Pregătirea lotului național de informatică
Alba-Iulia 2004

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

Arbori de intervale ("segment trees")
şi aplicații in Geometria Computațională

Problema 1

Se considera **N≤50.000** segmente in plan dispuse paralel cu axele OX/OY. Sa se determine care este numărul total de intersecții intre segmente.

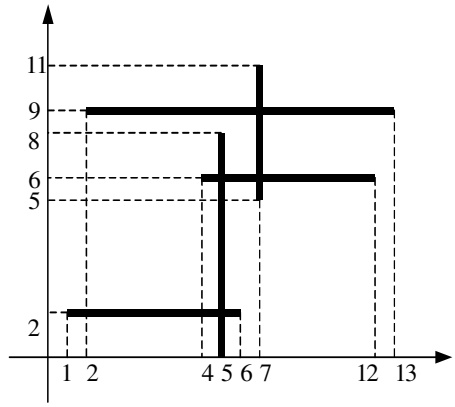
In fişierul „**segment.in**" se găseşte pe prima linie numărul **N** de segmente, iar pe fiecare din următoarele **N** linii cate patru numere naturale mai mici decât **50.000,** reprezentând coordonatele carteziene ale extremităților fiecărui segment.

Rezultatul se va scrie in „**segment.out**".

Timp de execuție: 1 secunda / test

Exemplu:

segment.in	segment.out
5	4
2 9 13 9	
4 6 12 6	
1 2 6 2	
5 0 5 8	
7 5 7 11	



Algoritmi de „baleiere" (*line sweeping*)

Folosind cunoştințe generale de geometrie analitica se poate obține un algoritm **O(N²)** dar acesta nu se va încadra in limita de timp.

Pentru rezolvarea acestei probleme vom folosi o tehnica cunoscuta sub numele de „baleiere" (*sweeping*) care este comuna multor algoritmi de geometrie computațională. In *baleiere,* o *dreapta de baleiere* verticala, imaginara, traversează mulțimea obiectelor geometrice, de obicei, de la stânga la dreapta. Baleierea oferă o metoda pentru ordonarea obiectelor geometrice, de obicei, plasându-le intr-o structura de date, pentru obținerea relațiilor dintre ele.

Algoritmii de baleiere, de obicei, gestionează doua mulțimi de date:

1. *Starea liniei de baleiere* da relația dintre obiectele intersectate de linia de baleiere

2. *Lista punct-eveniment* este o secvenţa de coordonate *x*, ordonate de la stânga la dreapta de obicei, care definesc pozițiile de oprire ale dreptei de baleiere; fiecare astfel de poziție de oprire se numeşte *punct eveniment;* numai in punctele eveniment se întâlnesc modificări ale stării liniei de baleiere

Pentru unii algoritmi lista punct-eveniment este determinata dinamic in timpul execuției algoritmului.

Soluție problema 1

Vom deplasa o dreapta de baleiere verticala, imaginara de la stânga la dreapta. Lista punct-eveniment va conține capetele segmentelor orizontale si ce fel de tip sunt (cap stânga sau cap dreapta) si segmentele verticale. Pe măsura ce ne deplasam de la stânga la dreapta, când întâlnim un capăt stâng inseram capătul in stările dreptei de baleiere si când întâlnim un capăt drept ştergem capătul din stările dreptei de baleiere. Când întâlnim un segment vertical, numărul de intersecții ale acestui segment cu alte segmente orizontale va fi dat de numărul capetelor de intervale care se afla in stările dreptei de baleiere cuprinse intre coordonatele *y* ale segmentului vertical.

Astfel, stările dreptei de baleiere sunt o structura de date pentru care avem nevoie de următoarele operații:

- **`INSEREAZA(y)`** : inserează capătul *y*
- **`STERGE(y)`** : şterge capătul *y*
- **`INTEROGARE(`y_1`,`y_2`)`** : întoarce numărul de capete cuprinse in intervalul [y_1, y_2]

Fie **`MAXC`** valoarea maxima a coordonatelor capetelor de segmente. Folosind un vector pentru a implementa aceste operațiile descrise mai sus vom obține o complexitate **`O(1)`** pentru primele doua operații si **`O(MAXC)`** pentru cea de-a treia. Astfel, complexitatea va fi **`O(N*MAXC)`** in cazul cel mai defavorabil. Putem comprima spațiul **`[0…MAXC]`** observând ca doar maxim **`N`** din valori din acest interval contează, si anume capetele segmentelor orizontale, astfel reducând a treia operație la **`O(N)`**, dar algoritmul va avea complexitatea **`O(N`**^{2}**`)`**, ceea ce nu aduce nici o îmbunătățire fata de algoritmul trivial.

Aceasta situație ne indemna sa căutam o structura de date mai eficienta. O prima varianta ar fi împărțirea vectorului in bucăți de $\sqrt{N}$ reducând complexitatea totala la **`O(N*`**$\sqrt{}$**`N)`**. In continuare vom prezenta o structura de date care oferă o complexitate logaritmica pentru operațiile descrise mai sus.

Arbori de intervale

Un arbore de intervale este un arbore binar in care fiecare nod poate avea asociata o structura auxiliara(anumite informații). Dându-se doua numere întregi **`st`** si **`dr`**, cu **`st<dr`**, atunci arborele de intervale **`T(st,dr)`** se construieşte recursiv astfel:

- consideram rădăcina **`nod`** având asociat intervalul **`[st,dr]`**
- daca **`st<dr`** atunci, vom avea asociat subarborele stâng **`T(st,mij)`**, respectiv subarborele drept **`T(mij+1,dr)`**, unde **`mij`** este mijlocul intervalului **`[st,dr]`**

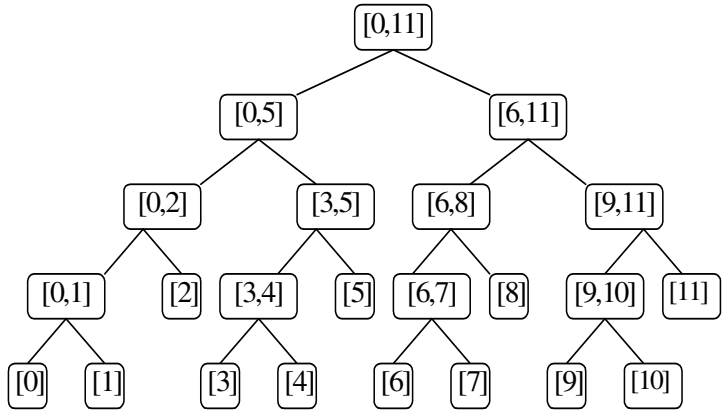
Intervalul **`[st,dr]`** asociat unui nod se numeşte *interval standard*. Frunzele arborelui sunt considerate *intervale elementare*, ele având lungimea **`1`**.

