamma = 3$. During the past thirty years, over 200 papers on this topics have been appeared in journals. Although the conjecture that all trees are graceful has been the focus of many of these papers, this conjecture is still unproved. Unfortunately there are few general results

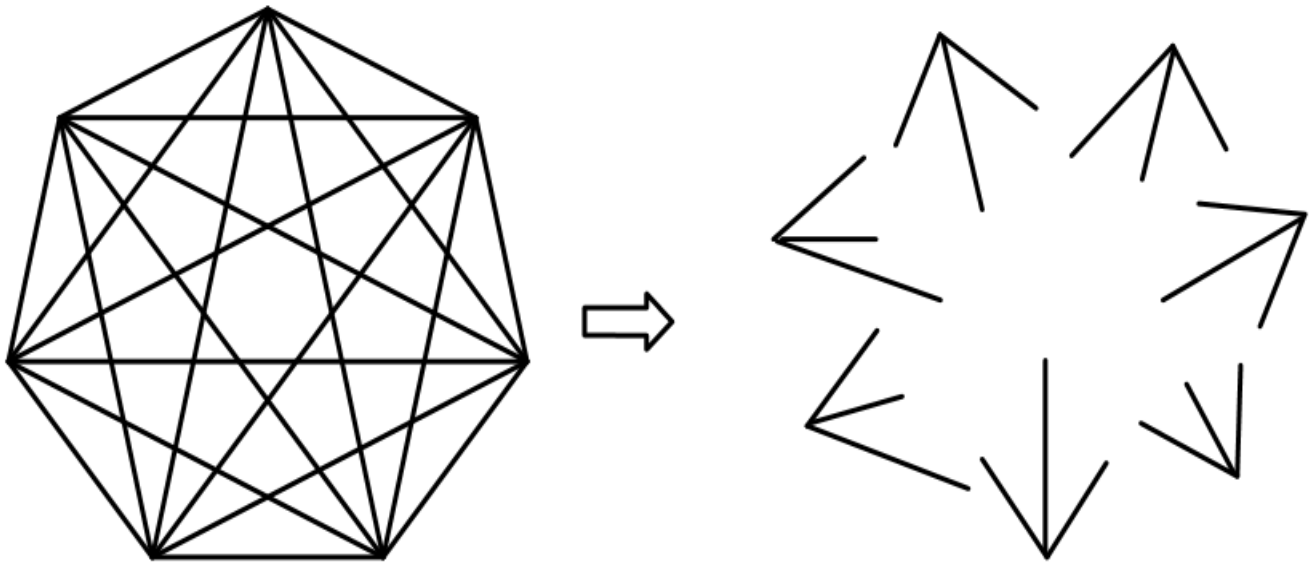

Figure 1.2: K_7 being decomposed into 7 isomorphic trees of size 3

in graph labeling. Indeed even for problems as narrowly focused as the ones involving the special classes of graphs, the labelings have been hard-won and involve a large number of cases. Finding a graph that possesses an α-labeling is another common approaches in many papers. The following condition (due to Rosa) is known to be necessary and in the case of cycles also sufficient for a 2-regular graph $G = (V, E)$ to have an α-labeling.
$\mid E(G) \mid \equiv 0 \pmod 4$. In 1982, Kotzig conjectured that this condition is also sufficient for a 2-regular graph with two components. In 1996, Abrham and Kotzig have shown that this conjecture is valid.

1.2 Approaches

As mentioned earlier that the topic about graceful labeling of graphs is an open problem, we found interest in demonstrating newer classes of graphs that can be gracefully labeled. Our research helped us to gain the knowledge about the prevailing classes of graceful graphs and trees and the general rule defined to label them. In order to make the topic understandable, our paper mentions a few examples of the prevailing classes of graceful trees and graceful graphs. The mentioned portion was the result of our studies in the related field. After studying through the prevailing data, we focused on finding out any newer classes that can be gracefully labeled. We found success when we brought out three such classes that will be discussed in this paper. All these three classes have a generalized rule for labeling. Algorithms have been developed based on these rules, which are also included. Moreover, some conditional categories of a particular class of graphs were also labeled gracefully. A general rule was not implied since the total class could not be labeled gracefully, yet. Further research will be conducted to prove this particular category is also graceful. This paper will provide an elaborate discussion about the topics mentioned.

CHAPTER 2
PRELIMINARIES

2.1 Basic Definition

A graph $G = (V, E)$ consists of two finite sets: $V(G)$, the vertex set of the graph, often denoted by just V, which is a nonempty set of elements called vertices, and $E(G)$, the edge set of the graph, often denoted by just E, which is a set of elements called edges. A connected acyclic graph is called tree.

2.1.1 Degree

The degree of a vertex in a graph is the number of vertices adjacent to it [3]. For example degree of each vertices in figure 2.1 is 5.

2.1.2 Adjacency

Two vertices are adjacent if they share an edge. [3]

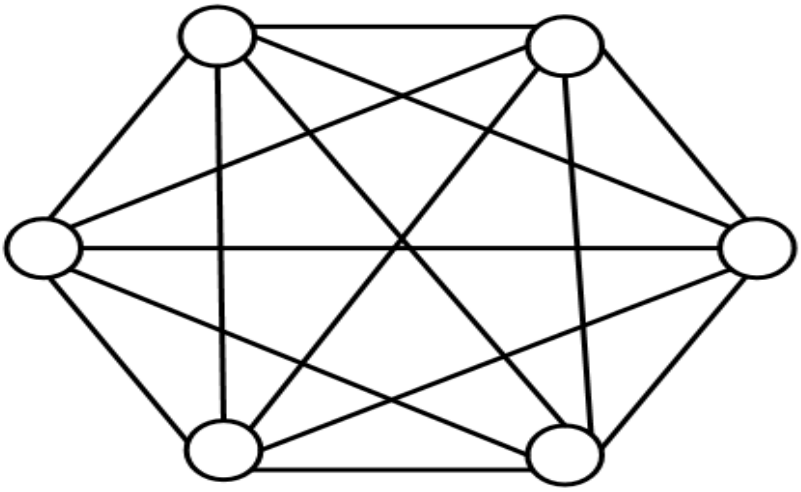

Figure 2.1: A complete graph K_6 with 6 vertices and 15 edges

2.1.3 Diameter

To find the diameter [3] of a graph, first find the distance between every pair of vertices. The diameter is the maximum of these minima.The diameter of a tree is much simpler; as there is always one and only one path of edges between any two vertices, there is only one possible distance. The diameter of the graph in Figure 2.1 is 1.

2.1.4 Complete Graph

A complete graph(see Figure 2.1) has one edge from every vertex to every other vertex. The complete graph with n vertices is represented by the symbol K_n and will have edges. [3]

2.1.5 Node

A node is another name for a vertex. Within this thesis node and vertex are used interchangeably.

2.1.6 Node, Mirrored

All of the work in this thesis is related to trees with labeled nodes. Mirror Nodes are a concept created to describe sets of nodes that do not need to have all possible labelings tested because they are part of identical structures. For example the three trees in Figure 2.2 are structurally identical. This means that any algorithm that looked at all three would be

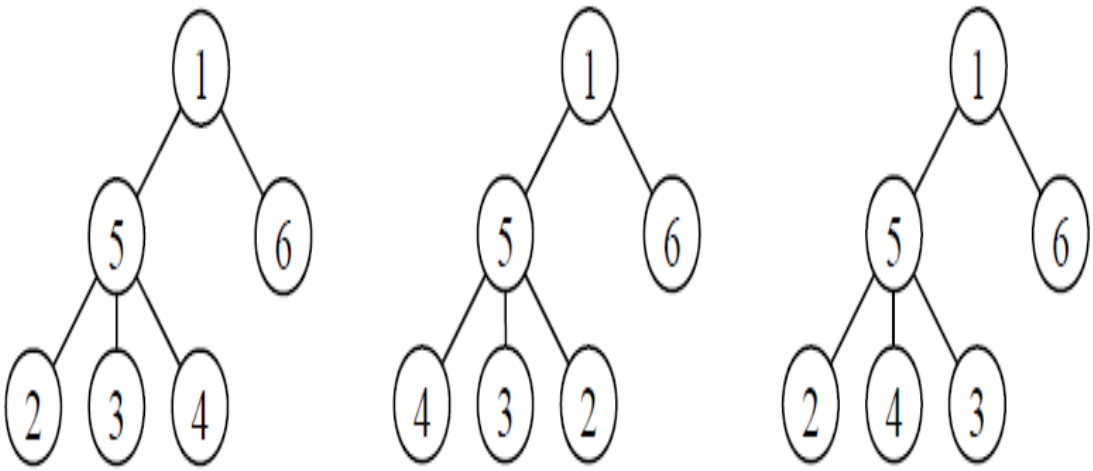

Figure 2.2: Example of mirrored nodes the 2, 3 and 4 can be rearranged at will.

wasting time on the second two.
Two nodes n_1 and n_2 are considered mirrored if:
1. The rooted tree with n_1 as the root is isomorphic to the rooted tree with n_2 as the root
2. n_1 and n_2 are adjacent or are adjacent to a common node.

2.1.7 Isomorphic

Two graphs are isomorphic if they have the same structure if, by rearranging their vertex identification, they can be shown to share the same set of edges. [3]

2.1.8 Bipartite Graph

A bipartite graph, also called a bigraph, is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent.

2.1.9 Labeling

A valuation O_G, is a one-to-one mapping of the vertex set of G into the set of non-negative integers. Let s be an edge between vertices v_i and v_j and let a_i and a_j be the labels of v_i and v_j , respectively. Then $b_s =\mid a_i a_j \mid$ is referred to as an induced label of edge s. Let V_{O_G} be the set of vertex labels and H_{O_G} be the set of induced edge labels in a valuation O_G of the graph G. Rosa [1] defines the following conditions on a valuation O_G of a graph G.
(a) $V_{O_G} \subset \{0, 1, \ldots, n\}$;

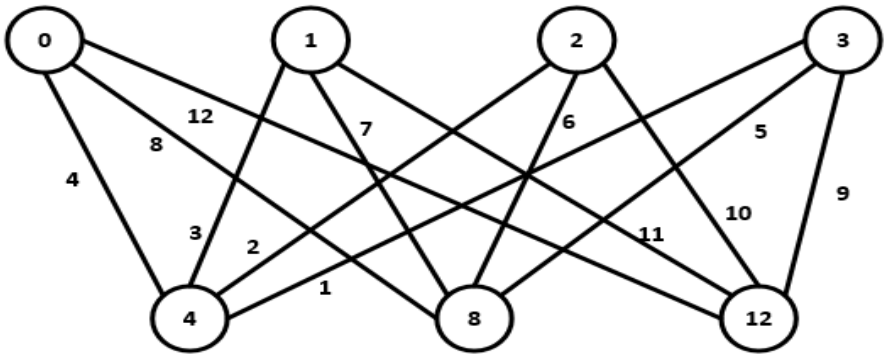

Figure 2.3: Example α valuation.

(b) $V_{O_G} \subset \{0, 1, \ldots, 2n\}$;
(c) $H_{O_G} \equiv \{1, 2, \ldots, n\}$;
(d) $H_{O_G} \equiv \{x_1, x_2, \ldots, x_n\}$, where $x_i = i$ or $x_i = 2n + 1i$,
(e) there exists $x, x \in \{0, 1, \ldots, n\}$, such that for an arbitrary edge (v_i, v_j) of the graph G either $a_i \leq x < a_j$ or $a_j \leq x < a_i$ holds.
A valuation satisfying the conditions
(a), (c), (e) is called an α-valuation,
(a), (c) is called a β-valuation,
(b), (c) is called a σ-valuation,
(b), (d) is called a ρ-valuation.
In 1976 Sheppard [4] used the term balanced labeling to denote α-labeling. Also, β- valuation has been called graceful numbering by Golomb [2] in 1972, and proper labelling by Sheppard [4] in 1976 However, it is most commonly called graceful labeling, and in this paper we use the terms α-valuation and graceful labeling interchangeably.

2.2 Previous Works

The concept of a graceful labeling has been introduced by Rosa [1] as a means of attacking the famous conjecture of Ringel that K_{2n+1} can be decomposed into $2n + 1$ subgraphs that are all isomorphic to a given tree with n edges. Rosa proved that if G is graceful and if all vertices of G are of even degrees, then $\mid E(G) \mid \equiv 0$ or $3 (\text{mod } 4)$ and named β valuation. Although most graphs are not graceful, graphs that have some sort of regularity of structure are graceful. Many variations of graceful labeling have been introduced in recent years

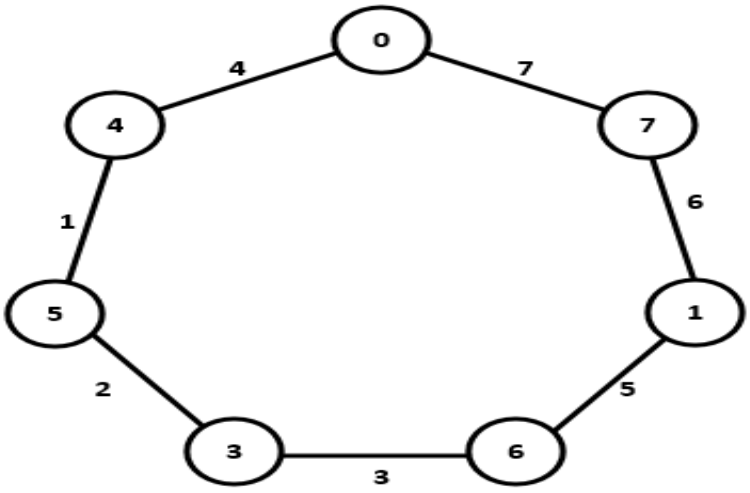

Figure 2.4: Example of β-valuation.

by researchers. All cycles C_n are graceful if and only if $n = 0$ or $3(\text{mod } 4)$. All snakes P_n, wheels W_n, helms H_n, crowns R_n, and complete bipartite graphs $K_{m,n}$ are graceful. The complete graphs K_n are graceful only if $n \leq 4$. It has been conjectured that all trees are graceful. Although this conjecture has been the focus of more that 200 papers, it is still an open problem. It has been shown that trees with atmost 27 vertices are graceful. Although more than 400 papers have been published on the subject of graph labeling,there are few particular techniques to be used by authors. The graceful labeling problem is to find out whether a given graph is graceful, and if it is graceful, how to label the vertices. The common approach in proving the gracefulness of special classes of graphs is to either provide formulas for gracefully labeling the given graph, or construct desired labeling from combining the famous classes of graceful graphs. Unfortunately, the process of gracefully labeling a particular graph G is a very tedious and difficult task for many classes of graphs.

CHAPTER 3
PREVAILING CLASSES OF GRACEFUL TREES

3.1 Chain or Path

A graph is called a path (see figure 3.1) if the degree of every vertex is ≤ 2 and there are no more than 2 endvertices [2]. An endvertex or leaf is a vertex of degree 1.Therefore a chain or path is a tree with only two leaves, or equivalently a tree in which all vertices have degree 0 or 1. Chain or path has following characteristics:

1. A chain or path, is the simplest type of tree: a single line of vertices.
2. A chain is a caterpillar (see figure 3.1) with no legs.

Theorem 1. Every path is graceful [1].

Proof. We demonstrate an algorithm to gracefully label any path P_n with n vertices. In a path the number of edges is one less that the number of vertices or $m = n - 1$. Labeling can begin at either end without loss of generality. The first vertex at one end is labeled 0,

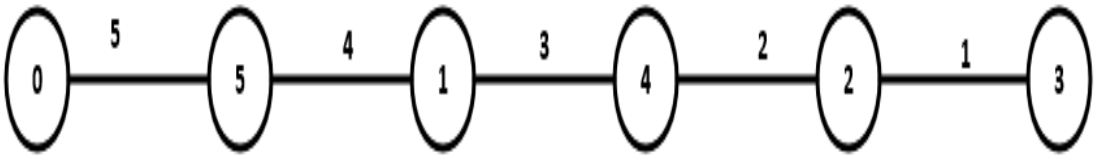

Figure 3.1: Gracefully labeled Path

the adjacent vertex is labeled $n-1$, the next adjacent, nonlabeled vertex is labeled 1, and we continue in this manner. Alternate vertices are incremently decreasing by 1. consider two cases where $n = 0(mod2)$. In both cases $k = \lfloor n/2 \rfloor$. For even case the edge labels beginning with the leftmost edge in figure are $\mid (n-1)-0 \mid, \mid (n-1)-1 \mid, \mid (n-2)-1 \mid, \ldots, \mid (n-k)-(k-1) \mid = n-1, n-2, n-3, \ldots, 1$. In detremining the last edge value recall $k = \lfloor n/2 \rfloor$. For cases where n is even, $k = \lfloor n/2 \rfloor = n/2$ and $(n-k)-(k-1) = n-k-k+1 = 1$. It is easy to see this is a graceful labeling since all numbers between 1 and $n-1$ or m are used in the edge labels. Similarly, when n is odd the edge values beginning on the left are $n-1, n-2, n-3, \ldots, 1$. In evaluating the right most edge value, recall $k = \lfloor n/2 \rfloor = n-1/2$ when n is odd. Then $n-k-k = n-n-1/2-n-1/2 = 1$. Again every value from 1 to $n-1$ or m is used. Rosa [1] also introduced α graceful labeling, which is a stronger standard, and therefore fewer graphs are α graceful. Graphs that are α graceful are also graceful. A graceful graph G is said to be α graceful if there exists a critical value α such that for every edge (u, v), either $f(u) \leq \alpha < f(v)$ or $f(v) \leq \alpha < f(u)$. In each α graceful graph, α is a positive integer and the vertices are said to have an α valuation. These α graceful graph must be bipartite, which implies that no α graceful graphs can have an odd cycle [5]. An example of α graceful labeling is shown in figure 3.1. Examine the vertex labels in the path shown in figure 3.1. Begining at the left end of the path the first vertex is labeled 0 and alternate vertices to the right increase by 1. Call these vertex labels {0,1,2,3,4,5} set A.

