The Road Coloring Problem

by

Naira Noor Rafiquee

Shaista Manzoor

Syed Azka Manzoor

Department of Mathematics

Faculty of Science

Islamic University of Science and Technology

Awantipora

Jammu and Kashmir

India

Project undertaken

At

Cluster Innovation Centre

University of Delhi

(January-March 2016)

Report submitted in partial fulfillment for the requirements of the degree of BSc in Actuarial and Financial mathematics in the Islamic University of Science and Technology, Awantipora Kashmir.

Certificate of Originality

The work embodied in this report entitled **"The Road Coloring Problem"** has been carried out by us at the **Cluster Innovation Centre, University of Delhi** under the supervision of **Prof Shobha Bagai**. I declare that the work and language included in this project report is free from any hind of plagiarism.

The work submitted is original and has not been submitted earlier to any institute or university for the award of any degree or diploma.

(**Name and signature**)

Acknowledgements

We would like to thank Professor Madan Mohan Chaturvedi, director Cluster Innovation Centre, Delhi University for providing us with an opportunity to work in the institution. We especially are indebted to Professor Shobha Bagai, coordinator of our project for helping us at every stage and giving right directions to our ideas. We would like to thank Miranda House Hostel for accommodating us during our stay. We would also like to pay regards to Mr. Abhijeet and Mr. Mayank Jain for helping us with the program terminology. We also give our regards to all the staff members of the institution who directly or indirectly helped us.

Abstract

This project deals with an interesting concept in the field of graph theory i.e. the road coloring problem that was conjectured in the year 1970 by Adler, Goodwyn and Benjamin Weiss. The problem was applied on *finite*, *strongly connected directed* graph that had constant *out-degree* and were *aperiodic*. The project deals with finding synchronized paths for vertices. Synchronized paths are paths that would lead to the same vertex no matter what the initial vertex is selected. An example of the problem given in the paper "The Road Coloring Problem" by Weifu Wang was studied and was solved using mat lab. The problem was applied on the roads around the Cluster Innovation Centre. The graph was formulated and various trees, cycles, paths arising from the graph were studied.

Key words: Finite, strongly connected, directed, aperiodic graphs, out-degree, road coloring problem

Contents

1. Introduction to Road Coloring Problem (RCP)

1.1 Introduction

1.2 Basic Terminology

(a) Graphs

(b) Trees

2. Literature Survey

3. Understanding the RCP

3.1 Introduction

3.2 Generation of trees

3.3 Generation of cycles and paths

3.4 Finding synchronized paths

4. Application of RCP-roads around DU

5. Further work

References

Appendix

Chapter 1

Introduction to Road Coloring Problem (RCP)

1.1 Introduction

The field of Graph Coloring came into being in the year 1852 when the Four Color Problem was postulated. Over the years it has developed into one of the most popular areas of Graph theory. Graph Coloring has a wide variety of applications to problems involving scheduling and assignments. Road Coloring is one of the recent developments in this field.

The Road Coloring theorem was conjectured in the year 1970 by Roy Adler, Goodwyn and Benjamin Weiss in [2] and was proved by Avraham Trahtman in [5] 2009. It is an edge coloring problem. The theorem is applicable for *strongly connected*, directed *finite* graphs that are *out regular* and *aperiodic*. It can be stated as, for a graph that has uniform out-degree and is aperiodic, there should exist a coloring for the edges such that for a certain sequence, regardless of the initial vertex, following the sequence of color will always lead to the same vertex. This theorem is extremely useful in the theory of automaton and symbolic dynamics.

1.2 Basic Terminology

(a) Graphs

Mathematicians have always tried to make things look simpler, be it squeezing observations into a theorem or making tables for huge data. In a similar attempt to make contacts among things/people visible graphs came into being. Graphs leave a longer impression than numbers as they communicate information visually. For this reason, graphs are often used in newspapers, magazines and businesses around

the world. Graphs deal with discrete type of data where vertices/nodes represent things and edges the connections.

Definition 1: A **graph** $G = (V, E)$ consists of V, a nonempty set of vertices & E, a set of edges where each edge has either one or two vertices associated with it, called its endpoints.

Definition 2: Graphs can be **finite** or **infinite** depending upon the elements in the vertex set/edge set. When the number of vertices/edges is finite then the graph is termed as a finite graph, similarly when vertices/edges are infinite the graph is infinite.

Definition 3: A graph is said to be **directed** when each edge has a specific direction. The directed edge associated with the ordered pair (u, v) is said to start at u and end at v.

Definition 4: A directed graph is said to be **aperiodic** if there is no integer $k > 1$ that divides the length of every cycle of the graph. In other words the greatest common divisor (*gcd*) of lengths of all its cycles is 1.

Definition 5: The vertices that are joined together with an edge are termed as **adjacent** while as the edge is called **incident** to the vertices.

Definition 6: The **degree** of a vertex of an undirected graph is the number of edges incident to the vertex, with loops counted twice. While for directed graphs, concepts like in and out degree are more viable, as the name suggests **in-degree** refers to the number of edges that are incoming and **out-degree** the number of edges that are outgoing at a vertex.

Definition 7: A graph is said to be **out-regular** if all its vertices have the same out-degree.

Definition 8: The **adjacency matrix** A of G is the $n*n$ zero-one matrix with 1 as it's $(i, j)^{th}$ entry when v_i and v_j are adjacent and 0 as its $(i, j)^{th}$ entry when they are not adjacent. Mathematically, $A = [a_{ij}]$ then

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of G,} \\ 0 & \text{otherwise.} \end{cases}$$

When algorithms are carried upon graphs the computations become cumbersome, adjacency matrices are relatively easier to comprehend rather than graphs.

Definition 9: A **path** is a finite sequence of edges which connects vertices that are all distinct from one another. Mathematically, a path of length n (n is a positive integer) can be defined as a sequence of n edges $e_1,e_2,\ldots,e_n$ 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_j has (for $j=1,\ldots,n$) the endpoints x_{i-1} and x_i. A path is a circuit if it begins and ends at the same vertex, i.e. if $u = v$ and has length greater than zero.

Definition 10: A path or circuit is said to be **simple** if it does not contain the same edge more than once. When there is a path between every distinct pair of vertices in a graph then such a graph is termed as connected. In road coloring problems we usually encounter graphs that are **strongly connected**, which means there is a path from a to b and from b to a whenever a and b are vertices in the graph.

Definition 11: **Planar graphs** are defined as the graphs that can be drawn in a plane without any edge crossing.

(b) Trees

Trees were first used by an English mathematician Arthur Cayley in the year 1857 for representing chemical compounds, which gave new dimensions to the field of organic chemistry. Trees are extensively used in computer science where they are used in creating algorithms. For instance, they are used to construct algorithms for locating items in a list.

Definition 1: **Trees** are connected undirected graphs that have no simple circuits i.e. it does not possess any kind of loop or circuit. Trees have a unique simple path between two vertices

Definition 2: Trees are said to be **rooted** if a specific node is designated as root of the tree and all the other edges are heading away from it. In trees there is exactly one parent of every child and the flow is either from top to bottom or vice versa.

Example of a rooted tree;

Organizations can be represented using rooted trees. Each vertex represents a position in the organization. The initial position of an edge represents higher authority and terminal it's subordinate.

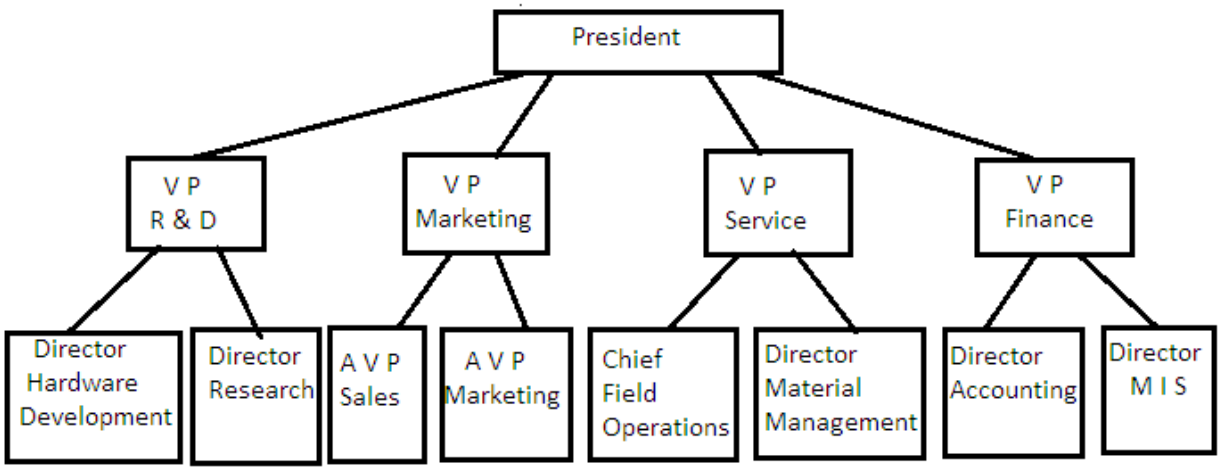


Figure 1

Definition 3: A vertex of a rooted tree is said to be a **leaf** if it has no child. While vertices that have children are called **internal vertices**. A tree with *n* vertices has *n-1* edges.

.

Chapter 2

Literature Survey

Graph coloring is a special case of graph labeling, where colors are assigned to the elements of a graph subjected to certain constraints. Coloring is to be done in a way that no two adjacent vertices/edges/planes share the same color. Graphs in which vertices are colored is called vertex coloring. Similarly, edge coloring refers to coloring of edges and face coloring to planes. The idea of graph coloring dates back to 1852 when a student of Augustus De Morgan, Francis Guthrie, noticed that the provinces in England could be colored using four different colors so that no adjacent province had the same color. He conjectured that the four color theorem was true i.e. given any separation of a plane into adjacent regions no more than four colors are required to color the plane. De Morgan publicized this idea throughout the mathematical community. The four color theorem was proved by American mathematicians Kenneth Appel and Wolfgang Haken [1] in the year 1976, before that a couple of incorrect proofs were published.

Another concept called the **Road Coloring problem** came into being in the year 1970. It was posed by Benjamin Weiss and Roy Adler [2], the problem revolves around the fact that given a finite number of roads, one should be able to draw a map, coded in various colors, which leads to a certain destination regardless of the initial point. A number of mathematicians in different eras tried to prove this theorem. One among them was G.L.Obrien who published his paper titled “The Road-Coloring Problem” in the year 1981 [3]. In his paper he stated that “if G is a finite strongly connected aperiodic directed graph with no multiple edges, and G contains a simple cycle of prime length which is a proper subset of G then G has a synchronizing coloring”, but no formal proof was given by him. Similarly in the year 1990, Joel Friedman wrote a paper [4] titled “On The Road Coloring

Problem" in which he took the graphs with out-degree 2 into consideration, then he colored the graph using red & blue in such a way that each vertex had one red edge and one blue edge leaving it. In 2005 a paper titled "A Min-Max theorem about the Road Coloring" was given by Rajneesh Hedge and Kamal Jain in [8]. Finally in the year 2009 the road coloring theorem was proved by an Israeli mathematician Avraham N. Trahtman using a number of theorems and lemmas in [5] which are mentioned in section 3.1. Trahtman's proof leads to an algorithm that finds a synchronized labeling with a cubic worst-case time complexity. In another paper titled "A quadratic algorithm for the Road Coloring Problem" given by Marie-Pierre Béal, Dominique Perrin in 2014 in which they changed cubic worst-case complexity to quadratic in time and linear in space. They also extended the Road Coloring Theorem to the periodic case.