Proprietate:

Pregătirea lotului naţional de informatică
Alba-Iulia 2004

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

Un arbore de intervale este un arbore binar echilibrat(diferenţa absoluta intre adâncimea subarborelui stâng si subarborelui drept este cel mult **1**). Astfel adâncimea unui arbore de intervale care conţine **N** intervale este **$[\log_2 N]+1$**.



Operaţii efectuate asupra unui arbore de intervale

- *Actualizare unui interval intr-un arbore de intervale*

Vom prezenta pseudocodul unei proceduri recursive care inserează un interval **[a,b]** intr-un arbore de intervale **T(st,dr)** cu rădăcina in nodul **nod**. Cea mai eficienta metoda de stocare in memorie a unui arbore de intervale este sub forma unui vector folosind aceeaşi codificare a nodurilor ca la *heap*-uri.

```
procedura ACTUALIZARE(nod, st, dr, a, b)
   daca (a ≤ st) si (dr ≤ b) atunci
      modifica structura auxiliara din nod
   altfel
      mij = (st+dr)/2
      daca (a <= mij) atunci ACTUALIZARE(2*nod, st, mij, a, b)
      daca (b > mij) atunci ACTUALIZARE(2*nod+1, mij+1, dr, a, b)
      actualizează structura auxiliara din nodul nod
```

- *Interogarea unui interval intr-un arbore de intervale*

Vom prezenta pseudocodul unei proceduri recursive care returnează informaţiile asociate unui interval **[a,b]** intr-un arbore de intervale **T(st,dr)** cu rădăcina in nodul **nod**.

```
procedura INTEROGARE(nod, st, dr, a, b)
   daca (a ≤ st) si (dr ≤ b) atunci
      returnează structura auxiliara din nod
   altfel
      mij = (st+dr)/2
      daca (a <= mij) atunci INTEROGARE(2*nod, st, mij, a, b)
      daca (b > mij) atunci INTEROGARE(2*nod+1, mij+1, dr, a, b)
      returnează structura auxiliara din fiul stâng si din fiul drept
```

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

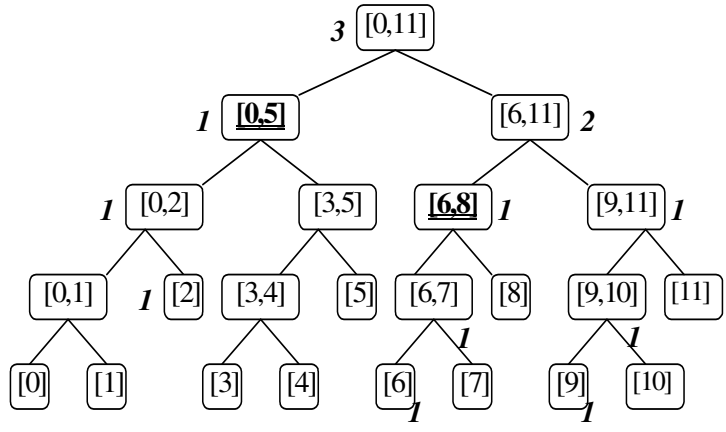
Vom demonstra in continuare ca operațiile prezentate mai sus au complexitatea $O(\log_2 N)$ pentru un arbore de N intervale. Este posibil ca intr-un nod sa aibă loc apel atât in fiul stâng cat si in cel drept. Acest lucru produce un cost adițional doar prima data când are loc. După prima „rupere in doua", oricare astfel de „rupere" nu va aduce cost adițional, deoarece unul din fii va fi mereu inclus complet in intervalul $[a,b]$. Cum înălțimea arborelui este pentru N intervale este $[\log_2 N]+1$ complexitatea operațiilor va fi tot $O(\log_2 N)$.

Pentru a retine in memorie un arbore de intervale pentru N valori, vom aveam de nevoie de $N+N/2+N/4+N/8\ldots=2*N-1$ locații de memorie (sunt $2*N-1$ noduri). Deoarece arborele nu este complet, trebuie verificat de fiecare data daca fii unui nod exista in arbore (aceasta verificare a fost omisa in pseudocodul de mai sus), altfel s-ar încerca accesarea de valori din vector care nu exista. Daca memorie disponibila in timpul concursului este suficienta, se poate declara vectorul care retine arborele de intervale de lungime 2^K astfel încât $2^K \geq 2*N-1$, simulând astfel un arbore complet si nefiind necesare verificările menționate mai sus.

Soluție problema 1 (continuare)

Vom folosi un arbore de intervale pentru a simula operațiile, care le făceam înainte pe un vector obişnuit, in timp logaritmic. Astfel, in fiecare nod din arborele din intervale vom retine cate capete exista in acel interval. Primele doua operații vor fi implementate folosind procedura **ACTUALIZARE()** de mai sus pentru intervalul $[y,y]$ in arborele $T(0, MAXC)$ si adunând $+1$, respectiv -1 la fiecare nod actualizat. Cea de-a treia operație poate fi realizata folosind procedura **INTEROGARE()** pe intervalul $[y_1,y_2]$. Astfel complexitatea se reduce la $O(N*\log_2 MAXC)$ Folosind aceeaşi tehnica de „comprimare" a coordonatelor se poate obține o complexitate $O(N*\log_2 N)$.

In figura urmatoare este descrisa structura arborelui de intervale, dupa actualizarea pentru intervalele [2,2], [6,6] si [9,9]. Sunt marcate intervalele care conduc la obtinerea numarului de segmente intersectate de primul segment vertical, obtinute in urma interogarii pe intervalul [0,8]



Problema 2

Preluata de la IOI 1998 (Ziua 2)

Pregătirea lotului național de informatică
Alba-Iulia 2004

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

Se considera **N≤50.000** dreptunghiuri in plan fiecare având laturile paralele cu axele OX/OY. Lungimea marginilor (contururilor) reuniunii tuturor dreptunghiurilor se va numi perimetru. Sa se calculeze perimetrul celor **N** dreptunghiuri.

