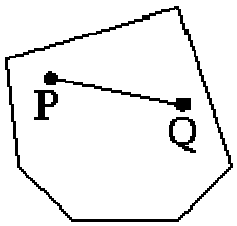
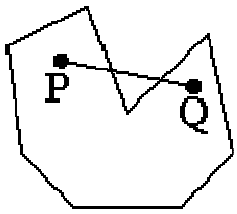
Geometrijski algoritmi (prost mnogougao, konveksni omotač skupa tačaka)

Podsećanje



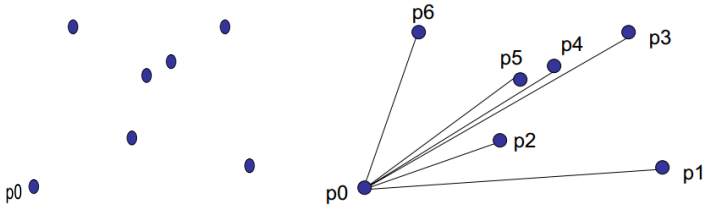
Convex



NonConvex

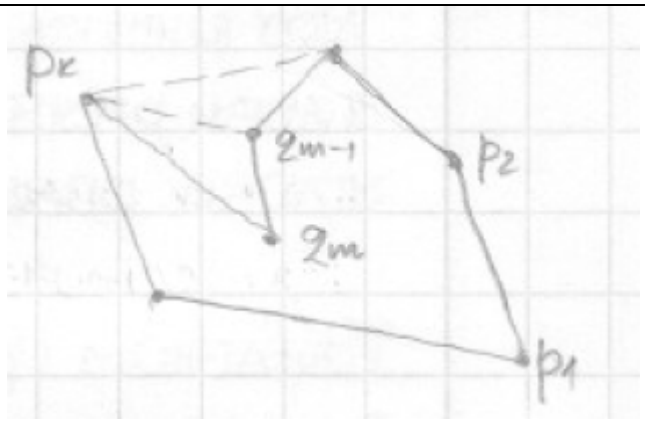
Prost mnogougao je onaj kod koga odgovarajući put nema preseka sa samim sobom, tj. jedine ivice koje imaju zajedničke tačke su susedne ivice sa njihovim zajedničkim temenom.

Algoritam Prost mnogougao(p_0, p_1, . . . ,p_{n-1})
Ulaz p_0, p_1, . . . ,p_{n-1} (skup tačaka u ravni)
Izlaz P (prost mnogougao sa temenima p_0, p_1, . . . ,p_{n-1})
begin
promeniti oznake tako da p_0 ima maksimalnu x koordinatu (ako ih ima više onda biramo onu sa minimalnom y koordinatom)
for i := 1 to n-1 do
  izračunati ugao α_i izmedju $p_0 - pi$ i x ose
sortirati tačke prema uglovima α_1, . . ., α_{n-1}
(u grupi sa istim uglom, sortirati ih prema rastojanju od p_0)
P je mnogougao definisan sortiranom listom tačaka
End



Konveksni omotač skupa tačaka je najmanji konveksni mnogougao koji sadrži sve tačke skupa. Graham je 1972. godine objavio rad u kome je predstavio relativno jednostavan algoritam za računanje konveksnog omotača konačnog skupa tačaka u ravni sa vremenskom složenošću O(n log n) u najgorem slučaju. Taj algoritam se često naziva prvim "computational geometry" algoritmom.

Algoritam Grahamov algoritam(p_0, p_1, . . . ,p_{n-1})
Ulaz p_0, p_1, . . . ,p_{n-1} (skup tačaka u ravni)
Izlaz q_0, q_1, . . . , q_{m-1} (konveksni omotač tačaka p_0, p_1, . . . ,p_{n-1})
begin
neka je p_0 tačka sa maksimalnom x koordinatom (ako ih ima više onda biramo onu sa minimalnom y koordinatom)
uređujemo tačke algoritmom Prost mnogougao(p_0, p_1, . . . ,p_{n-1})
$q0 := p0$;
$q1 := p1$;
$q2 := p2$;
$m := 2$;
for $k := 3$ to n-1 do
while ugao između ($q_{m-1} - q_m$, $q_m - p_k$) >=π do
(ugao se meri iz unutrašnjosti mnogougla)
  $m := m - 1$;
$m := m + 1$;
$qm = pk$;
end



U kojim oblastima se primenjuje određivanje konveksnog omotača datog skupa tačaka?

1. **Pattern recognition**: Akl, S.G. and Toussaint, G.T., Efficient convex hull algorithms for pattern recognition applications, Int. Joint Conf. on Pattern Recognition, Kyoto, Japan, (1978),483–487
2. **Data mining**: Böhm, C. and Kriegel, H., Determing the convex hull in large multidimensional databases, Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery, Lecture Notes in Computer Science, Springer-Verlag, 2114 (2001) 294–306.
3. **Stock cutting and allocation:** Freeman, H. and Shapira, R., Determining the minimum-area encasing rectangle
for an arbitrary closed curve, Comm. ACM, 18(7) (1975)
4. **Image processing:** Rosenfeld, A., Picture Processing by Computers, Academic Press, New York, 1969

Zadaci

1. Odrediti vremensku složenost datog Grahamovog algoritma u najboljem, najgorem i prosečnom slučaju za dati skup Q od n tačaka.
Napomena, ovo je CLR verzija algoritma i razlikuje se od Udi Manber verzije (i verzije u udzbeniku *Algoritmika*) u redu 1

GRAHAM_ALGORITAM(Q)

1. Find p_0 in Q with minimum y-coordinate (and minimum x-coordinate if there are ties).
2. Sorted the remaining points of Q (that is, Q − {p_0}) by polar angle in counterclockwise order with respect to p_0.
3. TOP [S] = 0 ▷ Lines 3-6 initialize the stack to contain, from bottom to top, first three points.
4. PUSH (p_0, S)
5. PUSH (p_1, S)
6. PUSH (p_2, S)
7. **for** i = 3 **to** m ▷ Perform test for each point p_3, ..., p_m.
8. **do while** the angle between NEXT_TO_TOP[S], TOP[S], and p_i makes a nonleft turn ▷ remove if not a vertex
9. **do** POP(S)
10. PUSH (S, p_i)
11. **return** S

Teorema: Grahamov algoritam ima vremensku složenost O(n log n).
Dokaz:

Red 1: vremenska složenost O(n) za nalaženje polazne tačke p_o (tačke sa najmanom y-koordinatom).

Red 2: sortiranje prema veličini tzv. polarnog ugla zahteva O(n lg n) vremenski složenost, (koristeći merge ili heap sort za sortiranje uglova). Štaviše, uklanjanje $n - m$ tačaka koje grade jednake uglove zahteva O(n) vremena

Redovi 3, 4, 5, 6 zahtevaju konstantno vreme O(1) za inicijalizaciju steka koji sadrži prve tri tačke

Red 7 i **For-loop** se izvršava $m - 2$ puta, te je vremenska složenost O(n).

Red 8 sadrži **while-loop** koja može iteratirati najviše za O(n), što nas vodi u ukupnu složenost od O(n^2). Svaki prolaz kroz **while** ciklus, zahteva izvršavanje uklanjanje sa steka, tj. POP proceduru.

Uočimo da postoji najviše jedna POP opercija za svaku PUSH operciju.

U našem algoritmu, bar tri tačke (p_0, p_1, p_m) se nikad ne uklanjaju sa steka.

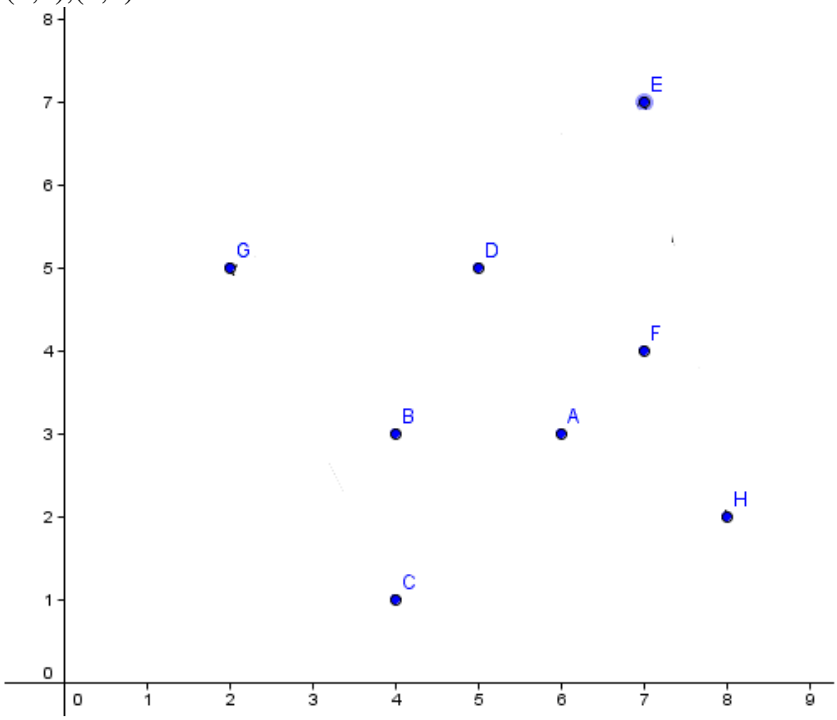
Dakle, izvrši se najviše $m - 2$ POP operacija. Stoga, amortizovana cena svih iteracija u while ciklusu je O(n)/n = O(1).
Gde ste se susreli sa amortizacijom vremenske složenosti? CLR knjiga poglavlje 17, algoritmi rangovskig statistika.

