CSE 373 Analysis of Algorithms April 6, 2014

Solutions to HW 3

Problem 1

[5–4] Let T be the BFS-tree of the graph G. For any e in G and $e \notin T$, we have to show that e is a cross edge. Suppose not, suppose $e = (x, y)$ is not a cross edge. Wlog, say x is an ancestor of y. This means that x was discovered before y in the BFS traversal. The way BFS works, all of x's children would have been discovered at the next level. This means that $e = (x, y) \in T$. But this gives a contradiction.

Problem 2

[5–7] Given pre-order and in-order traversals of a binary tree, is it possible to reconstruct the tree? Yes. We can do it as follows:

1. The first element of the pre-order traversal is the root.
2. Find the root in the in-order traversal, the elements to its left are in the left subtree and to its right are in the right subtree.
3. Recursively follows the above steps to construct the entire tree.

Given the pre-order and post-order traversals of a binary tree, is it possible to reconstruct the tree? No it is not possible to re-construct a general binary tree, but if the binary tree is *full*, it can be reconstructed. Take the first element of pre-order, that is root of tree. Since the tree is full, the element x must be left child of root. To find all the elements in the left-subtree, find x in the post-order traversal. Everything before x in the post-order belongs to the left subtree. Do this recursively to construct the entire tree.

Problem 3

[5–12] **For adjacency matrix:**

```
for i = 1 to V do
    for j = 1 to V do
        G^2(i, j) = 0
        for k = 1 to V do
            if G(i, k) == 1 and G(k, j) == 1 then
            G^2(i, j) == 1
```

For adjacency list

```
for u ∈ V do
    for v ∈ Adj(u) do
        for w ∈ Adj(v) do
            Add edge (u, w) in G^2
```

[5–13] **1. Minimum-size vertex cover of a tree**
In a vertex cover we need to have at least one vertex for each edge. Every tree has at least two leaves, meaning that there is always an edge which is adjacent to a leaf. Which vertex can we never go wrong picking? The non-leaf, since it is the only one which can also cover other edges! After trimming off the covered edges, we have a smaller tree. We can repeat the process until the tree as 0 or 1 edges. When the tree consists only of an isolated edge, pick either vertex. All leaves can be identied and trimmed in $O(n)$ time during a DFS.

2. Minimum-weight vertex cover of a tree
1. Perform a DFS, at each step update $Score[v][include]$, where v is a vertex and $include \in \{true, false\}$ indicates, whether we include vertex in the cover or not.
2. If v is a leaf, set $Score[v][false] = 0$, $Score[v][true] = w_v$, where w_v is the weight of vertex v.
3. During DFS, when moving up from the last child of the node v, update $Score[v][include]$:
$Score[v][false] = \sum_{c \in children(v)} Score[c][true]$
$Score[v][true] = w_v + \sum_{c \in children(v)} min(Score[c][true], Score[c][false])$
4. Extract actual cover by backtracking $Score$

3. Minimum-weight vertex cover of is a tree
Same as **2.**, just replace degree with weight.

[5–14] If the tree has more than one vertex, then yes. The remaining vertices are still the vertex cover because for every edge $e \in E$ incident on the leaves, their other end-point is still in the remaining tree.

Problem 4

[5–19]. Do a BFS of the tree starting at any node s and marking the distance of the nodes from the root s. We need to pick leaves (u, v) such that $d(u, s) + d(v, s)$ is the largest. One way to do this is to take the leaf x with maximum $d(x, s)$ and do a BFS with root $= x$. In the new tree pick the leaf y such that $d(x, y)$ is the maximum among all leaves. The diameter of the tree is then $d(x, y)$.

Problem 5

[5–25] Do a DFS from each vertex, if the DFS tree contains all other vertices of the graph, return true. Running time $O(n^2)$.

Problem 6

```
dfs (node u)
   for each node v connected to u, do
      set visited [v] = true
      dfs (v)


for each node u, do
   if u is not visited, then
```

set visited [u] = true
do component + = 1
dfs (u)

PROBLEM 7

[6–4] If the graph has distinct edge weights then it has a unique spanning tree and both Prim's and Kruskal's will output the same tree. But there exists two edges with the same weight, and the tie is broken arbitrarily, then both the algorithms may output different trees.

