



## Notebook - Maratona de Programação

Posso mandar um WA?

### Contents

|                                      |           |   |           |
|--------------------------------------|-----------|---|-----------|
| <b>1 Geometria</b>                   | <b>2</b>  | 4.8 Linear Diophantine Equation . . . . . | 14        |
| 1.1 Linear Transformation . . . . .  | 2         | 4.9 Miller Habin . . . . .                | 14        |
| 1.2 Inside Polygon . . . . .         | 2         | 4.10 Fft Simple . . . . .                 | 15        |
| 1.3 Delaunay . . . . .               | 2         | 4.11 Fft Tourist . . . . .                | 15        |
| 1.4 Convex Hull . . . . .            | 4         | 4.12 Matrix Exponentiation . . . . .      | 17        |
| 1.5 Sort By Angle . . . . .          | 4         | 4.13 Mulmod . . . . .                     | 17        |
| 1.6 Minkowski Sum . . . . .          | 4         | 4.14 Raiz Primitiva . . . . .             | 17        |
| 1.7 Tetrahedron Distance3d . . . . . | 4         | 4.15 Bigmod . . . . .                     | 17        |
| 1.8 Numintersectionline . . . . .    | 5         | 4.16 Berlekamp Massey . . . . .           | 18        |
| 1.9 Polygon Diameter . . . . .       | 6         | 4.17 Double Gcd . . . . .                 | 18        |
| 1.10 Polygon Cut Length . . . . .    | 6         | 4.18 Totient . . . . .                    | 18        |
| 1.11 Mindistpair . . . . .           | 6         | 4.19 Kitamasa . . . . .                   | 19        |
| 1.12 Rotating Callipers . . . . .    | 6         | 4.20 Mobius . . . . .                     | 19        |
| 1.13 Half Plane Intersect . . . . .  | 7         | <b>5 Grafos</b>                           | <b>19</b> |
| 1.14 2d . . . . .                    | 7         | 5.1 Ford . . . . .                        | 19        |
| 1.15 Intersect Polygon . . . . .     | 9         | 5.2 2sat . . . . .                        | 20        |
| 1.16 3d . . . . .                    | 9         | 5.3 Kahn . . . . .                        | 20        |
| <b>2 Algoritmos</b>                  | <b>10</b> | 5.4 Hungarian . . . . .                   | 20        |
| 2.1 Mst Xor . . . . .                | 10        | 5.5 Dfs Tree . . . . .                    | 20        |
| 2.2 Cdq . . . . .                    | 11        | 5.6 Lca . . . . .                         | 21        |
| 2.3 Histogram Rectangle . . . . .    | 12        | 5.7 Hld Aresta . . . . .                  | 21        |
| <b>3 Misc</b>                        | <b>12</b> | 5.8 Mcmf . . . . .                        | 22        |
| 3.1 Rand . . . . .                   | 12        | 5.9 Centroid . . . . .                    | 23        |
| 3.2 Bitwise . . . . .                | 12        | 5.10 Kosaraju . . . . .                   | 23        |
| 3.3 Template . . . . .               | 13        | 5.11 Dinic . . . . .                      | 23        |
| 3.4 Ordered Set . . . . .            | 13        | 5.12 Hld Vertice . . . . .                | 23        |
| <b>4 Math</b>                        | <b>13</b> | <b>6 Numeric</b>                          | <b>24</b> |
| 4.1 Randommod . . . . .              | 13        | 6.1 Lagrange Interpolation . . . . .      | 24        |
| 4.2 Division Trick . . . . .         | 13        | 6.2 Newton Raphson . . . . .              | 24        |
| 4.3 Inverso Mult . . . . .           | 13        | 6.3 Simpson's Formula . . . . .           | 25        |
| 4.4 Crt . . . . .                    | 13        | <b>7 DP</b>                               | <b>25</b> |
| 4.5 Gaussxor . . . . .               | 14        | 7.1 Largest Ksubmatrix . . . . .          | 25        |
| 4.6 Pollard Rho . . . . .            | 14        | 7.2 Dp Digitos . . . . .                  | 25        |
| 4.7 Fast Exponentiaion . . . . .     | 14        | 7.3 Partition Problem . . . . .           | 25        |
|                                      |           | 7.4 Aliens . . . . .                      | 25        |
|                                      |           | 7.5 Lis . . . . .                         | 26        |

|          |                                  |           |
|----------|----------------------------------|-----------|
| <b>8</b> | <b>Strings</b>                   | <b>26</b> |
| 8.1      | Manacher . . . . .               | 26        |
| 8.2      | Suffix Automaton . . . . .       | 26        |
| 8.3      | Edit Distance . . . . .          | 27        |
| 8.4      | Suffix Array . . . . .           | 27        |
| 8.5      | Lcs . . . . .                    | 27        |
| 8.6      | Eertree . . . . .                | 27        |
| 8.7      | Aho Corasick . . . . .           | 27        |
| 8.8      | Kmp . . . . .                    | 28        |
| 8.9      | Z Func . . . . .                 | 28        |
| 8.10     | Lcsubseq . . . . .               | 28        |
| 8.11     | Hash . . . . .                   | 28        |
| 8.12     | Suffix Array Radix . . . . .     | 29        |
| <b>9</b> | <b>ED</b>                        | <b>29</b> |
| 9.1      | Sparse Table . . . . .           | 29        |
| 9.2      | Color Update . . . . .           | 30        |
| 9.3      | Segtree Pa . . . . .             | 30        |
| 9.4      | Segtree Iterative Lazy . . . . . | 31        |
| 9.5      | Segtree Recursive . . . . .      | 31        |
| 9.6      | Segtree Maxsubarray . . . . .    | 31        |
| 9.7      | Segtree Implicita Lazy . . . . . | 32        |
| 9.8      | Segtree Iterative . . . . .      | 32        |
| 9.9      | Segtree Implicita . . . . .      | 32        |
| 9.10     | Mergesorttree . . . . .          | 33        |
| 9.11     | Segpersistente Mkthnum . . . . . | 33        |
| 9.12     | Cht . . . . .                    | 34        |
| 9.13     | Bit Kth . . . . .                | 34        |
| 9.14     | Bit . . . . .                    | 34        |
| 9.15     | Virtual Tree . . . . .           | 35        |
| 9.16     | Treap . . . . .                  | 35        |
| 9.17     | Minqueue . . . . .               | 36        |

# 1 Geometria

## 1.1 Linear Transformation

```
1 // Apply linear transformation (p -> q) to r.
2 point linear_transformation(point p0, point p1, point
  q0, point q1, point r) {
3   point dp = p1-p0, dq = q1-q0, num((dp^dq), (dp^dq
  ));
4   return q0 + point((r-p0)^(num), (r-p0)*(num))/(dp
  *dp);
5 }
```

## 1.2 Inside Polygon

```
1 // Convex O(logn)
2
3 bool insideT(point a, point b, point c, point e){
4   int x = ccw(a, b, e);
5   int y = ccw(b, c, e);
6   int z = ccw(c, a, e);
7   return !((x==1 or y==1 or z==1) and (x==-1 or y
  ==-1 or z==-1));
8 }
9
10 bool inside(vp &p, point e){ // ccw
11   int l=2, r=(int)p.size()-1;
12   while(l<r){
13     int mid = (l+r)/2;
14     if(ccw(p[0], p[mid], e) == 1)
15       l=mid+1;
16     else{
17       r=mid;
18     }
19   }
20   // bordo
21   // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)
  ==0) return false;
22   // if(r==2 and ccw(p[0], p[1], e)==0) return
  false;
23   // if(ccw(p[r], p[r-1], e)==0) return false;
24   return insideT(p[0], p[r-1], p[r], e);
25 }
26
27 // Any O(n)
28
29 int inside(vp &p, point pp){
30   // 1 - inside / 0 - boundary / -1 - outside
31   int n = p.size();
32   for(int i=0; i<n; i++){
33     int j = (i+1)%n;
34     if(line({p[i], p[j]}).inside_seg(pp))
35       return 0;
36   }
37   int inter = 0;
38   for(int i=0; i<n; i++){
39     int j = (i+1)%n;
40     if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
  [i], p[j], pp)==1)
41       inter++; // up
42     else if(p[j].x <= pp.x and pp.x < p[i].x and
  ccw(p[i], p[j], pp)==-1)
43       inter++; // down
44   }
45   if(inter%2==0) return -1; // outside
46   else return 1; // inside
47 }
48
49 }
```