Otuda sledi da NAJGORI slučaj za for ciklus je vremenske složenosti O(n). Najgori slučaj za Grahamov algoritam je

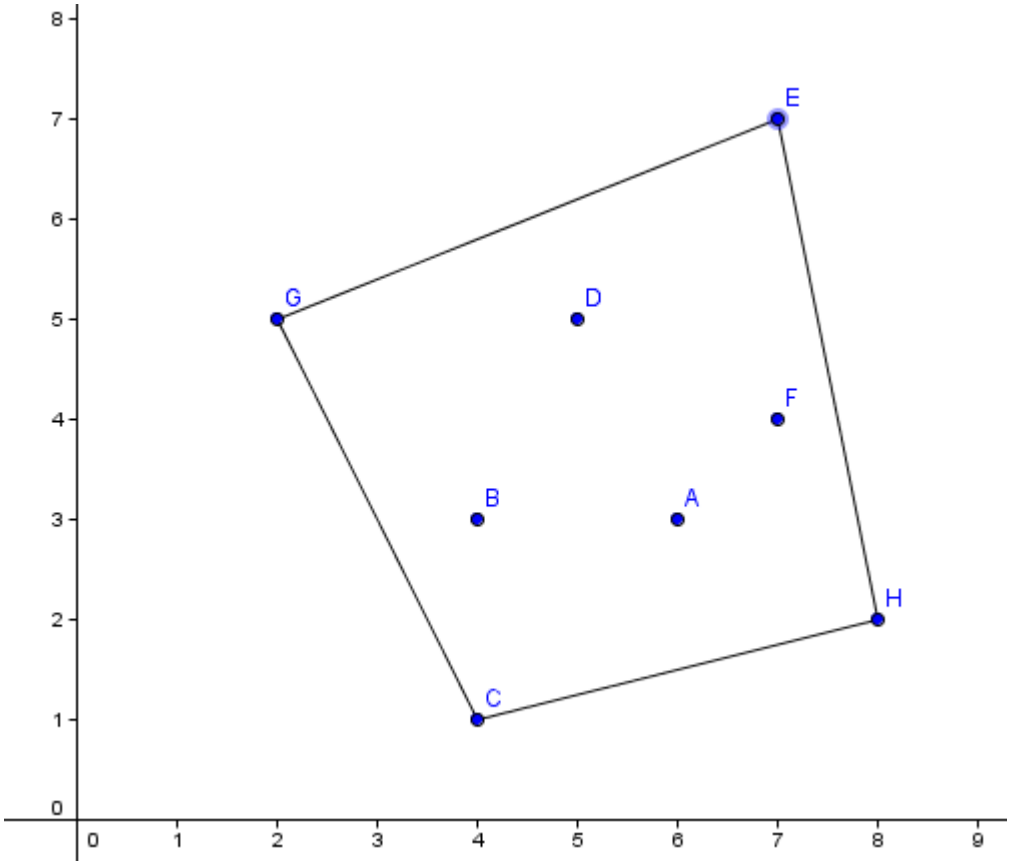
$$T(n) = O(n) + O(n \lg n) + O(1) + O(n) = O(n \lg n), \text{ gde je } n = |Q|.$$

Dakle, može biti najviše n guranja i n uklanjanja sa steka, te je složenost ovog dela algoritma O(n).
Složenošću dominira sortiranje, a za njega je poznata donja granica O(n log n)
Opšti slučaj: O(n log n).
Najgori slučaj: O(n log n).
Najbolji slučaj: O(n log n).

2. Upotrebom Graham-ovog algoritma nađite konveksni omotač za skup (6,3), (4,3), (4,1), (5,5), (7,7), (7,4) (2,5),(8,2)



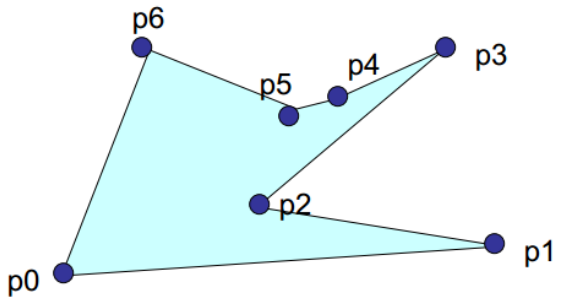
REŠENJE



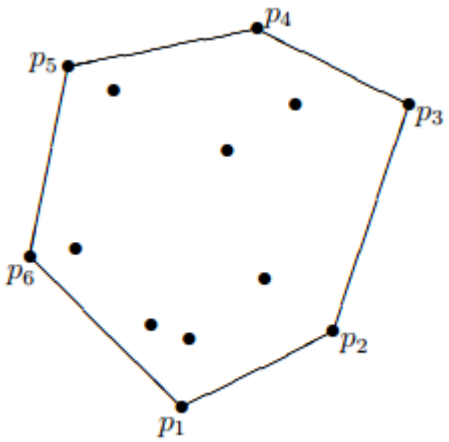
1. Najpre nađimo tačku p0 sa najmanjom y-koordinatom. To je C=(4,1).
2. Sledeći zadatak je sortiranje svih tačaka prema uglu koji zaklapaju sa početnom tačkom, merenom u smeru suprotnom od kretanja kazaljke na satu u odnosu na x-osu. Sortirajmo tačke prema algoritmu Prost mnogougao, tj. prema uglu između p0 i x koordinatne ose u smeru obrnuto od kazaljke na satu.
Uočimo da tačke A=(6,3) i F=(7,4) grade isti traženi polarni ugao.
Ali, tačka F(7,4) je udaljenija od p0, i uklanjamo A(6,3).
Poredak ostalih tačaka je
p1 = H(8,2),
p2 = F(7,4),
p3 = E(7,7),
p4 = D(5,5),
p5 = B(4,3),
p6 = G(2,5).
3. Najpre se na stek postave tačke p0, p1, p2, tj. tačke C, H, F
4. Redosled p1, p2, p3 (gledamo tacke H, F, E) skreće udesno, što možemo detektovati i računanjem
$(p3-p1)\times(p2-p1)=(-1,5) \times (-1,2)=(-1)(2)-5(-1)=-2+5=3>0$
gde $p3-p1=(x3-x1,y3-y1)=(7-8,7-2)=(-1,5)$
(Negativni rezultat bi bio usmeren ulevo.)
5. Stoga, uklanjamo p2 (tacku F) sa steka; nakon toga, the redosled p0, p1, p3 (tačke C, H, E) skreće ulevo.
6. Tačka p3 (tačka E) se postavlja na stek koji sad sadrži p0, p1, p3 (tačke C, H, E).
7. Redosled p1, p3, p4 skreće ulevo: p4 se postavlja na stek (tačke H, E, D).
8. Putanja p3, p4, p5 (tačke E, D, B) opet skreće ulevo t: postavlja se p5 (tačka B) na stek koji sad sadrži p0, p1, p3, p4, p5 (tačke C,H,E,D,B).
9. Na kraju, ispitajmo tačku p6 (tačku G). Redosled p4, p5, p6 (tačke D, B, G) je usmeren udesno: p5 (tačka B) se skida sa steka i razmatramo redosled p3, p4, p6 (tačke E, D, G) koji je pak desno usmeren. Stoga moramo ukloniti p4 (tačku D) sa steka koji sad sadrži p0, p1, p3 (tačke C, H, E).
10. Na kraju, putanja p1, p3, p6 (tačke H, E, G) je usmerena ulevo i postavljamo p6 (tačku G) na stek.
11. Algoritam terminira rad i stek sadrži p0, p1, p3, p6.
12. Stoga konveksni omotač sadrži ivice sa temenima:
(4,1), (8,2), (7,7) , (2,5).

3.

a) Da li je svaki prost mnogougao istovremeno i konveksan?
Ne.



b) Da li u Grahamovom algoritmu {q1,... q_m } = {p1, p2, . . .,pn}?
Ne. Na primer ovde konveksni omotač sadrži svega 6 tačaka polaznog skupa.



Da li u algoritmu Grahamov algoritam tačke q1 i q_m moraju pripadati konveksnom omotaču tačaka p1, p2, . . .,pn.

Da.
Prvi korak je identifikacija najniže (ili krajnje desne u verziji Udi Manber) tačke - označimo je sa 1. Ova tačka se mora nalaziti na omotaču.

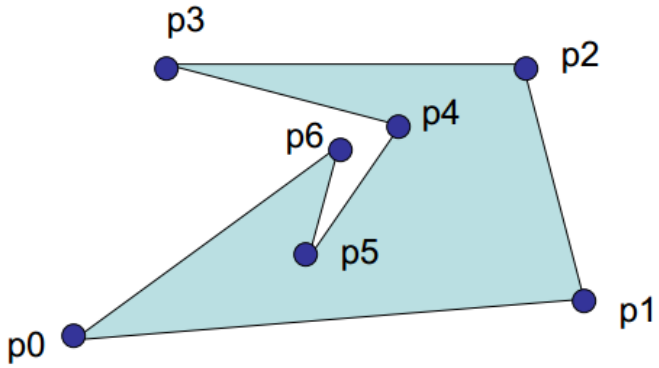
Sledeći zadatak je sortiranje svih tačaka prema uglu koji zaklapaju sa početnom tačkom, merenom u smeru suprotnom od kretanja kazaljke na satu u odnosu na x-osu. Tačka koja gradi najveći ugao se gura na stek ispred tačke 1.
Zatim redom obrađujemo tačke počev od one sa najmanjim uglom (tačka 2), dok ne dođemo do krajnje tačke (tačka n). Ovaj proces uključuje test za strogo levo skretanje od tačke sa vrha steka ka testiranoj tački. Ako je nađeno skretanje u levo, ta tačka se gura na stek i prelazi se na obradu naredne tačke. Ako nije nađeno skretanje u levo, tada se uklanja tačka sa vrha steka i ista tačka se ponovo proverava.

