FRBSUM - Editorial

17
4

PROBLEM LINK:

Practice
Contest

Author: Constantine Sokol
Tester: Mahbub
Editorialist: Jingbo Shang

DIFFICULTY:

Medium

PREREQUISITES:

Segment Tree, Binary Search, Persistent Segment Tree.

PROBLEM:

Given **A[1..N]**, and **M** queries about what is the smallest number that can't be represented by the sum of some subset of $A[L_i \ldots R_i]$ (so called forbidden sum).

EXPLANATION:

First of all, we need to know how to process a single query. That is, given a set of number, determine the forbidden sum. Consider a special case, the answer should be 1 if we have no ones. Based on this observation, we can firstly sort the number ascending. Take the numbers one by one while maintain a sum **S** (initially **S = 0**). The algorithm is as following:

```
S = 0
while (there is the next number x) {
    if x <= S + 1:
        S = S + x
    else
        break;
}
the forbidden sum is S + 1.
```

Suppose we have the next number **x**. If **x** is greater than **S + 1**, because of the numbers are ascending, **S + 1** will be never got. Therefore, there is a break. Otherwise, **1 .. S + x** are all possible.

And also, we can describe the algorithm in an alternative way (but helpful):

```
S = 0
while (true) {
    newS = the sum of {x | x <= S + 1}
    if (S == newS):
        break
    S = newS
}
the forbidden sum is S + 1.
```

we should observe that, the answer is smaller than the sum of **A[1..N]**, <= 10^9, denoted as **P**. Moreover, **S** is increasing exponentionally, as fast as the Fibonacci Number, because the newly added number should be greater than previous loop's **S**. Therefore, there will be only O(**log P**) loops.

Back to the original problem, for a given query, if we can fast perform the "newS = the sum of {x | x <= S + 1}" in the given **A[L .. R]**, such as in O(**log N**) or O(**log^2 N**) time, then we can solve this problem. Fortunately, we can ask Segment Tree for help.

For the static Segment Tree, we can maintain a sorted list in each node with their prefix sum (O(**N log N**) space is needed, because there are **O(N)** numbers stored in each level). When the query covered the whole interval stated by the node, binary search is adopted to fast locate the "x <= S + 1" position and return the prefix sum. Therefore, we can solve each query in O(**log^2 N**) time.

Furthermore, we can try Persistent Segment Tree to solve this problem in O(**log N**) time. Persistent Data Structure can help us to deal with the **L .. R** query into the delta of **1 .. R** query and **1 .. L - 1** query. Therefore, we can build the Persistent Segment Tree on their values (each node stands for an interval of values of **A[]**) and insert the A[i] from **i = 1 to N** one by one. After that, we can have **N + 1** roots stand for different prefixs (ranging from 0 to **N**. 0-th is an empty tree, **i**-th is the segment tree after inserted **A[1..i]**). Each node this time stores the total numbers have been inserted in to this subtree. For a given query **[L .. R]** and the **S** we maintained, we query the total number <= **S + 1** on the **R**-th tree and **(L-1)**-th tree, and then get the delta. This process is O(**log N**).

Tags:

editorial **×2,814**
medium **×563**
segment-tree **×345**
jan14 **×43**
binarysearch **×29**
persistence **×11**

Asked: **13 Jan '14, 15:11**
Seen: **5,736 times**
Last updated: **19 May, 21:35**

Related questions
Queries Editorial (CRANQRYS) - Crani 2015
PPLUCKY - Editorial
LEBOBBLE - Editorial
EQUAKE - Editorial
SEAD - Editorial
FIBTREE - Editorial
SSTORY - Editorial
ACM14KP3 - Editorial
SEABAL - Editorial
ACM14AM2-Editorial

Finally, using the Persistent Segment Tree, we can solve this problem in O(**N log N + M log P log N**).

AUTHOR'S AND TESTER'S SOLUTIONS:

Author's solution can be found here.
Tester's solution can be found here.

medium jan14 binarysearch segment-tree editorial persistence This question is marked "community wiki".

edited 22 Apr, 17:40
admin ♦♦
10.9k●346●472●486

asked **13 Jan '14, 15:11**
shangjingbo ♦♦
131●44●55●64
accept rate: 0%

7 Just a small comment about the O(log^2(N)) range sum query over the segment tree. A standard implementation does indeed have this time complexity. But the implementation can be optimized to O(log(N)) per query by using the technique called *fractional cascading*. This was not necessary for this problem, but someone might be interested in it.

mugurelionut (14 Jan '14, 01:59)

---

5 Answers: oldest newest **most voted**

4

Since I'm using java, I was very surprised to have one of the fastest implementation here, while I had a very bad worst case. Since I'm not too familiar with the presented datastructures I found myself another way, which I think is not very hard to implement. Note: **Understanding the first part of the editorial is also required to understand why this is working.**

First preprocessing:

- Loop over every number **N** at sequence **id**
- Put its **id** in seperate lists per ceilLog2(**N**) + also keep a list of sums per ceilLog2(**N**) for all numbers in the interval [0,id) with the same ceilLog2 value (I'll call it groups later)
- **ceilLog2(N)** = ceil of log2 N (e.g. {1} = 0, {2} = 1, {3,4} = 2, {5,6,7,8} = 3, ..)