3.2 Caterpillar

A caterpillar $C = (XUY, E)$ is a tree consisting of a path $P(C)$ with vertex set X and vertices Y not on $P(C)$, each joined to exactly one vertex of $P(C)$. So a caterpillar is a tree with one long path or chain of vertices and any number of paths of length 1 attached to the chain. The long chain is known as spine of the caterpillar. Original results on the gracefulness of caterpillars are due to Rosa [1]. Let the base of a tree T be the tree obtained by deleting the leaves of T and any edges incident with these vertices. In 1967, Rosa showed that ant tree that is a path or has a path as its base is graceful by constructing a bipartite labeling of such trees. Since a caterpillar is transformed into a path by the deletion of its leaves, the gracefulness of caterpillars follows by Rosa's construction. Figure 3.2 shows a α gracefully labeled caterpillar. The first vertex on the spine is labeled 0 and the adjacent vertices are labeled $\{24, 23, 22, 21, 20\}$ using the higher values on the leaf neighbors, so that 20 is the label for the next vertex on the spine and there are 4 leaf neighbors of vertex 0. Now the remaining non-labeled neighbors of 20 are labeled $\{1, 2, 3, 4\}$ with the lower values on the leaf neighbors and the value 4 on the next vertex on the spine. Continuing in this manner the caterpillar of 25 vertices is labeled. The edge values are $\{1, 2, 3, \ldots, 24\}$ and $m = 24$,

while the vertex values are from the set $\{0, 1, 2, \ldots, 24\}$.

Theorem 2. All caterpillars are α graceful [6].

Proof. We demonstrate an algorithm to α gracefully label any caterpillar. Let v_i represent the vertices on the spine of the caterpillar and $N(v_i)$ is the set of all vertices adjacent to v_i, while $\mid N(v_i) \mid$ is the number of vertices which are adjacent to v_i.

$$k_i = \begin{cases} |N(v_i)| - 1, \text{if i is the first or last vertex of the spine} \\ |N(v_i)| - 2, \text{otherwise} \end{cases}$$

1. Begin by labeling the first vertex on the spine 0 (figure 3.2). Alternate vertices of this spine beginning with the first vertex are odd. all other vertices on the spine are called even.
2. Leaf neighbors of 0 are labeled begining with $n-1$ and going in descending order ending with $n - k_1$ where $k_1 \equiv N(v_0) - 1$.
3. The next vertex on the spine, a neighbor of 0, is labeled $n - (k_1 + 1)$.
4. Leaf neighbors of this vertex $n - (k_1 + 1)$ are labeled 1 through k_2.
5. Continue in this manner. The odd vertices on the spine and the leaf neighbors of the even

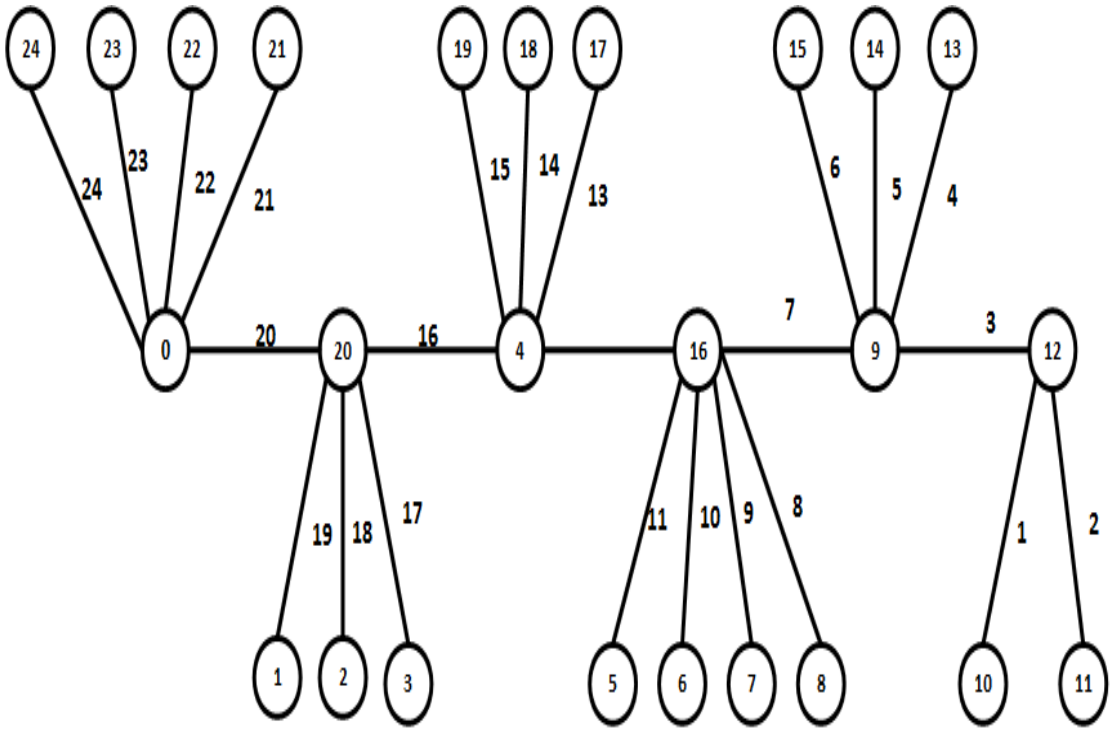

Figure 3.2: Gracefully labeled Caterpillar.

vertices on the spine are labeled in increasing order, while the leaf neighbors of the odd vertices on the spine are labeled in decreasing order. the last vertex on the spine is labeled $n-(k_1+k_3+\cdots+k_{i-3}+\lfloor i/2 \rfloor)$ if it is even. It is $n-(k_2+k_4+\cdots+k_{i-2}+\lfloor i-1/2 \rfloor)$ if odd. The last neighbor of this vertex is labeled $n-(k_2+k_4+\cdots+k_{i-2}+k_i+\lfloor i-1/2 \rfloor)$ or $n-(k_1+k_3+\cdots+k_{i-2}+k_i+\lfloor i/2 \rfloor)$ respectively.
6. $\alpha = min\{x, y\}$ where $x =$ the label of last vertex on the spine and,

$$y = \begin{cases} \text{minlabels of leaf neighbours, if x } < \text{labels of leaf neighbors} \\ \text{maxlabels of leaf neighbors, if x } > \text{labels of leaf neighbors} \end{cases}$$

3.3 Star

A star S_k(see Figure 3.3) is the complete bipartite graph $K_{1,k}$, a tree with one internal node and k leaves (but, no internal nodes and k + 1 leaves when $k \leq 1$). Alternatively, some

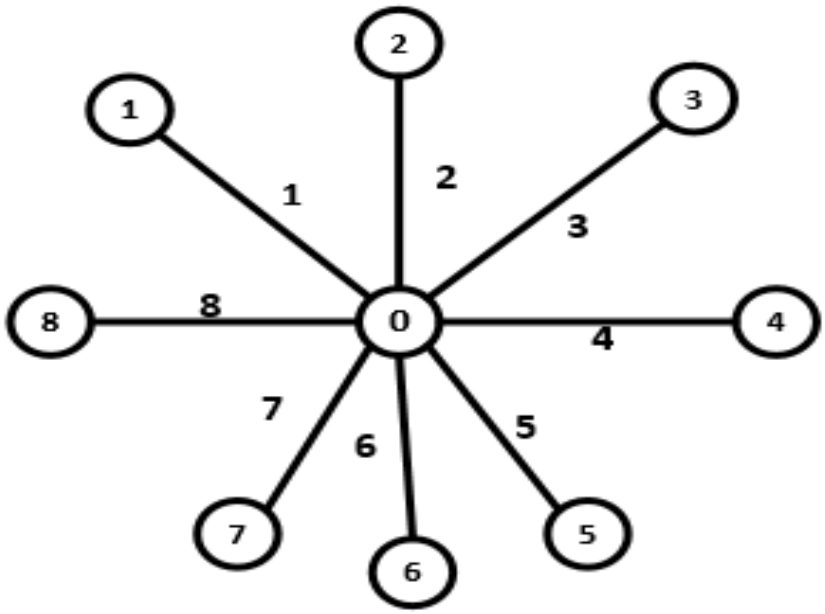

Figure 3.3: A Star.

authors define S_k to be the tree of order k with maximum diameter 2; in which case a star of k>2 has k 1 leaves.

3.4 Symmetrical Tree

A symmetrical tree (see Figure 3.4) is a rooted tree in which every level contains vertices of the same degree.It has been proved that all symmetrical trees are graceful. [7]

3.5 m-Stars

An m-Star (see Figure 3.5) has a single root node with any number of paths of length m attached to it. Cahit and Cahit also proved that all m-Stars a graceful. [8]

3.6 Olive Tree

An olive tree has a root node with k branches attached; the ith branch has length I. Pastel and Raynaud proved that all olive trees are graceful [9].
Figure3.6 is a sample figure.

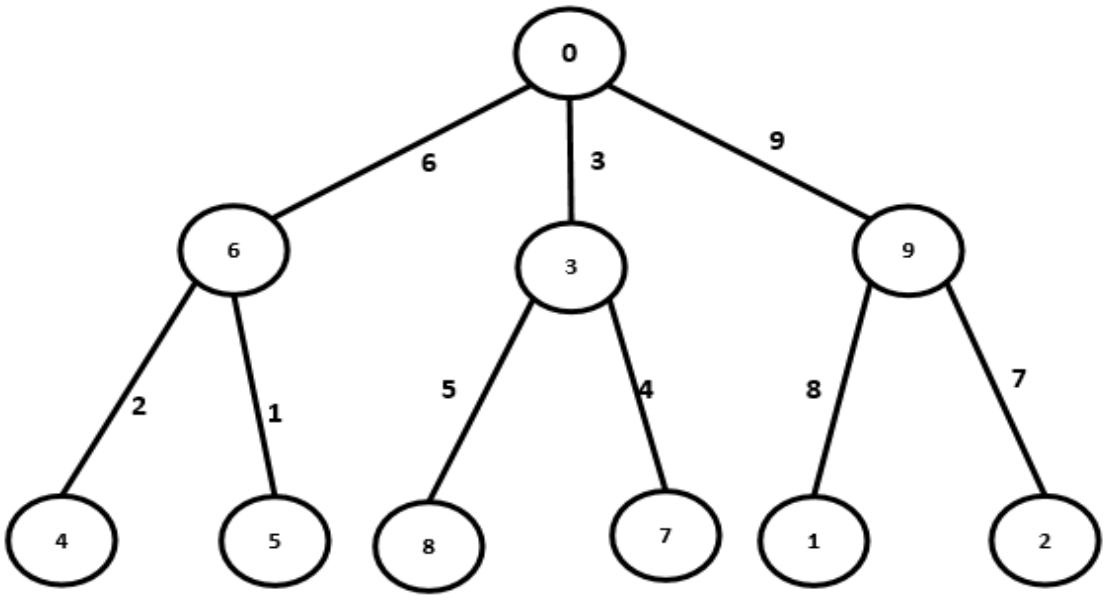

Figure 3.4: Gracefully labeled Symmetrical Tree.

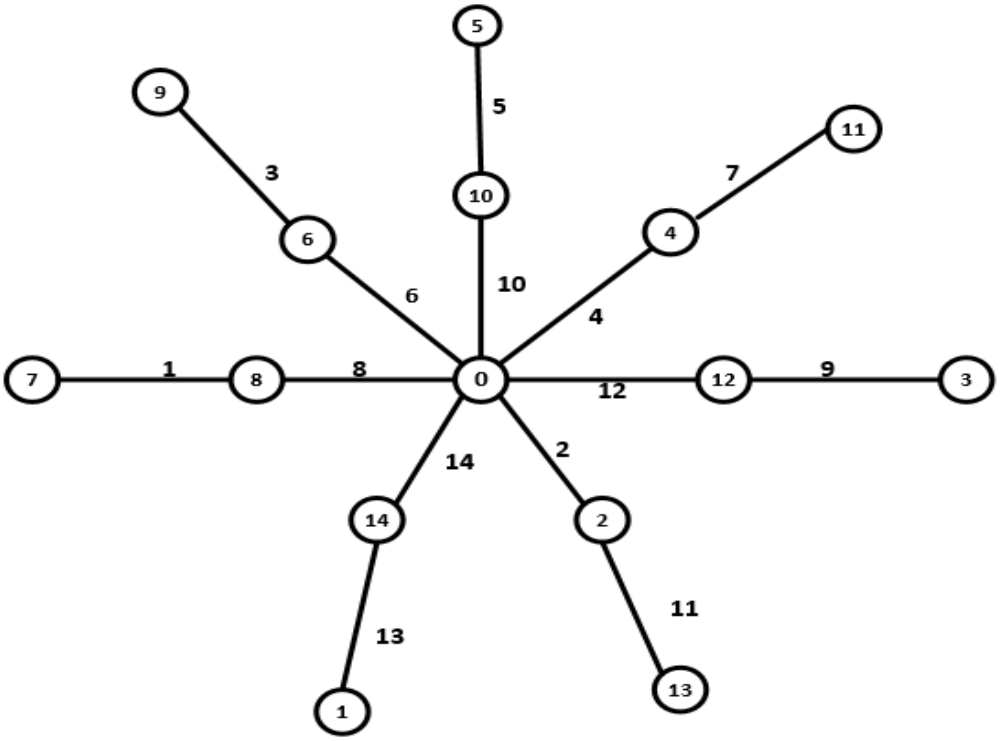

Figure 3.5: Gracefully labeled 2 star.

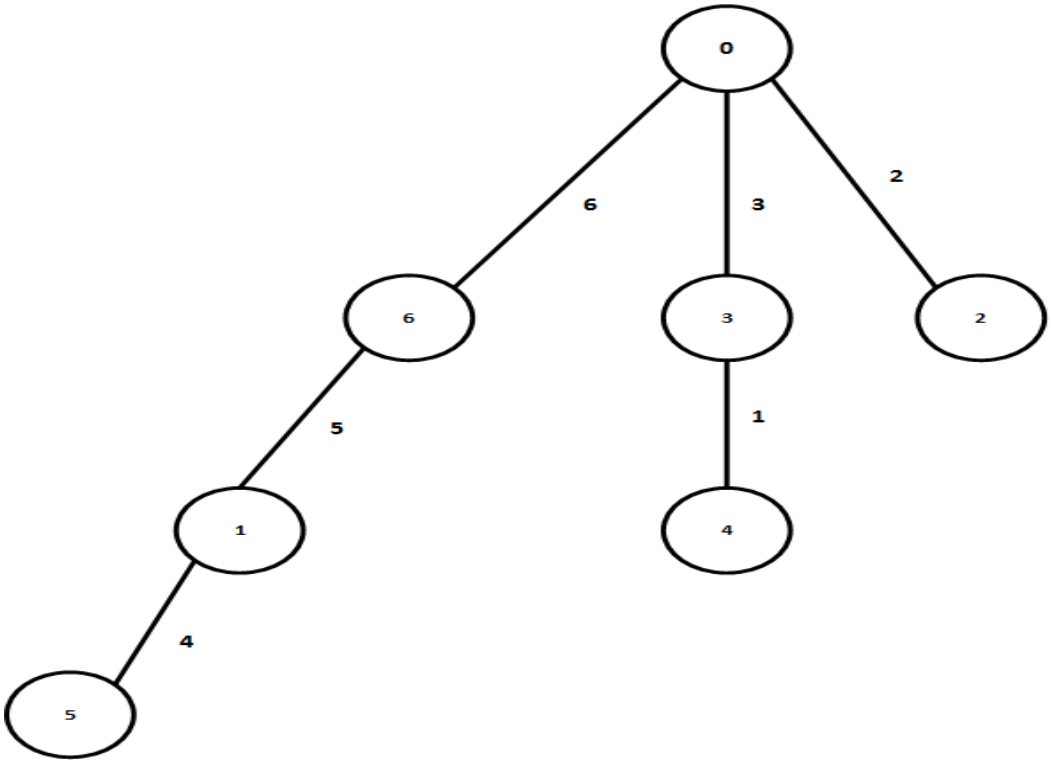

Figure 3.6: Gracefully labeled Olive.

3.7 Banana Tree

A banana tree (see Figure 3.7) is constructed by bringing multiple stars together at a single vertex.Banana trees have not been proved graceful, although Bhat-Nayak and Deshmukh have proven the gracefulness of certain classes of banana tree. [10]

3.8 Tp Tree

Hegde and Shetty defined a class of tree called Tp-trees (transformed trees) created by taking a gracefully labelled chain and shifting some of the edges (Figure 3.8), and proved that they can always be gracefully labelled using the original chain labels. [11]

3.9 Product Tree

Some proofs also show that certain graceful trees can be added together to give a larger graceful tree. Koh et al. [12] showed how rooted product trees are always graceful(see Figure 3.9).

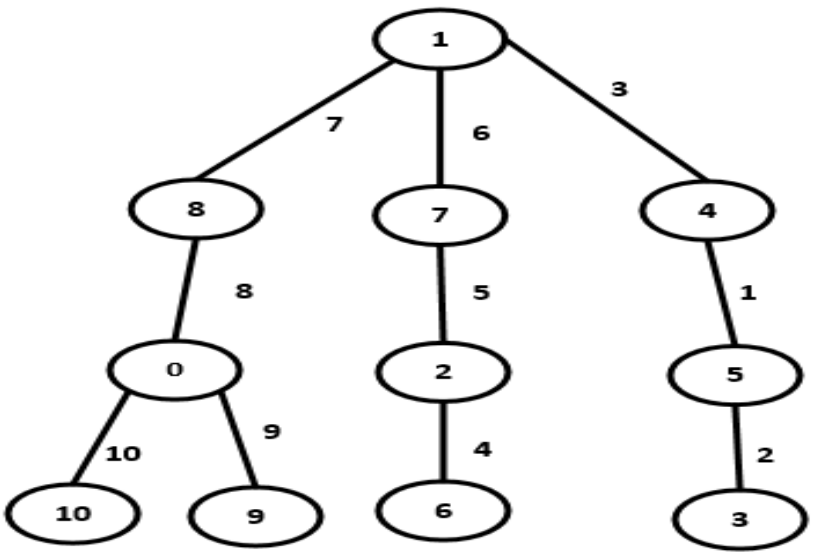

Figure 3.7: Gracefully labeled Banana tree.

