Material Geometria 2017

Faculdade de Computação
Universidade Federal de Uberlândia

Sumário

Capítulo 1

Geometria

1.1 Área da união de retângulos $O(nlog^2n)$

Listagem 1.1: Área da união de retângulos (N $\log^2 N$)

```
struct seg{
    ll y, tipo;
    seg(){}
    seg(ll a, ll b){
        y = a;
        tipo = b;
    }
    bool operator < (seg other) const{
        return y > other.tipo;
    }
};

struct event{
    ll x, y, y0, tipo;
    event(){}
    event(ll a, ll b, ll c, ll d){
        x = a;
        y = b;
        y0 = c;
        tipo = d;
    }
    bool operator < (event other) const{
        return x < other.x;
    }
};

ll ABRE = 0;
ll FECHA = 1;

multiset<pair<ll, ll>, greater<pair<ll, ll> > > ativo;
vector<event> evento;

ll sweep(){
    ll area=0;
    ll x=0, x1=0;
```

```
36      event eve;
37      ll cnt = 0;
38      ll U=0;
39      ll D=0;
40      //~ ll ans = 0;
41
42      for (ll i = 0; i < evento.size(); i++)
43      {
44          eve = evento[i];
45          cnt = 0;
46          x1 = eve.x;
47          //~ prllf("%s em x=%d\n", eve.tipo==ABRE ? "abre" : "fecha", eve.x
                );
48          for(auto it : ativo){
49              //~ oioi;
50              if(it.S == ABRE){
51                  if(cnt==0) U = it.F;
52                  cnt++;
53              }else if(it.S==FECHA){
54                  if(cnt==1){
55                      D = it.F;
56                      //~ prllf("x = %d  x1 = %d  U = %d  D = %d\n", x, x1,
                            U, D);
57                      area += (x1-x)*(U-D);
58                  }
59                  cnt--;
60              }
61          }
62
63          if(eve.tipo == ABRE){
64              ativo.insert(mp(eve.y0, ABRE));
65              ativo.insert(mp(eve.y, FECHA));
66          }else{
67              //~ for(auto i : ativo){
68                  //~ if(i.F==eve.y && i.S==ABRE) ativo.erase(i);
69                  //~ if(i.F==eve.y0 && i.S==FECHA) ativo.erase(i);
70              //~ }
71              ativo.erase(ativo.find(mp(eve.y0, ABRE)));
72              ativo.erase(ativo.find(mp(eve.y, FECHA)));
73          }
74          x = x1;
75          //~ x1 = eve.x;
76      }
77
78      return area;
79  }
80
81  int main () {
82      ll n, x, y, larg, alt, x0, y0;
83      cin >> n;
84      for (ll i = 0; i < n; i++)
85      {
86          cin >> x >> y >> x0 >> y0;
87          evento.pb(event(x, y, y0, ABRE));
88          evento.pb(event(x0, y, y0, FECHA));
89      }
90      sort(evento.begin(), evento.end());
91
92      ll area = sweep();
```

```
93      //~ cout << "Area total = " << area << endl;
94      cout << area << endl;
95
96      return 0;
97  }
```

1.2 Área da união de retângulos $O(nlogn)$

Listagem 1.2: Área da união de retângulos (N log N)

```
1  #define P(x,y) make_pair(x,y)
2  using namespace std;
3  class Rectangle{
4      public:
5      int x1 , y1 , x2 , y2;
6      static Rectangle empt;
7      Rectangle(){
8          x1=y1=x2=y2=0;
9      }
10     Rectangle(int X1 , int Y1 , int X2 , int Y2){
11         x1 = X1; y1=Y1;
12         x2 = X2; y2=Y2;
13     }
14     Rectangle intersect(Rectangle R){
15         if(R.x1 >= x2 || R.x2 <= x1) return empt;
16         if(R.y1 >= y2 || R.y2 <= y1) return empt;
17         return Rectangle(max(x1 , R.x1) , max(y1 , R.y1) , min(x2 , R.x2)
               , min(y2 , R.y2));
18     }
19 };
20 struct Event{
21     int x , y1 , y2 , type;
22     Event(){}
23     Event(int x , int y1 , int y2 , int type):x(x) , y1(y1) , y2(y2) ,
          type(type){}
24 };
25 bool operator < (const Event&A , const Event&B){
26     //if(A.x != B.x)
27     return A.x < B.x;
28     //if(A.y1 != B.y1) return A.y1 < B.y1;
29     //if(A.y2 != B.y2()) A.y2 < B.y2;
30 }
31 const int MX=(1<<17);
32 struct Node{
33     int prob , sum , ans;
34     Node(){}
35     Node(int prob , int sum , int ans):prob(prob) , sum(sum) , ans(ans){}
36 };
37 Node tree[MX*4];
38 int interval[MX];
39 void build(int x , int a , int b){
40     tree[x] = Node(0 , 0 , 0);
41     if(a==b){
42         tree[x].sum+=interval[a];
43         return;
```

```
44      }
45      build(x*2 , a , (a+b)/2);
46      build(x*2+1 , (a+b)/2+1 , b);
47      tree[x].sum = tree[x*2].sum + tree[x*2+1].sum;
48  }
49  int ask(int x){
50      if(tree[x].prob) return tree[x].sum;
51      return tree[x].ans;
52  }
53  int st , en , V;
54  void update(int x , int a , int b){
55      if(st>b || en<a) return;
56      if(a>=st && b<=en){
57          tree[x].prob+=V;
58          return;
59      }
60      update(x*2 , a , (a+b)/2);
61      update(x*2+1 , (a+b)/2+1 , b);
62      tree[x].ans = ask(x*2) + ask(x*2+1);
63  }
64  Rectangle Rectangle::empt = Rectangle();
65  vector < Rectangle > Rect;
66  vector < int > sorted;
67  vector < Event > sweep;
68  void compressncalc(){
69      sweep.clear();
70      sorted.clear();
71      for(auto R : Rect){
72          sorted.push_back(R.y1);
73          sorted.push_back(R.y2);
74      }
75      sort(sorted.begin() , sorted.end());
76      sorted.erase(unique(sorted.begin() , sorted.end()) , sorted.end());
77      int sz = sorted.size();
78      for(int j=0;j<sorted.size() - 1;j++)
79          interval[j+1] = sorted[j+1] - sorted[j];
80      for(auto R : Rect){
81          sweep.push_back(Event(R.x1 , R.y1 , R.y2 , 1));
82          sweep.push_back(Event(R.x2 , R.y1 , R.y2 , -1));
83      }
84      sort(sweep.begin() , sweep.end());
85      build(1,1,sz-1);
86  }
87  long long ans;
88  void Sweep(){
89      ans=0;
90      if(sorted.empty() || sweep.empty()) return;
91      int last = 0 , sz_ = sorted.size();
92      for(int j=0;j<sweep.size();j++){
93          ans+= 1ll * (sweep[j].x - last) * ask(1);
94          last = sweep[j].x;
95          V = sweep[j].type;
96          st = lower_bound(sorted.begin() , sorted.end() , sweep[j].y1) -
                sorted.begin() + 1;
97          en = lower_bound(sorted.begin() , sorted.end() , sweep[j].y2) -
                sorted.begin();
98          update(1 , 1 , sz_-1);
99      }
100 }
```

```
int main(){
//    freopen("in.in","r",stdin);
    int n;
    scanf("%d",&n);
    for(int j=1;j<=n;j++){
        int a , b ,c  , d;
        scanf("%d %d %d %d",&a,&b,&c,&d);
        Rect.push_back(Rectangle(a , b , c , d));
    }
    compressncalc();
    Sweep();
    cout<<ans<<endl;
}
```

1.3 Área da união de retângulos $O(nlogn)$ - Outra impl.

