CodeChef Discussion

Search Here... (●) questions ( ) tags ( ) users

# SUBARR - Editorial

2

## PROBLEM LINKS

Contest
Practice

**Author:** Constantine Sokol
**Tester:** Roman Rubanenko
**Editorialist:** Praveen Reddy Vaka

## DIFFICULTY:

easy

## PREREQUISITES:

Binary Indexed Tree (fenwick tree) / Segment Tree

## HINT

Solve the subtask3 and try to reduce the original problem to this special case.

## EXPLANATION

Let A be the given array (zero indexed).
Let sum(A,i..j) = A[i] + ... + A[j]
Let avg(A,i..j) = sum(A,i..j) / (j-i+1)

**Subtask1**: For each pair of indices i,j (i <=j) check if average of elements between i and j is greater than or equal to K.

```
count=0
for i = 0 to (N-1)
    for j = i to (N-1)
        if avg(i..j) >= K
            set count = count+1
output count as answer
```

This solution runs in O(N^3) and hence solves only the 1st subtask.

**Subtask2**: You can optimize the previous solution to run in O(N^2) by making a very small observation. To compute avg(i..j) you need to compute sum(i..j). You can compute sum(A, i..j) by just adding A[j] to sum(A, 1..(j-1)). Let us look at a refined algorithm using this.

```
count = 0
for i = 0 to (N-1)
    sum=0
    for j = i to (N-1)
        sum = sum+A[j]
        if sum >= K*(j-i+1)
            set count = count + 1
output count as answer
```

**Subtask3,4 and 5**: Subtask3 has an additional condition of K = 0. i.e., we have to find the number of sub-arrays whose average is >= 0. It suffices to finding the number of sub-arrays whose sum >=0. We shall now reduce our problem to this special case.

Create a new array B of length same as A and set B[i] = A[i] - K for all i.

Claim: avg(A,i..j) >= K if and only if sum(B,i..j) >= 0

Proof: First we prove avg(A,i..j) >= K implies sum(B,i..j) >= 0
(A[i] +.... +A[j])/(j-i+1) >= K
A[i] +.... +A[j]) >= K*(j-i+1) = K + ... + K (j-i+1times)
(A[i] - K) + ... + (A[j] - K) >= 0
B[i]+...+B[j] >= 0
We can prove sum(B,i..j) >= 0 implies avg(A,i..j) >= K by just going in the backward direction

Follow this question
**By Email:**
You are not subscribed to this question.

subscribe me

(you can adjust your notification settings on your profile)

**By RSS:**
Answers
Answers and Comments

Question tags:

| | |
|---|---|
| editorial | **×12,795** |
| easy | **×2,765** |
| segment-tree | **×1,291** |
| bit | **×307** |
| fenwick | **×146** |
| prefix-sum | **×64** |
| ltime07 | **×12** |
| subarr | **×9** |

question asked: **29 Dec '13, 15:37**

question was seen: **6,709 times**

last updated: **24 Jun '15, 15:44**

**Related questions**

PPLUCKY - Editorial
SEABAL - Editorial
TREEP - Editorial
AMSEQT - Editorial
WOUT - Editorial
CHN15D - Editorial
ACM14KP2 - Editorial
Editorial - PEAKS
ACM14KP4 - Editorial
MQRY Editorial

So the problem now boils down to finding the number of sub-arrays in B whose sum >= 0

Consider the PrefixSum array of B, let us call it P. (it will be of same length as B).
For each i we will have P[i] = B[0] + … + B[i]
We can compute this in O(n) time.
Note that sum(B,i..j) = P[j] - P[i-1] (imagine P[-1] to be 0)
Consider the following algorithm to compute our answer.

set count = 0
for i = 0 to (N-1)

- if P[i] >= 0 increment count by 1
- Find the number indices j such that P[j] <= P[i] and j < i, let us call the value x
- increment count by x

Output count

To find the number indices j such that P[j] <= P[i] and j < i, if we naively iterate through all indices < i our algorithm will end up being a O(n ^ 2) algorithm which is no better than our solution for subtask2. But it turns out that we can answer this particular query in O( log n) time by using an appropriate data structure.

Initially the datatstructure will not have any elements.
The idea is simple while processing a particular index i we will query the datastructure for the number of prefixSums in the data structure present already and less than or equal to current prefixSum. This has to done in O(log n) time.
We will add P[i] to the datastructure. This has to be done in O(log n) time.
Hence giving a runtime of O(n log n)

There are at least three data structures which will help you.

- Binary Indexed Tree - See setter's and tester's solution
- Segment Tree - See editorialist's solution
- Balanaced Binary Search Trees - We can use an augmented balanced BST similar to Order Statistic Tree (http://en.wikipedia.org/wiki/Order_statistic_tree). But implementing a balanced BST during a contest is very tough job and hence we will skip this approach.

Refer http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees#read to learn about Binary Indexed Trees

Refer to http://www.topcoder.com/tc?d1=tutorials&d2=lowestCommonAncestor&module=Static and link text to learn about segment trees. But these might be overkill to solve this problem.

Since we have values between -10^9 and +10^9 we can't use a BIT or a segment tree, hence we work with the positions of the values in the original array to index our data structures. Look at the solutions to see how to overcome this.