Bar tri tačke (q_1, q_2, q_m) se ne uklanjaju sa steka, te mora i tačka q_m biti deo omotača

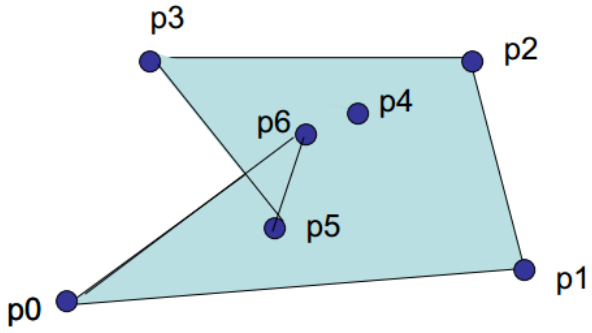
c) Da li je moguće naći konveksni omotač algoritmom manje vremenske složenosti od O(n log n)?

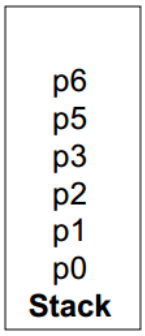
Da.
Evo lista uspešnih i neuspešnih pokušaja da se pronađe algoritam linearne složenosti O(n)
http://cgm.cs.mcgill.ca/~athens/cs601/intro.html

d) Da li je kod Grahamovog algoritma suvišan korak 2 (sortiranje tačaka u poretku prostog mnogougla)?
Ne.
Pretpostavimo suprotno, tj. da smo iz koraka 1 ušli u korak 3 sa nesortiranim skupom tačaka
p0,p1,p2,p3,p4,p5,p6



Na kraju, konacno rešenje bi bio stek, tj. omotač <p0,p1,p2,p3,p5,p6>





Međutim, konveksni omotač je <p0,p1,p2,p3>

4.

Planira se izgradnja novog autoputa sa ciljem je da on bude dostupan što većem broju stanovnika. Autoput ima oblik prave linije, koja je paralelna sa Y osom. Dužina puta može biti proizvoljno velika. Poznato je da će stanovnici određenog grada koristiti autoput ako najkraće rastojanje od njihovog grada do tog puta nije veće od graničnog rastojanja D.
U prvoj liniji standardnog ulaza nalazi se broj N (broj gradova, $N \leq 50000$) i broj D, koji su razdvojeni razmakom.
U narednih N redova sledi opis svakog grada koji čine tri broja razdvojena razmakom, najpre su upisane koordinate X i Y, a potom sledi broj stanovnika tog grada.
Kreirati program koji ispisuje maksimalan broj stanovnika koji bi mogao da koristi autoput čija izgradnja se planira.

Primer:

gradovi.txt	**autoput.out**
5 2	16000
11 9 1000	
2 5 2000	
7 5 10000	
5 8 1000	
9 2 5000	

Objašnjenje:
Autoput će biti prava *X*=7, tako da će biti dostupan trećem, četvrtom i petom gradu iz ulaza.

Rešenje

Ideja:
1. Najpre ćemo sortirati gradove prema X koordinatama.
2. Zatim, krenuvši sa leve na desnu stranu, vodimo računa o intervalu koji je manji ili jednak 2D. Kako novi grad „uđe" u interval, tako trenutnoj sumi dodajemo broj stanovnika grada, a kada grad više nije u intervalu, onda smanjimo sumu za njegov broj stanovnika. U svakom trenutku proveravamo da li trenutna suma prevazilazi maksimalnu sumu.

Pseudokod

SORT-GRADOVI

```
l = 0
max = 0
FOR d = 0 to n-1
        WHILE x[d] - x[l] > 2D
                l++
                sum = sum - s[l]
        sum += s[d]
        IF sum > max THEN
                max = sum
RETURN max
```

Složenost ovog algoritma je O(Nlog(N)). To je složenost sortiranja. Složenost petlje je linearna zbog toga što ukupan broj iteracija kroz WHILE petlju ne može da pređe N.

SPORIJE C REŠENJE algoritmom selection sort => slozenost $O(n^2)$

```
#include <stdio.h>
#include <stdlib.h>
using namespace std;
#define MAX 50001
// koristi se za uporedjivanje gradova po x koordinati
// i odgovarajucoj razmeni indeksa u nizu stanovnici
void selection_sort_1(int a[], int b[], int n) {
 int i, j;
 for (i = 0; i < n-1; i++)
 for (j = i+1; j < n; j++)
 if (a[i]>a[j])
 { /* trampa i-tog i j-tog clana nizova a, b*/
 int pom1, pom2;
 pom1=a[i]; a[i]=a[j]; a[j]=pom1;
 pom2=b[i]; b[i]=b[j]; b[j]=pom2;
 }
}
int gradovi [MAX];
int stanovnici [MAX];
int main()
{
 int n, d, i;
 int max = 0, sum = 0, l, r;
 scanf ("%d%d", &n, &d);
 for (i = 0; i < n; i++)
 scanf("%d%*d%d", &gradovi[i], &stanovnici[i]);
 //podsecanje *d ignorise ceo broj sa ulaza, jer nam ne treba y koordinata
 selection_sort_1(gradovi, stanovnici,n);
 l = 0; max = stanovnici[0]; sum = max;
 for (r = 1; r < n; r++){
 while (gradovi[r]-gradovi[l]>2*d)
 sum -= stanovnici[l++];
 sum += stanovnici[r];
 max = (max<sum)?sum:max;
 }
 printf("%d",max);
 return EXIT_SUCCESS;
}
```

BRŽE C++ REŠENJE upotrebom bibliotecke funkcije qsort

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define MAX 50001
// x i y su koordinate, a z je broj stanovnika
typedef struct{
 int x, y, z;
} city;
// koristi se za uporedjivanje gradova po x koordinati
// zbog quick sorta
int up(const void *px, const void* py){
 city* a = (city*) px;
 city* b = (city*) py;
 if (a->x < b->x) return -1;
 if (a->x > b->x) return 1;
 return 0;
}
int main()
{
 int x, y, z, n, d, i;
 int max = 0, sum = 0, l, r;
 city gradovi[MAX];

 cin >> n >> d;
 for (i = 0; i < n; i++){
 city c;
 cin >> c.x >> c.y >> c.z;
 gradovi[i] = c;
 }

qsort(gradovi, n, sizeof(city), up);

 l = 0; max = gradovi[0].z; sum = gradovi[0].z;
 for (r = 1; r < n; r++){
 while (gradovi[r].x-gradovi[l].x>2*d)
 sum -= gradovi[l++].z;
 sum += gradovi[r].z;
 max = (max<sum)?sum:max;
 //max >?= sum;
 }
 cout << max << endl;

 return EXIT_SUCCESS;
}
```

Za domaci: Implementirajte rešenje koristeći bibliotečku funkciju sort iz biblioteke algorithm.h

5. Dato je **n** pravougaonika u koordinatnom sistemu sa stranicama paralelnim osama. Svaki pravougaonik je dat koordinatom donjeg-levog i koordinatom gornjeg-desnog temena. Naći površinu njihovih preseka.

INPUT:

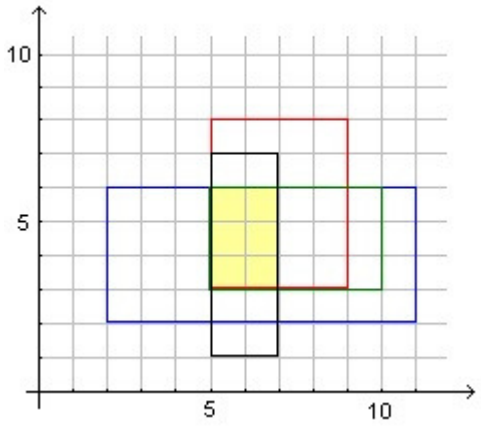
Prvi red standardnog ulaza sadrži ceo broj **n** (2 <= **n** <= 200 000). Potom se u narednih **n**redova učitava po četiri broja $\mathbf{x1_i}$, $\mathbf{y1_i}$, $\mathbf{x2_i}$ ($\mathbf{x1_i}$ < $\mathbf{x2_i}$) i $\mathbf{y2_i}$ ($\mathbf{y1_i}$ < $\mathbf{y2_i}$), 1 <= **i** <= **n**. Svaka koordinata je ceo broj iz intervala [0, 100 000].

OUTPUT:

Prvi red standardnog izlaza treba da sadrži jedan ceo broj koji predstavlja površinu preseka datih pravougaonika.

Ulaz: Izlaz:

4 **6**
2 2 11 6
5 3 9 8
5 3 10 6
5 1 7 7



Objašnjenje:
Pravougaonik sa donjim-levim temenom (5, 3) i gornjim-desnim temenom (7, 6) je presek datih pravougaonika.

Ulaz: Izlaz:
2 **0**
0 0 20 20
50 50 100 100

Objašnjenje:
Pravougaonici nemaju presek, pa je tražena površina 0.

Rešenje

```
#include <stdio.h>
#include <stdlib.h>

main()
{ long n,i, dx,dy,gx,gy,x1,x2,y1,y2;
   scanf("%ld",&n);
      scanf("%ld%ld%ld%ld",&dx,&dy,&gx,&gy);
  for(i=2;i<=n;i++)
  {
     scanf("%ld%ld%ld%ld",&x1,&y1,&x2,&y2 );
       if (x1>dx) dx=x1;
        if (x2<gx) gx=x2;
        if (y1>dy) dy=y1;
        if (y2<gy) gy=y2;
        if((gy-dy<1)||(gx-dx<1)){printf("0"); exit(0);}

