CS 4820, Spring 2014	**Homework 3, Problem 1**
Name: Piyush Maheshwari	
NetID: pm489	
Collaborators: None	

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

Solution

Problem Statement

Given a sequence of words W = $\{w_1, w_2, ..w_n\}$ where w_i contains c_i characters. Maximum length of each line of text is L. If w_i to w_k are assigned to a line then $\sum_{i}^{k-1}(c_i + 1) + c_k \leq L$ must hold for the line to be valid. We have to find a partition of set of words W into valid lines so that sum of squares of slack of all valid lines including the last one is minimized.

Building the algorithm

Let s_{ij} be the slack if we have a line which starts at w_i and ends at w_j. Using this definition $s_{ij} = \{\ L - \sum_{i}^{k-1}(c_i + 1) + c_k\ \}$ if $L \geq \sum_{i}^{k-1}(c_i + 1) + c_k$ else INF. Here INF stands for a very big number which is greater than all numbers possible in this problem. If s_{ij} is INF then this would mean that the line formed by words w_i to w_j is an invalid line. We will pre compute this slack matrix so that we don't have to re compute this values in our main algorithm.

Now consider the last word w_n. This word can be part of a line which can start at any $j < n$. Let OPT(n) be the minimum slack of words w_1 to w_n partitioned optimally.
OPT(n) = min { OPT(j-1) + s_{jn} } for j = 1 to i

This recurrence will give us optimal slacks for all prefix word set and our answer will be simply OPT(n). We can also find all the partition points by storing at which j each we the optimal value for OPT(i). We can store this value in an array *prev*[]. Hence the partition points can be recovered as prev[n], prev[prev[n]]... and so on until we reach the beginning of the word list i.e prev[i] = 1.

Also s_{ij} can be precomputed using the following recurrence :
s_{ii} = L - c_i $\forall i$ since this means that this is the only word in this line. Here it is also assumed that the input is such that $c_i \leq L\ \forall i$ otherwise the words cannot be partitioned.
$s_{ij} = s_{i,j-1} - c_j - 1\ \forall i \neq j$ since if we want words w_i to w_j in a line, then we can simple subtract c_j -1 from the slack of w_i and the preceding word to j. We also set s_{ij} equal to INF is the slack is negative to signify that it is an invalid partition so that it does not get included in the solution.

Algorithm

Algorithm 1

1: **procedure** PRETTY_PRINTER($W[], C[], L$)
2: $s[i,j] \leftarrow INF\ \forall i,j$ ▷ Slack if words i to word j form a line
3: **for all** $i = 1, n$ **do**
4: $s[i,i] \leftarrow L - c_i$
5: **for all** $j = i+1, n$ **do**
6: **if** $s[i,j-1] - c_j - 1 \geq 0$ **then**
7: $s[i,j] \leftarrow s[i,j-1] - c_j - 1$
8: **for all** $i = 1, n$ **do** ▷ We want to minimize the square of slacks
9: **for all** $j = i+1, n$ **do**
10: **if** $s[i,j] \neq INF$ **then**
11: $s[i,j] \leftarrow s[i,j] * s[i,j]$
12: $OPT[i] \leftarrow INF\ \forall i$ ▷ OPT(i): the minimum slack by partitioning words from 1 to i
13: $prev[i] \leftarrow 0\ \forall i$ ▷ prev(i): the starting point of the line in which w_i lies.
14: $OPT[0] \leftarrow 0$ ▷ Handling the base case since the min slack for zero words is zero
15: **for all** $i = 1, n$ **do**
16: **for all** $j = 1, i$ **do**
17: **if** $s_{ji} \neq INF$ **then** ▷ If w_j to w_i form a valid line
18: **if** $OPT[i] > OPT[j-1] + s_j k$ **then**
19: $OPT[i] \leftarrow OPT[j-1] + s_j k$
20: $prev[i] \leftarrow j$
21: **return** $OPT[n]$, $prev[]$

Proof of Correctness

Proof. We will prove this by induction. We claim that OPT(k) is the optimal slack value for words from w_1 to w_k. The base case for k= 0 is trivially true since the slack is zero for no words. Now consider OPT(k+1). Using the argument above we know that
optimal(n) = min { optimal(j-1) + s_{jn} } for j = 1 to i
Now assume that inductive hypothesis holds i.e OPT(j) is optimal slack for all j from 1 to k assuming strong induction
optimal(k+1) = min { OPT(j-1) + s_{jn} } for j = 1 to k+1, which means
optimal(k+1) = OPT(k+1) which proves the inductive step. □

Complexity Analysis

Steps 3-7 and 8-11 take $O(n^2)$. Steps 14-19 take $O(n^2)$. Partition points can be recovered as prev[n], prev[prev[n]]... and so on until we reach the beginning of the word list i.e prev[i] = 1. This takes $O(n)$ time.
Hence total time complexity $O(n^2)$
Total space complexity $O(n^2)$