Listagem 1.3: Área da união de retângulos (N log N) - Outra implementação

```
/** Union Of rectangle Area */

struct Edge {
    bool open;
    int x, yMin, yMax;
    Edge(int x, int y1, int y2, bool op) {
        this->x = x;
        yMin = y1, yMax = y2;
        open = op;
    }
    bool operator < (const Edge &e) const {
        return (x < e.x);
    }
};


int n, m, h[maxN << 1];
int sum[maxN << 5], counter[maxN << 5];
vector<Edge> edges;



void print ( )
{   int maxsize=ceil(log2(num));
     maxsize=2*pow(2,maxsize)-1;

   for(int i= 1 ; i<=maxsize ; i++ )
          cout<<sum[i]<<" ";

    cout<<endl;

}

void update(int p, int l, int r, int yMin, int yMax, bool open) {

  if (h[r] < yMin || yMax < h[l]) return;

```

```
  // if ymin is greater than h[r] which is array of sorted y coordinates
     or
  // ymax is less than h[l] then l - r is not the required range

 /*
suppose our figure is like this ::
          _(2,5)
        _|_|_ (3,4)
(0,3)  |_|_|_|
  (1,2) _|_|_ (3,2)
(0,1)  |_|_|_|
  Three rectangles are there
    1) 0,1 -> 3,2
    2) 0,3 -> 3,4
    3) 1,2 -> 2,5
  h   -> [ 1 , 2 , 3,  4, 5 ]
edges -> [ (0,1,2) (0,3,4) (1,2,5) (2,2,5) (3,1,2) (3,3,4) ]
       Segment Tree using updation
             (1,5)                          0         1          2          4
                           2         1          0
     (1,3)         (3,5)                  0 0       1 0        1 1        2
        2         1 1       0 1       0 0
  (1,2) (2,3)  (3,4) (4,5)             0 0 0 0   1 0 0 0    1 0 1 0    1 1
     1 0    1 0 1 0   0 0 1 0   0 0 0 0
Question is what is actually done over here ?
   1 ) Firstly we sorted the edges according to the x coordinate and h[]
       according to y coordinates.
   2 ) then we started moving horizontal using a vertical sweep line to
       encounter 2  types of events : Left Edge,  Right Edge
     i) Left Edge: on encountering a left edge, we move it into the active
         sets and update the total length
                  of vertical sweep line that is cut by the rectangualar
                     boxes , this is done by update function
                  here we have two variable ymin (lower left y coordinate
                     ), ymax (upper left y coordinate) .
                  using which we update the total length of sweep line
                     intersected at that event, and after updating
                  sum[1] gives the length of intersected sweep line at
                     that x coordinate.
     ii) Right Edge : On encountering a right edge, we again update the
        segment tree.
   3) Function of segment tree is to store the intesected length at
       various x positions namely the events, whenever
      we reach an edge we update the intersected length  , whenever we are
          at left edge we know that we have added an edge between
      ymin and ymax therefore we need to add this to the sum at this
         position, if we are at right edge we know that we need to remove
      an edge between ymin and ymax, we do so by reducing counter ,if
         counter is still not zero means there were overlapping rectss
      and we sum[p] = h[r]- h[l] .
     NOTE : at any time there will be only two types of rectangles in
        active sets we need to worry abt
        _____                        __
       |_____|        OR           |__|
          ______                 __|__|__
         |______|               |__|__|__|
  (One below or abv otr)     (overlapping once)
 */

```

```
80
81      int c = p << 1, mid = (l + r) >> 1;
82
83      if (yMin <= h[l] && h[r] <= yMax) {          // ymin --- h[l] --- h[r]
            --- ymax
84          counter[p] += open ? 1 : -1;
85          if (counter[p]) sum[p] = h[r] - h[l];  //if there is a rectangle
                at that posn that is bw h[l] and h[r] we will add that to
                length
86          else sum[p] = sum[c] + sum[c + 1];    // else we will just sumup
                of lengths above and beloew this region
87          return;
88      }
89
90      if (l + 1 >= r) return;
91
92      update(c, l, mid, yMin, yMax, open);
93      update(c + 1, mid, r, yMin, yMax, open);
94
95      if (counter[p]) sum[p] = h[r] - h[l];
96      else sum[p] = sum[c] + sum[c + 1];
97
98
99
100 }
101
102 int64 solve() {
103                                  // process height for horzntl. sweep line
104     sort(h + 1, h + m + 1);     //  Sorting the hieght according to the y
            coordinates
105     int k = 1;
106
107     FOR (i, 2, m) if (h[i] != h[k])  //  Deleting the same horizontal
            sweeplines
108        h[++k] = h[i];                //     as they are redundant
109            m = k;
110          num  = m;
111
112
113     for (int i = 0, lm = maxN << 4; i < lm; i++) // This is the
            initialization step of segment tree
114         sum[i] = 0, counter[i] = 0;
115
116
117
118     int64 area = 0LL;  //  Initializing the Area
119
120     sort(all(edges));  // Sorting according to x coordinates for ver. swp
            line
121
122     update(1, 1, m, edges[0].yMin, edges[0].yMax, edges[0].open);
123    // print();
124
125     for (int i = 1; i < edges.size(); i++) {
126         area += sum[1] * (int64)(edges[i].x - edges[i - 1].x);
127         update(1, 1, m, edges[i].yMin, edges[i].yMax, edges[i].open);
128       //  print();
129             }
130     return area;
```

```
131
132
133
134 }
135
136 int main() {
137       int n;
138       cin>>n;
139       int  x1, y1, x2, y2;
140
141         edges.clear();
142         m = 0;
143         FOR (i, 1, n) {
144             scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
145
146
147             if(x1==x2 && y1>y2)swap(y1,y2);
148             else if(y1==y2    && x1>x2)swap(x1,x2);
149
150             x2+=1;     // x1 and y1 are bottom left coordinates
151             y2+=1;     // x2 and y2 are top right coordinates
152
153
154             edges.pb(Edge(x1, y1, y2, true));    // Inserting the Left edge
155             edges.pb(Edge(x2, y1, y2, false));  // Inserting the Right
                    Edge
156
157             /*
158             (x1,y2)      (x2,y2)
159                 __________
160                |          |
161    LeftEdge <- |          |-> Right Edge
162                |__________|
163             (x1,y1)      (x2,y1)
164             */
165
166
167             h[++m] = y1; // Inserting the Lower y Coordinate 1 based
                    inddexiing
168             h[++m] = y2; // Inserting the Upper y Coordinate
169
170         }
171         printf("%lld\n", solve());
172
173     return 0;
174 }
```