  }

  printf("%ld",(gy-dy)*(gx-dx));
  return 0;
}
```

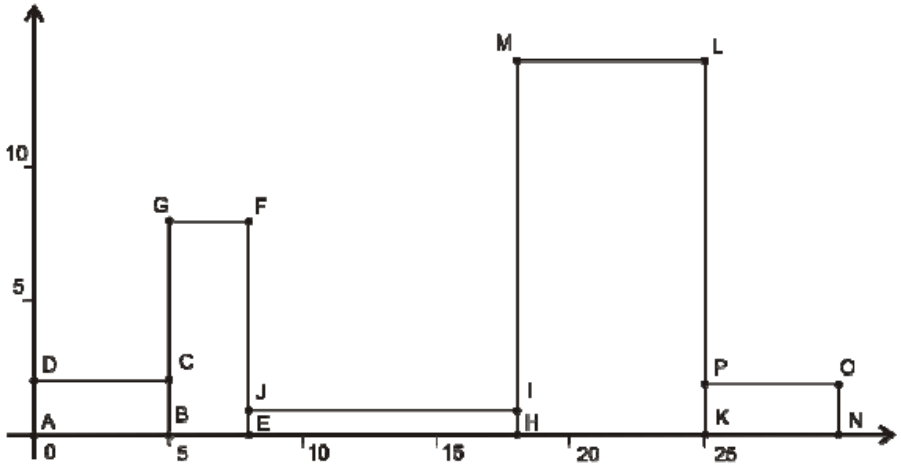
6. Dato je n pravougaonika koji su numerisani od 1 do *n*. Postavićemo svaki od tih pravougaonika na osu *OX* u smeru sleva nadesno redom poštujući redosled numeracije pravougaonika. Svaki pravougaonik je postavljen na osu *OX* svojom dužom ili kraćom stranicom. Potrebno je izračunati dužinu gornjeg dela konture tj. obim konture bez dužine leve, desne i donje ivice. Napisati C program koji će naći maksimalnu moguću dužinu gornjeg dela konture. Sa standardnog ulaza se učitava vrednost za n ($0 < n < 1000$), a potom se iz svake od n

linija učitava ju po dva cela broja a_i, b_i koji predstavljaju dužine stranica *i-tog* pravougaonika ($0<a_i<b_i<1000$) za svako $i = 1, 2, ..., n$.

Primer

ULAZ

```
5
2 5
3 8
1 10
7 14
2 5
```



IZLAZ

```
68
```

Objasnjenje primera: Na slici je predstavljen raspored pravougonika za koji se dobija maksimalna duzina gornjeg dela konture.

Gornji deo konture se sastoji od duzi *DC, CG*, *GF*, *FJ*, *JI*, *IM*, *ML*,*LP*, *PO*. Ukupna duzina je 5 + 6 + 3 + 7 + 10 + 13 + 7 + 12 + 5 = 68.

Rešenje:

Dinamičko programiranje koje koristi induktivni pristup: Označimo sa $f(i)$ maksimalnu dužinu gornje konture za prvih i pravougaonika takvih da poslednji pravougaonik među njima je smešten na osu OX svojom *kraćom* stranicom. Analogno, označimo sa $g(i)$ maksimalnu dužinu gornje konture za prvih i pravougaonika takvih da poslednji pravougaonik među njima je smešten na osu OX svojom *dužom* stranicom. Jasno da važi $f(1) = a_1$ i $g(1) = b_1$.

g(1)=b1,
f(2)=b1+b2-a1+a2
f(2)=a2+max{f(1)+|b2-b1|, g(1)+|b2-a1|}=a1+max{a1+|b2-b1|, b1+|b2-a1|}=
=a2+b1+|b2-a1|, jer je pravougaonik 1 postavljen dužom stranom

Koristimo opet konstrukciju algoritama indukcijom, tj. pretpostavimo da smo već izračunali vrednosti $f(i)$, $g(i)$ za neko i, $1 <= i < n$. Onda važi da je:

$$f(i+1) = a_{i+1} + \max\left\{ f(i) + \left|b_i - b_{i+1}\right|,\ g(i) + \left|a_i - b_{i+1}\right| \right\}$$

$$g(i+1) = b_{i+1} + \max\left\{ f(i) + \left|b_i - a_{i+1}\right|,\ g(i) + \left|a_i - a_{i+1}\right| \right\}$$

Iterativno (indukcijom po induktivnom koraku), dobijamo rezulte u vidu parova ($f(1)$, $g(1)$), ($f(2)$, $g(2)$), ...,($f(n)$, $g(n)$). Konačno, rešenje je najveći od brojeva $f(n)$ i $g(n)$.
U implementaciji ćemo redom vrednosti $f(i)$ i $g(i)$ pamtiti kao elemente niza $t[i][0]$ i $t[i][1]$.

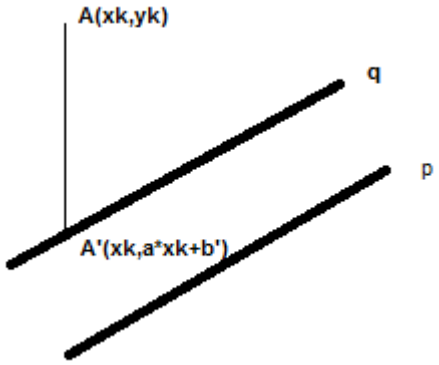
```
#include<iostream>
#include<cstdlib>
using namespace std;
const int N=1001;
int n;
int a[N], b[N], t[N][2];
int main()
{
```

```
cin >> n;
for(int i=1;i<=n;i++) cin >> a[i] >> b[i];
t[1][0]=a[1]; t[1][1]=b[1];


for(int i=2;i<=n;i++)
{
   t[i][0]=a[i]+ max(t[i-1][0]+abs(b[i-1]-b[i]),t[i-1][1]+abs(a[i-1]-b[i]));  // f(i)

  t[i][1]=b[i]+max(t[i-1][0]+abs(b[i-1]-a[i]),t[i-1][1]+abs(a[i-1]-a[i]));    //g(i)
}

cout << max(t[n][0], t[n][1]) << endl;

}
```

7. Konstruisati algoritam linearne složenosti koji za datih *n* tačaka i datu pravu *p* pronalazi pravu paralelnu sa *p* koja dati skup tačaka deli na dva podskupa jednake veličine (tačka prave se može uračunati u bilo koji od podskupova).

Rešenje

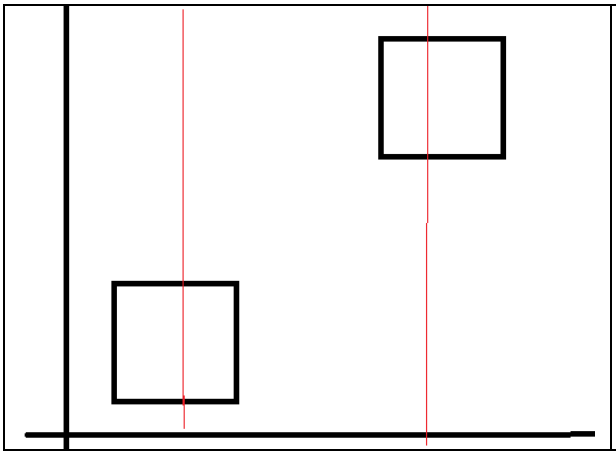
Jednačina prave *p*: y=ax + b
Jednačina prave *q* paralelne sa p: y=ax + b'

Za k=1..n rastojanje tačke A(x_k,y_k) od tačkeA'(x_k,ax_k + b') (A' je projekcija od A u vertikalnom smeru na pravu q u pravcu y-ose) glasi: d(A,A')=d_k=y_k-ax_k -b' ($d_k$je pozitivno/negativno ako je tačka iznad/ispod prave q)	

Da bi se zadovoljio zahtev iz formulacije onda se b' bira tako da polovina brojeva d_k bude veća ili jednaka od nule. Zapravo to znači da polovina brojeva y_k-ax_k će biti veća ili jednaka od b' . Otuda, za k=1..n se problem svodi na traženje medijane za y_k-ax_k,o čemu je bilo reči kod rangovskih statistika u petoj glavi knjige *Algoritmika* prof. Zivkovica.

8. Konstruisati algoritam linearne složenosti koji za datih *n* zgrada (datih svojim x,y koordinatama na mapi grada) i za datu tramvajsku prugu u vidu prave *p* pronalazi novu prugu paralelnu sa datom koja dati skup zgrada deli na dva podskupa jednake veličine (voditi racuna da ni stara ni nova pruga ne prolaze kroz zgradu).

9.



Dato je n<=100 disjunktnih kvadrata u ravni. Podeliti ravan (I kvadrant) vertikalnim linijama na 3 oblasti tako da sume površina kvadrata ili delova kvadrata u dobijenim oblastima budu jednake.
Napomena: Pretpostavka da su kvadrati u I kvadrantu i da su im strane paralelne koordinatnim osama. Svaki kvadrat je zadat kordinatama donjeg levog temena i dužinom stranice. Na standardni izlaz, u dva reda treba ispisati dva realna broja. Oni predstavljaju **X** koordinatu zamišljenih linija koje dele prvi kvadrant.

I način

Resenje: Upotreba binarne pretrage
Pri unosu donjeg levog temena kvadrata i stranica, izračuna max (najveća vrednost po x-osi do koje doseže desno teme nekog kvadrata), i ukupna površina svih kvadrata (kv).
Koristimo za određivanje prve i druge vertikalne prave funkciju binpretraga.

Binpretraga za prvu vertikalu za segment [0,max] će postaviti pravu(x1=binpretraga(0,max);), tako da levo od nje je aproksimativno trećina ukupne površine.
Koristimo binarnu pretragu tj. delimo segment [l,r] na pola, m=(l+r)/2, i izračunamo površinu levo od sredine i desno od koordinatnog početka.
a) Ako je rezultat veći od trećine ukupne površine, onda se poziva funkcija sa parametrima bin1(l,m);
b) U suprotnom, poziva se funkcija bin1(m,r);
c) Kraj rekurzije je kad se postigne tačnost r-l<=0.001.

