Identify the disconnected subgraphs from Adjacency Matrix.

By Ajay Pilania (IRM2014007)
Shubham Swarnkar (IWM2014001)
Ashok Maloth (IIM2014006)

Introduction

A graph is connected when there is a path between every pair of vertices. In a connected graph, there are no unreachable vertices. Our aim is to find the disconnected subgraphs in given directed and undirected graphs.

Algorithms

There are three algorithms to find disconnected subgraphs.

➔ **BFS**
Using breadth first search for undirected graphs.

➔ **DFS**
Using depth first search for undirected graphs.

➔ **Kosaraju's Algorithm**
Kosaraju's algorithm is used for directed graphs.

BFS (Breadth First Search)

```
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
```

```
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)
```

DFS (Depth First Search)

```
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
```

```
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)
```

Kosaraju's Algorithm

```
Procedure Dicsconnected_Components ( ADJ , N ) :
    visited [N] = {false}
    count = 0
    STACK ST
    FOR  i = 0 to N-1 :
        IF ( visited[i] == false ) :
            Fill_Stack (i , visited , ADJ , N , ST)
    Reverse_Edges(ADJ , N )
    visited[N] = {false}
    WHILE ( ST is not  empty ) :
        v =  ST.pop()
        IF ( visited [ v ] == false )
            DFS (v,visited,ADJ , N)   count = count + 1
```

Kosaraju's Algorithm

```
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)
    ST.push ( S )
```

```
Reverse_Edges(ADJ , N ) :
    // Transpose the ADJ matrix to reverse edges
    TMP  = ADJ              // copy ADJ to TMP
    FOR i = 0 to N-1 :
        FOR j = 0 to N-1
            ADJ [ i ][ j ] = TMP[ j ][ i ]
```

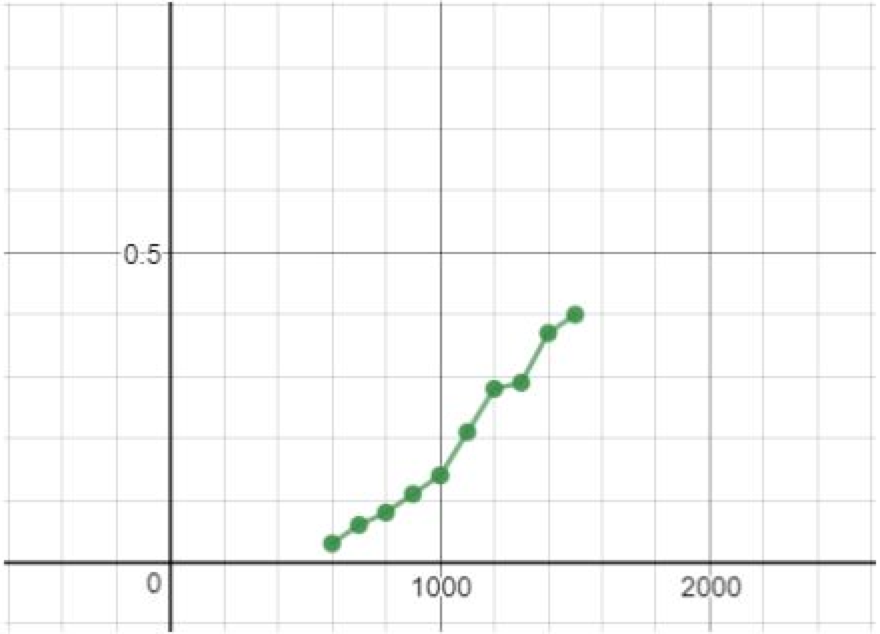

Fig. 1. Graph for Algorithm A (X=input size;y=time) |Algorithm B will have same graph

Time Complexity for BFS and DFS

Time Complexity : $O(N^2)$

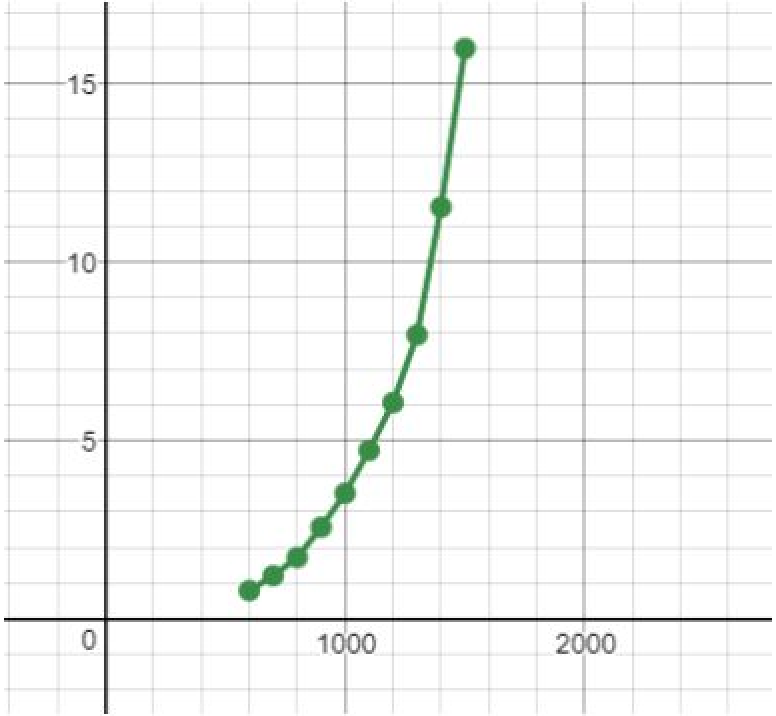

Fig. 2. Graph for Algorithm C (X=input size;y=time)

Time Complexity for Kosaraju's Algorithm

Time Complexity : $O(N^2)$


Thank you.