3.10 Lobstar

A lobster (see Figure 3.10) is a tree having a path from which every vertex has distance at most two.Morgan has proved that all lobsters with perfect matching are graceful. A small class of interlaced lobsters is constructed by joining symmetrical trees (rooted trees in which every level contains vertices of the same degree) successively at the roots to form a path containing these roots.Graceful lobsters with vertices on the central path attached to combination(s) containing all three types of branches preceded by the vertices attached to combination(s) containing two types of branches. Some lobsters satisfy this property with some restrictions on the number of odd, even, and pendant branches. [13]

3.11 Firecracker

A firecracker F (see Figure 3.11) is a tree consisting of a path $P(F)$ and a collection of stars, where each vertex on is joined to the central vertex of exactly one star. All firecrackers are graceful. [13]

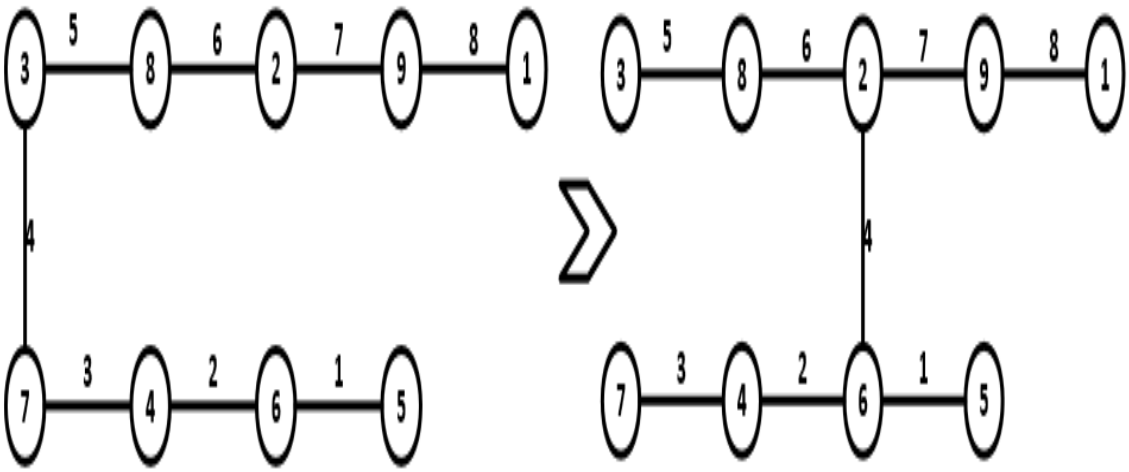

Figure 3.8: Gracefully labeled Tp tree.

3.12 Spraying Pipe

A spraying pipe [13] is a path $V_1, ..., V_n$ where each vertex V_i is joined to Mi paths at a leaf of each path, and where all paths have fixed length. A spraying pipe is interlaced if n is even

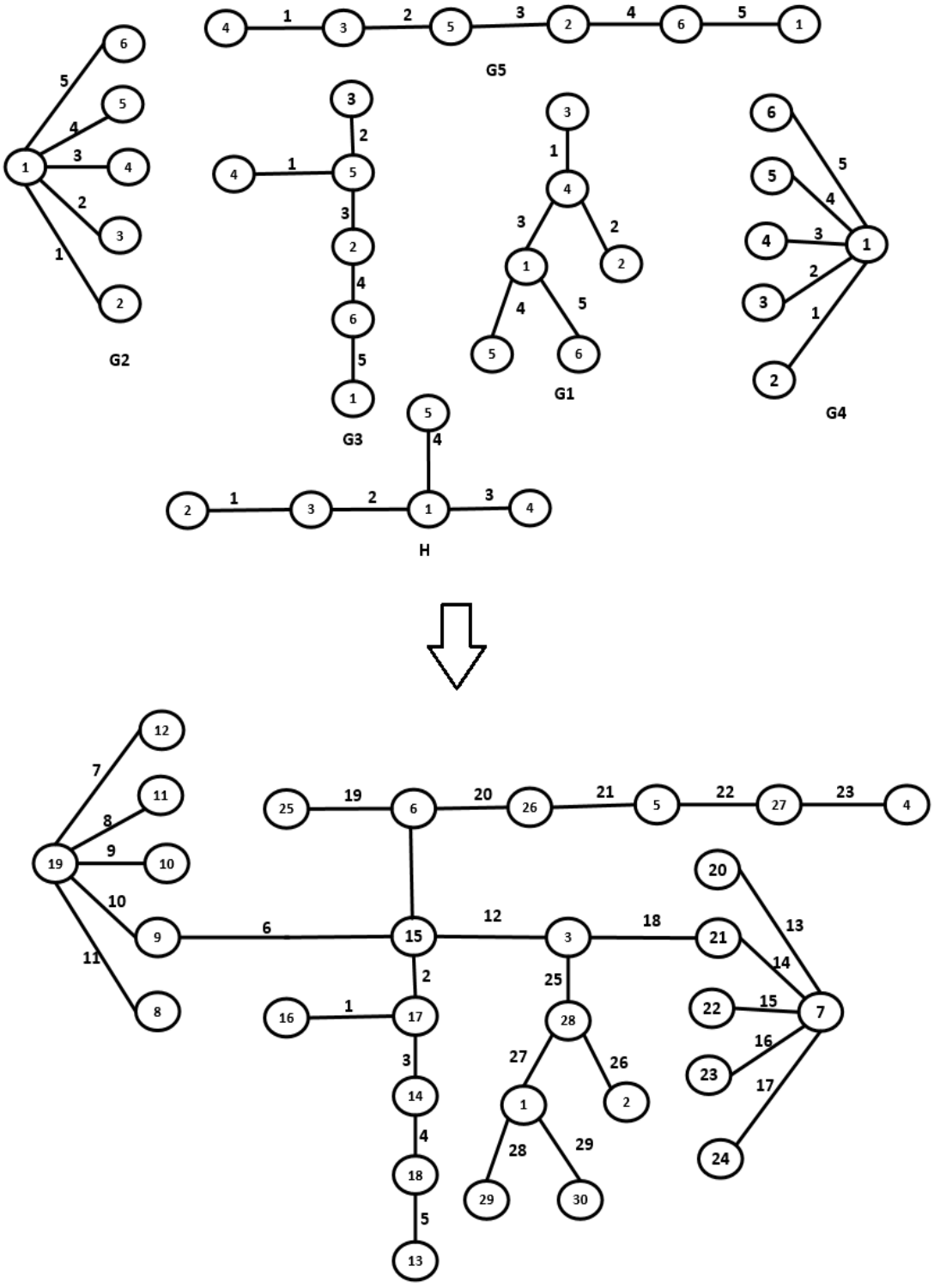

Figure 3.9: Gracefully labeled Product Tree.

and $m_{2i1} = m_{2i}$ for every $1 \leq i \leq n/2$.

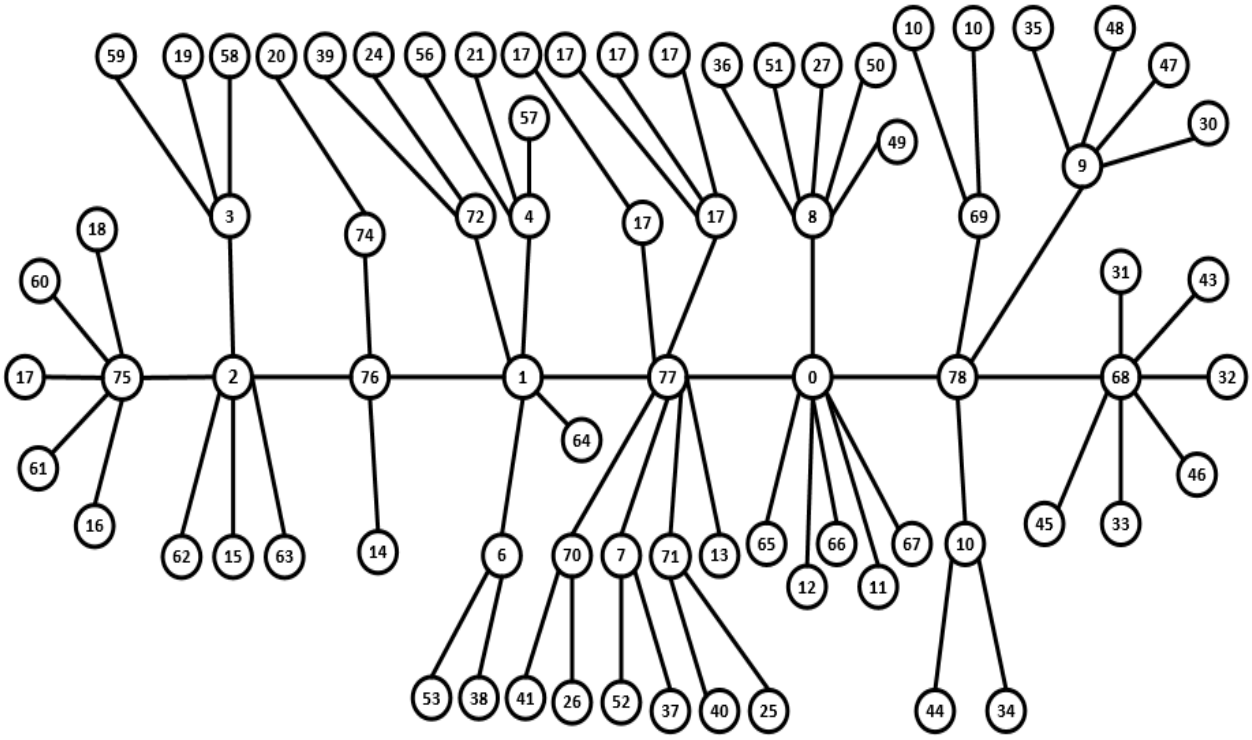

Figure 3.10: Gracefully labeled Lobstar.

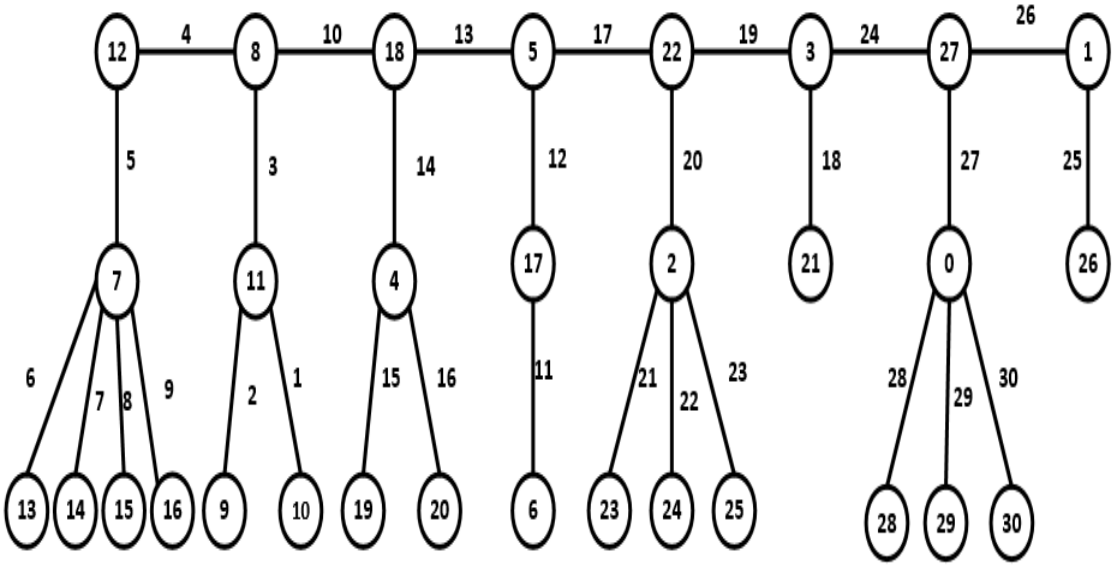

Figure 3.11: Gracefully labeled Firecracker.

CHAPTER 4
PREVAILING CLASSES OF GRACEFUL GRAPHS

The following necessary condition for gracefulness of a graph $G = (V, E)$ with $m = | V(G) |$ and $n = | E(G) |$ comes directly from the definition 2.1.7 :

Lemma 1. If G is a graceful graph then $mn + 1$. [1]

It is clear that the above lemma is satisfied for every connected graph. Using this condition we can rule out the existence of a graceful labeling for some disconnected graphs, for instance, 1-regular graphs with $n > 1$.

A connected graph G is called $Eulerian$ if $n > 0$ and the degree of every vertex of G is even. A necessary condition for the existence of a graceful labeling of an $Eulerian$ graph G is proved by Rosa [1].

4.1 Eulerian Graph

Theorem 3. If G is a graceful $Eulerian$ graph then $n \equiv 0$ or 3 (mod 4). [1]

In this theorem, an $Eulerian$ graph is any graph in which the degree of each vertex is even; it does not have to be connected. For example, K_6 in Figure 4.1 are Eulerian, but it has 10

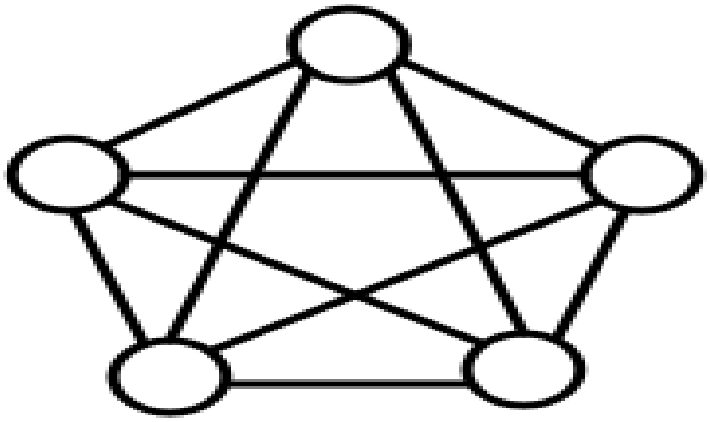

Figure 4.1: A complete graph K_5

and thus by the above theorem it is not graceful.
A generalization of Rosas theorem for $k-$graceful $Eulerian$ graphs is as follows.
Theorem 4. If an $Eulerian$ graph $G = (V, E)$ is k-graceful then either $n \equiv 0(mod4)$ or $n \equiv 1$ (mod 4) when k is even or $n \equiv 3$ (mod 4) when k is odd. [14]

4.2 Cycle

For cycle C_n, the necessary condition in theorem is also sufficient.
Theorem 5. The cycle C_n is graceful if and only if $n \equiv 0$ or 3 (mod 4). [1]
Rosa also proved the following result:
Theorem 6. The cycle C_n has an α-labeling if and only if $n \equiv 0$ (mod 4). [1]
Maheo and Thuillier [15] have generalized this result as follow.
Theorem 7. The cycle C_n is k-graceful if and only if either $n \equiv 0$ (mod 4) or $n \equiv 1$(mod 4) where k is even and $k \leq (n-1)/2$ or $n \equiv 3$(mod 4) where k is odd and $k \leq (n-1)/2$. [15]
We also know that.
Theorem 8. The cycle C_n is 1-sequential. [16]

4.3 Wheel

According to theorem 6 and the connection between 1-sequential and graceful graphs, we can conclude that all wheels are graceful:
Theorem 9. The wheel W_n is graceful for all $n \geq 3$. [17]
The following theorem and conjecture are due to Maheo and Thuillier.
Theorem 10. W_{2k+1} is k-graceful for any $k \geq 1$. [15]
Conjecture 2. W_{2k} is k-graceful with $k \neq 3, 4$. [5]

4.4 Crown

A crown R_n is formed by adding to the n points $v_1, v_2, v_3, \ldots, v_n$ of a cycle C_n, n more pendant points $u_1, u_2, u_3, \ldots, u_n$ and n more lines $(u_i, v_i), i = 1, 2, 3, \ldots, n$ for $n \geq 3$. Frucht has proved the following theorem.
Theorem 11. R_{2n} is graceful for any $n \geq 3$. [5]
We know that a graph admitting an α-labeling must be bipartite and, as such, can not contain cycles of odd length. It follows that wheels can not have an α-labeling since they contain triangles as subgraphs. For analogous reason, crowns can not have α-labeling if n is odd. For even values of n, Frucht has offered the following conjecture.

Conjecture 3. If $n \equiv 0(\text{mod } 2)$ then R_n has an α-labeling. [5]
In Figure 4.2, we can see a graceful labeling for R_5 and in Figure 4.3 an α-labeling for R_6.

