



Competitive Programming DeCal

Lecture #2 on 1/30/23 - Prefix Sums





Announcements

- Problemset 1 extended to be due next week **2/6 by 6:30pm**
- Additional resources!
 - We will now provide code for the topics covered in each lecture
 - For example: prefix sum implementation in Python and C++
- Office hours:
 - Wednesday 3:30pm to 4:30pm with **Youngmin Park** on Discord



Announcements

- Reminder:
 - 2 weekly problemset drops
 - 2 unexcused absences
- Grading Changes
 - **If you attend lecture, your weekly problemset grade will be based on effort**
 - **Otherwise the problemset will be based on correctness**
- Only the **first two problems** on a problemset are **required**



Navigating Codeforces

- Failed test cases
- Seeing others' submissions
- Any questions on using Codeforces?



Solution to Last Week's Problem A

- In Python, `input()` reads in a **whole line of input**!
- Use `input().split()` to get a list of each element on the line
- The list will contain strings!
 - Convert them to integers and add them



Motivation and Brute Force Solution

$O(N^2)$ isn't thaaaaat baaad... right?

1. Motivation and Brute Force Solution
2. Prefix Sum Arrays
3. Max Subarray Sum
4. Bonus and Variations



Motivation

Motivating Problem: Given an array of N integers, efficiently find the sum of the subarray between index i and j .

A **subarray** is a contiguous chunk of an array.

For example, if the array is $[1, 2, 5, 4, 6]$:

- $[2, 5, 4]$ is a subarray because $[1, \mathbf{2}, \mathbf{5}, \mathbf{4}, 6]$ ($i = 2, j = 4$)
- $[4, 6]$ is a subarray because $[1, 2, 5, \mathbf{4}, \mathbf{6}]$ ($i = 4, j = 5$)
- $[2, 4]$ isn't a subarray because in $[1, \mathbf{2}, 5, \mathbf{4}, 6]$, the 2 and the 4 aren't contiguous
- $[5, 2]$ isn't a subarray because the order is wrong



Brute Force Solution

We can loop over all indices of the subarray, and sum them up.

What's the time complexity of this solution?



Brute Force Solution

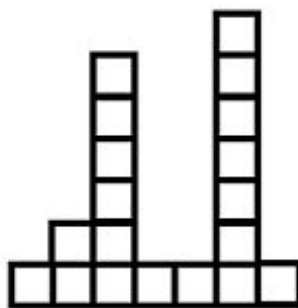
We can loop over all indices of the subarray, and sum them up.

This solution is $O(N)$, because in a worst-case scenario, the subarray would be the entire array.

If we only had a single subarray sum that we want to compute, this is usually fast enough for competitive programming problems. But what if we are asked to find the sum of **Q different subarrays**? Then, the time complexity becomes $O(N * Q)$.

Can we do better?

There is a fence in front of Polycarpus's home. The fence consists of n planks of the same width which go one after another from left to right. The height of the i -th plank is h_i meters, distinct planks can have distinct heights.



Fence for $n = 7$ and $h = [1, 2, 6, 1, 1, 7, 1]$

Polycarpus has bought a posh piano and is thinking about how to get it into the house. In order to carry out his plan, he needs to take exactly k consecutive planks from the fence. Higher planks are harder to tear off the fence, so Polycarpus wants to find such k consecutive planks that the sum of their heights is minimal possible.

Write the program that finds the indexes of k consecutive planks with minimal total height. Pay attention, the fence is not around Polycarpus's home, it is in front of home (in other words, the fence isn't cyclic).

Input

The first line of the input contains integers n and k ($1 \leq n \leq 1.5 \cdot 10^5$, $1 \leq k \leq n$) — the number of planks in the fence and the width of the hole for the piano. The second line contains the sequence of integers h_1, h_2, \dots, h_n ($1 \leq h_i \leq 100$), where h_i is the height of the i -th plank of the fence.

Output

Print such integer j that the sum of the heights of planks $j, j + 1, \dots, j + k - 1$ is the minimum possible. If there are multiple such j 's, print any of them.

input

```
7 3  
1 2 6 1 1 7 1
```

output

```
3
```



Prefix Sum Arrays

1. Motivation and Brute Force Solution
2. **Prefix Sum Arrays**
3. Max Subarray Sum
4. Bonus and Variations

`["P", "Pr", "Pre", "Pref", "Prefi", "Prefix"]`



What is a Prefix Sum Array?

