Extraction of all circuits starting and ending at specified vertices.

Pradeep Gangwar (IHM2016501)[1] Prakhar Chaturvedi (IHM2016001)[2] Praphull Mishra (IWM2016005)[3]

***Abstract*— The paper details the algorithm to detect all circuits present in an un-directed graph. Given a vertex and the adjacency list representation of the graph, the proposed algorithm will output all the circuits beginning from the given vertex.**

I. INTRODUCTION AND LITERATURE SURVEY

Graphs are mathematical structures that represent pair-wise relationship between objects. They perform important role in modeling data. They are useful in solving many real life applications such as social networks, transportation networks, document-link graphs and many more. Let us introduce the important notations and definitions that we will use frequently in this paper.

A. Basic Definitions

1) *Graph:* Graph is a pair G=(V,E) where,
   - V is a set of vertices (or nodes), and
   - E $\subseteq$ (VxV) is a set of edges.

   A graph may be directed or undirected, if graph is undirected, then adjacency relation defined by edges is symmetric.
2) *Closed walk:* A closed walk consists of a sequence of vertices starting and ending at the same vertex, with each two consecutive vertices in the sequence adjacent to each other in the graph.
3) *Circuit:* A circuit is a closed walk allowing repetitions of vertices but not edges. However, it can also be a simple cycle i.e. a closed walk with no repetitions of vertices and edges allowed.

B. Objective

Here we will propose algorithms to detect circuits present the the graph given the start and ending vertex. Two variations of the algorithm are presented below.

II. MOTIVATION

Graph are very important data structures. They are used in network related algorithms, routing, finding relation among objects, shortest paths and many more real-life applications. On E-Commerce websites graphs are used to show recommendations.Connecting with friends on social media, where each user is a vertex and the connection between two users is represented by an edge, is also an important application of graphs.

The adjacency matrix and incidence matrix representation of a graph have a very high space complexity. So for above mentioned real life problems, where the size of graph is large and the cost of allocating the space is more, these representations become costly. This motivates us to represent our graphs in some condensed and compressed form, so that space complexity should not be a problem and we can focus on other important aspects.

III. ALGORITHM DESIGN

This part of the report can be further divided into following sections.

A. Input Format

The user inputs the total number of vertices and the adjacency list representation of the undirected graph. The start vertex to detect the circuits is also specified. The first version of the algorithms prints the repeated circuits while the improved version prints only the unique circuits.

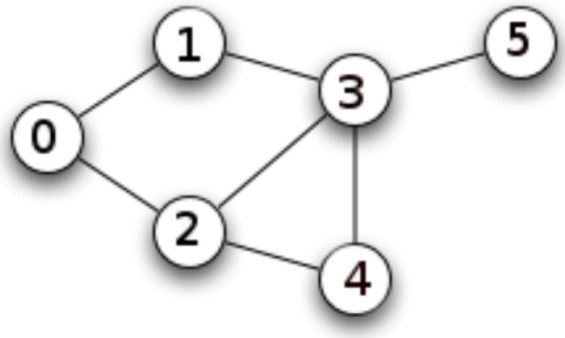

Fig. 1. An undirected graph containing 6 vertices and two circuits that begin from vertex 0.

B. Algorithm 1

The first version of the algorithm prints the repeated circuits as in from *Fig. 1* we observe the list of circuits beginning from vertex 0 are:

1) 0 1 3 2 0
2) 0 1 3 4 2 0
3) 0 2 4 3 1 0
4) 0 2 3 1 0

- We observe from the list that items 1 & 3 and items 2 & 4 are the same circuits. In next section, the modified algorithm proposed, solves this drawback.
- The depth first traversal method is applied to solve the problem of extracting circuits. Begin the traversal

from the start vertex provided as input by the user. The pseudo code of the algorithm *Extract Circuits* is given below Fig. 1.

- The time complexity of the above algorithm is $O(V * (V + E))$. The space complexity is $|(V + E)|$ for the adjacency list and $|V|$ each for the path and visited arrays hence giving a space complexity $O(V + E)$.

C. Algorithm 2

- The modified version of the above algorithm, **Extract Unique Circuits** prints only the unique circuits present in the graph.
- A boolean data structure extracted of size $|V|$ is used to keep track of the vertices whose circuits have already been extracted.
- When the same vertex is obtained while finding the circuit from another adjacent vertex, the recursive call breaks printing only unique circuits.
- The time complexity of this method is also $O(V * (V + E))$. The space complexity is $|(V + E)|$ for the adjacency list and $|V|$ each for the extracted, path and visited arrays hence giving a space complexity $O(V + E)$.

D. Algorithm

Algorithm 1 Algorithm 1 ((Extract All Circuit)

```
1:  procedure MAIN(s, d)                                   ▷ Main function
2:      s ← start vertex
3:      for each i ∈ G.Adj[ s] do
4:          extract_circuits(i, s)
5:
6:  procedure EXTRACT_CIRCUITS(s, d)  ▷ Driver function
7:      visited[ ] ← false
8:      path[ ]
9:      path_index ← 0
10:     extract_util(s, d, visited, path, path_index)
11:
12: procedure EXTRACT_UTIL(s, d, visited, path, path_index)
    ▷ Utility function
13:     visited[ s] ← true
14:     path[ path_index] ← s
15:     path_index + +
16:     if s = d and path_index > 2  then
17:         Print the output circuit
18:     else
19:         for each i ∈ G.Adj[ s] do
20:             if !(visited[ i] ) then
21:                 extract_util(s, d, visited, path, path_index)
22:     visited[ s] ← false
23:     path_index − −
```