## 1.3 Delaunay

```
1 typedef long long ll;
2
3 bool ge(const ll& a, const ll& b) { return a >= b; }
4 bool le(const ll& a, const ll& b) { return a <= b; }
5 bool eq(const ll& a, const ll& b) { return a == b; }
6 bool gt(const ll& a, const ll& b) { return a > b; }
7 bool lt(const ll& a, const ll& b) { return a < b; }
8 int sgn(const ll& a) { return a >= 0 ? a : 1 : 0 :
  -1; }
9
10 struct pt {
11   ll x, y;
12   pt() {}
13   pt(ll _x, ll _y) : x(_x), y(_y) {}
14   pt operator-(const pt& p) const {
15     return pt(x - p.x, y - p.y);
16   }
17   ll cross(const pt& p) const {
18     return x * p.y - y * p.x;
19   }
20   ll cross(const pt& a, const pt& b) const {
21     return (a - *this).cross(b - *this);
22   }
23   ll dot(const pt& p) const {
24     return x * p.x + y * p.y;
25   }
26   ll dot(const pt& a, const pt& b) const {
27     return (a - *this).dot(b - *this);
28   }
29   ll sqrLength() const {
30     return this->dot(*this);
31   }
32   bool operator==(const pt& p) const {
33     return eq(x, p.x) && eq(y, p.y);
34   }
35 };
36
37 const pt inf_pt = pt(1e18, 1e18);
38
39 struct QuadEdge {
40   pt origin;
41   QuadEdge* rot = nullptr;
42   QuadEdge* onext = nullptr;
43   bool used = false;
44   QuadEdge* rev() const {
45     return rot->rot;
46   }
47   QuadEdge* lnext() const {
48     return rot->rev()->onext->rot;
49   }
50   QuadEdge* oprev() const {
51     return rot->onext->rot;
52   }
53   pt dest() const {
54     return rev()->origin;
55   }
56 };
57
58 QuadEdge* make_edge(pt from, pt to) {
59   QuadEdge* e1 = new QuadEdge;
60   QuadEdge* e2 = new QuadEdge;
61   QuadEdge* e3 = new QuadEdge;
62   QuadEdge* e4 = new QuadEdge;
63   e1->origin = from;
64   e2->origin = to;
65   e3->origin = e4->origin = inf_pt;
66   e1->rot = e3;
67   e2->rot = e4;
68   e3->rot = e2;
69   e4->rot = e1;
70   e1->onext = e1;
71   e2->onext = e2;
72   e3->onext = e4;
```

```

73     e4->onext = e3;
74     return e1;
75 }
76
77 void splice(QuadEdge* a, QuadEdge* b) {
78     swap(a->onext->rot->onext, b->onext->rot->onext);
79     swap(a->onext, b->onext);
80 }
81
82 void delete_edge(QuadEdge* e) {
83     splice(e, e->oprev());
84     splice(e->rev(), e->rev()->oprev());
85     delete e->rev()->rot;
86     delete e->rev();
87     delete e->rot;
88     delete e;
89 }
90
91 QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
92     QuadEdge* e = make_edge(a->dest(), b->origin());
93     splice(e, a->lnext());
94     splice(e->rev(), b);
95     return e;
96 }
97
98 bool left_of(pt p, QuadEdge* e) {
99     return gt(p.cross(e->origin, e->dest()), 0);
100 }
101
102 bool right_of(pt p, QuadEdge* e) {
103     return lt(p.cross(e->origin, e->dest()), 0);
104 }
105
106 template <class T>
107 T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2,
108       T c3) {
109     return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 -
110       c1 * b3) +
111       a3 * (b1 * c2 - c1 * b2);
112 }
113
114 bool in_circle(pt a, pt b, pt c, pt d) {
115     // If there is __int128, calculate directly.
116     // Otherwise, calculate angles.
117     #if defined(__LP64__) || defined(_WIN64)
118         __int128 det = -det3<__int128>(b.x, b.y, b.
119           sqLength(), c.x, c.y,
120           c.sqLength(), d.x,
121           d.y, d.sqLength());
122         det += det3<__int128>(a.x, a.y, a.sqLength(), c.
123           x, c.y, c.sqLength(), d.x,
124           d.y, d.sqLength());
125         det -= det3<__int128>(a.x, a.y, a.sqLength(), b.
126           x, b.y, b.sqLength(), d.x,
127           d.y, d.sqLength());
128         det += det3<__int128>(a.x, a.y, a.sqLength(), b.
129           x, b.y, b.sqLength(), c.x,
130           c.y, c.sqLength());
131         return det > 0;
132     #else
133         auto ang = [(pt l, pt mid, pt r) {
134             ll x = mid.dot(l, r);
135             ll y = mid.cross(l, r);
136             long double res = atan2((long double)x, (long
137               double)y);
138             return res;
139         }];
140         long double kek = ang(a, b, c) + ang(c, d, a) -
141         ang(b, c, d) - ang(d, a, b);
142         if (kek > 1e-8)
143             return true;
144         else
145             return false;
146     #endif
147 }
148
149 pair<QuadEdge*, QuadEdge*> build_tr(int l, int r,
150   vector<pt>& p) {
151     if (r - l + 1 == 2) {
152         QuadEdge* res = make_edge(p[l], p[r]);
153         return make_pair(res, res->rev());
154     }
155     if (r - l + 1 == 3) {
156         QuadEdge* a = make_edge(p[l], p[l + 1]), *b =
157         make_edge(p[l + 1], p[r]);
158         splice(a->rev(), b);
159         int sg = sgn(p[l].cross(p[l + 1], p[r]));
160         if (sg == 0)
161             return make_pair(a, b->rev());
162         QuadEdge* c = connect(b, a);
163         if (sg == 1)
164             return make_pair(a, b->rev());
165         else
166             return make_pair(c->rev(), c);
167     }
168     int mid = (l + r) / 2;
169     QuadEdge* ldo, *ldi, *rdo, *rdi;
170     tie(ldo, ldi) = build_tr(l, mid, p);
171     tie(rdi, rdo) = build_tr(mid + 1, r, p);
172     while (true) {
173         if (left_of(rdi->origin, ldi)) {
174             ldi = ldi->lnext();
175             continue;
176         }
177         if (right_of(ldi->origin, rdi)) {
178             rdi = rdi->rev()->onext;
179             continue;
180         }
181         break;
182     }
183     QuadEdge* basel = connect(rdi->rev(), ldi);
184     auto valid = [&basel](QuadEdge* e) { return
185       right_of(e->dest(), basel); };
186     if (ldi->origin == ldo->origin)
187         ldo = basel->rev();
188     if (rdi->origin == rdo->origin)
189         rdo = basel;
190     while (true) {
191         QuadEdge* lcand = basel->rev()->onext;
192         if (valid(lcand)) {
193             while (in_circle(basel->dest(), basel->
194               origin, lcand->dest(),
195                 lcand->onext->dest())) {
196                 QuadEdge* t = lcand->onext;
197                 delete_edge(lcand);
198                 lcand = t;
199             }
200         }
201         QuadEdge* rcand = basel->oprev();
202         if (valid(rcand)) {
203             while (in_circle(basel->dest(), basel->
204               origin, rcand->dest(),
205                 rcand->oprev()->dest())) {
206                 QuadEdge* t = rcand->oprev();
207                 delete_edge(rcand);
208                 rcand = t;
209             }
210         }
211         if (!valid(lcand) && !valid(rcand))
212             break;
213         if (!valid(lcand) ||
214             (valid(rcand) && in_circle(lcand->dest(),
215               lcand->origin,
216                 rcand->origin,
217                 rcand->dest())))
218             return make_pair(lcand, rcand);
219     }
220 }

