CSCI 570 - Summer 2015 - HW5 Solutions

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

2. (*Chapter 5, Q 5*) Let $Y_{i,k}$ denote the substring $y_i, y_{i+1}, \cdots, y_k$. Denote the optimal segmentation for $Y_{1,k}$ as $X_1, X_2, \cdots, X_m$. A key observation to make in this problem is that $X_1, X_2, \cdots, X_{m-1}$ is an optimal segmentation for the prefix of $Y_{1,k}$ excluding X_m (because otherwise we could substitute the solution for the prefix by an optimal one and get a better solution).

   Given this observation, let $Opt(k)$ represent the quality of an optimal segmentation of the substring $Y_{1,k}$. We have the following recurrence:

$$OPT(k) = \max_{1 \le i \le k} \{OPT(i-1) + Quality(Y_{i,k})\}. \quad (1)$$

   We can begin solving the above recurrence with the initial condition that $OPT(0) = 0$ and then go on to compute $OPT(k)$ for $k = 1, 2, \cdots, n$, i.e.,

---

$OPT(0) = 0$;
for $k = 1, \cdots, n$ **do**
  $OPT(k) = \max_{1 \le i \le k} \{OPT(i-1) + Quality(Y_{i,k})\}$;
  Record the index i which achieves the maximum, denoted as $i^*(k)$;
end for
Return $OPT(n)$;
Track back through the array OPT by checking from $i^*(n)$ to $i^*(1)$ to recover the optimal segmentation.

   It takes $O(n)$ time to compute each $OPT(k)$;there are $O(n)$ iterations; it takes $O(n)$ time to trace back to recover the optimal segmentation. So the overall complexity is $O(n^2)$.

3. (*Chapter 6, Q 6*) Let $W = \{w_1, w_2, \cdots, w_n\}$ be the set of ordered words which we wish to print. In the optimal solution, if the last line contains p words, then the previous lines constitute an optimal solution for the sub problem with the set $\{w_1, \cdots, w_{n-p}\}$. Otherwise, by replacing with an optimal solution for the previous lines, we would get a better solution.

   For $i \le j$, let $S(i, j)$ represent the slack of a line containing the words $w_i, \cdots, w_j$. To facilitate later derivation, we set $S(i, j) = +\infty$ if these

words exceed the length of a whole line L. Therefore, we have

$$S(i,j) = \begin{cases} L - \sum_{m=i}^{j-1}(c_m+1) - c_j, & \texttt{if } L \geq \sum_{m=i}^{j-1}(c_m+1) - c_j \\ +\infty, & \texttt{otherwise} \end{cases}, \quad (2)$$

where the summation $\sum_{m=i}^{j-1} c_m = 0$, if $i > j-1$. In order to reduce the complexity, $S(i,j)$ can be further computed based on the value of $S(i,j-1)$, i.e.,

$$S(i,i) = \begin{cases} L - c_i, L \geq c_i \\ +\infty, \ otherwise \end{cases}, \quad (3)$$

$$S(i,j) = \begin{cases} S(i,j-1) - c_j - 1, & \text{if } S(i,j-1) < +\infty \text{ and } S(i,j-1) \geq c_j + 1 \\ +\infty, & otherwise, \end{cases} \quad (4)$$

where $j \geq i+1$.

Define $OPT(k)$ as the sum of squares of slacks for the optimal solution with the words $w_1, \cdots, w_k$. As noted above, the optimal solution must have the following

$$OPT(k) = \min_{1 \leq m \leq k} \left\{ (S(m,k))^2 + OPT(m-1) \right\}, \quad (5)$$

and the base case is: $OPT(0) = 0$; Then the algorithm is as follows:

---

for $i = 1, \cdots, n$ **do**
  Compute $S(i,i)$ according to (3)
  if $i < n$ **then**
    for $j = i+1, \cdots, n$ **do**
      Compute $S(i,j)$ according to (4);
    end for
  end if
end for
$OPT(0) = 0$;
for $k = 1, \cdots, n$ **do**
  $OPT(k) = \min_{1 \leq m \leq k} \left\{ (S(m,k))^2 + OPT(m-1) \right\}$;
  Record the index m which achieves the minimum, denoted as $m^*(k)$;
end for
Return $OPT(n)$;
Track back through the array OPT by checking from $m^*(n)$ to $m^*(1)$ to recover the optimal solution.

It takes $O(n)$ time to compute all $S(i,j)$ with fixed i by considering values of j in increasing order; thus, we can compute all $S(i,j)$ in $O(n^2)$ time. Each iteration of computing $OPT(k)$ takes $O(n)$ time, and there are $O(n)$ iterations. The tracking back operation takes $O(n)$ time. Therefore, the overall time complexity is $O(n^2)$.

4. (*Chapter 6, Q 10*)

a) Consider the following example: there are totally 4 minutes, the numbers of steps that can be done respectively on the two machines in the 4 minutes are listed as follows (in time order):
   - Machine A: 2, 1, 1, 200
   - Machine B: 1, 1, 20, 100

   The given algorithm will choose A then move, then stay on B for the final two steps. The optimal solution will stay on A for the four steps.

b) An observation is that, in the optimal solution for the time interval from minute 1 to minute i, you should not move in minute i, because otherwise, you can keep staying on the machine where you are and get a better solution ($a_i > 0$ and $b_i > 0$). For the time interval from minute 1 to minute i, consider that if you are on machine A in minute i, you either (i) stay on machine A in minute $i-1$ or (ii) are in the process of moving from machine B to A in minute $i-1$.

   Now let $OPT_A(i)$ represent the maximum value of a plan in miniute 1 through i that ends on machine A, and define $OPT_B(i)$ analogously for B. If case (i) is the best action to make for minute $i-1$, we have $OPT_A(i) = a_i + OPT_A(i-1)$; otherwise, we have $OPT_A(i) = a_i + OPT_B(i-2)$. In sum, we have

$$OPT_A(i) = a_i + \max\{OPT_A(i-1), OPT_B(i-2)\}. \quad (6)$$

   Similarly, we get the recursive relation for $OPT_B(i)$:

$$OPT_B(i) = b_i + \max\{OPT_B(i-1), OPT_A(i-2)\}. \quad (7)$$

   The algorithm initializes $OPT_A(0) = 0$, $OPT_B(0) = 0$, $OPT_A(1) = a_1$ and $OPT_B(1) = b_1$. Then the algorithm can be written as follows:

$OPT_A(0) = 0$; $OPT_B(0) = 0$;
$OPT_A(1) = a_1$; $OPT_B(1) = b_1$;
for $i = 2, \cdots, n$ **do**
  $OPT_A(i) = a_i + \max\{OPT_A(i-1), OPT_B(i-2)\}$;
  Record the action (either stay or move) in minute $i-1$ that achieves the maximum.
  $OPT_B(i) = b_i + \max\{OPT_B(i-1), OPT_A(i-2)\}$;
  Record the action in minute $i-1$ that achieves the maximum.
end for
Return $\max\{OPT_A(n), OPT_B(n)\}$;
Track back through the arrays OPT_A and OPT_B by checking the action records from minute $n-1$ to minute 1 to recover the optimal solution.

It takes $O(1)$ time to complete the operations in each iteration; there are $O(n)$ iterations; the tracing backs takes $O(n)$ time. Thus, the overall complexity is $O(n)$.