Identify and count the disconnected subgraphs from adjacency matrix

Ajay Pilaniya, Shubham Swarnkar, and Ashok Maloth

Abstract—This report studies about the various method to
find the disconnected subgraphs using Adjacency Matrix representation (undirected and directed). This paper also discusses the time complexity of all the methods.
Keywords - Graph Reperesentaion, BFS, DFS, Kosaraju

Index Terms—

1 Introduction

A graph is connected when there is a path between every pair of vertices. In a connected graph, there are no unreachable vertices. A graph that is not connected is disconnected. A graph G is said to be disconnected if there exist two nodes in G such that no path in G has those nodes as endpoints. A subgraph is a subset of given vertices and edges of the graph. The term disconnected subgraph represents the part of the graph where all vertices are connected.

2 Problem Description

Given a graph G in the form of adjacency matrix ADJ[N][N] where N is the number of vertices. ADJ[i][j] will be 1 if there is an edge between i and j otherwise it will be 0. Now we need to find and display the number of disconnected subgraphs in G.

3 Approach

3.1 Using Breadth First Search (BFS) for undirected graph

```
Procedure Dicsconnected_Components ( ADJ , N ) :
    visited [N] = {false}
    count = 0
    FOR   i  = 0 to N-1 :
        IF ( visited[i] == false ) :
    BFS(i,visited,ADJ , N)
    count = count + 1
       END IF
    END FOR
    END Procedure
    Procedure BFS ( source , visited , ADJ , N) :
    QUEUE Q
    Q.enqueue(source)
    Visited[source] = true;
    WHILE ( Q is not empty ) :
    V = Q.dequeue()
    Print V
    FOR all T adjacent of V :
    IF ( visited[T] == false ) :
    visited[T] = true;
    q.enqueue(T)
    END IF
    END FOR
    END WHILE
    END Procedure
```

- Time Complexity : $O(N^2)$
- Space Complexity : O(N)

3.2 Using Depth First Search (DFS) for undirected graph

```
Procedure Dicsconnected_Components ( ADJ , N ) :
    visited [N] = {false}
    count = 0
    FOR   i  = 0 to N-1 :
        IF ( visited[i] == false ) :
    DFS(i,visited,ADJ , N)
    count = count + 1
       END IF
    END FOR
    END Procedure
    Procedure DFS ( S , visited , ADJ , N) :
    Visited[source] = true;
    Print source
    FOR all T adjacent of S :
    IF ( visited[T]  == false) :
    DFS (T , visited , ADJ , N)
    END IF
    END FOR
    END Procedure
```

- Time Complexity : O(N2)
- Space Complexity : O(N)

3.3 Using Kosaraju's Algorithm for directed graph

```
Procedure Dicsconnected_Components ( ADJ , N ) :
    visited [N] = {false}
    count = 0
    STACK ST
```


- *Email: ajaypilaniyaap@gmail.com (Ajay Pilaniya)*
- *Email: iwm2014001@iiita.ac.in (Shubham Swarnkar)*
- *Email: iim2014006@iiita.ac.in (Ashok Maloth)*

```
FOR  i = 0 to N-1 :
IF ( visited[i] == false ) :
Fill_Stack (i , visited , ADJ , N , ST)
END IF
END FOR
Reverse_Edges(ADJ , N )
visited[N] = {false}
WHILE ( ST is not  empty ) :
v =  ST.pop()
IF ( visited [ v ] == false )
DFS (v,visited,ADJ , N)   // DFS is same as Algorithm III(B)
count = count + 1
END IF
END WHILE
Procedure Fill_Stack ( S , visited , ADJ , N , ST ) :
Visited[s] = true
FOR all T adjacent of s :
IF (visited[T] == false) :
Fill_Stack (T , visited , ADJ , N ,ST)
END IF
END FOR
ST.push ( S )
END Procedure
Procedure Reverse_Edges(ADJ , N ) :
TMP  = ADJ
FOR i = 0 to N-1 :
FOR j = 0 to N-1
ADJ [ i ][ j ] = TMP[ j ][ i ]
END FOR
END FOR
END Procedure
```

- Time Complexity : $O(N^2)$
- Space Complexity : O(N)

4 Time Complexity Analysis

5 Conclusion

We observed that for undirected graphs we can find disconnected subgraphs by using BFS and DFS but for directed graphs we can not find using simple BFS/DFS so we applied Kosaraju's algorithm. In this document we used adjacency matrix for the input so the time taken for BFS and DFS will be $O(N^2)$.

TABLE 1
Time taken for each algorithm

N	A	B	C
600	0.03	0.03	0.79
700	0.06	0.06	1.21
800	0.08	0.09	1.72
900	0.11	0.11	2.57
1000	0.14	0.15	3.52
1100	0.21	0.20	4.72
1200	0.28	0.28	6.06
1300	0.29	0.29	7.95
1400	0.37	0.37	11.54
1500	0.40	0.42	15.99