Eccentricity of Incidence and Adjacency Matrix

Mudit Rathore[1], Ishani Mishra[2], and Gugloth Lavanya[3]

[1]Indian Institute of Information Technology Allahabad, ICM2015502
[2]Indian Institute of Information Technology Allahabad, IWM2015008
[3]Indian Institute of Information Technology Allahabad, ISM2015003

***Abstract*— In recent times, graph theory has played an increasingly important role in social network analysis, networking, routing, biological systems, path and route problems and many more. The eccentricity of a node in a graph is defined as the length of a longest shortest path starting at that node. Eccentricity distribution over all nodes is a relevant descriptive property of the graph, and its extreme values allow the derivation of measures such as the radius, diameter, center and periphery of the graph. The nodes with the highest eccentricity values realize the diameter of the graph and form the graph periphery, whereas the nodes with the lowest values realize the radius and form the center of the graph, and finding exactly these nodes can be useful within various application areas mentioned above.**
This paper details an algorithm to find out the eccentricity of a graph. Given the matrix representation of a graph, the proposed algorithm will output its eccentricity.

I. INTRODUCTION

A Graph [1] ***G*** is defined as an ordered pair of a finite set ***V*** of vertices and a finite set ***E*** of edges, where a vertex is a node and an edge is a connection between any two vertices. A graph can be represented using Adjacency Matrix and Incidence Matrix.

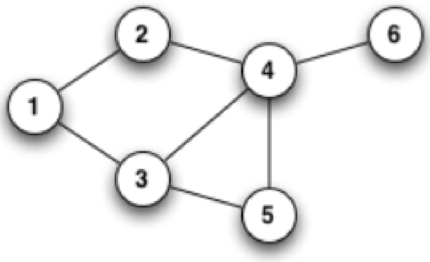
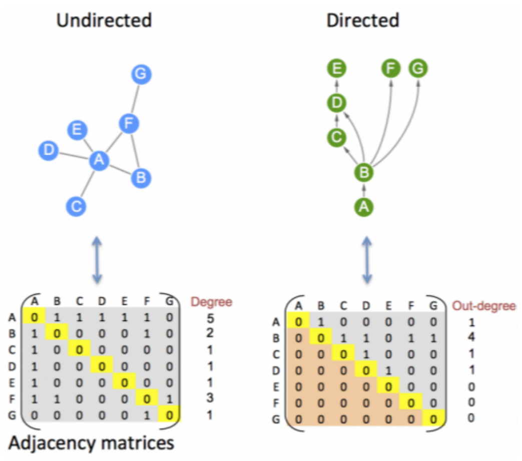


Fig. 1. A simple undirected graph

Adjacency is a relation between two vertices of a graph. A vertex v is adjacent to vertex u if there is an edge from vertex u to vertex v i.e. edge(u,v) $\in$ E.
An *adjacency matrix* is a matrix holding this relation. It is a square matrix A used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices (v_i, v_j) are adjacent or not, in the graph. If vertex vi is adjacent to vertex v_j, then a_{ij} = 1 and a_{ji} = 1, if they are not connected then a_{ij} = 0 and a_{ji} = 0. In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix of order $|V|$ with zeros on its diagonal a_{ii} = 0 which represents loop condition.

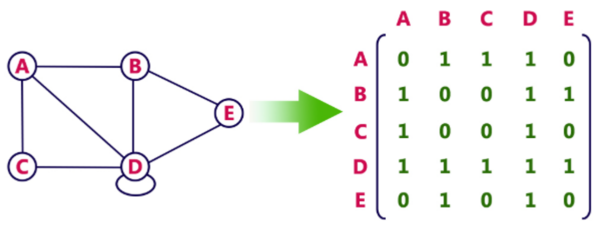


	A	B	C	D	E	F	G	Degree
A	0	1	1	1	1	1	0	5
B	1	0	0	0	0	1	0	2
C	1	0	0	0	0	0	0	1
D	1	0	0	0	0	0	0	1
E	1	0	0	0	0	0	0	1
F	1	1	0	0	0	0	1	3
G	0	0	0	0	0	1	0	1

	A	B	C	D	E	F	G	Out-degree
A	0	1	0	0	0	0	0	1
B	0	0	1	1	0	1	1	4
C	0	0	0	1	0	0	0	1
D	0	0	0	0	1	0	0	1
E	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0