4.5 Helm

A helm H_n, $n \geq 3$, is the graph obtained from a crown R_n by adding a new vertex joined to every vertex of the unique cycle of the crown. Ayel and Favaron proved that.
Theorem 12. The helm H_n is graceful for every $n \geq 3$. [18]

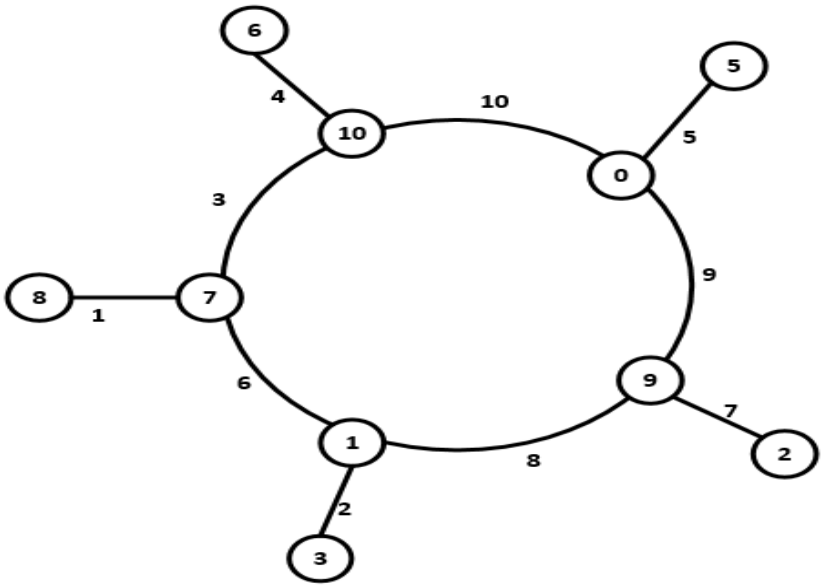

Figure 4.2: Graceful Labeling of R_5.

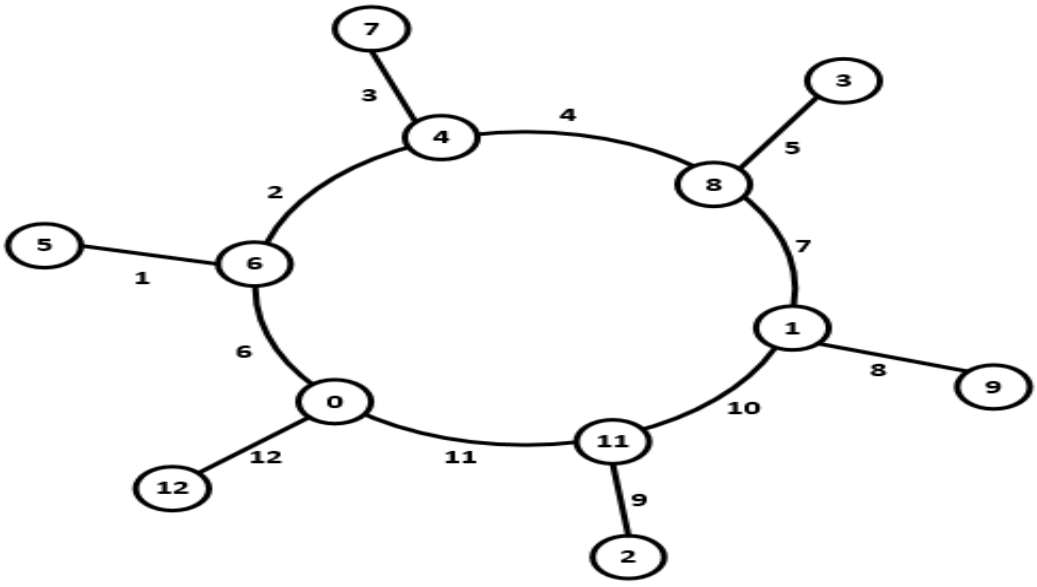

Figure 4.3: α Labeling of R_6.

For example, Figure 4.4 shows that H_5 is graceful.

4.6 Chord

A chord of a cycle is an edge joining two otherwise non adjacent vertices of a cycle. Bondendiek conjectured that any cycle with a chord is graceful. This conjecture has been proved by Delorme et al.

Theorem 13. The graph consisting of a cycle plus a chord is graceful. [19]

Let P_k be a path with k edges and $k+1$ vertices. Koh and Yap defined a cycle with a P_k-chord as a cycle with a path P_k joining two nonconsecutive vertices of the cycle. They proved that these graphs are graceful when $k = 2$. Thereafter Punnim and Pabhapate proved the general case $k \geq 3$.

Theorem 14. A cycle with a P_k-chord is graceful for all $k \geq 1$. [20]

In 1990, Zhi-Zheng generalized the above theorem by proving the following result.

Theorem 15. Apart from four exceptional cases, simple graphs consisting of three independent paths joining two vertices are graceful. [21]

Examples of graceful labeling of cycles with a P_1-chord and P_3-chord can be seen in Figure 4.6. Koh et al. also introduced the concept of a cycle with k-consecutive chords. A cycle with k-consecutive chords is a graph formed from a cycle by joining a cycle vertex v to k consecutive vertices of the cycle in such a way that v is not adjacent to any of these. Koh and others proved the following result about this kind of graph.

Theorem 16. A cycle C_n with k-consecutive chords is graceful if $k = 2, 3, \ldots, n-3$. [20]

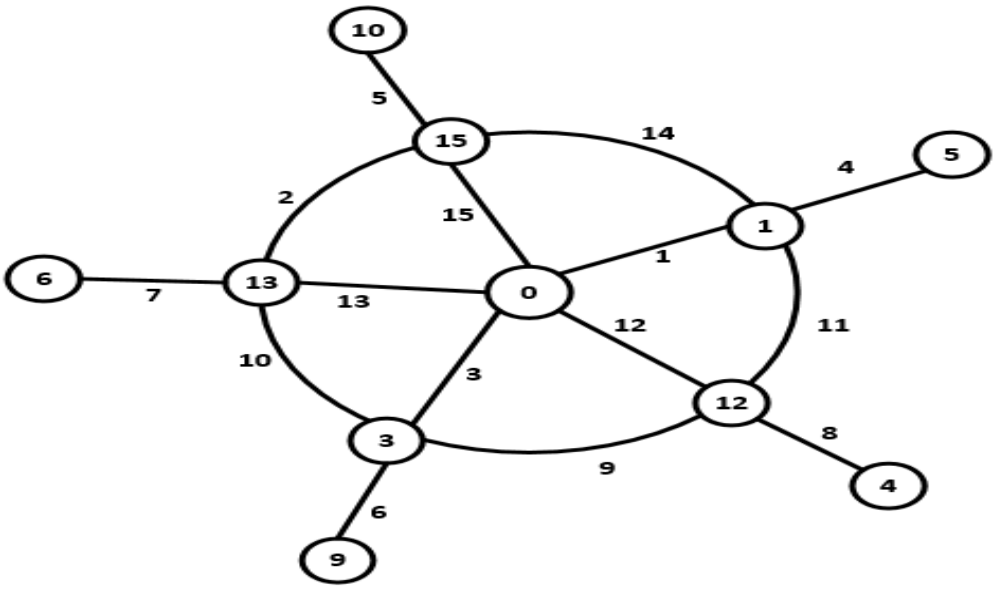

Figure 4.4: Graceful Labeling of H_5.

4.7 Dragon

A dragon $D_n(m)$ is a graph obtained by joining the end point of path P_m to the cycle C_n. Truszcynski has proved the following theorem related to dragons.

Theorem 17. The dragon $D_n(m)$ is graceful for $n \geq 3, m \geq 1$. [21]

The following conjecture is also due to Truszcynski.

Conjecture 3. All graphs with a unique cycle are graceful except $C_n, n \equiv 1, 2 \pmod 4$. [21]

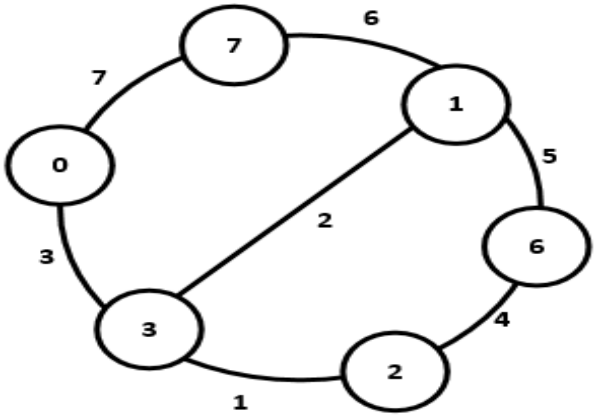

Figure 4.5: Graceful Labeling of C_6.

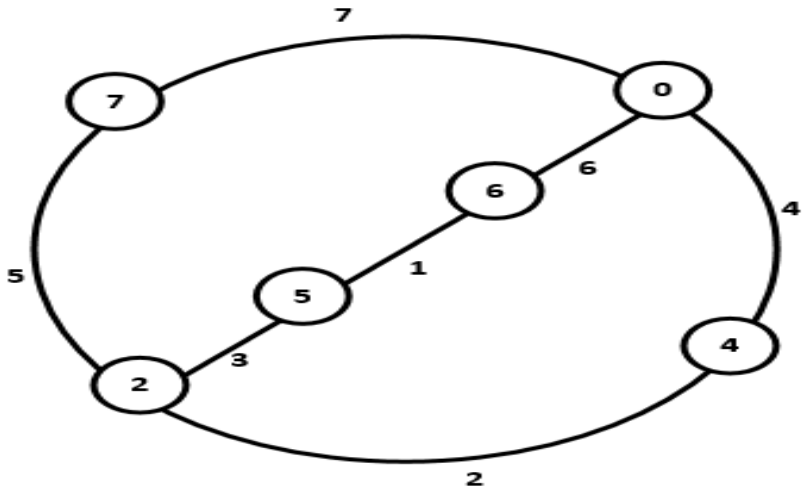

Figure 4.6: Graceful Labeling of C_4.

4.8 Triangular Snake

Rosa [1] has defined a triangular snake (or Δ-snake) as a connected graph in which all blocks are triangles and the block-cut-point graph is a path. Let Δn-snake be a Δ snake with n blocks. Since a Δn-snake is an $Eulerian$ graph, according to theorem 1 it can only be graceful if $3_n \equiv 0$ or 3 (mod 4) $\Rightarrow n \equiv 0$ or 1 (mod 4). Moulton verified that this result is also sufficient.

Theorem 18. Every Δn-snake is graceful if and only for $n \equiv 0$ or 1 (mod 4). [22]

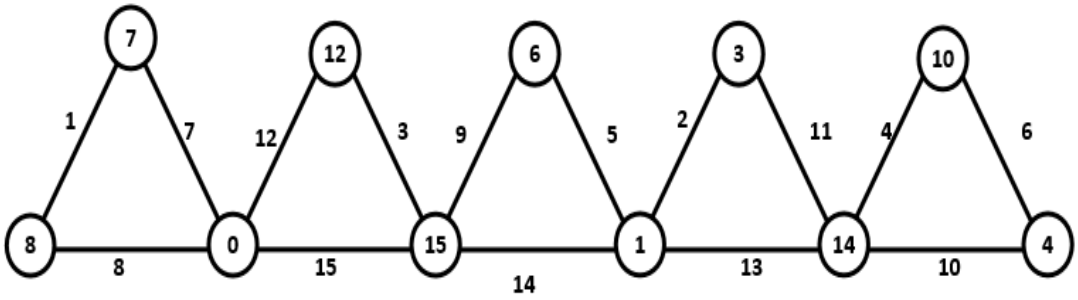

Figure 4.7: Graceful Labeling of $\Delta 5$ snake.

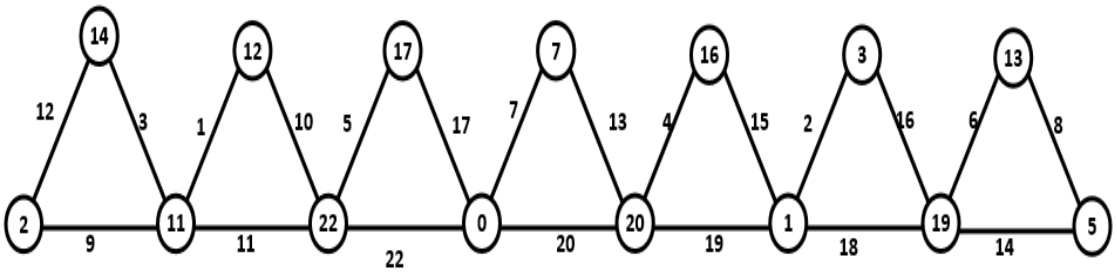

Figure 4.8: Graceful Labeling of $\Delta 7$ snake.

In order to deal with other cases, Moulton also defined a new concept as follows.

Definition. An almost graceful labeling of a graph $G = (V, E)$ with $n =\mid E(G) \mid$ and $m =\mid V(G) \mid$ is a one-to-one mapping f of the vertex set $V(G)$ into the set $\{0, 1, 2, \ldots, n-1\} \cup \{n$ or $n + 1\}$ such that the set of edge labels induced by the absolute value of the difference of the labels of the adjacent vertices is $\{1, 2, 3, \ldots, n - 1\} \cup \{n$ or $n + 1\}$.

Notice that the above definition includes graceful labeling as special case. Next Moulton has strengthened the theorem 16 as follows.

Theorem 19. Every Δn-snake for $n \equiv 2$ or 3 (mod 4) is almost graceful. [22]

The graceful labeling of $\Delta 5$-snake and an almost graceful labeling of $\Delta 7$-snake are shown in Figure 4.7 and Figure 4.8 repectively.

CHAPTER 5
NEW APPROACHES ON GRACEFUL GRAPHS

5.1 Linear Dice

This type of graph consists of C_4 graphs joined with a single vertex. Each C_4 is termed as a dice. Among two consecutive dices, there is a common node. The procedure followed for the graceful labeling of the Linear Dice Chain can be demonstrated by Figure 5.4.
In this case, let us denote the number of nodes as n, number of edges as m and number of dices as x. The procedure is capable enough to label a graph with any number of dices, gracefully. The following description enunciates the values of the assumed variables.
The value of x can be received from the user or analyst. The values of n and m are dependent on x. x must be positive integer. The values of n and m will depend on x in the following way.
$n = (x \times 4)(x - 1)$

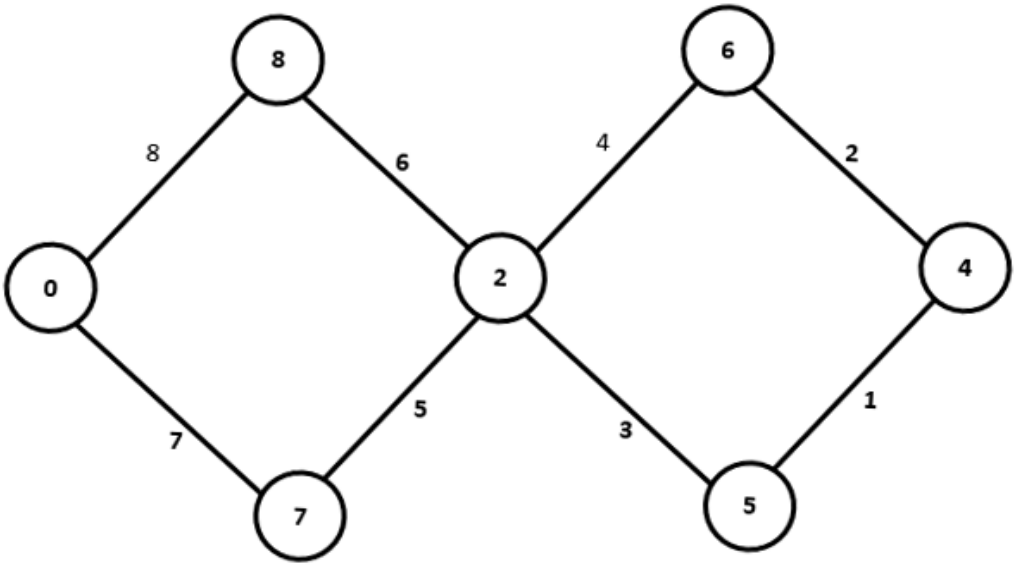

Figure 5.1: Gracefully labeled Linear Dice with 2 C_4.

$m = x \times 4$

From the above equations, it is obvious that $m > n$. This phenomenon can be defined as, since

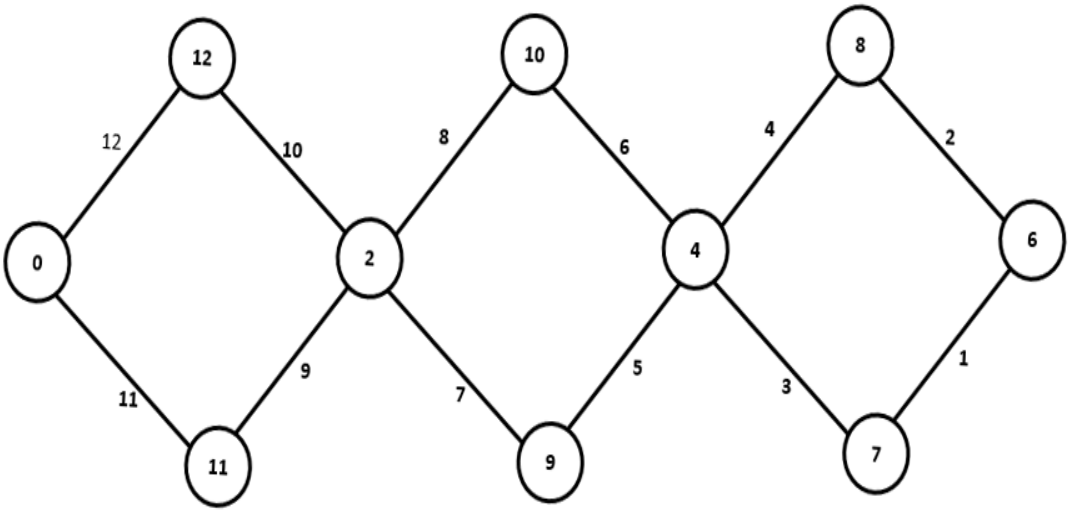

Figure 5.2: Gracefully labeled Linear Dice with 3 C_4.