```
#include <stdio.h>
#include <math.h>

#define eps 1e-8
#define eps2 1e-4
#define KVADRAT(x) ((x)*(x))

#define MAXN 100
#define MAXX 30000


int n ;
double x [MAXN], y [MAXN], d [MAXN] ;
double ukupno = 0.0 ;

double NadjiPovrsinu(double cx) {
  double povrsina = 0.0 ; int i;
  for (i = 0 ; i < n ; i++)
    if (cx >= x [i])
      if (cx <= x [i] + d [i]) povrsina += d [i] * (cx - x [i]) ;
      else povrsina += KVADRAT(d [i]) ;
  return povrsina ;
}

void binpretraga(double xl, double xr, double povrsina, double *rx) {
    double tekPovrsina;
  while (xl< (xr-eps2)) {
    *rx = (xl + xr) / 2 ;
    tekPovrsina = NadjiPovrsinu(*rx) ;
    if (fabs(tekPovrsina- povrsina)<eps) break ;  //resenje!!!
    if (tekPovrsina < (povrsina-eps)) xl = *rx ;
    else xr = *rx ;
  }
}
```

```
int main() {
   int i; double x1, x2 ;
  scanf("%d", &n) ;
  for (i = 0 ; i < n ; i++)
  { scanf("%lf %lf %lf", x + i, y + i, d + i) ;
          ukupno += KVADRAT(d [i]) ;
  }


  binpretraga(0, MAXX, ukupno/3, &x1) ;
  binpretraga(0, MAXX, 2.0*ukupno/3, &x2) ;

  printf("%.2lf\n%.2lf\n", x1, x2) ;
  return 0 ;
}
```

II način
Resenje: Upotreba binarne pretrage
Pri unosu donjeg levog temena kvadrata i stranica, izračuna **max** (najveća vrednost po **x**-osi do koje doseže desno teme nekog kvadrata), i ukupna površina svih kvadrata (**kv**).
Koristimo za određivanje prve vertiklane prave funkciju bin1, a za određivanje druge vertikale funkciju bin2.

1. **bin1** za segment **[0,max] će** postaviti pravu**(x1=bin1(0,max);)**, tako da levo od nje je aproksimativno trećina ukupne površine.
Koristimo binarnu pretragu tj. delimo segment **[l,r]** na pola**, m=(l+r)/2,** i izračunamo površinu levo od sredine i desno od koordinatnog početka.

a) Ako je rezultat veći od trećine ukupne površine, onda se poziva funkcija sa parametrima **bin1(l,m);**
b) U suprotnom, poziva se funkcija **bin1(m,r);**
c) Kraj rekurzije je kad se postigne tačnost **r-l<=0.001.**

2. Slično, funkcija **bin2** za segment **[0,max],** postavlja pravu **x2:=bin2(0,max) tako da desno od nje** leži trećina ukupne površine.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAX 100

float a[MAX], x[MAX];

float bin1(float l, float r, int n, float kv)
{float p,m, rez;
  int i;
if (r-l<=0.001) rez=(l+r)/2;
          else {
              m=(l+r)/2;    p=0;
              for (i=0;i<n;i++)
               if (x[i]+a[i]<=m) p=p+(a[i]*a[i]);
                else if  ((x[i]<=m)  &&  (x[i]+a[i]>=m)) p=p+a[i]*(m-x[i]);

              if (kv<=3*p) rez=bin1(l,m,n,kv);
                      else rez=bin1(m,r,n,kv);
              }
 return rez;
}
```

```
float bin2(float l, float r, int n, float kv)
{
   float p, m, rez;
  int i;
if (r-l<=0.001) rez=(l+r)/2;
            else {
                m=(l+r)/2;
                p=0;
                for (i=0;i<n;i++)
                 if (x[i]>=m) p=p+a[i]*a[i];
                  else if ((x[i]<=m)  &&  (x[i]+a[i]>=m))p=p+a[i]*(x[i]+a[i]-m);
                 if (kv>=3*p) rez= bin2 (l, m, n, kv);
                            else rez= bin2 (m, r, n, kv);
            }

 return rez;
}


int main()
{
  int  i,n;
  float y,kv=0,x1,x2,max;
  scanf("%d",&n);
  max=0;
  for (i=0;i<n;i++)
  {
     scanf("%f%f%f", &x[i],&y,&a[i]);
     if (x[i]+a[i]>max) max=x[i]+a[i];
     kv=kv+a[i]*a[i];
  }

   x1=bin1(0,max,n,kv);
   x2=bin2(0,max,n,kv);
   printf("%f -- %f\n", x1,x2);
   return 0;
}
```

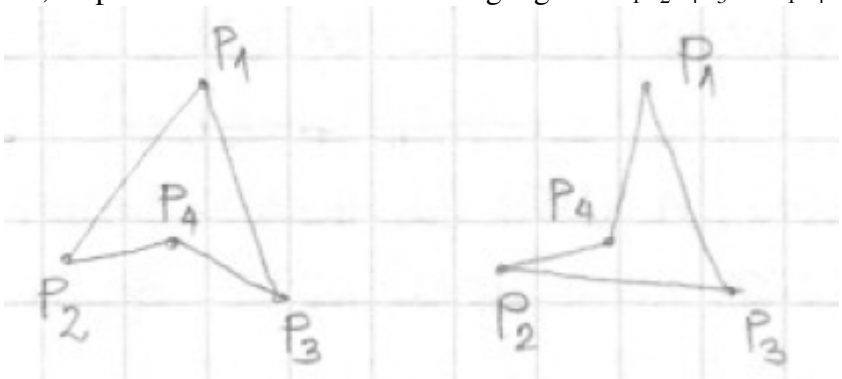
10.
U ravni je dato N pravougaonika sa dva naspramna temena. Poznato je da su stranice prevougaonika paralelne koordinatnim osama. Svi pravougaonici su zatvorene oblasti (dakle, sadrže svoju granicu). Smatra se da se pravougaonici seku ako imaju bar jednu zajedničku tačku (prema tome i na granici). Odrediti koliko likova nastaje spajanjem povezanih pravougaonika u jedan lik.

Rešenje. U rešenju se koristi funkcija Presek koja proverava da li se dva pravougaonika seku da bi se primenom DFS algoritma (pretrage po dubini) svi povezani pravougaonici spojili u jedan lik.

```
#include <iostream.h>
const int n=10;

struct TRect
{
double x1,y1,x2,y2;
}Pravougaonik[n]={
{10,5,20,20},{12,12,15,15},{8,20,21,21},{5,5,2,2},{4,4,1,1},{5,5,2,1},{12,12,10,10},{12,12,13,13},{1,3,3,5},
{2,2,5,5} };

bool used[n]; /* niz markera koji se koristi u DFS algoritmu*/

double min(double a, double b)
```

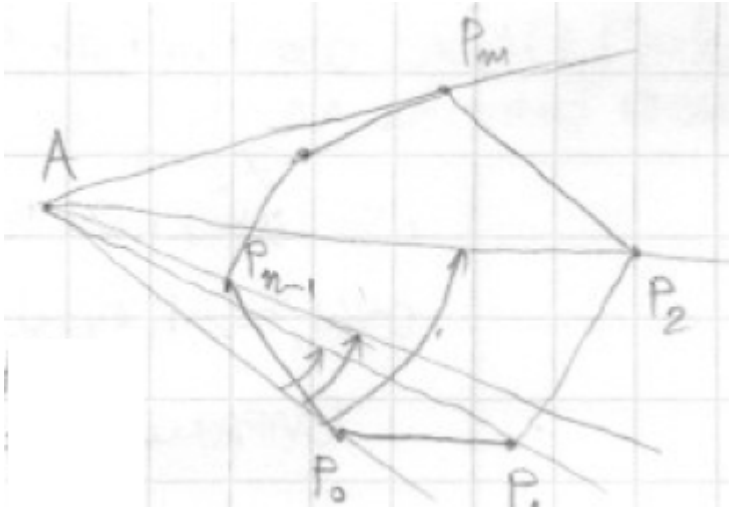
```
{return (a>b)? b:a; }
double max(double a, double b)
{return (a>b)? a:b; }

void Uredi(TRect &A)//(x1,y1)-donji levi ugao, (x2,y2)-gornji desni ugao
{
  double t;
       if (A.x1>A.x2) {t=A.x1; A.x1=A.x2; A.x2=t;}
       if (A.y1>A.y2) {t=A.y1; A.y1=A.y2; A.y2=t;}
}

bool Presek(TRect A, TRect B) // da li se pravougaonici A, B seku
{
  Uredi(A);
  Uredi(B);
   return (max(A.x1,B.x1)<=min(A.x2,B.x2)) && (max(A.y1,B.y1)<=min(A.y2,B.y2));
}

void DFS(int i) // pretraga po dubini
{
   used[i]=true;
    cout << i << " ";
   for (int k=0;k<n; k++)
        if (Presek(Pravougaonik [i], Pravougaonik [k]) && !used[k]) DFS(k);
}
void Ispis() // ispis likova
{
for (int i=0; i<n; i++) used[i]=false;
int k=0;
for (int i=0; i<n; i++)
if (!used[i])
{
  cout << "Lik " << ++k << "-> ";
   DFS(i);
   cout << endl;
}
}
void main()
{Ispis();}
```

11. Neka je S proizvoljan skup tačaka u ravni. Da li postoji samo jedan mnogougao sa skupom temena S?
REŠENJE:
Ne, na primer možemo formirati mnogouglove $P_1P_2P_4P_3$ i $P_1P_4P_2P_3$



12. Dat je konveksan mnogougao $P_0P_1 \ldots P_{n-1}$ i tačka A van njega. Kako se mogu konstruisati dve prave oslonca mnogougla kroz tačku A algoritmom složenosti $O(\log n)$? Prava oslonca konveksnog mnogougla je prava koja sadrži barem jednu tačku mnogougla i sve ostale tačke mnogougla nalaze se sa iste strane prave.