```

```

202         basel = connect(rcand, basel->rev());
203     else
204         basel = connect(basel->rev(), lcand->rev
205     ());
206 }
207 return make_pair(ldo, rdo);
208 }
209 vector<tuple<pt, pt, pt>> delaunay(vector<pt> p) {
210     sort(p.begin(), p.end(), [](const pt& a, const pt
211     & b) {
212         return lt(a.x, b.x) || (eq(a.x, b.x) && lt(a.
213     y, b.y));
214     });
215     auto res = build_tr(0, (int)p.size() - 1, p);
216     QuadEdge* e = res.first;
217     vector<QuadEdge*> edges = {e};
218     while (lt(e->onext->dest().cross(e->dest(), e->
219     origin), 0))
220         e = e->onext;
221     auto add = [&p, &e, &edges]() {
222         QuadEdge* curr = e;
223         do {
224             curr->used = true;
225             p.push_back(curr->origin);
226             edges.push_back(curr->rev());
227             curr = curr->lnext();
228         } while (curr != e);
229     };
230     add();
231     p.clear();
232     int kek = 0;
233     while (kek < (int)edges.size()) {
234         if (!(e = edges[kek++])->used)
235             add();
236     }
237     vector<tuple<pt, pt, pt>> ans;
238     for (int i = 0; i < (int)p.size(); i += 3) {
239         ans.push_back(make_tuple(p[i], p[i + 1], p[i

```

## 1.4 Convex Hull

```

1 vp convex_hull(const vp P)
2 {
3     sort(P.begin(), P.end());
4     vp L, U;
5     for(auto p: P){
6         while(L.size()>=2 and ccw(L[L.size()-2], L.
7     back(), p)!=1)
8             L.pop_back();
9         L.pb(p);
10    }
11    reverse(P.begin(), P.end());
12    for(auto p: P){
13        while(U.size()>=2 and ccw(U[U.size()-2], U.
14    back(), p)!=1)
15            U.pop_back();
16        U.pb(p);
17    }
18    L.pop_back();
19    L.insert(L.end(), U.begin(), U.end()-1);
20    return L;
21 }

```

## 1.5 Sort By Angle

```

1 int quarter(point a)
2 {

```

```

3     if(a.x>0 and a.y>=0) return 0;
4     if(a.x<=0 and a.y>0) return 1;
5     if(a.x<0 and a.y<=0) return 2;
6     return 3;
7 }
8
9 point c;
10 bool comp(point a, point b) //ccw
11 {
12     a=a-c;b=b-c;
13     int qa = quarter(a);
14     int qb = quarter(b);
15     if(qa==qb)
16         return (a^b)>0;
17     else
18         return qa<qb;
19 }
20
21 c = center(A);
22 sort(A.begin(), A.end(), comp);

```

## 1.6 Minkowski Sum

```

1 vp mk(const vp &a, const vp &b){
2     int i = 0, j = 0;
3     for(int k = 0; k < (int)a.size(); k++){if(a[k]<a[i
4     ]){
5         i = k;
6     }
7     for(int k = 0; k < (int)b.size(); k++){if(b[k]<b[j
8     ]){
9         j = k;
10    }
11    vp c;
12    c.reserve(a.size() + b.size());
13    for(int k = 0; k < int(a.size()+b.size()); k++){
14        point pt{a[i] + b[j]};
15        if((int)c.size() >= 2 and !ccw(c[c.size()-2],
16    c.back(), pt))
17            c.pop_back();
18        c.pb(pt);
19        int q = i+1, w = j+1;
20        if(q == int(a.size())) q = 0;
21        if(w == int(b.size())) w = 0;
22        if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
23    = q;
24        else j = w;
25    }
26    if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()-
27    2]))
28        c.pop_back();
29    if(!ccw(c.back(), c[0], c[1])){
30        c[0]=c.back();
31        c.pop_back();
32    }
33    c.shrink_to_fit();
34    return c;
35 }

```

## 1.7 Tetrahedron Distance3d

```

1 bool nulo(point a){
2     return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0))
3     ;
4 }
5 ld misto(point p1, point p2, point p3){
6     return (p1^p2)*p3;
7 }
8
9 ld dist_pt_face(point p, vp v){

```

```

10     assert(v.size()==3);
11
12     point v1 = v[1]-v[0];
13     point v2 = v[2]-v[0];
14     point n = (v1^v2);
15
16     for(int i=0;i<3;i++){
17         point va = p-v[i];
18         point vb = v[(i+1)%3]-v[i];
19         point ve = vb^n;
20         ld d = ve*v[i];
21         //se ponto coplanar com um dos lados do
prisma (va^vb eh nulo),
22         //ele esta dentro do prisma (poderia
desconsiderar pois distancia
23         //vai ser a msm da distancia do ponto ao
segmento)
24         if(!nulo(va^vb) and (v[(i+2)%3]*ve>d) ^ (p*ve
>d)) return LLINF;
25     }
26
27     //se ponto for coplanar ao triangulo (e dentro do
triangulo)
28     //vai retornar zero corretamente
29     return fabs(misto(p-v[0],v1,v2)/norm(n));
30 }
31
32 ld dist_pt_seg(point p, vp li){
33     return norm((li[1]-li[0])^(p-li[0]))/norm(li[1]-
li[0]);
34 }
35
36 ld dist_line(vp l1, vp l2){
37     point n = (l1[1]-l1[0])^(l2[1]-l2[0]);
38     if(nulo(n)) //retas paralelas - dist ponto a reta
39     return dist_pt_seg(l2[0],l1);
40
41     point o1o2 = l2[0]-l1[0];
42     return fabs((o1o2*n)/norm(n));
43 }
44 // retas paralelas e intersecao nao nula
45 ld dist_seg(vp l1, vp l2){
46
47     assert(l2.size()==2);
48     assert(l1.size()==2);
49
50     //pontos extremos do segmento
51     ld ans = LLINF;
52     for(int i=0;i<2;i++){
53         for(int j=0;j<2;j++){
54             ans = min(ans, norm(l1[i]-l2[j]));
55
56         //verificando distancia de ponto extremo com
ponto interno dos segs
57         for(int t=0;t<2;t++){
58             for(int i=0;i<2;i++){
59                 bool c=true;
60                 for(int k=0;k<2;k++){
61                     point va = l1[i]-l2[k];
62                     point vb = l2[!k]-l2[k];
63                     ld ang = atan2(norm((vb^va)), vb*va);
64                     if(ang>PI/2) c = false;
65                 }
66                 if(c)
67                     ans = min(ans, dist_pt_seg(l1[i],l2));
68             }
69             swap(l1,l2);
70         }
71
72         //ponto interno com ponto interno dos segmentos
73         point v1 = l1[1]-l1[0], v2 = l2[1]-l2[0];
74         point n = v1^v2;
75         if(!nulo(n)){
76             bool ok = true;
77             for(int t=0;t<2;t++){
78                 point n2 = v2^n;
79                 point o1o2 = l2[0]-l1[0];
80                 ld escalar = (o1o2*n2)/(v1*n2);
81                 if(escalar<0 or escalar>1) ok = false;
82                 swap(l1,l2);
83                 swap(v1,v2);
84             }
85             if(ok) ans = min(ans, dist_line(l1,l2));
86         }
87     }
88     return ans;
89 }
90
91 ld ver(vector<vp> &vet){
92     ld ans = LLINF;
93     // vertice - face
94     for(int k=0;k<2;k++){
95         for(int pt=0;pt<4;pt++){
96             for(int i=0;i<4;i++){
97                 vp v;
98                 for(int j=0;j<4;j++){
99                     if(i!=j) v.pb(vet[!k][j]);
100                 }
101                 ans = min(ans, dist_pt_face(vet[k][pt
], v));
102             }
103         }
104     }
105     // edge - edge
106     for(int i1=0;i1<4;i1++){
107         for(int j1=0;j1<i1;j1++){
108             for(int i2=0;i2<4;i2++){
109                 for(int j2=0;j2<i2;j2++){
110                     ans = min(ans, dist_seg({vet[0][
i1], vet[0][j1]},
111                                             {vet[1][
i2], vet[1][j2]}));
112                 }
113             }
114         }
115     }
116     return ans;
117 }
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

## 1.8 Numintersectionline

