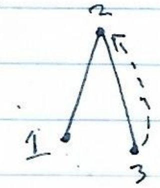
# HW 7

Christopher Marcom

## 22.5-3

Prof Bacon's Algorithm will not work.

Suppose a graph G:

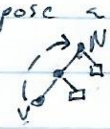



where G has verticies {1, 2, 3} and edges (2,1), (2,3), (3,2)

A DFS starting at 2 would have a finish time less than a DFS at 1, because a DFS at 2 could explore 3 before 1.
A DFS at 3 could explore the entire graph in a single SSC.
Due to this we would not get the expected SSCs of {2, 3} and {1} and a DFS at 3 would give us {3, 2, 1} even though there is no edge (1,3).

## 22-1 (A)

Suppose a tree $T_1$:




At (V,N) we have a back edge. In BFS, there is already a path between V & N which does not involve moving up the tree. This tree contradicts property 1 since the only children in a tree are a single edge away which means there cannot be another path to the child that is ≠ 1 edge away

Suppose a tree $T_2$:


Similarly, in $T_2$, there cannot be any forward edges as the forward child (V) would already be processed in BFS, meaning it cannot have another path to it from an ancestor (N).

To verify property 2, we know that BFS only explores unexplored, adjacent nodes. the distance from root to vertex is at least u.d. +1, but if we backtrack from v to root, it is at most u.d. +1 (on edge (u,v)), thus verifying the property

Suppose a tree $T_3$:


In BFS, cross edges must be within ±1 depth of a node otherwise we violate property 1, which we just proved. Consequently, u and v are commutative in property 3 since trees are unordered and v may be processed before u, thus proving property 3 where cross edges have a depth of ±1.

(3) To find and analyze an algorithm to find T of G, we must consider 2 cases:
a) increasing the weight of edge e
b) decreasing the weight of edge e

a) If e is increased, e must be in T or it is not part of the MST when the weight is increased, otherwise $T_{new}$ becomes a graph with one cycle. If e is an edge in T, then we delete e, separating its children into 2 subtrees we then use BFS/DFS to find the min weight edge of the 2 subtrees of e, which takes $O(V+E)$ time. We must mark all vertices during BFS/DFS so that when we retrace G, we find the min weight edge by searching the marked endpoints in $O(E)$ time.

b) If e is decreased, nothing needs to be changed if e is in T, taking $O(E)$ time. However, if e was in T, say, we delete e, then we still get 2 subtrees, but e becomes ideal with respect to its subtrees since decreasing the weight of e makes tracing T quicker.
However, if e is not in T, G must contain a cycle since there is a unique path from the root of T to e. We would simply use BFS/DFS to find the max weight of e in $O(V+E)$ time.

[Source: cs.jhu.edu/teaching/~zlin39/hw6.pdf]

4) a) Let us implement an algorithm using a priority queue and a variant of Dijkstra's algorithm, which would produce a shortest path tree.
Also, keep an array with all possible distances and point to the current min distance at all times.
Additionally, each node in the array points to a list of elements with the distance in the head.
EXTRACT-MIN takes $O(1)$ time as it simply pops from the list, moving the pointer to a non empty list. This takes $O(VW)$ time since there are up to $VW$ distances in the queue ($V$ vertices with $W$ weights).
DECREASE-KEY takes $O(1)$ time as it pops elements from one list and pushes to another.
This takes worst case: $O(VW + V + E)$ time as DECREASE-KEY only decreases to the current min pointer's value, which may be at the end of the array.

[Source: Wikipedia]

b) I did not have time to test this due to lack of testing data! Please deduct the necessary points.