REŠENJE:

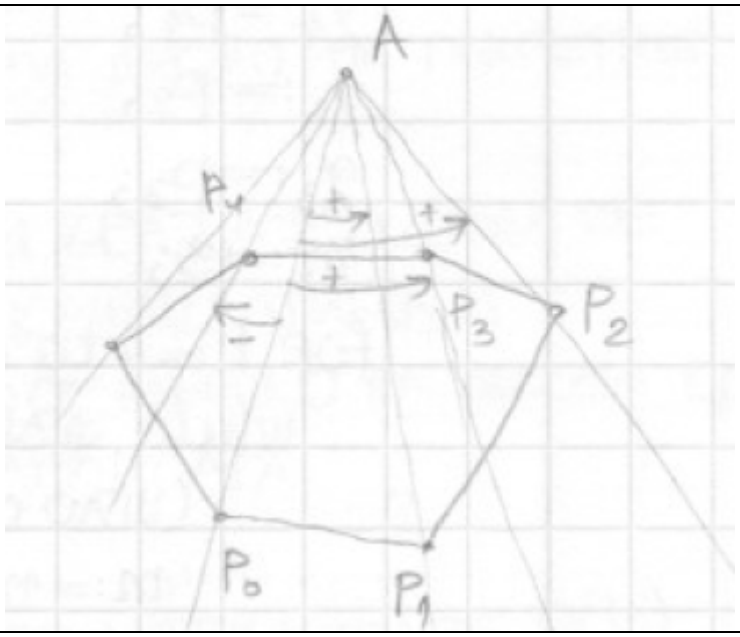


Obeležimo ugao $\alpha_i = \angle P_0 A P_i$, i=0..n-1
Važi da je $-\pi < \alpha_i < \pi$ i neka je $\alpha_n = 0$

Ako su svi uglovi istog znaka, onda je prava AP_0 prava oslonca, a druga prava oslonca APm odredjena je temenom Pm za koje ugao α_m ima najvecu apsolutnu vrednost.

Kako nalazimo Pm?
Pm nalazimo binarnom pretragom intervala 1,2,...,n-1 jer $|\alpha_i|$ najpre raste, pa potom opada.
U dve srednje tacke i,i+1 posmatra se razlika $\alpha_{i+1} - \alpha_i$ i ako je ta razlika:
veca od 0 => m>=i+1
manja od 0 => m<i
jednaka 0 => $\alpha_{i+1} = \alpha_i$ => m=i, ali prava oslonca sadrzi celu stranicu PmPm+1

Ako nisu svi uglovi α_i istog znaka, onda postoje dve mogucnosti:
Uglovi rastu, opadaju, pa rastu
Uglovi opadaju, rastu, pa opadaju

Binarnom pretragom nalazimo dva susedna temena sa uglovima suprotnog znaka. Oni razdvajaju interval indeksa na dva podintervala i u njima se, tražeći maksimalni ugao po apsolutnoj vrednosti, binarnom pretragom nalazi po jedna prava oslonca.	

13. a) Za tačku *P* u ravni se kaže da *dominira* nad tačkom *Q* ako su *x* i *y* koordinata tačke *P* veće od odgovarajućih koordinata tačke *Q*. Tačka *P* je *maksimalna* u datom skupu tačaka *S* ako ni jedna tačka iz *S* ne dominira nad njom. Konstruisati algoritam složenosti *O (n* log *n*) za nalaženje svih maksimalnih tačaka datog skupa *S* od *n* tačaka.

b) Rešiti odgovarajući problem u tri dimenzije (definicija dominacije će se odnositi na sve tri koordinate).

Rešenje

a) Mogu u skupu i sve tačke da budu maksimalne.
Ako su sve tačke različite, mora bar jedna da bude maksimalna (SP: ako tačka nije max, onda postoji neka koja njom dominira, pa je ona max ili ne ,...)

Lema 1: Ako tačka P nije max, onda njom dominira neka max tačka.

Lema2: Ako skupu tačaka S dodamo novu tačku R koja ne dominira niti jednom od starih max tačaka za k=1..n: M_{12},..., M_k iz S, onda one ostaju maksimalne, a nova tačka R maksimalna ako njom ne dominira nijednaod starih maksimalnih tačaka.

Lema3: Ako ima više tačaka sa istom x koordinatom,onda eventualno maksimalna tačka može biti samo ona sa najvećom y koordinatom.

Opis algoritma:

1. Sortiramo u opadajućem poretku sve tačke po vrednosti x kooordinate. Ukoliko ima više tacaka sa istom x koordinatom, zadržavamo samo onu sa najvećom y koordinatom. (Lema 3)
2. Pretpostavimo da je dobijen niz P1, P2,..., Pk, k<=n

Tacka P1 (sa najvećom x koordinatom) je sigurno maksimalna tačka.

3. Neka je Y vrednost y koordinate tačke P1.
4. Prema lemi 1 i lemi 2 tačka P2 je maksimalna akko njome ne dominira tačka P1, tj. ako je y koordinata tačke P2 veća od Y.
5. Ako je njena y koordinata veća od Y, onda se tačka P2 dodaje skupu maksimalnih tačaka, dok Y uzima vrednost y koordinate tačke P2.
6. Analogno dalje: tačka P3 je max akko njome ne dominiraju do tada otkrivene max tačke, tj. akko je y koordinata tačke P3 veća od Y.
7. Opisana procedura ponavlja se za i=4, 5, ..., k

b)

1. slično delu a) izvrši se sortiranje tacaka po opadajućim x koordinatama.
2. Za tačke se cuvaju u sortiranom redosledu projekcije nadjenih maksimalnih tacaka u ravni yOz
3. Ako nad projekcijom nove tačke dominira neka projekcija eliminiše se nova tačka.

Inace, nova tačka eliminiše neke od projekcija nakon umetanja svoje projekcije.
U delu 3 se za proveru dominantnih projekcija može koristiti binarna pretraga.

14. Dat je skup intervala na pravoj, predstavljenih koordinatama krajeva. Konstruisati algoritam složenosti $O(n \log n)$ za nalaženje svih intervala koji se sadrže u nekom drugom intervalu iz skupa.

Rešenje

Algoritam IntervaliPresek (parovi koordinata levih i desnih krajeva duži)

Ulaz: (l1,r1), ..., (ln,rn) /*parovi koordinata levih i desnih krajeva duži*/

Izlaz: (lk,rk)
/*parovi koordinata levih i desnih krajeva duži koje se sadrže u drugoj duži */
Ideja: duž nije sadržana u drugoj duži ako je levi kraj duži manji od
levog kraja druge duži
ili je desni kraj duži veći od desnog kraja druge duži.
POJAČANA IH: U skupu sa manje od n duži umemo da odredimo i
označimo duži koje su sadržane u nekoj drugoj duži istog skupa i
umemo da odredimo do tada najveći desni kraj

Opis algoritma:

1. sortiranje duži u rastućem poretku po njihovim levim krajevima
(ako dve duži imaju isti levi kraj, dodatno se sortiraju po
desnim krajevima u opadajućem poretku)

2. u prvom koraku ne biva markirana prva duž (jer nije sadržana ni
u jednoj duži, jer desni krajebi duži su u opadajućem poretku ako im je jednak
levi kraj), a do tada najveći desni kraj duži je vrednost desnog kraja
prve duži

3. pretpostavimo da je problem rešen za prvih n-1 duži (tako što su markirane sve
duži koje su sadržane u nekoj drugoj duži i da je poznat do tada najveći
desni kraj za tih n-1 duži)

4. Razmotrimo n-tu duž:
Ako je njen desni kraj veći od do tada najvećeg desnog kraja, onda ona nije
sadržana ni u jednoj duži, te je ne markiramo i njen desni kraj stavimo za
do tada najveći desni kraj.
U suprotnom, ta duž je sadržana u drugoj duži, te je markiramo.

(BSO pretpostavimo da ne postoje duži čiji su i levi i desni krajevi jednaki).
Nakon primene algoritma markirane su duži koje su sadržane u drugoj.

15. Neka je P prost (ne obavezno konveksni) mnogougao sadržan u datom pravougaoniku R i neka je q proizvoljna tačka unutar pravougaonika R. Konstruisati algoritam složenosti $O(n \log n)$ za odredjivanje takve tačke r van pravougaonika R za koju važi da je broj ivica mnogougla P koje seče duž $q - r$ minimalan.

16. Data su dva konveksna mnogougla cikličkim rasporedom svojih temena. Konstruisati algoritam linearne vremenske složenosti za određivanje preseka ovih mnogouglova. Izlaz (takođe konveksan mnogougao) treba da bude predstavljen ciklički uredjenom listom temena.

Rešenje

1. Sva temena sortiramo prema x koordinatama
2. Kroz svako teme povlačimo vertiklanu pravu i na taj način delimo oba mnogougla na trapeze. (čija je osnova paralelna y-osi, trougao je specijalan slučaj trapeza).
3. U jednoj po jednoj vertikalnoj traci, pronalaze se za const vreme (jer oba trapeza imaju najviše 4 temena) preseci po 2 trapeza i za vreme O(n) odgovarajuće preseke objedinjujemo u mnogougao.

17. Na ceremoniji otvaranja Svetskog kupa postoji plesna tačka gde mnoga deca širom sveta pokušavaju da kreiraju veliki krug na poljani. Taj krug predstavlja simbol toleranicije i multikulturalnog prijateljstva. Oni su uspeli da naprave savršeni krug, ali pošto deca nisu baš marljivo vežbala, onda nisu bili ravnomerno raspoređeni po krugu. Vi ste to brzo uočili i želite da znate koje je minimalno rastojanje između neka dva deteta.

Ulaz

Prva linija standardnog ulaza sadrži broj N koji predstavlja broj dece. Svaka od sledećih N linija sadrži dva realna broja zaokružena na dva decimalna mesta – koordinate svakog deteta.
Garantuje se da će sve tačke biti na krugu.

Izlaz

Prva i jedina linija standardnog ulaza treba da sadrži jedan realan broj (zaokružen na dva decimale) – Euklidsko rastojanje između dva najbliža deteta. Euklidsko rastojanje između tačke $(x1, y1)$ i $(x2, y2)$ je: $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$

Primer

Ulaz:	Izlaz:
5 1.00 4.00 -0.50 -1.60 4.00 1.00 3.12 3.12 -1.60 -0.50	1.56

Objašnjenja primera

Deca na tačkama sa koordinatama $(-0.50, -1.60)$ i $(-1.60, -0.50)$ su najlbliža i rastojanje među njima je 1.56 .