Algorithm 2 Algorithm 2 (Extract Unique Circuit)

```
1:  procedure MAIN(s, d)                                   ▷ Main function
2:      s ← start vertex
3:      extracted[ ] ← false
4:      for each i ∈ G.Adj[ s] do
5:          extract_circuits(i, s)
6:          extracted[ i] ← true
7:  procedure EXTRACT_CIRCUITS(s, d)  ▷ Driver function
8:      visited[ ] ← false
9:      path[ ]
10:     path_index ← 0
11:     extract_util(s, d, visited, path, path_index)
12: procedure EXTRACT_UTIL(s, d, visited, path, path_index)
    ▷ Utility function
13:     visited[ s] ← true
14:     path[ path_index] ← s
15:     path_index + +
16:     if s = d and path_index > 2  then
17:         Print the output circuit
18:     else
19:         for each i ∈ G.Adj[ s] do
20:             if !(visited[ i] ) and !(extracted[ i] ) then
21:                 extract_util(s, d, visited, path, path_index)
22:     visited[ s] ← false
23:     path_index − −
```

E. Output Format

The output consists of all the circuits that are possible starting and ending at that vertex.

IV. ANALYSIS

A. Run Time Analysis

Figure 2 is a plot depicting the run time analysis when the number of vertices were varied, while keeping the number of edges fixed to values 45 and 50 in two cases as shown in the plot. The time in seconds, is the time taken to extract all the unique circuits present in the randomly generated graphs. The number of circuits extracted is printed above the line and varies depending on the nature of the graph.

Figure 3 is a plot depicting the run time analysis when the number of edges were varied, while keeping the number of vertices fixed to values 10 and 12 in two cases as shown in the plot. The time in seconds, is the time taken to extract all the unique circuits present in the randomly generated graphs. The number of circuits extracted is printed above the line and varies depending on the nature of the graph.

V. DISCUSSION

A. Understanding the graph using adjacency list

Adjacency list contains the array of size equal to number of vertices. Each item in array consist of list vertices which

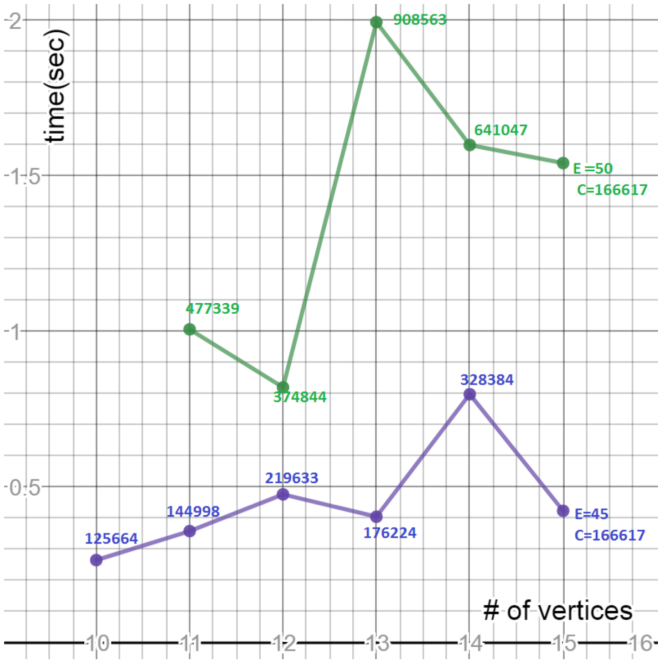

Fig. 2. Plot of time vs. number of vertices(with number of edges fixed).

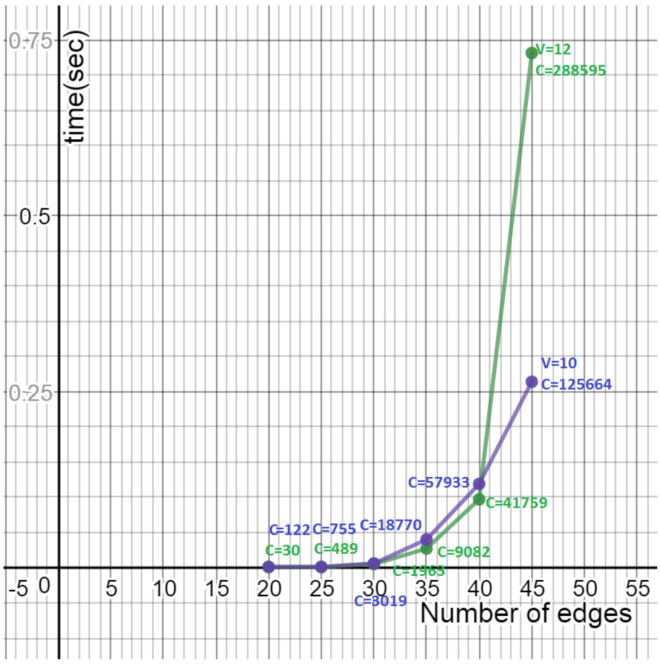

Fig. 3. Plot of time vs. number of edges(with number of vertices fixed).

are adjacent to ith vertex.

B. About extraction of circuits

We now know that circuit is a closed walk in which vertices an be repeated but not edges. In this paper we were given a vertex and we were asked to find a circuit which starts and ends at same vertex. This can be also attributed as cycle. We used DFS approach to search for all the vertex and trace the path. We stop when we reach to the same vertex again.

VI. CONCLUSION

The above paper details the implementation of the algorithms which solves the problem of extraction of all circuits in a graph given the starting vertex. The solution has been optimized to print only unique paths, using optimum space and time possible.

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] Introduction to Algorithms English By Thomas H. Cormen , By Charles E. Leiserson , Ronald L. Rivest ,Clifford Stein(3rd edition)