Generate Set of Trees from a Graph where Centre remains same

Divija Palleti
IHM2016005[1]

Nidhish Kumar Reddy
ICM2016006[2]

Nanda Kishore
IWM2016002[3]

***Abstract*— The objective of this report is to generate set of trees from graph where centre of the given graph and the newly formed tree remains the same.**

***Index Terms*— Distance Matrix, Incidence Matrix, Eccentricity, Radius, Breadth first Search.**

I. INTRODUCTION

Graph is a collection of finite sets of vertices or nodes **(V)** and edges **(E)**. In order to describe a graph into any representation, relationship between edges and vertices should be preserved. There are variety of ways to represent a graph. One of them is Incidence Matrix.

A spanning tree **T** of an un-directed graph **G** is a sub graph that is a tree which includes all of the vertices of G, with minimum possible number of edges. In general, a graph may have several spanning trees, but a graph that is not connected will not contain a spanning tree

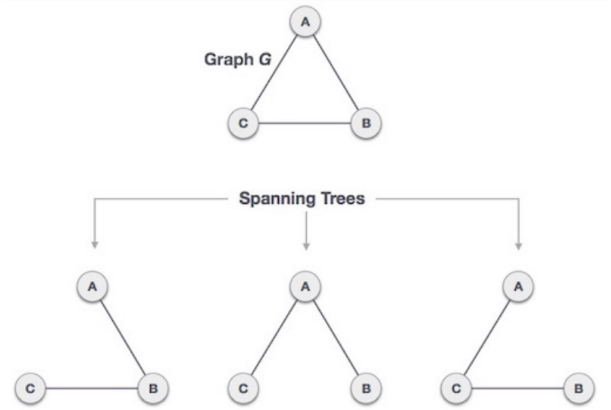

Fig. 1. Spanning Trees

The center of a graph is the set of all vertices of minimum eccentricity,that is, the set of all vertices u where the greatest distance d(u,v) to other vertices v is minimal. Equivalently, it is the set of vertices with eccentricity equal to the graph's radius.

II. MOTIVATION

Graphs are one of the ancient and powerful tools that visually represent sets of information and therefore the relations among them. They have a really long history of serving to individuals communicate, perceive the important world, and solve scientific issues. Graph visualization and connectivity helps users gain insight into data by turning the information components and their internal relationships into graphs. Graph connectivity leverages the human visual system to support data discovery. it's been wide employed in several applications, like social networks and net communications. It is shown how to generate the path matrix or corresponding lists which arise in various applications involving graphs and their spanning trees.

III. ALGORITHM DESIGN

A. Background

Given a graph G=(V,E), where V is set of nodes and E is set of edges

Let's denote d[i,j] as shortest distance between nodes i,j:

- Eccentricity of node as maximal distance from that node to other:**E(v)=max d[v,u]** where u belongs to set of vertices.
- Having values of eccentricity of all nodes, we can define radius of graph as minimal one among them:**R(G)=min (E[v])** where v belongs to set of vertices.
- Center of graph is set of nodes with eccentricity equal to the radius of graph:**C(G)::E[v]=R(G)**

B. Methodology

We have divided this particular problem in 3 stages.

1) Stage-1: In this stage we computed the distance matrix of the given graph.In distance matrix, **distance[i][j]** is the shortest distance between vertex i and vertex j. Distance from vertex i to vertex i is 0. (i.e)The main diagonal values in distance matrix is 0. So to calculate the shortest distance between vertex i and j,we used modified Breadth first search algorithm.

- The idea is to perform BFS from one of given input vertex(u). At the time of BFS maintain an array of distance[n] and initialize it to zero for all vertices.
- Now, suppose during BFS, vertex x is popped from queue and we are pushing all adjacent non-visited vertices(i) back into queue.
- At the same time we should update distance[i] = distance[x] + 1.
- Finally distance[v] gives the shortest distance between u and v.

2) Stage-2: In this stage we generate all spanning trees of the given graph. A Spanning tree should contain (V-1)edges. Since a Spanning tree is a sub graph, we generated all possible combinations of (V-1) edges from E edges. After Generating all possible sub graphs of (V-1) edges. In this manner we retains the sub graphs which satisfy Tree

conditions(which are spanning trees of the given graph). The conditions are

- Contains all vertices of Given Graph.
- Doesn't contain any cycle.
- contains (V-1) edges.

3) Stage-3: In this stage we calculate the centres of given graph and obtained spanning tress. A graph is defined as set of points known as Vertices and line joining these points is known as Edges. It is a set consisting of where V is vertices and E is edge.