Ograničenja

$2 \leq N \leq 10^5$

Sve koordinate će biti u intervalu $[-10^6, 10^6]$

Vremensko ograničenja: **0.5 s**
Memorijsko ograničenje: **16 MB**

Tehnika pokretne linije

Postoji niz problema (nalaženje preseka horizontalnih i vertikalnih duži, nalaženje najvećeg pravougaonika u histogramu,...) koji se rešavaju tzv. tehnikom pokretne linije.

Ideja SWEEP LINE algoritama je *skenirati* geometrijske objekte, geometrijska ponašanja od značaja i slično pomicanjem zamišljene linije preko njih. Pomicanjem linije prelazimo preko geometrijskih objekata određenim redosledom i tim redosledom rešavamo problem. Onaj potproblem preko kog je prešla linija je rešen.

18.

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.

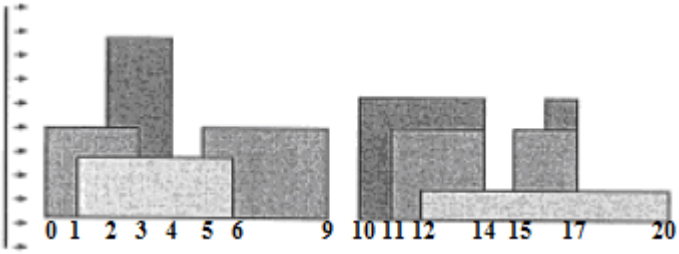
ULAZ

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 ($0<=a<b<=10^9$, $1<=c<=10^9$). Ta tri broja opisuju pojedinačni pravougaonik: a je početna x koordinata pravougaonika, b je krajnja *x* koordinata, dok je *c* visina.

IZLAZ

Ispisati koliku povrsinu slike prekrivaju soliteri (pravougaonici).

ULAZ	IZLAZ
9	59
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	



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 dogadjaje, onda cemo prvi dogadjaj notirati kao (a,c), a drugi cemo notirati kao (b,-c). Visina je uvek pozitivna, te ce negatovna vrednost –c oznacavati da se radi o desnoj ivici pravougaonika, a ne o levoj ivici.

Dogadjaje sortiramo i obilazimo sleva nadesno. Ako je prosli dogadjaj bio na *x* koordinati *prosliX*, a trenutni je na koordinati *a*, tada znamo da se svi pravougaonici cija 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;
}
```

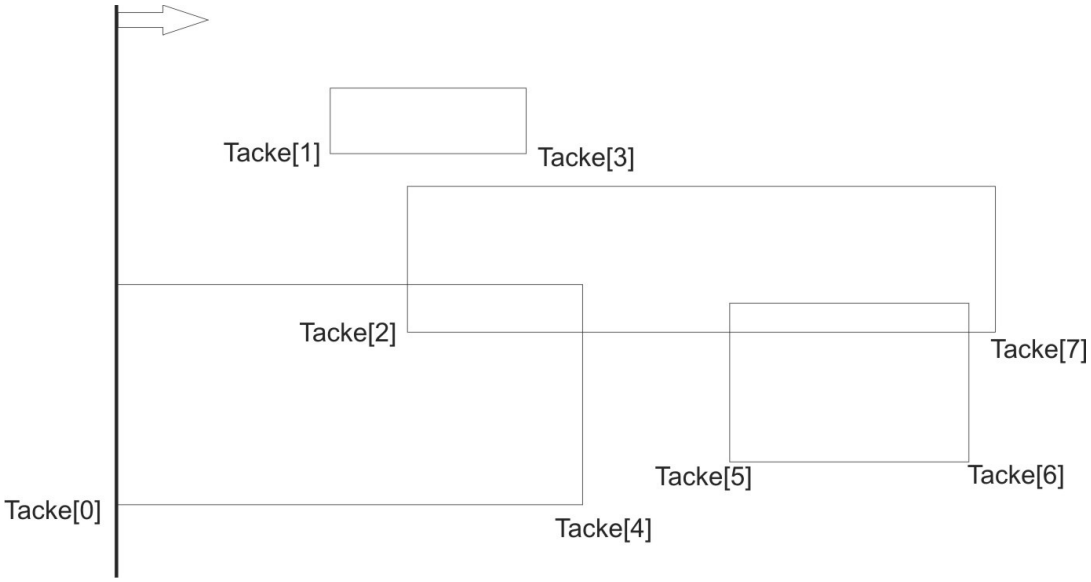
Prostorna složenost: O(n)
Vremenska složenost: O(n log n) jer se nad multisetom obavlja n operacija koje imaju složenost O(log n).

Inače, čak i da koordinate na ulazu programa nisu celobrojne, zadatak bismo mogli rešiti na isti način.
Na sličan način se može rešiti i problem nalaženja najvećeg pravougaonika u histogramu.

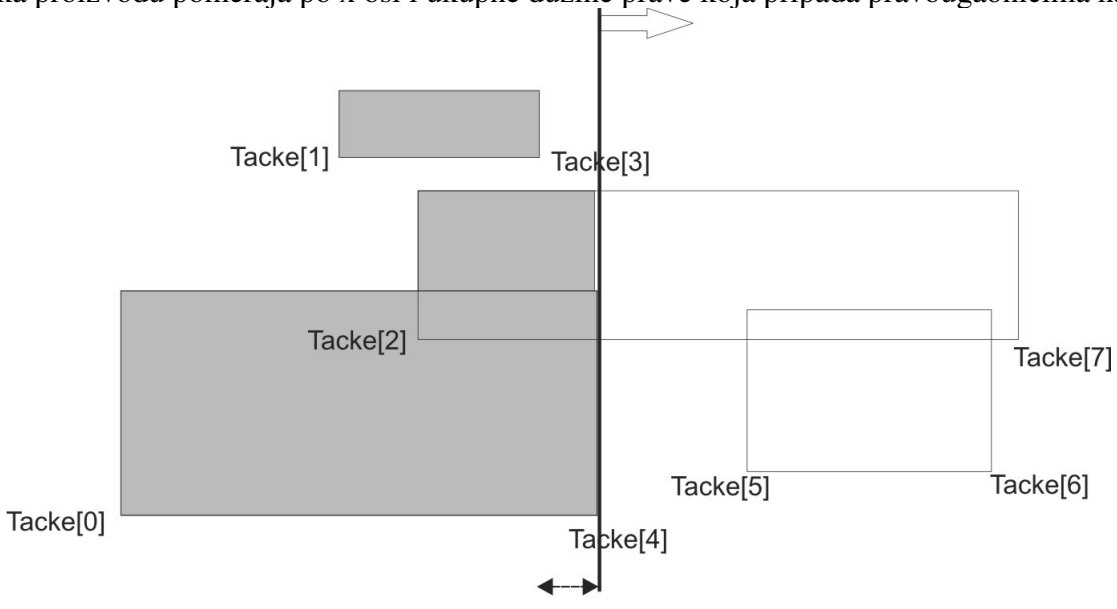
Uopšteniji problem: Unija pravougaonika

Problem površine unije pravougaonika se pojavljuje u mnogim zadacima, pa stoga postoji i dosta ideja za njegovu realizaciju. Line sweep algoritam ce nam pomoći da rešimo ovaj problem u složenosti O(NlogN), gde N predstavlja maksimalni raspon kordinata po samo jednoj osi.

Najpre ubacimo sva donja leva i gornja desna temena u niz i to, prvima dodajemo atribut znak = 1, a drugima znak=-1. Soriramo taj niz, najpre po x, a zatim po znaku . Sada postavimo pravu na koordinatu x, prve tačke niza Tacke.



Sada pomerimo pravu do sledeceg člana niza Tačke. Time ćemo 'prebrisati' neku oblast. Površina te oblasti će biti jednaka proizvodu pomeraja po x osi i ukupne dužine prave koja pripada pravougaonicima na tom



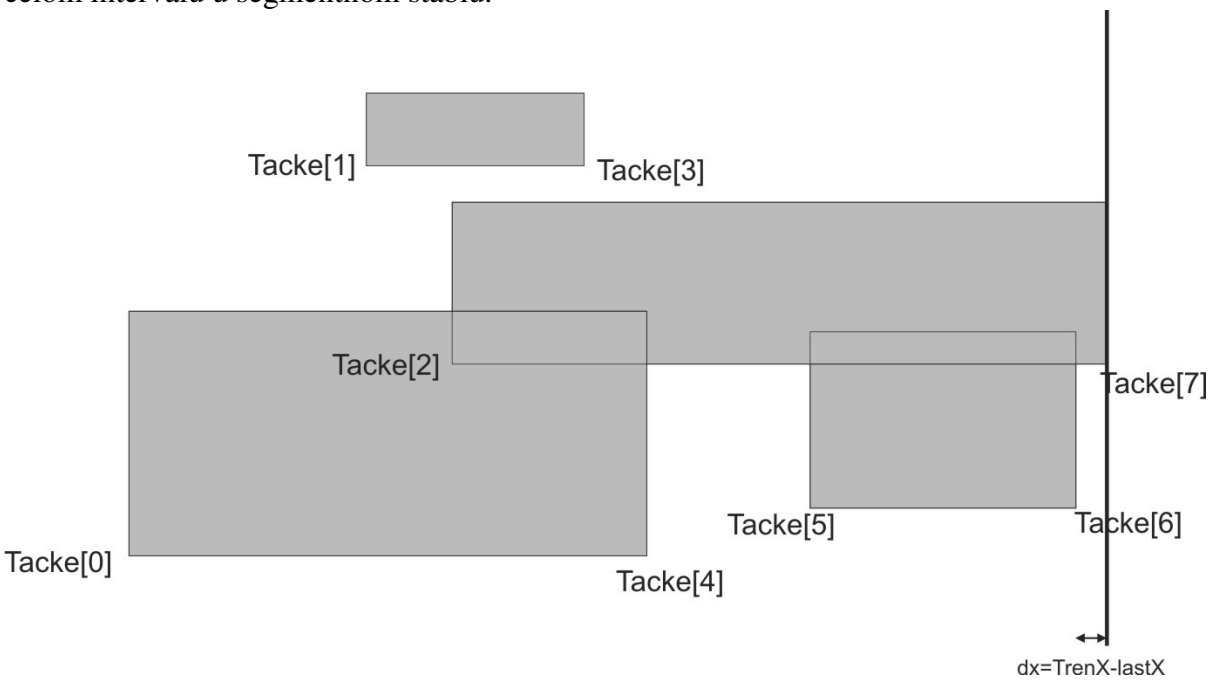
intervalu.

Sada, razmotrimo kako možemo da implementiramo ovu 'pravu'. Nama zapravo, samo treba struktura koja će moći da izvrši sledeće operacije :

- ObeležiDaPravaUlaziUPravougaonik(početak,kraj)

- ObrišiDeoPraveKojiJeUlazioUPravougaonik(početak,kraj) (potrebno ako smo stigli do kraja jednog pravougaonika, tada ne želimo da naša prava više ostavlja 'trag' tog pravouganioka.)
- dužinaPraveuPravougaonicima()

Struktura podataka koja nam je potrebna je segmentno stablo. Medjutim, ni samo poznavanje segmentnog stabla nije dovoljno, već je potrebno znati i lazy propagation metodu, kojom je moguće 'update'ovati vrednosti celom intervalu u segmentnom stablu.



Implementacija ove ideje i nije toliko zahtevna ukoliko se dobro poznaju navedene metode.

Pri svakom potezu pomeranja 'prave' mi radimo sledeće korake.
- Povrsina= Povrsina + PomerajPoX*dužinaPraveuPravougaonicima.
- Izmeni Segmento stablo u intervalu (y0,y1) za vrednost x[i].znak.

19. **Provera kolinearnosti**

Dato je N (3<=N<=26) tačaka u ravni nazivom i koordinatama. Kreirajte program koji pronalazi sve trouglove čija tri temena mogu biti date tačke.
ULAZ: Prva linija standardnog ulaza sadrži broj tačaka N. U sledećih N redova standardnog ulaza nalazi se naziv, apcisa i ordinata svake tačke. Apcisa i ordinata su celi brojevi iz intervala (–2000, 2000).
IZLAZ
Ispisati u svakoj liniji standardnog izlaza nazive temena koji obrazuju trouglove. Temena svakog trougla ispisati u leksikografskom poretku. Ako zadate tačke ne obrazuju niti jedan trougao ispisati poruku Nema trouglova.

ULAZ	IZLAZ
5 A 0 0 B 5 0 C 0 4 D 10 0 E 0 8	A B C A B E A C D A D E B C D B C E B D E C D E

REŠENJE:
Važno je uočiti da za date dimenzije ulaznih parametra N, X, Y, možemo na izlazu imati relativno veliki broj trojki.
IDEJA:

proveriti uređene trojke tačaka (i,j,k) i svka trojka tačaka koja nije kolinearna obrazuje trougao
Vremenska složenost provere kolineranosti za tri tačke je O(1)

Resenje 1:

