

COMP 3711 Design and Analysis of Algorithms
2015 Fall Semester
Solutions to Assignment 4

1. 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 .

2. 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.

3. 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:

```
1 topologically sort the vertices of  $G$ 
2 for each vertex  $v \in V$ 
3    $v.d \leftarrow -\infty, v.p \leftarrow nil$ 
4  $s.d \leftarrow 0$ 
5 for each vertex  $u$  in topological order
6   for each vertex  $v \in Adj[u]$ 
7     if  $v.d < u.d + w(v)$  then
8        $v.d \leftarrow u.d + w(v), v.p \leftarrow u$ 
9  $v \leftarrow t$ 
10 while  $v \neq s$  do
11   print  $v$ 
```

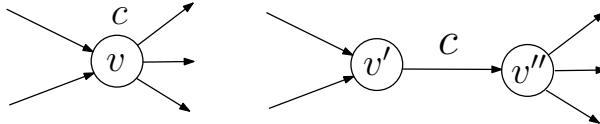
```

12      $v \leftarrow v.p$ 
13 print  $s$ 

```

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

4. For each vertex v with capacity c , we can split it into two vertices v' and v'' with no capacity, but add an edge between from v' to v'' with capacity c . All the incoming edges to v will now go into v' , while all outgoing edges from v will leave from v'' . An example is shown below:



It is obvious that this preserves the maximum flow in the original network. Also, for each vertex in the original network, we introduce one more vertex and one more edge, so the total number of vertices is at most $V + V = O(V)$, and the total number of edges is at most $E + V = O(E)$.

To solve the escape problem, we create a source vertex s and add an edge from s to each of the m starting points. For every two neighboring vertices, we add two antiparallel edges (which then can be removed using the technique in the lecture slides). Finally, we add a target vertex t and add an edge from each vertex on the boundary to t . There is no capacity on the edges, but each vertex has a capacity of 1. If this network's maximum flow has value m , then there is an escape plan; otherwise the answer is no. The running time is $O(E|f^*|) = O(E m) = O(n^2 m)$.