# CSE 221

Assignment 3

Mazidul Islam Rabbo

section 9

24301500

1a) $\sum deg(v) = 2m$

In any graph, one edge creates one indegree for reciever and one out degree for sender, so 2 degrees per edge in total, so number of degrees (total) is 2 x no. of edges

b) maximum number of edges from n nodes is given by, $\frac{n(n-1)}{2}$, there are 9 nodes in total, so total edges that can be there is $\frac{9 \times 8}{2} = 9 \times 4 = 36$, there are 14 edges here already, so 22 more edges can be added.

2a) We want to apply Dijkstra's Algorithm.

First we set distance[E] = 0, all others $\infty$.
Now we relax this node, From E visit F, set distance[F] = 2.
Relaxing E done. Now F has 2 path, we always choose lowest available ~~totaldist~~path, so we go to A, set distance[A] = 2+3 = 5
now we go to C from A. Set distance[C] = 5+1 = 6.
and also goto B from A so. distance[B] = 5+4 = 9. Now, from C goto D, and set distance[D] = 6+1 = 7. and from D goto G, set distance[G] = 7+2 = 9

so we end up with,

E=0, F=2, A=5, C=6, D=7, B=9, G=9.

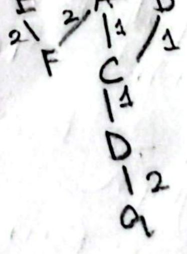
b) We need to find MST for this. We store all the edges and their weights in an array then sort that array in ascending order according to the weights.

Keep all nodes in seperate sets at first

{A} {B} {C} {D}
{E} {F} {G}
{H}

A—C—1
B—C—1
C—D—1
D—G—2
B—D—2
E—F—2
A—F—3
C—G—4
A—B—4
A—E—6
F—C—9

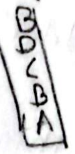
now take each edges in the order, if the nodes are different sets, we select those edge and unify the sets, if they are in same set, we discard that edge.



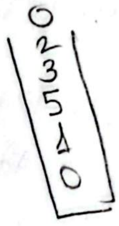
→ this is our connection, with total weight = 9

3 a) Adjacency Matrix

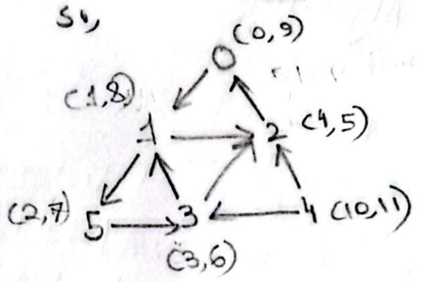
b) i) Run DFS from any point of graph (say 0) and find outtime for all nodes (at what time does our node get out of recursion stack), use a global timer for this. Now make a transpose graph from there and run a DFS in descending order on nodes according to outtime. While running DFS, if we enter an already visited node, all nodes in the recursion stack from top to that visited node are SCC.

for example, 

} here. BCD are SCC.

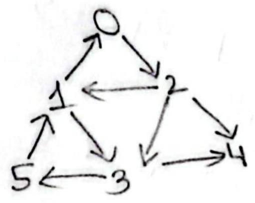


So,



order of visit 4, 0, 1, 5, 3, 2

Transpose graph



cant go anywhere

{4} is SCC
{0,1,2} is SCC
{0,1,2,3,5} is SCC

so final ans {4} is SCC, {0,1,2,3,5} is SCC

ii) If an edge gets removed, we see if u and v were part of same SCC, if no, no need to do anything, if yes, we start DFS from u to try and reach v; if we can reach v, no change needed. If no way to get to v, we can only focus on the specific nodes from that SCC, and run the same algorithm only on edges containing those nodes

c) Step 1: Run DFS and record when each node finishes (find out-times)
step 2: Reverse all the connections to find transpose graph
step 3: Run DFS again on transposed graph in the order of the out time (descending), if all nodes visited in one run then all the buildings are accessible, else no.

4. a 1. Dijkstra is prefferd

2.

priority queue

[ ~~(0,A)~~, ~~(3,B)~~, ~~(6,C)~~, ~~(7,E)~~, ~~(9,G)~~, (10,H)
~~(5,C)~~

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| distance = | 0 | ~~∞~~ 3 | ~~∞~~ ~~6~~ 5 | ∞ | ~~∞~~ 7 | ∞ | ~~∞~~ 9 | ~~∞~~ 10 |

b. Each city v is split into two nodes, v normal which represents arriving at city v after a light road, in this state, we can take any route, and $v_{restricted}$, which represents arriving at city v after a heavy road. In this state, we can only take light road.