[6–5]. Both Prim's and Kruskal's algorithm work with negative edge weights. It is because the correctness of the algorithm does not depend on the weights being positive. Kruskal sorts the edges according to weights (which doesn't change when negative weights are present) and chooses the best edge from it each time. In Prim's the tree evolves by adding the least weight edge that connects a tree vertex to a non-tree vertex. The edge selection is not effected by negative weights.

PROBLEM 8

[6–9] Note that you have to include all vertices of G and T must be connected (but it does not necessarily have to be a spanning tree).

This problem is different from minimum spanning tree – consider the graph on {a,b,c} which is a triangle and each edge has weight -1. A min-spanning tree would include any two edges like {ab, ac} with weight -2, while min-connected subset would include all three edges with weight -3.

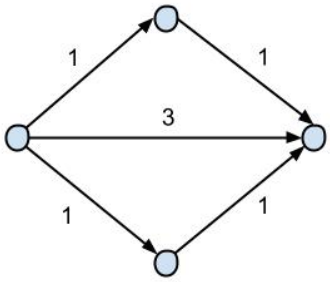
An efficient algorithm to compute minimum weight connected subset T is:

1. Find the minimum spanning tree T of G.
2. For each edge $e \in G$ with $w_e < 0$, add e to T.

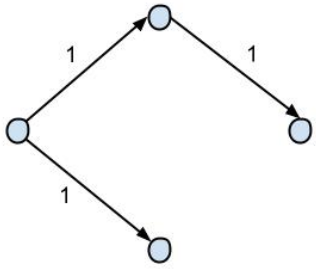
This takes $O(E \log V)$ if we use Kruskal for step 1.

PROBLEM 9

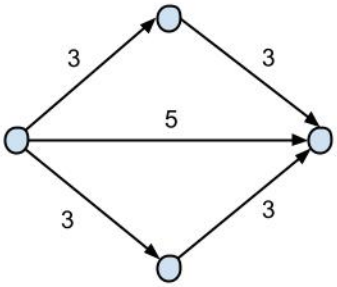
[6–15] No, it is not necessary. Consider the following example. Let G be the graph below and T be the shortest path spanning tree rooted at the leftmost vertex as shown. If we add $k = 2$ to all the edge weights and get graph G'. Then the shortest path spanning tree will change to T'.



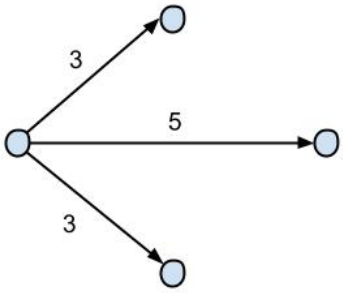
(A) Original graph G



(B) T



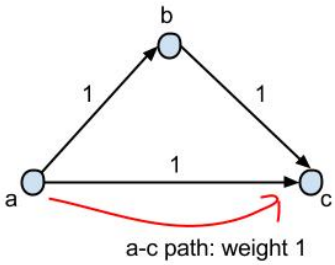
(C) New graph G'



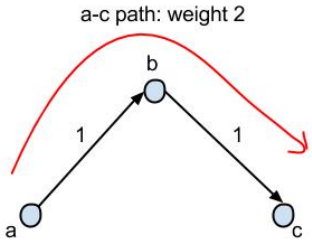
(D) T'

PROBLEM 10

[6–17] Is the path between a pair of vertices in a minimum spanning tree of an undirected graph necessarily the shortest (minimum weight) path? – No, see the example given below.

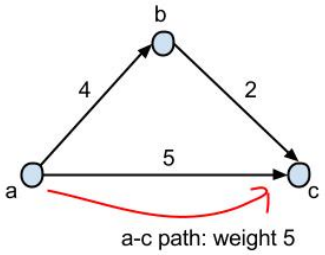


(E) Shortest path in G

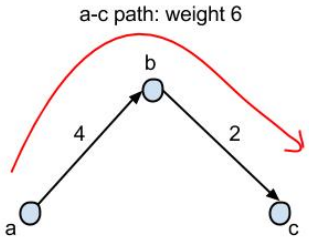


(F) Shortest path in T

Suppose that the minimum spanning tree of the graph is unique. Is the path between a pair of vertices in a minimum spanning tree of an undirected graph necessarily the shortest (minimum weight) path? – No, see the example given below.



(G) Shortest path in G'



(H) Shortest path in T'