CS 570 - Fall 2015 - HW 3

1. Read Section 3.6 and Section 4.1-4.2 in the textbook.

2. The police department in a city has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up soon, and there is just enough time to run a linear-time algorithm.

   (a) Formulate this as a graph problem and design a linear-time algorithm. Explain why it can be solved in linear time.

   (b) Suppose it now turns out that the mayor's original claim is false. She next makes the following claim to supporters gathered in the Town Hall: "If you start driving from the Town Hall (located at an intersection), navigating one-way streets, then no matter where you reach, there is always a way to drive legally back to the Town Hall." Formulate this claim as a graph problem, and show how it can also be verified in linear time.

   Solution:

   (a) The mayor is merely contending that the graph of the city is a strongly-connected graph, denoted as G. Form a graph of the city (intersections become nodes, one-way streets become directed edges). Suppose there are n nodes and m edges. The algorithm is:

   1) Choose an arbitrary node s in G, and run BFS from s. If all the nodes are reached the BFS tree rooted at s, then go to step 2), otherwise, the mayor's statement must be wrong. This operation takes time $\mathcal{O}(m+n)$.
   2) Reverse the direction of all the edges to get the graph G^{inv}, this step takes time $\mathcal{O}(m+n)$.
   3) Do BFS in G^{inv} starting from s. If all the nodes are reached, then the mayor's statement is correct; otherwise, the mayor's statement is wrong. This step takes time $\mathcal{O}(m+n)$.

Explanation: BFS on G tests if s can reach all other nodes; BFS on G^{inv} tests if all other nodes reach s. If these two conditions are not satisfied, the mayor's statement is wrong obviously; if these two conditions are satisfied, any node v can reach any node u by going through s.

(b) Now the mayor is contending that the intersections which are reachable from the city form a strongly-connected component. Run the first step of the previous algorithm while setting the Town Hall as s (test to see which nodes are reachable from Town Hall). Remove any nodes which are unreachable from the town hall, and this is the component which the mayor is claiming is strongly connected. Run the previous algorithm on this component to verify it is strongly connected.

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

Solution:

1) Run the topology ordering algorithm for graph G (shown at the bottom of page 102) with a little bit modification. At each iteration of the algorithm, we try to delete a node with no incoming edge from graph G. If the algorithm finds a topological ordering for the whole graph G (no node left), then return G as a DAG. The only way for the algorithm to be stuck is if at a certain iteration all the remaining vertices have at least one incoming edge. Then we confirm that G must contain a circle. Denote the remaining graph as G'. The complexity is $\mathcal{O}(n+m)$.
2) If the algorithm dose not return a topological ordering in step 1) but provides a subgraph G'. Reverse the edges in G' to get G'^{inv}. This step is to guarantee that all the nodes in G'^{inv} has at least one outgoing edge. The reversing procedure takes $\mathcal{O}(m)$ time (Note that there is no isolated nodes in G').
3) Start traversing from an arbitrary vertex v in G'^{inv} along the outgoing edges until we revisit certain node, say node u. This procedure takes no more than n steps. During this procedure, we need to record the predecessor of each visited node. After the traversing procedure stops, we just trace back from u according to the predecessor records and return the loop we found. The traversing and tracing back procedure takes $\mathcal{O}(n)$ time.

In sum, the complexity is $\mathcal{O}(n + m)$.

4. 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 $d_1 < d_2 < ... < d_n$ be the locations of all the gas stations along the route where d_i is the distance from USC to the gas station (assume the distance between USC and Santa Monica is d_n). 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.

Solution:

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 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 $\{g_1, h_2, \ldots, h_k\}$ 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 $\{g_1, g_2, \ldots, g_{c-1}, g_c, h_{c+1}, \ldots, h_k\}$ 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 $\{g_1, g_2, \ldots, g_k\}$ is an optimal solution. Furthermore, m must be equal to k, so the greedy algorithm is an optimal solution.

The running time is $\mathcal{O}(n)$ since we at most make one computation/decision at each gas station.

5. Solve Kleinberg and Tardos, Chapter 4, Exercise 3.

Solution:

Assume the greedy algorithm currently in use fits boxes $b_1, b_2, \cdots, b_j$ into the first k trucks. We prove that no other algorithm can fit more boxes into k trucks, i.e., if an algorithm fits boxes $b_1, b_2, \cdots, b_i$ into the first k trucks, then $i \leq j$.