Given an array A of integers, the each element in the prefix sum array P contains the sum of all elements from the beginning up to that element in A . In other words,

$$P[i] = \sum_{k=0}^i A[k]$$

You can also think of this as a cumulative sum or a running total!

For example, if we had:

$$A = [3, 1, 4, 1, 5, 9, 2, 6]$$

Then the corresponding prefix sum array would be:

$$P = [3, 4, 8, 9, 14, 23, 25, 31]$$



Easy Construction

Super easy! Just iterate through the elements of A while maintaining a running total of their sum, and update every element of P as you go through!

```
def make_prefix_sum(A):  
    N = len(A)  
    P = array(N)  
    sum = 0  
    for i in (1, N):  
        sum += A[i]  
        P[i] = sum  
    return P
```



Efficient Subarray Sum Queries

Try applying prefix sums to solve subarray sum queries efficiently on your own!

Given an array A of N integers, and a sequence of Q queries in the form of $[(s_0, e_0), (s_1, e_1), (s_2, e_2), \dots]$ each asking for the sum of values in $A[s:e]$, find all sums in faster than $O(NQ)$ time.

Example input:

$$A = [3, 1, 4, 1, 5, 9, 2, 6] \quad Q = [(2, 4), (1, 7), (0, 5), (6, 6)]$$

Example output:

$$[10, 28, 23, 2]$$

Prefix sum provided for your convenience:

$$P = [3, 4, 8, 9, 14, 23, 25, 31]$$



Efficient Subarray Sum Queries (cont.)

Since each query in the form of (s, e) is really asking us for:

$$Q(s, e) = \sum_{k=s}^e A[k]$$

But we know that:

$$P[i] = \sum_{k=0}^i A[k]$$

We can simply take the difference between $P[e]$ and $P[s - 1]$! This is because $P[e]$ over-sums the first s elements, but we can subtract that sum off with $P[s - 1]$:

$$Q(s, e) = P[e] - P[s - 1] = \sum_{k=0}^e A[k] - \sum_{k=0}^{s-1} A[k] = \sum_{k=s}^e A[k]$$



Efficient Subarray Sum Queries (cont.)

Since each query in the form of (s, e) is really asking us for:

$$A = [3, 1, 4, \underline{1, 5, 9}, 2, 6]$$

$$Q(s, e) = \sum_{k=s}^e A[k]$$

$$P = [3, 4, 8, 9, 14, 23, 25, 31]$$

But we know that:

$$P[i] = \sum_{k=0}^i A[k]$$

We can simply take the difference between $P[e]$ and $P[s - 1]$! This is because $P[e]$ over-sums the first s elements, but we can subtract that sum off with $P[s - 1]$:

$$Q(s, e) = P[e] - P[s - 1] = \sum_{k=0}^e A[k] - \sum_{k=0}^{s-1} A[k] = \sum_{k=s}^e A[k]$$



Efficient Subarray Sum Queries (cont.)

Since each query now only involves retrieving two numbers and subtracting them, each query can be solved in $O(1)$ time after prefix sum array is constructed. With Q queries, we have $O(Q)$

Constructing prefix sum array is $O(N)$.

This gives us $O(N + Q)$ overall!

```
def subarray_sum_queries (A, Q):  
    P = make_prefix_sum (A)  
    for (s, e) in Q:  
        print (P[e] - P[s - 1])
```

Ilya the Lion wants to help all his friends with passing exams. They need to solve the following problem to pass the IT exam.

You've got string $s = s_1s_2...s_n$ (n is the length of the string), consisting only of characters "." and "#" and m queries. Each query is described by a pair of integers l_i, r_i ($1 \leq l_i < r_i \leq n$). The answer to the query l_i, r_i is the number of such integers i ($l_i \leq i < r_i$), that $s_i = s_{i+1}$.

Ilya the Lion wants to help his friends but is there anyone to help him? Help Ilya, solve the problem.

Input

The first line contains string s of length n ($2 \leq n \leq 10^5$). It is guaranteed that the given string only consists of characters "." and "#".

The next line contains integer m ($1 \leq m \leq 10^5$) — the number of queries. Each of the next m lines contains the description of the corresponding query. The i -th line contains integers l_i, r_i ($1 \leq l_i < r_i \leq n$).

Output

input

Copy

```
#..###  
5  
1 3  
5 6  
1 5  
3 6  
3 4
```

output

Copy

```
1  
1  
2  
2  
0
```



Solution

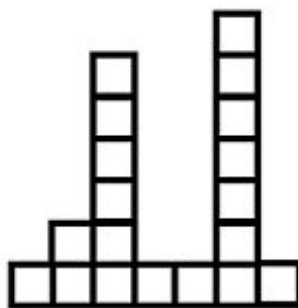
- Make a prefix sum array on the count of matching neighbors
- Spend $O(n)$ time creating the prefix sum array
- Each of the m queries can now be solved in $O(1)$ time!
- Total time complexity of $O(n+m)$



Other uses of prefix sum?

