## Basic coding/usaco

### Wocs and Cows

Problem Statement:

Help, the wocs have invaded Farmer John's farm! Farmer John has N cows (0 <= N <= 10^7) standing in a line. However, some cows have been replaced by the malicious wocs. Thankfully, Farmer John has a secret weapon to defend against the wocs: The Cowinator. In one shot, the Cowinator can convert any number consecutive wocs back into cows. However, firing a shot can be very expensive. Farmer John wants to know how many shots the Cowinator would need to convert all wocs back into cows, without hitting any of his precious cows.

Input Format (Stdin):

Line 1: N

Line 2: N digits, each being either 0 or 1, with 1 representing a cow and 0 representing a woc

Example:

14

10001101001000

Output Format (Stdout):

Line 1: an integer representing the number of shots necessary

For every problem, be sure to include a newline at the end. For python, this is accomplished by simply using print. For java, this is accomplished by using println instead of print. For c++, this is accomplished by using std::endl. Otherwise, “\n” can also be used in all languages to add a newline to the end.

Example:

4

Explanation:

Shot 1: 10001101001000 -> 11111101001000

Shot 2: 11111101001000 -> 11111101001111

Shot 3: 11111101001111 -> 11111111001111

Shot 4: 11111111001111 -> 11111111111111

* Grader created

Flag: \_\_

### Wocs and Cows 2

Problem Statement:

In an alternate dimension, the nhojs have invaded Farmer Bessie's johns. Farmer Bessie has N johns (1 <= N <= 10^6) standing in a line. However, some johns have been replaced by the malicious nhojs. Thankfully, Farmer Bessie has a secret weapon to defend against the wocs: The Johninator. In one shot, the Johninator can convert any number consecutive nhojs back into johns, and due to alternate dimensional technology, all johns hit will be completely safe! However, the Johninator also costs 1 apple for every john it hits. Additionally, Bessie only has a total of M (1 <= M <= 50) shots left! Farmer Bessie wants to know the minimum amount of apples the Johninator would need to convert all nhojs back into johns, given only M shots.

Input Format (Stdin):

Line 1: N M

Line 2: N digits, each being either 0 or 1, with 1 representing a john and 0 representing a nhoj

Example:

19 4

0011100010110011100

Output Format (Stdout):

Line 1: an integer representing the number of apples necessary

For every problem, be sure to include a newline at the end. For python, this is accomplished by simply using print. For java, this is accomplished by using println instead of print. For c++, this is accomplished by using std::endl. Otherwise, “\n” can also be used in all languages to add a newline to the end.

Example:

1

Explanation:

Shot 1: 0011100010110011100 -> 1111100010110011100, 0 apples

Shot 2: 1111100010110011100 -> 1111100010111111100, 0 apples

Shot 3: 1111100010111111100 -> 1111100010111111111, 0 apples

Shot 4: 1111100010111111111 -> 1111111111111111111, 1 apple

* Grader created

Flag: \_\_

### Wocs and Cows 3

Problem Statement:

After the first woc invasion, Farmer John has expanded his farm, and his technology! Farmer John now has N x M (1 <= N, M <= 1000) cows, standing in a N x M grid! Unfortunately, the wocs have invaded again, replacing some of his cows with wocs. Farmer John's new weapon, the latest Cowinator 2.0, can convert a square of any size of wocs back into cows in one shot. However, Farmer John only has 1 shot of the Cowinator 2.0. Farmer John wants to know how many wocs he can convert back into cows in 1 shot, without hitting any cows.

Input Format (Stdin):

Line 1: N M

Line 2..N+1: M digits, each being either 0 or 1, with 1 representing a cow and 0 representing a woc

Example:

5 5

11111

10011

10011

11011

11101

Output Format (Stdout):

Line 1: an integer representing the number of wocs hit

For every problem, be sure to include a newline at the end. For python, this is accomplished by simply using print. For java, this is accomplished by using println instead of print. For c++, this is accomplished by using std::endl. Otherwise, “\n” can also be used in all languages to add a newline to the end.

