Etan Soclof

DPWordWrapping

Dynamic Programming Approach

Given an array of words, W, initialize an array, P, that stores the minimum penalty for words 1…i. In this array, P[i] stores the total minimum penalty when word[i] is the last word to be displayed and, therefore, W[i] is also the last word on whatever line it happens to be on. When adding another word, W[i+1], we need to determine whether rearranging words 1…i will yield a smaller penalty or not. To determine this, we can iterate through every possible line configuration of the line that ends with W[i+1]. If the line which ends with word[i+1] begins with word[j], that means that the previous line ends with W[j-1]. Because P[j-1] stores the minimum penalty when j-1 is the last word on its line, we can add the value of P[j-1] to the penalty of a line containing words j…i+1. Consider every value of j and find the minimum. Increment i and repeat recursively.

Finding the penalty of a line containing words j…i+1 is simple. Simply take the length allowed for each line, M, and subtract the length of words j…i+1 and spaces to find the penalty amount of empty space at the end of the line, p. As the algorithm increments j, simply add the length of word[j-1] + 1 to p. This calculation, throughout the entire program, is $O(n^2)$ where n is the number of words to be displayed. (Note: the requirements doc says that runtime should be O(n x M) but I could not figure out how to get that runtime. This is the closest that I could figure out.)

Recurrence Relation

Let $OPT(W_i)$ be the minimum **total cost** penalty for a layout containing W[0]…W[i]. Let M be the length allowed for each line. Let W[i] be the length of the word at W[i] and W[j…i] be the cumulative length of words from W[j] to W[i]. Given the above, we can define the following recurrence relation:

$$OPT(W_i) = \begin{cases} (M - W[i])^2 & \text{if } i \text{ is } 0 \\ min\left\{ OPT(W_{j-1}) + (M - W[j \dots i])^2 \right. & \text{otherwise} \end{cases}$$

where j starts from 0 and is incremented by 1 to i.

An example is in order ('cuz this is so darn confusing, pardon my language):

Consider an array Strings W = {"ab", "cd", "e"}; and a maximum length of M=5.

1. First, consider only W[0]. The penalty of a layout containing W[0] is $(M - W[0].length)^2 = (5 - 2)^2 = 9$. So $OPT(W_0) = 9$.
2. Next, consider W[0] and W[1]. There are two possibilities as to how the line that ends with W[1] is configured. It can either start with W[0], or it can start with W[1]. Try both possibilities:
   A. If it starts with W[0], the penalty is $(M - (W[0].length + W[1].length + 1))^2 = 0$.
   B. If it starts with W[1], the total penalty is the penalty of the entire layout with the last line ending with W[0] + the penalty of the line ending with W[1]. We already have tabulated the total penalty of the layout where the last line ends with W[0] in step 1: $OPT(W_0) = 9$. The penalty of the line starting and ending with W[1] = $(M - W[1].length)^2 = 9$. Add $OPT(W_0) = 9$ and the penalty of the line ending with W[1] to get the total penalty of this layout configuration: 18.
   C. 0 < 18. Therefore, $OPT(W_1) = 0$.
3. Next, consider W[0], W[1], and W[2]. There are three possibilities as to how the line that ends with W[2] is configured. It can either start with W[0], W[1], or W[2].

A. If it starts with W[0], note that M - (W[0].length + W[1].length + W[2].length + 2) = -2. These three words cannot fit on a single line.
B. If it starts with W[1], the total penalty is the penalty of the entire layout with the last line ending with W[0] + the penalty of the line ending with W[2]. We already have tabulated the penalty of the line ending with W[0] in step 1: $OPT(W_0) = 9$. The penalty of the line starting with W[i] and ending with W[2] = (M - (W[1].length + M[2].length + 1))^2 = 1. Add $OPT(W_0)$ and the penalty of the line ending with W[2] to get 10.
C. If it starts with W[2], the total penalty is the penalty of the entire layout with the last line ending with W[1] + the penalty of the line ending with W[2]. We already have tabulated the total penalty of the layout in the case that the last line ends with W[1] in step 2: $OPT(W_1) = 0$. The penalty of the line starting and ending with W[2] = (M - W[2].length)^2 = 16. Add $OPT(W_1) = 0$ and the penalty of the line ending with W[1] to get the total penalty of this layout configuration: 16.
D. 10 < 16. Therefore, $OPT(W_1) = 10$.

Optimal Layout Construction

When $OPT(W_i)$ is determined, the value of j that yielded that optimal penalty is recorded in a layout array L at index i. Note that j represents the first word on the line in which W[i] is the last word. Therefore, the line which ends with W[i] starts from W[L[i]] (because, once again, L[i] = j) and ends with W[i]. Once we have determined that the line that ends with W[i] starts with W[L[i]], we know that the preceding line ends with W[L[i] - 1] (because, L[i] -1 = j - 1). Therefore, we move to index L[i] - 1 in L to determine the first word on this line. Repeat until all lines have been constructed.

Big-O and Space Requirements

My algorithm is O(n^2) where n is the number of words to be displayed. M, the maximum length of a line, does not factor into the runtime at all. It is simply used as a method of determining the penalty of a particular line (penalty of a line = (M - W[i])). The algorithm is O(n^2) because for each additional word i considered, we must consider all of the possibilities of how the line that ends with i is constructed. It can start with word 0, or word 1… word i. So, for every individual value of i, finding the optimal penalty is O(i), which is essentially O(n). Because the above calculation must be done for every single word, the runtime of the entire algorithm is O(n^2). The algorithm uses two arrays whose length is equal to the number of words. Therefore, the space cost is O(n).