```

1 int main()
2 {
3     int lim = 1e6;
4     Segtree st(lim+100);
5     int n, m, y, x, l, r;
6     cin >> n >> m;
7
8     int open=-1, close=INF; // open -> check -> close
9     vector< pair<int, pii> > sweep;
10
11     ll ans = 0;
12     for(int i=0;i<n;i++){ // horizontal
13         cin >> y >> l >> r;
14         sweep.pb({l, {open, y}});
15         sweep.pb({r, {close, y}});
16     }
17     for(int i=0;i<m;i++){ // vertical
18         cin >> x >> l >> r;
19         sweep.pb({x, {l, r}});
20     }
21     sort(sweep.begin(), sweep.end());
22
23     // set<int> on;
24     for(auto s: sweep){
25         if(s.ss.ff==open){
26             st.update(s.ss.ss, 1);
27             // on.insert(s.ss.ss);
28         }
29         else if(s.ss.ff==close){
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

30         st.update(s.ss.ss, -1);
31         // on.erase(s.ss.ss);
32     }
33     else{
34         ans += st.query(s.ss.ff, s.ss.ss);
35         // auto it1 = on.lower_bound(s.ss.ff);
36         // auto it2 = on.upper_bound(s.ss.ss);
37         // for(auto it = it1; it!=it2; it++){
38         //     intersection -> (s.ff, it);
39         // }
40     }
41 }
42
43 cout << ans << endl;
44
45 return 0;
46 }

```

## 1.9 Polygon Diameter

```

1 double diameter(const vector<point> &p) {
2     vector<point> h = convexHull(p);
3     int m = h.size();
4     if (m == 1)
5         return 0;
6     if (m == 2)
7         return dist(h[0], h[1]);
8     int k = 1;
9     while (area(h[m - 1], h[0], h[(k + 1) % m]) >
10            area(h[m - 1], h[0], h[k]))
11         ++k;
12     double res = 0;
13     for (int i = 0, j = k; i <= k && j < m; i++) {
14         res = max(res, dist(h[i], h[j]));
15         while (j < m && area(h[i], h[(i + 1) % m], h
16            [(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j]))
17             ++j;
18     }
19     return res;
20 }

```

## 1.10 Polygon Cut Length

```

1 // Polygon Cut length
2 ld solve(vp &p, point a, point b){ // ccw
3     int n = p.size();
4     ld ans = 0;
5
6     for(int i=0;i<n;i++){
7         int j = (i+1) % n;
8
9         int signi = ccw(a, b, p[i]);
10        int signj = ccw(a, b, p[j]);
11
12        if(signi == 0 and signj == 0){
13            if((b-a) * (p[j]-p[i]) > 0){
14                ans += param(a, b, p[j]);
15                ans -= param(a, b, p[i]);
16            }
17        }else if(signi <= 0 and signj > 0){
18            ans -= param(a, b, inter_line({a, b}, {p[
19            i], p[j]})[0]);
20        }else if(signi > 0 and signj <= 0){
21            ans += param(a, b, inter_line({a, b}, {p[
22            i], p[j]})[0]);
23        }
24    }
25 }

```

```

23
24     return abs(ans * norm(b-a));
25 }

```

## 1.11 Mindistpair

```

1 ll MinDistPair(vp &vet){
2     int n = vet.size();
3     sort(vet.begin(), vet.end());
4     set<point> s;
5
6     ll best_dist = LLINF;
7     int j=0;
8     for(int i=0;i<n;i++){
9         ll d = ceil(sqrt(best_dist));
10        while(j<n and vet[i].x-vet[j].x >= d){
11            s.erase(point(vet[j].y, vet[j].x));
12            j++;
13        }
14
15        auto it1 = s.lower_bound({vet[i].y - d, vet[i
16        ].x});
17        auto it2 = s.upper_bound({vet[i].y + d, vet[i
18        ].x});
19
20        for(auto it=it1; it!=it2; it++){
21            ll dx = vet[i].x - it->x;
22            ll dy = vet[i].y - it->y;
23            if(best_dist > dx*dx + dy*dy){
24                best_dist = dx*dx + dy*dy;
25                // vet[i] e inv(it)
26            }
27        }
28        s.insert(point(vet[i].y, vet[i].x));
29    }
30    return best_dist;
31 }

```

## 1.12 Rotating Callipers

```

1 int N;
2
3 int sum(int i, int x){
4     if(i+x>N-1) return (i+x-N);
5     return i+x;
6 }
7
8 ld rotating_callipers(vp &vet){
9     N = vet.size();
10    ld ans = 0;
11    // 2 triangulos (p1, p3, p4) (p1, p2, p3);
12    for(int i=0;i<N;i++){ // p1
13        int p2 = sum(i, 1); // p2
14        int p4 = sum(i, 3); // p4
15        for(int j=sum(i, 2);j!=i;j=sum(j, 1)){ // p3
16            if(j==p2) p2 = sum(p2, 1);
17            while(sum(p2, 1)!=j and areaT(vet[p2],
18            vet[i], vet[j]) < areaT(vet[sum(p2, 1)], vet[i],
19            vet[j]))
20                p2 = sum(p2, 1);
21            while(sum(p4, 1)!=i and areaT(vet[p4],
22            vet[i], vet[j]) < areaT(vet[sum(p4, 1)], vet[i],
23            vet[j]))
24                p4 = sum(p4, 1);
25
26            ans = max(ans, area(vet[i], vet[p2], vet[
27            j], vet[p4]));
28        }
29    }
30    return ans;
31 }

```

```
27 }
```

## 1.13 Half Plane Intersect

```
1 // Half plane intersect O(n3)
2 vp half_plane_intersect(vector<line> &v){
3     vp ret;
4     int n = v.size();
5     for(int i=0; i<n; i++){
6         for(int j=i+1; j<n; j++){
7             point crs = inter(v[i], v[j]);
8             if(crs.x == INF) continue;
9             bool bad = 0;
10            for(int k=0; k<n; k++){
11                if(v[k].eval(crs) < -EPS){
12                    bad = 1;
13                    break;
14                }
15            }
16            if(!bad) ret.push_back(crs);
17        }
18    }
19    return ret;
20 }
```

## 1.14 2d

```
1 #define vp vector<point>
2
3 // typedef ll cod;
4 // bool eq(cod a, cod b){ return (a==b); }
5 typedef ld cod;
6 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
7
8 struct point{
9     cod x, y;
10    int id;
11    point(cod x=0, cod y=0): x(x), y(y){}
12
13    point operator+(const point &o) const{
14        return {x+o.x, y+o.y};
15    }
16    point operator-(const point &o) const{
17        return {x-o.x, y-o.y};
18    }
19    point operator*(cod t) const{
20        return {x*t, y*t};
21    }
22    point operator/(cod t) const{
23        return {x/t, y/t};
24    }
25    cod operator*(const point &o) const{ // dot
26        return x * o.x + y * o.y;
27    }
28    cod operator^(const point &o) const{ // cross
29        return x * o.y - y * o.x;
30    }
31    bool operator<(const point &o) const{
32        if(!eq(x, o.x)) return x < o.x;
33        return y < o.y;
34    }
35    bool operator==(const point &o) const{
36        return eq(x, o.x) and eq(y, o.y);
37    }
38 }
39
40 };
41
42 ld norm(point a){ // Modulo
43     return sqrt(a*a);
44 }
45
```

