CSCI 570 - Fall 2021 - HW 3

Due September 9, 2021

1 You have N ropes each with length L_1, L_2, ..., L_N, and we want to connect the ropes into one rope. Each time, we can connect 2 ropes, and the cost is the sum of the lengths of the 2 ropes. Develop an algorithm such that we minimize the cost of connecting all the ropes.

Use a min-heap to sort the ropes and pop the 2 lowest cost ropes each time and connect them. Then add the summed length back into the heap. Continue until you are left with only 1 string in the heap.

2 You have a bottle that can hold L liters of liquid. There are N different types of liquid with amount L_1, L_2, ..., L_N and with value V_1, V_2, ..., V_N. Assume that mixing liquids doesn't change their values. Find an algorithm to store the most value of liquid in your bottle.

Sort the liquids by V/L. Put the max of that liquid into your bottle and continue for the remaining space in your bottle.

3 Suppose you were to drive from USC to Santa Monica along I-10. Your gas tank, when full, holds enough gas to go p miles, and you have a map that contains the information on the distances between gas stations along the route. Let $d_1 < d_2 \ldots < d_n$ be the locations of all the gas stations along the route where d_i is the distance from USC to the gas station. We assume that the distance between neighboring gas stations is at most p miles. Your goal is to make as few gas stops as possible along the way. Give the most efficient algorithm to determine at which gas stations you should stop and prove that your strategy yields an optimal solution. Give the time complexity of your algorithm as a function of n.

The greedy strategy we adopt is to go as far as possible before stopping for gas. That is when you are at the i^{th} gas station, if you have enough gas to go to the $i + 1^{th}$ gas station, then skip the i^{th} gas station. Otherwise stop at the i^{th} station and fill up the tank.
Let $\{g_1, g_2, \ldots, g_m\}$ be the set of gas stations at which our algorithm made us refuel. We next prove that our choice is optimal.
Let $\{h_1, h_2, \ldots, h_k\}$ be an optimal solution.
Since it is not possible to get to the $g_1 + 1^{th}$ gas station without stopping, any solution should stop at either g_1 or a gas station before g_1, thus $h_1 \leq g_1$. If $h_1 < g_1$, then we can swap its first stop with g_1 without changing

the number of stops. The new solution we obtain g_1, h_2 , ... , h_k is legal since when leaving g_1 we have at least as much fuel now as we had before swapping. Hence $\{g_1, h_2, \ldots, h_k\}$ is an optimal solution.
Assume that $\{g_1, g_2, \ldots, g_{c-1}, h_c, \ldots, h_k\}$ is an optimal solution (induction hypothesis). From the greedy strategy taken by our algorithm, $h_c \leq g_c$. If $h_c < g_c$, then by swapping g_c and h_c,we get g_1,g_2,...,g_{c-1},g_c,h_{c+1},...,h_k which is indeed a legal solution. The legality follows from the same reasoning as above. That is, when leaving g_c we now have at least as much fuel as we did before swapping and can hence reach the destination. Thus $\{g_1,g_2,\ldots,g_c,h_{c+1},\ldots,h_k\}$ is an optimal solution.
By induction, it thus follows that $\{g_1, g_2 , \ldots , g_m\}$ is an optimal solution. The running time is O(n) since we at most make one computation/decision at each gas station.

4 (a) Consider the problem of making change for n cents using the fewest number of coins. Describe a greedy algorithm to make change consisting of quarters(25 cents), dimes(10 cents), nickels(5 cents) and pennies(1 cents). Prove that your algorithm yields an optimal solution. (Hints: consider how many pennies, nickels, dimes and dime plus nickels are taken by an optimal solution at most.)

(b) For the previous problem, give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Assume that each coin's value is an integer. Your set should include a penny so that there is a solution for every value of n.

(a) Denote the coins values as $c_1 = 1$, $c_2 = 5$, $c_3 = 10$, $c_4 = 25$.
1) if $n = 0$, do nothing but return.
2) Otherwise, find the largest coin c_k, $1 \leq k \leq 4$, such that $c_k \leq n$. Add the coin into the solution coin set S.
3) Subtract x from n, and repeat the steps 1) and 2) for $n - c_k$.

For the proof of optimal, we first prove the following claim: Any optimal solution must take the largest c_k, such that $c_k \leq n$. Here we have the following observations for an optimal solution:
1. Must have at most 2 dimes; otherwise we can replace 3 dimes with quarter and nickel.
2. If 2 dimes, no nickels; otherwise we can replace 2 dimes and 1 nickel with a quarter.
3. At most 1 nickel; otherwise we can replace 2 nickels with a dime.
4. At most 4 pennies; otherwise can replace 5 pennies with a nickel.

Correspondingly, an optimal solution must have
• Total value of pennies: ≤ 4 cents.
• Total value of pennies and nickels: $\leq 4 + 5 = 9$ cents.
• Total value of pennies, nickels and dimes: $\leq 2\times10+4 = 24$ cents.
Therefore,

• If $1 \leq n < 5$, the optimal solution must take a penny.
• If $5 \leq n < 10$, the optimal solution must take a nickel; otherwise, the total value of pennies exceeds 4 cents.
• If $10 \leq n < 25$, the optimal solution must take a dime; otherwise, the total value of pennies and nickels exceeds 9 cents.
• If $n \geq 25$, the optimal solution must take a quarter; otherwise, the total value of pennies, nickels and dimes exceeds 24 cents.

