Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

24th November 2016

Data Structures & Algorithm Solutions to Sheet P9 AS 16

Solution P9.1 *Line breaks.*

Again, we will use dynamic programming for this task: for every $i \in \{0, \ldots, n\}$, let m_i be the total penalty if the last line break was just before word w_i (not considering the special case for last word of a paragraph here). We may set $m_0 = 0$, as a break before the word w_0 is the start of the text. Note that the actual characters of the text did not matter, only the word lengths.

Now to compute m_i if we know all previous values of $m_{j<i}$, consider how can the solution with the last break before w_i look: For some $x > 0$, it will consist of the best solution with the last line break before w_{i-x}, a line break and then x words $w_{i-x}, w_{i-x+1}, \ldots, w_{i-1}$ on the last line. The penalty of the best solution with the last line break before w_{i-x} is already known in m_{i-x}, and it is straightforward to compute the penalty for line with words $w_{i-x}, w_{i-x+1}, \ldots, w_{i-1}$. If we try all the possible values of $0 < x \leq i$ such that the last line has length at most w, we take m_i as the minimal total penalty over all such x. We might need to check $O(w)$ values of x since $x \leq (w+1)/2$ (there can be at most $(w+1)/2$ space-separated words on a line).

Now for the last line of paragraph, we only need to change the way m_n is computed: We still consider all feasible values of x (with the last line fitting into w characters) and take minimum of m_{n-x}, but do not add the last line penalty. This can be accomplished with a simple condition.

This gives us solution with running time $O(nw^2)$, since we try $O(w)$ values of x for every of n words, and in every case it can take us $O(w)$ time to calculate the penalty of a given range of words $w_{i-x}, w_{i-x+1}, \ldots, w_{i-1}$ on a single line.

However, this can be made faster by for example pre-computing the sum of word lengths as $s_i = \sum_{j=0}^{i} |w_j|$, and then the length of line with words $w_a, \ldots, w_b$ is $s_b - s_a + (b - a - 1)$ (last part for the spaces). Another solution is to start with $x = 1$ and increment it, and in every step remember the length of the considered last line so far, incrementing it with every added word – the penalty is then computed in $O(1)$ time.

With either solution, we get $O(nw)$ time solution.

Solution programs

On the lecture website, you can find a solution running in time $O(nw)$ that uses the last approach. The solution source contains further comments on the implementation.

Data

The tests were mostly with random word lengths, with w from 20 to 1000 and with n from 100 to 60 000 in various combinations. Every judge test also contained 1-3 small additional special

cases, similarly to the example and the test data provided.

Notes on submitted solutions.

The problem tempted to use 2D arrays for subproblems of the type $m_{i,j}$ ="best penalty for words $w_i, \ldots, w_{i+j}$" or a similar schema. While this approach could generally work, the array that would have been $n \times n$ would not fit into memory for larger n, and filling it would be too slow anyway. A small improvement was to only have a similar 2D array $n \times w$ for words on single line, and some of those solutions could possibly work in $O(nw^2)$ or maybe even $O(nw)$ time, but the general idea was just a complication of the proposed solution with 1D array.