- Eccentricity is defined as the maximum distance of one vertex from other vertex. The maximum distance between a vertex to all other vertices is considered as the eccentricity of the vertex. It is denoted by e(V).
- A radius of the graph exists only if it has the diameter. The minimum among all the maximum distances between a vertex to all other vertices is considered as the radius of the Graph G. It is denoted as r(G).
- Centre consists of all the vertices whose eccentricity is minimum. Here the eccentricity is equal to the radius. For example, if the school is at the center of town it will reduce the distance buses has to travel.

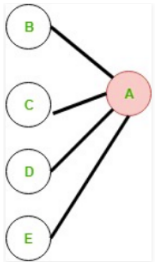

Fig. 2. A is the centre

C. Algorithm-1

Finding Shortest distance from given vertex to all vertices

Algorithm 1 Distance_Matrix

procedure MINEDGEBFS($edge[], u, v, n$)
  $queueQ$
  $distance[u] \leftarrow 0$
  $Q.push(u)$
  $visited[u] \leftarrow true$
  for !q.empty() **do**
    $x \leftarrow Q.front()$
    $Q.pop()$
    for i=0 to edges[x].size() **do**
      if visited[edge[x][i]] **then**
        $continue$
      $distance[edge[x][i]] = distance[x] + 1$
      $Q.push(edge[x][i])$
      $visited[edge[x][i]] = 1$
  $returndistance[v]$;

D. Algorithm-2

Finds all possible spanning trees of given graph

Algorithm 2 Spanning_Trees

procedure SUBGRAPH::ISCYCLIC()(vertex,visted [],parent)
  $visited[vertex] \leftarrow true$
  for iterate adjacency list of vertex **do**
    if !visited[*i] **then**
      if isCyclic(*i, visited, vertex) **then**
        $returntrue$
    if *i != parent **then**
      $returntrue$
  $returntrue$
procedure SUBGRAPH::ISTREE()
  for i=0 to (no_vert) **do**
    $visited[i] \leftarrow false$
  if isCyclic(0,visited,-1) **then**
    $returnfalse$
  for i=0 to (no_vert) **do**
    if !visited[i] **then**
      $returnfalse$
  $returntrue$
procedure PRINT($v, no\_vert - 1$)
  for i=0 to (no_vert-1) **do**
    $z = 0$
    for j=0 to (no_vert) **do**
      if I[j][v[i]]==1 **then**
        $ones[z] = j$
        $z++$
    g.addedge(ones[0],ones[1]);
  if g.isTree() **then**
    for i=0 to (no_vert-1) **do**
      trees[t][i] ←v[i]
   $t++$
   no_trees ←t−

procedure GO($offset, no\_vert - 1$)
  $k \leftarrow no\_vert - 1$
  if k==0 **then**
    $print(combination)$
    $return$
  for i=offset to (edges.size-k) **do**
    $combination.push_back(edges[i])$
    $go(i + 1, k - 1)$
    $combination.pop_back()$

E. Algorithm-3

Finding Centre of Graph or Tree from obtained Distance matrix

Algorithm 3 Centres

```
procedure CENTRE(distance[][])
    rad ← INT_MAX
    for i=0 to (no_vert) do
        ecc[i] ← −1
    for i=0 to (no_vert) do
        for j=0 to (no_vert) do
            e[i] ← max(e[i], d[i][j])
    for i=0 to (no_vert) do
        rad ← min(rad, e[i])
    for i=0 to (no_vert) do
        if e[i] == rad then
            centre.push_back(i)
```

IV. ANALYSIS

A. Analysis

1) Algorithm-1: In the Algorithm 1 to find the distance between 2 vertices we used modified BFS. We know that time complexity of BFS is $O(V+E)$.
. Since we need to apply this function to every vertex in graph.So the time complexity will be directly proportional to $O(V*V(V+E))$.Since E=V-1 for spanning trees.So the time complexity will be directly proportional to $O(V^3)$

$$time_{method-1} \propto O(V^3)$$

$$\implies O(V^3) \text{ - A Cubic Time Computation}$$

2) Algorithm - 2: In Algorithm-2 We generate all spanning trees. Since to check a graph whether it is a tree or not. We need to check doesn't contain any cycle. For checking cycle in a un-directed graph we used BFS method. So the time complexity for checking a graph is tree or not is directly proportional to $O(V+E)$. Since E=V-1 for spanning trees.The time complexity is directly proportional to $O(V^2)$. Since we get E**C**(V-1) combinations. This shows that the final time complexity is proportional to $O(E\mathbf{C}(V-1)*(V^2))$.In dense graph E directly proportional to V^2.

$$time_{method-1} \propto O(V^3)$$

$$\implies O(V^3) \text{ - A Cubic Time Computation}$$

3) Algorithm-3: The algorithm-3 is for generating centres of the graph or tree.Algorithm takes more time in calculating eccentricities of all vertices.So the time complexity is directly proportional to $O(V*V)$,i.e $O(V^2)$

$$time_{algorithm-3} \propto O(V^2)$$

$$\implies O(V^2) \text{ - A Quadratic Time Computation}$$

V. EXPERIMENTAL STUDY

The best way to study an algorithm is by graphs and profiling.