```
#include <iostream>
using namespace std;

bool kolinearne(int x0, int y0, int x1, int y1, int x2, int y2)
{ int a1=x1-x0, a2=y1-y0;
 int b1=x2-x0, b2=y2-y0;
 return a1*b2==a2*b1;
}

int main()
{
 char imeTacke[26];
 int x[26], y[26];

 int n;
 cin >> n;
 for(int i=0; i<n; i++)
        cin >> imeTacke[i] >> x[i] >> y[i];

 bool nadjeno = false;
 for(int i=0;i<n;i++)
 for(int j=i+1; j<n; j++)
 for(int k=j+1; k<n; k++)
    if(!kolinearne(x[i],y[i],x[j],y[j],x[k],y[k]))
        { cout <<imeTacke[i]<<" "<< imeTacke[j]<<" "<< imeTacke[k]<<endl;
          nadjeno = true;
        }

 if(!nadjeno) cout << "Nema trouglova." << endl;
 return 0;
}
```

Resenje 2: **// provera orijentacija tačke u odnosu na pravu pomoću vektorskog proizvoda**

```
 #include <iostream>
using namespace std;

struct point{
 int x;
 int y;
 char i;
};

point tacke[32];

int n,br=0;

///*provera orijentacije tacke C u odnosu na pravac određen tačkama A, B
double orientacija(point a,point b,point c){
 return a.x*b.y+a.y*c.x+b.x*c.y-(c.x*b.y+c.y*a.x+b.x*a.y);
}

bool pr(int a,int b,int c){
 return orientacija(tacke[a],tacke[b],tacke[c])!=0;
```

```
}

int main(){
 cin>>n;
 for (int i=0;i<n;++i){
    cin>>tacke[i].i;
    cin>>tacke[i].x;
    cin>>tacke[i].y;
 }

 for (int i=0;i<n;++i){
    for (int j=i+1;j<n;++j){
       for (int k=j+1;k<n;++k){
          if (pr(i,j,k)){cout<<tacke[i].i<<" "<<tacke[j].i<<" "<<tacke[k].i<<"\n";++br;}
        }
    }
 }

if (br==0){cout<<"Nema trouglova.\n";}
 return 0;
}
```

20. Date su koordinate 3 nekolinearne tačke, A, B, C, u ravni i to redom koordinate: x_A, y_A, x_B, y_B, x_C, y_C. Sve koordinate su realni brojevi iz segmenta $[-10^6, 10^6]$

Napisati program koji sa standardnog ulaza učitava koordinate u datom redosledu i pronalazi tačku D na pravcu određenom duži AB, tako da D je smeštena na najmanjem rastojanju prema tački C. Proogram treba da na standardni izlaz ispise celobrojni deo rastojanja izmedju tacaka C i D.

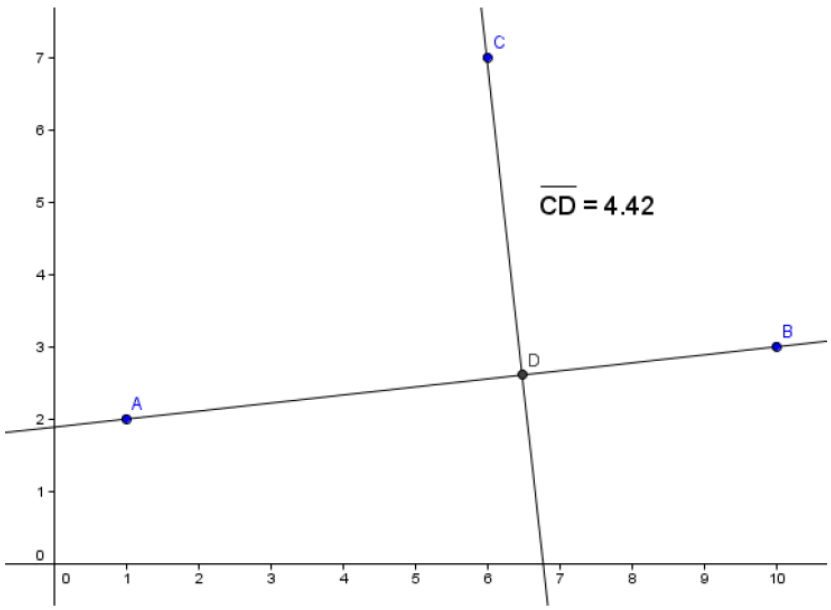
Primer

ULAZ:

```
1 2 10 3 6 7
```

IZLAZ:

```
4
```



Rešenje:

```
#include<iostream>
#include<cmath>
```

```
using namespace std;

struct dpoint {double x; double y;};
dpoint p[4];

double norm(dpoint p1, dpoint p2)
{
 return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}


double ccw(dpoint p0, dpoint p1, dpoint p2)
{
 double dx1=p1.x-p0.x; double dy1=p1.y-p0.y;
 double dx2=p2.x-p0.x; double dy2=p2.y-p0.y;
 return dx1*dy2-dy1*dx2;
}

int compute_p()
{
  double acx=p[0].x-p[2].x;
  double acy=p[0].y-p[2].y;
  double bax=p[1].x-p[0].x;
  double bay=p[1].y-p[0].y;

  double t = (-acy*bay-acx*bax)/(bax*bax+bay*bay);

  p[3].x=p[0].x+t*(p[1].x-p[0].x);
  p[3].y=p[0].y+t*(p[1].y-p[0].y);

  double c0 = ccw(p[2],p[3],p[0]);
  double c1 = ccw(p[2],p[3],p[1]);
  if(c0*c1<0) return 3;
  if(fabs(c0)<fabs(c1)) return 0; else return 1;

}

int main()
{ for(int i=0; i<3; i++) cin >> p[i].x >> p[i].y;
  int ir=compute_p();
  cout << (int)norm(p[2],p[ir]) << endl;
}
```

21.
Dat je skup tačaka u ravni.
Treba da nađemo dva pravougaonika kojima su stranice paralelne sa koordinatnim osama, gde se jedan pravougaonik nalazi potpuno levo od drugog, i sve tacke se nalaze u jednom od ta dva pravougaonika. Naći takav par pravougaonika sa minimalnom zbirnom povrsinom.
ULAZ
Prvi red sadrži broj tačaka N.
Narednih N redova sadrže X i Y kordinatu za svaku tačku.
Dato je da ne postoje 2 tačke sa istom X koordinatom.
IZLAZ
Broj koji odgovara zbiru površina dva najmanja pravougaonika koji sadrže sve tačke.
PRIMER

ULAZ

5 1 1 2 5 3 3 4 4 5 2
IZLAZ 8

- Vremensko ograničenje: **1 s**
- Memorijsko ograničenje: **1000 mb**