- Sometimes it is not obvious that we need to find a range sum efficiently!

There is a fence in front of Polycarpus's home. The fence consists of n planks of the same width which go one after another from left to right. The height of the i -th plank is h_i meters, distinct planks can have distinct heights.



Fence for $n = 7$ and $h = [1, 2, 6, 1, 1, 7, 1]$

input

7 3

1 2 6 1 1 7 1

output

3

Polycarpus has bought a posh piano and is thinking about how to get it into the house. In order to carry out his plan, he needs to take exactly k consecutive planks from the fence. Higher planks are harder to tear off the fence, so Polycarpus wants to find such k consecutive planks that the sum of their heights is minimal possible.

Write the program that finds the indexes of k consecutive planks with minimal total height. Pay attention, the fence is not around Polycarpus's home, it is in front of home (in other words, the fence isn't cyclic).

Input

The first line of the input contains integers n and k ($1 \leq n \leq 1.5 \cdot 10^5$, $1 \leq k \leq n$) — the number of planks in the fence and the width of the hole for the piano. The second line contains the sequence of integers h_1, h_2, \dots, h_n ($1 \leq h_i \leq 100$), where h_i is the height of the i -th plank of the fence.

Output

Print such integer j that the sum of the heights of planks $j, j + 1, \dots, j + k - 1$ is the minimum possible. If there are multiple such j 's, print any of them.



Solution!

- Make a prefix sum array with the heights of the fences
- There are $n - k + 1$ segments of length k
- We can now calculate each range in $O(1)$
- Check all ranges of length k and take the minimum
- Total time complexity $O(n)$

input

7 3

1 2 6 1 1 7 1

output

3



Max Subarray Sum

1. Motivation and Brute Force Solution
2. Prefix Sum Arrays
3. **Max Subarray Sum**
4. Bonus and Variations



Motivating Problem

Given an array of N integers, find the maximum sum of any subarray in the array.

The array values can be positive or negative.



Brute Force Solution

Iterate over all possible subarrays, and for each one, iterate over every element to calculate its sum.

There are $O(N^2)$ subarrays of the array, and iterating over each one takes $O(N)$ time, so this is very slow (specifically, $O(N^3)$). Can we do better?



Better Solution

Using prefix sums, we can find the sum of each subarray in $O(1)$. This speeds up the algorithm to $O(N^2)$.

This is still a bit slow. Can we do better?




Even Better Solution

If we construct the prefix sum array P , where $P[i]$ is the sum of the prefix of the array ending at index i , every subarray sum can be represented as $P[i] - P[j]$, for some i and j .

Iterate over all possible right endpoints of the subarray ($P[i]$ in the expression above). To make the expression as large as possible, we should maximize $P[i]$ and minimize $P[j]$, so to find the maximum subarray sum **ending at a certain element i** , we should choose j such that $P[j]$ is as low as possible.

If we do this over all indices of the array, we can take the maximum across all right endpoints and find the maximum subarray sum in $O(N)$.



Bonus and Variations

1. Motivation and Brute Force Solution
2. Prefix Sum Arrays
3. Max Subarray Sum
4. **Bonus and Variations**



USACO: Haybale Stacking

<http://www.usaco.org/index.php?page=viewproblem2&cpid=104>



Other Variations of Prefix Sums

- Second/higher order
- Construct original array given a prefix sum array
- Prefix product/min/max/xor instead of sum
- Generalizations to multiple dimensions ("pre-matrix" sum)
- Precomputation for other algorithms (like binary search!)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to count the number of subarrays having sum x .

Input

The first input line has two integers n and x : the size of the array and the target sum x .

The next line has n integers a_1, a_2, \dots, a_n : the contents of the array.

Output

Print one integer: the required number of subarrays.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $-10^9 \leq x, a_i \leq 10^9$

Example

Input:

```
5 7
2 -1 3 5 -2
```

Output:

```
2
```



Bonus Problem!

<https://cses.fi/problemset/task/1661>

Attendance

Please fill out

<https://docs.google.com/forms/d/e/1FAIpQLScUIGtr0Mtm0ZedO9QOeAlu4gQx1P1qSdKd3gaT4wh8I9VKQ/viewform> for attendance