1) Graphs-Time-Complexity: We have seen that $t_{Algo-2} >= t_{Algo-1}$. So,the graphs for Algo-1 lies below Method1.We can clearly see that Algo-2 and Algo-1 is of order V^3 for connected graphs.

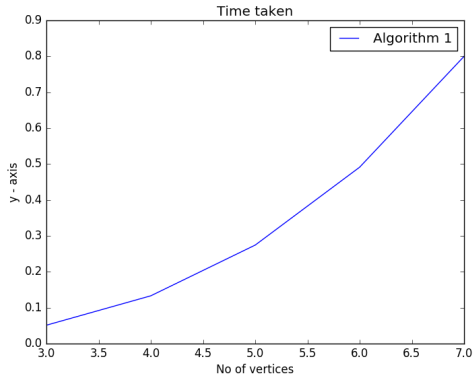

Fig. 3. Algorithm 1 Complexity

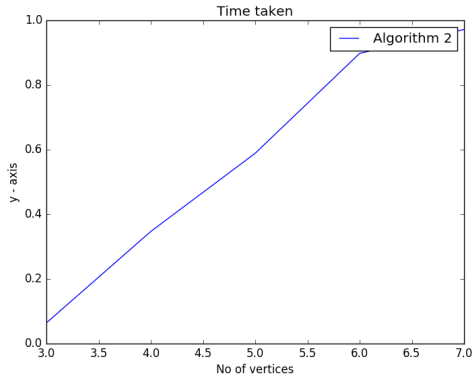

Fig. 4. Algorithm 2 complexity.

2) Profiling:

$Table1 : NumberOfComputations$

V	t_{Algo1}	t_{Algo2}	t_{Algo3}
3	0.0512	0.0064	0.036
4	0.1331	0.347	0.041
5	0.2744	0.589	0.052
6	0.4913	0.898	0.063
7	0.8000	0.972	0.076

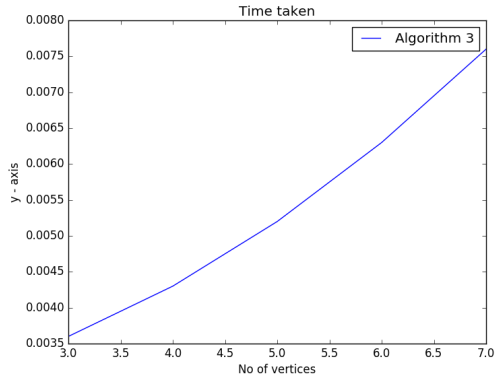

Fig. 5. Algorithm-3 complexity.

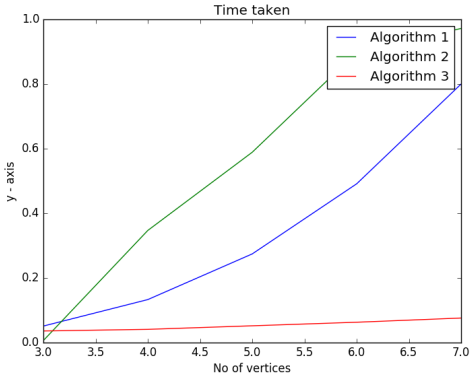

Fig. 6. Comparison of Algo 1,2 and 3 complexity.

- We can seen that $t_{Algo2} >= t_{Algo1} >= t_{Algo3}$
- We can also see that time increases as the value of n increases and $time_{Algo-1} \propto V^3$, $time_{Algo-2} \propto V^3$,$time_{Algo-3} \propto V^2$

VI. DISCUSSIONS

To solve this problem we have divided it into 3 Stages, which generates set of trees from graph, where centre remains same. We have also provided a comparative study of various algorithms which can be used in solving this problem.

VII. CONCLUSIONS

- We can see that the Complexity of Distance matrix generation in above Algorithm-1 is $O(V^3)$ Where V is the number of rows in the given adjacency matrix.
- We can see that the Complexity of Spanning trees generation in above Algorithm-2 is $O(V^3)$ Where V is the number of vertices.
- We can see that the Complexity of centres generation in above Algorithm-3 is $O(V^2)$ Where V is the number of vertices.

VIII. APPLICATIONS

Practical implementations of the presented algorithm and the information obtained through the present algorithm can be used in finding the shortest path and can also as intermediate information in generating the spanning tree. It can be used to gain knowledge about various molecules, their structure and properties.

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)
[3] $https://codeforces.com/blog/entry/17974/$
[4] $https://en.wikipedia.org/wiki/Incidence\_matrix$