```
46 int ccw(point a, point b, point e){ // -1=dir; 0=
47     collinear; 1=esq;
48     cod tmp = (b-a)^(e-a); // from a to b
49     return (tmp > EPS) - (tmp < -EPS);
50 }
51 bool nulo(point a){
52     return (eq(a.x, 0) and eq(a.y, 0));
53 }
54 point rotccw(point p, ld a){
55     // a = PI*a/180; // graus
56     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
57     +p.x*sin(a)));
58 }
59 point rot90cw(point a) { return point(a.y, -a.x); };
60 point rot90ccw(point a) { return point(-a.y, a.x); };
61 ld proj(point a, point b){ // a sobre b
62     return a*b/norm(b);
63 }
64 ld angle(point a, point b){ // em radianos
65     ld ang = a*b / norm(a) / norm(b);
66     return acos(max(min(ang, (ld)1), (ld)-1));
67 }
68 ld angle_vec(point v){
69     // return 180/PI*atan2(v.x, v.y); // graus
70     return atan2(v.x, v.y);
71 }
72 ld order_angle(point a, point b){ // from a to b ccw
73     (a in front of b)
74     ld aux = angle(a,b)*180/PI;
75     return ((a^b)<=0 ? aux:360-aux);
76 }
77 bool angle_less(point a1, point b1, point a2, point
78     b2){ // ang(a1,b1) <= ang(a2,b2)
79     point p1((a1*b1), abs((a1^b1)));
80     point p2((a2*b2), abs((a2^b2)));
81     return (p1^p2) <= 0;
82 }
83 ld area(vp &p){ // (points sorted)
84     ld ret = 0;
85     for(int i=2;i<(int)p.size();i++)
86         ret += (p[i]-p[0])^(p[i-1]-p[0]);
87     return abs(ret/2);
88 }
89 ld areaT(point &a, point &b, point &c){
90     return abs((b-a)^(c-a))/2.0;
91 }
92 point center(vp &A){
93     point c = point();
94     int len = A.size();
95     for(int i=0;i<len;i++)
96         c=c+A[i];
97     return c/len;
98 }
99
100 point forca_mod(point p, ld m){
101     ld cm = norm(p);
102     if(cm<EPS) return point();
103     return point(p.x*m/cm, p.y*m/cm);
104 }
105
106 point mirror(point m1, point m2, point p){
107     // mirror point p around segment m1m2
108     point seg = m2-m1;
109     ld t0 = ((p-m1)*seg) / (seg*seg);
110     point ort = m1 + seg*t0;
111     point pm = ort-(p-ort);
112     return pm;
113 }
114
```



```

115 ld param(point a, point b, point v){
116     // v = t*(b-a) + a // return t;
117     // assert(line(a, b).inside_seg(v));
118     return ((v-a) * (b-a)) / ((b-a) * (b-a));
119 }
120
121 bool simetric(vector<point> &a){ //ordered
122     int n = a.size();
123     c = center(a);
124     if(n&1) return false;
125     for(int i=0;i<n/2;i++)
126         if(!collinear(a[i], a[i+n/2], c))
127             return false;
128     return true;
129 }
130
131
132
133 ///////////////
134 // Line //
135 ///////////////
136
137 struct line{
138     point p1, p2;
139     cod a, b, c; // ax+by+c = 0;
140     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
141     line(point p1=0, point p2=0): p1(p1), p2(p2){
142         a = p1.y-p2.y;
143         b = p2.x-p1.x;
144         c = -(a*p1.x + b*p1.y);
145     }
146     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c){
147         // Gera os pontos p1 p2 dados os coeficientes
148         // isso aqui eh um lixo mas quebra um galho
149         kkkkkk
150         if(b==0){
151             p1 = point(1, -c/a);
152             p2 = point(0, -c/a);
153         }else{
154             p1 = point(1, (-c-a*1)/b);
155             p2 = point(0, -c/b);
156         }
157     }
158     cod eval(point p){
159         return a*p.x+b*p.y+c;
160     }
161     bool inside(point p){
162         return eq(eval(p), 0);
163     }
164     point normal(){
165         return point(a, b);
166     }
167
168     bool inside_seg(point p){
169         return (
170             ((p1-p) ^ (p2-p)) == 0 and
171             ((p1-p) * (p2-p)) <= 0
172         );
173     }
174
175 };
176
177 // be careful with precision error
178 vp inter_line(line l1, line l2){
179     ld det = l1.a*l2.b - l1.b*l2.a;
180     if(det==0) return {};
181     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
182     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
183     return {point(x, y)};
184 }
185
186 // segments not collinear
187 vp inter_seg(line l1, line l2){
188     vp ans = inter_line(l1, l2);
189     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
190         inside_seg(ans[0]))
191         return {};
192     return ans;
193 }
194
195 ld dist_seg(point p, point a, point b){ // point -
196     seg
197     if(((p-a)*(b-a)) < EPS) return norm(p-a);
198     if(((p-b)*(a-b)) < EPS) return norm(p-b);
199     return abs((p-a)^(b-a))/norm(b-a);
200 }
201
202 ld dist_line(point p, line l){ // point - line
203     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
204 }
205
206 line bisector(point a, point b){
207     point d = (b-a)*2;
208     return line(d.x, d.y, a*a - b*b);
209 }
210
211 line perpendicular(line l, point p){ // passes
212     through p
213     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
214 }
215
216 // Circle //
217
218 struct circle{
219     point c; cod r;
220     circle() : c(0, 0), r(0){}
221     circle(const point o) : c(o), r(0){}
222     circle(const point a, const point b){
223         c = (a+b)/2;
224         r = norm(a-c);
225     }
226     circle(const point a, const point b, const point
227         cc){
228         c = inter_line(bisector(a, b), bisector(b, cc
229         ));
230         r = norm(a-c);
231     }
232     bool inside(const point &a) const{
233         return norm(a - c) <= r + EPS;
234     }
235 };
236
237 pair<point, point> getTangentPoint(circle cr, point p
238 ) {
239     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
240     point p1 = rotccw(cr.c-p, -theta);
241     point p2 = rotccw(cr.c-p, theta);
242     assert(d1 >= cr.r);
243     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
244     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
245     return {p1, p2};
246 }
247
248 circle incircle(point p1, point p2, point p3){
249     ld m1 = norm(p2-p3);
250     ld m2 = norm(p1-p3);
251     ld m3 = norm(p1-p2);
252     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
253     ld s = 0.5*(m1+m2+m3);
254     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;

```

```

253     return circle(c, r);
254 }
255
256 circle circumcircle(point a, point b, point c) {
257     circle ans;
258     point u = point((b-a).y, -(b-a).x);
259     point v = point((c-a).y, -(c-a).x);
260     point n = (c-b)*0.5;
261     ld t = (u~n)/(v~u);
262     ans.c = ((a+c)*0.5) + (v*t);
263     ans.r = norm(ans.c-a);
264     return ans;
265 }
266
267 vp inter_circle_line(circle C, line L){
268     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)*(ab) / (ab*ab));
269     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s / (ab*ab);
270     if (h2 < -EPS) return {};
271     if (eq(h2, 0)) return {p};
272     point h = (ab/norm(ab)) * sqrt(h2);
273     return {p - h, p + h};
274 }
275
276 vp inter_circle(circle C1, circle C2){
277     if(C1.c == C2.c) { assert(C1.r != C2.r); return {}; }
278     point vec = C2.c - C1.c;
279     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r;
280     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 = C1.r*C1.r - p*p*d2;
281     if (sum*sum < d2 or dif*dif > d2) return {};
282     point mid = C1.c + vec*p, per = point(-vec.y, vec.x) * sqrt(max((ld)0, h2) / d2);
283     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
284     return {mid + per, mid - per};
285 }
286
287 // minimum circle cover O(n) amortizado
288 circle min_circle_cover(vector<point> v){
289     random_shuffle(v.begin(), v.end());
290     circle ans;
291     int n = v.size();
292     for(int i=0; i<n; i++) if(!ans.inside(v[i])){
293         ans = circle(v[i]);
294         for(int j=0; j<i; j++) if(!ans.inside(v[j])){
295             ans = circle(v[i], v[j]);
296             for(int k=0; k<j; k++) if(!ans.inside(v[k])){
297                 ans = circle(v[i], v[j], v[k]);
298             }
299         }
300     }
301     return ans;
302 }

```

## 1.15 Intersect Polygon

```

1 bool intersect(vector<point> A, vector<point> B) //
  Ordered ccw
2 {
3     for(auto a: A)
4         if(inside(B, a))
5             return true;
6     for(auto b: B)
7         if(inside(A, b))
8             return true;
9
10    if(inside(B, center(A)))
11        return true;
12

```