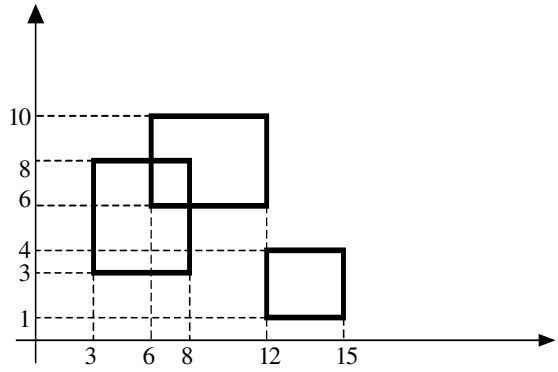
In fişierul „**drept.in**" se găseşte pe prima linie numărul **N** de dreptunghiuri, iar pe fiecare din următoarele **N** linii cate patru numere naturale mai mici ca **50.000**, reprezentând coordonatele carteziene ale coltului stânga sus, respectiv dreapta jos ale fiecărui dreptunghi.

Rezultatul se va scrie in fişierul „**drept.out**".

Timp maxim de execuție: 1 secunda / test

Exemplu:

drept.in	**drept.out**
3 **3 8 8 3** **6 10 12 6** **12 4 15 1**	**44**



Soluție problema 2

Folosind un raționament asemănător ca la prima problema, constatam necesitatea unui algoritm de baleiere. Vom descompune problema in doua parți: prima data calculam perimetrul folosind doar laturile stânga si dreapta ale dreptunghiurilor pentru a calcula perimetrul pe OY; apoi putem sa rotim dreptunghiurile si folosind aceeaşi procedura calculam perimetrul pe OX folosind doar laturile sus si jos ale dreptunghiurilor.

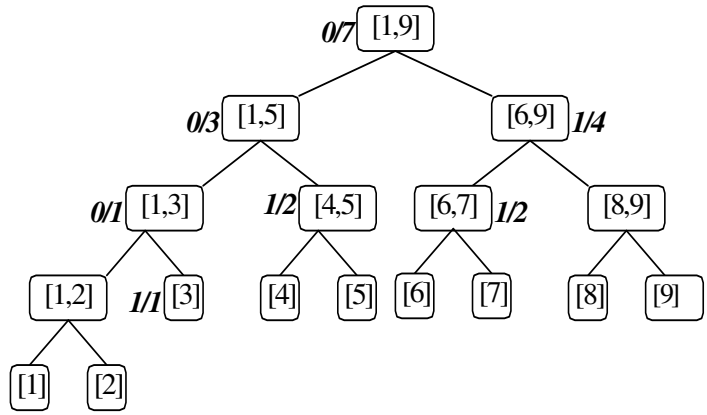
Fiecare latura (stânga sau dreapta) va fi un punct eveniment. Sortam laturile crescător după coordonata *x* si procedam de la stânga la dreapta. Când întâlnim o latura stânga marcam in stările dreptei de baleiere, intervalul de unitati ocupat de latura pe OY, iar când întâlnim o latura dreapta demarcam in stările dreptei de baleiere intervalul de unitati ocupat de latura. Intre oricare doua puncte eveniment consecutive are loc o modificare a perimetrului total pe OY. Astfel, după fiecare actualizare a stărilor dreptei de baleiere se va retine care este numarul de unitati marcate pana in prezent (nu se tine cont daca o unitate este marcata de mai multe ori). Vom aduna la perimetrul pe OY total de fiecare data, diferența absoluta intre numarul de unitati marcate pana in prezent si valoarea imediat anterioara (înaintea actualizării) a numărului de unitati marcate. Aşadar, este necesara o structura de date care se efectueze următoarele operații in timp eficient (preferabil logaritmic):

- **MARCHEAZA(a,b)** : marchează intervalul [*a*, *b*]
- **DEMARCHEAZA(a,b)** : demarchează intervalul [*a*, *b*]

- **`INTEROGARE()`** : returnează numărul total de coordonate marcate pana in prezent

Putem folosi un arbore de intervale pentru a obține o complexitate **`O(log₂ MAXC)`** sau **`O(log₂ N)`** pentru primele doua operații si **`O(1)`** pentru ce-a de treia. In fiecare nod din arbore reținem de cate ori a fost marcat intervalul respectiv si cate unitati din intervalul respectiv au fost marcate. Primele doua operații pot fi implementate folosind procedura **`ACTUALIZARE()`** iar a treia operația va returna valoarea numărului de unitati care au fost marcate din rădăcina arborelui de intervale.

In figura urmatoare este descrisa structura arborelui de intervale, dupa marcarea intervalelor [*3…8*] si [*6…10*] (un interval [y_1…y_2] reprezinta intervalul de unitati [y_1, y_2-1] din arbore).



Problema 3

Se dau **`N≤100.000`** puncte in plan de coordonate numere naturale mai mici ca **`2.000.000.000`**. Sa se răspundă la **`M≤1.000.000`** întrebări de forma „cate puncte din cele **`N`** exista in dreptunghiul cu coltul stânga sus in **`(x₁,y₁)`** si coltul dreapta jos in **`(x₂,y₂)`**?"

In fişierul „**`puncte.in`**" se va găsi pe prima linie numerele **`N`** si **`M`**. Pe următoarele **`N`** linii se găsesc coordonatele punctelor. Pe următoarele **`M`** linii se vor găsi cate patru numere naturale reprezentând coordonatele colturilor dreptunghiurilor.

In fişierul „**`puncte.out`**" se vor găsi **`M`** numere naturale reprezentând răspunsurile la întrebări.

Timp maxim de execuție: 1 secunda / test

Exemplu:

`puncte.in`	**`puncte.out`**
`6 1`	**`2`**
`2 8`	
`5 3`	
`6 5`	
`8 7`	
`8 1`	
`10 10`	
`4 8 9 4`	

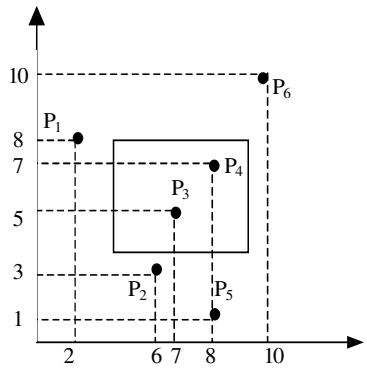
Soluție problema 3

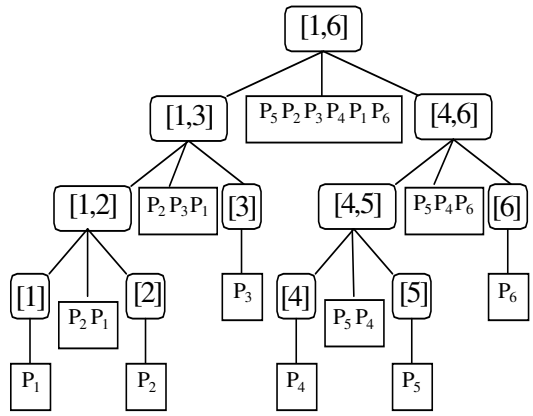
Pregătirea lotului național de informatică
Alba-Iulia 2004

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

Problema determinarii numarului de puncte din interiorul unui dreptunghi este o particularizare **2D** pentru problema numita in literatura de specialitate „Orthogonal Range Search". Varianta aleasa pentru acest caz, consta intr-un arbore de intervale, care permite determinarea numarului de puncte din interiorul unui dreptunghi cu o complexitate $O(\log_2^2 N)$.