Compared with the greedy algorithm and the optimal algorithm, since both algorithms take the largest value coin c_k from n cents, then the problem reduces to the coin changing of n - c_k cents, which, by induction, is optimally solved by the greedy algorithm.

(b) Coin combinations = {1, 15, 20} cents coins.Consider this example n = 30 cents. According to the greedy algorithm, we need 11 coins: $30 = 1\times20+10\times1$; but the optimal solution is 2 coins $30 = 2\times15$.

5 Solve Kleinberg and Tardos, Chapter 3, Exercise 3.

We run the algorithm described in Section 3.6 (bottom of page 106) with the following modification. If the input G is a DAG, then at each iteration of the algorithm, there exists a vertex with no incoming edges and the algorithm finds a topological ordering thereby establishing that the input G is indeed a DAG. Hence the only way for the algorithm to be stuck is if at a certain iteration all the vertices have at least one incoming edge. In this case, pick an arbitrary vertex v_1 and follow one of its incoming edges to a predecessor vertex (say v_2). Now follow one of v_2's incoming edges to go to one of its predecessors (say v_3) and so on. We can repeat this step as often as we want since every vertex has an incoming edge. Since the graph has at most n vertices, after at most n steps we have to revisit a vertex (say $v_k = v_i$ for some $k > i$). Clearly v_i,v_{i+1},...,v_k forms a cycle.

6 Solve Kleinberg and Tardos, Chapter 4, Exercise 4.

Let the two input sequences be $S = s_1, s_2, ..., s_n$ and S' = (r_1,r_2,...,r_m). We propose the following greedy algorithm. Find the first event in S that is the same as r_1. If you cannot find such an event, output "no" and terminate. Say s_{i1} is the first event in S that is the same as r1. Remove the first i1 events from S, that is S = (s_{i1+1}, s_{i1+2}, ... , s_n). In the second iteration, find the first event in S that is the same as r2. If you cannot find such an event, output "no" and terminate. Say s_{i2} is the first event in S that is the same as s2. Set S = (s_{i2+1},s_{i2+2},...,s_n) and so on. If the algorithm runs successfully for m, iterations then output "yes".
Clearly, if the algorithm runs successfully for m iterations, then S' is indeed a substring of S (since (s_{i1},s_{i2},...,s_{im}) = S') and thus our algorithm is correct.
The harder direction is to prove that if S' is a substring of S, then our

algorithm indeed outputs "yes". We prove the following slightly stronger claim. Claim: If (s_{k1},s_{k2},...,s_{km}) = S', then $s_{ij} = s_{kj}$ and $i_j \leq k_j$ for all j $\in$ {1,2,...,m}
We prove the claim by induction on j. The case j = 1 follows from the first step of our algorithm. Assume (induction hypothesis) that the claim holds for j = c − 1. The induction hypothesis implies that the algorithm ran successfully for at least c − 1 steps. Since $s_{kc} = r_{kc}$ and $k_c > k_{c\text{-}1} \geq i_{kc\text{-}1}$, the c^{th} step of the algorithm ran successfully and found s_{ic} such that $s_{ic} = s_{kc}$. Further since ic is the smallest index greater than ic-1 such that $s_{ic} = r_{ic}$, it is true that $k_c \geq i_c$ and the claim follows.
The running time of the algorithm is O(n+m) as we examine each element in the sequences at most once.

7 (Not Graded) There are N tasks that need to be completed by 2 computers A and B. Each task i has 2 parts that takes time a_i(first part) and b_i (second part) to be completed. The first part must be completed before starting the second part. Computer A does the first part of all the tasks while computer B does the second part of all the tasks. Computer A can only do one task at a time, while computer B can do any amount of tasks at the same time. Find an O(nlogn) algorithm that minimizes the time to complete all the tasks, and give a proof of why the solution is optimal.

Sort the tasks in decreasing order of bi. Perform the tasks in that order. Basically computer A does the first parts in that order, and computer B starts every second part after computer A finishes the first part.

WLOG, let's rearrange the tasks in order such that $b_1 \geq b_2 \geq \ldots \geq b_N$. Let us also perform our tasks in this order, and let us call this ordering *greedy*. Then the time T it takes to complete the tasks in this ordering is

$$T(greedy) = max(a_1 + b_1, a_1 + a_2 + b_2, \ldots, \sum_{i=1}^{N} a_i + b_N)$$

So suppose the greedy is not the optimal ordering, so there exists an ordering called optimal such that $T(optimal) < T(greedy)$. So we choose k such that $T(greedy) = \sum_{i=1}^{k} a_i + b_k$. Now the optimal ordering has some random ordering which we don't know, but let us index the ordering as ϕ. For example, suppose the ordering was 3, 6, 2, 1, ..., 8. Then $\phi(1) = 3, \phi(2) = 6, \ldots, \phi(N) = 8$. Let h be the smallest number such that $\{1, 2, \ldots, k\} \subseteq \{\phi(1), \phi(2), \ldots, \phi(h)\}$. We have that $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{h} a_{\phi(i)}$. Also, $b_k \leq b_{\phi(h)}$, since $\phi(h) \leq k$. Then we have the following inequality

$$T(greedy) = \sum_{i=1}^{k} a_i + b_k \leq \sum_{i=1}^{h} a_i + b_{\phi(h)} \leq T(optimal)$$

This is clearly a contradiction.