1.4 Estruturas

Listagem 1.4: Estruturas

```
1 struct pv{
2     ld x, y;
3     pv(){}
4     pv(ld _x, ld _y){
```

```
5           x = _x;
6           y = _y;
7       }
8
9       int getQuad(){
10          if(maiorIgual(x, 0.0) && maiorIgual(y, 0.0)) return 1;
11          if(menor(x, 0.0) && maiorIgual(y, 0.0)) return 2;
12          if(menorIgual(x, 0.0) && menor(y, 0.0)) return 3;
13          return 4;
14      }
15
16      pv operator + (pv other){
17          return pv(x+other.x, y+other.y);
18      }
19
20      pv operator - (pv other){
21          return pv(x-other.x, y-other.y);
22      }
23
24      pv operator * (ld k){
25          return pv(x*k, y*k);
26      }
27
28      pv operator / (ld k){
29          return pv(x/k, y/k);
30      }
31
32      bool operator == (pv other){
33          return igual(x, other.x) && igual(y, other.y);
34      }
35  };
36
37  struct line{
38      pv p0, v;
39      line(){}
40      line(pv _p0, pv _v){
41          p0 = _p0;
42          v = _v;
43      }
44  };
45
46  struct seg{
47      pv a, b;
48      seg(){}
49      seg(pv _a, pv _b){
50          a = _a;
51          b = _b;
52      }
53  };
54
55  struct circle{
56      pv centro;
57      ld r;
58      circle(){}
59      circle(pv _centro, ld _r){
60          centro = _centro;
61          r = _r;
62      }
63  };
```

1.5 Funções básicas

Listagem 1.5: Funções básicas

```
1  bool igual(ld a, ld b){
2      return fabs(a-b) < EPS;
3  }
4
5  bool maior(ld a, ld b){
6      return a > EPS + b;
7  }
8
9  bool menor(ld a, ld b){
10     return a + EPS < b;
11 }
12
13 bool maiorIgual(ld a, ld b){
14     return maior(a, b) || igual(a, b);
15 }
16
17 bool menorIgual(ld a, ld b){
18     return menor(a, b) || igual(a, b);
19 }
20
21 ld cross(pv a, pv b){
22     return a.x * b.y - a.y * b.x;
23 }
24
25 ld dot(pv a, pv b){
26     return a.x * b.x + a.y * b.y;
27 }
28
29 ld distPt(pv a, pv b){
30     return hypot(a.x-b.x, a.y-b.y);
31 }
32
33 //norma de um vetor
34 ld norma(pv a){
35     return sqrt(dot(a, a));
36 }
37
38 //retorna a menor distancia entre um ponto e uma reta
39 ld distPtReta(pv pt, line r){
40
41     return fabs(cross(r.v, pt - r.p0)) / norma(r.v);
42
43 }
44
45 //retorna a menor distancia de um ponto a um segmento
46 ld distPtSeg(pv pt, seg r){
47     if(maiorIgual(dot(pt - r.a, r.b - r.a), 0.0) && maiorIgual(dot(pt - r.
        b, r.a - r.b), 0.0))
48         return distPtReta(pt, line(r.a, r.b-r.a));
49
50     return min(distPt(pt, r.a), distPt(pt, r.b));
51 }
52
53
```

```
54 //angulo entre a reta horizontal com y = 0 e o vetor v
55 ld polarAngle(pv v){
56     ld x = v.x;
57     ld y = v.y;
58 
59     ld aux = atan2(y, x);
60 
61     if(menor(aux, 0.0)) aux += 2.0*PI;
62 
63     return aux;
64 }
65 
66 //radiano para grau
67 ld toDegree(ld rad){
68     return rad * 180 / PI;
69 }
70 
71 //angulo entre dois vetores (menor angulo)
72 ld angle2Vec(pv v, pv u){
73     ld ang1 = polarAngle(v);
74     ld ang2 = polarAngle(u);
75     ld ans = 0.0;
76 
77     ans = min(fabs(ang1 - ang2), fabs(2.0*PI+ang1 - ang2));
78     ans = min(ans, fabs(2.0*PI+ang2 - ang1));
79 
80     return ans;
81 }
82 
83 //checa se os pontos a, b, c estão em sentido anti horário
84 bool ccw(pv a, pv b, pv c){
85     ld aux = cross(b-a, c-b);
86 
87     return maior(aux, 0.0);
88 }
89 
90 bool colinear(pv a, pv b, pv c){
91     return igual(cross(b-a, c-b), 0.0);
92 }
93 
94 //retorna o sinal de um número
95 int sinal(ld res){
96     if(maior(res, 0.0)) return 1;
97     if(menor(res, 0.0)) return -1;
98     return 0;
99 }
```

1.6 Funções de comparação

Listagem 1.6: Funções de comparação

```
1 //ordena por x, se empatar por y
2 bool compareConvexHull(pv a, pv b){
3     if(igual(a.x, b.x)) return menor(a.y, b.y);
4     return menor(a.x, b.x);
```

```
}

//função pra ordenar em relação ao angulo polar só com cross
bool comparePolarAngle(pv a, pv b){
    if(a==b) return true;
    int q1 = a.getQuad();
    int q2 = b.getQuad();

    if(q1==q2){
        if(igual(cross(a, b), 0.0)) return menor(norma(a), norma(b));
        return maior(cross(a, b), 0.0);
    }
    return q1 < q2;
}
```

1.7 Funções de reta

Listagem 1.7: Funções de reta

```
//checa se duas retas são paralelas
bool retasParalelas(line r, line s){
    pv v = r.v;
    pv u = s.v;
    return igual(cross(v, u), 0.0);
}

//checa se duas retas são iguais
bool retasIguais(line r, line s){
    pv p0 = r.p0;
    pv p1 = s.p0;
    pv v = r.v;
    pv u = s.v;

    return retasParalelas(r, s) && igual(cross(v, p1-p0), 0.0);
}

//retorna o ponto de interseccao de duas retas
pv ptInterReta(line r, line s){
    pv v = r.v;
    pv u = s.v;

    pv p0 = r.p0;
    pv p1 = s.p0;

    long double t = (u.x * (p0.y - p1.y) + u.y * (p1.x - p0.x)) / cross(v,
        u);

    return pv(p0.x + (v.x * t), p0.y + (v.y * t));
}

//função que retorna a mediatriz
line getMediatriz(pv a, pv b){
    pv vet, medio, perp;
    medio = (a+b)/2.0;
    vet = b-a;
```

```
36      perp = pv(-vet.y, vet.x);
37
38      return line(medio, perp);
39  }
```