Astfel, se sorteaza coordonatele **x** ale punctelor si se construieste un arbore de intervale. Primul nivel al arborelui va contine toate coordonatele **x**. Cei doi fii ai lui vor contine prima jumatate, respectiv a doua jumatate a punctelor (sortate dupa coordonata **x**) s.a.m.d. Pentru fiecare interval de coordonate **x**, se mentin sortate coordonatele **y** corespunzatoare punctelor care au coordonatele **x** in intervalul respectiv. Astfel, memoria folosita este $O(N*\log_2 N)$. Pentru a construi efectiv se foloseste o abordare asemanatoare algoritmului de sortare prin interclasare: in fiecare nod pentru a obtine lista de coordonate **y** ordonate se interclaseaza listelele celor doi fii (deja ordonate). Cand se ajunge intr-o frunza, lista nodului este formata dintr-un singur punct.



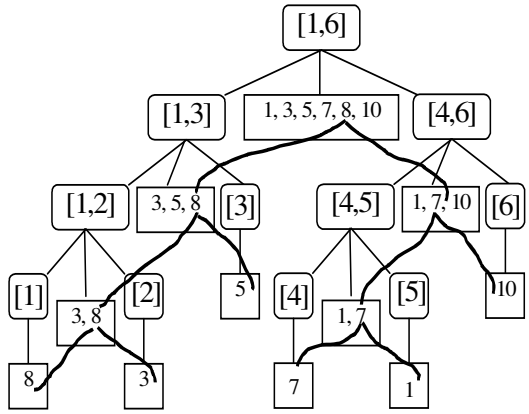


Pentru fiecare dreptunghi, se executa căutarea in arborele de intervale pentru segmentul $[x_1, x_2]$ (deoarece se folosesc doar cele **N** coordonate **x** ale punctelor, se vor potrivi capetele acestui interval folosind căutarea binara la cea mai apropiata coordonata **x** de fiecare capăt). Pentru fiecare interval din arbore atins, se executa doua căutări binare printre coordonatele **y** corespunzătoare coordonatelor **x** din acel interval, pentru a determina cate puncte din intervalul respectiv se afla in intervalul $[y_1, y_2]$ (adică in interiorul dreptunghiului).

Complexitatea $O(\log_2^2 N)$ poate fi redusa, folosind o metoda descoperita independent de *Willard* si *Lueker* in anul *1978*. Se observa ca pentru fiecare coordonata **y** dintr-un nod daca presupunem ca se efectuează o căutare binara, atunci putem determina in timpul interclasării poziția din fiecare fiu pe care o va returna căutarea binara pentru aceeaşi valoare. Este evident ca pentru o valoare **y** dintr-un nod, care a provenit dintr-un fiu in timpul interclasării, exista o unica valoare maxima **y'** din celalalt fiu astfel incat $y' \le y$. Aşadar, ținem pentru fiecare valoare **y** dintr-un nod doi indicii care identifica pozițiile corespunzătoare efectuării unei căutări binare in fii pentru valoarea **y**. Astfel, in timpul căutării in loc sa se efectueze căutări binare, se pot parcurge aceşti indici care oferă in $O(1)$ poziția in lista căutata, reducând complexitatea totala la $O(\log_2 N)$.

Pregătirea lotului național de informatică
Alba-Iulia 2004

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti



Probleme propuse

1) (*preluata de la Olimpiada Baltica de Informatica, Polonia 2001*)

Se considera **N≤50.000** dreptunghiuri in plan fiecare având laturile paralele cu axele OX/OY. Sa se calculeze aria ocupata de reuniunea celor **N** dreptunghiuri.

In fişierul „**drept2.in**" se găseşte pe prima linie numărul **N** de dreptunghiuri, iar pe fiecare din următoarele **N** linii cate patru numere naturale mai mici ca **50.000**, reprezentând coordonatele carteziene ale coltului stânga sus, respectiv dreapta jos ale fiecărui dreptunghi.

Rezultatul se va scrie in fişierul „**drept2.out**".

Timp maxim de execuție: 1 secunda / test

2) (*preluata de la Olimpiada Naționala de Informatica, Polonia 2001)*

Se considera **N≤50.000** puncte in plan de coordonate numere naturale mai mici ca **50.000**. Sa se determine unde se poate aşeza un dreptunghi de lungime **DX** si lățime **DY** astfel încât numărul de puncte incluse in dreptunghi sa fie maxim.

In fişierul „**puncte.in**" se găseşte pe prima linie numerele **N**, **DX** si **DY**. Pe următoarele **N** linii se găsesc coordonatele punctelor.

In fişierul „**puncte.out**" se vor găsi cinci numere naturale reprezentând coordonatele colturilor stânga sus si dreapta jos a aşezării dreptunghiului si numărul maxim de puncte din dreptunghi.

Timp maxim de execuție: 1 secunda / test

3) (*preluata de la Barajul pentru Lotul Național, România 2003)*

Se considera **N≤100.000** puncte in plan de coordonate numere naturale mai mici ca **100.000**. Sa se determine unde se pot aşeza doua dreptunghi de lungime **DX** si lățime **DY**, fara sa se intersecteze, astfel încât numărul de puncte incluse in cele doua dreptunghiuri sa fie maxim.

In fişierul „**puncte2.in**" se găseşte pe prima linie numerele **N**, **DX** si **DY**. Pe următoarele **N** linii se găsesc coordonatele punctelor.

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

In fişierul „**puncte2.out**" se vor găsi noua numere naturale reprezentând coordonatele colturilor stânga sus si dreapta jos a aşezării primului dreptunghi, respectiv a celui de-al doilea, cat si numărul maxim de puncte incluse in ambele dreptunghiuri.