Chapter 3

Understanding the RCP

3.1 Introduction

As we have already discussed the road coloring problem in chapter 1, we now know that for every graph that is directed, finite, strongly connected, out-regular and aperiodic there must exist a synchronizing coloring. Now we will apply this theorem on a graph that is given in the paper [6] titled "The Road Coloring Problem" by Weifu Wang and is also discussed in Wikipedia [7]. Weifu Wang has given synchronizing coloring for two vertices of the graph (*B* & *D*), now we will find the sequence for the rest of the vertices. For finding such sequences we require trees, paths and cycles of the vertices of the graph. The problem is to find out a sequence of colors using which one can reach to a destination from any other point within a network regardless of the initial point. Avraham in [5] with his series of lemmas and theorems guarantees the existence of synchronizing coloring for the graphs that satisfy constraints of RCP. The lemmas and theorems used by Avraham to prove the theorem are stated below;

Theorem 1: There exists a partition of Γ on F-maximal sets (of the same weight)

Theorem 2: Let us consider a coloring of AGW graph Γ. Stability of states is a binary relation on the set of states of the obtained automaton; denote this relation by ρ. Then ρ is a congruence relation, Γ/ρ presents an AGW graph and synchronizing coloring of Γ/ρ implies synchronizing recoloring of Γ.

Lemma 1: Let w be the weight of F-maximal set of the AGW graph Γ via some coloring. Then the size of every F-clique of the coloring is the same and equal to $w(\Gamma)/w$ (the size of partition of Γ on F-maximal sets).

Lemma 2: Let F be F-clique via some coloring of AGW graph Γ. For any word s the set $F s$ is also an F-clique and any state [vertex] p belongs to some F-clique.

Lemma 3: Let A and B ($|A| > 1$) be distinct F-cliques via some coloring without stable pairs of the AGW graph Γ. Then $|A|-|A \cap B| = |B|-|A \cap B| > 1$.

Lemma 4: Let some vertex of AGW graph Γ have two incoming bunches. Then any coloring of Γ has a stable couple.

Lemma 5: Let N be a set of vertices of level n from some tree of the spanning subgraph S of AGW graph Γ. Then in a coloring of Γ where all edges of S have the same color α, any F-clique F satisfies $|F \cap N| \leq 1$.

Lemma 6: Let AGW graph Γ have a spanning subgraph R of only disjoint cycles (without trees). Then Γ also has another spanning subgraph with exactly one vertex of maximal positive level.

Lemma 7: Let any vertex of an AGW graph Γ have no two incoming bunches. Then Γ has a spanning subgraph such that all its vertices of maximal positive level belong to one nontrivial tree

Theorem 3: Any AGW graph Γ has a coloring with stable couples.

Theorem 4: Every AGW graph Γ has synchronizing coloring.

Now, we will check whether the graph fits in the road coloring problem or not.

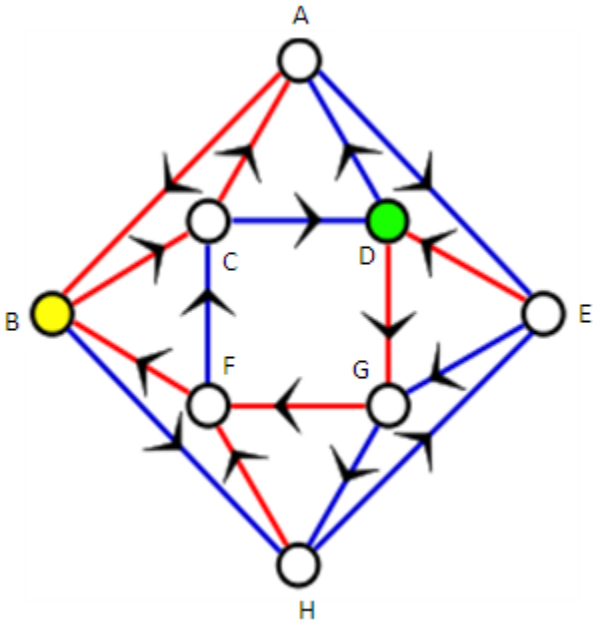


Figure 1

The above graph is finite as it contains 8 vertices and 16 edges. The out-degree at each vertex is two which makes it out-regular. The graph is strongly connected as there exists more than one path between each pair of vertices. In order to find whether this graph is aperiodic or not, we need to show that the gcd of lengths of all its cycles is one. The cycles *ABCA*, *AEGFCA* have lengths 3 and 5 respectively, so we can infer that the gcd of lengths of these two cycles is one. The gcd of lengths of all the cycles is one which is verified in section 3.3. Thus, Figure 1 satisfies all the constraints of RCP that guarantees the existence of synchronizing coloring for each vertex. To find that sequence we require cycles, paths and rooted trees of every vertex. Trees are drawn because it is relatively easier to concatenate cycles and paths through them.

3.2 Generation of trees

Algorithm for the program:

The number of vertices is represented by n and the out-degree by d. The tree is constructed in such a way that color 1 will always be encountered to the left side of the tree. For Figure 1 n =8 and d =2. The vertices *A, B,..., H* can be numerically represented as 1,2,…, 8.

Step 1 Construct the adjacency matrix of Figure 1. Denote red outward connection with 1 and blue outward connection with 2.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	1	0	0	2	0	0	0
2	0	0	1	0	0	0	0	2
3	1	0	0	2	0	0	0	0
4	2	0	0	0	0	0	1	0
5	0	0	0	1	0	0	2	0
6	0	1	2	0	0	0	0	0
7	0	0	0	0	0	1	0	2
8	0	0	0	0	2	1	0	0

Step 2 Create a 1x8 array, initialized with zeroes.

1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0

Step 3 Increment vertex 1 (*A*) by 1 in the array and keep rest of the entries as it is.

Currnode will point towards vertex 1 (Currnode is the current position of the cursor).

1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0

Step 4 Check at what position in the adjacency matrix row 1 corresponds to value 1, which is vertex 2 in our case.

Step 5 As the entry in the array at vertex 2 is 0, therefore, the currnode will shift to vertex 2. Now, add 1 to vertex 2 in the array.

1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	0

Step 6 Check at what position in the adjacency matrix row 2 corresponds to value 1, which is vertex 3 in this case.

Step 7 The entry in the array at vertex 3 is 0, therefore, the currnode will shift to vertex 3. Now, add 1 to vertex 3 in the array.

1	2	3	4	5	6	7	8
1	1	1	0	0	0	0	0

Step 8 Check at what position in the adjacency matrix row 3 corresponds to value 1, which is vertex 1.

Step 9 The entry in the array at vertex 3 is not 0, therefore, the currnode is not shifted to vertex 3 and remains at 3. Now, add 1 to vertex 3 in the array.

1	2	3	4	5	6	7	8
1	1	2	0	0	0	0	0

Step 10 Check at what position in the adjacency matrix row 3 corresponds to value 2, which is vertex 4 in our case.

Step 11 The entry in the array at vertex 4 is 0, therefore, the currnode will shift to vertex 4. Now, add 1 to vertex 4 in the array.

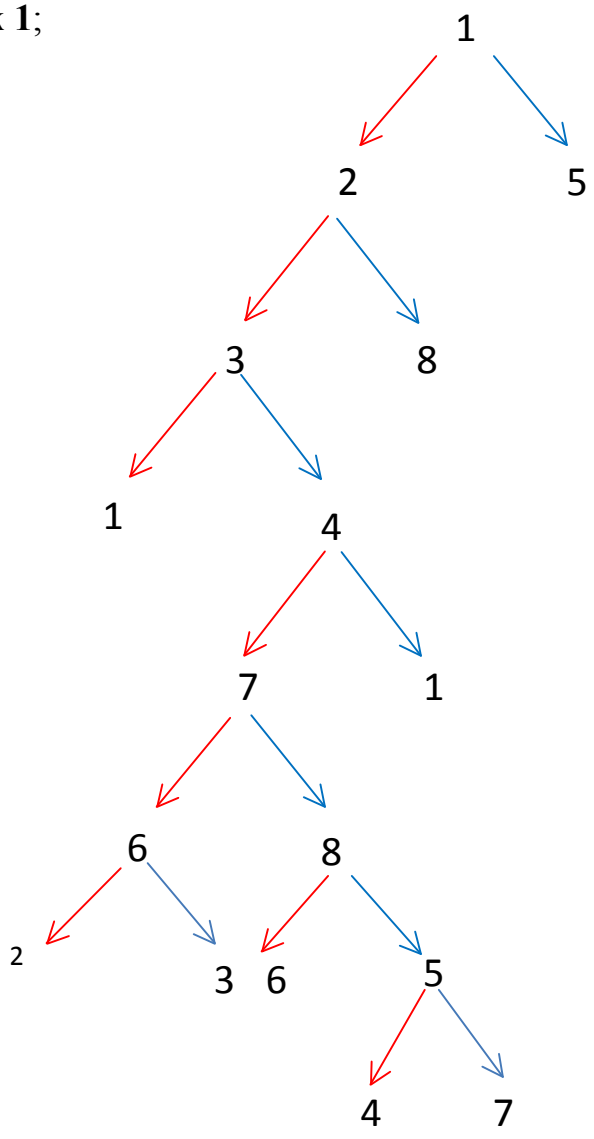
1	2	3	4	5	6	7	8
1	1	2	1	0	0	0	0

Step 12 Repeat the procedure until the sum of the all the entries in the array is $n * d = 8*2 = 16$.

(The out degree of the graph is 2, when each entry of the array equals to the out degree then the tree becomes complete i.e. each vertex has encountered all its children.)

1	2	3	4	5	6	7	8
2	2	2	2	2	2	2	2

The tree of vertex 1;



The above logic provided the tree of vertex 1, similarly trees of other vertices can be obtained by incrementing that vertex in step (3). The trees for the other vertices is given in Appendix 1.

3.3 Generation of cycles and paths

Algorithms for the program:

The cycles of a vertex can be found with the help of its tree. Steps for finding cycles:

Step 1 Start with a vertex (say) *X,* follow color 1.

Step 2 Keep following color 1 till there is repetition. If any entry other than initial vertex is repeated then go back to the previous entry (say) *U* and replace color 1 with 2.

Step 3 At vertex *U* follow color 1 and repeat step 2.