1.8 Funções de polígono

Listagem 1.8: Funções de polígono

```
1  //calcula o perimetro de um poligono
2  ld perimetro(vector<pv> &polygon){
3      if((int)polygon.size()<=1) return 0.0;
4
5      if(!(polygon[0] == polygon[(int)polygon.size()-1]))
6          polygon.pb(polygon[0]);
7
8      ld ans = 0.0;
9      for (int i = 0; i < (int)polygon.size()-1; i++)
10     {
11         ans += distPt(polygon[i], polygon[i+1]);
12     }
13     return ans;
14 }
15
16 //calcula a area de um poligono
17 ld area(vector<pv> &polygon){
18     if((int)polygon.size()<=2) return 0.0;
19
20     if(!(polygon[0] == polygon[(int)polygon.size()-1]))
21         polygon.pb(polygon[0]);
22
23     ld ans = 0.0;
24     for (int i = 0; i < (int)polygon.size()-1; i++)
25     {
26         ans += cross(polygon[i], polygon[i+1]);
27     }
28     return fabs(ans)*0.5;
29 }
30
31 //gera a convex hull do vector pt
32 void convexHull(vector<pv> &polygon){
33     if((int)polygon.size() <= 2) return;
34     pv p1[(int)polygon.size() + 2];
35     pv p2[(int)polygon.size() + 2];
36     int sz1 = 0, sz2 = 0;
37
38     sort(polygon.begin(), polygon.end(), compareConvexHull);
39     for (int i = 0; i < (int)polygon.size(); i++)
40     {
41         while (sz1 > 1 && ccw(p1[sz1-2], p1[sz1-1], polygon[i]))
42         {
43             sz1--;
44         }
45         p1[sz1++] = polygon[i];
46
```

```
47          while (sz2 > 1 && !ccw(p2[sz2-2], p2[sz2-1], polygon[i]))
48          {
49              sz2--;
50          }
51          p2[sz2++] = polygon[i];
52      }
53
54      polygon.clear();
55      for (int i = 0; i < sz1; i++)
56      {
57          polygon.pb(p1[i]);
58      }
59
60      for (int i = sz2-2; i >= 1; i--)
61      {
62          polygon.pb(p2[i]);
63      }
64  }
65
66  //checar se ponto esta dentro de poligono em O(n) - comparação de ângulo
67  bool insidePolygon(pv p, vector<pv> &polygon){
68      if((int)polygon.size()<= 2) return false;
69
70      if(!(polygon[0] == polygon[(int)polygon.size()-1])){
71          polygon.pb(polygon[0]);
72      }
73      bool naBorda = false;
74
75      ld ang = 0.0;
76      for (int i = 0; i < polygon.size()-1; i++)
77      {
78          naBorda = naBorda || onSegment(p, seg(polygon[i], polygon[i+1]));
79          if(ccw(p, polygon[i], polygon[i+1])){
80              ang += angle2Vec(polygon[i] - p, polygon[i+1]-p);
81          }else{
82              ang -= angle2Vec(polygon[i] - p, polygon[i+1]-p);
83          }
84      }
85      return igual(fabs(ang), 2.0*PI) || naBorda;//se quiser totalmente
            dentro tem que ver se ele ta na borda
86  }
87
88  //checa se um ponto esta dentro de um triangulo (a, b, c)
89  bool insideTriangle(pv p, pv a, pv b, pv c){
90      if(onSegment(p, seg(a, b))) return true;
91      if(onSegment(p, seg(a, c))) return true;
92      if(onSegment(p, seg(b, c))) return true;
93
94      return sinal(cross(b-a, p-a)) == sinal(cross(c-b, p-b)) && sinal(cross
            (c-b, p-b)) == sinal(cross(a-c, p-c));
95  }
96
97  //checa se um ponto esta dentro de um polígono em O(log n)
98  bool insidePolygonLogN(pv p, vector<pv> &polygon){//NAO PODE TER PONTOS
        COLINEARES
99      //CUIDADO COM POLIGONOS DEGENERADOS
100     if((int)polygon.size() == 0) return false;
101
102     int sz;
```

```
103     if(polygon[0] == polygon[(int)polygon.size()-1]) sz = (int)polygon.
            size()-1;
104     else sz = (int)polygon.size();
105
106     if(sz <= 2) return false;
107
108     if(ccw(polygon[0], polygon[1], polygon[2])){
109         if(!(polygon[0] == polygon[(int)polygon.size()-1])) polygon.pb(
                polygon[0]);
110         reverse(polygon.begin(), polygon.end());
111     }
112
113     int lo = 1, hi = sz-1, mid;
114     int c;
115     while (lo <= hi)
116     {
117         mid = (lo + hi)/2;
118         c = sinal(cross(polygon[mid] - polygon[0], p - polygon[0]));
119         if(c < 0){
120             lo = mid+1;
121         }else if(c > 0){
122             hi = mid-1;
123         }else{
124             if(mid == 1){
125                 return insideTriangle(p, polygon[0], polygon[mid], polygon
                        [mid+1]);
126             }else{
127                 return insideTriangle(p, polygon[0], polygon[mid-1],
                        polygon[mid]);
128             }
129         }
130     }
131     mid = (lo+hi)/2;
132     if(mid == 0 || mid == sz-1) return false;
133
134     return insideTriangle(p, polygon[0], polygon[mid], polygon[mid+1]);
135 }
136
137 bool insidePolygonLogNConfiavel(pv p, vector<pv>& polygon){
138     vector<pv> lower, upper;
139     convexHull(polygon, lower, upper);//PRECISA QUE A CONVEX HULL RETORNE
            A LOWER E UPPER HULL
140     ld xMax, xMin, yMax, yMin;
141     ld up1, up2, dw1, dw2;
142
143
144     xMax = upper[(int)upper.size()-1].x;
145     xMin = upper[0].x;
146
147     for (int i = 0; i < (int)upper.size(); i++)
148     {
149         yMax = max(yMax, upper[i].y);
150         yMin = min(yMin, upper[i].y);
151         if(upper[i].x == xMin){
152             up1 = max(up1, upper[i].y);
153             dw1 = min(dw1, upper[i].y);
154         }
155         if(upper[i].x == xMax){
156             up2 = max(up2, upper[i].y);
```

```
157             dw2 = min(dw2, upper[i].y);
158         }
159     }
160     for (int i = 0; i < (int)lower.size(); i++)
161     {
162         yMax = max(yMax, lower[i].y);
163         yMin = min(yMin, lower[i].y);
164         if(upper[i].x == xMin){
165             up1 = max(up1, lower[i].y);
166             dw1 = min(dw1, lower[i].y);
167         }
168         if(upper[i].x == xMax){
169             up2 = max(up2, lower[i].y);
170             dw2 = min(dw2, lower[i].y);
171         }
172     }
173     if(p.x < xMin || p.x > xMax || p.y < yMin || p.y > yMax) return false;
174 
175     if(p.x == xMin && (p.y < dw1 || p.y > up1)) return false;
176     if(p.x == xMax && (p.y < dw2 || p.y > up2)) return false;
177 
178     //nesse momento, o ponto tem x > xMin e x < xMax
179     int lo = 0, hi = (int)upper.size()-1;
180     int mid;
181     int ans1, ans2;
182     while (lo <= hi)
183     {
184         mid = (lo+hi)/2;