Timp maxim de execuție: 1 secunda / test

4) (*preluata de la concursul internațional USACO January Open, 2004)*

Se considera **N≤250.000** dreptunghiuri in plan fiecare având laturile paralele cu axele OX/OY, care nu se intersectează si nu se ating, dar pot fi incluse unul in altul. Se numeşte „închisoare" un dreptunghiuri înconjurat de alte dreptunghiuri. Sa se determine numărul maxim de dreptunghiuri de care poate fi înconjurata o „închisoare" si cate astfel de „închisori" maxime exista.

In fişierul „**inchis.in**" se găseşte pe prima linie numărul **N** de dreptunghiuri, iar pe fiecare din următoarele **N** linii cate patru numere naturale mai mici ca **1.000.000.000**, reprezentând coordonatele carteziene ale coltului stânga sus, respectiv dreapta jos ale fiecărui dreptunghi.

In fişierul „**inchis.out**" se găsesc doua numere naturale: numărul maxim de dreptunghiuri de care poate fi înconjurata o „închisoare" si cate astfel de „închisori" maxime exista.

Timp maxim de execuție: 1 secunda / test

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

Surse la problemele prezentate

```
{/* Sursa Pascal Problema "Segment"
  * Complexitate: O(N * lg MAXC)
  */}
type entry =record
      x, y1, y2, sgn:integer;
     end;
     sir = array[0..50000]of entry;

var a:sir;
    i, n, x1, y1, x2, y2, v : word;
    T : array[0..131072]of word;
    Res: Int64;

function cmp (i, j : entry):integer;
begin
    if i.x <> j.x then cmp:=i.x-j.x
    else cmp:=j.sgn - i.sgn;
end;

function query (n,l,r,a,b :word):word;
var m, tt:word;
begin
 tt := 0;
 if (a <= l)and(r <= b) then tt:=T[n]
 else begin
   m := (l + r) div 2;
   if (a <= m) then
     tt:=tt+query(2 * n, l, m, a, b);
   if (b > m) then
   tt:=tt+query(2*n+1, m+1, r, a, b);
 end;
    query := tt;
end;

procedure update(n, l, r, p,v: word);
var m : word;
begin
 m := (l + r) div 2;
 inc(T[n],v);
 if (l < r)  then begin
   if (p <= m) then
      update(2 * n, l, m, p, v)
   else
      update(2*n + 1, m + 1, r, p, v);
 end;
end;

procedure QuickSort(var A: sir; Lo, Hi: Integer);
procedure Sort(l, r: Integer);
var i, j: integer; x, y : entry;
begin
  i := l; j := r; x := a[(l+r) DIV 2];
  repeat
   while cmp(a[i],x)<0 do i:= i + 1;
   while cmp(x,a[j])<0 do j:= j - 1;
   if i <= j then begin
      y:=a[i]; a[i]:=a[j];a[j] := y;
      i := i + 1; j := j - 1;
   end;
  until i > j;
  if l < j then Sort(l, j);
  if i < r then Sort(i, r);
end;
begin  Sort(Lo,Hi); end;
```

```
{/* Sursa C++ Problema "Segment"
  * Complexitate: O(N * lg MAXC)
  */}
#include <stdio.h>
#include <stdlib.h>

struct entry
{
    int x, y1, y2, sgn;
} A[50000];
typedef struct entry entry;

int N, T[131072];
long long Res;

int cmp(const void *i,const void *j)
{
   entry *ei = (entry *) i, *ej =
   (entry *) j;

    if (ei->x != ej->x)
       return ei->x - ej->x;
    return ej->sgn - ei->sgn;
}

int query(int n, int l, int r, int a, int b)
{
    int m, t = 0;

   if (a <= l && r <= b)
       return T[n];
   else
   {
    m = (l + r) / 2;
    if (a <= m)
     t += query(2 * n, l, m, a, b);
    if (b > m)
     t +=query(2*n+1,m+1,r, a, b);
    }
    return t;
}

void update(int n, int l, int r, int p, int v)
{
    int m = (l + r) / 2;

    T[n] += v;
    if (l == r) return;
    if (p <= m)
       update(2 * n, l, m, p, v);
    else
       update(2*n+1,m+1, r, p, v);
}

int main(void)
{
    FILE *f;
    int i, n, x1, y1, x2, y2;

    f = fopen("segment.in", "r");
    fscanf(f, "%d", &N);
    for (n = i = 0; i < N; i++)
    {
     fscanf(f, "%d %d %d %d", &x1, &y1, &x2, &y2);
```

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

```
begin
    assign(input, 'segment.in');
    reset(input); readln(n); v:=0;
    for i := 0 to N - 1 do begin
      readln(x1, y1, x2, y2);
      if (y1 = y2) then begin
        A[v].x := x1; A[v].sgn := +1;
        A[v].y1 := y1; inc(v);
        A[v].x := x2; A[v].sgn := -1;
        A[v].y1 := y1; inc(v); end
      else
        if (x1 = x2) then begin
          A[v].x := x1; A[v].sgn := 0;
          A[v].y1:=y1; A[v].y2 := y2;
          inc(v); end;
    end;
   close(input);QuickSort(A, 0, v-1);
   Res:=0;
   for i := 0 to v - 1 do begin
     if (A[i].sgn = 0)  then
Res:=Res+query(1,0,49999,A[i].y1,A[i].y2)
    else
 update(1,0,49999, A[i].y1,A[i].sgn);
end;
assign(output, 'segment.out'); rewrite(output);
writeln(Res); close(output); end.
```

```
      if (y1 == y2)
        A[n].x = x1, A[n].sgn = +1,
        A[n++].y1 = y1,
        A[n].x = x2, A[n].sgn = -1,
        A[n++].y1 = y1;
      else
        if (x1 == x2)
         A[n].x = x1, A[n].sgn = 0,
         A[n].y1=y1,A[n++].y2 = y2;
    }
    fclose(f);

    qsort(A, n, sizeof(entry), cmp);

    for (i = 0; i < n; i++)
    if (A[i].sgn == 0)
       Res += query(1, 0, 49999, A[i].y1,
A[i].y2);
    else
       update(1, 0, 49999, A[i].y1, A[i].sgn);

    f = fopen("segment.out", "w");
    fprintf(f, "%lld\n", Res);
    fclose(f);

    return 0; }
```

