Eidgenössische
Technische Hochschule
Zürich

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

Department Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

17th November 2016

Algorithmen & Datenstrukturen Exercise Sheet P9 AS 16

Hand-in: Before Thursday, 24th November 2016 10:00 via the online judge (source code only).

Exercise P9.1 *Line breaks.*

One of the basic problems of typesetting is breaking the words into lines and then breaking the lines into pages, with the resulting layout as beautiful and readable as possible. Your task is a (greatly simplified) version of the first part, that is to decide where to place the line breaks in a given text.

The input text T is a sequence of paragraphs $P_1, \ldots, P_k$ that are processed independently, and each paragraph P_i is a list of n_i words and has given page width $w_i > 0$. Every word in paragraph P_i has length at most w_i.

The output is the same text with one or more consecutive words on one line, every two words on a line are separated by exactly one space. All the characters (including spaces) have the same width and so the length $\text{len}(L)$ of line L is the number of word-characters and spaces in between them, for example "`This is an example.`" has length $4 + 1 + 2 + 1 + 2 + 1 + 8 = 19$.

The goal is to format the text such that every line of paragraph P_i has length at most w_i, but also as nicely as possible: Informally, we want all the lines to have length as close to w_i as possible. And formally, we assign every line L penalty $(w_i - len(L))^2$, that is the square of the number of spaces you would need to add to make the line exactly w_i characters long. For example a line with length exactly w has penalty 0 and a line with just one single-character word would have penalty $(w_i - 1)^2$. The last line of the resulting paragraph is an exception – the penalty of this line is always 0, as the length of the last line does not matter for typesetting.

For every paragraph, the goal is to find the optimal line breaks that minimize the sum of the penalties of the resulting lines. Note that a *"greedy"* solution that would make every line as long as possible before starting a new line is generally not optimal – see the example below.

Input The input consists of several paragraphs. The first line of the file contains the integer $k > 0$, the number of paragraphs to follow.

Each paragraph P_i is independent of the others and consists of several lines: The first line contains the integers $n_i > 0$, the number of words of the paragraph, and $w_i > 0$, the width of the page, separated by a space. The following n_i lines contain the words of the paragraph, one word each.

The words may consist of English letters, numbers and the following characters[1]: `-.,?!:;'"()[]{}`

[1] But the exact set should not matter to your program anyway.

Output For every paragraph, the output should contain a single line with the smallest possible penalty.

Grading You will get 1 bonus point for every 100 judge points, rounded down. Your may get up to 200 judge points. The program should be reasonably efficient and work in $O(w_1n_1+\cdots+w_kn_k)$ time to get full points.

Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=18985&problem=DA_P9.1`, enroll password is "`quicksort`".

Examples

Input

```
2
9 9
A
B
C
D
E
F
GGGGGGGGG
HHH
I.
8 18
Lorem
ipsum
dolor
sit
amet,
consectetur
adipiscing
elit.
```

Output

```
32
107
```

Example optimal formatting with marks for page width and line penalties.

```
A B C    | 16
D E F    | 16
GGGGGGGGG| 0
HHH I.   | 0 (last line)

Lorem ipsum       | 49
dolor sit amet,   | 9
consectetur       | 49
adipiscing elit.  | 0 (last line)
```

Notes For this exercise, we provide a program template as an Eclipse project archive on

the lecture website, which will load the input for you. The archive also contains more test data for your local testing. Additionally, we introduce the `Judge.java` program that will run your program `Main.java` on all the provided tests – just open and run `Judge.java` in the project as you would run `Main.java`. This does not change the way that your `Main.java` works and is just an extra tool. Also, the data are of course different and smaller than the data that will be used in the online judge.

Please let us know if you have any feedback on the provided local Judge.