Name: Amit Nadkarni
arn6851@rit.edu

Lab Assignment 3

QUESTION:- PRETTY PRINTER PROBLEM

To give an efficient algorithm to find the partition of set of words *W* into valid lines, so that the sum of the squares of the slack (including the last line) is minimized.

The idea behind pretty printer is to arrange words in the line such that the number of extra spaces in the lines is balanced, meaning, no line has more number of extra spaces than the line with few amount of extra spaces.

Why the cost function is Square of the slack calculated?

Consider the paragraph with three lines, where in first line has 3 extra spaces (slack) , and other two lines are justified properly. Hence total number of extra spaces are 3+0+0. Now consider the paragraph with three lines having one extra space on each line. Hence, even here the total number of extra space is 1+1+1=3. But this alignment has least slack computation. Hence squaring the slack cost will ensure the alignment and minimize the slack cost.

Recurrence Expression

Optimal Cost [n]= Minimum (**Summation over m from 1 to n**
Optimal Cost [m-1] + Cost Matrix[m][n]) …...n > 0

0 … otherwise

Thus the recurrence expression shows that to compute the optimal cost to arrange the words in the line, we make use of the optimal cost required to place the word before it , and decide whether to place the word on the same line or on the new line. Thus it has the overlapping sub problem.

An array, Optimal Cost stores the minimum cost required to arrange the words on the line.

Algorithm:

Input: Sequence of words.

Output: Text containing the minimum slack.

1) Store the words in the in a string array

2) Calculate the length of each word in the above array and store it in an array

3) Decide the length of the line (i.e. Number of characters allowed on each line)

4) Compute the slack of each word being placed on the line. Slack is the number of extra spaces left on each line, Slack is the difference between the extra spaces on the line and length of each word placed on the line. (Goal is to minimize this slack).

5) Compute the cost of each word to be considered as the part of the output, square the slack computed from the earlier step. If the slack is negative, I.e the length of words has exceeded the capacity of the line then, in the cost matrix we denote as infinity in the cost matrix.

6) Using the above recurrence expression we calculate the optimal cost of arranging the words on the line such that the overhead cost is minimized.

7) Store the index at which the minimum cost of arranging each is computed in the array of size equivalent to the number of words. This tracks the starting of the new line.

Example:

Input sequence: " Thy in
my house"

Output Text: "Thy
in my
house"

Length of the line is 7.

Dynamic Programming tables:

Slack Matrix (Length of the line – length of the word)

0	**1**	**2**	**3**	**4**
1	4	1	-2	-8
2	-	5	2	-4
3	-	-	5	1
4	-	-	-	2

Cost Matrix (Calculated by squaring the slack computed)

0	**1**	**2**	**3**	**4**
1	16	1	Infinity	infinity
2	-	25	4	infinity
3	-	-	25	1
4	-	-	-	4

Optimal Cost Matrix of Arranging words from 1 to n (Computed using recurrence expression)

0	**1**	**2**	**3**	**4**
0	16	1	20	4

Index matrix to track the starting of the new line

0	1	1	2	4
0	**1**	**2**	**3**	**4**

From the above array, the indexes depict the word and the value at index -1 depicts the staring of new line after that word. For eg: New Line started after the word 4-1=3 , i.e third word. Second line started after the first word 2-1, first word.

Time Complexity Analysis

Time required: Big-oh (n ^ 2) where *n* is the number of words in the input sequence.

Constrains:

1. Length of the word should be less than that of the size of the line specified.

2. Hyphen and the word in continuation is not considered.