ARSHITHA BASAVARAJ

ASSIGNMENT 7

1a. To find the strongly connected components of a graph $G = (V, E)$, transpose of $G$, $G^T$ is used.

$\Rightarrow G^T = (V, E^T)$ [ $O(V+E)$ to create this transpose graph, $G^T$ ]

The reasoning is that $G$ & $G^T$ have the same strongly connected components.

i.e., If u & v are reachable from each other in graph G, then u & v are also reachable from each other in graph $G^T$.

Only then ~~it~~ they are strongly connected.

The strongly connected components of a directed graph, G can be found using ~~2~~ two DFS's, one on G & one on $G^T$.

Scanned by CamScanner

PSEUDO CODE

Strongly-connected-components (G)

1. call DFS(G) to compute finishing times F[u] for each vertex u.

2. Compute $G^T$

3. call DFS($G^T$), but in the main loop of DFS, the vertices are traversed in decreasing F[u]

4. output the vertices of each tree in depth first forest formed in line 3 as a separate strongly connected component

This algorithm runs in $\Theta(V+E)$ time.

Q a7 Logic:

In the already available MST, we find the path between u and v. This would be a sub-path in the MST.

Once path is found between u and v, we find the edge within that path that has the maximum weight.

If this max. weight is greater than the weight (w) of the new edge that we are adding, then that edge is removed & replaced with our new edge.

## Algorithm

① Using the predecessor array $\pi[v]$, find the path between u to v.

a. while tracking the path store the max. weight of the path & the corresponding edge.

② if the edge with max. weight > the new edge's weight.

a. remove the existing edge

b. reverse the parents/predecessors of the vertices in the path.

c. predecessor [u] = v.

Scanned by CamScanner

∵ for ①, we'll have to traverse at most ~~[illegible]~~ all the vertices, ① will ~~the~~ take $\Theta(V)$

② $\Theta(1)$ since we'd only be making changes to the predecessor array.

$\Rightarrow$ overall runtime is $\Theta(V)$

Scanned by CamScanner

2b) Logic: By modifying the Prim's algorithm to have the new vertex and initialising it's key to infinity & running the Prim's algorithm we get the updated MST.

Modified Prim (G, root, newVertex)

$\Theta(V+1)$ ⇐ key [newVertex] = $\infty$

$\Theta(V)$ key [$v$] ← $\infty$ $\forall$ $v \in V$ , key [root] = 0

$\Theta(V)$ priority Queue ← all vertices

$\Theta(V)$ while (priority Queue != empty)

$\Theta(V \log V)$ $u$ ← extract-min (priority Queue)

$\Theta(2E) = \Theta(E)$ for each $v$ adj to $u$

if $v \in$ PQ & $w(u,v) <$ key[$v$]

$\Theta(E \log V)$ key [$v$] ← $w(u,v)$

$\Theta(E)$ $\pi$ [$v$] ← $u$

$\Theta(1)$ dequeue ($u$)

$\Theta((E+V)\log V)$

$\Rightarrow \Theta(E \log V)$

Depending on the data structure we use and how dense the graph is runtime varies.

| | unordered linked list | min-heap |
|---|---|---|
| For a dense graph $E \approx V^2$ | $\Theta(V^2)$ | $\Theta(V^2 \log V)$ |
| For a sparse graph $E \approx V$ | $\Theta(V^2)$ | $\Theta(V \log V)$ |

1b) Algorithm

① Number the vertices from 0 to $n-1$.

② Add an edge from vertex $i$ to $(i+1) \bmod V$

(only if the edge doesn't already exist)

This connects all the vertices in a cycle which itself is biconnected.