Introduction to Algorithms	**Solution Set 3**
CS 4820, Spring 2014	

(1) *(10 points)* Solve Chapter 6, Exercise 6 in Kleinberg & Tardos.

Solution

Let $S[i,j]$ denote the slack of a line that consists of the words $w_i, \ldots, w_j$: $S[i,j] := L - \left(\sum_{k=i}^{j} c_k\right) - (j-i)$. That is, $w_i, \ldots, w_j$ cannot be assigned to the same line if and only if $S[i,j] < 0$. Define the cost of a formatting to be the sum of the squares of the slacks of all lines in the formatting; an optimal formatting is a valid formatting with the minimum cost.

Let $\mathrm{OPT}[i]$ be the cost of an optimal formatting of $\{w_1, \ldots, w_n\}$. Then we have

$$\mathrm{OPT}[0] = 0.$$

Suppose $w_j, \ldots, w_i$ are assigned to the last line in an optimal formatting of $\{w_1, \ldots, w_i\}$. Then the cost of the formatting is given as the sum of the cost of the previous lines (i.e. $\mathrm{OPT}[j-1]$) and the square of the last line's slack (i.e. $(S[j,i])^2$); therefore, we can compute $\mathrm{OPT}[i]$ simply by trying every choice of j:

$$\mathrm{OPT}[i] = \min_{j \in \{1,\ldots,i\}: S[j,i] \geq 0} \left[\mathrm{OPT}[j-1] + (S[j,i])^2\right].$$

$\mathrm{OPT}[n]$ is the desired optimum. This recurrence yields the natural algorithm where one evaluates $\mathrm{OPT}[0], \ldots, \mathrm{OPT}[n]$ in the increasing order of the index. The actual optimal formatting is easily obtained by 'tracing back' the table. This process may be facilitated by recording $\mathrm{OptChoice}[i] := \arg\min_{j \in \{1,\ldots,i\}: S[j,i] \geq 0} [\mathrm{OPT}[j-1] + (S[j,i])^2]$.

The correctness of the algorithm can be shown by induction on i. The induction hypothesis is that for every $i \leq i_0$, $\mathrm{OPT}[i]$ computed by the algorithm is the cost of an optimal formatting of $\{w_1, \ldots, w_n\}$, as originally defined. We already argued the validity of the recurrence; this argument establishes the induction step in the proof of correctness.

Running time. $S[i,j]$ can be computed in $O(n)$ time; each time we compute one of n entries of OPT, we iterate over $O(n)$ choices of j. Thus, a naïve implementation of the given algorithm runs in $O(n^3)$ time.

Note that $S[i,j]$ can be computed in $O(1)$ time given $\sum_{k=i}^{j} c_k$. We can incrementally compute this sum in $O(1)$ time as we try each choice of j; hence, the algorithm can be improved to run in $O(n^2)$ time.

(2) *(10 points)* You're consulting for the state highway authority on a project to determine where they should place speed limit signs on a new highway. Assume that the highway has a length of L miles, and that points on the highway are identified by non-negative numbers d representing their distance from the west endpoint. The highway authority wants to enforce two types of speed constraints: a maximum speed of $s_{\max}$ miles per hour, which applies to the entire length of the highway, and local speed limits which can be expressed by ordered pairs of integers (s_i, d_i) expressing the constraint that the cars should be moving at a speed *no faster than* s_i miles per hour when passing through point d_i on the highway. There are n such constraints, numbered (s_1, d_1) through (s_n, d_n) with $d_1 < d_2 < \cdots < d_n$.

A speed limit sign is designated by a pair of integers (s, d), meaning that at point d on the highway there is a sign telling drivers that their speed must be at most s when traveling between d and d', where $d' > d$ is the position of the next speed limit sign on the highway, or $d' = L$ if there are no speed limit signs after d. For reasons that make sense only to bureaucrats and CS 4820 professors, the highway authority has decided that they can only use k speed limit signs, where k is a positive number less than n. A collection of speed limit signs is *valid* if:

1. There are at most k signs.
2. One of the signs is at the highway's starting point, i.e. at 0.
3. A driver who obeys the posted speed limits cannot possibly violate the maximum speed constraint $s_{\max}$, nor any of the local constraints $(s_1, d_1), \ldots, (s_n, d_n)$.
4. None of the signs is located at a point in the set $\{d_1, \ldots, d_n\}$.

Subject to these conditions, the highway authority wants to allow drivers to get from 0 to L as rapidly as possible while obeying the speed limits. Design an efficient algorithm that computes the minimum time T such that there is a valid collection of speed limit signs allowing drivers to get from 0 to L in T hours.

