HW-5

CSCI-665

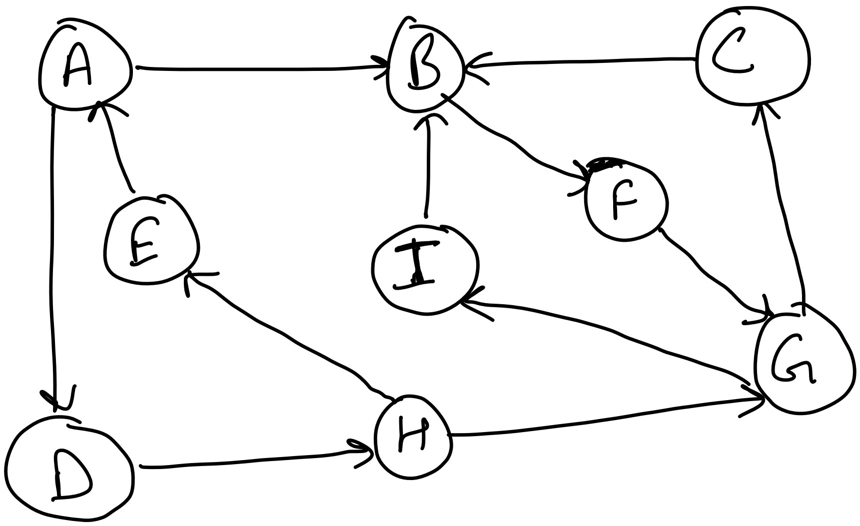
Karan Ahluwalia

ka7982

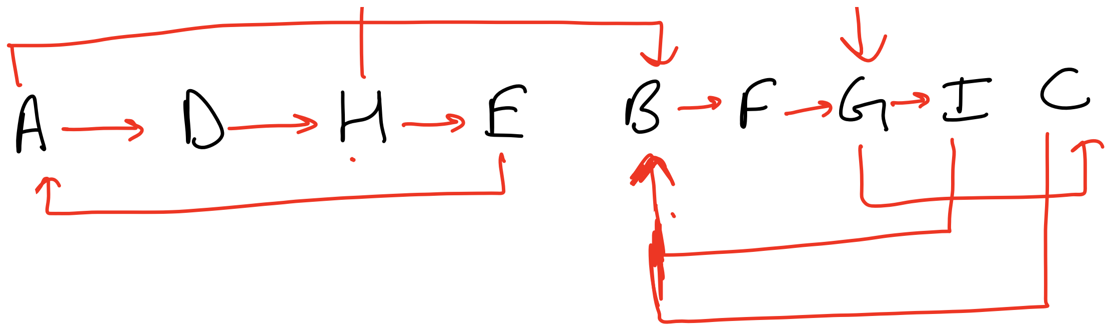
Rishabh Arora

ra8851

Problem-1



Vertex	Visit	Finish
A	1	9
B	2	5
C	5	1
D	7	8
E	9	6
F	3	4
G	4	3
H	8	7
I	6	2

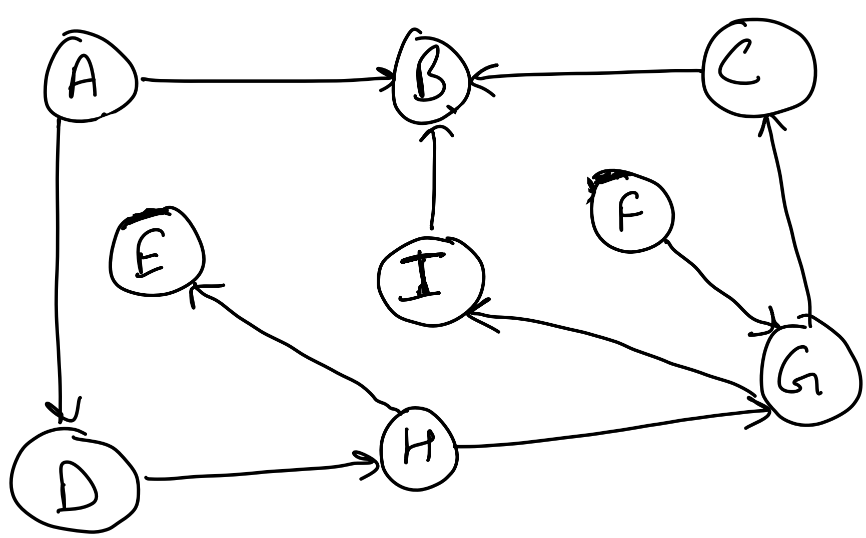


Number of edges pointed the wrong way are 3.