185         if(p.x >= upper[mid].x && p.x <= upper[mid+1].x){
186             ans1 = mid;
187             break;
188         }
189         if(p.x < upper[mid].x){
190             hi = mid-1;
191         }else{
192             lo = mid+1;
193         }
194     }
195 
196     lo = 0; hi = (int)lower.size()-1;
197     while (lo <= hi)
198     {
199         mid = (lo+hi)/2;
200         if(p.x >= lower[mid].x && p.x <= lower[mid+1].x){
201             ans2 = mid;
202             break;
203         }
204         if(p.x < lower[mid].x){
205             hi = mid-1;
206         }else{
207             lo = mid+1;
208         }
209     }
210 
211     if(onSegment(p, seg(upper[ans1], upper[ans1+1]))) return true;
212     if(onSegment(p, seg(lower[ans2], lower[ans2+1]))) return true;
213 
214     return sinal(cross(upper[ans1+1]-upper[ans1], p - upper[ans1]))==-1 &&
         sinal(cross(lower[ans2+1] - lower[ans2], p - lower[ans2]))==1;
```

```
215 }
216
217 //FUNÇÃO DO JUNIOR (% significa cross)
218 bool PointIsInsideConvexPolygon(vector<pv> &a, pv p) {
219   pv vp = p - a[0];
220   if(cross((a[1] - a[0]), vp) > 0) return 0;
221   int lo = 1, hi = a.size() - 1;
222   while(lo < hi) {
223     int md = (lo + hi + 1) >> 1;
224     if(cross((a[md] - a[0]), vp) < 0) lo = md;
225     else hi = md - 1;
226   }
227   if(hi == a.size() - 1) return false;
228   return cross(((a[lo + 1] - a[lo]), (p - a[lo]))) < 0;
229 }
230
231 //retorna a interseccao dos poligonos
232 int interPolygon(vector<pv> &polygon1, vector<pv>& polygon2, vector<pv> &
      interseccao){
233
234     if(!(polygon1[0] == polygon1[(int)polygon1.size() - 1])) polygon1.pb(
          polygon1[0]);
235     if(!(polygon2[0] == polygon2[(int)polygon2.size() - 1])) polygon2.pb(
          polygon2[0]);
236     interseccao.clear();
237
238     bool tem = false;
239     vector<seg> segmentos;
240     pv pt;
241
242     for (int i = 0; i < (int)polygon1.size()-1; i++)
243     {
244         segmentos.pb(seg(polygon1[i], polygon1[i+1]));
245     }
246     for (int i = 0; i < (int)polygon2.size()-1; i++)
247     {
248         segmentos.pb(seg(polygon2[i], polygon2[i+1]));
249     }
250
251     int res;
252     for (int i = 0; i < (int)segmentos.size(); i++)
253     {
254         for (int j = i+1; j < (int)segmentos.size(); j++)
255         {
256             res = interSegmento(segmentos[i], segmentos[j], pt);
257             if(res == 1){
258                 if(insidePolygonLogN(pt, polygon1) && insidePolygonLogN(pt
                      , polygon2)){
259                     interseccao.pb(pt);
260                     tem = true;
261                 }
262             }else if(res == 2){
263                 if(onSegment(segmentos[i].a, segmentos[j])){
264                     pt = segmentos[i].a;
265                     if(insidePolygonLogN(pt, polygon1) &&
                          insidePolygonLogN(pt, polygon2)){
266                         interseccao.pb(pt);
267                         tem = true;
268                     }
```

```
269                 }
270                 if(onSegment(segmentos[i].b, segmentos[j])){
271                     pt = segmentos[i].b;
272                     if(insidePolygonLogN(pt, polygon1) &&
                            insidePolygonLogN(pt, polygon2)){
273                         interseccao.pb(pt);
274                         tem = true;
275                     }
276                 }
277                 if(onSegment(segmentos[j].a, segmentos[i])){
278                     pt = segmentos[j].a;
279                     if(insidePolygonLogN(pt, polygon1) &&
                            insidePolygonLogN(pt, polygon2)){
280                         interseccao.pb(pt);
281                         tem = true;
282                     }
283                 }
284                 if(onSegment(segmentos[j].b, segmentos[i])){
285                     pt = segmentos[j].b;
286                     if(insidePolygonLogN(pt, polygon1) &&
                            insidePolygonLogN(pt, polygon2)){
287                         interseccao.pb(pt);
288                         tem = true;
289                     }
290                 }
291             }
292         }
293 
294     }
295     if(!tem) return false;
296 
297     convexHull(interseccao);
298 
299     return true;
300 }
301 
302 //funcao usada na cutPolygon para não repetir vértices em um corte
303 void adiciona(vector<pv> &polygon, pv pt){
304     if((int)polygon.size() == 0){
305         polygon.pb(pt);
306     }else{
307         if(!(polygon[(int)polygon.size()-1] == pt))
308             polygon.pb(pt);
309     }
310 }
311 
312 //NAO ACEITA PONTOS COLINEARES
313 //0 - não cortou ou cortou na borda
314 //1 - as duas partes tem area
315 int cutPolygon(line r, vector<pv> &polygon, vector<pv> &lp, vector<pv> &rp
        ){
316     if(!(polygon[0] == polygon[(int)polygon.size()-1]))
317         polygon.pb(polygon[0]);
318 
319     lp.clear();
320     rp.clear();
321     int lado;
322 
323     int cortou = 0;
```

```
324     pv pt;
325     for (int i = 0; i < (int)polygon.size()-1; i++)
326     {
327         lado = sinal(cross(r.v, polygon[i] - r.p0));
328 
329         //teste de ponto
330         if(lado == 1){//esta à esquerda da linha
331             adiciona(lp, polygon[i]);
332         }else if(lado == -1){//esta à direita da linha
333             adiciona(rp, polygon[i]);
334         }
335 
336         //teste de intersecção
337         if(sinal(cross(r.v, polygon[i] - r.p0)) != sinal(cross(r.v,
                polygon[i+1] - r.p0))){//segmento está inteiro na reta (cortou
                na borda)
338             //nao faz nada pois os dois vertices serão adicionados na
                    verificacao acima
339             cortou = 1;
340             pt = ptInterReta(line(polygon[i], polygon[i+1]-polygon[i]), r)
                    ;
341             adiciona(lp, pt);
342             adiciona(rp, pt);
343         }
344     }
345     if((int)lp.size() > 2) adiciona(lp, lp[0]);
346     else lp.clear();
347     if((int)rp.size() > 2) adiciona(rp, rp[0]);
348     else rp.clear();
349     return cortou;
350 }
```

