CSCI 570 - Fall 2016 - HW 6

due October 14, 2016

1. Reading Assignment: Kleinberg and Tardos, Chapter 6.1-6.4.

2. Problem 5 from Chapter 6.

   Let $Y_{i,k}$ denote the substring $y_i y_{i+1} \dots y_k$. Let $Opt(k)$ denote the quality of an optimal segmentation of the substring $Y_{1,k}$. An optimal segmentation of this substring $Y_{1,k}$ will have quality equalling the quality last word (say $y_i \dots y_k$) in the segmentation plus the quality of an optimal solution to the substring $Y_{1,i}$. Otherwise we could use an optimal solution to $Y_{1,i}$ to improve $Opt(k)$ which would lead to a contradiction.

   $$Opt(k) = \max_{0<i<k} Opt(i) + quality(Y_{i+1,k})$$

   We can begin solving the above recurrence with the initial condition that $Opt(0) = 0$ and then go on to compute $Opt(k)$ for $k = 1, 2, \dots, n$ keeping track of where the segmentation is done in each case. The segmentation corresponding to $Opt(n)$ is the solution and can be computed in $\Theta(n^2)$ time.

3. Problem 6 from Chapter 6.

   Let $W = \{w_1, w_2, \dots, w_n\}$ be the set of ordered words which we wish to print. In the optimal solution, if the first line contains k words, then the rest of the lines constitute an optimal solution for the sub problem with the set $\{w_{k+1}, \dots, w_n\}$. Otherwise, by replacing with an optimal solution for the rest of the lines, we would get a solution that contradicts the optimality of the solution for the set $\{w_1, w_2, \dots, w_n\}$.

   Let $Opt(i)$ denote the sum of squares of slacks for the optimal solution with the words $\{w_i, \dots, w_n\}$. Say we can put at most the first p words from w_i to w_n in a line, that is, $\sum_{t=i}^{p+i-1} c_t + p - 1 \le L$ and $\sum_{t=1}^{p+i} w_t + p > L$. Suppose the first k words are put in the first line, then the number of extra space characters is

$$s(i,k) := L - k + 1 - \sum_{t=i}^{i+k-1} c_t$$

So we have the recurrence

$$Opt(i) = \begin{cases} 0 & \text{if } p \geq n-i+1 \\ \min_{1\leq k\leq p}\{(s(i,k))^2 + Opt(i+k)\} & \text{if } p < n-i+1 \end{cases}$$

Trace back the value of k for which $Opt(i)$ is minimized to get the number of words to be printed on each line. We need to compute $Opt(i)$ for n different values of i. At each step p may be asymptotically as big as L. Thus the total running time is $O(nL)$.

4. Problem 7 from Chapter 6.

   For $j \in \{1, 2, \ldots, n\}$, let $OPT(j)$ denote the maximum possible return an investor can make by selling the stock on day j.

   If the investor held the stock on day $j-1$, then $OPT(j) = OPT(j-1) + (p(j) - p(j-1))$. Otherwise (the share hasn't been bought yet as on day $j-1$), the investor can make at most $p(j) - p(j) = 0$ money by selling it on day j (ie, by buying and selling on day j). Hence we have the recurrence $OPT(j) = max(OPT(j-1) + p(j) - p(j-1), 0)$.

   Compute $OPT(j)$ starting from the initial condition $OPT(1) = 0$ and the optimal day to sell is $argmax_j OPT(j)$. Once we have found the best day to sell, the best day to buy is the day on which the share price is the smallest among all the days occurring on/before the best sell day.

5. Problem 12 from Chapter 6.

   Let $OPT(j)$ denote the minimum cost of a solution for the first j servers (Note that a copy of the file should be in server j for every such solution).

   If k (with $k < j$) is the highest index server that contains a copy of the file, then the access cost for the servers $(k+1, \ldots, j)$ is $(j-k-1) + (j-k-2) + \ldots + 2 + 1 + 0 = \binom{j-k}{2}$. For placing a copy at server j, we pay a cost of c_j. Thus we have the following recurrence

$$OPT(j) = c_j + \min_{0\leq k<j}(OPT(k) + \binom{j-k}{2})$$

   with initial conditions $OPT(0) = 0$.