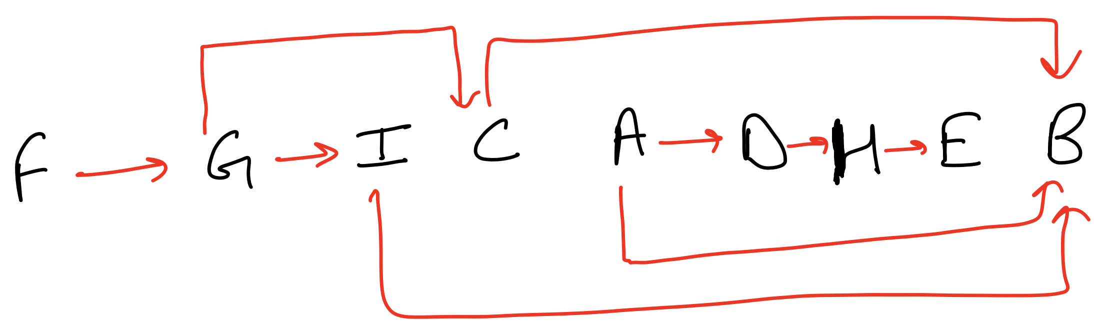
Removing only 2 edges can make the above graph Acyclic:

B → F

E → A



Vertex	Visited	Finish
A	1	5
B	2	1
C	6	6
D	3	4
E	5	2
F	7	9
G	8	8
H	4	3
I	9	7



Thus, the algorithm fails to correctly identify the minimum number of edges that must be removed from the directed graph to make it acyclic.

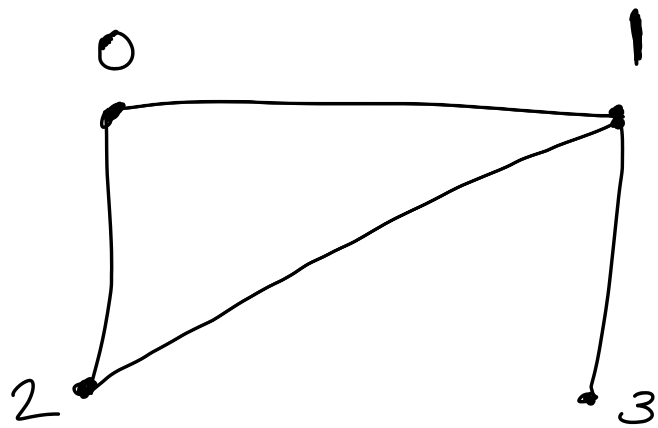
Problem - 2

We are using recursive DFS to find the fewest steps of directions possible which allows to move from point A to point B.

Time Complexity is $O(m+n)$ since we are using DFS with adjacency list.

Example-1

$n = 4$, $m = 4$, start $= 0$, end $= 3$



Path 1: $0 \rightarrow 1 \rightarrow 3$

Count array

1	1	0	0
0	1	2	3

Sum for possible directions $= \boxed{2}$

stored as minimum

Path 2: $0 \rightarrow 2 \rightarrow 1 \rightarrow 3$

Count array

0	1	2	3
1	1	0	0

(0,1) is unvisited.

Sum = 2

Here, Count array is initialized to 0 in the beginning. We only increase the count when the number of unvisited edges from a vertex is > 1. Otherwise, the counter for that vertex remains 0. As soon as we reach the destination, the last visited vertex count is reintialized to zero. which helps in recalculating sum for other paths.

For optimization, we only run the DFS if the path being evaluated does not cross the minimum sum we already found.

<u>Correctness:</u>

Since, we are tracking all paths possible in the graph from A to B and keeping track of minimum possible directions.

Problem-3

We are looking for strongly connected components in the graph.

Time complexity is $O(m+n)$ since we either run BFS or DFS using an adjacency list.

We consider each strongly connected component as an independent node and evaluate all possible incoming and outgoing edges to and from each node. $O(m)$ complexity.

Visited Nodes Order

Finish Nodes Order

SCC calculation done in $O(m+n)$ time.

Every vertex and edge is visited once, Hence, running in $O(m+n)$ time.