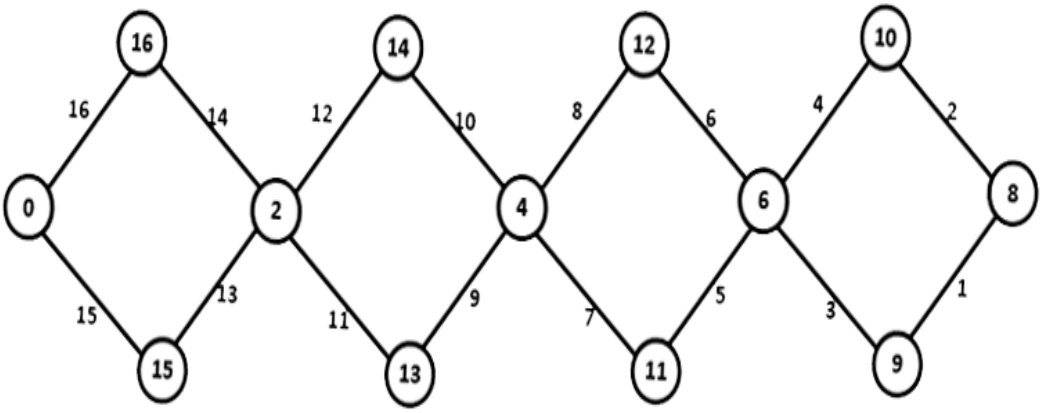

Figure 5.3: Gracefully labeled Linear Dice with 4 C_4.

two consecutive dices have a common node, one vertex is reduced for each pair. However, the total graph is a chain. So the two end vertices are not common to any other dices. As a

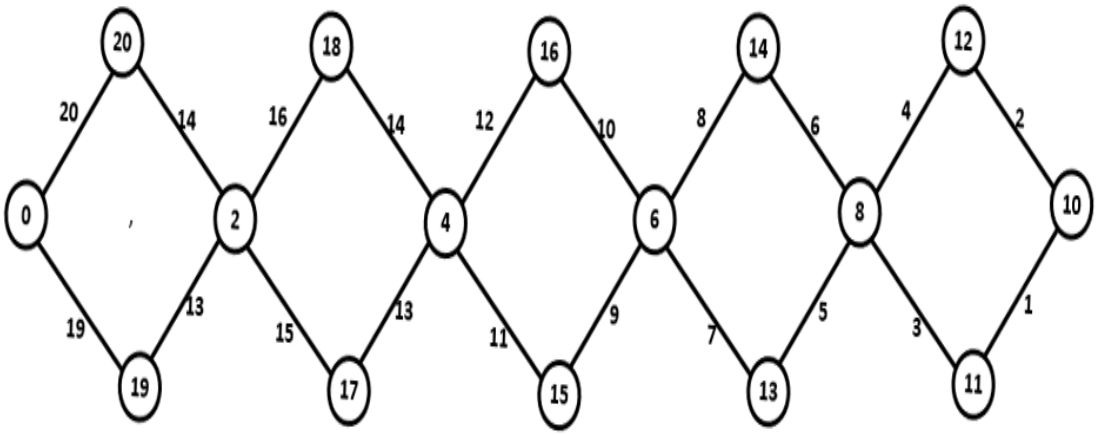

Figure 5.4: Gracefully labeled Linear Dice with 5 C_4.

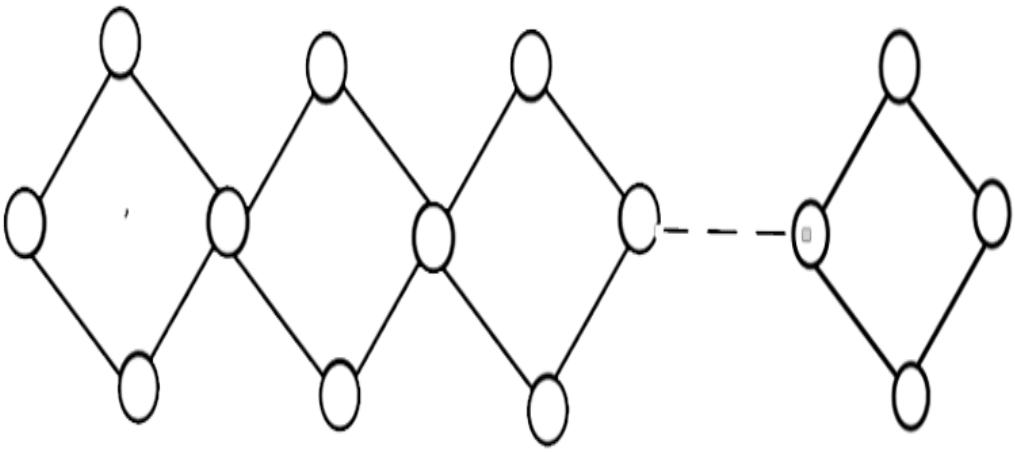

Figure 5.5: Gracefully labeled Linear Dice with n C_4.

result, $(x1)$ is to be subtracted from the total number to obtain n. On the other hand, each dice has four edges. Hence, the value of m can be clearly understood.
The available labels for the edges will be 1 to m. Each number within the range will be used for the label of any one of the edges.
The available labels for the nodes will be from 0 to m. The common nodes should be labeled with an interval of one from the opposite nodes. As a result, x labels will be unused. The node labels are to be started from 0. So, there will be x extra labels.
Following the above criteria, graph of any length of this class can be labeled gracefully. The first free node is to be labeled with the least available label. The opposite node is to be labeled with the number at the difference of one. The top node is to be assigned with the maximum available label and the bottom node with its immediate previous number. This technique to be followed for all the remaining dices.

5.2 Dice−Path Chain

This is a derivation of Linear Dice Chains. The linear dice chains have a common node between two consecutive dices. This type of graphs is derived by separating the common nodes and adding an edge between each consecutive pairs of dices. Figure **??** can demonstrate the formation procedure. Each dice is a C_4 graph. In order to elaborate the formation

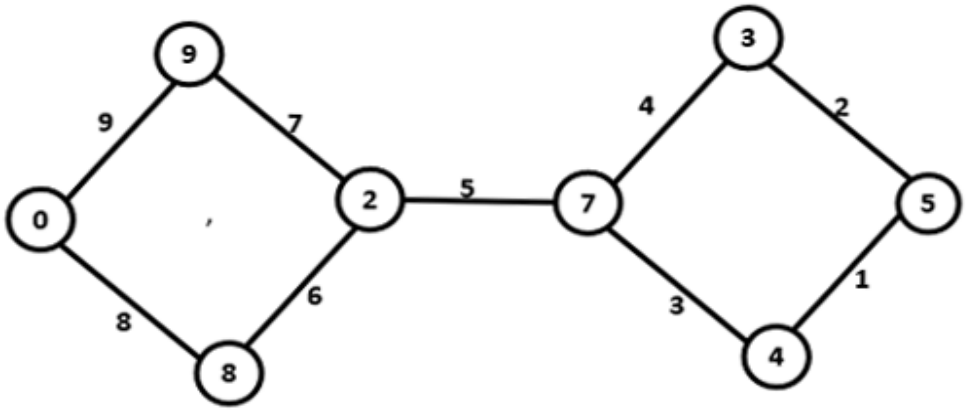

Figure 5.6: Gracefully labeled Dice−Path Chain with 2 C_4.

procedure, let us consider, x be the number of dices, n the number of nodes and m the number of edges. The value of x can be user input. It must be a positive integer. The values of n

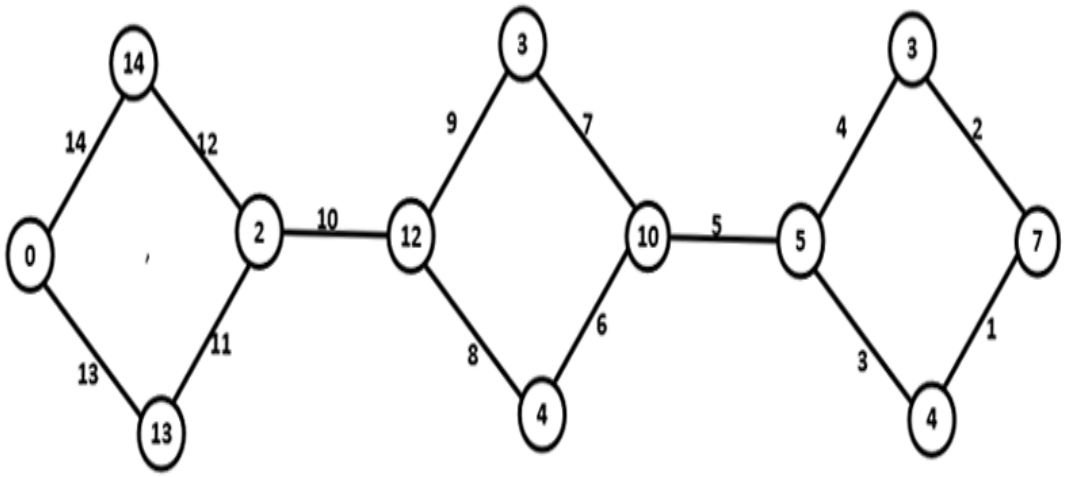

Figure 5.7: Gracefully labeled Dice−Path Chain with 3 C_4.

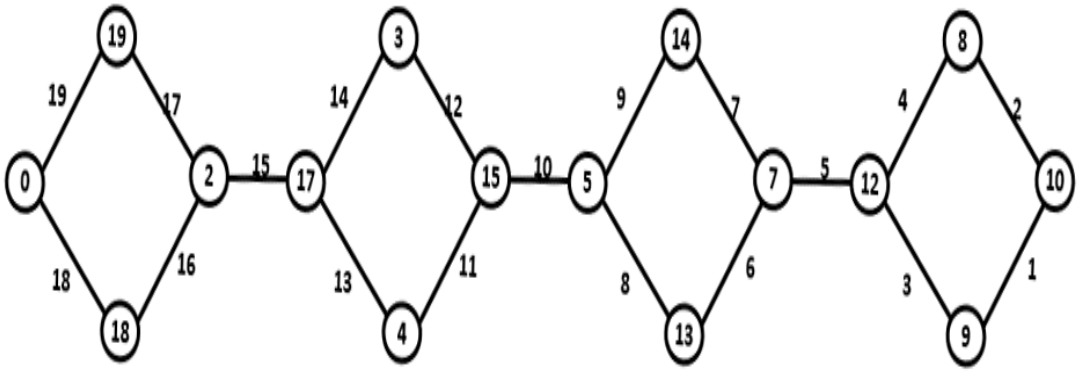

Figure 5.8: Gracefully labeled Dice−Path Chain with 4 C_4.

and m are dependent on the value of x. The following expressions can define the relation.
$n = x \times 4$
$m = (x \times 4) + (x1)$
From the above relations, it is clear that $m>n$. The number of nodes is simply the number of dices multiplied by 4. This is because no nodes are common between more than one dices. Hence, the number of nodes can be found out straight forward. The number of edges will require the addition of an extra value. This is because each pair of consecutive dices is connected by an edge. Since the graph forms a chain, such extra edges will be $(x1)$ in number. This extra value is added to the total number to get the number of edges.
The number of available labels for edges is 1 to m. Each label will be used for one particular edge without repetition.
The number of available labels for nodes is from 0 to m. Since $m>n$, there will be some extra available labels. Experimentally, it is found that there will be x number of extra labels. Number of edges is $(x1)$ more than that of nodes. Moreover, there is one more available label for nodes than edges. Hence, the total number of extra labels for nodes will be equal to the number of dices.
Following the above criteria, graphs containing any number of dices can be labeled gracefully.
The first free node is to be assigned with the minimum available label and its opposite one with the label at the difference of two. The top node is to be assigned with the maximum

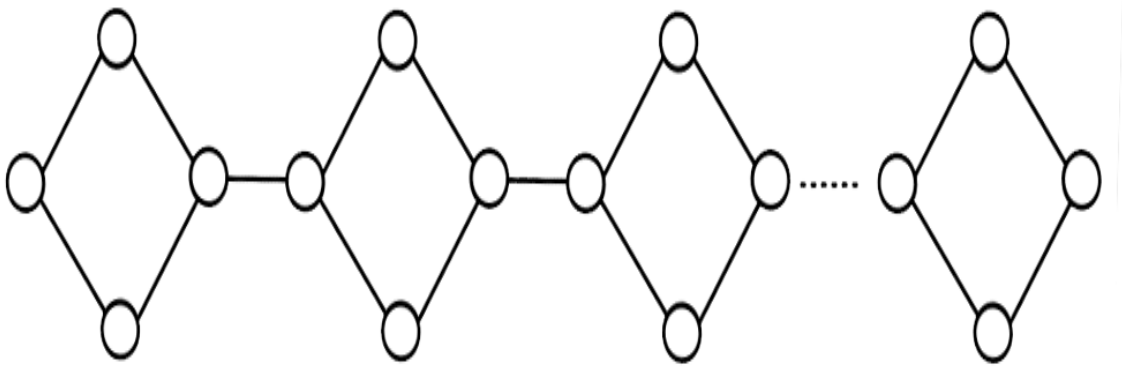

Figure 5.9: Gracefully labeled Dice−Path Chain with n C_4.

available label and its opposite with the immediate previous one. This procedure will be termed as forward labeling.
For the second dice, since its previous dice followed forward labeling, it will follow reverse labeling. Hence, the node that is connected with the previous dice with an edge will be assigned with the maximum available label and its opposite one will be given the label at the difference of two. The top node will be assigned the minimum available value and its opposite one, the number immediately next to it.
Altering the forward and reverse labeling procedure will gracefully label the total graph.

5.3 Linear Dice Chain with Connected End-Vertices

We have discussed earlier about the linear dices. They are the sequence of C_4 graphs connected by a common vertex within each pair. The difference between this class with linear dice class is that, the two end vertices (the floating vertices, one at the first dice and the other at the n-th dice) are connected by an edge. The graph may be denoted as containing nC_4 where n is an integer. Such graphs could be labeled gracefully for n = 2 to 5. Such graphs are demonstrated by the following figures.

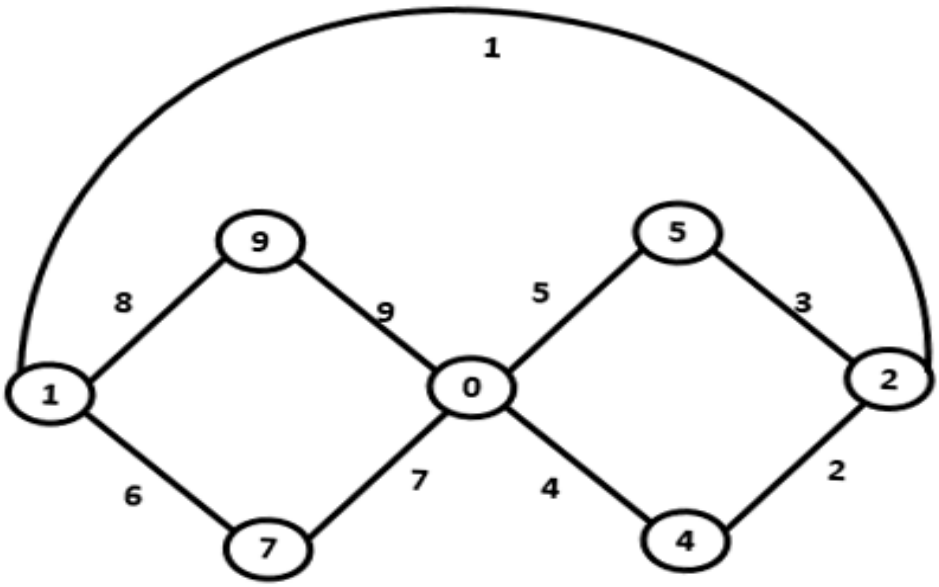

Figure 5.10: Linear Dice Chain with Connected End-Vertices with 2 C_4.

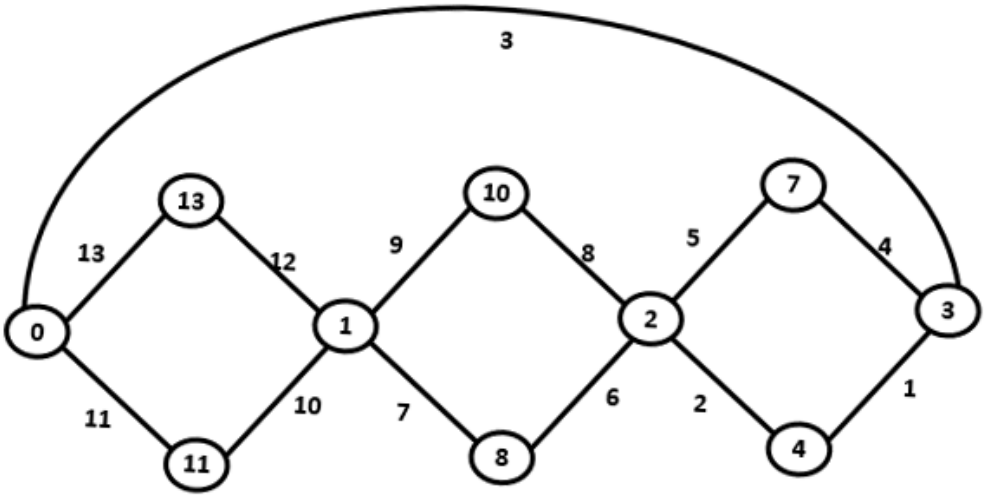

Figure 5.11: Linear Dice Chain with Connected End-Vertices with 3 C_4.