```

13     return false;
14 }

```

## 1.16 3d

```

1 // typedef int cod;
2 // bool eq(cod a, cod b){ return (a==b); }
3
4 #define vp vector<point>
5 typedef ld cod;
6 bool eq(cod a, cod b){ return fabs(a - b) <= EPS; }
7
8 struct point
9 {
10     cod x, y, z;
11     point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z) {}
12
13     point operator+(const point &o) const{
14         return {x+o.x, y+o.y, z+o.z};
15     }
16     point operator-(const point &o) const{
17         return {x-o.x, y-o.y, z-o.z};
18     }
19     point operator*(cod t) const{
20         return {x*t, y*t, z*t};
21     }
22     point operator/(cod t) const{
23         return {x/t, y/t, z/t};
24     }
25     bool operator==(const point &o) const{
26         return eq(x, o.x) and eq(y, o.y) and eq(z, o.z);
27     }
28     cod operator*(const point &o) const{ // dot
29         return x*o.x + y*o.y + z*o.z;
30     }
31     point operator^(const point &o) const{ // cross
32         return point(y*o.z - z*o.y,
33                     z*o.x - x*o.z,
34                     x*o.y - y*o.x);
35     }
36 };
37
38 ld dist(point a, point b){
39     return sqrt((a-b)*(a-b));
40 }
41 bool nulo(point a){
42     return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0));
43 }
44
45 ld norm(point a){ // Modulo
46     return sqrt(a*a);
47 }
48 ld proj(point a, point b){ // a sobre b
49     return (a*b)/norm(b);
50 }
51 ld angle(point a, point b){ // em radianos
52     return acos((a*b) / norm(a) / norm(b));
53 }
54
55 cod triple(point a, point b, point c){
56     return (a * (b^c)); // Area do paralelepipedo
57 }
58
59 point normilize(point a){
60     return a/norm(a);
61 }
62
63 struct plane{
64     cod a, b, c, d;
65     point p1, p2, p3;

```

```

66 plane(point p1=0, point p2=0, point p3=0): p1(p1) 33 // End Template //
67 , p2(p2), p3(p3){ 34
68     point aux = (p1-p3)^(p2-p3); 35 const int N = 2e5+10;
69     a = aux.x; b = aux.y; c = aux.z; 36
70     d = -a*p1.x - b*p1.y - c*p1.z; 37 struct DSU {
71 } 38     int n;
72 plane(point p, point normal){ 39     map<int, int> parent;
73     normal = normilize(normal); 40     map<int, vi> comp;
74     a = normal.x; b = normal.y; c = normal.z; 41
75     d = -(p*normal); 42     int find(int v) {
76 } 43         if(v==parent[v])
77 // ax+by+cz+d = 0; 44             return v;
78 cod eval(point &p){ 45         return parent[v]=find(parent[v]);
79     return a*p.x + b*p.y + c*p.z + d; 46     }
80 } 47
81 }; 48 void join(int a, int b) {
82 49     a = find(a);
83 cod dist(plane pl, point p){ 50     b = find(b);
84     return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d 51     if(a!=b) {
85 ) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c); 52         if((int)comp[a].size()<(int)comp[b].size
86 } 53         swap(a, b);
87 point rotate(point v, point k, ld theta){ 54
88 // Rotaciona o vetor v theta graus em torno do 55         for(auto v: comp[b])
89 eixo k 56             comp[a].pb(v);
90 // theta *= PI/180; // graus 57     comp[b].clear();
91 return ( 58     parent[b]=a;
92     v*cos(theta)) + 59 }
93 ((k^v)*sin(theta)) + 60
94 (k*(k*v))*(1-cos(theta)) 61 }
95 } 62 };
63
64 int trie[MAX][2];
65 set<int> idx[MAX];
66 int finish[MAX];
67 int nxt = 1;
68
69 void add(int s){
70     int node = 0;
71     for(int i=30;i>=0;i--){
72         bool c = (s & (1<<i));
73         if(trie[node][c] == 0)
74             node = trie[node][c] = nxt++;
75         else
76             node = trie[node][c];
77         finish[node]++;
78     }
79 }
80
81 void remove(int s){
82     int node = 0;
83     for(int i=30;i>=0;i--){
84         bool c = (s & (1<<i));
85         node = trie[node][c];
86         finish[node]--;
87     }
88 }
89
90 int min_xor(int s){
91     int node = 0;
92     int ans = 0;
93     for(int i=30;i>=0;i--){
94         bool c = (s & (1<<i));
95         if(finish[trie[node][c]] != 0)
96             node = trie[node][c];
97         else{
98             ans ^= 1 << i;
99             node = trie[node][!c];
100         }
101     }
102     return ans;
103 }
104

```

## 2 Algoritmos

### 2.1 Mst Xor

```

1 // omg why just 2 seconds
2 #include <bits/stdc++.h>
3 // #define int long long
4 #define ff first
5 #define ss second
6 #define ll long long
7 #define ld long double
8 #define pb push_back
9 #define eb emplace_back
10 #define pii pair<int, int>
11 #define pll pair<ll, ll>
12 #define ti tuple<int, int, int>
13 #define vi vector<int>
14 #define vl vector<ll>
15 #define vii vector<pii>
16 #define sws ios_base::sync_with_stdio(false);cin.tie(
17 NULL);cout.tie(NULL);
18 #define endl '\n'
19 #define teto(a, b) (((a)+(b)-1)/(b))
20 #define all(x) x.begin(), x.end()
21 #define forn(i, n) for(int i = 0; i < (int)n; i++)
22 #define forne(i, a, b) for(int i = a; i <= b; i++)
23 #define dbg(msg, var) cerr << msg << " " << var <<
24 endl;
25
26 using namespace std;
27
28 const int MAX = 6e6+10;
29 const ll MOD = 1e9+7;
30 const int INF = 0x3f3f3f3f;
31 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
32 const ld EPS = 1e-6;
33 const ld PI = acos(-1);
34

```

```

105
106 int32_t main()
107 {sws;
108
109     int n;
110     cin >> n;
111     vi x(n);
112     for(int i=0;i<n;i++){
113         cin >> x[i];
114
115         sort(x.begin(), x.end());
116         x.erase(unique(x.begin(), x.end()), x.end());
117         n = x.size();
118
119         DSU dsu;
120
121         ll mstsum = 0;
122
123         vi pais;
124         for(int i=0;i<n;i++){
125             add(x[i]);
126             dsu.parent[x[i]] = x[i];
127             dsu.comp[x[i]].pb(x[i]);
128             pais.pb(x[i]);
129         }
130
131         while((int)pais.size()!=1){
132             vector<ti> edges;
133             for(auto p: pais){
134                 vi &nodes = dsu.comp[p];
135                 // erase
136                 for(auto u: nodes) remove(u);
137
138                 // query
139                 ti ed = {LLINF, 0, 0};
140                 for(auto u: nodes){
141                     int xr = min_xor(u);
142                     ed = min(ed, {xr, u, xr^u});
143                 }
144                 edges.pb(ed);
145
146                 // add back
147                 for(auto u: nodes) add(u);
148             }
149
150             for(auto [xr, u, v]: edges){
151                 if(dsu.find(u)!=dsu.find(v)){
152                     // u, v -> mst
153                     // cout << "mst = " << u << " " << v
154
155                     mstsum += xr;
156                     dsu.join(u, v);
157                 }
158             }
159             vi pais2;
160             for(auto p: pais)
161                 if(p==dsu.find(p))
162                     pais2.pb(p);
163             swap(pais, pais2);
164
165             cout << mstsum << endl;
166
167             return 0;
168         }
169 }

```

## 2.2 Cdq

