Eidgenössische
Technische Hochschule
Zürich

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

Departement of Computer Science
Markus Püschel, David Steurer

3. December 2018

Datenstrukturen & Algorithmen — Blatt P11 — HS 18

Please remember the rules of honest conduct:

- Programming exercises are to be solved alone
- Do not copy code from any source
- Do not show your code to others

Hand-in: Sunday, 16. December 2018, 23:59 clock via Online Judge (source code only).
Questions concerning the assignment will be discussed as usual in the forum.

Those who cannot remember the past are condemned to repeat it.
— Dynamic Programming

Exercise P11.1 *Parenthesis.*

Input

You are given an arithmetic expression containing n numbers and $n-1$ operators, each either $+$ or $-$. Your goal is to perform the operations in an order that maximizes the value of the expression. That is, insert parentheses into the expression so that its value is maximized.

For example, for the expression $6-3-2+5$, the optimal pacing of parhenthisis is: $6-(3-2)+5=10$.

Input The first line of the input is an integer N, $1 \leq N \leq 10^5$, denoting the number of integers in the expression. The next line contains an expression of the form $v_1\ op_1\ v_2\ op_2\ v_3\ op_3 \ldots op_{N-1}\ v_N$ where $op_i \in \{+,-\}$ for $i \in [1 \ldots N-1]$ and $1 \leq v_j \leq 10^9$ for $j \in [1 \ldots N]$

Output The output contains a single line that represents the maximum value of the expression that can be obtained by placing zero or more pairs of parenthesis.

Grading You get 3 bonus points if your program works for all inputs. Your algorithm must complete the solution in $O(N)$ time complexity, and $O(N)$ space complexity. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD8H11P1`. The enrollment password is "`asymptotic`".

Example

Input:

```
4
6 - 3 - 2 + 5
```

Output:

```
10
```

Notes

For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD8H11P1.Parenthesis.zip` The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class.

Exercise P11.2 *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 $len(L)$ 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]: `-.,?!:;'"()[]{}`

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

Grading This exercise awards no points. The program should be reasonably efficient and work in $O(w_1 n_1 + \cdots + w_k n_k)$ time complexity. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD8H11P2`. The enrollment password is "`asymptotic`".

Examples

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

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 an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD8H11P2.LineBreaks.zip` The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class.