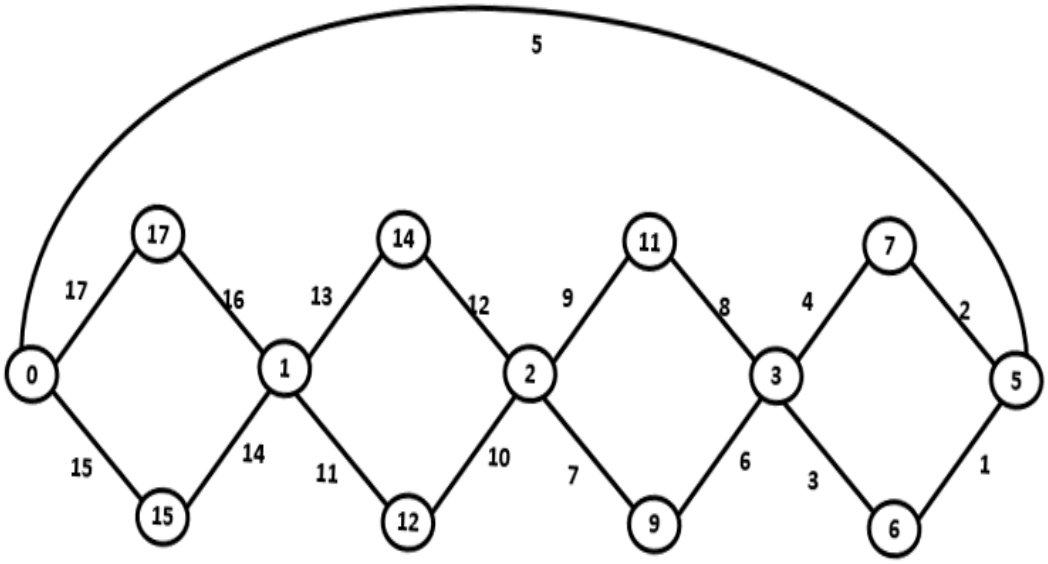

Figure 5.12: Linear Dice Chain with Connected End-Vertices with 4 C_4.

5.4 Double Star

Star is a category of graph, as well as trees, which contains a central node connected by any number of arms. A star with n arms is represented by $S_{1,n}$. It has been proved earlier that all stars are graceful. However, through our research, we successfully combined two stars together and labeled the graph gracefully. Both stars can have any number of arms, not necessarily the same number. A central node will connect one arm of each star. The arms through which the central node will be connected will depend on the labeling procedure.
If there are $n1$ arms in the first star and $n2$ arms in the second star, then there will be a total of $n1+n2+3$ vertices and $n1+n2+2$ edges. Hence, $n = n1+n2+3$ and $m = n1+n2+2$. Available labels for vertices will be 0 to $n1$ and that of edges will be 1 to $n1$.
The labeling procedure will initialize by denoting the first three available node labels to the two center vertices of the stars and the connecting node. After that, the arms of the first star are labeled, starting from the maximum available label for nodes and decreasing sequentially. After the first star is labeled, the arms of the second star are labeled, starting from the available maximum label number. When both the stars have been labeled, it is to be found out which edge labels are still unoccupied. It is to be mentioned that that the edge labels can be found out by the difference of the node labels of the adjacent vertices. There will be two unused edge labels left. The center node is connected to such arms of the stars that these two edge labels are used. Hence, the whole graph becomes gracefully labeled.

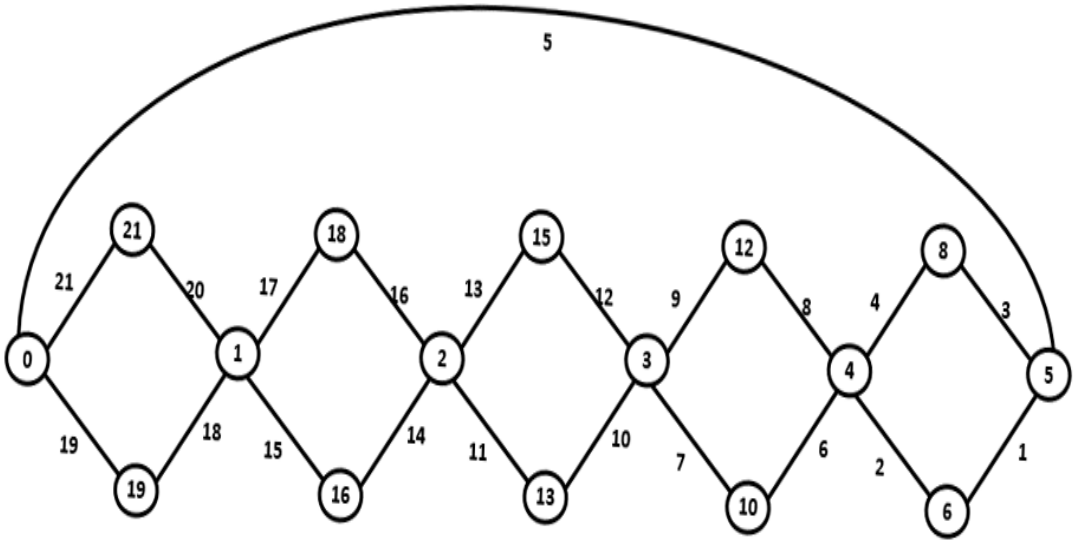

Figure 5.13: Linear Dice Chain with Connected End-Vertices with 5 C_4.

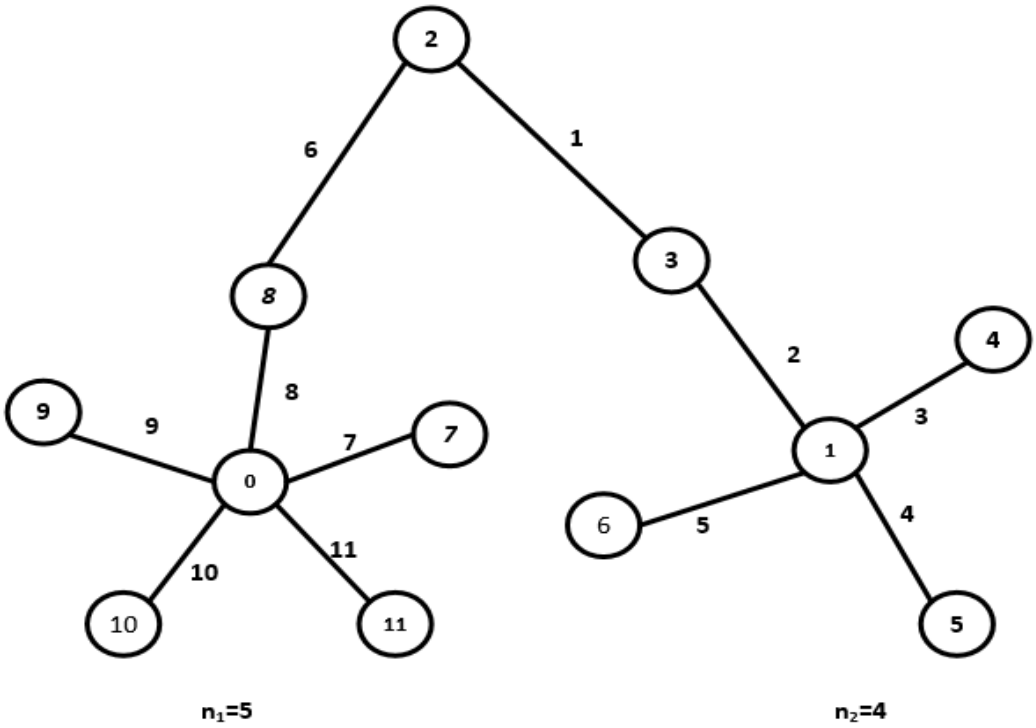

Figure 5.14: Gracefully labeled Double Star.

CHAPTER 6
EXPERIMENTAL RESULTS

6.1 Device

The codes implemented for our research were developed in a device running of Window OS. The minimum requirement for the example demonstration is the installation of Flash Player (minimum version 11). An update web browser will help in that case. The codes can be executed smoothly in Windows or Linux based devices.

6.2 Language

The language used for the development of the samples provided with this paper is Action-Script 3.0. It is an object-oriented programming language which was originally developed by Macromedia Inc., later dissolved by Adobe Systems. It is a superset of the syntax and semantics of the language JavaScript. The language is primarily used for the development of websites and software targeting the Adobe Flash Player platform.
Since the arrival of Flash Player 9 alpha, ActionScript version 3.0 was introduced. This version is compiled and run on a version of ActionScript Virtual Machine. ActionScript 3.0 executes up to 10 times faster than legacy ActionScript code due to Just-In-Time compiler enhancements. ActionScript 3.0 provides better options for user interactivity and the platform is open source. Hence, it provides better advantages to the developer whenever there is the possibility for user interaction and graphical manipulation. The original console screen acts as a blank canvas and thus, the developer gets a total option to execute any graphical illustrations required.
Since the gracefully labeled graphs produced by the following examples are dependent up on the input by the user, hence the algorithm is executed in the background procedure after the inputs are taken and the developer manages to display the result sets in the form of gracefully labeled graphs.

6.3 IDE

The IDE(Integrated Development Environment) used for the development of the algorithms and the example codes provided with this paper was Adobe Flash Professional CS6. It is a part of the Adobe CS6 Master Collection.
Adobe Flash Professional is a multimedia authoring program used to create content for Adobe Engagement Platform. It is used to develop web applications, games, movies, content for mobile phones and other embedded devices. The platform supports the scripting language ActionScript 3.0 in case of user interaction and graphical manipulation.
The version, Adobe Flash Professional CS6 as released in 2012. It was upgraded from the previous versions by integrating the support of HTML5(Hyper Text Markup Language) and the ability to generate spread sheets.

6.4 Experimental Data

Various theories have been developed regarding graceful labeling. However, graphical demonstration has been scarce. One of the main motives of our research was to find out the possible ways of graphical demonstration, so that readers and eager researchers could get a practical example. Hence, we developed the algorithms and implemented their corresponding codes. The codes are attached with this paper. Some examples of our code implementations are provided in this section.

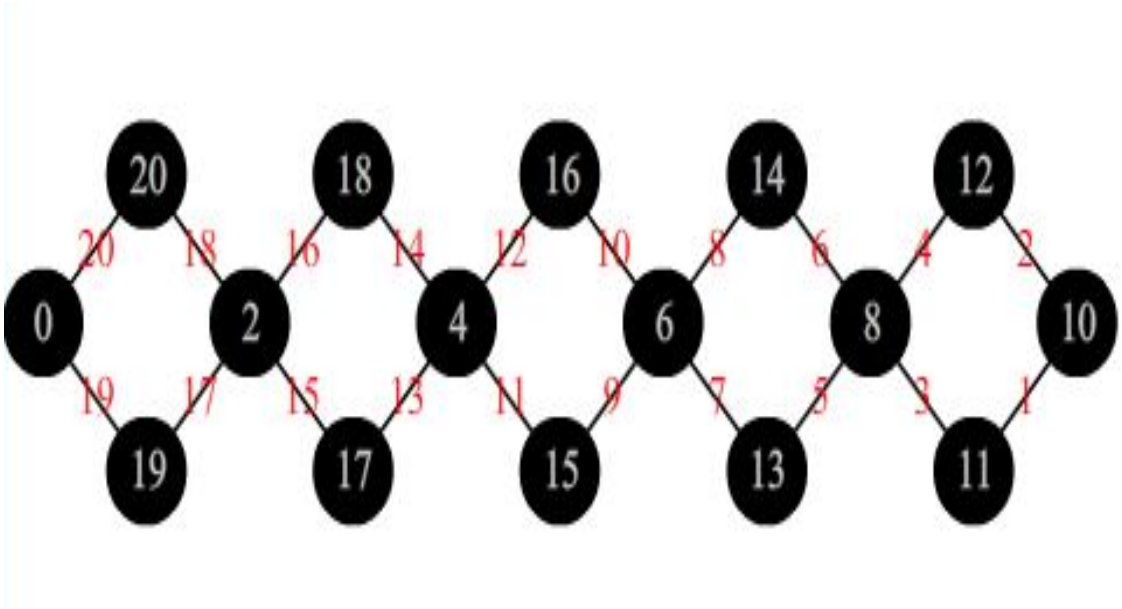

Figure 6.1: Graceful Labeling of Linear Dice.

6.4.1 Code Specification Linear Dice

The pseudo code 1 labels a linear dice by taking the manual input of number of dices. The number of dices is denoted by the integer variable 'diceNumber'. After the simulation of the following pseudo code, three arrays will contain the result, viz. topRow, middleRow and bottomRow. The arrays, as their name suggests, contains the labels of the nodes in top row, middle row and bottom row respectively in sequential order.

6.4.2 Input Format for Linear Dice

The input for the linear dice labeling is an integer number. Inserting the number and pressing simulate will show the graph with the provided number of dices, gracefully labeled. In the example 6.1, the number given as input was 5.

6.4.3 Test Cases for Linear Dice

User can choose to produce a graphical representation based of the result arrays in any platform they prefer. A simple example of such an illustration 6.1 with actionscript 3.0 is provided with this paper.

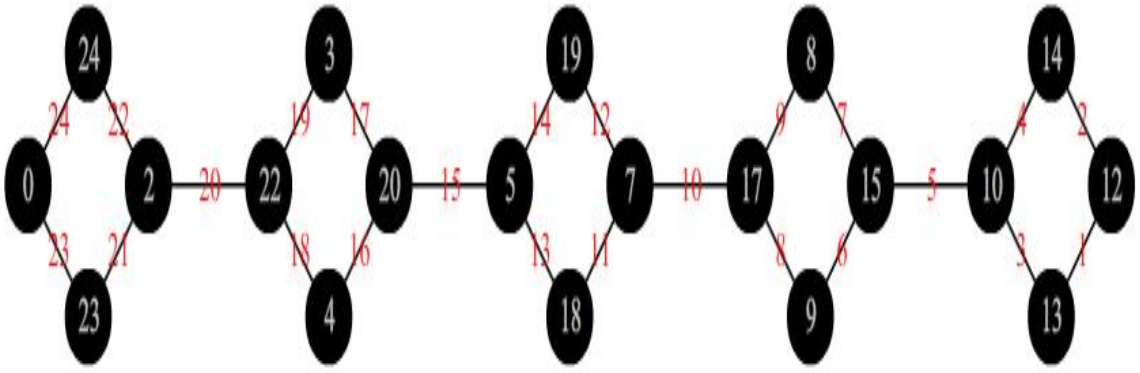

Figure 6.2: Graceful Labeling of Dice−Path Chain.

6.4.4 Code Specification for Dice−Path Chain

The pseudo code 2 takes the number of dices as the parameter. The number of dices in the required chain to be formed is denoted by the integer 'diceNumber'. User can manually provide the value. After the simulation, the gracefully labeled vertices will be denoted by the values in the array vertexNumber. The index of the array will determine which of the four endings of the dice the number points. If x is the value in vertexNumber$[i]$, then the value of i mod(4) will determine the position of x.
If i mod(4) $= 1, x$ is the label of the left corner of the $((i/4) + 1)-$th dice.
If i mod(4) $= 2, x$ is the label of the right corner of the $((i/4) + 1)-$th dice.
If i mod(4) $= 3, x$ is the label of the top corner of the $((i/4) + 1)-$th dice.
If i mod(4) $= 0, x$ is the label of the bottom corner of the $((i/4) + 1)-$th dice.

6.4.5 Input Format for Dice−Path Chain

The input for the dice path chain graph is also similar to that of the linear dice. The input should be an integer number denoting the number of dices in the graph. Pressing simulate button will produce the output, gracefully labeled graph. In the example 6.2, the input was 5.

6.4.6 Test Cases for Dice−Path Chain

Graphical illustrations can be executed any language the user prefers. A simple example of such an illustration 6.2 with in actionscript 3.0 is provided with this paper.

6.4.7 Code Specification for Double Star

In order to execute the pseudo code 3 , two integer variables has to be provided to the function. The number of arms for each star will be provided as armsNumber1 and armsNumber2 respectively, where armsNumber1 is a positive integer and armsNumber2 is a negative integer. After the simulation, the vertexNumber array will contain the labels of nodes in graceful manner. The sequence will be as follows :
1. Initial index (0) will contain label of the center node of first star
2. Following armsNumber1 indices will contain the labels of the arms of the first star
3. The next index will contain the label of the center node of the second star
4. Following armsNumber2 indices will contain the labels of the arms of the second star
5. The last index, i.e. the index indicating the number of vertices will contain the label of the joining node (n−th vertex)

Apart from the array, two variables join1 and join2 will contain the edge numbers of the edges joining the $n-th$ vertex with the observed arms of the first star and second star respectively.

6.4.8 Input Format for Double Star

The input for the simulation of the double star are to integer number. The first input textbox is for the number of arms of the first star, which must be a positive integer. The second textbox is for the number of arms of the second star. Pressing simulate will provide the required graphical illustration of the gracefully labeled graph. In the provided test case, the inputs were 5 and 4.

6.4.9 Test Cases for Double Star

User may use any language to produce a graphical representation or use the values in any way necessary. This paper will contain a simple example 6.3 executed in actionscript 3.0.

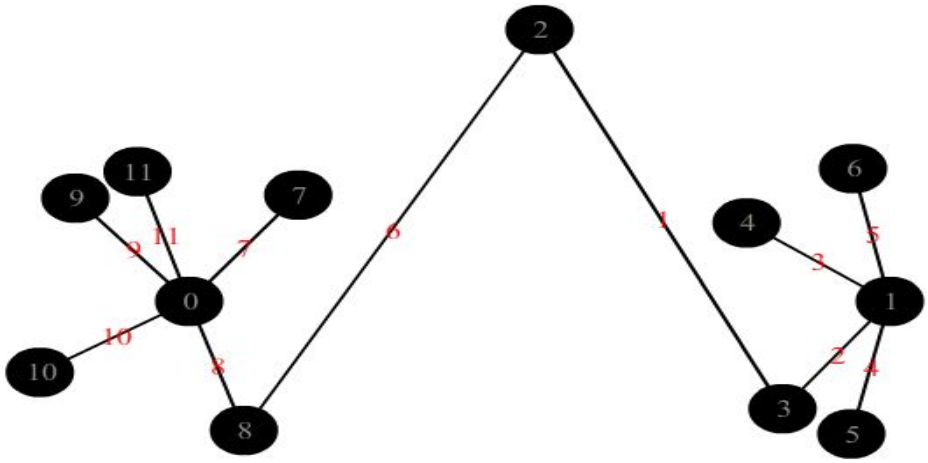

Figure 6.3: Graceful Labeling of Double Star.