Then we construct edges,

If $w \leq 4$ (light road), create an edge $u_{normal} \rightarrow v_{normal}$ with weight w. and u restricted to $v_{normal}$ with weight w.

If $w > 4$, create an edge $u_{normal} \Rightarrow v_{restricted}$ with weight w. only.

Now run dijkstra on this graph

Final result is distance to $x = \min(\text{distance}(x_{normal}), \text{distance}(x_{restricted}))$

c) No, this is a directional graph so it wont work. To make it work the graph needs to be made bidirectional

5 a)

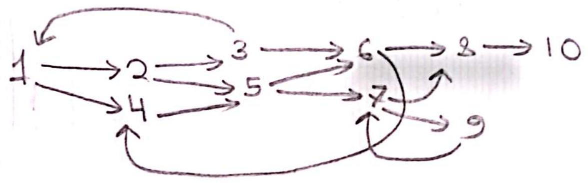
$$\begin{tikzcd}
T1 \arrow[r] \arrow[dr] & T3 \arrow[dr] & & \\
 & T4 \arrow[r] \arrow[dr] & T6 \arrow[r] & T8 \\
T2 \arrow[ur] \arrow[dr] & & T7 \arrow[ur] & \\
 & T5 \arrow[ur] & &
\end{tikzcd}$$

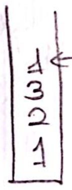
b) T1, T2, T3, T4, T5, T6, T7, T8
$\longrightarrow$

c) yes, T1 and T2 have same indegree of 0, so they can be alternated, or T3, T5 have same indegree of 1 so it can be alternated, if we do T1 and T2 first, we can also alternate T4 with T3 and T5 or do T3 right after T1 and such

d) If T8 → T2 is introduced, a cycle is introduced and there would no longer be any valid sequence, as it becomes impossible to start any task within that loop.

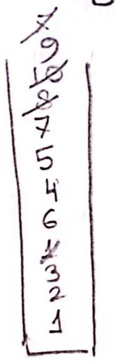
6. a) Cycle detection: DFS as it can identify back edges or ancestors in recursion stack much more easily.





← visiting node detected, so cycle exists

Edge classification: DFS to classify edge into 4 types (Tree, Back, Forward and Cross).



3→1, 9→7 — back edge

1→2, 2→3, 3→6, 6→4, 4→5, 5→7, 7→9, 7→8, 8→10 — tree edges

5→6, cross edge

would be better if we weren't detecting cycle and doing edge classification as

Memory constraints – BFS ^ its more of tall depth graph rather than a wide graph, BFS uses less memory for less wide graph while DFS uses less memory for less tall graph. But for our current case, we use DFS

b) If we reach a critical node, rather than starting a recursion on its neighbor, we create a local queue to work on all its neighbors only then proceed normally with DFS.

c) A cycle exists if we encounter an edge (u,v) where v is currently in the recursion stack, so

$d(v) < d(u)$ and $f(v)$ is not yet set.

for back edge: $d(v) < d(u) < f(u) < f(v)$

for cross edge: $f(v) < d(u)$

d) Cycle detection becomes simpler, any edge pointing towards an previously visited node that is not the immediate parent is a cycle

For Edge classification, Forward and cross edges cant exist

DFS still better for traversal

c) True, Assume there is a cycle in the reachable subgraph, In any directed cycle, there must be one node v that is discovered first in DFS. Eventually the traversal must explore an edge (u,v) that closes the loop. Since v was the first in the cycle to be discovered, it is an ancestor of u in the DFS tree. By definition, the (u,v) is a back edge

Therefor if no back edge is detected, no cycle can exist.

8) $O(V+E) \rightarrow$ time complexity

In worst case, DFS visits every reachable Vertex once and examines every outgoing edge once. The critical node Constraint doesnt change this.

$O(V) \rightarrow$ space complexity

In worst case, the stack depth is V, or the graph is a line.

The critical node constant adds at most O(out degree) space, where out degree at max is $V-1$, but increasing width means depth decreases, it still remains O(V).