For example, cycles of vertex 8;

8 (1) 6 (1) **2** (1) 3(1) 1 (1) **2** **Rejected**

The above path is rejected as the entry 2 is repeated which will lead to the formation of cycle 2-3-1-2 and the algorithm will repeat the cycle of 2, thus not providing us the cycle of 8.

8 (1) **6** (1) 2 (1) 3 (1) 1 (2) 5 (1) 4 (1) 7 (1) **6** **Rejected**

8 (1) 6 (1) 2 (1) 3 (1) 1 (2) 5 (1) 4 (1) 7 (2) **8** **Accepted**

8 (1) 6 (1) 2 (1) 3(1) **1** (2) 5 (1) 4 (2) **1** **Rejected**

8 (1) **6** (1) 2 (1) 3 (1) 1 (2) 5 (2) 7 (1) **6** **Rejected**

8 (1) 6 (1) 2 (1) 3 (1) 1 (2) 5 (2) 7 (2) **8** **Accepted**

and so on.

In a similar manner paths from 1 to 2, 1 to 3,…, 1 to 8, 2 to 1, 2 to 3,…, 2 to 8,…,8 to 7 can be calculated.

The cycles and paths obtained using mat lab for vertex 1 are given below:

For $i = 1$

Cycle	**Length of cycle**
1 2 3 1	3
1 2 3 4 1	4
1 2 8 6 3 1	5
1 2 8 6 3 4 1	6
1 2 8 5 4 7 6 3 1	8
1 2 8 5 4 1	5
1 2 8 5 7 6 3 1	7
1 2 8 5 7 6 3 4 1	8
1 5 4 7 6 2 3 1	7
1 5 4 7 6 3 1	6
1 5 4 7 8 6 2 3 1	8
1 5 4 7 8 6 3 1	7
1 5 4 1	3
1 5 7 6 2 3 1	6
1 5 7 6 2 3 4 1	7
1 5 7 6 3 1	5
1 5 7 6 3 4 1	6
1 5 7 8 6 2 3 1	7
1 5 7 8 6 2 3 4 1	8
1 5 7 8 6 3 1	6
1 5 7 8 6 3 4 1	7

The cycles for rest of the vertices are given in Appendix 2.

The length of cycles are 3,4,5,7,….,8,3. Thus we can conclude that the gcd of lengths of all the cycles is one.

Paths

For $i = 1$ to $j = 2$

1	2					
1	5	4	7	6	2	
1	5	4	7	8	6	2
1	5	7	6	2		
1	5	7	8	6	2	

The rest of the paths are given in Appendix 3.

3.4 Finding synchronized paths

Up until now all the trees, paths and cycles are obtained with the help of above algorithms. Next step is to find out a sequence of colors that is to be followed such that no matter what the initial point is, the sequence will lead to the target. An idea that can be applied to find such a sequence is a common path amongst the various paths towards a particular vertex. For instance, let destination be vertex *H* i.e. we need to find out a sequence that will lead us to *H* regardless of our starting point. For this, we will calculate all the paths from every vertex (other than *H*) to *H*. Paths from *A* to *H* will be represented as *AH* and paths from *B* to *H* will be denoted by *BH*. The common path between *AH*, *BH*, *CH*,…,*GH* will be the desired synchronizing path for the vertex *H*. Very often there aren't such common paths so an alternative is to be used.

We can also find such synchronized paths using concatenation i.e. combining paths with cycles. This concept will be elaborated with the following example;

Step 1 Select a path (say) *A*-*B*, right concatenate the path with a cycle of *B* (say) B_1.

Step 2 Check whether the new path *A*-*B*-*B* is synchronized i.e. will it lead all the other vertices to *B*.

Step 3 If *A*-*B*-*B* is not a synchronized coloring then concatenate other cycles of *B* ($B_2, B_3, \ldots, B_n$) to the path *A*-*B* one by one then repeat step 2.

Step 4 In case if no synchronized coloring is obtained then switch over to some other path from *A* to *B* and again start right concatenation of cycles of *B*.

Step 5 Repeat the process till a synchronizing coloring is achieved.

The synchronizing paths of each vertex calculated manually for Figure 1 are given below:

A blue-blue-red-blue-blue-red-blue-blue-red-blue

The above synchronizing coloring was obtained by right concatenation of the path *B*-*H*-*E*-*D*-*A* with a cycle *A*-*E*-*D*-*A* of the vertex *A*. Similarly, rest of the sequences where found by concatenation.

B blue-red-red-blue-red-red-blue-red-red

C blue-blue-red-blue-blue-red-red-blue-red-red-red

D blue-blue-red-blue-blue-red-blue-blue-red

E red-blue-blue-red-blue-blue-red-blue-blue

F blue-red-blue-red-blue-red-red-blue-red

G blue-blue-blue-red-blue-blue-red-blue-blue-red-red

H red-red-blue-red-red-blue-red-red-blue

Chapter 4

Application of RCP-roads around DU

With the help of all the concepts regarding RCP we can now move a step further by applying it on real life problems. We will consider an actual road and try to fit in the road coloring problem. Let us take the key roads of Delhi University around the Cluster Innovation Centre as our sample. The sample can be taken from the Google map.

Figure 2: Roads around CIC

The road coloring problem can't be applied on the above map so we will convert this into a graph that will be easier to comprehend.

The points at which roads intersect will be marked as vertices and edges will represent roads. All the roads between the vertices are two-way. The graph will look like;

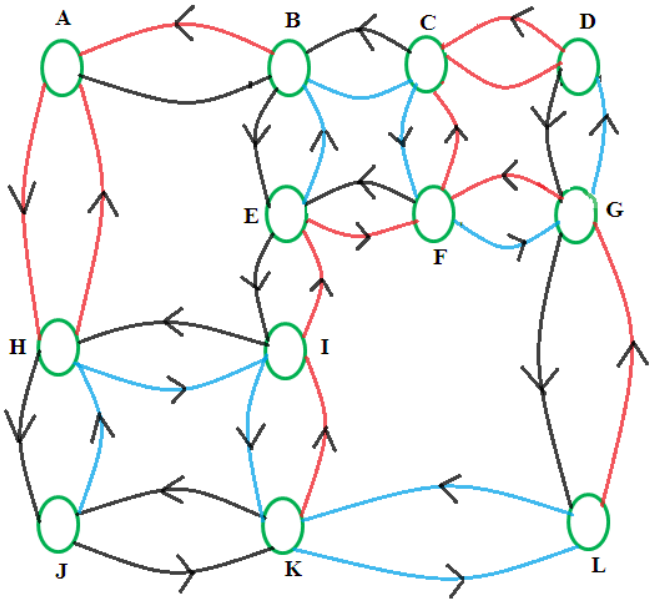


Figure 3

Now we will check whether the graph satisfies the conditions of RCP. The graph consists of 12 vertices and 32 edges which implies it is a finite graph. It is strongly connected as more than one edge connects each pair of vertices. Now, in order to apply RCP to this problem we need to show that the graph is out-regular and aperiodic.

For out-regularity;

The out-degree of each vertex is given below:

A	**2**	*E*	3	*I*	*3*
B	3	*F*	3	*J*	2
C	3	*G*	3	*K*	3
D	**2**	*H*	3	*L*	**2**

The graph is called out-regular when out-degree of each vertex is same. Here, the out-degree of the vertices isn't same. In order to make the out-degrees equal, we will add one edge at vertex *A* pointing towards *D* making ITS out-degree 3. Such a path can be added if there is a route from *A* to *D* directly or indirectly (i.e. in between vertices if any will be neglected). In a similar manner we can add paths from *D* to *L*, *L* to *J* and *J* to *A* so that all vertices have uniform out-degree (i.e. 3). Therefore, the graph becomes out-regular and will look like;

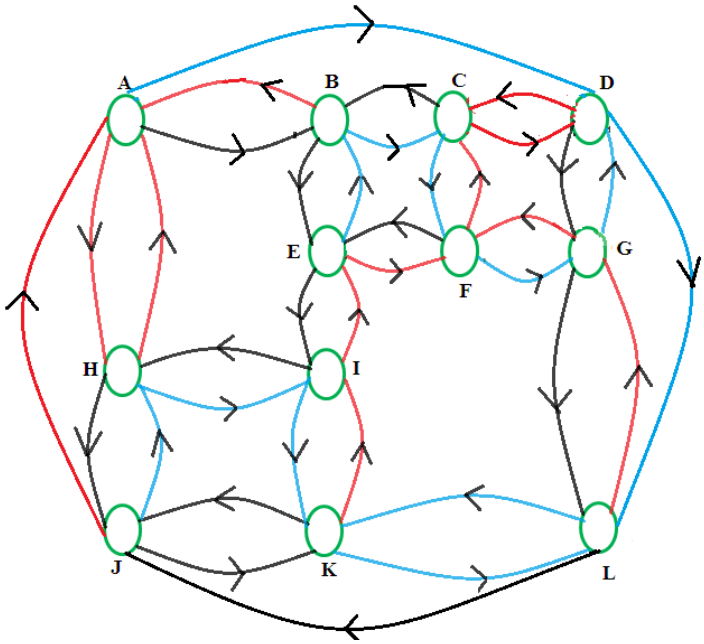
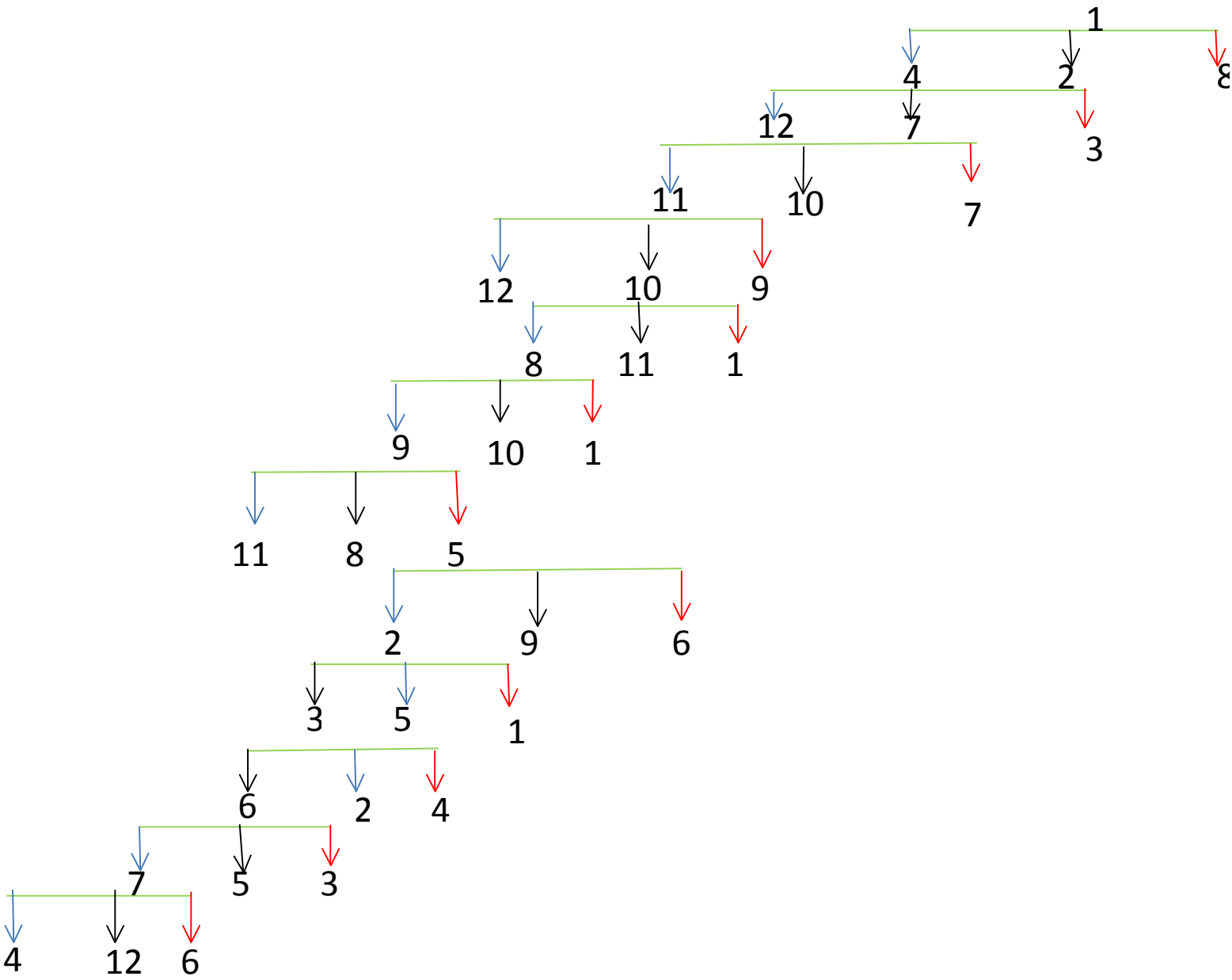