Fig. 2. Adjacency Matrix

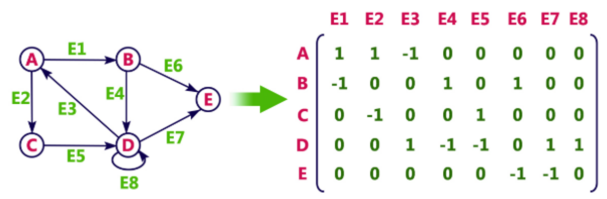
The *incidence Matrix* A of a graph G(V,E) with V as a set of vertices and E as the set of edges is a $|V| * |E|$ matrix $A = [a_{ij}]$, where the if rows correspond to the vertices and the of columns correspond to edges. For an undirected graph, if j^{th} edge is incident on the i^{th} vertex then a_{ij} = 1 if the 0 otherwise. For a Directed Graph a_{ij} = -1 if the tail of the j^{th} edge incident on the ith vertex, a_{ij} = 1 if the head of the j^{th} edge incident on the i^{th} vertex and a_{ij} = 0 otherwise.



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0

Fig. 3. Incidence Matrix for undirected graph

The distance is a metric space d: V x V $\rightarrow Z^{+U0}$. this satisfies the following properties [2]:
1. d(u,v) $>$ 0 and d(u,u) = 0
2. d(u,v) = d(v,u)



$$\begin{array}{c|cccccccc} & E1 & E2 & E3 & E4 & E5 & E6 & E7 & E8 \\ \hline A & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ B & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ C & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ D & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 1 \\ E & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \end{array}$$

Fig. 4. Incidence Matrix for directed graph

3. d(u,v) + d(v,z) = d(u,z)

The **eccentricity**[3] ∈(v) of a graph vertex v in a connected graph G(V,E) is the maximum graph distance between v and any other vertex u of G. For a disconnected graph, all vertices are defined to have infinite eccentricity [4].

∈(v) = max d(v,u) | u ∈ V(G))

The maximum eccentricity[5] is the graph diameter[6]. The minimum graph eccentricity is called the graph radius.

A graph's **diameter** is the length max$_{(u,v)}$d(u,v) of the "longest shortest path" between any two graph vertices (u,v) of a graph, where d(u,v) is a graph distance. In other words, a graph's diameter is the largest number of vertices which must be traversed in order to travel from one vertex to another when paths which backtrack, detour, or loop are excluded from consideration.

The **radius** of a graph is the minimum graph eccentricity of any graph vertex in a graph. A disconnected graph therefore has infinite radius. A Center of a graph is a vertex with eccentricity equal to the radius r. The diameter, radius and center of a graph can be found by computing the distances between all pairs of vertices. It takes $O(VE + V_2 * logV)$ time for a general graph.

Diameter = max $\in (v)|v \in V(G))$
Radius = min $\in (v)|v \in V(G))$
A general observation is:
Radius(G) $<=$ Diameter(G) $<<$ 2 * Radius(G) [7].

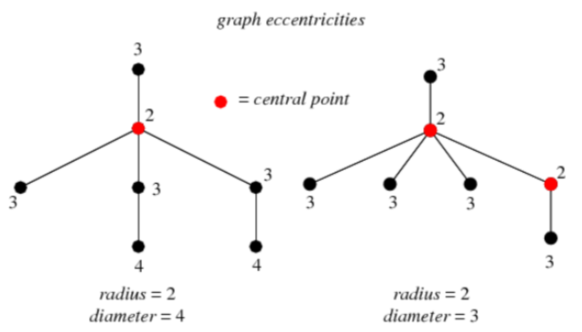


Fig. 5. Adjacency Matrix

A. Applications of eccentricity

In routing networks, it is interesting to know exactly which nodes form the periphery of the network and thus have the highest worst-case response time to any other device. Thus, eccentricity can be seen as an extreme measure of centrality, i.e., the relative importance of a node in a graph.
Social networks can be modeled using graphs and the properties of the networks as well as the actors within them can be studied and explored using graph theoretic means. One particular application of graph theory in social network analysis is that of identifying the most 'important' or 'central' actor or actors in a social network.

II. METHODOLOGY

The paper deals with finding the eccentricity of a graph given its incidence or adjacency matrix. The proposed algorithm will output the same.

A. Input

Input from the user is either an incidence matrix or an adjacency matrix representation of the graph followed by the user entering the number of vertices and edges (in case of incidence matrix as input). If incidence matrix is entered, it is converted into its adjacency matrix representation. Then the algorithm is run and the eccentricity is found out.

B. Converting incidence matrix into adjacency matrix:

1. Traverse the incidence matrix column-wise.
2. For each column, identify the two row entries for which the matrix cell entries are 1 and 1 (in case of undirected graph) or 1 and -1 (in case of directed graph). Say they are u and v.
3. Update the adjacency matrix (initially filled with zeros) of the corresponding (u,v) to 1.
4. If the graph is undirected, update (u,v) and (v,u) of the adjacency matrix both as 1.
5. Repeat 2, 3, 4 till all columns of the incidence matrix are not seen.