Sursa Pascal Problema „dreptunghi" v
{baleiere + arbori de intervale
Complexitate: O(N * lg MAXC) }

```
type edge = record
      x,y1,y2,sgn:integer;
     end;
     sir  = array[0..50000] of edge;
var  H,V: sir;
     Cnt, Sum:array[0..131072]of word;
     i, x1, y1, x2, y2, N: Word;

function cmp (i, j :edge):integer;
begin
    if i.x <> j.x then cmp:=i.x - j.x
    else cmp:=j.sgn - i.sgn;
end;

procedure update(n,l,r,a,b,c: word);
var m:word;
begin
 if (a <= l)and( r <= b) then
      inc(Cnt[n],c)
 else begin
  m := (l + r) div 2;
  if (a <= m) then
      update(2 * n, l, m, a, b, c);
  if (b > m) then
    update(2*n + 1, m+1, r, a, b, c);
 end;
 if Cnt[n]>0 then Sum[n]:=(r - l + 1)
 else Sum[n]:=Sum[2*n] + Sum[2*n + 1];
end;

procedure QuickSort(var A: sir; Lo,Hi: Integer);
procedure Sort(l, r: Integer);
var  i, j: integer; x, y : edge;
begin
  i := l; j := r; x := a[(l+r) DIV 2];
```

Sursa C++ Problema „dreptunghi"
{baleiere + arbori de intervale
Complexitate: O(N * lg MAXC) }

```
#include <stdio.h>

#include <stdlib.h>

#define abs(x)((x) < 0 ? (-(x)) :(x))

struct edge
{
    int x, y1, y2, sgn;
} H[100000], V[100000];
typedef struct edge edge;

int N, Cnt[131072], Sum[131072];

int cmp(const void *i, const void *j)
{
    edge *ei=(edge*) i, *ej=(edge*) j;
    if (ei->x != ej->x)
       return ei->x - ej->x;
    return ej->sgn - ei->sgn;
}

void update(int n, int l, int r, int a, int b, int
c)
{
   int m;
   if (a <= l && r <= b)
      Cnt[n] += c;
   else
   {
    m = (l + r) / 2;
    if (a <= m)
     update(2 * n, l, m, a, b, c);
    if (b > m)
     update(2*n+1,m+1, r, a, b, c);
```

```
  repeat
    while cmp(a[i],x)<0 do i := i + 1;
    while cmp(x,a[j])<0 do j := j - 1;
    if i <= j then begin
      y:=a[i]; a[i]:=a[j]; a[j]:=y;
      i := i + 1; j := j - 1;
    end;
  until i > j;
  if l < j then Sort(l, j);
  if i < r then Sort(i, r);
end;
begin  Sort(Lo,Hi); end;

function solve( e: sir) : integer;
var i, t, p:word;
 begin
 t := 0; P := 0;
 QuickSort(E, 0, 2*N-1);
 fillchar(Cnt, sizeof(Cnt),0);
 fillchar(Sum, sizeof(Sum),0);
 for i:= 0 to 2 * N - 1 do  begin
   update(1,0,49999,E[i].y1,E[i].y2-1, E[i].sgn)
   inc(P,abs(t - Sum[1]));
   t := Sum[1];
  end;
  solve:= P;
end;

Begin
 assign(input, 'drept.in');
 reset(input);
 readln(n);
 for i := 0 to N - 1 do begin
    readln(x1, y1, x2, y2);
    V[i].x := x1; V[i].y1 := y2;
    V[i].y2 := y1; V[i].sgn := +1;
    V[i+N].x:=x2; V[i + N].y1:=y2;
    V[i+N].y2:=y1; V[i+N].sgn:=-1;
    H[i].x:=y1; H[i].y1:=x1;
    H[i].y2:=x2; H[i].sgn := -1;
    H[i+N].x:=y2; H[i+N].y1:=x1;
    H[i+N].y2 := x2; H[i+N].sgn:=+1;
  end;
 close(input);
 assign(output,'drept.out');
 rewrite(output);
 writeln(solve(H) + solve(V));
 close(output);
end.
```

```
    }
    Sum[n] = Cnt[n] ? (r - l + 1) :
    (Sum[2 * n] + Sum[2 * n + 1]);
}

int solve(edge E[])
{
    int i, t = 0, P = 0;

    qsort(E,2*N, sizeof(edge), cmp);
    memset(Cnt, 0, sizeof(Cnt));
    memset(Sum, 0, sizeof(Sum));

    for (i = 0; i < 2 * N; i++)
        update(1, 0, 49999, E[i].y1, E[i].y2,
E[i].sgn),
        P += abs(t-Sum[1]),t=Sum[1];
    return P;
}

int main(void)
{
    FILE *f;
    int i, x1, y1, x2, y2;

    f = fopen("drept.in", "r");
    fscanf(f, "%d\n", &N);
    for (i = 0; i < N; i++)
    {
      fscanf(f, "%d %d %d %d",&x1,&y1, &x2, &y2);

        V[i].x = x1, V[i].y1 = y2,
        V[i].y2 = y1, V[i].sgn = +1;
        V[i+N].x=x2, V[i+N].y1 = y2,
        V[i+N].y2=y1, V[i+N].sgn = -1;

        H[i].x = y1, H[i].y1 = x1,
        H[i].y2 = x2, H[i].sgn = +1;
        H[i+N].x=y2,H[i+N].y1 = x1,
        H[i+N].y2=x2, H[i+N].sgn = -1;
    }
    fclose(f);

    f = fopen("drept.out", "w");
    fprintf(f, "%d\n", solve(V) + solve(H));
    fclose(f);

    return 0;
}
```

```
/* Sursa Pascal Problema "Puncte"
 * Rezolvare: arbori de intervale
* Complexitate: O(N*lg N) preprocesare + O(lg^2 N)
query */
type point =record x, y : integer; end;

type sir = array[0..50]of point;
     list = array[0..50]of integer;
     bilist = array[0..18] of list;
var P: sir; X:list; T : bilist;
    i, n, m, x1, y1, x2, y2, v : word;

function cmp (i, j : point):integer;
begin cmp:=i.x-j.x; end;

procedure build(lv, l, r:integer);
```

```
/* Sursa C++  Problema "Puncte"
* Rezolvare: arbori de intervale
* Complexitate: O(N*lg N) preprocesare + O(lg^2 N)
query */
#include <stdio.h>
#include <stdlib.h>
#define MAXN 100000
#define LGN  18
struct point
{ int x, y;
} P[MAXN];
typedef struct point point;
int N, M, X[MAXN], T[LGN][MAXN], x1, y1, x2, y2;

int cmp(const void *i, const void *j)
{
```

