Constructive Problems

Jason Zhang

Introduction

What are Constructive Problems?

Constructive Problems are a general class of competitive programming problems that involve actually making some object that satisfies some given properties and constraints.

The objects in question could include graphs and arrays, but it could be as simple as just a number.

The issue with constructive problems is that there isn't one good algorithm to solve them. In fact, there often isn't any named algorithm. You have to be creative and think on your feet.

Simple Example and Non-Example

Here are two simple problems, one of which is constructive and one of which isn't.

Problem (Constructive).
Given n, find a sorted array of length n.

Problem (Non-Constructive).
There are two players Alice and Bob. Given the amount of goals scored and the amount of points each goal was worth for each player, find out who won.

Notice that in the first problem, we are asked to make something. In the second problem, we simply have to determine who won without having to make anything.

*It will not always be this clear cut! Sometimes, you will be asked to find a single value but in order to do so you must construct something, even if the problem does not explicitly say that.

Motivating Problem

Problem

The problem we will be examining is Codeforces Round 1003 (Div. 4) E. Skibidus and Rizz.

Problem.

Given a binary string *t*, let *x* be the number of 0s and *y* be the number of 1s. Let the **balance-value** of *t* be max(x - y, y - x).

You will receive numbers *n*, *m*, *k*. You need to create a binary string with exactly *n* 0s and *m* 1s such that the maximum balance-value among all the substrings is exactly *k*. Output -1 if it isn't possible.

Note that ($0 \le n, m \le 2 \cdot 10^5$; $1 \le k \le n + m$; $1 \le n + m$).

Statement Analysis

The **balance-value** is the difference between the number of 0s and 1s. So this problem basically wants an $n+m$ length binary string where two constraints are satisfied:

- There is a substring where the difference between the number of 0s and 1s is exactly k .
- There is no substring where the difference between the number of 0s and 1s is more than k.

General Approaches

Quick Disclaimer

Disclaimer

Constructive problems often require a lot of mathematical intuition. If you have good mathematical intuition, then constructive problems may come naturally to you. If you don't, I will offer some strategies to mitigate this disparity.

Generalized Steps

Here is how you would generally approach any constructive problem:

1. Solve a few test cases by hand.
2. Find some patterns/“templates” that satisfy the construction.
3. Implement/create your solution.

Step 1

The first step is to solve some test cases by hand. You should look at the sample test cases, but also make your own. Also, try as hard as you can to make edge cases.

If you have the time and patience, you can create a program to run through every possible case and check if it satisfies the construction. This will give you a correct answer (or all of them). This can be really useful if it is hard or error-prone to do the cases by hand, but the downside is that you have to code this up and it will be really slow.

Step 2

This is the part the requires a bit of intuition. Hopefully, step 1 helped with this somewhat. You need to find some sort of template that will always satisfy the construction.

There may be some constructions that don't fit this template, but that's fine as long as your template version does the same job.

Additionally, you may have to make multiple templates/patterns depending on what the input case is (for example, one pattern for when a certain number is even and another for when it is odd). This is fine unless you find yourself creating 20 different templates, in which case it may be time to reconsider.

Step 3

At this point, you should be equipped with everything needed to solve the problem. This step is simply coding everything up and making things fit the template, which may involve some of the algorithms you learned previously.

Application to Simple Example

Instead of giving you a wall of text on how to use this, we can apply these steps to a simple example.

Problem.
Given n, find a sorted array of length n.

Step 1 - Solve a few test cases by hand

1. n = 2
    a. [1,2]
    b. [4,6]
    c. [1,10000]
2. n = 3
    a. [1,5,7]
    b. [1,2,3]
    c. [2,3,4]
    d. [4,5,99]
3. n = 1
    a. [1]
    b. [2]
4. n = 0 <- edge case
    a. []