Figure 4

The cycles *ABEIHA*, *BDGLKIEB*, *EFGLKIE* have lengths 5, 7 and 6 respectively, so we can infer that gcd of lengths of these cycles is one. The rest of the cycles were calculated using mat lab and it was found that the gcd of the lengths of all its cycles was one. Therefore, the graph satisfies all the constraints of RCP which guarantees the existence of synchronizing paths for every vertex. In order to obtain the synchronized coloring we require the trees, cycles and paths for every vertex. Trees, cycles and paths for the graph can be obtained using the same algorithm that

was used in chapter 3 by replacing n with 12 and d with 3. The tree for vertex 1 will look like;



Finding synchronized paths

The synchronizing paths of each vertex that were obtained manually by concatenating paths and cycles for Figure 3 are given below:

A red-blue-black-blue-red-black-red

Path D-C-F-E-B-A

Cycle A-B-A

B red-black-red-black-black-black-black-red-black-red-black

Path A-H-J-A-B

Cycle1 B-E-I-H-A-B

Cycle2 B-A-B

C blue-blue-black-black-blue-red-black-blue-blue-black-blue-red-blue-red

Cycle1 C-F-G-L-J-H-A-B-C

Cycle2 C-F-E-B-A-D-C

D black-blue-red-blue-blue-black-red-blue

Path A-B-C-D

Cycle D-L-K-J-A-D

E blue-blue-blue-black-blue-blue-red-red-black

Path A-D-L-K-J-H-I-E

Cycle E-F-E

F blue-red-blue-red-blue-blue-blue-blue-blue-red-red-red

Path E-B-A-D-C-F

Cycle F-G-D-A-B-K-I-E-F

G black-black-black-red-blue-black-black-black-red-blue-black

Path B-E-I-H-A-D-G

Cycle G-L-J-A-D-G

H black-red-blue-blue-black-black-red-black-blue-blue-black-red-red

Path C-B-A-D-L-J-K-I-H

Cycle H-I-K-J-A-H

I red-black-black-blue-black-blue-blue-black-red-red-blue

Path C-D-G-L-K-J-H-I

Cycle I-H-A-H-I

J blue-blue-blue-red-black-black-red-black-blue-red-blue-black-black-black

Path A-D-L-K-I-H-J

Cycle1 J-L-C-D-L-J

Cycle2 J-K-J

K black-black-red-blue-black-blue-red-black-red-blue-blue-blue

Path C-B-E-F-G-L-K

Cycle K-I-H-A-D-L-K

L blue-black-red-blue-black-blue-blue

Path A-D-G-F-L

Cycle L-K-L

So we were able to find out synchronizing paths of all the vertices. Now we can positively conclude that RCP is applicable for real world problems.

Further work

In this project we worked on the programs of obtaining trees and cycles of vertices and paths between every pair of vertex. The programs are mentioned in the Appendix. The program for the synchronizing coloring could also have been prepared but due to the shortage of time the paths were obtained manually. Additional work can be done by providing a program for concatenation that will make it easier to find the synchronizing coloring. The road coloring problem can be used in some other fields as well. For instance, this theorem could be used in the theory of automaton and symbolic dynamics.

Reference

[1] Appel, Kenneth; Haken, Wolfgang (October 1977), "Solution of the Four Color Map Problem", *Scientific American* **237** (4), pp. 108–121

[2] R.L. Adler, B. Weiss. *Similarity of automorphisms of the torus*, Memoirs of the Amer. Math. Soc., Providence, RI, 98(1970).

[3] O'Brien, G.L.(1981), "The road-colouring problem", Israel Journal of Mathematics 39 (1-2): 145-154

[4] J. Friedman(1990), "On The Road Coloring Problem", Proceedings of the American Mathematical Society, 110(1990), 1133-1135.

[5] Trahtman, Avraham N. (2009), "The Road Coloring Problem", Israel Journal of Mathematics 172(1): 51-60

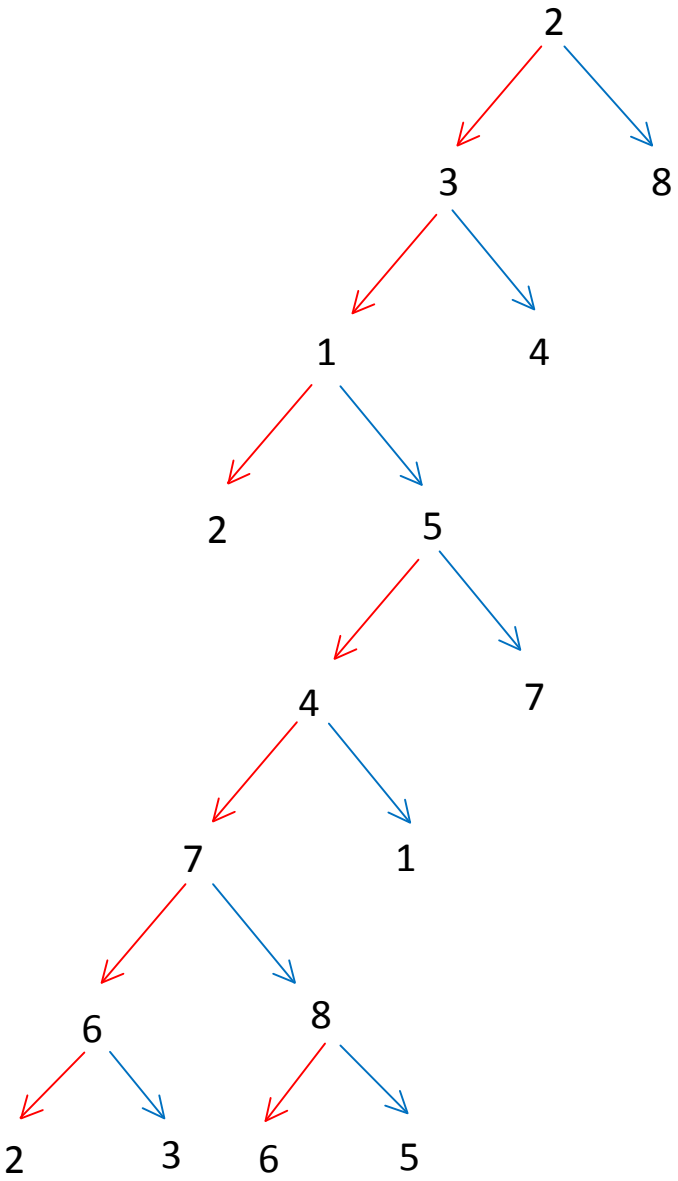
[6] Wang, Weifu, (2011), "The Road Coloring Problem" :1-6

[7] https://en.wikipedia.org/wiki/Road_coloring_theorem

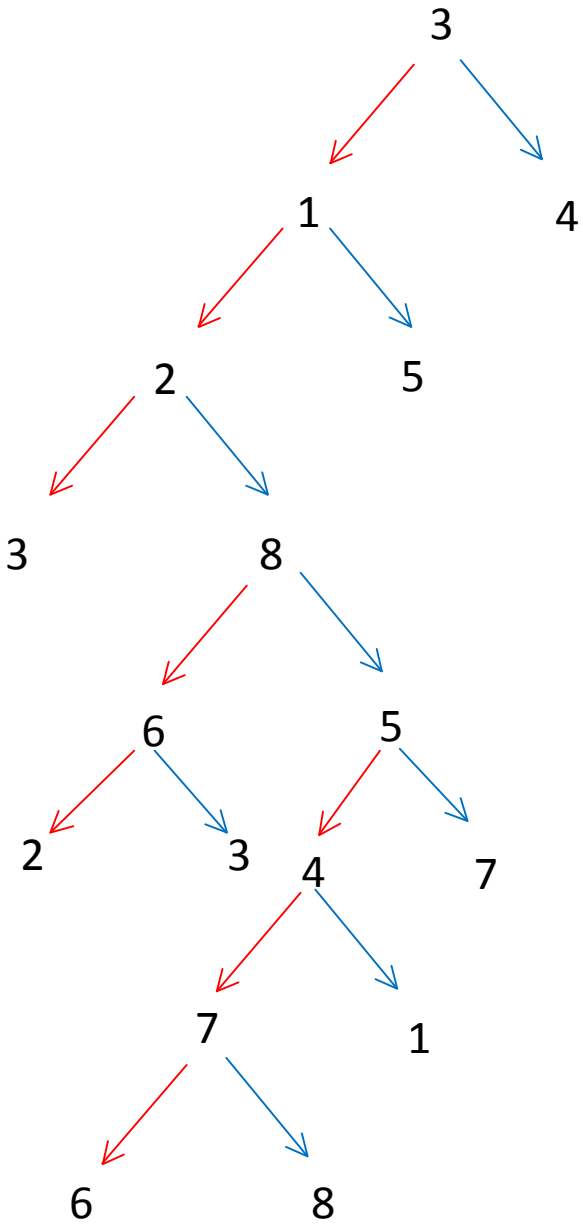
[8] Rossen, Kenneth H, "Discrete Mathematics and its Application", 7th edition, Mc Graw Hill, 2012