1.9 Funções de segmento

Listagem 1.9: Funções de segmento

```
1  //checa se ponto p está no segmento s
2  bool onSegment(pv p, seg s){
3      pv a = s.a;
4      pv b = s.b;
5      if(maiorIgual(dot(p-a, b-a), 0.0) && maiorIgual(dot(p-b, a-b), 0.0) &&
           igual(cross(p-a, p-b), 0.0)) return true;
6      return false;
7  }
8  
9  //checa se 2 segmentos se cruzam (pode ser na borda)
10 bool temInterSegmento(seg r, seg s){
11     pv a = r.a;
12     pv b = r.b;
13     pv c = s.a;
14     pv d = s.b;
15 
16     return sinal(cross(b-a, d-a)) != sinal(cross(b-a, c-a)) && sinal(cross
           (d-c, a-c)) != sinal(cross(d-c, b-c));
17 }
```

```
18
19 //checa se tem interseccao entre segmentos (0 - nao tem, 1 - tem, 2 -
       paralelos (SE TOCAM MAS NAO RETORNA O PONTO) )
20 int interSegmento(seg r, seg s, pv &ans){
21     pv a = r.a;
22     pv b = r.b;
23     pv c = s.a;
24     pv d = s.b;
25     if(retasIguais(line(a, b-a), line(c, d-c))){
26
27         if(onSegment(c, r) || onSegment(d, r) || onSegment(a, s) ||
               onSegment(b, s)){
28             return 2;
29         }
30         return 0;
31     }
32     if(temInterSegmento(r, s)){
33         ans = ptInterReta(line(a, b-a), line(c, d-c));
34         return 1;
35     }
36     return 0;
37 }
```

1.10 Line Sweep (Balão)

Listagem 1.10: Line Sweep (Balão)

```
1  struct pv{
2      ll x, y;
3      pv(){}
4      pv(ll _x, ll _y){
5          x = _x;
6          y = _y;
7      }
8      pv operator - (pv other){
9          return pv(x-other.x, y-other.y);
10     }
11 };
12
13 struct event{
14     pv p;
15     int tipo, id;
16     event(){}
17     event(pv _p, int _tipo, int _id){
18         p = _p;
19         tipo = _tipo;
20         id = _id;
21     }
22     bool operator < (event other) const{
23         if(p.x == other.p.x) return tipo < other.tipo;
24         return p.x < other.p.x;
25     }
26 };
27
28 struct seg{
```

```
29      pv a, b;
30      seg(){}
31      seg(pv _a, pv _b){
32          a = _a;
33          b = _b;
34      }
35  };
36
37
38  int dp[MAXN][LOG];
39  int qtdSeg, qtdBalao;
40  vector<seg> segmento;
41  vector<event> evento;
42  vector<pv> vertice;
43  vector<int> g[MAXN];
44
45  ll cross(pv a, pv b){
46      return a.x*b.y - a.y*b.x;
47  }
48
49  ll func(pv a, pv b, pv c){
50      return cross(b-a, c-b);
51  }
52
53  bool compare(int _1, int _2){
54      pv a = segmento[_1].a;
55      pv b = segmento[_1].b;
56      pv c = segmento[_2].a;
57      pv d = segmento[_2].b;
58
59      if(a.x < c.x){
60          if(func(a, c, b) > 0)
61              return false;
62          return true;
63      }else{
64          if(func(c, a, d) > 0)
65              return true;
66          return false;
67      }
68  }
69  set<int, bool(*)(int, int)> s(compare);
70
71  void adicionar(int id){
72      s.insert(id);
73      set<int, bool(*)(int, int)>::iterator it;
74      if(segmento[id].a.y > segmento[id].b.y){
75          it = s.find(id);
76          it++;
77          if(it == s.end()){
78              vertice[id] = pv(segmento[id].a.x, INF);
79          }else{
80              if(segmento[*it].a.y == segmento[*it].b.y){
81                  vertice[id] = pv(segmento[id].a.x, segmento[*it].a.y);
82              }else{
83                  g[id].pb(*it);
84              }
85          }
86      }
87  }
```

```
88
89 void remover(int id){
90     set<int, bool(*)(int, int)>::iterator it;
91     if(segmento[id].a.y < segmento[id].b.y){
92         it = s.find(id);
93         it++;
94         if(it==s.end()){
95             vertice[id] = pv(segmento[id].b.x, INF);
96         }else{
97             if(segmento[*it].a.y == segmento[*it].b.y){
98                 vertice[id] = pv(segmento[id].b.x, segmento[*it].a.y);
99             }else{
100                g[id].pb(*it);
101            }
102        }
103    }
104    s.erase(id);
105 }
106
107 void consultar(int id, int x){
108    set<int, bool(*)(int, int)>::iterator it;
109    if(s.size()==0){
110        vertice[id] = pv(x, INF);
111    }else{
112        it = s.begin();
113        if(segmento[*it].a.y == segmento[*it].b.y){
114            vertice[id] = pv(x, segmento[*it].a.y);
115        }else{
116            g[id].pb(*it);
117        }
118    }
119 }
120
121 void sweep(){
122    sort(evento.begin(), evento.end());
123    for (int i = 0; i < evento.size(); i++)
124    {
125        if(evento[i].tipo == ABRE)
126            adicionar(evento[i].id);
127        else if(evento[i].tipo == FECHA)
128            remover(evento[i].id);
129        else
130            consultar(evento[i].id, evento[i].p.x);
131    }
132 }
133
134 void build(){
135    for (int i = 0; i < qtdSeg + qtdBalao; i++)
136    {
137        if(g[i].size() == 0){
138            dp[i][0] = i;
139        }else{
140            dp[i][0] = g[i][0];
141        }
142    }
143    for (int j = 1; j < LOG; j++)
144    {
145        for (int i = 0; i < qtdSeg + qtdBalao; i++)
146        {
```

```
147             dp[i][j] = dp[dp[i][j-1]][j-1];
148         }
149     }
150 }
151
152 int pula(int u){
153     int d = MAXN;
154     for (int i = 0; i < LOG; i++)
155     {
156         if(d & (1<<i)){
157             u = dp[u][i];
158         }
159     }
160     return u;
161 }
162
163 void reset(){
164     segmento.clear();
165     evento.clear();
166     vertice.clear();
167     for (int i = 0; i < qtdSeg+qtdBalao; i++)
168     {
169         g[i].clear();
170     }
171 }
172
173 int main(){
174     ios_base::sync_with_stdio(0);
175     cin.tie(0);
176
177     while(cin >> qtdSeg >> qtdBalao){
178         reset();
179         ll x, y, X2, Y2;
180         for (int i = 0; i < qtdSeg; i++)
181         {
182             cin >> x >> y >> X2 >> Y2;
183             if(x < X2){
184                 segmento.pb(seg(pv(x, y), pv(X2, Y2)));
185             }else{
186                 segmento.pb(seg(pv(X2, Y2), pv(x, y)));
187             }
188             evento.pb(event(segmento[i].a, ABRE, i));
189             evento.pb(event(segmento[i].b, FECHA, i));
190             vertice.pb(pv(-1, -1));
191         }
192
193         for (int i = 0 + qtdSeg; i < qtdBalao + qtdSeg; i++)
194         {
195             cin >> x;
196             evento.pb(event(pv(x, 0), CONSULTA, i));
197             vertice.pb(pv(-1, -1));
198         }
199
200         sweep();
201         build();
202         int ult;
203         for (int i = 0 + qtdSeg; i < qtdBalao+qtdSeg; i++)
204         {
205             ult = pula(i);
```

```
206             if(vertice[ult].y == INF) cout << vertice[ult].x << "\n";
207             else cout << vertice[ult].x << " " << vertice[ult].y << "\n";
208         }
209     }
210     return 0;
211 }
```