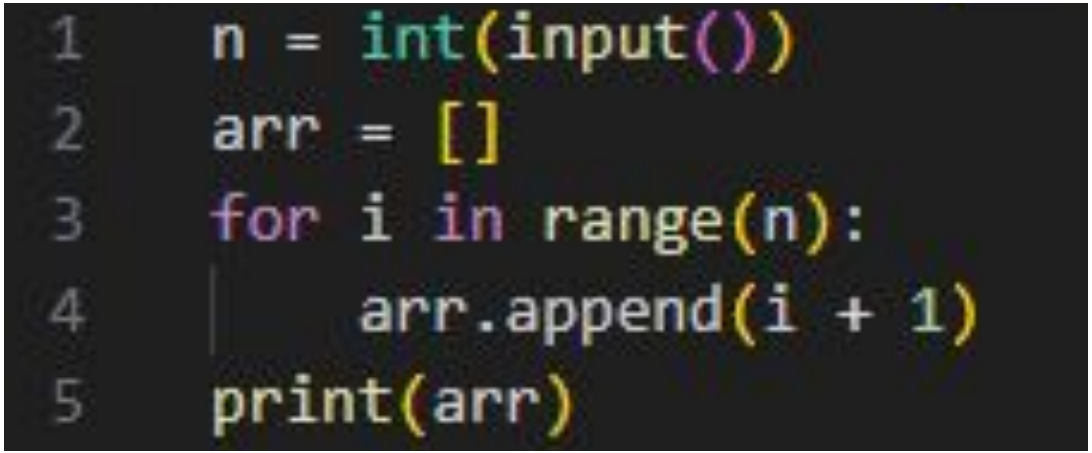
Step 2 - Find a pattern

Notice that [1,...,n] works. Notice that not every sorted array is of the form [1,...,n], but every *n* has a sorted array of the form [1,...,n]. So it is sufficient. This is our pattern.

(if n = 0 then just []).

Step 3 - Implement

Here, we simply code it up.



Solving the Motivating Problem

Reminder

If you need a reminder on the problem:

Problem.

Given a binary string *t*, let *x* be the number of 0s and *y* be the number of 1s. Let the **balance-value** of *t* be max(x - y, y - x).

You will receive numbers *n*, *m*, *k*. You need to create a binary string with exactly *n* 0s and *m* 1s such that the maximum balance-value among all the substrings is exactly *k*. Output -1 if it isn't possible.

Note that ($0 \le n, m \le 2 \cdot 10^5$; $1 \le k \le n + m$; $1 \le n + m$).

Step 1 - Test Cases

We could create a program to generate and solve naively test cases for us, but that would take way too long and it would be slow (naive brute-force would be around $O(2^{(n+m)}(n+m+1)(n+m)^2)$). So we will do it by hand.

1. n = 1, m = 2, k = 1
2. n = 2, m = 1, k = 1
3. n = 4, m = 3, k = 2
4. n = 8, m = 3, k = 2
5. n = 5, m = 0, k = 5
6. n = 5, m = 1, k = 6
7. n = 2, m = 4, k = 3

Solutions

1. 101
2. 010
3. 0100101, 0010101
4. -1
5. 00000
6. -1
7. 111010, 011011

Step 2

Insight 1.
The choice of 1 and 0 doesn't really matter. They could be any character, or even swapped. Basically, the answer isn't really different if n and m swapped places. So, assume n ≥ m (there are more 0s than 1s or they are equal). If there were less 0s than 1s, we could change all n's to m's and 0's to 1's and vice-versa.

The solution to n = 1, m = 2, k = 1 vs. n = 2, m = 1, k = 1 is the same but the 0s and 1s flipped.

101 vs. 010

Step 2

Insight 2.
Let us try to find the -1 cases. These are the cases where NO possible construction works. If the balance-value of the entire string is > *k*, then it would be impossible to find a valid construction. This occurs when n - m > k. If k > n, then it is also hopeless since at best our balance-value would be n.

n = 5, m = 1, k = 3 : -1 because the balance-value of the entire string no matter what is 4 which is greater than *k*.

n = 5, m = 1, k = 6 : -1 because at best our balance value is 5.

*These are the two trivial cases that do not have a valid construction. As of right now, we don't know if there are any more. We should move on and try to solve the problem for the rest of the cases. If we find another case that doesn't work later on, we can revisit this.

Step 2

Pattern.

Look at how we solved our test cases. For the possible ones with large numbers, a pattern seems noticeable. There are a lot of 10s and 01s. The balance-value of these two are 0 and at most 1. Additionally, the balance value of a string with just 0s or just 1s is the length of that string. So, what if we do (assuming $n \geq m$):

(k 0s) (a bunch of 10s) (leftover 1s, hopefully less than k).