Pregătirea lotului naţional de informatică
Alba-Iulia 2004

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

```
var m,i,j,k:integer;
begin
  m := (l + r) shr 1 ;
  if (l = r) then T[lv][l] := P[l].y
  else begin
   build(lv+1,l,m); build(lv+1, m+1,r);
   i:=l; j:=m+1; k:=l;
   while (i<=m)or(j<=r) do
   if (j>r)or(i<=m)end(T[lv+1,i]<T[lv+1,j])
   then begin
     T[lv,k]:=T[lv+1][i];inc(k);inc(i);end
    else begin
     T[lv,k]:=T[lv+1,j];inc(k);inc(j);end
  end; end;

function search(A:list;l,r,v:integer):
                                    integer;
var t, step, p:integer;
begin
 if (v < A[l]) then search:=l-1;
 step:=1;
 while step<(r-l+1) do step:= step shl 1;
  p:=l;
  while step>0  do begin
    if (p+step<=r)end(A[p+step]<=v)then
                inc(p,step);
     step:= step shr 1;
  end; search:=p; end;

function query (lv, l, r:  word):word;
var m, tt:word;
begin
   tt := 0;
   if (x1 <= l) end (r <= x2) then begin
     if (y2 < T[lv][l])or(y1 > T[lv][r])
        then tt :=0
     else
     if (y1<T[lv][l])end(T[lv][r]<y2) then
        tt:=r-l+1
     else
       tt:=search(T[lv],l,r,y2)-
           search(T[lv],l,r,y1-1) end
  else begin
    m := (l + r) shr 1;
    if x1<=m then tt:=tt+query(lv+1, l, m);
    if x2>m then tt:=tt+query(lv+1,m+1, r);
  end;
  query:=tt; end;

procedure QuickSort(var A: sir; Lo, Hi: Integer);
procedure Sort(l, r: Integer);
var i, j: integer; x, y : point;
begin
  i := l; j := r; x := a[(l+r) DIV 2];
  repeat
    while cmp(a[i],x) < 0 do i := i + 1;
    while cmp(x,a[j]) < 0 do j := j - 1;
    if i <= j then begin
      y := a[i]; a[i] := a[j]; a[j] := y;
      i := i + 1; j := j - 1; end;
  until i > j;
  if l < j then Sort(l, j);
  if i < r then Sort(i, r);
end;
begin  Sort(Lo,Hi); end;

begin
 assign(input, 'puncte.in'); reset(input);
 assign(output, 'puncte.out');
 rewrite(output); readln(n, m);
 for i:=0 to N-1 do readln(P[i].x, P[i].y);
```

```
  return ((point *)i)->x-((point *)j)->x;
}

void build(int lv, int l, int r)
{
  int m = (l + r) >> 1, i, j, k;
  if (l == r) { T[lv][l] = P[l].y; return; }
 build(lv+1, l, m), build(lv + 1, m+1, r);
 for (i=l, j=m + 1, k=l; i<=m || j <= r; ) if (j>r
|| (i<=m && T[lv+1][i]<T[lv+1][j]))
      T[lv][k++] = T[lv + 1][i++];
    else
       T[lv][k++] = T[lv + 1][j++]; }

int search(int A[], int l, int r, int v)
{
    int p, step;
    if (v < A[l]) return l - 1;
    for (step=1; step<(r-l + 1);step<<= 1);
    for (p = l; step; step >>= 1)
    if (p + step <= r && A[p + step] <= v)
       p += step;
    return p;
}

int query(int lv, int l, int r)
{
    int m, t = 0;
    if (x1 <= l && r <= x2)
    {
       if (y2 < T[lv][l] || y1 > T[lv][r])
          return 0;
       if (y1 < T[lv][l] && T[lv][r] < y2)
          return (r - l + 1);
       t=search(T[lv], l, r, y2) -
         search(T[lv], l, r, y1 - 1);
   }
  else
 {m = (l + r) >> 1;
  if (x1 <= m) t += query(lv + 1, l, m);
  if (x2 > m) t += query(lv + 1, m + 1, r);
  }
  return t;
}

int main(void)
{
    FILE *fin, *fout;
    int i;
    fin = fopen("puncte.in", "r");
    fout = fopen("puncte.out", "w");
    fscanf(fin, "%d %d", &N, &M);
    for (i = 0; i < N; i++)
      fscanf(fin, "%d %d", &P[i].x, &P[i].y);
    qsort(P, N, sizeof(point), cmp);
    for (i = 0; i < N; i++) X[i] = P[i].x;
    build(0, 0, N - 1);
    for (; M; M--)
    {
     fscanf(fin, "%d %d %d %d", &x1, &y2, &x2,
&y1);
     if (x2 < X[0] || x1 > X[N - 1])
       fprintf(fout, "0\n");
     else
      {
   x1 = search(X, 0, N - 1, x1 - 1) + 1;
   x2 = search(X, 0, N - 1, x2);
 fprintf(fout, "%d\n", query(0, 0, N - 1));
       }
    }
    fclose(fin), fclose(fout);
```

```
 uickSort(P, 0, N-1);
 for i := 0 to N - 1 do  X[i]:= P[i].x;
 build(0, 0, N - 1);
 while M>0 do begin
  readln(x1, y2, x2, y1);
  if (x2<X[0])or(x1> X[N-1])then writeln(0)
  else begin
    x1 := search(X, 0, N - 1, x1 - 1) + 1;
    x2 := search(X, 0, N - 1, x2);
    writeln(query(0, 0, N - 1));
  end;
 dec(m);end;
close(output);end.
```

```
   return 0;
}
```

```
/* Sursa Pascal Problema "Puncte"
 * Rezolvare: arbori de intervale + rafinament
Willard-Lueker
 * Complexitate: O(N*lg N) preprocesare + O(lg N)
query */
type point =record x, y :integer; end;

type sir = array[0..50]of point;
 list = array[0..50]of integer;
 bilist = array[0..18]of list;
 trilist=array[0..18,0..50,0..2]of integer;
var P: sir; X:list; Ptr:trilist; T:bilist;
    i, n, m, x1, y1, x2, y2, v : word;

function cmp (i, j : point):integer;
begin cmp:=i.x-j.x; end;

procedure build(lv, l, r:integer);
var m,i,j,k:integer;
begin
 m := (l + r) shr 1 ;
 if (l = r) then T[lv][l] := P[l].y
 else begin
   build(lv+1,l,m); build(lv+1, m+1, r);
   i:=l; j:=m+1; k:=l;
   while (i<=m)or(j<=r) do
    if(j>r)or(i<=m)end(T[lv+1,i]<T[lv+1,j])
    then begin
     Ptr[lv][k][0] :=i; Ptr[lv][k][1]:=j-1;
     T[lv,k]:=T[lv+1][i];inc(k);inc(i); end
    else begin
     Ptr[lv][k][0]:=i-1;Ptr[lv][k][1] := j;
     T[lv,k]:=T[lv+1,j];inc(k); inc(j); end
  end; end;

function search(A:list;l,r,v:integer):
                                    integer;
var t, step, p:integer;
begin
 if (v < A[l]) then search:=l-1;
 step:=1;
 while step<(r-l+1) do step:=step shl 1;
 p:=l;
 while step>0  do begin
   if (p + step <= r) end (A[p+step]<=v)
           then inc(p,step);
   step:= step shr 1;
 end; search:=p; end;

function query(lv,l,r, p1, p2: word):word;
var m, tt:word;
begin
   tt := 0;
```

```
/* Sursa C++ Problema "Puncte"
 * Rezolvare: arbori de intervale + rafinament
Willard-Lueker
 * Complexitate: O(N*lg N) preprocesare + O(lg N)
query */

#include <stdio.h>
#include <stdlib.h>
#define MAXN 100000
#define LGN  18
struct point
{ int x, y;
} P[MAXN];
typedef struct point point;

int N, M, x1, y1, x2, y2;
int X[MAXN], T[LGN][MAXN], Ptr[LGN][MAXN][2];

int cmp(const void *i, const void *j)
{return ((point *)i)->x - ((point *)j)->x;
}

void build(int lv, int l, int r)
{
 int m = (l + r) >> 1, i, j, k;
 if (l == r) { T[lv][l] = P[l].y; return;}
 build(lv+1, l, m), build(lv+1, m+1, r);
 for (i=l, j=m+1, k=l; i <= m || j <= r; )
    if (j>r || (i <= m && T[lv + 1][i] < T[lv +
1][j]))
    Ptr[lv][k][0]=i, Ptr[lv][k][1]=j - 1,
    T[lv][k++] = T[lv + 1][i++];
    else
    Ptr[lv][k][0]=i-1, Ptr[lv][k][1] = j,
    T[lv][k++] = T[lv + 1][j++];
}

int search(int A[], int l, int r, int v)
{ int p, step;
  if (v < A[l]) return l - 1;
  for (step=1; step<(r - l + 1);step<<= 1);
    for (p = l; step; step >>= 1)
     if (p + step <= r && A[p + step] <= v)
       p += step;
    return p;
}

int query(int lv,int l,int r,int p1,int p2)
{
    int m, t = 0;
        if (x1 <= l && r <= x2)
    {
       if (p2 < l || p1 > r) return 0;
```