1.11 Rotação de ponto

Listagem 1.11: Rotação de ponto

```
1 //angulo deve estar em radiano
2 pv rotacionaHorario(pv p, double rad)
3 {
4     rad=-rad;
5     return pv(p.x * cos(rad) - p.y * sin(rad), p.x * sin(rad) + p.y * cos(
          rad));
6 }
7
8 pv rotacionaAntiHorario(pv p, double rad)
9 {
10    return pv(p.x * cos(rad) - p.y * sin(rad), p.x * sin(rad) + p.y * cos(
          rad));
11 }
12
13 int main()
14 {
15    //le o ponto
16    //chama a funcao pra rotacionar
17    return 0;
18 }
```

1.12 Rotating calipers

Listagem 1.12: Rotating calipers

```
1 int rotatingCalipers(vector<pv> &up, vector<pv> &dn){
2     int ans = 0;
3
4     int i = 0, j = dn.size()-1;
5
6     while(i < (int)up.size() - 1 || j > 0){
7         // Entrou aqui: up[i] e dn[j] eh um antipodal pair
8         ans = max(ans, distPt(up[i],dn[j]));//NAO TIRAR A RAIZ NO DISTPT
              PARA EVITAR PRECISAO
9
10        if(i == (int)up.size()-1) j--;
11        else if(j == 0) i++;
12        else{
13            // Verifica qual o menor angulo a ser rotacionado p utilizar
                  na rotacao
```

```
14              if((up[i+1].y - up[i].y) * (dn[j].x - dn[j-1].x)
15                  > (dn[j].y - dn[j-1].y) * (up[i+1].x - up[i].x ))
16                  i++;
17              else
18                  j--;
19          }
20      }
21      return ans;
22  }
```

1.13 Sweep Circle (esconde-esconde)

Listagem 1.13: Sweep Circle (esconde-esconde)

```
1  #define ABRE 0
2  #define QUERY 1
3  #define FECHA 2
4
5  struct pv{
6      ll x, y;
7      pv(){}
8      pv(ll _x, ll _y){
9          x = _x;
10         y = _y;
11     }
12
13     pv operator - (pv other){
14         return pv(x-other.x, y-other.y);
15     }
16 };
17
18 ll cross(pv a, pv b){
19     return a.x * b.y - a.y * b.x;
20 }
21
22 struct event{
23     pv pt;
24     int tipo, quadrante, id;
25     event(){}
26     event(pv _pt, int _tipo, int _quadrante, int _id){
27         pt = _pt;
28         tipo = _tipo;
29         quadrante = _quadrante;
30         id = _id;
31     }
32     bool operator < (event other) const{
33         ll c;
34         if(quadrante == other.quadrante){
35             c = cross(pt, other.pt);
36             if(c == 0){
37                 return tipo < other.tipo;
38             }
39             return c > 0;
40         }
41         return quadrante < other.quadrante;
```

```
42      }
43  };
44
45  struct seg{
46      pv a, b;
47      int id;
48      seg(){}
49      seg(pv _a, pv _b){
50          a = _a;
51          b = _b;
52          id = -1;
53      }
54      seg(pv _a, pv _b, int _id){
55          a = _a;
56          b = _b;
57          id = _id;
58      }
59  };
60
61
62  ld distPt(pv a, pv b){
63      return hypot(a.x-b.x, a.y-b.y);
64  }
65
66
67  ll dot(pv a, pv b){
68      return a.x * b.x + a.y * b.y;
69  }
70
71  int sinal(ll res){
72      if(res > 0) return 1;
73      if(res < 0) return -1;
74      return 0;
75  }
76
77  //checa se 2 segmentos se cruzam (pode ser na borda)
78  bool temInterSegmento(seg r, seg s){
79      pv a = r.a;
80      pv b = r.b;
81      pv c = s.a;
82      pv d = s.b;
83
84      return sinal(cross(b-a, d-a)) != sinal(cross(b-a, c-a)) && sinal(cross
            (d-c, a-c)) != sinal(cross(d-c, b-c));
85  }
86
87  void printaPonto(pv a){
88      printf("(%lld, %lld)  ", a.x, a.y);
89
90  }
91
92  vector<pv> crianca;
93  vector<seg> parede, tmpParede;
94  vector<event> eve;
95  int qtdParede, qtdCrianca, qtdProcura;
96  pv origin;
97
98  bool compareSet(int id1, int id2){
99      pv a = tmpParede[id1].a;
```

```
100     pv b = tmpParede[id1].b;
101     pv c = tmpParede[id2].a;
102     pv d = tmpParede[id2].b;
103 
104     return cross(d-a, b-a) > 0;
105 }
106 
107 //~ set<int, bool(*)(int, int)> active(compareSet);
108 set<int> active;
109 
110 int getQuadrante(pv pt){
111     ll x = pt.x;
112     ll y = pt.y;
113     if(x >= 0 && y >= 0) return 1;
114     if(x < 0 && y >= 0) return 2;
115     if(x <= 0 && y < 0) return 3;
116     if(x > 0 && y < 0) return 4;
117 }
118 
119 bool needCut(seg r){
120     return r.a.y < 0 && r.b.y > 0;
121 }
122 
123 int visivel(pv pt){
124     //inicio bruta
125     for (auto i : active)
126     {
127         seg r = tmpParede[i];
128         if(temInterSegmento(seg(pv(0, 0), pt), r)) return 0;
129     }
130     return 1;
131 
132     //fim bruta
133     if((int)active.size() == 0) return 1;
134     seg r = tmpParede[*active.begin()];
135     if(temInterSegmento(seg(pv(0, 0), pt), r)) return 0;
136     return 1;
137 }
138 
139 int sweep(){
140     sort(eve.begin(), eve.end());
141 
142     int ans = 0;
143 
144     for (int i = 0; i < (int)eve.size(); i++)
145     {
146         if(eve[i].tipo == ABRE){
147             active.insert(eve[i].id);
148         }
149         else if(eve[i].tipo == FECHA){
150             active.erase(eve[i].id);
151         }else ans += visivel(eve[i].pt);
152     }
153     return ans;
154 }
155 
156 int solve(int idx){
157     origin = crianca[idx];
158     eve.clear();
```

```
159     tmpParede.clear();
160     pv pt;
161     for (int i = 0; i < qtdCrianca; i++)
162     {
163         if(i==idx) continue;
164         pt = crianca[i] - origin;
165         eve.pb(event(pt, QUERY, getQuadrante(pt), -1));
166     }
167 
168     int cnt = 0;
169     seg nw;
170     for (int i = 0; i < qtdParede; i++)
171     {
172         nw = seg(parede[i].a - origin, parede[i].b - origin, cnt);
173         if(!needCut(nw)){
174             if(cross(nw.a, nw.b) > 0){
175                 eve.pb(event(nw.a, ABRE, getQuadrante(nw.a), cnt));
176                 eve.pb(event(nw.b, FECHA, getQuadrante(nw.b), cnt));
177             }else{
178                 eve.pb(event(nw.b, ABRE, getQuadrante(nw.b), cnt));
179                 eve.pb(event(nw.a, FECHA, getQuadrante(nw.a), cnt));
180             }
181             tmpParede.pb(nw);
182             cnt++;
183         }else{
184             if(cross(nw.b - nw.a, pv(0, 0) - nw.a) > 0){
185                 eve.pb(event(pv(1,0), ABRE, 1, cnt));
186                 eve.pb(event(nw.b, FECHA, getQuadrante(nw.b), cnt));
187                 eve.pb(event(nw.a, ABRE, getQuadrante(nw.a), cnt));
188                 eve.pb(event(pv(1, 0), FECHA, 4, cnt));
189             }else{
190                 eve.pb(event(nw.b, ABRE, getQuadrante(nw.b), cnt));
191                 eve.pb(event(pv(-1, 0), FECHA, 2, cnt));
192                 eve.pb(event(pv(-1, 0), ABRE, 3, cnt));
193                 eve.pb(event(nw.a, FECHA, getQuadrante(nw.a), cnt));
194             }
195             tmpParede.pb(nw);
196             cnt++;
197         }
198     }
199     return sweep();
200 }
201 
202 void reset(){
203     eve.clear();
204     tmpParede.clear();
205     parede.clear();
206     crianca.clear();
207 }
208 
209 int main(){
210     ios_base::sync_with_stdio(0);
211     cin.tie(0);
212 
213     ll x, y, x0, y0;
214     while (cin >> qtdProcura >> qtdCrianca >> qtdParede)
215     {
216         reset();
217         for (int i = 0; i < qtdCrianca; i++)
```

```
218         {
219             cin >> x >> y;
220             crianca.pb(pv(x, y));
221         }
222         for (int i = 0; i < qtdParede; i++)
223         {
224             cin >> x >> y >> x0 >> y0;
225             if(y < y0) parede.pb(seg(pv(x, y), pv(x0, y0), -1));
226             else parede.pb(seg(pv(x0, y0), pv(x, y), -1));
227         }
228         for (int i = 0; i < qtdProcura; i++)
229         {
230             cout << solve(i) << "\n";
231         }
232     }
233 
234 
235     return 0;
236 }
```

