2019 povrsina preklopljenih histograma

Autor zadatka atanasov

vreme	memorija	ulaz	izlaz
0,5 s	64 Mb	standardni ulaz	standardni izlaz

Na slici grada nalazi se n solitera pravougaonog oblika (mogu se preklapati). Svaki soliter je definisan početnom i krajnjom x koordinatom i visinom. Napisati program koji će izračunati površinu koji soliteri prekrivaju.

ULAZ
U prvoj liniji standardnog ulaza zadat je ceo broj n (1<=100 000), a potom se u narednih n linija zadaju po tri cela broja u a, b, c u svakoj liniji ($0 \leq a < b \leq 10^9, 1 \leq c \leq 10^9$). Ta t
broja opisuju pojedinačni pravougaonik: a je početna x koordinata pravougaonika, b je krajnja x koordinata, dok je c visina.
IZLAZ
Ispisati koliku površinu slike prekrivaju soliteri (pravougaonici).
ULAZ
9
10 14 4
0 3 3
2 4 6
1 6 2
16 17 4
11 14 3
12 20 1
5 9 3
15 17 3
IZLAZ
59

Na slici dodana je linija koju zamišljamo kako se pomiče preko slike s leva na desno (skenira sliku). Pritom ta linija prelazi preko pravougaonika i postupno sabira površinu i kad dođe na desnu stranu znamo rešenje. Ta linija prelazi preko nekog od sledeća dva događaja:

Naišli smo na levu ivicu pravougaonika visine *c*
Naišli smo na desnu ivicu pravougaonika visine *c*

Kako pomičemo liniju, pamtimo sve pravougaonike preko kojih linija prelazi. Kad se pojavi nova leva ivica pravougaonika, zapamtimo u ***multiset <int> sweep*** njegovu visinu. Klasa ***multiset*** iz STL biblioteke ***set*** moze da ima vise istih elemenata, a ostala svojstva su ista kao kod klase ***set*** iz iste biblioteke.

Kada se pojavi desna ivica pravougaonika, izbacimo njegovu visinu iz sweep. Kako bismo mogli obilaziti dogadjaje sleva nadesno (kako se pomice zamisljene linije), moramo dogadjaje sortirati po *x* osi od manjeg prema vecem. Svaki pravougaonik iz unosa opisan je brojevima a,b,c i opisuje dva dogadjaja:

Leva ivica pravougaonika visine c je na koordinati a
Desna ivica pravougaonika visine c je na koordinati b

Kako bismo razlikovali događaje, onda ćemo prvi događaj notirati kao (a,c), a drugi ćemo notirati kao (b,-c).

Visina je uvek pozitivna, te će negativna vrednost –c označavati da se radi o desnoj ivici pravougaonika, a ne o levoj ivici.

Događaje sortiramo i obilazimo sleva nadesno. Ako je prošli događaj bio na *x* koordinati *prosliX*, a trenutni je na koordinati *a*, tada znamo da se svi pravougaonici čija je visina zapisana u setu protezu od koordinate *prosliX* do *a,* te zato povecavamo ukupunu povrsinu za (a-prosliX)*najveciPravougaonikTrenutnoUSetu

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;

int main()
{   int n; cin >> n;
   vector<pair<int,int> > dogadjaj;
   int a,b,c;
   for(int i=0;i<n;i++)
   {  cin >> a >> b >> c;
   dogadjaj.push_back(make_pair(a,c)); //zapisujemo dogadjaj: leva ivica pravougaonika
   dogadjaj.push_back(make_pair(b,-c)); //desna ivica pravougaonika
   }
   sort(dogadjaj.begin(), dogadjaj.end()); //sortiramo zapisane dogadjaje

   long long resenje=0; //resenje moze da prevazidje int
   multiset <int> sweep; //kreiramo multiset, a ne set, jer moze postojati vise solitera iste visine

 sweep.insert(0); /*vazno je da ubacimo 0, kako bismo pri dohvatu najveceg solitera uvek imali sta dohvatati, inace bi se srusio program*/

   sweep.insert(dogadjaj[0].second); //ubacujemo 1. dogadjaj u sweep, a to je sigurno leva ivica pravougaonika
   int prosliX=dogadjaj[0].first; //pamtimo na kojoj x koordinati je taj dogadja

   for(int i=1; i<dogadjaj.size();i++)  //obilazimo ostale dogadjaje
   {  a=dogadjaj[i].first;
```

