CPSC 413 — Solutions for Quiz #7

Takehome Test on Dynamic Programming

November 29, 1999

Consider the problem of neatly printing a paragraph on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$, measured in characters. We want to print this paragraph neatly on a number of lines so that no word is split between lines. Each line holds a maximum of M characters (including spaces).

Our criterion of "neatness" is as follows. If a given line contains words $i, i+1, \ldots, j$ and we leave exactly one space between words, the number of extra spaces at the end of the line is

$$M - j + i - \sum_{k=i}^{j} l_k.$$

We wish to minimize the sum, over all lines except the last, of the **cubes** of the numbers of extra spaces at the end of lines.

1. *15 marks:* Design and write down a dynamic programming or memoization algorithm that could be used to print a paragraph of n words neatly on a printer. Your algorithm should accept M and $l_1, l_2, \ldots, l_n$ as input. It should report the index of the last word on each line as output.

   Your algorithm should use a number of operations that is polynomial in n.

   Solution: The required output for this problem is a sequence of integers $\langle i_1, i_2, \ldots, i_h \rangle$, where $1 \le i_1 < i_2 < \cdots < i_h = n$, so that i_j is the index of the last word appearing on line j, for $1 \le j \le h$.

   This problem is an optimization problem. A possible output sequence is *valid* if and only if

   $$i_j - (i_{j-1} + 1) + \sum_{k=i_{j-1}+1}^{i_k} l_k \le M$$

   (where $i_{-1} = 0$) for $1 \le j \le h$, so that the words on any given line fit into the space available for them.

Suppose $\langle i_1, i_2, \ldots, i_h \rangle$ is a valid output sequence, and let

$$p\left(\langle i_1, i_2, \ldots, i_h \rangle\right) = \sum_{j=1}^{h-1} \left(M - i_j + i_{j-1} + 1 - \sum_{k=i_{j-1}+1}^{i_j} l_k \right)^3$$

where, once again, $i_{-1} = 0$. A valid sequence $\langle i_1, i_2, \ldots, i_h \rangle$ is *optimal* and, therefore, a correct solution, if $p\left(\langle i_1, i_2, \ldots, i_h \rangle\right)$ is as small as possible.

It is necessary (and sufficient) that $l_i \leq M$ for $1 \leq i \leq n$, so that each word fits onto a line, in order for a solution to exist. This will be assumed for the rest of this solution.

Now suppose that

$$(n-1) + \sum_{k=1}^{n} l_k \leq M;$$

then all the input words could be fit onto a single line of text, so that $\langle n \rangle$ would be a valid sequence. Since $p\left(\langle n \rangle\right) = 0$ and $p\left(\langle i_1, i_2, \ldots, i_h \rangle\right) \geq 0$ for any other valid sequence $\langle i_1, i_2, \ldots, i_h \rangle$, $\langle n \rangle$ would be optimal in this case.

Otherwise, suppose $1 \leq j < n$ and that the first j words fit onto a line — that is, suppose

$$(j-1) + \sum_{k=1}^{j} l_k \leq M.$$

Then the number of extra blank spaces at the end of the first line is

$$M - j + 1 - \sum_{k=1}^{j} l_k,$$

and the best way to fit the remaining words onto the remaining lines of text is to end the second line at word i_2, the third line at word i_3, and so on, where $\langle i_2, i_3, \ldots, i_n \rangle$ is a solution for an instance of a slightly generalized version of this problem with $n-j$ input words of lengths $l_{j+1}, l_{j+2}, \ldots, l_n$ and indices $j+1, j+2, \ldots, n$ (instead of $1, 2, \ldots, n-j$), and line width M.

