2/12

CS 1510
Algorithm Design

Dynamic Programming

Problems 18, 20, and 22

Due Wednesday September 24, 2014

Buck Young and Rob Brown

62.6% 67/107

Problem 18

Input: Ordered list of n words and an ideal line-length L. With the length of the ith word being w_i.

Output: A layout of lines (none of which can extend beyond L) which minimizes the total penalty. Total penalty is defined as the maximum of all line penalties (the line penalty for a line of length K is L - K).

Pruning rules:

1) If any node creates a line longer than L, prune it

2) If any nodes at the same level have the same length (therefore the same words on the line), prune all but the minimum total penalty.

Algorithm:

2

What is A supposed to hold? The last line may not be the worst.

```
1:  // Initialize multi-dimensional array of size $n$+1 by $L$+1 to positive infinity
2:  A = new array($n$+1)($L$+1)
3:  A[ * ][ * ] = $\infty$
4:
5:  for lvl = 0 to $n$ - 1 do
6:      for t = 1 to $L$ do
7:          // Initialize this levels penalty based on the word by itself on a line
8:          A[ lvl + 1 ][ $w_{lvl+1}$ ] = $L$ - $w_{lvl+1}$
9:
10:         // Set current cell to be current min in row ("left child")
11:         A[ lvl + 1 ][ t ] = min( A[ lvl + 1 ][ t ], A[ lvl + 1 ][ t - 1 ] )
12:
13:         // Consider if current word was added to the line above ("right child")
14:         if t - $w_{lvl+1}$ is in bounds && A[ lvl ][ t - $w_{lvl}$ ] - $w_{lvl+1}$ $\geq$ 0 then
15:             A[ lvl + 1 ][ t ] = min( A[ lvl + 1 ][ t ], A[ lvl ][ t - $w_{lvl+1}$ ] - $w_{lvl+1}$ )
```

} you must include all words...

*L*2 – *L*3: This initializes a multi-dimensional array with indices 0 to n and 0 to L and sets all cells to have a value of positive infinity.

*L*5 – *L*6: From top to bottom and left to right

*L*8: First, in order to fill out the next line, consider the word for that level. Go to the cell which matches the length of this word and put in the value as if this word were on a line by itself.

*L*11: Now, iterate through each cell in the row and fill in a value based on the minimum between itself and its left neighbor.

*L*14 – *L*15: If we can complete the next operations, lets do them – note we must be wary of being out of bounds to the left and we also don't want to fill in a value less than 0. At any given cell, we want to consider the minimum between what is already there and what the possible penalty could be if we added this word to the line above. The if statement checks to make sure we would not extend beyond L (otherwise we would be in the less than zero range).

Backtracing: This will give us a table where our last-line penalty is in the lower rightmost cell. From here, we can backtrace to find our layout which minimizes the total penalty. In order to do this, we head up a level and back by the length of the word. By doing this we can determine if our word was added to the line above or not. If it was not, simply write the word down as the last line. If it was, continue the process until you have a completed line. Once you have a completed line, head up a level and start in the lower-rightmost cell as if this level was the lowest boundary of the array. This lower-rightmost cell's value is the line penalty for the line we are about to construct. Continue with the instructions until you have all the lines in the layout. An example table and backtrace is provided below.

Example Input: "THIS IS A GROUP WORDS", $L = 6$

					t					
		0	1	2	3	4	5	6		
	0	∞	∞	∞	∞	∞	∞	∞		
	1	∞	∞	∞	∞	2	2	2		
lvl	2	∞	∞	4	4	4	4	⓪	$p_{L1} = 0$	(can add to line above)
	3	∞	5	5	3	3	3	3		
	4	∞	∞	∞	∞	∞	1	⓪	$p_{L2} = 0$	(can add to line above)
	5	∞	∞	∞	∞	∞	1	①	$p_{L3} = 1$	(cannot add to line above)

Generated Output:
($p = 0$) THIS IS
($p = 0$) A GROUP
($p = 1$) WORDS
($p_{total} = 1$)