```

1 // LIS 3D problem
2
3 struct Segtree{
4     vi t;

```

```

5     int n;
6
7     Segtree(int n){
8         this->n = n;
9         t.assign(2*n, 0);
10    }
11
12    int merge(int a, int b){
13        return max(a, b);
14    }
15
16    void build(){
17        for(int i=n-1;i>0;i--){
18            t[i] = merge(t[i<<1], t[i<<1|1]);
19        }
20
21    int query(int l, int r){
22        int resl = -INF, resr = -INF;
23        for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
24            if(l&1) resl = merge(resl, t[l++]);
25            if(r&1) resr = merge(t[--r], resr);
26        }
27        return merge(resl, resr);
28    }
29
30    void update(int p, int value){
31        p+=n;
32        for(t[p]=max(t[p], value); p >>= 1;){
33            t[p] = merge(t[p<<1], t[p<<1|1]);
34        }
35    };
36
37    struct point{
38        int x, y, z, id;
39        bool left;
40        point(int x=0, int y=0, int z=0): x(x), y(y), z(z)
41        ){
42            left = false;
43        }
44        bool operator<(point &o){
45            if(x != o.x) return x < o.x;
46            if(y != o.y) return y > o.y;
47            return z < o.z;
48        }
49    };
50
51    void cdq(int l, int r, vector<point> &a, vi &dp){
52        if(l==r) return;
53
54        int mid = (l+r) / 2;
55
56        cdq(l, mid, a, dp);
57
58        // compress z
59        set<int> uz; map<int, int> idz;
60        for(int i=l;i<=r;i++) uz.insert(a[i].z);
61        int id = 0;
62        for(auto z: uz) idz[z] = id++;
63
64        vector<point> tmp;
65        for(int i=l;i<=r;i++){
66            tmp.pb(a[i]);
67            tmp.back().x = 0;
68            tmp.back().z = idz[tmp.back().z];
69            if(i<=mid)
70                tmp.back().left = true;
71        }
72
73        Segtree st(id);
74
75        sort(tmp.rbegin(), tmp.rend());
76

```

```

77     for(auto t: tmp){
78         if(t.left){
79             st.update(t.z, dp[t.id]);
80         }else{
81             dp[t.id] = max(dp[t.id], st.query(0, t.z
82             -1)+1);
83         }
84     }
85     cdq(mid+1, r, a, dp);
86 }
87
88 int32_t main()
89 {sws;
90
91     int n; cin >> n;
92
93     vector<point> vet(n);
94     for(int i=0;i<n;i++){
95         cin >> vet[i].x >> vet[i].y >> vet[i].z;
96     }
97
98     sort(vet.begin(), vet.end());
99
100     for(int i=0;i<n;i++){
101         vet[i].id = i;
102     }
103
104     vi dp(n, 1);
105
106     cdq(0, n-1, vet, dp);
107
108     int ans = 0;
109     for(int i=0;i<n;i++){
110         ans = max(ans, dp[i]);
111     }
112
113     cout << ans << endl;
114
115     return 0;
116 }

```

## 2.3 Histogram Rectangle

```

1 ll bestRectangle(vi hist){
2     int n = hist.size();
3     stack<ll> s;
4     s.push(-1);
5     ll ans = hist[0];
6     vl left_smaller(n, -1), right_smaller(n, n);
7     for(int i=0;i<n;i++){
8         while(!s.empty() and s.top() != -1 and hist[s.
9         top()]>hist[i]){
10             right_smaller[s.top()] = i;
11             s.pop();
12         }
13         if(i>0 and hist[i]==hist[i-1])
14             left_smaller[i] = left_smaller[i-1];
15         else
16             left_smaller[i] = s.top();
17         s.push(i);
18     }
19     for(int j=0;j<n;j++){
20         ll area = hist[j]*(right_smaller[j]-
21         left_smaller[j]-1);
22         ans = max(ans, area);
23     }
24     return ans;
25 }

```

## 3 Misc

### 3.1 Rand

```

1 mt19937 rng(chrono::steady_clock::now().
2     time_since_epoch().count());
3 uniform_int_distribution<int> distribution(1,n);
4
5 num = distribution(rng); // num no range [1, n]
6 shuffle(vec.begin(), vec.end(), rng); // shuffle
7
8 ull mix(ull o){
9     o+=0x9e3779b97f4a7c15;
10    o=(o^(o>>30))*0xbf58476d1ce4e5b9;
11    o=(o^(o>>27))*0x94d049bb133111eb;
12    return o^(o>>31);
13 }
14 ull hash(pii a) {return mix(a.first ^ mix(a.second))
15 ;}

```

### 3.2 Bitwise

```

1 // Bitwise
2 #pragma GCC target("popcnt")
3 unsigned char a = 5, b = 9; // a = (00000101), b
4 = (00001001)
5
6 AND - a&b // The result is 00000001
7 (1)
8 OR - a|b // The result is 00001101
9 (13)
10 XOR - a^b // The result is 00001100
11 (12)
12 NOT - ~a // The result is 11111010
13 (250)
14 Left shift - b<<1 // The result is 00010010
15 (18)
16 Right shift - b>>1 // The result is 00000100
17 (4)
18
19 // Exchange two int variables
20
21 a^=b;
22 b^=a;
23 a^=b;
24
25 // Even or Odd
26
27 (x & 1)? printf("Odd"): printf("Even");
28
29 // Turn on the j-th bit
30
31 int S = 34; //(100010)
32 int j = 3;
33
34 S = S | (1<<j);
35
36 // Turn off the j-th bit
37
38 int S = 42; //(101010)
39 int j = 1;
40
41 S &= ~(1<<j)
42
43 S == 40 //(101000)
44
45 // Check the j-th element
46
47 int S = 42; //(101010)
48 int j = 3;
49
50

```

```

43     T = S & (1<<j); // T = 0
44
45 // Least significant bit (lsb)
46
47     int lsb(int x){ return x&-x; }
48
49 // Exchange o j-th element
50
51     S ^= (1<<j)
52
53 // Position of the first bit on
54
55     T = (S & (-S))
56     T -> 4 bit ligado //(1000)
57
58 // Most significant digit of N
59
60     double K = log10(N);
61     K = K - floor(K);
62     int X = pow(10, K);
63
64 // Number of digits in N
65
66     X =floor(log10(N)) + 1;
67
68 // Power of two
69
70     bool isPowerOfTwo(int x){ return x && (!(x&(x
-1)))}; }
71
72 // Turn off the first bit 1
73     m = m & (m-1);
74
75 // Built-in functions
76
77     // Number of bits 1
78     __builtin_popcount()
79     __builtin_popcountll()
80
81     // Number of leading zeros
82     __builtin_clz()
83     __builtin_clzll()
84
85     // Number of trailing zeros
86     __builtin_ctz()
87     __builtin_ctzll()
88
89 // floor(log2(x))
90
91     int flog2(int x){ return 32-1-__builtin_clz(x
); }
92
93     int flog2ll(ll x){ return 64-1-
__builtin_clzll(x); }

```

### 3.3 Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 using ll = long long;
5 using ii = pair<int, int>;
6 using vi = vector<int>;
7 #define pb emplace_back
8 #define mp make_pair
9
10 const ll MOD = 998'244'353;
11 const int MAX = 2e5 + 5;

```

### 3.4 Ordered Set

```

1 #include <ext/pb_ds/assoc_container.hpp>

```

```

2 #include <ext/pb_ds/tree_policy.hpp>
3
4 #include <ext/pb_ds/detail/standard_policies.hpp>
5
6 using namespace __gnu_pbds; // or pb_ds;
7
8 template<typename T, typename B = null_type>
9 using ordered_set = tree<T, B, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
10 // find_by_order / order_of_key

```

## 4 Math

### 4.1 Randommod

```

1 int randommod() {
2     auto primo = [](int num) {
3         for(int i = 2; i*i <= num; i++) {
4             if(num%i == 0) return false;
5         }
6         return true;
7     };
8     uniform_int_distribution<int> distribution
    (10000000007, 15000000000);
9     int num = distribution(rng);
10    while(!primo(num)) num++;
11    return num;
12 }

```

### 4.2 Division Trick

```

1 for(int l = 1, r; l <= n; l = r + 1) {
2     r = n / (n / l);
3     // n / i has the same value for l <= i <= r
4 }

```

### 4.3 Inverso Mult

```

1 // gcd(a, m) = 1 para existir solucao
2 // ax + my = 1, ou a*x = 1 (mod m)
3 ll inv(ll a, ll m) { // com gcd
4     ll x, y;
5     gcd(a, m, x, y);
6     return ((x % m) + m) % m;
7 }
8
9 ll inv(ll a, ll phim) { // com phi(m), se m for primo
    entao phi(m) = p-1
10     ll e = phim-1;
11     return fexp(a, e);
12 }

