Report: Flipping Game

CSX 3009 Algorithm Design Term Project

Name: Tanat Arora

ID: 6410381

Problem Statement

In the "Flipping Game," we are given a sequence of integers, where each integer can be either 0 or 1. The objective is to make exactly one move by selecting two indices ($1 \leq i \leq j \leq n$) and flipping all values in that range. The goal is to maximize the number of ones in the sequence after this single move.

Input

- The first line of the input contains an integer **n** ($1 \leq n \leq 100$), representing the number of elements in the sequence.
- The second line of the input contains **n** integers: a1, a2, ..., an. Each of these values is either 0 or 1.

Output

- Print an integer, the maximal number of 1s that can be obtained after exactly one move.

Analysis

To solve this problem, we employ a straightforward algorithm:

1. **Input Reading:**
    - We read the integer **n**, which signifies the number of elements in the sequence.
    - We read the sequence of 0s and 1s and store them in a list.

2. **Initialization:**
    - We initialize three variables:
        - **one**: To count the number of 1s in the sequence.
        - **cumSum**: To keep track of the cumulative sum.
        - **flip**: To store the maximum number of ones after one move.
3. **Counting Ones:**
    - We loop through the sequence to count the number of 1s and store the count in the **one** variable.
4. **Iterating Through the Sequence:**
    - We loop through the sequence again.
    - For each element, we check if it's a 0 or a 1.
    - If it's a 0, we increment **cumSum**. If it's a 1, we decrement **cumSum**. This helps track the cumulative sum of elements.
5. **Updating the Maximum:**
    - After each update to **cumSum**, we update the **flip** variable with the maximum value between the current **flip** and **cumSum**. This ensures that **flip** always stores the maximum cumulative sum encountered so far.
6. **Resetting Cumulative Sum:**
    - If the cumulative sum **cumSum** becomes negative at any point, we reset it to 0. This allows us to start calculating a new cumulative sum from the current position in the sequence.
7. **Checking for Special Case:**
    - We check if the entire sequence consists of 1s. If it does, the best move is to flip all elements to 0s except one, so we print **n-1**.
8. **Final Output:**
    - If the sequence contains both 0s and 1s, we print the result, which is the count of 1s (**one**) plus the maximum cumulative sum (**flip**).

Test Case Explained

As an example, consider the following test case:

Input:

5

1 0 0 1 0

Output:

4

Explanation:

The initial 'one' count is 2 (two 1s in the sequence).

Initially, 'cumSum' is 0.

- Loop 1: i = 0, nums[0] == 1, 'cumSum' remains 0.
- Loop 2: i = 1, nums[1] == 0, 'cumSum' becomes 1.
- Loop 3: i = 2, nums[2] == 0, 'cumSum' becomes 2.
- Loop 4: i = 3, nums[3] == 1, 'cumSum' becomes 1.
- Loop 5: i = 4, nums[4] == 0, 'cumSum' becomes 2.

'flip' is updated to 2, 1, 2, 2, 2.

Final 'flip' is 2.

Since there are not all 1s in the sequence, we print = 2 + 2 = 4.

So, the maximum number of ones after one move is 4.

Special Cases:

For cases like when the input is all 1's. We return one less than the length of the sequence (n), This is done because we are required to flip some number, so even if its all 1's, we flip one of them to 0 and then take the rest as the maximum number of 1's we can get.

Submission

General

#	Author	Problem	Lang	Verdict	Time	Memory	Sent	Judged		
225538006	Practice: Tanat04	327A - 16	PyPy 3-64	Accepted	124 ms	16 KB	2023-09-27 18:34:32	2023-09-27 18:34:32		Compare

→ Source

Copy

```
n = int(input())
nums = list(map(int, input().split()))

one = 0
cumSum = 0
flip = 0

for i in range(n):
    if nums[i] == 1:
        one += 1
for i in range(n):

    if nums[i] == 0:
        cumSum += 1
    else:
        cumSum -= 1

    flip = max(flip, cumSum)

    if cumSum < 0:
        cumSum = 0
if one == n:
    print(n - 1)
else:
    print(one + flip)
```

Click to see test details

References

- Problem Statement on Codeforces
- Problem Solution Reference