6.5 Additinal implementations

6.5.1 Star

The input for the star labeling demonstration is a number between 1 to 17. In the example 6.4, the input is 17.

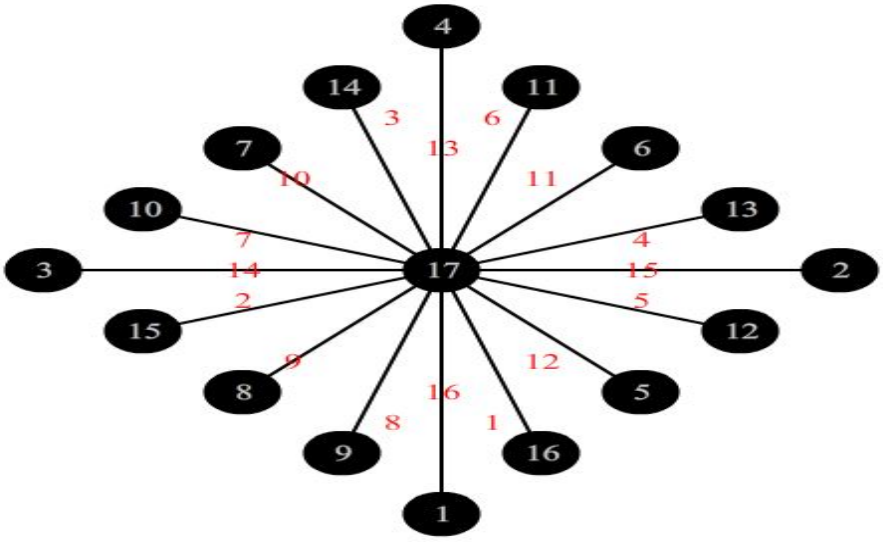

Figure 6.4: Graceful Labeling of Star.

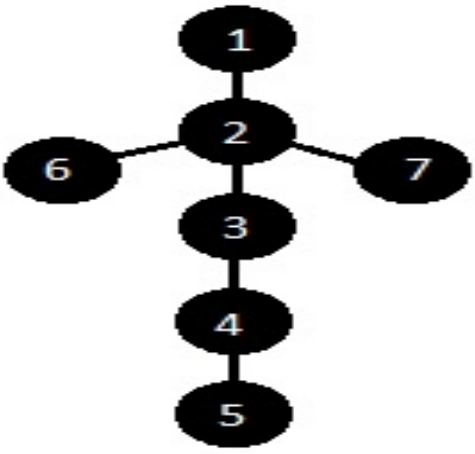

Figure 6.5: Input graph representation for caterpillar demonstration.

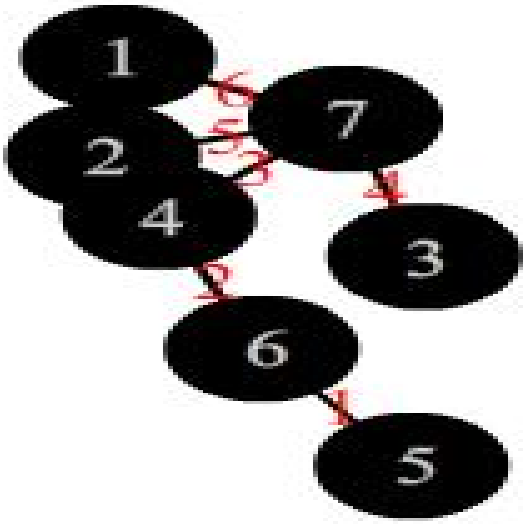

Figure 6.6: Output graph representation for caterpillar demonstration.

6.5.2 Input Format for Caterpillar

The adjacent nodes for the edges are to be provided. Then, the total number of nodes. After that, pressing the simulate button will give the output.
In the example 6.5, the provided edges were 1 2, 2 3, 3 4, 4 5, 2 6, 2 7. Then the total number of nodes provided was 7.

6.5.3 Test Cases for Caterpillar

For example 6.6

CHAPTER 7
CONCLUSION

Graceful graphs have a range of practical application domains, including radio astronomy, X-ray crystallography, cryptography, and experimental design, coding theory and communication network addressing, such as Multiprotocol Label Switching (MPLS) multicasting using Caterpillars and Graceful Graphs. Graph theory has turned out to be a vast area with innumerable applications in the field of social networks, data organization, communication network, discrete mathematics and so on. The Graceful Tree Conjecture was initially interesting, mostly because of its connection to Ringel's Conjecture, but soon became famous in its own right. Despite efforts of many researchers, only limited progress has been made over the last few decades. To date, only some very restricted classes of trees have been shown to be graceful. Although some progress has been made on the relaxed labeling, we still are far from a solution. This paper presents classes of graceful graphs and trees, theorems regarding graceful labeling along with some new results on the gracefulness of three classes of graphs. We look to obtain even greater results in the future regarding graphs in these three classes. Hopefully similar results can be achieved for larger graphs as well. Graceful labeling has been studied for over three decades, and the topic continues to be a fascinating one in the world of graph theory and discrete mathematics. An abundance of published papers and results exist, yet various unsolved problems and unproven conjectures continue to allow the undertaking of even more research, with the hopes that new results will be obtained.

REFERENCES

[1] A. K. Khan, R. Vatsa, S. Roy, and B. Das, "On certain valuations of the vertices of a graph," in *Theory of Graphs(International Symposium, Rome, July 1966)*, pp. 349–355, 1967.

[2] S. W. Golomb, *How to number a graph, Graph Theory and Computing*. Academic Press, Newyork, 1972.

[3] B. Bollobas, "Graph theory: an introductory course," 1979.

[4] D. A. Sheppard, "The factorial representation of balanced labelled graphs," in *Discrete Math*, pp. 379–388, 1976.

[5] R. W. Frucht, "Graceful numbering of wheels and related graphs," pp. 219–319, Annals of the New York Academy of Science, 1979.

[6] C. M. Cavalier, "Graceful labeling," pp. 4–40, University of South Carolina, 2009.

[7] J. C. Bermond and D. Sotteau, "Graph decompositions and g-designs," 1975.

[8] I. Cahit and R. Cahit, "On the Graceful Numbering of Spanning Trees," *Information Processing Letters*, vol. 3, pp. 115–118, 1975.

[9] A. M. Pastel and H. Raynaud, "Les oliviers sont gracieux," *Colloq. Grenoble*, 1978.

[10] V. N. Bhat-Nayak and U. N. Deshmukh, "New families of graceful banana trees," *Proceedings of The Indian Academy of Sciences-mathematical Sciences*, vol. 106, pp. 201–216, 1996.

[11] S. M. Hegde and S. Shetty, "On Graceful Trees," pp. 192–197, 2002.

[12] K. M. Koh, D. G. Rogers, and T. Tan, "Products of graceful trees," *Discrete Mathematics*, vol. 31, pp. 279–292, 1980.

[13] E. Robeva, "An Extensive Survey of Graceful Trees," 2011.

[14] B. D. Acharya, *On d-sequential graphs*. 1983.

[15] M. Maheo and H. Thullier, "On d-graceful graphs," in *Ars Combinat*, pp. 181–192, 1982.

[16] D. W. Bange, A. E. Barkauskas, and P. J. Slater, "Simply sequential and graceful graphs," pp. 155–162, 1979.

[17] H. K. Cornelis Hoede, *All wheels are graceful*. 1987.

[18] J. Ayel and O. Favaron, *Helms are graceful*. Academic Press, Toronto, Ontario, 1984.

[19] C. Delorme, M. Maheo, H. Thuillier, K. M. Koh, and H. K. Teo, *Cycles with a chord are graceful*, vol. 4. 1980.

[20] K. M. Koh and N. Punnim, *On graceful graphs: Cycles with 3-consecutive chords*. Bull. Malaysian Math, 1982.

[21] M. Truszczynski, *Graceful unicyclic graphs*. 1984.

[22] D. Moulton, "Graceful labeling of triangular snakes," in *Ars Combinat*, pp. 3–13, 1989.

APPENDIX A
ALGORITHMS

A.1 Algorithm

Algorithm 1 A gracefully labeled linear dice

LinearDice(diceNumber)
totalEdges = diceNumber $\times$ 4
for $i = 1$ to diceNumber $+1$ **do**
  middleRow$[i] = (i - 1) \times 2$
end for
for $i = 1$ to diceNumber and j =totalEdges **do**
  topRow$[i] = j$
  $j = j - 2$
end for
for $i = 1$ to diceNumber and j =totalEdges 1 **do**
  bottomRow$[i] = j$
  $j = j - 2$
end for

Algorithm 2 A gracefully labeled dice path chain

DicePath Chain(diceNumber)
totalEdges = (diceNumber $\times$ 4) + (diceNumber 1)
maxAvailable = totalEdges
minAvailable = 0
for $i = 1$ to (diceNumber $\times 4$) **do**
  if labeled forward **then**
    vertexNumber$[i]$ =minAvailable
    vertexNumber$[i+1]$ =minAvailable $+2$
    minAvailable = minAvailable $+3$
    vertexNumber$[i+2]$ = maxAvailable
    vertexNumber$[i+3]$ = maxAvailable 1
    maxAvailable = maxAvailable 2
  else
    vertexNumber$[i]$ = maxAvailable
    vertexNumber$[i+1]$ = maxAvailable 2
    maxAvailable = maxAvailable 3
    VertexNumber$[i+2]$ = minAvailable
    vertexNumber$[i+3]$ = minAvailable $+1$
    minAvailable = minAvailable$+2$
  end if
end for

Algorithm 3 A gracefully labeled double star

```
DoubleStar(armsNumber1, armsNumber2)
count = 0
totalVertex = armsNumber1 + armsNumber2 +3
vertexNumber[armsNumber1 +1] = 1
for i = 1 to armsNumber1 do
   vertexNumber[i] =(totalVertex 1) i + 1
end for
for i = 0 to totalVertex 1 do
   if i not used for any vertex then
      vertexNumber[totalVertex −1] = i
      break
   end if
end for
for i = 1 to armsNumber1 do
   edge[vertexNumber[i] vertexNumber[0]] = true
end for
for i = (armsNumber1 +2) to (armsNumber1 +2+ armsNumber2) do
   edge[vertexNumber[i] vertexNumber[armsNumber1 +1]] = true
end for
join1 = vertexNumber[totalVertex −1]
join2 = vertexNumber[totalVertex −1]
if armsNumber1 is 0 then
   join1 = vertexNumber[totalVertex −1] vertexNumber[0]
   edge[join1] = true
   count = count +1
end if
if armsNumber2 is 0 then
   join2 = vertexNumber[totalVertex 1]4 vertexNumber[armsNumber1+1]
   edge[join2] = true
end if
for i = 1 to totalVertex 1 do
   if edge[i] is false then
      if count is 0 then
         join1 = i
         count = count +1
         join2 = i
      end if
   end if
end for
```

APPENDIX B
CODES

B.1 Codes

The following code will initialize Linear Dice

```
1  import flash.events.MouseEvent;
2 var diceNumber:int;
3 var diceText:String;
4 var topRow:Array = new Array;
5 var midRow:Array = new Array;
6 var botRow:Array = new Array;
7 var totalEdges:int;
8 var i:int;
9 var j:int;
10 simulateButton.buttonMode = true;
11 simulateButton.addEventListener(MouseEvent.CLICK, simulate);
12 function simulate(e:MouseEvent):void{
13         diceText = diceInput.text;
14         diceNumber = Number(diceText);
15         totalEdges = diceNumber * 4;
16         for(i = 1; i <= (diceNumber + 1); i++){
17                 midRow[i] = (i - 1) * 2;
18         }
19         for(j = totalEdges, i = 1; i <= diceNumber; i++, j -= 2){
20                 topRow[i] = j;
21         }
22         for(j = (totalEdges - 1), i = 1; i <= diceNumber; i++, j -= 2){
23                 botRow[i] = j;
24         }
25 }
```

The following code will execute Linear Dice

```
1  import flash.geom.Point;
```

```
var nodeX:Array = new Array;
var nodeY:Array = new Array;
var n:node;
var l:nodelabel;
var e:edgelabel;
this.graphics.lineStyle(2);
this.graphics.beginFill(0xFFFFFF);
var xCoor:Number = 20;
var yCoor:Number = 300;
for(i = 1; i <= (diceNumber + 1); i++){
        n = new node();
        addChild(n);
        n.x = nodeX[i] = xCoor;
        n.y = nodeY[i] = yCoor;
        l = new nodelabel();
        addChild(l);
        l.x = xCoor;
        l.y = yCoor;
        l.nodeNumber.text = midRow[i].toString();
        xCoor += 100;
}
xCoor = 70;
yCoor = 250;
for(i = 1; i <= (diceNumber); i++){
        n = new node();
        addChild(n);
        n.x = xCoor;
        n.y = yCoor;
        this.graphics.moveTo(xCoor, yCoor);
        this.graphics.lineTo(nodeX[i], nodeY[i]);
        e = new edgelabel;
        addChild(e);
        e.x = (xCoor + nodeX[i]) / 2;
        e.y = (yCoor + nodeY[i]) / 2;
        trace(e.x + " " + e.y);
        e.edgeNumber.text = (topRow[i] - midRow[i]).toString();
        this.graphics.moveTo(xCoor, yCoor);
        this.graphics.lineTo(nodeX[i + 1], nodeY[i + 1]);
        e = new edgelabel;
```

```
41          addChild(e);
42          e.x = (xCoor + nodeX[i + 1]) / 2;
43          e.y = (yCoor + nodeY[i + 1]) / 2;
44          e.edgeNumber.text = (topRow[i] - midRow[i + 1]).toString();
45          l = new nodelabel();
46          addChild(l);
47          l.x = xCoor;
48          l.y = yCoor;
49          l.nodeNumber.text = topRow[i].toString();
50          xCoor += 100;
51  }
52  xCoor = 70;
53  yCoor = 350;
54  for(i = 1; i <= (diceNumber); i++){
55          n = new node();
56          addChild(n);
57          n.x = xCoor;
58          n.y = yCoor;
59          this.graphics.moveTo(xCoor, yCoor);
60          this.graphics.lineTo(nodeX[i], nodeY[i]);
61          e = new edgelabel;
62          addChild(e);
63          e.x = (xCoor + nodeX[i]) / 2;
64          e.y = (yCoor + nodeY[i]) / 2;
65          e.edgeNumber.text = (botRow[i] - midRow[i]).toString();
66          this.graphics.moveTo(xCoor, yCoor);
67          this.graphics.lineTo(nodeX[i + 1], nodeY[i + 1]);
68          e = new edgelabel;
69          addChild(e);
70          e.x = (xCoor + nodeX[i + 1]) / 2;
71          e.y = (yCoor + nodeY[i + 1]) / 2;
72          e.edgeNumber.text = (botRow[i] - midRow[i + 1]).toString();
73          l = new nodelabel();
74          addChild(l);
75          l.x = xCoor;
76          l.y = yCoor;
77          l.nodeNumber.text = botRow[i].toString();
78          xCoor += 100;
79  }
```

The following code will initialize Dice−Path Chain

```
1  import flash.events.MouseEvent;
2 var diceText:String;
3 var diceNumber:int;
4 var totalEdges:int;
5 var maxAvail:int;
6 var minAvail:int;
7 var forward:Boolean = true;
8 var i:int;
9 var vertexNumber:Array = new Array;
10 simulateButton.buttonMode = true;
11 simulateButton.addEventListener(MouseEvent.CLICK, simulate);
12 function simulate(e:MouseEvent):void{
13         diceText = diceInput.text;
14         diceNumber = Number(diceText);
15         totalEdges = (diceNumber * 4) + (diceNumber - 1);
16         maxAvail = totalEdges;
17         minAvail = 0;
18         for(i = 1; i <= (diceNumber * 4); i += 4){
19                 if(forward){
20                         forward = !forward;
21                         vertexNumber[i] = minAvail;
22                         vertexNumber[i+1] = minAvail + 2;
23                         minAvail += 3;
24
25                         vertexNumber[i+2] = maxAvail;
26                         vertexNumber[i+3] = maxAvail - 1;
27                         maxAvail -= 2;
28                 }
29                 else{
30                         forward = !forward;
31                         vertexNumber[i] = maxAvail;
32                         vertexNumber[i+1] = maxAvail - 2;
33                         maxAvail -= 3;
34
35                         vertexNumber[i+2] = minAvail;
36                         vertexNumber[i+3] = minAvail + 1;
37                         minAvail += 2;
38                 }
```

```
39          }
40 }
```