1) Complexity Analysis: The time complexity of conversion is $O(V * E)$ because all edges are checked for their end vertices in a double loop traversing till $|E|$ and $|V|$ respectively.

C. Algorithm

The algorithm is a recursive approach to identify all possible path lengths present in the graph from every vertex to every other vertex. A vector declared globally stores all path lengths. We repeat the same above approach in order to reach from start vertex to the destination vertex. We change the start vertex to the next reachable vertex in every function call while keeping the destination fixed in order to trace the path. We initialize the boolean array to store the visited state for all the vertices present in the graph and $pathindex$ stores the length of the present path being traversed. Towards the end of the algorithm, when all path lengths have been calculated and stored, the maximum of all the lengths, which is the eccentricity of the graph, is outputted.

Since searching for adjacent vertices is a tedious procedure if representation of graph is of type matrix. Adjacency list proves to be a better alternative. So after the user enters the matrix, it is converted to adjacency list. Now the traversal to find the adjacent vertices becomes easier.

Algorithm 1 Eccentricity of Graph

```
1: procedure MAIN()
2:     Input the Graph
3:     for each i ∈ Vertex do
4:         for each j ∈ Vertex do
5:             Initialize($i, j$)
6:         end for
7:     end for
8:     Eccentricity Graph ← Longest Shortest path
9: end procedure

10: function INITIALIZE($s, d$)
11:     visited[ ] ← false // array of size V
12:     path[] // array of size V
13:     path_index ← 0
14:     Eccentricity(s,d,visited, path, path_index)
15: end function

16: function ECCENTRICITY($s, d, visited, path, path\_index$)
17:     visited[s] ← true
18:     path[path_index] ← s
19:     path_index ++
20:     if s == d then
21:         m.push_$back$(path_index)
22:     else
23:         for each i ∈ G.adjacency[s] do
24:             if !(visited[i]) then
25:                 Eccentricity(i,d,visited,path,path_index)
26:             end if
27:         end for
28:     end if
29:     path_index −−
30:     $visited[s] ← false$
31: end function
```

1) Complexity Analysis: The time complexity of the proposed the Algorithm is $O(V^3 * E)$.
This can be understood as, main() calls each vertex as destination from every other vertex, that is, each vertex becomes a source once and becomes a destination $V - 1$ number of times. Therefore, total calls of the recursive function from main() is of the order $O(V^2)$.
The recursive function calls itself for adjacent vertices of the current vertex being examined. Accessing the adjacent vertices in the list will take $O(E)$ time since we are accessing the adjacent vertices from the list representation of the graph. This recursion call will happen for maximum V vertices. Hence the total complexity of the *eccentricity()* function becomes $O(V * E)$.
Hence in totality, space complexity for the total program will be $O(V^2 * V * E)$ or $O(V^3 * E)$.

The space complexity is $O(|V * V|)$ for the adjacency matrix or incidence matrix and that of the list formed from the matrix will be $O(V + E)$. $O(|V|)$ will be the space complexity for the path and visited Boolean matrix. Hence giving the space matrix of $O(|V * V| + V + E)$ or $O(|V * V|)$.

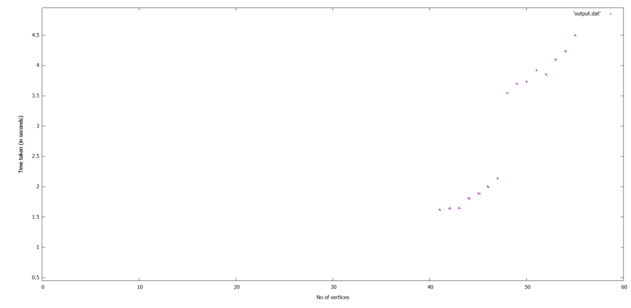


Fig. 6. Graph for average case complexity using randomized input. The purple dots represent points marked on a graph with time on Y-axis and number of vertices on the X-axis

III. RESULTS

The proposed algorithm prints the eccentricity of the user input matrix representation of a graph.