Appendix 1

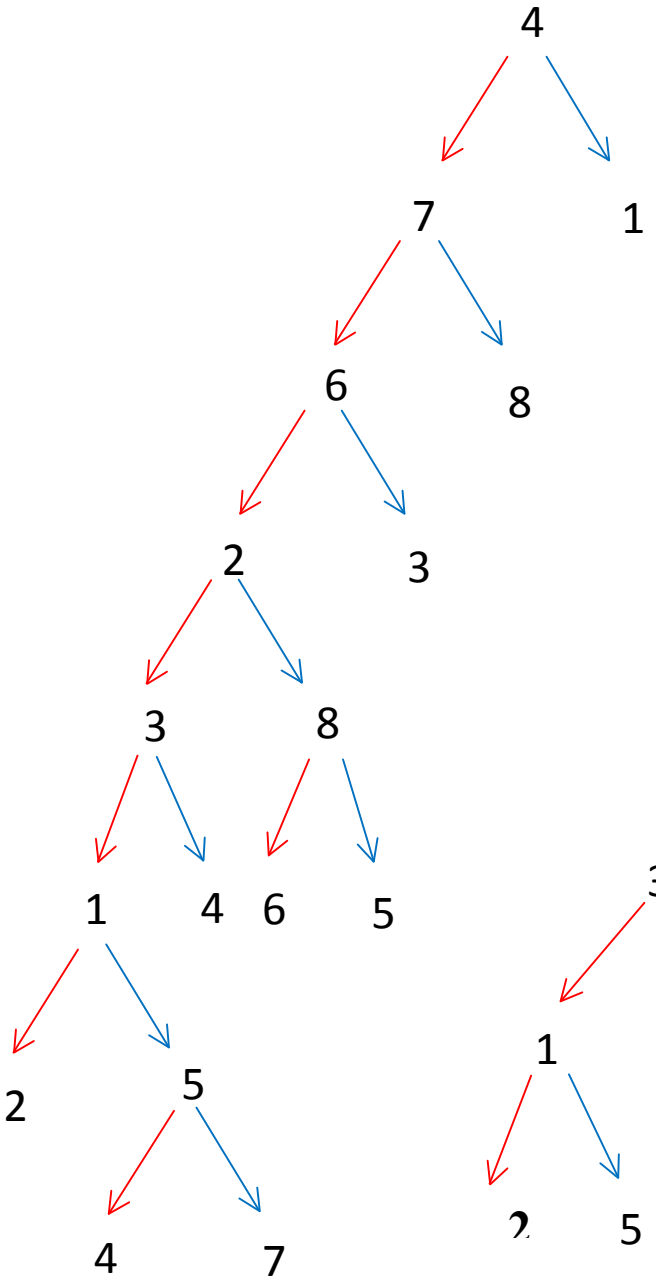
Tree of vertex 2



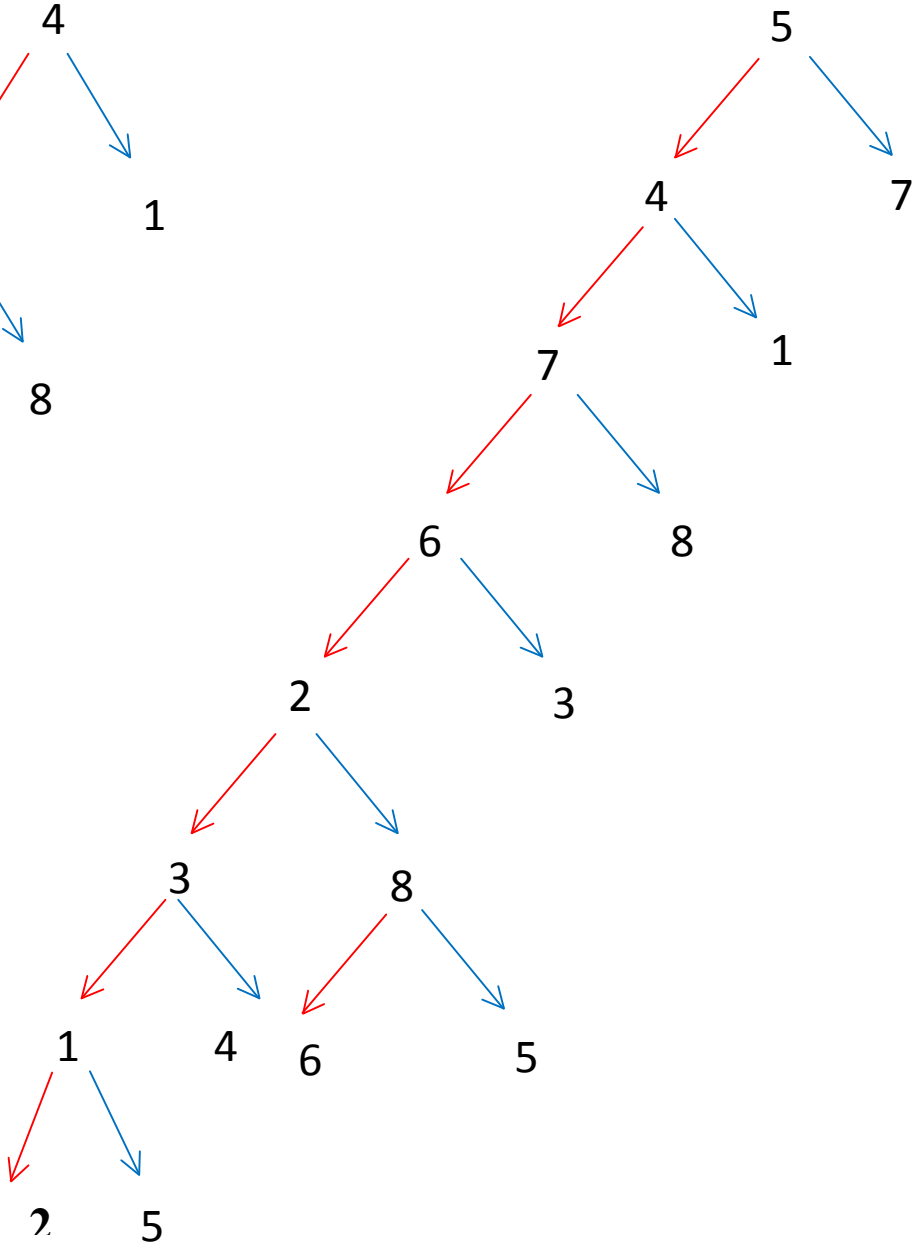
Tree of vertex 3



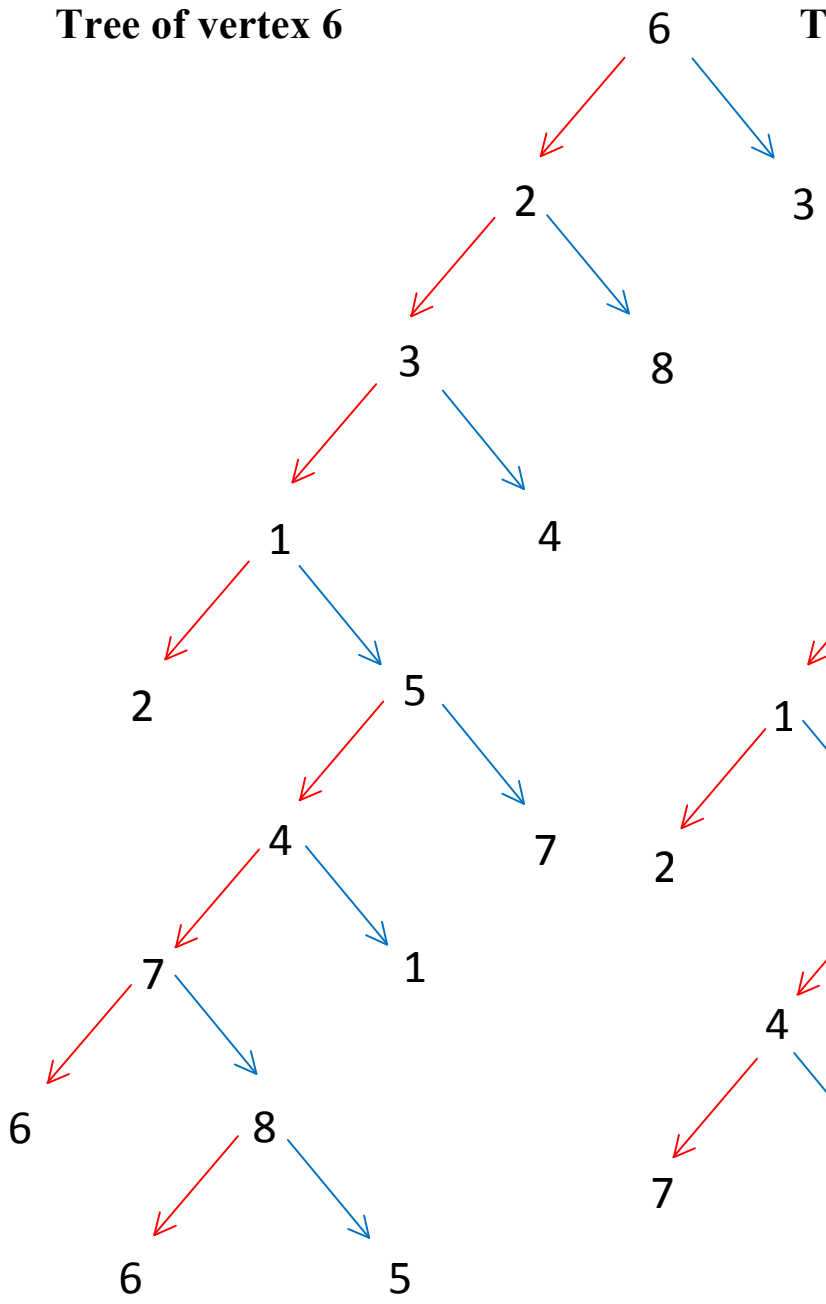
Tree of vertex 4



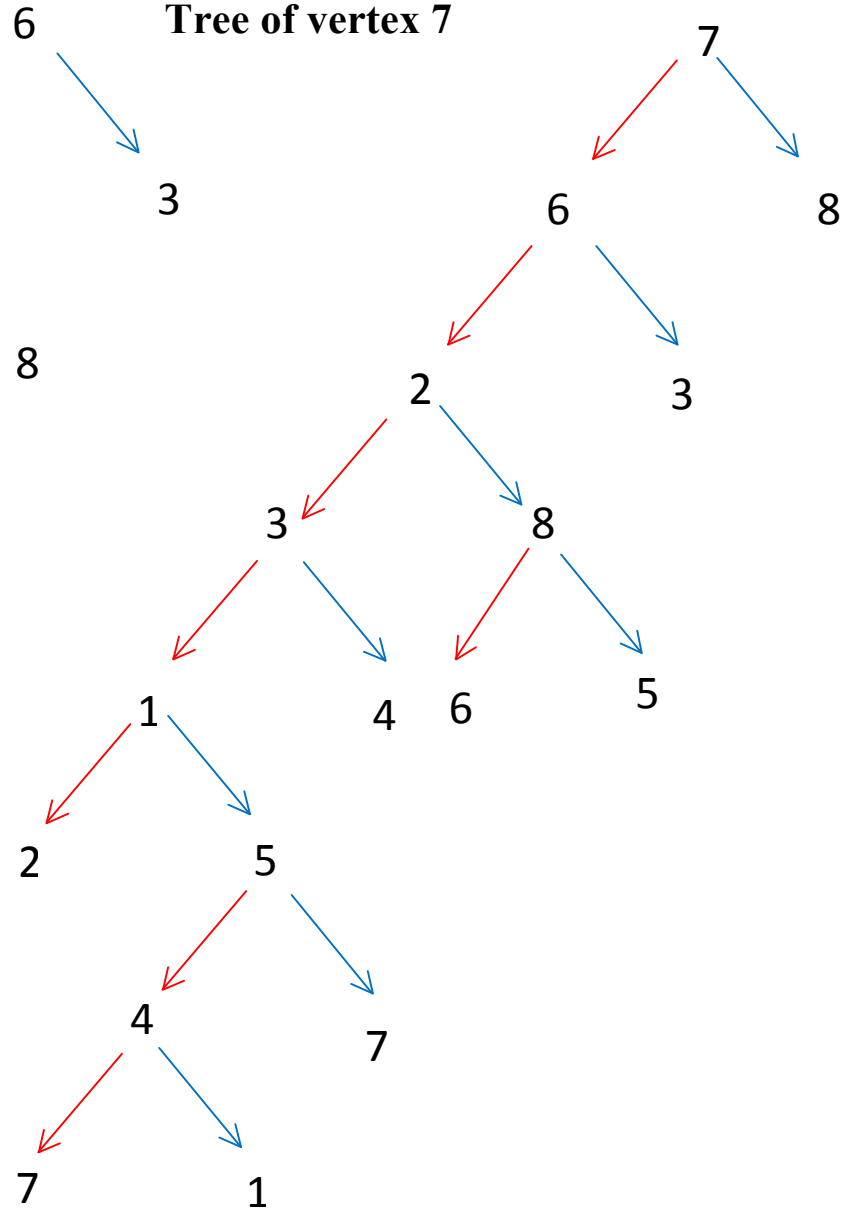
Tree of vertex 5



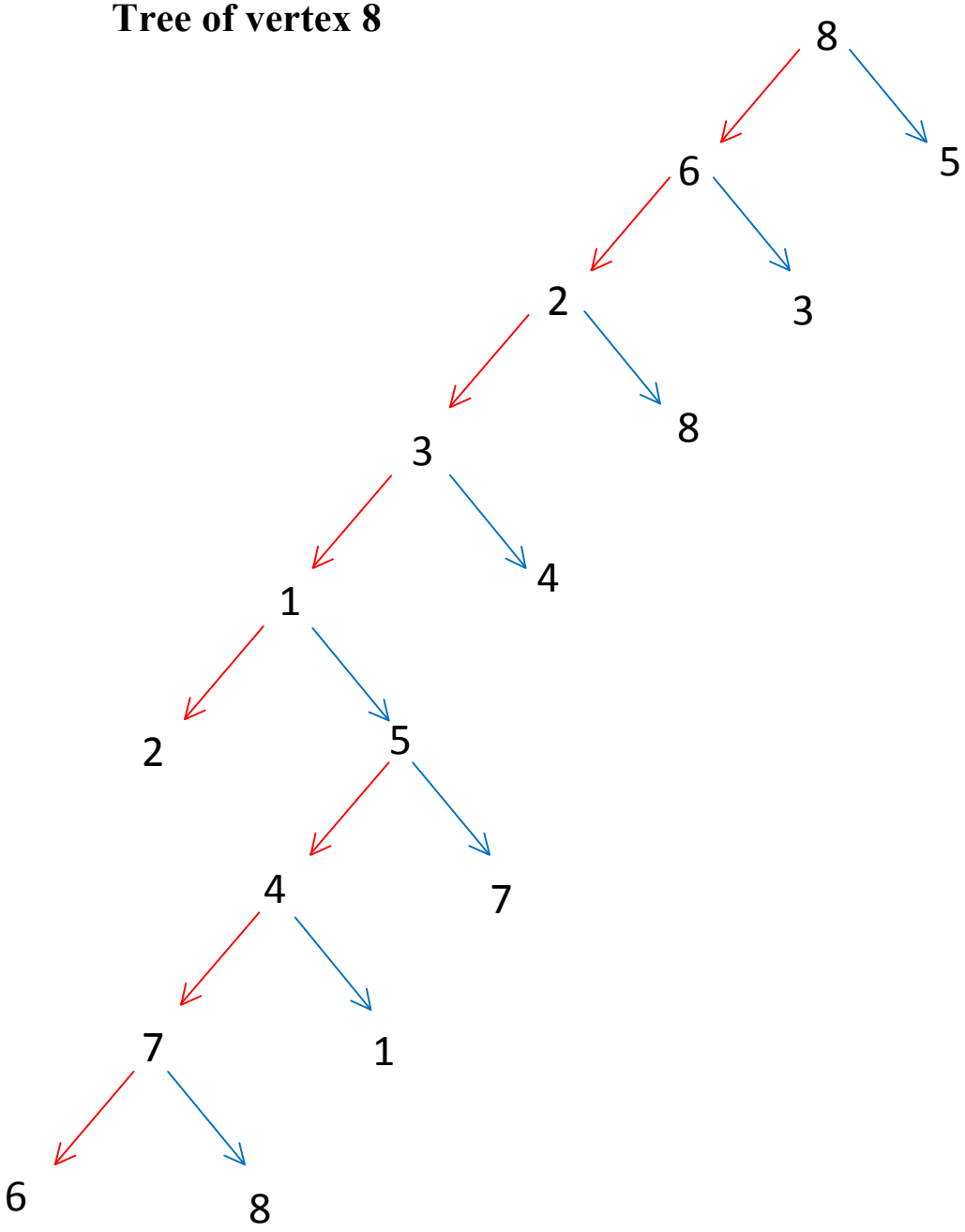
Tree of vertex 6



Tree of vertex 7



Tree of vertex 8



Appendix 2

For $i = 2$

Cycle	Length of cycle
2 3 1 2	3
2 3 1 5 4 7 6 2	7
2 3 1 5 4 7 8 6 2	8
2 3 1 5 7 6 2	6
2 3 1 5 7 8 6 2	7
2 3 4 7 6 2	5
2 3 4 7 8 6 2	6
2 3 4 1 2	4
2 3 4 1 5 7 6 2	7
2 3 4 1 5 7 8 6 2	8
2 8 6 2	3
2 8 6 3 1 2	5
2 8 6 3 4 1 2	6
2 8 5 4 7 6 2	6
2 8 5 4 7 6 3 1 2	8
2 8 5 4 1 2	5
2 8 5 7 6 2	5
2 8 5 7 6 3 1 2	7
2 8 5 7 6 3 4 1 2	8

For $i = 3$

Cycle	Length of cycle
3 1 2 3	3
3 1 2 8 6 3	5
3 1 2 8 5 4 7 6 3	8
3 1 2 8 5 7 6 3	7
3 1 5 4 7 6 2 3	7
3 1 5 4 7 6 3	8
3 1 5 4 7 8 6 2 3	8
3 1 5 4 7 8 6 3	7
3 1 5 7 6 2 3	6
3 1 5 7 6 3	5
3 1 5 7 8 6 2 3	7
3 1 5 7 8 6 3	6
3 4 7 6 2 3	5
3 4 7 6 3	4
3 4 7 8 6 2 3	6
3 4 7 8 6 3	5
3 4 1 2 3	4
3 4 1 2 8 6 3	6
3 4 1 2 8 5 7 6 3	8
3 4 1 5 7 6 2 3	7
3 4 1 5 7 6 3	6
3 4 1 5 7 8 6 2 3	8
3 4 1 5 7 8 6 3	7

For $i = 4$