The following code will execute Dice−Path Chain

```
1  import flash.geom.Point;
2 var nodeX:Array = new Array;
3 var nodeY:Array = new Array;
4 var n:node;
5 var e:edgelabel;
6 var l:nodelabel;
7 this.graphics.lineStyle(2);
8 this.graphics.beginFill(0xFFFFFF);
9 var xCoor:Number = 20;
10 var yCoor:Number = 300;
11 for(i = 1; i <= (diceNumber * 4); i += 4){
12          n = new node();
13          addChild(n);
14          n.x = nodeX[i] = xCoor;
15          n.y = nodeY[i] = yCoor;
16          l = new nodelabel();
17          addChild(l);
18          l.x = xCoor;
19          l.y = yCoor;
20          l.nodeNumber.text = vertexNumber[i].toString();
21          if(i != 1){
22                  this.graphics.moveTo(nodeX[i], nodeY[i]);
23                  this.graphics.lineTo(nodeX[i - 3], nodeY[i - 3]);
24                  e = new edgelabel;
25                  addChild(e);
26                  e.x = (nodeX[i] + nodeX[i - 3]) / 2;
27                  e.y = (nodeY[i] + nodeY[i - 3]) / 2;
28                  if((vertexNumber[i] - vertexNumber[i - 3]) >= 0){
29                          e.edgeNumber.text = (vertexNumber[i] -
30                                          vertexNumber[i - 3]).toString()
31                  }
32                  else{
33                          e.edgeNumber.text = ((vertexNumber[i] -
34                          vertexNumber[i - 3]) * (-1)).toString();
35                  }
```

```
36          }
37          n = new node();
38          addChild(n);
39          n.x = nodeX[i + 1] = xCoor + 100;
40          n.y = nodeY[i + 1] = yCoor;
41          l = new nodelabel();
42          addChild(l);
43          l.x = xCoor + 100;
44          l.y = yCoor;
45          l.nodeNumber.text = vertexNumber[i + 1].toString();
46          n = new node();
47          addChild(n);
48          n.x = nodeX[i + 2] = xCoor + 50;
49          n.y = nodeY[i + 2] = yCoor - 50;
50          l = new nodelabel();
51          addChild(l);
52          l.x = xCoor + 50;
53          l.y = yCoor - 50;
54          l.nodeNumber.text = vertexNumber[i + 2].toString();
55          this.graphics.moveTo(nodeX[i], nodeY[i]);
56          this.graphics.lineTo(nodeX[i + 2], nodeY[i + 2]);
57          e = new edgelabel;
58          addChild(e);
59          e.x = (nodeX[i] + nodeX[i + 2]) / 2;
60          e.y = (nodeY[i] + nodeY[i + 2]) / 2;
61          if((vertexNumber[i] - vertexNumber[i + 2]) >= 0){
62                  e.edgeNumber.text = (vertexNumber[i] -
63                                  vertexNumber[i + 2]).toString();
64          }
65          else{
66                  e.edgeNumber.text = ((vertexNumber[i] -
67                  vertexNumber[i + 2]) * (-1)).toString();
68          }
69          this.graphics.moveTo(nodeX[i + 1], nodeY[i + 1]);
70          this.graphics.lineTo(nodeX[i + 2], nodeY[i + 2]);
71          e = new edgelabel;
72          addChild(e);
73          e.x = (nodeX[i + 1] + nodeX[i + 2]) / 2;
74          e.y = (nodeY[i + 1] + nodeY[i + 2]) / 2;
```

```
75          if((vertexNumber[i + 1] - vertexNumber[i + 2]) >= 0){
76                  e.edgeNumber.text = (vertexNumber[i + 1] -
77                                  vertexNumber[i + 2]).toString();
78          }
79          else{
80                  e.edgeNumber.text = ((vertexNumber[i + 1] -
81                  vertexNumber[i + 2]) * (-1)).toString();
82          }
83          n = new node();
84          addChild(n);
85          n.x = nodeX[i + 3] = xCoor + 50;
86          n.y = nodeY[i + 3] = yCoor + 50;
87          l = new nodelabel();
88          addChild(l);
89          l.x = xCoor + 50;
90          l.y = yCoor + 50;
91          l.nodeNumber.text = vertexNumber[i + 3].toString();
92          this.graphics.moveTo(nodeX[i], nodeY[i]);
93          this.graphics.lineTo(nodeX[i + 3], nodeY[i + 3]);
94          e = new edgelabel;
95          addChild(e);
96          e.x = (nodeX[i] + nodeX[i + 3]) / 2;
97          e.y = (nodeY[i] + nodeY[i + 3]) / 2;
98          if((vertexNumber[i] - vertexNumber[i + 3]) >= 0){
99                  e.edgeNumber.text = (vertexNumber[i] -
100                                 vertexNumber[i + 3]).toString();
101         }
102         else{
103                 e.edgeNumber.text = ((vertexNumber[i] -
104                 vertexNumber[i + 3]) * (-1)).toString();
105         }
106         this.graphics.moveTo(nodeX[i + 1], nodeY[i + 1]);
107         this.graphics.lineTo(nodeX[i + 3], nodeY[i + 3]);
108         e = new edgelabel;
109         addChild(e);
110         e.x = (nodeX[i + 1] + nodeX[i + 3]) / 2;
111         e.y = (nodeY[i + 1] + nodeY[i + 3]) / 2;
112         if((vertexNumber[i + 1] - vertexNumber[i + 3]) >= 0){
113                 e.edgeNumber.text = (vertexNumber[i + 1] -
```

```
114                                     vertexNumber[i + 3]).toString();
115         }
116         else{
117                 e.edgeNumber.text = ((vertexNumber[i + 1] -
118                 vertexNumber[i + 3]) * (-1)).toString();
119         }
120         xCoor += 200;
121 }
```

The following code will initialize Double Star

```
1  import flash.events.MouseEvent;
2 var arms1:String;
3 var arms2:String;
4 var armsNumber1:int;
5 var armsNumber2:int;
6 var totalVertex:int;
7 var join1:int;
8 var join2:int;
9 var vertexNumber:Array = new Array;
10 var edgeNumber:Array = new Array;
11 var vertexCheck:Array = new Array;
12 var edgeCheck:Array = new Array;
13 var i:int;
14 var j:int;
15 simulateButton.buttonMode = true;
16 simulateButton.addEventListener(MouseEvent.CLICK, simulateCode);
17 function simulateCode(e:MouseEvent):void{
18         arms1 = inputArms1.text;
19         armsNumber1 = Number(arms1);
20         arms2 = inputArms2.text;
21         armsNumber2 = Number(arms2);
22         totalVertex = armsNumber1 + armsNumber2 + 3;
23         for(i = 0; i < totalVertex; i++){
24                 vertexCheck[i] = false;
25                 edgeCheck[i] = false;
26         }
27         vertexNumber[0] = 0;
28         vertexCheck[0] = true;
29         vertexNumber[armsNumber1 + 1] = 1;
```

```
30          vertexCheck[1] = true;
31          for(i = 1; i <= armsNumber1; i++){
32                  vertexNumber[i] = (totalVertex - 1) - i + 1;
33                  vertexCheck[(totalVertex - 1) - i + 1] = true;
34          }
35          var secondStarStart:int = totalVertex - armsNumber1;
36          for(i = (armsNumber1 + 2), j = 1;
37                  i < (armsNumber1 + 2) + armsNumber2; i++, j++){
38                  vertexNumber[i] = secondStarStart - j;
39                  vertexCheck[secondStarStart - j] = true;
40          }
41          for(i = 0; i< totalVertex; i++){
42                  if(vertexCheck[i] == false){
43                          vertexNumber[totalVertex - 1] = i;
44                          vertexCheck[i] = true;
45                          break;
46                  }
47          }
48          for(i = 1; i <= armsNumber1; i++){
49                  edgeCheck[vertexNumber[i] - vertexNumber[0]] = true;
50          }
51          for(i = (armsNumber1 + 2); i < (armsNumber1 + 2) + armsNumber2;
52                  i++){
53                  edgeCheck[vertexNumber[i] - vertexNumber[armsNumber1 +
54
55          }
56          var count:int = 0;
57          join1 = vertexNumber[totalVertex - 1];
58          join2 = vertexNumber[totalVertex - 1];
59          if(armsNumber1 == 0){
60                  join1 = vertexNumber[totalVertex - 1] - vertexNumber[0]
61                  edgeCheck[join1] = true;
62                  count++;
63          }
64          if(armsNumber2 == 0){
65                  join2 = vertexNumber[totalVertex - 1] -
66
67                  edgeCheck[join2] = true;
68          }
```

```
69          for(i = 1; i < totalVertex; i++){
70                  if(edgeCheck[i] == false){
71                          if(count == 0){
72                                  join1 = i;
73                                  count++;
74                          }
75                          else if(count == 1 && join2 ==
76
77                                  join2 = i;
78                          }
79                  }
80          }
81 }
```

The following code will execute Double Star

```
1  import flash.geom.Point;
2 var nodeX:Array = new Array;
3 var nodeY:Array = new Array;
4 var n:node;
5 var l:nodelabel;
6 var e:edgelabel;
7 this.graphics.lineStyle(2);
8 this.graphics.beginFill(0xFFFFFF);
9 n = new node();
10 addChild(n);
11 n.x = nodeX[0] = 200;
12 n.y = nodeY[0] = 350;
13 l = new nodelabel();
14 addChild(l);
15 l.x = nodeX[0];
16 l.y = nodeY[0];
17 l.nodeNumber.text = vertexNumber[0];
18 n = new node();
19 addChild(n);
20 n.x = nodeX[armsNumber1 + 1] = 600;
21 n.y = nodeY[armsNumber1 + 1] = 350;
22 l = new nodelabel();
23 addChild(l);
24 l.x = nodeX[armsNumber1 + 1];
```

```
25 l.y = nodeY[armsNumber1 + 1];
26 l.nodeNumber.text = vertexNumber[armsNumber1 + 1];
27 n = new node();
28 addChild(n);
29 n.x = nodeX[totalVertex - 1] = 400;
30 n.y = nodeY[totalVertex - 1] = 150;
31 l = new nodelabel();
32 addChild(l);
33 l.x = nodeX[totalVertex - 1];
34 l.y = nodeY[totalVertex - 1];
35 l.nodeNumber.text = vertexNumber[totalVertex - 1];
36 var ranX:Number;
37 var ranY:Number;
38 var sign:Boolean = false;
39 for(i = 1; i <= armsNumber1; i++){
40         ranX = Math.random() * (300 - 100) + 100;
41         trace("X : " + ranX.toString());
42         nodeX[i] = ranX;
43         if(sign){
44                 ranY = 350 + Math.sqrt( 10000 -
45                                         ((ranX - 200)*(ranX - 200)));
46                 sign = !sign;
47         }
48         else{
49                 ranY = 350 - Math.sqrt( 10000 -
50                                         ((ranX - 200)*(ranX - 200)));
51                 sign = !sign;
52         }
53         nodeY[i] = ranY;
54         n = new node();
55         addChild(n);
56         n.x = nodeX[i];
57         n.y = nodeY[i];
58         l = new nodelabel();
59         addChild(l);
60         l.x = nodeX[i];
61         l.y = nodeY[i];
62         l.nodeNumber.text = vertexNumber[i];
63         this.graphics.moveTo(nodeX[0], nodeY[0]);
```

```
64          this.graphics.lineTo(nodeX[i], nodeY[i]);
65          e = new edgelabel();
66          addChild(e);
67          e.x = (nodeX[0] + nodeX[i]) / 2;
68          e.y = (nodeY[0] + nodeY[i]) / 2;
69          e.edgeNumber.text = vertexNumber[i].toString();
70  }
71  for(i = (armsNumber1 + 2);
72          i < (armsNumber1 + 2) + armsNumber2; i++){
73          ranX = Math.random() * (700 - 500) + 500;
74          nodeX[i] = ranX;
75          if(sign){
76                  ranY = 350 - Math.sqrt( 10000 -
77                                          ((ranX - 600)*(ranX - 600)));
78                  sign = !sign;
79          }
80          else{
81                  ranY = 350 + Math.sqrt( 10000 -
82                                          ((ranX - 600)*(ranX - 600)));
83                  sign = !sign;
84          }
85          nodeY[i] = ranY;
86          n = new node();
87          addChild(n);
88          n.x = nodeX[i];
89          n.y = nodeY[i];
90          l = new nodelabel();
91          addChild(l);
92          l.x = nodeX[i];
93          l.y = nodeY[i];
94          l.nodeNumber.text = vertexNumber[i];
95          this.graphics.moveTo(nodeX[armsNumber1 + 1],
96                                                  nodeY[armsNumber1 + 1]
97          this.graphics.lineTo(nodeX[i], nodeY[i]);
98          e = new edgelabel();
99          addChild(e);
100         e.x = (nodeX[armsNumber1 + 1] + nodeX[i]) / 2;
101         e.y = (nodeY[armsNumber1 + 1] + nodeY[i]) / 2;
102         e.edgeNumber.text=(-vertexNumber[armsNumber1+1]
```

```
103                                             + vertexNumber[i]).toString();
104 }
105 var joinNode1:int;
106 var joinNode2:int;
107 for(i = 0; i < totalVertex; i++){
108         if((vertexNumber[i]==(vertexNumber[totalVertex-1]
109                                             + join1)) && armsNumber1 != 0){
110                 joinNode1 = i;
111         }
112         if((vertexNumber[i]==(vertexNumber[totalVertex-1]
113                                             + join2)) && armsNumber2 != 0){
114                 joinNode2 = i;
115         }
116 }
117 if(armsNumber1 == 0){
118         joinNode1 = 0;
119 }
120 if(armsNumber2 == 0){
121         joinNode2 = armsNumber1 + 1;
122 }
123 var eText:Number;
124 this.graphics.moveTo(nodeX[totalVertex - 1],
125                                              nodeY[totalVertex - 1]);
126 this.graphics.lineTo(nodeX[joinNode1],
127                                              nodeY[joinNode1]);
128 e = new edgelabel();
129 addChild(e);
130 e.x = (nodeX[totalVertex - 1] + nodeX[joinNode1]) / 2;
131 e.y = (nodeY[totalVertex - 1] + nodeY[joinNode1]) / 2;
132 if((- vertexNumber[totalVertex - 1] +
133                                             vertexNumber[joinNode1]) >= 0){
134         e.edgeNumber.text=(-vertexNumber[totalVertex-1]
135                                     + vertexNumber[joinNode1]).toString();
136 }
137 else{
138         e.edgeNumber.text = ((- vertexNumber[totalVertex - 1]
139                             + vertexNumber[joinNode1]) * (-1)).toString();
140 }
141 this.graphics.moveTo(nodeX[totalVertex - 1],
```

```
142                                             nodeY[totalVertex - 1]);
143 this.graphics.lineTo(nodeX[joinNode2],
144                                             nodeY[joinNode2]);
145 e = new edgelabel();
146 addChild(e);
147 e.x = (nodeX[totalVertex - 1] + nodeX[joinNode2]) / 2;
148 e.y = (nodeY[totalVertex - 1] + nodeY[joinNode2]) / 2;
149 if((- vertexNumber[totalVertex - 1] + vertexNumber[joinNode2]) >= 0){
150         e.edgeNumber.text = (- vertexNumber[totalVertex - 1]
151                             + vertexNumber[joinNode2]).toString();
152 }
153 else{
154         eText = (- vertexNumber[totalVertex - 1] +
155                              vertexNumber[joinNode2]) * (-1);
156         e.edgeNumber.text = (eText).toString();
157 }
```

The following code will demonstrate a gracefully labeled Star

```
1  import flash.events.MouseEvent;
2 import flash.geom.Point;
3 import flash.events.Event;
4 import flash.utils.Timer;
5 import flash.events.TimerEvent;
6 var masterTimer:Timer = new Timer(1000);
7 masterTimer.addEventListener(TimerEvent.TIMER, time);
8 messageBox.text = "Enter a number between 1 to 17";
9 simulateButton.buttonMode = true;
10 simulateButton.addEventListener(MouseEvent.CLICK, checkLimit);
11 var vertexArray:Array;
12 var nodeX:Array;
13 var nodeY:Array;
14 var edgeArray:Array;
15 var nodeLabel:Array;
16 nodeX = new Array;
17 nodeY = new Array;
18 nodeX = [0, 400, 400, 600, 200, 400, 500, 500,
19                  300, 300, 350, 250, 450, 550, 550, 350, 250, 450];
20 nodeY = [0, 300, 500, 300, 300, 100, 400, 200, 200,
21                  400, 450, 250, 150, 350, 250, 150, 350, 450];
```

```
22 var edgeX:Array = [0, 400, 400, 500, 300, 400, 450, 450, 325,
23                   325, 375, 300, 425, 500, 500, 375, 300, 425];
24 var edgeY:Array = [0, 300, 400, 300, 300, 200, 375, 225, 225,
25                   375, 425, 275, 175, 325, 275, 175, 325, 425];
26 var vertex:int;
27 var nodeDrawn:int = 0;
28 var currentNode:int = 1;
29 var i:int;
30 var l:nodelabel;
31 var e:edgelabel;
32 var n:node;
33 function checkLimit(e:MouseEvent):void{
34         vertex = Number(noOfHands.text);
35         if(vertex < 1 || vertex > 17){
36                 messageBox.text = "NOT IN LIMIT";
37         }
38         else{
39                 simulate();
40         }
41 }
42 function simulate():void{
43         simulateButton.visible = false;
44         noOfHands.visible = false;
45         vertexInput.visible = false;
46         messageBox.visible = false;
47         vertexArray = new Array;
48         edgeArray = new Array;
49         nodeLabel = new Array;
50         for(i = 1; i <= vertex; i++){
51                 vertexArray[i] = 0;
52                 if(i < vertex){
53                         edgeArray[i] = 0;
54                 }
55         }
56         n = new node();
57         addChild(n);
58         n.x = nodeX[1] = 400;
59         n.y = nodeY[1] = 300;
60         nodeDrawn++;
```

```
61          l = new nodelabel();
62          addChild(l);
63          l.x = nodeX[1];
64          l.y = nodeY[1];
65          l.nodeNumber.text = vertex.toString();
66          vertexArray[1] = vertex;
67          this.graphics.lineStyle(2);
68          this.graphics.beginFill(0xFFFFFF);
69          masterTimer.start();
70  }
71  function time(event:TimerEvent):void{
72          if(nodeDrawn < vertex){
73                  n = new node();
74                  addChild(n);
75                  nodeDrawn++;
76                  n.x = nodeX[nodeDrawn];
77                  n.y = nodeY[nodeDrawn];
78                  this.graphics.moveTo(nodeX[1], nodeY[1]);
79                  this.graphics.lineTo(nodeX[nodeDrawn],
80                                  nodeY[nodeDrawn]);
81                  l = new nodelabel();
82                  addChild(l);
83                  l.x = nodeX[nodeDrawn];
84                  l.y = nodeY[nodeDrawn];
85                  l.nodeNumber.text = (nodeDrawn - 1).toString();
86                  e = new edgelabel();
87                  addChild(e);
88                  e.x = edgeX[nodeDrawn];
89                  e.y = edgeY[nodeDrawn];
90                  e.edgeNumber.text = (
91                  vertex - nodeDrawn + 1).toString();
92          }
93          else{
94                  masterTimer.stop();
95          }
96  }
```