1.14 União do volume das caixas

Listagem 1.14: União do volume das caixas

```
1  // OBS: nesse problema cada caixa era definido por um ponto (x,y,z) e pela
       origem (0,0,0)
2  #include <bits/stdc++.h>
3  
4  using namespace std;
5  
6  #define ft first
7  #define sd second
8  #define mp make_pair
9  
10 typedef long long ll;
11 
12 typedef struct ponto{
13     ll x, y, z;
14 } Ponto;
15 
16 typedef struct node{
17     ll altMin;
18     ll altMax;
19     ll soma;
20 } Node;
21 
22 
23 bool cmp(Ponto a, Ponto b){
24     if(a.x != b.x)
25         return a.x > b.x;
26     if(a.y != b.y)
27         return a.y > b.y;
28     return a.z > b.z;
29 }
30 
```

```
Ponto v[100005];

// par de altura max / soma das alturas
Node st[400005];
ll lz[400005];

void build(int no, int l, int r){
    lz[no] = 0;
    st[no].altMin = st[no].altMax = st[no].soma = 0LL;

    if(l == r)
        return;

    build(no*2, l, (l+r)/2);
    build(no*2+1, 1+(l+r)/2, r);
    return;
}

void prop(int no, int l, int r){
    if(!lz[no])
        return;

    st[no].altMin = st[no].altMax = lz[no];
    st[no].soma = (r-l+1)*lz[no];
    if(l != r)
        lz[no*2] = lz[no*2+1] = lz[no];

    lz[no] = 0;
    return;
}

int queryAlt(int no, int l, int r, int maior, int altura){ // me fala a
    primeira posicao <= maior q eh "mais baixa q o i" (tem z < vi.z)
    prop(no, l, r);
    if(st[no].altMax < altura)
        return l;
    if(l == r || st[no].altMin >= altura)
        return maior;
    return min(queryAlt(no*2, l, (l+r)/2, maior, altura), queryAlt(no*2+1,
        1+(l+r)/2, r, maior, altura));
}

ll querySum(int no, int l, int r, int ini, int fim){
    prop(no, l, r);
    if(l > fim || r < ini)
        return 0LL;
    if(l >= ini && r <= fim)
        return st[no].soma;
    return querySum(no*2, l, (l+r)/2, ini, fim) + querySum(no*2+1, 1+(l+r)
       /2, r, ini, fim);
}

void update(int no, int l, int r, int ini, int fim, ll val){
    prop(no, l, r);
    if(l > fim || r < ini)
        return;
    if(l >= ini && r <= fim){
        lz[no] = val;
        prop(no, l, r);
```

```
87          return;
88      }
89
90      update(no*2, l, (l+r)/2, ini, fim, val);
91      update(no*2+1, 1+(l+r)/2, r, ini, fim, val);
92
93      st[no].altMin = min(st[no*2].altMin, st[no*2+1].altMin);
94      st[no].altMax = max(st[no*2].altMax, st[no*2+1].altMax);
95      st[no].soma = st[no*2].soma + st[no*2+1].soma;
96  }
97
98  main(){
99      while(1){
100         ll n, m;
101         scanf("%lld %lld", &n, &m);
102
103         if(!n && !m) break;
104
105         ll maxy = 0;
106         for(int i=0;i<n;i++){
107             scanf("%lld %lld %lld", &v[i].x, &v[i].y, &v[i].z);
108             maxy = max(v[i].y, maxy);
109         }
110         maxy += 2;
111
112         sort(v, v+n, cmp);
113         build(1, 1, maxy);
114
115         v[n].x = 0; // seta o menor x com zero
116         ll ans = 0;
117         for(int i=0;i<n;i++){
118             int l = queryAlt(1, 1, maxy, maxy, v[i].z); // me fala a
                    primeira posicao q eh "mais baixa q eu"
119             if(l > v[i].y)
120                 continue;
121
122             ll vaux = querySum(1, 1, maxy, l, v[i].y);
123             ll vun = (v[i].y-l+1) * v[i].z - vaux;
124             //printf("o volume de cada unidade eh %lld * %lld - %lld = %
                    lld\n", v[i].y-l+1, v[i].z, vaux, vun);
125             //printf("a qtdd de unidades eh %lld\n", v[i].x, v[i+1].x, v[i
                    ].x-v[i+1].x);
126             //printf("o volume total do bloco eh %lld\n", vun * (v[i].x));
127             //printf("vou dar update de %lld no range %d %lld\n\n", v[i].z
                    , l, v[i].y);
128
129             ans +=  vun * v[i].x;
130             update(1, 1, maxy, l, v[i].y, v[i].z);
131         }
132         printf("%lld\n", m*m*m-ans);
133     }
134 }
```