Let

$$P(j) = \sum_{g=2}^{h-1} \left(M - i_g + i_{g-1} + 1 - \sum_{k=i_{g-1}+1}^{i_g} l_k \right)^3$$

so that $P(j)$ is the smallest possible value of the function to be minimized, on the inputs described above, and so that

$$p\left(\langle i_1, i_2, \ldots, i_h \rangle\right) = \left(M - j + 1 - \sum_{k=1}^{j} l_k \right)^3 + P(j).$$

Furthermore, let $P(0)$ be the smallest possible value of the function to be minimized, on the originally given inputs $l_1, l_2, \ldots, l_n$ and M. Then

$$P(0) = \begin{cases} 0 & \text{if } n - 1 + \displaystyle\sum_{k=1}^{n} l_k \le M, \\ \displaystyle\min_{\substack{1 \le j \le n \\ j-1+\sum_{k=1}^{j} l_k \le M}} \left(\left(M - j + 1 - \sum_{k=1}^{j} l_k \right)^3 + P(j) \right) & \text{otherwise,} \end{cases}$$

and, more generally, if $0 \le i \le n-1$, let

$$P(i) = \begin{cases} 0 & \text{if } n - i + 1 + \displaystyle\sum_{k=i+1}^{n} l_k \le M, \\ \displaystyle\min_{\substack{i+1 \le j \le n \\ j-i-1+\sum_{k=1}^{j} l_k \le M}} \left(\left(M - j + i + 1 - \sum_{k=i+1}^{j} l_k \right)^3 + P(j) \right) & \text{otherwise.} \end{cases}$$

Similarly, if $0 \le i < n$ and $seq(j)$ is a correct output for inputs $l_{j+1}, l_{j+2}, \ldots, l_n$ (with indices $i_{j+1}, i_{j+2}, \ldots, i_n$) and M, then it is possible to choose $seq(i)$ such that

$$seq(i) = \begin{cases} \langle n \rangle & \text{if } n - i + 1 + \displaystyle\sum_{k=i+1}^{n} l_k \le M, \\ \langle j \rangle + seq(j) & \text{otherwise,} \end{cases}$$

where j is chosen so that $i + 1 \le j \le n$, $j - i - 1 + \sum_{k=i+1}^{j} l_k \le M$, and $P(i) = (M - j + i + 1)^3 + P(j)$, and where "+" denotes the concatenation of two sequences.

The algorithm shown in Figure 1 uses dynamic programming to solve this problem. It maintains two arrays of length n: For $0 \le i < n$, penalty$[i]$ is used to store the value $P(i)$ described above, and index$[i]$ stores the first entry of above sequence $seq(i)$. In order to make the algorithm asymptotically more efficient, a variable wordlength_sum is used to keep track of the sum of the lengths of the words that might be included on a given line. The algorithm first fills the entries of the two arrays, and then uses the array index to print out the desired output sequence.

2. *5 marks:* Analyze the running time requirements of your algorithm.

Solution: The above algorithm uses $\Theta(n^2)$ operations in the worst case.

A version of the algorithm that does not use a variable like wordlength_sum to accumulate sums of words lengths, but that computes these from the inputs whenever they are needed, would probably use $\Theta(n^3)$ operations in the worst case, instead.

```
for i := n − 1 down to 0 do
    if n − i − 1 + Σ_{k=i+1}^{n} l_k ≤ M then
      index[i] := n
      penalty[i] := 0
    else
      index_candidate := i + 1
      wordlength_sum := l_{i+1}
      penalty_candidate := (M − wordlength_sum)^3 + penalty[i + 1]
      j := i + 2
      while j ≤ n and j − i − 1 + wordlength_sum + l_j ≤ M do
          wordlength_sum := wordlength_sum + l_j
          if (M − j + i + 1 − wordlength_sum)^3 + penalty[j] < penalty_candidate then
              index_candidate := j
              penalty_candidate := (M − j + i + 1 − wordlength_sum)^3 + penalty[j]
          end if
          j := j + 1
      end while
      index[i] := index_candidate
      penalty[i] := penalty_candidate
    end if
end for
current_index := index[0]
print current_index
while current_index < n do
    current_index := index[current_index]
    print current_index
end while
```

Figure 1: A Dynamic Programming Algorithm for Printing