We prove this claim by induction on k:

- For $k = 1$: the greedy fits as many boxes into one truck as possible, it is clear that no other algorithm can fix more boxes into the truck, thus, $i \leq j$.
- Assume it holds for $k - 1$, i.e., if the greedy algorithm fits boxes $b_1, b_2, \cdots, b_j$ into the first $k - 1$ trucks, and the other algorithm fits boxes $b_1, b_2, \cdots, b_i$ into the first $k - 1$ trucks, then $i \leq j$.
- For k: the greedy algorithm fits j' boxes into the first $k - 1$ trucks, the other algorithm fits i' boxes into the first $k - 1$ trucks, and $i' \leq j'$; now, for the kth truck, the other algorithm packs in boxes $b_{i'+1}, \cdots, b_i$; since $i' \leq j'$, the greedy algorithm is able at least to fit all the boxes $b_{j'+1}, \cdots, b_i$ into the kth truck, and it may be able to fit more.

6. Suppose you are given two sets A and B, each containing n positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the ith element of set A, and let b_i be the ith element of set B. You then receive a payoff of $\prod_{i=1}^{n} a_i^{b_i}$. Give an algorithm that will maximize your payoff. Prove that your algorithm maximizes the payoff, and state its running time.

   Solution:

   Sort both A and B in the same order (either both ascending or both descending). This takes $\mathcal{O}(nlogn)$. What's important is that the largest element of A is matched with the largest of B, the second-largest of each are matched, and so on (greedy solution).

   For purposes of the proof, fix the order of A to be in ascending order, i.e., $a_1 \leq a_2 \leq \cdots \leq a_n$; we will consider all solutions in terms of how their B arranged relative to this A. My solution has B sorted ascendingly.

   For any arbitrary solution (could be an optimal solution): $\{b_i\}$, where b_i is the element of B matched with a_i in this solution. Suppose there exist an inversion $b_i > b_j$, for $i > j$. Now we prove that undoing this inversion will make the result no less than the original one, and has the effect of multiplying the original solution's quantity by this value:

$$\frac{a_i^{b_j} \times a_j^{b_i}}{a_i^{b_i} \times a_j^{b_j}} = \frac{a_j^{b_i - b_j}}{a_i^{b_i - bj}} = \left(\frac{a_j}{a_i}\right)^{b_i - b_j}$$

Because $\frac{a_j}{a_i} \geq 1$ and $b_i - b_j > 0$, the result above should be no less than 1, i.e., every inversion relative to this solution can be removed without negatively affecting the quantity, which indicates the optimality of the greedy solution.

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

Solution:

Let the two input sequences be $S = (s_1, s_2, \ldots, s_n)$ and $S' = (r_1, r_2, \ldots, r_m)$.

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_{i_1} is the first event in S that is the same as r_1. Remove the first i_1 events from S, that is $S = (s_{i_1+1}, s_{i_1+2}, \ldots, s_n)$. In the second iteration, find the first event in S that is the same as r_2. If you cannot find such an event, output "no" and terminate. Say s_{i_2} is the first event in S that is the same as r_2. Set $S = (s_{i_2+1}, s_{i_2+2}, \ldots, s_n)$ 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 $(s_{i_1}, s_{i_2}, \ldots, s_{i_m}) = S'$) and thus and our algorithm is correct.

The harder part is to prove that if S' is a substring of S, then our algorithm indeed outputs "yes". Then we first need to prove the following claim.

Claim: If $(s_{k_1}, s_{k_2}, \ldots, s_{k_m}) = S'$, where $k_1 < k_2 < \cdots < k_m$, then $s_{i_j} = s_{k_j}$ and $i_j \leq k_j$ for all $j \in \{1, 2, \ldots, m\}$

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 $k_{c-1} \geq i_{c-1}$, then $\{s_{k_{c-1}+1}, \cdots, s_n\} \subseteq \{s_{i_{c-1}+1}, \cdots, s_n\}$. Moreover, $s_{k_c} = r_c \in \{s_{k_{c-1}+1}, \cdots, s_n\}$, we have $r_c \in \{s_{i_{c-1}+1}, \cdots, s_n\}$. Further since i_c is the smallest index greater than i_{c-1} such that $s_{i_c} = r_c$, it is true that $k_c \geq i_c$ and the claim follows.

Then for any subsequence S' of S, the algorithm can return "yes".

The running time of the algorithm is $\mathcal{O}(n)$ as we examine each element in the sequence S at most once (The answer $\mathcal{O}(n + m)$ is also fine).

8. (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 and 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.

Solution:

(a) Algorithm:

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) Substract $c_k$ from $n$, and repeat the steps 1) and 2) for $n - c_k$.
```

For the proof of optimality, 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:

- Must have at most 2 dimes; otherwise we can replace 3 dimes with quarter and nickel.
- If 2 dimes, no nickels; otherwise we can replace 2 dimes and 1 nickel with a quarter.
- At most 1 nickel; otherwise we can replace 2 nickels with a dime.
- 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: $\leq 2 \times 10 + 4 = 24$ 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.

Comparing 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 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: $30 = 1 \times 20 + 10 \times 1$; but the optimal solution is 2 coins $30 = 2 \times 15$.