Which indeed would have a maximum balance value of *k*.

Prove it!

If you are constrained on time and fairly confident, you don't need to prove it. If you are sure in your intuition, then start implementing. But, let's be safe and prove this template works.

Let's formalize our template a bit more. We want "a bunch of 10s", specifically enough to exhaust the remaining 0s left. Therefore, we need n - k 10s. At this point, we must ask how many 1s are left? Well, it is exactly m - (n - k) = m - n + k. Our formalized template is below:

(k 0s)(n - k 10s)(m - n + k 1s)

*Note that some of these segments may be length 0, which is still completely fine! This covers cases like 00000.

Proof.

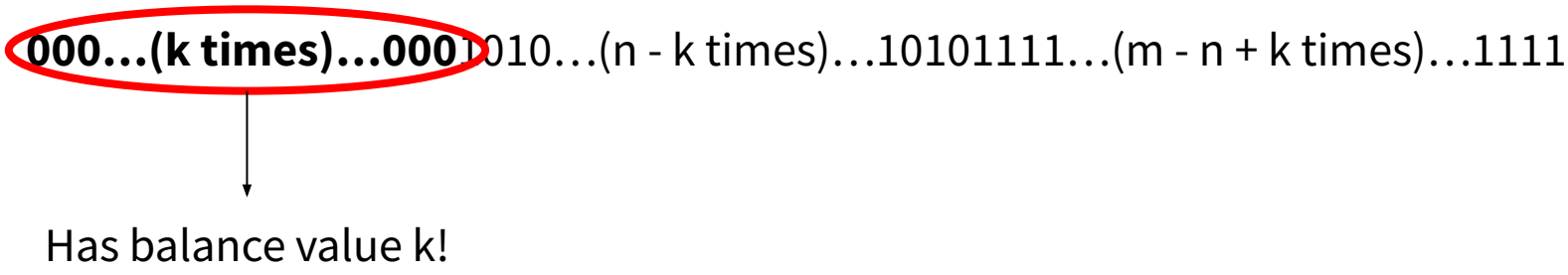
We will show two things to be true of our template:

- There is a substring where the difference between the number of 0s and 1s is exactly k .
- There is no substring where the difference between the number of 0s and 1s is more than k.

Because of our previous analysis, we know this to be equivalent to the problem statement.

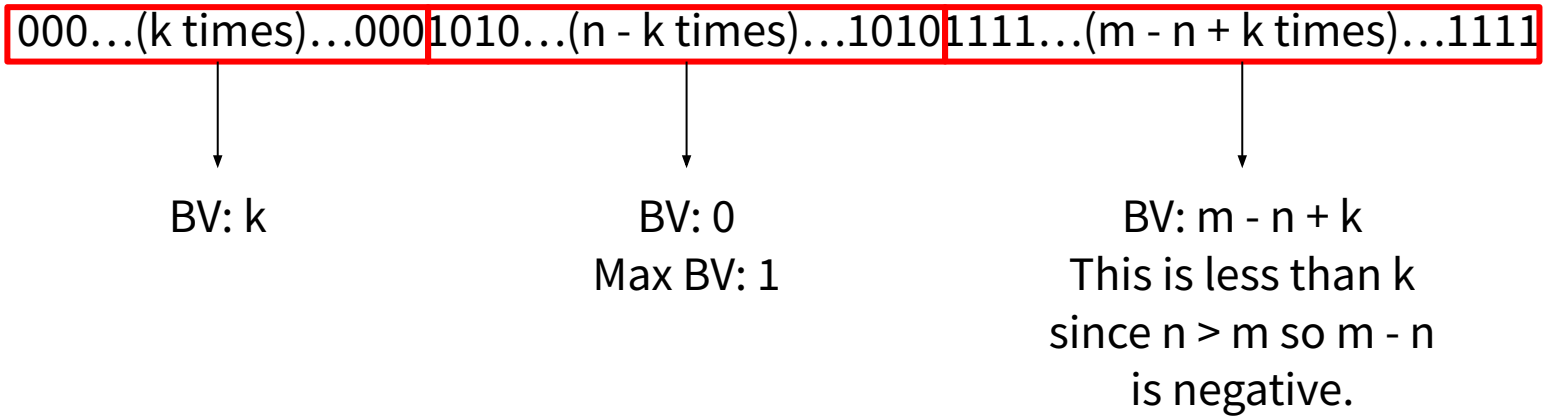
Proof.