```
      b=dogadjaj[i].second;

      /*povecavamo resenje za (a-prosliX)*najveciPravougaonikTrenutnoUSetu
       i vodimo racuna da rezultat bude tipa long long.
       Najveca vrednost (multi)seta se nalazi pre kraja seta, te zato
       uzimamo iterator kraja i vracamo se unazad operatorom --,
       te uz operator * pristupamo vrednosti na koju pokazuje iterator
      */
      resenje+=((long long)a-prosliX)* *(--sweep.end());
      prosliX=a; //azuriramo vrednost promenljive prosliX

       /*izbacujemo vrednost iz multiseta ako se radi o desnom rubu
pravougaonika */
      if (b<0) sweep.erase(sweep.find(-b));
           //OPREZ: ne smemo koristiti poziv sweep.erase(-b), jer bi se obrisali svi
clanovi s vrednoscu -b
      /*u suprotnom, ubacujemo vrednost u multiset */
      else sweep.insert(b);
   }

   cout << resenje << endl;
   return 0;
}
```

Prostorna složenost: O(n)
Vremenska složenost: O(n log n) jer se nad multisetom obavlja n operacija koje imaju složenost O(log n).

Inače, čak i da koordinate na ulazu programa nisu celobrojne, zadatak bismo mogli rešiti na isti način.

2.

1 s	64 MB	standardni ulaz	standardni izlaz

Dat je niz a, koji se sastoji od n pozitivnih celih brojeva. Napisati program koji će da prebroji koliko ima parova brojeva l i r takvih da $1 \leq l < r \leq n$ i niz $b = a_1 a_2 \ldots a_l a_r a_{r+1} \ldots a_n$ ima ne više od k inverzija.

Dva cela broja u nizu b obrazuju inverziju kada god veći broj se pojavljuje pre manjeg broja. tj. kažemo za par članova b_i , b_j niza b da obrazuju inverziju ako važi da $1 \leq i < j \leq |b|$ i $b_i > b_j$, gde je |b| veličina niza b.

ULAZ

U prvoj liniji standardnog ulaza data su dva cela broja n i k ($2 \leq n \leq 10^5$, $0 \leq k \leq 10^{18}$) — veličina niza a i maksimalan broj dozvoljenih inverzija.

Sledeća linija sadrži n pozitivnih celih brojeva razdvojenih jednim razmakom a_1, a_2, ..., a_n ($1 \leq a_i \leq 10^9$) — elementi niza a.

IZLAZ

U jedinoj liniji standardnog izlaza ispisati samo jedan broj — broj parova (rešenje zadatka).

Primeri

Primer 1

Ulaz

3 1

1 3 2

Izlaz

3

Primer 2

Ulaz

5 1

5 4 3 2 1

Izlaz

1

Primer 3

Ulaz

3 1

1 2 3

Izlaz

3

Primer 4

Problem možemo rešavati tehnikom dva pokazivača i RMQ (range minimum query).

Najpre preslikajmo sve elemente ulaznog niza u segment [0..n-1].

Tokom rešavanja zadatka održavamo dva n-dimenziona RMQ (koje redom označimo sa A_1 i A_2).

A_{1i} sadrži broj pojava broja ***i*** u tekućem levom podnizu.
A_{2i} sadrži broj pojava broja ***i*** u levom podnizu.

Neka su svih ***n*** brojeva dodati u A_2.

Sada ćemo koristitu operaciju suma segmenata.
Koristeći A_1 i A_2, pamtimo broj inverzija smanjivanjem pokazivaca ***r*** ili ***l***.

Iteriramo pokazivačem ***r*** od ***n - 1*** do ***1***, održavajući pokazivač l (inicijalno postavljen na ***n- 1***).

Dok je tekući broj inverzija veći od date vrednosti ***k*** i dok je l >=0, smanjujemo ***l***. Potom, za svako r korektna vrednost za l je ***l + 1*** (0-indeksiranje).

Vremenska složenost je O(N log N).

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <numeric>

using namespace std;

int count_inversions_in_segment(vector<int> &segment_tree, int n, int a, int b) {

   a += n-1;
   b += n-1;
   int s = 0;
   while (a <= b) {

      if (a % 2 == 1)
         s += segment_tree[a++];
      if (b % 2 == 0)
         s += segment_tree[b--];

      a /= 2;
      b /= 2;
   }
   return s;
}

void add_element_to_tree(vector<int> &segment_tree, int index, int height, int num) {

   index += height - 1;
   while(index != 0) {

      segment_tree[index] += num;
      index /= 2;
   }
}

int main () {

   int n;
   unsigned long long k;
   ios_base::sync_with_stdio(false);
```

```
cin >> n >> k;
vector<int> v(n);
for(int i=0; i<n; i++)
   cin>>v[i];

int height = pow(2, ceil(log2(n)));
vector<int> segment_tree(2 * height, 0);
vector<int> segment_tree1(2 * height, 0);
vector<int> segment_tree2(2 * height, 0);

vector<int> s = v;
sort(s.begin(), s.end());
for (int i = 0; i < n; i++) {

   int pos = distance(s.begin(), lower_bound(s.begin(), s.end(), v[i])) + 1;
   v[i] = pos;
}

vector<unsigned long long> big(n);
vector<unsigned long long> small(n);
unsigned long long inversions = 0;

for (int i = 0; i < n; i++) {

   inversions += big[i] = count_inversions_in_segment(segment_tree, height, v[i]+1, height);
   add_element_to_tree(segment_tree, v[i], height, 1);
}

for (int i = n-1; i >= 0; i--) {

   small[i] = count_inversions_in_segment(segment_tree1, height, 0, v[i]-1);
   add_element_to_tree(segment_tree1, v[i], height, 1);
}

unsigned long long pairs = 0, curInv = inversions;
int l = 0, d = 1;

while(l < n-1 && d < n) {

   unsigned long long inv = 0;

   if(curInv > k) {

      inv = count_inversions_in_segment(segment_tree2, height, v[d]+1, height);
      add_element_to_tree(segment_tree2, v[d], height, 1);

      curInv -= big[d] + small[d] - inv;
      d++;
   }
   else {
```

```
        int num = count_inversions_in_segment(segment_tree2, height, 0, v[l+1]-1);
        add_element_to_tree(segment_tree2, v[l+1], height, -1);

        pairs += n-d;
        curInv += big[l+1] + small[l+1] - num;
        l++;

        if(l == d) {

          inv = count_inversions_in_segment(segment_tree2, height, v[d]+1, height);
          add_element_to_tree(segment_tree2, v[d], height, 1);

          curInv -= big[d] + small[d] - inv;
          d++;
        }
      }
    }

    cout << pairs << endl;

    return 0;
}
```

