COMP 3711 Design and Analysis of Algorithms
2014 Fall Semester
Solutions to Assignment 4

1. The algorithm will be similar to BFS or DFS, except that we only visit vertices that are active. And whenever we visit a vertex, we spread its influence to its neighbors. The following code shows the BFS way of doing this by using a queue to store all the active vertices. You can also use a stack, which will make the algorithm more similar to DFS.

```
Q = empty
Q.enqueue(r)
count = 1
for each vertex v
    d(v) = 0   // d stores the total amount of influence received by v
    color(v) = WHITE
color(r) = BLACK
while (Q is not empty)
     v = Q.dequeue()
     for each u in Adj[v]
         if color(u) = WHITE then
             d(u) = d(u) + w(v, u)
             if d(u) >= t(u) then
                 color(u) = BLACK
                 Q.enqueue(u)
                 count = count + 1
return count
```

   Since the algorithm does no more work than BFS, the running time is $O(V + E)$.

2. e_1 and e_2 must always be in the MST, while e_3 may not. For the counter example on e_3, simply consider a triangle with three edges. The MST only contains e_1 and e_2, but not e_3.

   Below we prove that e_1 and e_2 must both belong to the MST. Suppose to the contrary that $e_1 = (u, v)$ is not in the MST T. We add e_1 to T. This creates a cycle containing e_1. Let e' be some other edge on this cycle. Since e_1 is the smallest-weight edge in the graph, we have $w(e') > w(e_1)$. We now remove e' from T and this breaks the cycle, and turns T back to a spanning tree. Now we see that the new tree has a smaller total weight, which contradicts the assumption that T is the MST, and this completes the proof.

   Finally, we observe that this proof also works on e_2, since the cycle contains at least three edges, and we can still find an e' on this cycle such that $w(e') > w(e_2)$. However, this proof doesn't work on e_3.

3. First we prove the "if" part. Suppose there is a path from u to v and all edges on the path are cheaper than e. If we add e to this path, we get a cycle on which e is the heaviest edge. Thus by the cycle property proved in the tutorial, e cannot be in the MST.

   Next we prove the "only if" part. Suppose there is no path connecting u and v using only edges cheaper than e. We will try to find a cut $(S, V - S)$ such that e is the cheapest edge crossing the cut. Then the cut lemma would imply that e must belong to the MST. We simply put u into S, as well as all vertices reachable from s using only edges cheaper than e. The rest of the vertices are then $V - S$. Clearly, e crosses this cut, and there is no edge cheaper than e that crosses the cut, since if there were, we would have expanded S in the first place. This completes the proof.

To use this theorem to check whether e belongs to the MST, we just delete all edges heavier than e and e itself, and then check whether u and v are still connected, using either BFS or DFS. This takes time $O(V + E) = O(E)$, since we have $E \geq V - 1$ as the graph is connected.

4. Let $d(s, v)$ be the longest distance from s to v. Then the recurrence is

$$d(s, v) = \max_{u,(u,v)\in E} d(s, u) + w(v).$$

Then the algorithm will be very similar to the one in the lecture notes:

```
topologically sort the vertices of G
for each vertex v ∈ V
    v.d ← −∞, v.p ← nil
s.d ← 0
for each vertex u in topological order
    for each vertex v ∈ Adj[u]
        if v.d < u.d + w(v) then
            v.d ← u.d + w(v), v.p ← u
v ← t
while v ≠ s do
    print v
    v ← v.p
print s
```

The running time of the algorithm is $O(V + E)$.