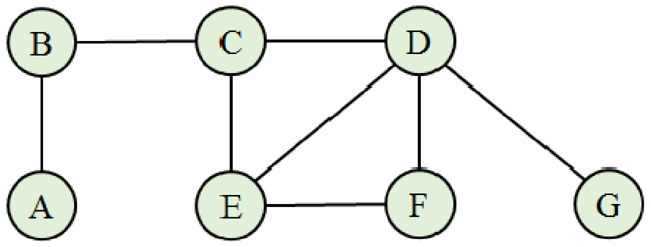


Fig. 7. A simple undirected graph

1) Input: Undirected graph adjacency matrix: Enter the type of matrix you want to traverse. Enter 1 for incidence, 2 for adjacency matrix: 2
Enter total number of vertices: 7
Enter the (n x n) Adjacency Matrix:
0 1 0 0 0 0 0
1 0 1 0 0 0 0
0 1 0 1 1 0 0
0 0 1 0 1 1 1
0 0 1 1 0 1 0
0 0 0 1 1 0 0
0 0 0 1 0 0 0

Eccentricity of the graph is: 6

2) Input: Undirected graph incidence matrix: Enter the type of matrix you want to traverse. Enter 1 for incidence, 2 for adjacency matrix: 1

```
Enter total number of vertices: 7
Enter the total number of edges in your graph: 8
Enter the type of the graph. Enter 1 for directed or 2 for
undirected: 2
Enter the (v x e) incidence matrix:
1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0
0 0 0 1 1 0 1 1
0 0 1 0 1 1 0 0
0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 1
```

Eccentricity of the graph is: 6

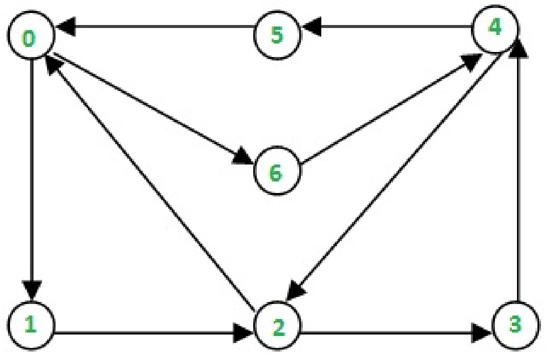


Fig. 8. Input directed graph

3) Input: Directed graph adjacency matrix: Enter the type of matrix you want to traverse. Enter 1 for incidence, 2 for adjacency matrix: 2

```
Enter total number of vertices: 7
Enter the (n x n) Adjacency Matrix:
0 1 0 0 0 0 1
0 0 1 0 0 0 0
1 0 0 1 0 0 0
0 0 0 0 1 0 0
0 0 1 0 0 1 0
1 0 0 0 0 0 0
0 0 0 0 1 0 0
```

Eccentricity of the graph is: 6

4) Input: Directed graph incidence matrix: Enter the type of matrix you want to traverse. Enter 1 for incidence, 2 for adjacency matrix: 1

```
Enter total number of vertices: 7
Enter the total number of edges in your graph: 10
Enter the type of the graph. Enter 1 for directed or 2 for
undirected: 1
Enter the (v x e) incidence matrix:
1 0 -1 0 1 0 -1 0 0 0
0 0 0 0 0 0 1 -1 0 0
0 0 0 0 -1 1 0 1 -1 0
0 0 0 0 0 0 0 0 1 -1
0 -1 0 1 0 -1 0 0 0 1
-1 1 0 0 0 0 0 0 0 0
0 0 1 -1 0 0 0 0 0 0
```

Eccentricity of the graph is: 6

IV. DISCUSSION

The complexity of the given algorithm is the same for best, worst and average cases because it is traversing all possible paths from every vertex to every other vertex, and calculating the path lengths.
There could be another solution to this problem. The longest path can be calculated the length of which will be the eccentricity of the graph, but, the longest path problem for a general graph is not as easy as the shortest path problem because the longest path problem doesn't have optimal substructure property. In fact, the longest path problem is NP-Hard for a general graph.

REFERENCES

[1] J. Cook, M. Wootters, V. Williams, R. Verma, S. Hildick-Smith, and G. Valiant, *Graphs.* [On-line]. Available: https://web.stanford.edu/class/cs161/Lectures/Lecture9/CS161Lecture09.pdf. [2017].

[2] N. Deo. *Graph Theory with applications to engineering and computer science*. Prentice-Hall of India, 2007, pp. 45-48.

[3] W. Mathworld *Graph Eccentricity.* [On-line]. Available: http://mathworld.wolfram.com/GraphEccentricity.html.

[4] Yu, G. and Feng, L., 2013. On connective eccentricity index of graphs. MATCH Commun. Math. Comput. Chem, 69(3), pp.611-628. [2013].

[5] Bang Ye Wu, Kun-Mao Chao. *"Spanning Trees and Optimization Problems: A note on Eccentricities, diameters, and radii"*. Chapman Hall/CRC Press, USA. [2004].

[6] W. Mathworld *Graph Diameter.* [On-line]. Available: http://mathworld.wolfram.com/GraphDiameter.html.

[7] Ilic, Aleksandar, Guihai Yu, and Lihua Feng. *"On the eccentric distance sum of graphs."* J. Math. Anal. Appl 381, no. 2. pp. 590-600 [2011].