Problem Set 5 Solutions

Calvin Walker

Problem 1:

a)

	s	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**t**
k=1	0	4	5	8	∞	∞	∞	∞	∞	∞
k=2	0	3	5	8	1	9	∞	∞	13	∞
k=3	0	3	5	8	0	9	10	17	3	20
k=4	0	3	5	6	0	9	10	7	2	10
k=5	0	3	5	6	0	7	10	6	2	9
k=6	0	3	5	6	0	7	8	6	2	9
k=7	0	3	5	4	0	7	8	6	2	9
k=8	0	3	5	4	0	5	8	6	2	9
k=9	0	3	5	4	0	5	6	6	2	9
k=10	0	3	5	2	0	5	6	6	2	9

Table 1: Simulated Execution of the Bellman-Ford Algorithm

b) For all verticies $v \in V(G) \setminus \{c, e, f\}$, the shortest distance from s to v is properly computed. This is because c, e and f are part of a negative cycle in G. Looking at the 10th iteration, we see that we can reach c with a cost of 2, and e with a cost of 5. However, e is only a cost of 1 away from c, and both are within 2 edges of s, a clear warning sign of the failure.

Problem 2:

At each timeslot, we can either stay at the current stage, or travel to another. Let $h'(i, j)$ be the max happiness up to time j at stage i. Then we have the following recurrance relation:

$$h'(i, j) = h_{ij} + \max\{h'(i, j-1),\ \max_{i'}\{h'(i',\ j-2)\}\}$$

Algorithm: Initialize a $k \times n$ matrix H' such that $h'_{i1} = h_{i1}\ \forall i \in [k]$, and let the other entires of H' be zero. Then, for each time $j \in [2 \dots n]$, iterate over each $i \in [k]$, and let $H'(i, k) = h_{ij} + \max\{H'(i, j-1),\ \max_{i'}\{H'(i',\ j-2)\}\}$. Since at times $j \leq 2$ there is no valid entry for $H'(i',\ j-2)$, we return 0 if a call is made outside the bounds of H'. Finally, we return the maximum entry in the last column of H', $\max_{i'} H'(i', n)$, to obtain the maximum total enjoyment from attending the music festival.

Runtime: For each time 1 through n, we compute the maximum enjoyment at the k possible stages, each of which must check the k stages at time $j-2$. So the total runtime is $O(nk^2)$.

Proof of Correctness: For each stage, we can partition the festival schedules that finish by time j at stage i based on whether they were traveling in time $j-1$ or not.

1. If they were not traveling at time $j-1$ then they must be at the stage i at time $j-1$. These schedules consist of all schedules finishing at stage i at time $j-1$. The best such schedule has enjoyment $h'(i, j-1)$.

2. If they were traveling at time $j-1$ they can be at any stage $i' \in [k]$ at time $j-2$. These schedules consist of all schedules finishing at time $j-2$. The best such schedule has enjoyment $\max_{i'} h'(i', j-2)$.

Thus, the maximum enjoyment at stage i at time j is equal to $h_{ij} + \max\{h'(i, j-1),\ \max_{i'}\{h'(i',\ j-2)\}\}$

Problem 3:

a) This is tree, since it is connected and acyclic.

b) Algorithm: Let $S = (V, E)$ be the directed graph of company S. Let $T(v)$ be the amount of time it takes to notify all of the employees below employee $v \in V$ in the graph. For each employee v, they notify their employees $u_i \in \{u \mid (v, u) \in E\}$ in descending order of $T(u_i)$. We say employee u_i is the i'th to be notified by v (starting at 1), and have the recurrance relation:

$$T(v) = \max_{u_i}\ T(u_i) + i$$

If there are no employees below $v \in S$, we say $T(v) = 0$. So we can obtain the correct ordering for all employees by calling $T(CEO)$, and storing the intermediate results for each employee.

Runtime: For each of the n employees, we sort their employees by $T(.)$, so the runtime is $O(n^2 \log n)$

Explanation: It is intuitive that we should notify the employees who will take the most time to have all the employees below them notified first, since the time it takes for the whole company to be wearing pink hats will be upperbound by the most time intensive parts of the tree. A more rigourous way to look at this is minimizing $T(v)$. Consider notifying an employee u directly below v. If we know employee u takes $T(u)$ days to notify their employees, and we notify them first, then $T(v) = T(u) + 1$, if we notify some other employee u' first then $T(v) = \max\{T(u') + 1, T(u) + 2\}$. If $T(u) > T(u')$ then we can clearly minimize $T(v)$ by notifying u first.

Problem 4:

Algorithm: We can simplify this problem by considering the following subproblem. On the interval $[l, r]$, which pokemon in $p_l \dots p_r$ could win the battle? One way of solving this subproblem would be to consider all the pokemon who can win on $[l + 1, r]$, and seeing if p_l can beat one of these pokemon, or if some of these pokemon can beat p_l. Let $W(l, r)$ be the set of possible winners over $[l, r]$. We say $p_i > p_j$ if p_i beats p_j. Then, we have the following recurrance relation:

$$W(l, r) = \{p_i \in W(l + 1, r) \mid p_i > p_l\} \cup \{p_j \in W(l, r - 1) \mid p_j > p_r\}$$

Where p_r and p_l are also added to $W(l, r)$ if they can beat a pokemon in $W(l, r - 1)$ or $W(l + 1, r)$ respectivley. There is also the base case that if $r - l = 1$ we return the winning pokemon betwen p_l and p_r. We simply return the result of $W(1, n)$ to obtain a list of all the possible winners of the battle.

Explanation: One way to think about the problem is to consider the circumstances in which pokemon p_i is a possible winner. There must be a possible winner within the intervals to the left and right of him on the line who he can beat. Since we start the recursion from each end of the line, the algorithm effectivley checks both directions for each pokemon. For instance, p_i is only added to the potential winners on $W(i, n)$ if he can beat a pokemon from $W(i + 1, n)$, and only added to $W(1, i)$ if he can beat a pokemon from $W(1, i - 1)$. Similarly, the possible winners from these smaller intervals are only kept if they can beat him. So by the last recursive call, only potential winners on the interval $[1, n]$ are remaining.

Runtime: Since the result of a single battle can be found in constant time, for some pokemon p_i, we can compare them to the pokemon in $W(i + 1, n)$ and $W(1, i - 1)$ in linear time. We do this for each of the n pokemon, so the runtime is $O(n^2)$.