Example:

Vertex	Neighbor
1	2, 0
2	3, 0
3	1, 4, 0
4	5, 0
5	8, 6, 0
6	4, 7, 0

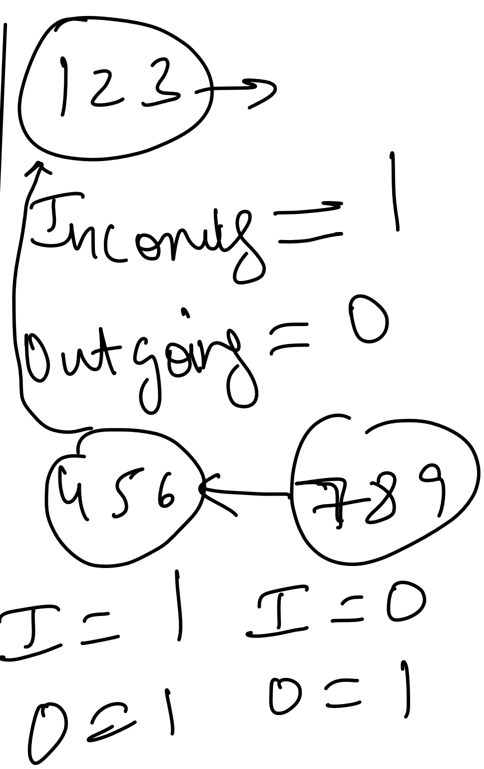
Visited	Finish
1	9
2	8
3	7
4	6
5	5
6	4

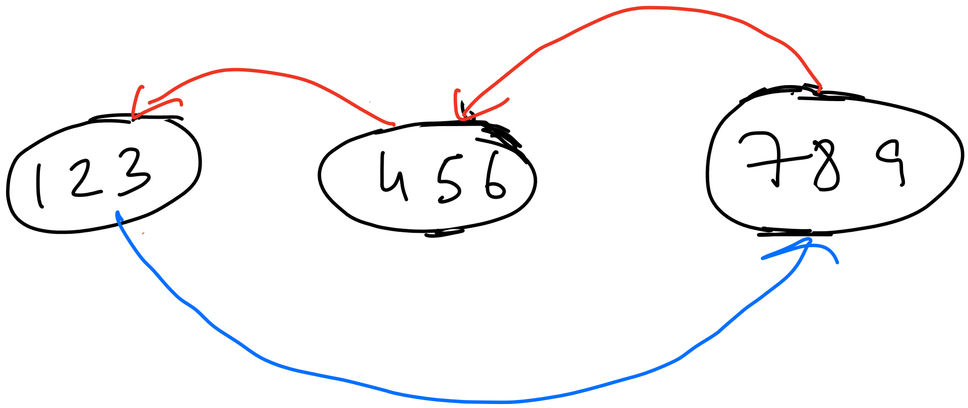
7	8, 0	7	3
8	9, 0	8	2
9	7, 0	9	1

Order : 1, 2, 3, 4, 5, 6, 7, 8, 9

Reverse Adjacency List :

Vertex	Neibhors
1	3, 0
2	1, 0
3	2, 0
4	3, 6, 0
5	4, 0
6	5, 0
7	6, 9, 0
8	5, 7, 0
9	8, 0





Edge from SCC (1,2,3) to (7,8,9) would help in making graph a connected graph.

Ans 4

A list(graph_list) of size m(number of edges) is initialized, which will store the set the vertex belongs to.

Then afterwards, we create three graphs which store, all elements in sets A & B and a full graph, containing all the elements.

After graphs are created, we calculate the strongly connected components of graphs in Set A & Set B.
For calculation of SCC, we have used the DictGraph. This a graph without weights. Weights are not needed for finding SCC.

Initialization of lists and SCC computation won't take more than $O(m+n)$, where

m = Number of edges

n = Number of vertices

Now, after finding the SCC in graphs, we will find minimum spanning Tree.

If the total number of SCCs (in both sets) is more than 3, then we can not find the solution, therefore, we will check for a total of 1, 2 & 3 SCCs.

Now, if SCCs is 3, that means there can be two

SCCs in a set and one in another. After find which one has more SCCs, lets say Set x. Now, we need to find the minimum weight edges which connects the SCCs of Set x to the SCC of the other Set. After finding the two edges, we add all the edges from set A which do not leave the set, all the edges from set B which do not leave the set and the two minimum weight edges computed earlier and add them all in a graph and find the minimum spanning tree.

This sol^n will give the minimum spanning tree as we need to join both SCC from set x and the minimum weighted edge will give the most cost efficient tree.

Now, if SCC is 1.
which means there is only one SCC in either sets. This means that we only need to find the minimum spanning of the graph in set whose count of SCC is one. No further calculation is need for this one.

Now, if total Sccs = 2
with one Scc each in both
the sets.
This the most computation heavy
condition.

In this case, we first find the two
minimum weight edges which travels
from one set to another. Now
after these edges are computed (This
is a O(m) task), we make
a graph with all the edges
from set A & set B and
the minimum calculated edges and
find the minimum spanning tree.
This will give our solution.

Now, Complexity of making a graph = $O(m+n)$

SCC computation = $O(m+n)$

Kruskal's algo to form MST = $O(m \log n)$