Solution

Lemma 1. *In any optimal solution, if a speed limit sign is placed at some location $d > 0$, then $|d - d_i| = 1$ for some i.*

Proof. Assume without loss of generality that the local speed constraints are numbered such that $d_1 < d_2 < \cdots < d_n$. Suppose the optimal solution places a speed limit sign (s, d) at some location $d > 0$, and let $i = \min\{j \mid d_j > d\}$. If the sign immediately preceding (s, d) in the optimal solution sets a speed limit faster than s, then the sign at (s, d) should be moved to exactly $d_i - 1$ to allow drivers to go at the faster speed for as long as possible. If the preceding sign sets a speed limit slower than s, then the sign at (s, d) sign should be moved to exactly $d_{i-1} + 1$ (or to 0, if $i = 1$) to allow drivers to speed up to the faster speed as soon as possible. □

Now let $P = \{p_0 < p_1 < p_2 < ... < p_m\}$ be the set of all positions where a speed limit sign could possibly be posted in the optimal solution, according to Lemma 1. In other words, $p_0 = 0$ and $\{p_1, p_2, \ldots, p_m\}$ contains all the numbers in the set $\{d_1 - 1, d_1 + 1, d_2 - 1, d_2 + 1, \ldots, d_n - 1, d_n + 1\}$. For convenience, we also define $p_{m+1} = L$, the length of the road. Let $\mathrm{OPT}(i, j)$ denote the

minimum length of time that it takes drivers to reach location p_i while obeying a valid collection of at most j speed limit signs. To write down the algorithm, we need to introduce one more piece of notation: for $r < i$ we define $s(r, i)$ to be the *fastest constant speed* that a driver can legally maintain between points p_r and p_i on the road. Thus

$$s(r, i) = \min \{s_{\max}, \min\{s_j \,|\, p_r < d_j < p_i\}\}. \tag{1}$$

We can compute all the values $s(r, i)$ in $O(n^2)$ time. Proving this is actually slightly subtle; here's one way to do it. First, after performing $O(n \log n)$ work to sort the set $\{d_1, d_2, \ldots, d_n\}$ we can assume that $d_1 < d_2 < \cdots < d_n$. For $1 \le a \le b \le n$, define $\sigma(a, b)$ to be the minimum element of the set $\{s_a, s_{a+1}, \ldots, s_b\}$. Using the formula

$$\sigma(a, b) = \begin{cases} s_a & \text{if } a = b \\ \min\{\sigma(a, b-1), s_b\} & \text{if } a < b \end{cases}$$

we can fill in all the values in the table $\sigma(a, b)$ in $O(n^2)$ time using an outer loop over $a = 1, 2, \ldots, n$ and an inner loop over $b = a, a+1, \ldots, n$. Now for $i = 0, 1, \ldots, m$ define $a(i)$ to be the minimum index j such that $d_j > p_i$ (or $n+1$ if no such j exists) and define $b(i)$ to be the maximum index j such that $d_j < p_i$ (or 0 if no such j exists). Having already sorted the sequences $d_1, d_2, \ldots, d_n$ and $p_0, p_1, \ldots, p_m$, it takes only $O(n)$ time to compute all the values $a(i), b(i)$ for $i = 0, 1, \ldots, m$. Finally, we can compute $s(r, i)$ for any $0 \le r < i \le m$ in $O(1)$ time using the fact that

$$s(r, i) = \begin{cases} \sigma(a(r), b(i)) & \text{if } a(r) \le b(i) \\ s_{\max} & \text{otherwise.} \end{cases}$$

Having thus explained an efficient way to fill in the entries in the table $s(r, i)$, we can now present the dynamic programming algorithm to find an optimal placement of speed limit signs.

Algorithm 1 (SpeedLimit)

Require: $p_0, p_1, \ldots, p_{m+1}$ are in increasing order.

1: Compute $s(r, i)$ for all $0 \le r < i \le m + 1$.
2: $\mathrm{OPT}(0, j) = 0$ for $j = 1, 2, \ldots, k$.
3: **for** $i = 1, 2, .., m + 1$ **do**
4: $\quad \mathrm{OPT}(i, 1) = p_i / s(0, i)$.
5: $\quad$ **for** $j = 2, .., k$ **do**
6: $\quad\quad \mathrm{OPT}(i, j) = \min_{r<i} \left\{ \mathrm{OPT}(r, j-1) + \frac{p_i - p_r}{s(r,i)} \right\}$.
7: $\quad$ **end for**
8: **end for**
9: Output $\mathrm{OPT}(m + 1, k)$.

The running time is $O(n^2 k)$ because it takes $O(n^2)$ steps to compute all of the values $s(r, i)$, and the main loop of the dynamic program has $O(nk)$ loop iterations with $O(n)$ work per iteration. The algorithm's correctness follows, as usual, by induction on the number of entries that have been computed in the dynamic programming table. The induction hypothesis is that all the previously-computed entries are correct. The induction step follows from the correctness of the formula given in Step 6 of the algorithm. To justify that formula, we use Lemma 1 which implies that the fastest way to reach p_i using a valid collection of j speed limit signs involves placing the

j-th sign at some location p_r for $r < i$. The speed limit posted on that sign can not be greater than $s(r, i)$ — by the definition of $s(r, i)$ — so the minimum time required to drive from the j-th speed limit sign to p_i is $\frac{p_i - p_r}{s(r,i)}$. The minimum time required to drive from 0 to p_r using $j - 1$ speed limit signs is $\mathrm{OPT}(r, j - 1)$. Adding these terms, we obtain the formula listed in Step 6 of the algorithm.