Cycle	Length of cycle
4 7 6 2 3 1 5 4	7
4 7 6 2 3 4	5
4 7 6 2 8 5 4	6
4 7 6 3 1 2 8 5 4	8
4 7 6 3 1 5 4	6
4 7 6 3 4	4
4 7 8 6 2 3 1 5 4	8
4 7 8 6 2 3 4	6
4 7 8 6 3 1 5 4	7
4 7 8 6 3 4	5
4 7 8 5 4	4
4 1 2 3 4	4
4 1 2 8 6 3 4	6
4 1 2 8 5 4	5
4 1 2 8 5 7 6 3 4	8
4 1 5 4	3
4 1 5 7 6 2 3 4	7
4 1 5 7 6 3 4	6
4 1 5 7 8 6 2 3 4	8
4 1 5 7 8 6 3 4	7

For $i = 5$

Cycle	Length of cycle
5 4 7 6 2 3 1 5	7
5 4 7 6 2 8 5	6
5 4 7 6 3 1 2 8 5	8
5 4 7 6 3 1 5	6
5 4 7 8 6 2 3 1 5	8
5 4 7 8 6 3 1 5	7
5 4 7 8 5	4
5 4 1 2 8 5	5
5 4 1 5	3
5 7 6 2 3 1 5	6
5 7 6 2 3 4 1 5	7
5 7 6 2 8 5	5
5 7 6 3 1 2 8 5	7
5 7 6 3 1 5	5
5 7 6 3 4 1 2 8 5	8
5 7 6 3 4 1 5	6
5 7 8 6 2 3 1 5	7
5 7 8 6 2 3 4 1 5	8
5 7 8 6 3 1 5	6
5 7 8 6 3 4 1 5	7
5 7 8 5	3

For $i = 6$

Cycle	Length of cycle
6 2 3 1 5 4 7 6	7
6 2 3 1 5 4 7 8 6	8
6 2 3 1 5 7 6	6
6 2 3 1 5 7 8 6	7
6 2 3 4 7 6	5
6 2 3 4 7 8 6	6
6 2 3 4 1 5 7 6	7
6 2 3 4 1 5 7 8 6	8
6 2 8 6	3
6 2 8 5 4 7 6	6
6 2 8 5 7 6	5
6 3 1 2 8 6	5
6 3 1 2 8 5 4 7 6	8
6 3 1 2 8 5 7 6	7
6 3 1 5 4 7 6	6
6 3 1 5 4 7 8 6	7
6 3 1 5 7 6	5
6 3 1 5 7 8 6	6
6 3 4 7 6	4
6 3 4 7 8 6	5
6 3 4 1 2 8 6	6
6 3 4 1 2 8 5 7 6	8
6 3 4 1 5 7 6	6
6 3 4 1 5 7 8 6	7

