C. Maximum Subarray Sum

time limit per test: 2 seconds
memory limit per test: 256 megabytes

You are given an array $a_1, a_2, \ldots, a_n$ of length n and a positive integer k, but some parts of the array a are missing. Your task is to fill the missing part so that the **maximum subarray sum*** of a is exactly k, or report that no solution exists.

Formally, you are given a binary string s and a partially filled array a, where:

- If you remember the value of a_i, $s_i = 1$ to indicate that, and you are given the real value of a_i.
- If you don't remember the value of a_i, $s_i = 0$ to indicate that, and you are given $a_i = 0$.

All the values that you remember satisfy $|a_i| \le 10^6$. However, you may use values up to 10^{18} to fill in the values that you do not remember. It can be proven that if a solution exists, a solution also exists satisfying $|a_i| \le 10^{18}$.

*The **maximum subarray sum** of an array a of length n, i.e. $a_1, a_2, \ldots a_n$ is defined as $\max_{1 \le i \le j \le n} S(i, j)$ where $S(i, j) = a_i + a_{i+1} + \ldots + a_j$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains two numbers n, k ($1 \le n \le 2 \cdot 10^5, 1 \le k \le 10^{12}$).

The second line of each test case contains a binary (`01`) string s of length n.

The third line of each test case contains n numbers $a_1, a_2, \ldots, a_n$ ($|a_i| \le 10^6$). If $s_i =$ 0, then it's guaranteed that $a_i = 0$.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, first output `Yes` if a solution exists or `No` if no solution exists. You may print each character in either case, for example `YES` and `yEs` will also be accepted.

If there's at least one solution, print n numbers $a_1, a_2, \ldots, a_n$ on the second line. $|a_i| \le 10^{18}$ must hold.

Example

input Copy

```
10
3 5
011
0 0 1
5 6
11011
4 -3 0 -2 1
4 4
0011
0 0 -4 -5
6 12
110111
1 2 0 5 -1 9
5 19
00000
0 0 0 0 0
5 19
11001
-8 6 0 0 -5
5 10
10101
10 0 10 0 10
1 1
1
0
```

Codeforces Round 1023 (Div. 2)

Finished

Practice

→ Virtual participation

Virtual contest is a way to take part in past contest, as close as possible to participation on time. It is supported only ICPC mode for virtual contests. If you've seen these problems, a virtual contest is not for you - solve these problems in the archive. If you just want to solve some problem from a contest, a virtual contest is not for you - solve this problem in the archive. Never use someone else's code, read the tutorials or communicate with other person during a virtual contest.

Start virtual contest

→ Clone Contest to Mashup

You can clone this contest to a mashup.

Clone Contest

→ Submit?

Language: GNU G++23 14.2 (64 bit, ms

Choose file: Choose File No file chosen

Submit

→ Last submissions

Submission	Time	Verdict
321867303	May/29/2025 08:34	Accepted
318524482	May/05/2025 19:39	Wrong answer on pretest 2
318520566	May/05/2025 19:27	Wrong answer on pretest 2
318519653	May/05/2025 19:25	Wrong answer on pretest 2
318518688	May/05/2025 19:22	Wrong answer on pretest 2
318515714	May/05/2025 19:13	Wrong answer on pretest 1
318483655	May/05/2025 18:09	Wrong answer on pretest 1

→ Problem tags

binary search constructive algorithms dp implementation math *1500

No tag edit access

→ Contest materials

- Announcement (en)

```
3 5
111
3 -1 3
4 5
1011
-2 0 1 -5
```

output Copy

```
Yes
4 0 1
Yes
4 -3 5 -2 1
Yes
2 2 -4 -5
No
Yes
5 1 9 2 2
Yes
-8 6 6 7 -5
Yes
10 -20 10 -20 10
No
Yes
3 -1 3
Yes
-2 4 1 -5
```

Note

In test case 1, only the first position is not filled. We can fill it with 4 to get the array $[4, 0, 1]$ which has maximum subarray sum of 5.

In test case 2, only the third position is not filled. We can fill it with 5 to get the array $[4, -3, 5, -2, 1]$. Here the maximum subarray sum comes from the subarray $[4, -3, 5]$ and it is 6, as required.

In test case 3, the first and second positions are unfilled. We can fill both with 2 to get the array $[2, 2, -4, -5]$ which has a maximum subarray sum of 4. Note that other outputs are also possible such as $[0, 4, -4, -5]$.

In test case 4, it is impossible to get a valid array. For example, if we filled the third position with 0, we get $[1, 2, 0, 5, -1, 9]$, but this has a maximum subarray sum of 16, not 12 as required.

- Tutorial (en)