Example:

4

Explanation:

11111

10011

10011

11011

11101

* Grader created

Flag: \_\_

### Missing Goldfish

Problem Statement:

Bessie lost her pet goldfish in her school! Her school is arranged in a series of classrooms and hallways, with N classrooms (2 <= N <= 10^6) numbered 0 - N-1, and M hallways (1 <= M <= 10^6), each with a length of 10 meters. Hallways connect two classrooms in any direction. Bessie knows the arrangement of classrooms and hallways in her school, and the room where she lost her goldfish. However, always being a perfectionist, she wants to compute the minimum distance that she will have to travel to reach her goldfish, starting from classroom 0.

Input Format (Stdin):

Line 1: N, M

Line 2: G (1 <= G <= N), representing the number of the classroom with the goldfish

Line 3..M+2: 2 numbers, A and B (1 <= A < B <= N), with A and B representing two classrooms directly connected by a hallway.

Example:

5 5

4

0 1

1 2

2 3

3 4

1 4

Output Format (Stdout):

Line 1: an integer representing the minimum distance

For every problem, be sure to include a newline at the end. For python, this is accomplished by simply using print. For java, this is accomplished by using println instead of print. For c++, this is accomplished by using std::endl. Otherwise, “\n” can also be used in all languages to add a newline to the end.

Example:

20

Explanation:

Step 1: 0 -> 4, 10 total meters

Step 2: 1 -> 4, 20 total meters

* Grader created

Flag: \_\_

### Magical Cakes

Problem Statement:

Bessie the cow has found a field of C (0 < C < 10^8) magical cakes! The cakes are randomly distributed with a random original width Cw at the position Ci on a 1 dimensional number line bounded by (0 <= Ci <= 10^12), where Ci represents the left coordinate of the cake (inclusive) and Ci + Cw represents the right coordinate of the cake (exclusive) . Bessie has also been learning to sing recently, but she’s very shy about her singing voice. Each time she sings, the magical cakes, being magical cakes, grow towards positive x by a factor of their original width.

In other words, each cake starts with a width Cw at point Ci. If bessie sings once, then each cake now has width 2 \* Cw starting at point Ci. If bessie sings twice, then each cake has width 3 \* Cw. If bessie sings n times, then each cake has width (n-1) \* Cw.

Bessie wants the total lengths of all the cakes to be at least N, but when a magical cake grows on top of another magical cake and they overlap, that is only treated as one cake segment, not two. However, Bessie doesn’t like to sing. How many times does bessie need to sing to ensure there is enough cake?

Input Format (Stdin):

Line 1: C, N

Line 2..C+1: 2 numbers, Ci and Cw respectively, separated by spaces.

Example:

4 30

1 3

7 1

17 3

24 4

Output Format (Stdout):

Line 1: an integer representing the minimum number of times bessie needs to sing

For every problem, be sure to include a newline at the end. For python, this is accomplished by simply using print. For java, this is accomplished by using println instead of print. For c++, this is accomplished by using std::endl. Otherwise, “\n” can also be used in all languages to add a newline to the end.

Example:

3

Explanation:

In the following example, each of the cakes are labeled 1, 2, 3, and 4. A cake is denoted in the example with its label on its starting position, and with @ symbols to represent the spaces that the cake takes up.

0123456789012345678901234567890123456789012345678901234567890

\_1@@\_\_\_2\_\_\_\_\_\_\_\_\_3@@\_\_\_\_4@@@\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0 Songs, total cakes = 11

\_1@@@@@2@\_\_\_\_\_\_\_\_3@@@@@\_4@@@@@@@\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 Songs, total cakes = 22

\_1@@@@@2@@\_\_\_\_\_\_\_3@@@@@@4@@@@@@@@@@@\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 Songs, total cakes = 28

\_1@@@@@2@@@@@\_\_\_\_3@@@@@@4@@@@@@@@@@@@@@@\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 Songs, total cakes = 35

Take note that cake 1 eventually overlaps with cake 2 and cake 3 overlaps with cake 4. However, while cake 1 eventually outspeeds cake 2 and overtakes it, cake 3 is overshadowed by cake 4 as it grows slower.

* Grader created

Flag: \_\_