For $i = 7$

Cycle	Length of cycle
7 6 2 3 1 5 4 7	7
7 6 2 3 1 5 7	6
7 6 2 3 4 7	5
7 6 2 3 4 1 5 7	7
7 6 2 8 5 4 7	6
7 6 2 8 5 7	5
7 6 3 1 2 8 5 4 7	8
7 6 3 1 2 8 5 7	7
7 6 3 1 5 4 7	6
7 6 3 1 5 7	5
7 6 3 4 7	4
7 6 3 4 1 2 8 5 7	8
7 6 3 4 1 5 7	6
7 8 6 2 3 1 5 4 7	8
7 8 6 2 3 1 5 7	7
7 8 6 2 3 4 7	6
7 8 6 2 3 4 1 5 7	8
7 8 6 3 1 5 4 7	7
7 8 6 3 1 5 7	6
7 8 6 3 4 7	5
7 8 6 3 4 1 5 7	7
7 8 5 4 7	4
7 8 5 7	3

For $i = 8$

Cycle	Length of cycle
8 6 2 3 1 5 4 7 8	8
8 6 2 3 1 5 7 8	7
8 6 2 3 4 7 8	6
8 6 2 3 4 1 5 7 8	8
8 6 2 8	3
8 6 3 1 2 8	5
8 6 3 1 5 4 7 8	7
8 6 3 1 5 7 8	6
8 6 3 4 7 8	5
8 6 3 4 1 2 8	6
8 6 3 4 1 5 7 8	7
8 5 4 7 6 2 8	6
8 5 4 7 6 3 1 2 8	8
8 5 4 7 8	4
8 5 4 1 2 8	5
8 5 7 6 2 8	5
8 5 7 6 3 1 2 8	7
8 5 7 6 3 4 1 2 8	8
8 5 7 8	3

Appendix 3

For $i = 1$ to $j = 3$

1 2 3
1 2 8 6 3
1 2 8 5 4 7 6 3
1 2 8 5 7 6 3
1 5 4 7 6 2 3
1 5 4 7 6 3
1 5 4 7 8 6 2 3
1 5 4 7 8 6 3
1 5 7 6 2 3
1 5 7 6 3
1 5 7 8 6 2 3
1 5 7 8 6 3

For $i = 1$ to $j = 4$

1 2 3 4
1 2 8 6 3 4
1 2 8 5 4
1 2 8 5 7 6 3 4
1 5 4
1 5 7 6 2 3 4
1 5 7 6 3 4
1 5 7 8 6 2 3 4
1 5 7 8 6 3 4

For *i* = 1 to *j* = 5

1	2	3	4	7	8	5
1	2	8	5			
1	5					

For *i* = 1 to *j* = 6

1	2	3	4	7	6	
1	2	3	4	7	8	6
1	2	8	6			
1	2	8	5	4	7	6
1	2	8	5	7	6	
1	5	4	7	6		
1	5	4	7	8	6	
1	5	7	6			
1	5	7	8	6		

For *i* = 1 to *j* = 7

1	2	3	4	7		
1	2	8	6	3	4	7
1	2	8	5	4	7	
1	2	8	5	7		
1	5	4	7			
1	5	7				

For *i* = 1 to *j* = 8

1 2 3 4 7 8
1 2 8
1 5 4 7 6 2 8
1 5 4 7 8
1 5 7 6 2 8
1 5 7 8

For *i* = 2 to *j* = 1

2 3 1
2 3 4 1
2 8 6 3 1
2 8 6 3 4 1
2 8 5 4 7 6 3 1
2 8 5 4 1
2 8 5 7 6 3 1
2 8 5 7 6 3 4 1

For *i* = 2 to *j* = 3

2 3
2 8 6 3
2 8 5 4 7 6 3
2 8 5 7 6 3

For $i = 2$ to $j = 4$

2	3	1	5	4			
2	3	4					
2	8	6	3	1	5	4	
2	8	6	3	4			
2	8	5	4				
2	8	5	7	6	3	4	

For $i = 2$ to $j = 5$

2	3	1	5				
2	3	4	7	8	5		
2	3	4	1	5			
2	8	6	3	1	5		
2	8	6	3	4	1	5	
2	8	5					

For $i = 2$ to $j = 6$

2	3	1	5	4	7	6	
2	3	1	5	4	7	8	6
2	3	1	5	7	6		
2	3	1	5	7	8	6	
2	3	4	7	6			
2	3	4	7	8	6		
2	3	4	1	5	7	6	
2	3	4	1	5	7	8	6
2	8	6					

2 8 5 4 7 6
2 8 5 7 6

For *i*= 2 to *j* = 7

2 3 1 5 4 7
2 3 1 5 7
2 3 4 7
2 3 4 1 5 7
2 8 6 3 1 5 4 7
2 8 6 3 1 5 7
2 8 6 3 4 7
2 8 6 3 4 1 5 7
2 8 5 4 7
2 8 5 7

For *i* = 2 to *j* = 8

2 3 1 5 4 7 8
2 3 1 5 7 8
2 3 4 7 8
2 3 4 1 5 7 8
2 8

For *i* = 3 to *j* = 1

3 1
3 4 1

For *i* = 3 to *j* = 2

3	1	2					
3	1	5	4	7	6	2	
3	1	5	4	7	8	6	2
3	1	5	7	6	2		
3	1	5	7	8	6	2	
3	4	7	6	2			
3	4	7	8	6	2		
3	4	1	2				
3	4	1	5	7	6	2	
3	4	1	5	7	8	6	2

For *i* = 3 to *j* = 4

3	1	2	8	5	4
3	1	5	4		
3	4				

For *i*= 3 to *j* = 5

3	1	2	8	5			
3	1	5					
3	4	7	6	2	8	5	
3	4	7	8	5			
3	4	1	2	8	5		
3	4	1	5				

For *i* = 3 to *j* = 6

3	1	2	8	6			
3	1	2	8	5	4	7	6
3	1	2	8	5	7	6	
3	1	5	4	7	6		
3	1	5	4	7	8	6	
3	1	5	7	6			
3	1	5	7	8	6		
3	4	7	6				
3	4	7	8	6			
3	4	1	2	8	6		
3	4	1	2	8	5	7	6
3	4	1	5	7	6		
3	4	1	5	7	8	6	

For *i* = 3 to *j* = 7

3	1	2	8	5	4	7
3	1	2	8	5	7	
3	1	5	4	7		
3	1	5	7			
3	4	7				
3	4	1	2	8	5	7
3	4	1	5	7		

For *i* = 3 to *j* = 8

3	1	2	8				
3	1	5	4	7	6	2	8

3	1	5	4	7	8		
3	1	5	7	6	2	8	
3	1	5	7	8			
3	4	7	6	2	8		
3	4	7	8				
3	4	1	2	8			
3	4	1	5	7	6	2	8
3	4	1	5	7	8		

For *i* = 4 to *j* = 1

4	7	6	2	3	1	
4	7	6	3	1		
4	7	8	6	2	3	1
4	7	8	6	3	1	
4	1					

For *i* = 4 to *j* = 2

4	7	6	2			
4	7	6	3	1	2	
4	7	8	6	2		
4	7	8	6	3	1	2
4	1	2				
4	1	5	7	6	2	
4	1	5	7	8	6	2

For *i* = 4 to *j* = 3

4 7 6 2 3
4 7 6 3
4 7 8 6 2 3
4 7 8 6 3
4 1 2 3
4 1 2 8 6 3
4 1 2 8 5 7 6 3
4 1 5 7 6 2 3
4 1 5 7 6 3
4 1 5 7 8 6 2 3
4 1 5 7 8 6 3

For *i* = 4 to *j* = 5

4 7 6 2 3 1 5
4 7 6 2 8 5
4 7 6 3 1 2 8 5
4 7 6 3 1 5
4 7 8 6 2 3 1 5
4 7 8 6 3 1 5
4 7 8 5
4 1 2 8 5
4 1 5

For *i* = 4 to *j* = 6

4 7 6
4 7 8 6

4	1	2	8	6			
4	1	2	8	5	7	6	
4	1	5	7	6			
4	1	5	7	8	6		

For *i* = 4 to *j* = 7

4	7						
4	1	2	8	5	7		
4	1	5	7				

For *i* = 4 to *j* = 8

4	7	6	2	8			
4	7	6	3	1	2	8	
4	7	8					
4	1	2	8				
4	1	5	7	6	2	8	
4	1	5	7	8			

For *i* = 5 to *j* = 1

5	4	7	6	2	3	1	
5	4	7	6	3	1		
5	4	7	8	6	2	3	1
5	4	7	8	6	3	1	
5	4	1					
5	7	6	2	3	1		
5	7	6	2	3	4	1	

5	7	6	3	1			
5	7	6	3	4	1		
5	7	8	6	2	3	1	
5	7	8	6	2	3	4	1
5	7	8	6	3	1		
5	7	8	6	3	4	1	

For *i* = 5 to *j* = 2

5	4	7	6	2			
5	4	7	6	3	1	2	
5	4	7	8	6	2		
5	4	7	8	6	3	1	2
5	4	1	2				
5	7	6	2				
5	7	6	3	1	2		
5	7	6	3	4	1	2	
5	7	8	6	2			
5	7	8	6	3	1	2	
5	7	8	6	3	4	1	2

For *i* = 5 to *j* = 3

5	4	7	6	2	3	
5	4	7	6	3		
5	4	7	8	6	2	3
5	4	7	8	6	3	
5	4	1	2	3		

5 4 1 2 8 6 3
5 7 6 2 3
5 7 6 3
5 7 8 6 2 3
5 7 8 6 3

For *i* = 5 to *j* = 4

5 4
5 7 6 2 3 4
5 7 6 3 4
5 7 8 6 2 3 4
5 7 8 6 3 4

For *i* = 5 to *j* = 6

5 4 7 6
5 4 7 8 6
5 4 1 2 8 6
5 7 6
5 7 8 6

For *i* = 5 to *j* = 7

5 4 7
5 7

For *i* = 5 to *j* = 8

5 4 7 6 2 8
5 4 7 6 3 1 2 8

5 4 7 8
5 4 1 2 8
5 7 6 2 8
5 7 6 3 1 2 8
5 7 6 3 4 1 2 8
5 7 8

For *i* = 6 to *j* = 1

6 2 3 1
6 2 3 4 1
6 2 8 5 4 1
6 3 1
6 3 4 1

For *i* = 6 to *j* = 2

6 2
6 3 1 2
6 3 4 1 2

For *i* = 6 to *j* = 3

6 2 3
6 3

For *i* = 6 to *j* = 4

6 2 3 1 5 4
6 2 3 4

6 2 8 5 4
6 3 1 2 8 5 4
6 3 1 5 4
6 3 4

For *i* = 6 to *j* = 5

6 2 3 1 5
6 2 3 4 7 8 5
6 2 3 4 1 5
6 2 8 5
6 3 1 2 8 5
6 3 1 5
6 3 4 7 8 5
6 3 4 1 2 8 5
6 3 4 1 5