   Compute $OPT(j)$ in increasing order of j using the recurrences and at each step record the highest index server where the last but one copy was placed (ie, k that minimized the second term in the recurrence). Then $OPT(n)$ gives the minimum cost for solving the problem and by tracking back the last but one server copy at each step we can find the optimal configuration.

At each step, it takes $\mathcal{O}(j)$ to compute $OPT(j)$. Hence the total running time is $\mathcal{O}(n^2)$.

6. Problem 16 from Chapter 6.

   For each subtree T' of T, we define $OPT(T')$ to be the number of rounds it takes for everyone in T' to be notified, once the root has the message. Suppose T' has child sub-trees $T_1^{(T')}, T_2^{(T')}, \ldots, T_{d_{T'}}^{(T')}$, and we label them so that $OPT(T_1^{(T')}) \geq OPT(T_2^{(T')}) \geq \cdots \geq OPT(T_{d_{T'}}^{(T')})$. For tree T', here the root node should talk to its child sub-tree in decreasing order of the time it takes for their sub-trees (recursively) to be notified. This greedy strategy must be optimal for T' (You can use the swapping method you learned in class for greedy algorithm to show its optimality, which is left to you as an exercise). Then the recurrence can be expressed as

$$OPT(T') = \max_{1 \leq j \leq d'_T} \left\{ j + OPT(T_j^{(T)}) \right\}$$

   Base case: if T is simply a leaf node, then $OPT(T') = 0$. The full algorithm builds up the values $OPT(T')$ using the recurrence.

7. Solution:

   Based on the assumption that you have to make at least one cut, an observation can be made: for a rope with length i, if the optimal cut includes a cut at length j, $1 \leq j \leq i-1$, then at least one of the following 4 cases must happen for the left part (with length j) and right part (with length $i-j$) after the cutting:

   - no cut on the left part, optimal cut on the right part;
   - optimal cut on the left part, no cut on the right part;
   - optimal cut on the left part, optimal cut on the right part;
   - no cut on the left part, no cut on the right part;

   Define $OPT(i)$ as the maximum product of the lengths of smaller ropes after cutting the rope of length i. Based on above observation, we have the following recursive relation for length $i \geq 2$:

$$OPT(i) = \max_{1 \leq j \leq \lfloor \frac{i}{2} \rfloor} \{\max\{j \times OPT(i-j), OPT(j) \times (i-j), OPT(j) \times OPT(i-j), j \times (i-j)\}\}$$

   where j only need to takes values up toi 2because of the symmetry of the rope; if $j = 1$, there is no possible cut on the part with length 1, we simply define $OPT(1) = 1$.

Base case: $OPT(2) = 1$, because it has to be cut at least once and this is the only one way to cut.

8. Solution:

First index the bills from the leftmost one to the rightmost one as $1, \ldots, n$. Their corresponding values are $a_1, \ldots, a_n$. Define $OPT(i, j)$ as the maximum amount of money that Alice can get from the remaining bill sequence $a_i, , a_j$ if she gets the first turn to pick a bill, where $1 \leq i \leq n-1; 2 \leq j \leq n; i \leq j$. Since n is even, whenever it is Alices turn to pick the bill, the remaining bill sequence must have even number of bills, i.e., $j - i + 1$ is positive and even. Thus, here is the recursive relation:for $j \geq i + 3$, and $j - i + 1$ is even

$$OPT(i, j) = \max\{Left(i, j), Right(i, j)\}$$

where

$$Left(i, j) = \begin{cases} OPT(i+2, j) + a_i & \text{if } a_{i+1} \geq a_j \\ OPT(i+1, j-1) + a_i, & \text{otherwise} \end{cases} \tag{1}$$

$$Right(i, j) = \begin{cases} OPT(i+1, j-1) + a_j & \text{if } a_i \geq a_{j-1} \\ OPT(i, j-2) + a_j & \text{otherwise} \end{cases} \tag{2}$$

Here $Left(i, j)$ represents the maximum amount of money that Alice can get if she picks the leftmost bill of $a_i, \ldots, a_j$; $Right(i, j)$ represents the maximum amount of money that Alice can get if she picks the rightmost bill of $a_i, \ldots, a_j$.
Base case: $OPT(i, i + 1) = max a_i, a_{i+1}$, for $i = 1, \ldots, n1$.