Handle queries:

We can further use this datastructure by summing up per log2 value, starting of course by 0th group. Always adding up all the sums in the interval while the total sum at least is the minimum value for the group being 2^b (`1<<b` for our bitwisers) where b is the group. *Very likely to the code sample of the editorial, but in log n steps*. Sample:

```
int b=0;
int total=1+sums[R][b]-sums[L][b];
b++;
while(total >= (1<<b) || find(nums[b],ns,total,L,R)){
    total+=sums[R][b]-sums[L][b];
    b++;
}
// total is the forbidden sum
```

This still is very fast but we **may miss** some numbers because we can reach higher group with one of the lower numbers in the group. E.g:

```
{1,2,3,6}
=> 1(+1) >= 1<<1
=> 3(+1) >= 1<<2
=> 6(+1) >= 1<<3  // FALSE but we would make 7 >= 6 and therefore go further with next interval
```

Therefore I used a naughty trick *(maybe u've noticed the find method)*, scanning in the interval of the list that we also created in the preprocessing part. Remember that this is per group and we can also use binary search to quickly skip to the interval since we used **id**s. If there exists a number which is smaller or equal to the current total we continue the loop, adding the sum (trivial if you understand).

It's probably so fast because it handles a lot of trivial cases very fast. Can you agree that for this problem, this algorithm's worst case is rather rare?

http://www.codechef.com/viewsolution/3194303

link edited **14 Jan '14, 06:12**

answered **14 Jan '14, 06:07**
samjay
427●1●5●10
accept rate: 11%

1

I have problem in understanding the query part. If we consider the example given in the sample input. Then for the query 1 4 we will get two nodes in the segment tree of interval 1-3 and 4. Then how we will apply binary search to get the answer. Can anyone tell me. Thankyou.

link

answered **14 Jan '14, 19:04**
hatim009
454●2●11●22
accept rate: 8%

0

please someone explain me how this part works:

S = 0 while (there is the next number x) { if x <= S + 1: S = S + x else break; }

though i have understand nearly why this method works but it would be great if someone explains me how to come up with such an idea to solve this question.....

link edited 26 Jan '14, 11:43 answered 26 Jan '14, 11:35

zealf
1.1k•6•11•26
accept rate: 3%

0

This can be done with a persistent trie as well , the implementation is quite short as well .

http://www.codechef.com/viewsolution/6982912

link answered 19 May, 21:35

rajat1603
162•5
accept rate: 0%

-18

```
#include<iostream>
#include<cmath>
using namespace std;
int min;
struct node
{
    int data;
    node* next;
};
class Test
{
    public :
    int l;
    int r;
};
node* add_beg(node* &n,int k)
{
    node *t=new (node);
    t->data=k;
    t->next=n;
    return t;
}
void add(node* &n,int k)
{
    if(n==NULL)
    {
        n=new(node);
        n->data=k;
        n->next=NULL;
    }
    else if(n->data==k)
    return;
    else if(n->data>k)
    {
        n=add_beg(n,k);
    }
    else if(n->next&&n->next->data>k)
    {
        node *t=new (node);
        t->data=k;
        t->next=n->next;
        n->next=t;
    }
    else
    add(n->next,k);
}
int subset(int *A,int size,int k)
{
    int sum=0;
    int rem;
    for(int i=0;i<size;i++)
    {
        rem=k%2;
        k=k/2;
        if(rem==1)
        {
            sum+=A[i];
        }
    }
    return sum;
}
void test(int low,int heigh,int *A)
{
```

```
    node *n=NULL;
    bool flag=false;
    int size=heigh-low+1;
    int sum,index=0;
    for(int i=low-1;i<pow(2,size)+low-1;i++)
    {
        sum=subset(A,size,i);
        add(n,sum);
    }
    while(n!=NULL)
    {
        if(index!=n->data)
        {
            cout<<index<<"\n";
            flag=true;
            break;
        }
        index++;
        n=n->next;
    }
    if(!flag)
    cout<<index<<"\n";
}
int main()
{
    int *A,N,M,sum,index=0;
    Test *tst;
    cin>>N;
    A=new int[N];
    for(int i=0;i<N;i++)
    {
        cin>>A[i];
    }
    cin>>M;
    tst=new Test[M];
    for(int i=0;i<M;i++)
    {
        cin>>tst[i].l;
        cin>>tst[i].r;
    }
    cout<<"\n\n";
    for(int i=0;i<M;i++)
    {
        test(tst[i].l,tst[i].r,A);
    }
    return 0;
}
```

link

edited 14 Jan '14, 01:53
aichemzee
422•6•12•24

answered 14 Jan '14, 01:37
yaduvir
-18
accept rate: 0%

10 if you wanted to add the most senseless comment in forum, this was a good try...

betlista ♦♦ (14 Jan '14, 01:42)

did he wanted to ask some question ?

ashishnegi001 (15 Jan '14, 10:03)

[hide preview] community wiki