# CS 170 Fall 2006 — Solutions to Midterm 2

November 16, 2006

**Problem 1 [2 × 7 = 14 points]**

1. False. We start with the all-false assignment.

2. True. Suppose, for the sake of contradiction, $X$ is the symbol with the highest frequency and $Y$ is another symbol which is a higher leaf than $X$. Then, exchanging $X$ and $Y$ gives a shorter code which is impossible since Huffman coding gives an optimal code.

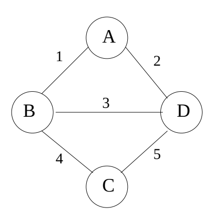
3. True. Replace each edge of weight $w$ by $w$ unit length edges. The number of edges now is at most $10|E|$ and the time taken in this is $O(|E|)$. We now do a BFS starting from $s$ and *continuing only till we reach $t$ or explore the entire component of $s$*. We can take care of the 0-length edges as follows: If $u$ is being processed by the BFS and $(u, v)$ is a 0-length edge, then push $v$ to the *front* rather than the back of the queue. Let $e$ be the number of edges in the component of $s$. Since this is connected, $v \leq e - 1$, where $v$ is the number of vertices in the component. Hence, the time taken is $O(v + e) = O(e) = O(|E|)$.

4. True. After the first `Find`, the depth of $x$ becomes 1. The two unions can increase it at most by 1 each. Hence, at the time of the next `Find`, $x$ is at most 3 levels below the root of its tree, which means the cost of `Find` is at most 3.

5. True. The solution is $O(\log n)$, which is also $O(n)$.

6. True. It works in time $O(|V||E|)$ in general. If $|V| \leq |E|$, this is $O(|E|^2)$. If $|V| > |E|$, note that the length of the longest path can be at most $|E|$ (there are no more edges). Hence, we only need to update all the edges $|E|$ times, instead of $|V| - 1$ times. The time taken is again $O(|E|^2)$.

7. False. The longest edge in each cycle is guarenteed not to be in the tree, but the shortest edge need not be in the tree either. The following graph provides a counterexample: the only MST is made of edges $AB$, $AD$ and $BC$. $BD$, which is the shortest edge in the cycle $BCD$, is not present in the tree. In fact, it is the longest edge in the cycle $ABD$, hence both parts of the given statement cannot hold




simultaneously.

**Problem 2 [6 points]**

**Kruskal's algorithm:** CD, BD. **[1.5 pts]**

**Dijkstra's algorithm from A:** AC, BD. **[1.5 pts]**

**Prim's algorithm:** BD, CD. **[1.5 pts]**

**Bellman-Ford algorithm:** Does not make sense. Bellman-Ford has no specified order in which it updates edges. Hence, any two edges other than $AB$ might be processed. **[1.5 pts]**

**Problem 3 [15 points]**

There are various ways of solving this problem. One solution is to construct a new graph which is identical to the given graph except that weights are on the edges instead of the vertices. We set the weight of the edge $(u, v)$ as $w(v)$, which is the weight of the the vertex $v$ in the original graph. Since the hint suggests using Dijkstra's algorithm, we can assume the weights to be positive and run Dijkstra on the new graph. **[10 pts]**

We claim that the shortest path in the new graph is also the shortest path in the original graph. Let $s, u_1, \ldots, u_k, t$ be any $s - t$ path in the original graph. The weight of the path is the sum of weights of all the vertices on it and is equal to $w(s)+w(u_1)+\ldots+w(u_k)+w(t)$. The weight of the same path in the new graph is $w(s, u_1) + w(u_1, u_2) + \ldots + w(u_{k-1}, u_k) + w(u_k, t)$, which is equal to $w(u_1) + w(u_2) + \ldots + w(u_k) + w(t)$. Thus, the weights of *all* the paths in the old and new graphs only differ by $w(s)$. Hence, the shortest paths in the two graphs will be the same. **3 pts**

The time taken in creating the new graph is $O(|V|+|E|)$. Since, we then run Dijkstra on the new graph, and the number of vertices and edges in the new graph is the same, the total running time is $O((|V|+|E|) \log |V|)$, using a binary heap implementation. **[2 pts]**

**Problem 4 [15 points]**

- The problem with $C[i]$ is that it does not permit us to write a recurrence. Suppose we know $C[1], \ldots, C[i]$ and want to find $C[i + 1]$. However, $C[1], \ldots, C[i]$ do not give us information about any sequence ending at $i$ to which we may possibly add $a[i + 1]$. For example, in the sequences $1, 2, -1, -1, 3$ and $-1, -1, 1, 2, 3$, $C[4] = 3$ for both, but the longest sequence includes $a[5]$ in the second one but not in the first.

- $C[0] = 0$ and $D[0] = 0$.

- $D[i] = \max\{0, a[i], D[i-1] + a[i]\}$, $C[i] = \max\{C[i-1], D[i]\}$

- Output $= C[n]$

- We go through all elements in the array to find an $i$ such that $D[i] = C[n]$. This $a[i]$ is the ending point of a sequence which has sum equal to the optimal. We then start at $a[i]$ and proceed backwards adding $a[i-1, a[i-2], \ldots, a[j]$ till the first $j$ such that $a[i] + a[i-1] + \ldots + a[j] = C[n]$. Then $a[j], \ldots, a[i]$ is the required set.

- We solve $2n+2$ subproblems ($C[0], \ldots, C[n]$ and $D[0], \ldots, D[n]$) and each takes constant time. Hence, the total running time is $O(n)$. Also, the final search through the array to find the set takes $O(n)$ time.