Respect – rešenje Uroš Maleš

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int NMAX=1e5+7;

int n, a[NMAX], p[NMAX];
ll uk, k;
int f[2][NMAX];

int calc(int koji, int x)
{
    if(koji==0) x=n-x+1;
    int res=0;
    for(;x;x-=x&(-x)) res+=f[koji][x];
    return res;
}

void add(int koji, int x, int sta)
{
    if(koji==0) x=n-x+1;
    for(;x<=n;x+=x&(-x)) f[koji][x]+=sta;
}
```

```
int main()
{
    ios_base::sync_with_stdio(false);
    cin >> n >> k;
    for(int i=1;i<=n;i++) cin>>a[i], p[i]=a[i];
    sort(p+1,p+n+1);
    for(int i=n;i;i--)
    {
        a[i]=lower_bound(p+1,p+n+1,a[i])-p;
        uk+=calc(1, a[i]-1);
        add(1,a[i],1);
    }
    int r=1, l;
    ll ans=0;
    for(l=1;l<=n;l++)
    {
        uk+=calc(0,a[l]+1)+calc(1,a[l]-1);
        add(0,a[l],1);      // dodajemo i uracunavamo l
        while((uk>k || r<=l) && r<=n)
        {
            //brisemo r
            uk-=calc(0,a[r]+1)+calc(1,a[r]-1);
            add(1,a[r],-1);
            r++;
        }
        ans+=n-r+1;
    }
    cout<<ans;
    return 0;
}
```