For *i* = 6 to *j* = 7

6 2 3 1 5 4 7
6 2 3 1 5 7
6 2 3 4 7
6 2 3 4 1 5 7
6 2 8 5 4 7
6 2 8 5 7
6 3 1 2 8 5 4 7
6 3 1 2 8 5 7
6 3 1 5 4 7

6	3	1	5	7			
6	3	4	7				
6	3	4	1	2	8	5	7
6	3	4	1	5	7		

For *i* = 6 to *j* = 8

6	2	3	1	5	4	7	8
6	2	3	1	5	7	8	
6	2	3	4	7	8		
6	2	3	4	1	5	7	8
6	2	8					
6	3	1	2	8			
6	3	1	5	4	7	8	
6	3	1	5	7	8		
6	3	4	7	8			
6	3	4	1	2	8		
6	3	4	1	5	7	8	

For *i* = 7 to *j* = 1

7	6	2	3	1		
7	6	2	3	4	1	
7	6	2	8	5	4	1
7	6	3	1			
7	6	3	4	1		
7	8	6	2	3	1	
7	8	6	2	3	4	1

7 8 6 3 1
7 8 6 3 4 1
7 8 5 4 1

For *i* = 7 to *j* = 2

7 6 2
7 6 3 1 2
7 6 3 4 1 2
7 8 6 2
7 8 6 3 1 2
7 8 6 3 4 1 2
7 8 5 4 1 2

For *i* = 7 to *j* = 3

7 6 2 3
7 6 3
7 8 6 2 3
7 8 6 3
7 8 5 4 1 2 3

For *i* = 7 to *j* = 4

7 6 2 3 1 5 4
7 6 2 3 4
7 6 2 8 5 4
7 6 3 1 2 8 5 4
7 6 3 1 5 4

7 6 3 4
7 8 6 2 3 1 5 4
7 8 6 2 3 4
7 8 6 3 1 5 4
7 8 6 3 4
7 8 5 4

For *i* = 7 to *j* = 5

7 6 2 3 1 5
7 6 2 3 4 1 5
7 6 2 8 5
7 6 3 1 2 8 5
7 6 3 1 5
7 6 3 4 1 2 8 5
7 6 3 4 1 5
7 8 6 2 3 1 5
7 8 6 2 3 4 1 5
7 8 6 3 1 5
7 8 6 3 4 1 5
7 8 5

For *i* = 7 to *j* = 6

7 6

7 8 6

For *i* = 7 to *j* = 8

7 6 2 8
7 6 3 1 2 8
7 6 3 4 1 2 8
7 8

For *i* = 8 to *j* = 1

8 6 2 3 1
8 6 2 3 4 1
8 6 3 1
8 6 3 4 1
8 5 4 7 6 2 3 1
8 5 4 7 6 3 1
8 5 4 1
8 5 7 6 2 3 1
8 5 7 6 2 3 4 1
8 5 7 6 3 1
8 5 7 6 3 4 1

For *i* = 8 to *j* = 2

8 6 2
8 6 3 1 2
8 6 3 4 1 2
8 5 4 7 6 2

8 5 4 7 6 3 1 2
8 5 4 1 2
8 5 7 6 2
8 5 7 6 3 1 2
8 5 7 6 3 4 1 2

For *i* = 8 to *j* = 3

8 6 2 3
8 6 3
8 5 4 7 6 2 3
8 5 4 7 6 3
8 5 4 1 2 3
8 5 7 6 2 3
8 5 7 6 3

For *i* = 8 to *j* = 4

8 6 2 3 1 5 4
8 6 2 3 4
8 6 3 1 5 4
8 6 3 4
8 5 4
8 5 7 6 2 3 4
8 5 7 6 3 4

For *i* = 8 to *j* = 5

8 6 2 3 1 5

8 6 2 3 4 1 5
8 6 3 1 5
8 6 3 4 1 5
8 5

For *i* = 8 to *j* = 6

8 6
8 5 4 7 6
8 5 7 6

For *i* = 8 to *j* = 7

8 6 2 3 1 5 4 7
8 6 2 3 1 5 7
8 6 2 3 4 7
8 6 2 3 4 1 5 7
8 6 3 1 5 4 7
8 6 3 1 5 7
8 6 3 4 7
8 6 3 4 1 5 7
8 5 4 7
8 6 5 7

Appendix 4

Mat lab code for formation of trees

```
%% Tree formation for all nodes as root nodes
for i=1:8
    t(i) = tree(num2str(i));

    flag = zeros(1,n);

    currNode = 1;

    while(sum(flag)~=n*d)
        if(flag(str2double(t(i).get(currNode)))<d)
            for j=1:12
                temp = j;
                if(adj(str2double(t(i).get(currNode)),
j)==flag(str2double(t(i).get(currNode)))+1)
                    flag(str2double(t(i).get(currNode))) =
flag(str2double(t(i).get(currNode))) + 1;
                    if(flag(temp)==0)
                        t(i) = t(i).addnode(currNode, num2str(temp));
                        temp2 = t(i).getchildren(currNode);
                        currNode = temp2(length(temp2));
                    else
                        t(i) = t(i).addnode(currNode, num2str(temp));
                    end
                    break;
                end
            end
        else
            currNode = t(i).getparent(currNode);
        end
    end

    disp('The tree for the given graph is-')
    disp(t(i).tostring())
end
```

Mat lab code for obtaining cycles and paths

```
%% Determining all cycles
count = zeros(1,8);
for i=1:8
    fprintf('For i = %d',i)
    disp(' ')
    stack = cell(1);

    % Determine the first straight path
    currNode = 1;
    path = str2double(t(i).get(currNode));
    index = currNode;
    while (~isRep(path))
        stack = push(stack, 1);                          % push 1 to stack
        currNode = t(i).getchildren(currNode);
        currNode = currNode(1);                          % make currNode to
the first children to the left
        path = [path str2double(t(i).get(currNode))];    % add current node to
path
        index = [index currNode];
        % if the current node is a leaf node => jump to the node with same
        % value parent node
        if(t(i).isleaf(currNode))
            temp = find(t(i)==t(i).get(currNode));
            for m=1:length(temp)
                if(~t(i).isleaf(temp(m)))
                    currNode = temp(m);
                    break;
                end
            end
        end
    end
    % if the obtained path is required solution
    if(path(1)==path(length(path)))
        disp(path)
        count(i) = count(i)+1;
    end

    % Determining remaining paths
    while 1
        while(stack{1}==d)
            stack = pop(stack);
            path = path(1:(length(path)-1));
            index = index(1:(length(index)-1));
        end
        if(isempty(stack{1}))
            break;
        end
        [stack,val] = pop(stack);
        stack = push(stack, val+1);
        % if the current node is a leaf node => jump to the node with same
        % value parent node
        if(t(i).isleaf(index(length(index)-1)))
            temp = find(t(i)==t(i).get(index(length(index)-1)));
```

```
            for m=1:length(temp)
                if(~t(i).isleaf(temp(m)))
                    temp2 = temp(m);
                    break;
                end
            end
        else
            temp2 = index(length(index)-1);
        end
        temp = t(i).getchildren(temp2);
        path(length(path)) = str2double(t(i).get(temp(val+1)));
        index(length(index)) = temp(val+1);
        currNode = temp(val+1);
        while (~isRep(path))
            % if the current node is a leaf node => jump to the node with
same
            % value parent node
            if(t(i).isleaf(currNode))
                temp = find(t(i)==t(i).get(currNode));
                for m=1:length(temp)
                    if(~t(i).isleaf(temp(m)))
                        currNode = temp(m);
                        break;
                    end
                end
            end
            stack = push(stack, 1);                          % push 1 to stack
            currNode = t(i).getchildren(currNode);
            currNode = currNode(1);                          % make currNode
to the first children to the left
            path = [path str2double(t(i).get(currNode))];    % add current
node to path
            index = [index currNode];
        end
        % if the obtained path is required solution
        if(path(1)==path(length(path)))
            disp(path)
            count(i) = count(i)+1;
        end
    end
end

%% Determine no loop paths from node 1 -> 2,3,...,n && 2 -> 1,3,4,...,n &&
... n -> 1,2,3,...,n-1
for i=1:8
    for j=1:8
        if i~=j
            fprintf('For i = %d to j = %d',i,j)
            disp(' ')
            stack = cell(1);

            % Determine the first straight path
            currNode = 1;
            path = str2double(t(i).get(currNode));
            index = currNode;
            while (~isRep(path) && path(length(path))~=j)
```

```
                stack = push(stack, 1);                             % push 1 to
stack
                currNode = t(i).getchildren(currNode);
                currNode = currNode(1);                             % make
currNode to the first children to the left
                path = [path str2double(t(i).get(currNode))];   % add current
node to path
                index = [index currNode];
                % if the current node is a leaf node => jump to the node with
same
                % value parent node
                if(t(i).isleaf(currNode))
                    temp = find(t(i)==t(i).get(currNode));
                    for m=1:length(temp)
                        if(~t(i).isleaf(temp(m)))
                            currNode = temp(m);
                            break;
                        end
                    end
                end
            end
            % if the obtained path is required solution
            if(path(length(path))==j)
                disp(path)
            end

            % Determining remaining paths
            while 1
                while(stack{1}==d)
                    stack = pop(stack);
                    path = path(1:(length(path)-1));
                    index = index(1:(length(index)-1));
                end
                if(isempty(stack{1}))
                    break;
                end
                [stack,val] = pop(stack);
                stack = push(stack, val+1);
                % if the current node is a leaf node => jump to the node with
same
                % value parent node
                if(t(i).isleaf(index(length(index)-1)))
                    temp = find(t(i)==t(i).get(index(length(index)-1)));
                    for m=1:length(temp)
                        if(~t(i).isleaf(temp(m)))
                            temp2 = temp(m);
                            break;
                        end
                    end
                else
                    temp2 = index(length(index)-1);
                end
                temp = t(i).getchildren(temp2);
                path(length(path)) = str2double(t(i).get(temp(val+1)));
                index(length(index)) = temp(val+1);
                currNode = temp(val+1);
                while (~isRep(path) && path(length(path))~=j)
```

```
                    % if the current node is a leaf node => jump to the node
with same
                    % value parent node
                    if(t(i).isleaf(currNode))
                        temp = find(t(i)==t(i).get(currNode));
                        for m=1:length(temp)
                            if(~t(i).isleaf(temp(m)))
                                currNode = temp(m);
                                break;
                            end
                        end
                    end
                    stack = push(stack, 1);                          % push 1
to stack
                    currNode = t(i).getchildren(currNode);
                    currNode = currNode(1);                          % make
currNode to the first children to the left
                    path = [path str2double(t(i).get(currNode))];    % add
current node to path
                    index = [index currNode];
                end
                % if the obtained path is required solution
                if(path(length(path))==j)
                    disp(path)
                end
            end
        end
    end
end
```