Our first segment is k 0s, which trivially shows that the balance value of that segment is k. This satisfies the first condition.



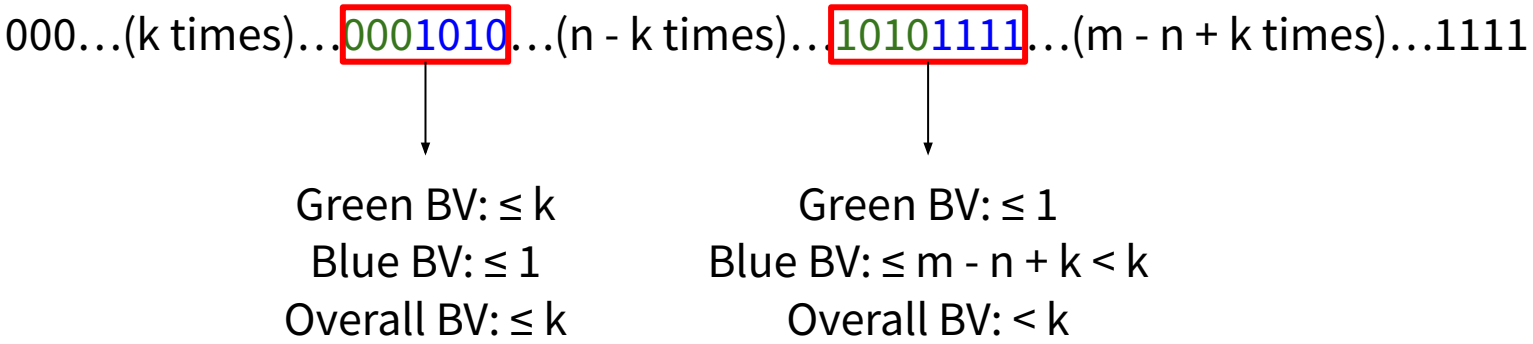
Proof.

Let us dissect our string to see if the second condition holds.



Proof.

Here are some more cases:



Since the middle section contributes 0, the last case which spans all three sections is equivalent to (k 0s) (m - n + k 1s) which has balance value ≤ n - m.

Proof.

A few more things to note before we wrap up this proof:

1. If n < m, then we swap the 0 and 1 in every place in the proof. Our template would be (k 1s) (m - k 01s) (n - m + k 0s).
2. If k > n, then we wouldn't be able to form our (k 0s) segment which means we can't make our template and thus it is impossible.
3. Similarly, if n - m > k then our last segment of m - n + k 1s would need negative 1s, impossible.
4. There is no other impossible case since the construction works.

This completes the proof. □

Step 3 - Implementation

Let us assign p = max(n, m) and q = min(n, m). Let us also assign p_sym to be the symbol associated with p and q_sym to be the symbol associated with q. That is, if n ≥ m then p_sym = “0” and q_sym = “1”, while if n < m then p_sym = “1” and q_sym = “0”.

Now, we check if either k > p or p - q > k. If so, -1.

Otherwise, output p_sym * k + (q_sym + p_sym) * (p - k) + q_sym * (q - p + k).

*If I write “1” * 2 that means ‘The string “1” 2 times’, or just “11”.

Step 3 - Implementation

In the original problem, there were *t* test cases to solve.

```python
t = int(input()) # number of test cases
for _ in range(t):
    n, m, k = [int(i) for i in input().split()]
    p = max(n, m)
    q = min(n, m)
    p_sym = "0"
    q_sym = "1"
    if n < m:
        p_sym = "1"
        q_sym = "0"
    if k > p or p - q > k:
        print(-1)
        continue
    print(p_sym * k + (q_sym + p_sym) * (p - k) + q_sym * (q - p + k))
```

E - Skibidus and Rizz	PyPy 3-64	Accepted	187 ms	6100 KB