Also check out this question on quora about BIT vs Segment Tree http://www.quora.com/Data-Structures/How-does-one-decide-when-to-use-a-Segment-Tree-or-Fenwick-Tree

## SOLUTIONS

**Setter's Solution**: SUBARR.cpp
**Tester's Solution**: SUBARR.cpp
**Editorialist's Solution**: SUBARR.java

**Setter's and Tester's Solution Explanation**
You can't use a BIT directly to compute the answer as we can't use our prefix sums to index the BIT data structure as they can be arbitrary integers and we can't have an array of such huge size. Here is how the setter's and tester's solution overcome this.

Create n pairs <sum, index=""> and now sort them on increasing value of sum, if sum is equal then use index. Create a BIT data structure of size n (we store data based on indices). Process the pairs in sorted order when we are processing a pair P all the pairs processed will have sum < P.sum or ( = P.sum and index < P.index) . So we have sufficient information to perform the step "Find the number indices j such that P[j] <= P[i] and j < i, let us call the value x" in our algorithm proposed earlier. The value of x is nothing but the count of all indices < P.index stored in the BIT which can be simply found in O(log n) time by querying the BIT. Now update the BIT with the value P.index. Note that we don't process the prefix sums in the order we proposed in our algorithm but since we process all the prefix sums we still get the same answer.

**Editorialist Solution Explanation**
The structure used is a standard segment tree. Since the values of our prefix sums can be arbitrary and can't directly use the segment tree to represent these value what we have done instead is sort the array and index the elements from 0 to (N-1) (could be less if the array had duplicates) and use these indices to refer to the array values and vice versa. At the last level of the complete binary tree representing segment tree the leaf nodes represent the indices 0 to N-1 and rest of the leaves can be simply ignored. Each leaf stores the count of number of times the value corresponding to that leaf has been inserted in the segment tree so far. Each internal node stores the sum of counts stored in all the leaf nodes which fall under the left subtree of that node. Now let us see how we can perform insert and the query operations.

When we want to increase the count of a value P into the segment tree, we get the corresponding index i of P in the sorted array. Then we locate the $i^{th}$ node from the left in the last level of the segment tree and increase the value of that node. Now we traverse up to the root by moving one level up at a time and since there can be only parent in the tree this path will be unique. When we are travelling up this path all we need to see if we have reached the current path from the left or the right. If we came from left that indicates the node we processed initially was in the left sub tree of the current node so we simply increment its count by 1. If we came from the right side we simply do nothing as the node we processed lies in the right sub-tree.

When we want to query for number of elements less than equal to P in the segment tree, we get the corresponding index i of P in the sorted array. We have a variable ans initialized to 0. Then we locate the i<sup>th</sup> node from the left in the last level of the segment tree and add the value of that node to ans. Then we traverse up to the root just like we did in the insert step. But this time we do just the opposite, if we are coming from the left we do nothing and if we are coming from the right we simply add up the value to ans and do nothing when we come from the left. Now ans we have count of all the elements less than equal to P. If you are convinced why this works just draw up a tree take few examples and see.

This question is marked "community wiki".

editorial fenwick ltime07 segment-tree subarr easy prefix-sum bit

edited 16 Jun '15, 11:37
1★ vicky002 ♦♦
[256]●1●3●12

asked 29 Dec '13, 15:37
2★ vaka ♦♦
[243]●13●18●22
accept rate: 16%

I am not getting the update and sum part of the BIT, why we are adding 1 and finding sum till a[i]. Plz explain it.

4★ codersan (29 Dec '13, 17:40)

Please explain the Editorialist's Solution(segment tree implementation).Can't understand that.

3★ ss_1994 (31 Dec '13, 13:47)

i could nt understand why sorting is used . can anyone explain ? Here BIT is made using indexes ? Am i getting it right ?

4★ shubham1402 (01 Jan '14, 16:40)

1 @codersan @ss_1994 @shubham1402 I had been busy with certain things and not feeling particularly well currently. I will definitely reply back to your queries as soon as I can.

2★ vaka ♦♦ (01 Jan '14, 23:14)

@ss_1994 I have updated the editorial to explain the editorialist's solution.

2★ vaka ♦♦ (06 Jan '14, 08:41)

showing 5 of 6 show all

## One Answer:

oldest answers newest answers popular answers

7 This is the best Lunch Time question I have come across. I appreciate the author. I would like to add that lunch time questions should be like these. Including complex algorithms doesn't really motivate many students. For that, the short challenge is there. Questions of lunch time should be more of thought-provoking and make students think. The algorithms involved should be ones which we normally encounter, rather than having suffix trees, treaps, etc.

link | award points

This answer is marked "community wiki".

edited 04 Jan '14, 08:09

answered 29 Dec '13, 15:52
4★ pushkarmishra
[1.3k]●15●62●81
accept rate: 4%

1 Only two sub tasks of this question can be solved using a O(n^2) algorithm. But we need to make a simple observation and more over use a slightly advanced (not as advanced as suffix tree or treap) data structure to solve this problem.

2★ vaka ♦♦ (06 Jan '14, 09:12)

A lunch time has 4 questions, the questions are set such that two of them are very easy one of them is will be a simple implementation and the other one will have a simple algorithm. The other two will need some clever algorithms most of the times. But even for these the essential idea is you can solve few sub tasks easily and the rest can be solved only using an optimal algorithm. Even the hardest question SFUNC this lunch time had two easily approachable sub tasks. Even if you solves these small subtasks these help you a lot in understanding the question better and in turn the editorial also

2★ vaka ♦♦ (06 Jan '14, 09:14)

## Your answer

[hide preview] community wiki:

Preview

Type the text

Privacy & Terms

Answer the question