Code Jam 2020 - Round 2

Emacs++

Problem

In 2016's Distributed Code Jam, we introduced the Lisp++ language for Lisp fans who prefer a higher density of parentheses. Here is a reminder of how the language's syntax works:

A Lisp++ program is a string of balanced parentheses. More formally, a Lisp++ program consists of one of the following. (In this specification, *C* stands for some program code — not necessarily the same code each time.)

- `()` Literally, just an opening parenthesis and a closing parenthesis. We say that this `(` matches this `)`, and vice versa.
- `(`*C*`)` A program within a pair of enclosing parentheses. We say that this `(` matches this `)`, and vice versa.
- *CC* Two programs (not necessarily the same), back to back.

This year, we are pleased to announce Emacs++, a text viewer for Lisp++. Emacs++ displays a Lisp++ program of length **K** as a single long line with a cursor that you can move around. The cursor is a "block cursor" that is always located *on* one of the **K** characters in the program, rather than between characters.

At any point, you can perform one of the following three actions to move the cursor. (i represents the current position of the cursor, counting starting from 1 for the leftmost position.)

- Move the cursor one character to the left (or, if the cursor is already on the leftmost character, does nothing). This takes $\mathbf{L_i}$ seconds.
- Move the cursor one character to the right (or, if the cursor is already on the rightmost character, does nothing). This takes $\mathbf{R_i}$ seconds.
- Teleport the cursor to the parenthesis matching (as described above) the parenthesis that is the i-th character. This takes $\mathbf{P_i}$ seconds.

We think Emacs++ will be simple for power users, but we still need to understand how efficient it is. We have a single Lisp++ program and list of **Q** queries about that program; each query consists of a start position $\mathbf{S_j}$ and an end position $\mathbf{E_j}$. To answer the j-th query, you must determine the smallest possible amount of time N_j (in seconds) that it will take to take the cursor from position $\mathbf{S_j}$ to position $\mathbf{E_j}$, if you make optimal decisions.

Please output the sum of all of those N_j values.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of a test case contains two integers **K**, which is the length of the Lisp++ program, and **Q**, which is the number of queries.

The second line of a test case contains a string **P** of **K** characters, each of which is either `(` or `)`, representing a Lisp++ program (string of balanced parentheses), as described above.

The third, fourth, and fifth lines of a test case each contain **K** integers. The i-th integers in these lines are the values $\mathbf{L_i}$, $\mathbf{R_i}$, and $\mathbf{P_i}$, respectively, that are described above.

The sixth and seventh lines of a test case each contain **Q** integers. The j-th integers in these lines are $\mathbf{S_j}$ and $\mathbf{E_j}$, respectively, that are described above.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the sum of the N_j values that are described above.

Limits

Time limit: 60 seconds per test set.
Memory limit: 1GB.
$1 \le \mathbf{T} \le 100$.
$\mathbf{K} = 10^5$ and $\mathbf{Q} = 10^5$, for at most 9 test cases.
$2 \le \mathbf{K} \le 1000$ and $1 \le \mathbf{Q} \le 1000$, in all other cases.
length of **P** = **K P** is a string of balanced parentheses, as described above.
$1 \le \mathbf{S_j} \le \mathbf{K}$, for all j.
$1 \le \mathbf{E_j} \le \mathbf{K}$, for all j.

Test Set 1 (Visible Verdict)

$\mathbf{L_i} = 1$, for all i.
$\mathbf{R_i} = 1$, for all i.
$\mathbf{P_i} = 1$, for all i.

Test Set 2 (Hidden Verdict)

$1 \le \mathbf{L_i} \le 10^6$, for all i.
$1 \le \mathbf{R_i} \le 10^6$, for all i.
$1 \le \mathbf{P_i} \le 10^6$, for all i.

Sample

Sample Input

```
1
12 5
(()((((()))))
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
7 4 4 12 5
12 11 10 1 6
```

Sample Output

```
Case #1: 10
```

In the sample case, which obeys the limits for Test Set 1, all of the time costs are the same (1 second per move).
The shortest times for the queries are as follows:

1. Move right from 7 five times to 12 taking 5 seconds.
2. Teleport from 4 to 11 taking 1 second.
3. Teleport from 4 to 11, then move left to 10 taking 2 seconds.

4. Teleport from 12 to 1, taking 1 second.
5. Move right from 5 to 6 taking 1 second.

Thus, the sum of query times is 5+1+2+1+1 = 10 seconds.