Prof. Dana Lica C.N. "I.L.Caragiale" Ploieşti

```
   if (x1 <= l) end (r <= x2) then begin
     if (p2 < p1)or(y1 > r) then tt :=0
     else
     if (p1 < l)end(r < p2)then  tt:=r-l+1
     else begin
       if T[lv][p1]=T[0][y1] then dec(p1);
       tt := p2 - p1;
     end; end
   else begin
    m := (l + r) shr 1;
   if (x1 <= m) then
    tt:=tt+query(lv+1,l,m,Ptr[lv][p1][0],
                          Ptr[lv][p2][0]);
   if (x2>m) then
    tt:=tt+query(lv+1,m+1,r,Ptr[lv][p1][1],
                          Ptr[lv][p2][1]);
   end;
   query:=tt;
end;

procedure QuickSort(var A: sir; Lo, Hi: Integer);
procedure Sort(l, r: Integer);
var i, j: integer; x, y : point;
begin
{…….}
end;

begin
 assign(input, 'puncte.in'); reset(input);
 assign(output, 'puncte.out');
 rewrite(output); readln(n, m);
 for i:=0 to N-1 do readln(P[i].x, P[i].y);
 QuickSort(P, 0, N-1);
 for i := 0 to N-1 do  X[i]:= P[i].x;
 build(0, 0, N - 1);
 while M>0 do begin
  readln(x1, y2, x2, y1);
  if (x2 < X[0])or(x1 > X[N-1])or
     (y2<T[0,0])or( y1>T[0,N-1]) then
         writeln(0)
 else begin
   x1:=search(X, 0, N - 1, x1 - 1) + 1;
   x2 := search(X, 0, N - 1, x2);
   y1:= search(T[0], 0, N-1, y1-1)+1;
   y2 := search(T[0],0, N - 1, y2);
   writeln(query(0, 0, N - 1,y1, y2));end;
 dec(m);
end; close(output); end.
```

```
       if (p1<l && r<p2) return (r-l + 1);
       if (T[lv][p1] == T[0][y1]) p1--;
       t = p2 - p1;
    }
    else
    {
     m = (l + r) >> 1;
    if (x1 <= m)
  t += query(lv + 1, l, m, Ptr[lv][p1][0],
  Ptr[lv][p2][0]);
    if (x2 > m) t += query(lv + 1, m + 1, r,
Ptr[lv][p1][1], Ptr[lv][p2][1]);
    }
    return t;
}

int main(void)
{
    FILE *fin, *fout;
    int i;

    fin = fopen("puncte.in", "r");
    fout = fopen("puncte.out", "w");
    fscanf(fin, "%d %d", &N, &M);
    for (i = 0; i < N; i++)
      fscanf(fin, "%d %d",&P[i].x,&P[i].y);
    qsort(P, N, sizeof(point), cmp);
    for (i = 0; i < N; i++) X[i] = P[i].x;
    build(0, 0, N - 1);
    for (; M; M--)
    {
      fscanf(fin, "%d %d %d %d", &x1, &y2, &x2,
&y1);
      if (x2 < X[0] || x1 > X[N - 1] || y2 <
T[0][0] || y1 > T[0][N -
1])
          fprintf(fout, "0\n");
    else
     {
     x1 = search(X, 0, N-1, x1-1)+1,
     x2 = search(X, 0, N-1, x2);
     y1 = search(T[0],0,N-1,y1-1) + 1,
     y2 = search(T[0], 0, N - 1, y2);
fprintf(fout,"%d\n", query(0,0,N-1,y1,y2));
       }
    }
    fclose(fin), fclose(fout); return 0;}
```