```

### 4.4 Crt

```

1 tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
2     if (!a) return {b, 0, 1};
3     auto [g, x, y] = ext_gcd(b%a, a);
4     return {g, y - b/a*x, x};
5 }
6
7 struct crt {
8     ll a, m;
9
10    crt() : a(0), m(1) {}
11    crt(ll a_, ll m_) : a(a_), m(m_) {}
12    crt operator * (crt C) {
13        auto [g, x, y] = ext_gcd(m, C.m);
14        if ((a - C.a) % g) a = -1;
15        if (a == -1 or C.a == -1) return crt(-1, 0);
16        ll lcm = m/g*C.m;

```

```

17         ll ans = a + (x*(C.a-a)/g % (C.m/g))*m;
18         return crt((ans % lcm + lcm) % lcm, lcm);
19     }
20 };

```

## 4.5 Gaussxor

```

1 struct Gauss {
2     array<ll, LOG_MAX> vet;
3     int size;
4     Gauss() : size(0) {
5         fill(vet.begin(), vet.end(), 0);
6     }
7     Gauss(vector<ll> vals) : size(0) {
8         fill(vet.begin(), vet.end(), 0);
9         for(ll val : vals) add(val);
10    }
11    bool add(ll val) {
12        for(int i = LOG_MAX-1; i >= 0; i--) if(val &
13            (1LL << i)) {
14            if(vet[i] == 0) {
15                vet[i] = val;
16                size++;
17                return true;
18            }
19            val ^= vet[i];
20        }
21        return false;
22    }
23 };

```

## 4.6 Pollard Rho

```

1 ll mul(ll a, ll b, ll m) {
2     ll ret = a*b - (ll)((ld)1/m*a*b+0.5)*m;
3     return ret < 0 ? ret+m : ret;
4 }
5
6 ll pow(ll a, ll b, ll m) {
7     ll ans = 1;
8     for (; b > 0; b /= 2ll, a = mul(a, a, m)) {
9         if (b % 2ll == 1)
10            ans = mul(ans, a, m);
11    }
12    return ans;
13 }
14
15 bool prime(ll n) {
16     if (n < 2) return 0;
17     if (n <= 3) return 1;
18     if (n % 2 == 0) return 0;
19
20     ll r = __builtin_ctzll(n - 1), d = n >> r;
21     for (int a : {2, 325, 9375, 28178, 450775,
22         9780504, 795265022}) {
23         ll x = pow(a, d, n);
24         if (x == 1 or x == n - 1 or a % n == 0)
25             continue;
26
27         for (int j = 0; j < r - 1; j++) {
28             x = mul(x, x, n);
29             if (x == n - 1) break;
30         }
31         if (x != n - 1) return 0;
32     }
33     return 1;
34 }
35
36 ll rho(ll n) {
37     if (n == 1 or prime(n)) return n;
38     auto f = [n](ll x) {return mul(x, x, n) + 1;};
39 }

```

```

38     ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
39     while (t % 40 != 0 or gcd(prd, n) == 1) {
40         if (x==y) x = ++x0, y = f(x);
41         q = mul(prd, abs(x-y), n);
42         if (q != 0) prd = q;
43         x = f(x), y = f(f(y)), t++;
44     }
45     return gcd(prd, n);
46 }
47
48 vector<ll> fact(ll n) {
49     if (n == 1) return {};
50     if (prime(n)) return {n};
51     ll d = rho(n);
52     vector<ll> l = fact(d), r = fact(n / d);
53     l.insert(l.end(), r.begin(), r.end());
54     return l;
55 }

```

## 4.7 Fast Exponentiaion

```

1 ll fexp(ll b, ll e, ll mod) {
2     ll res = 1;
3     b %= mod;
4     while(e){
5         if(e & 1LL)
6             res = (res * b) % mod;
7         e = e >> 1LL;
8         b = (b * b) % mod;
9     }
10    return res;
11 }

```

## 4.8 Linear Diophantine Equation

```

1 // Linear Diophantine Equation
2 int gcd(int a, int b, int &x, int &y)
3 {
4     if (a == 0)
5     {
6         x = 0; y = 1;
7         return b;
8     }
9     int x1, y1;
10    int d = gcd(b%a, a, x1, y1);
11    x = y1 - (b / a) * x1;
12    y = x1;
13    return d;
14 }
15
16 bool find_any_solution(int a, int b, int c, int &x0,
17     int &y0, int &g)
18 {
19     g = gcd(abs(a), abs(b), x0, y0);
20     if (c % g)
21         return false;
22
23     x0 *= c / g;
24     y0 *= c / g;
25     if (a < 0) x0 = -x0;
26     if (b < 0) y0 = -y0;
27     return true;
28 }
29 // All solutions
30 // x = x0 + k*b/g
31 // y = y0 - k*a/g

```

## 4.9 Miller Habin

```

1 ll mul(ll a, ll b, ll m) {
2     return (a*b-ll(a*(long double)b/m+0.5)*m)%m;
3 }

```

```

3 }
4
5 ll expo(ll a, ll b, ll m) {
6     if (!b) return 1;
7     ll ans = expo(mul(a, a, m), b/2, m);
8     return b%2 ? mul(a, ans, m) : ans;
9 }
10
11 bool prime(ll n) {
12     if (n < 2) return 0;
13     if (n <= 3) return 1;
14     if (n % 2 == 0) return 0;
15
16     ll d = n - 1;
17     int r = 0;
18     while (d % 2 == 0) {
19         r++;
20         d /= 2;
21     }
22
23     // com esses primos, o teste funciona garantido
24     // funciona para n <= 3*10^24 com os primos ate
25     41
26     for (int i : {2, 325, 9375, 28178, 450775,
27         9780504, 795265022}) {
28         if (i >= n) break;
29         ll x = expo(i, d, n);
30         if (x == 1 or x == n - 1) continue;
31
32         bool deu = 1;
33         for (int j = 0; j < r - 1; j++) {
34             x = mul(x, x, n);
35             if (x == n - 1) {
36                 deu = 0;
37                 break;
38             }
39         }
40         if (deu) return 0;
41     }
42     return 1;
43 }

```

## 4.10 Fft Simple

```

1 struct num{
2     ld a {0.0}, b {0.0};
3     num(){
4         num(ld na) : a{na}{}
5         num(ld na, ld nb) : a{na}, b{nb} {}
6         const num operator+(const num &c) const{
7             return num(a + c.a, b + c.b);
8         }
9         const num operator-(const num &c) const{
10            return num(a - c.a, b - c.b);
11        }
12        const num operator*(const num &c) const{
13            return num(a*c.a - b*c.b, a*c.b + b*c.a);
14        }
15        const num operator/(const int &c) const{
16            return num(a/c, b/c);
17        }
18    };
19
20 void fft(vector<num> &a, bool invert){
21     int n = a.size();
22     for(int i=1,j=0;i<n;i++){
23         int bit = n>>1;
24         for(; j&bit; bit>>=1)
25             j^=bit;
26         j^=bit;
27         if(i<j)
28             swap(a[i], a[j]);

```

```

29     }
30     for(int len = 2; len <= n; len <= 1){
31         ld ang = 2 * PI / len * (invert ? -1 : 1);
32         num wlen(cos(ang), sin(ang));
33         for(int i=0;i<n;i+=len){
34             num w(1);
35             for (int j=0;j<len/2;j++){
36                 num u = a[i+j], v = a[i+j+len/2] * w;
37                 a[i+j] = u + v;
38                 a[i+j+len/2] = u - v;
39                 w = w * wlen;
40             }
41         }
42     }
43     if(invert)
44         for(num &x: a)
45             x = x/n;
46
47 }
48
49 vl multiply(vl const& a, vl const& b){
50     vector<num> fa(a.begin(), a.end());
51     vector<num> fb(b.begin(), b.end());
52     int n = 1;
53     while(n < int(a.size() + b.size() )
54         n <= 1;
55     fa.resize(n);
56     fb.resize(n);
57     fft(fa, false);
58     fft(fb, false);
59     for(int i=0;i<n;i++)
60         fa[i] = fa[i]*fb[i];
61     fft(fa, true);
62     vl result(n);
63     for(int i=0;i<n;i++)
64         result[i] = round(fa[i].a);
65     while(result.back()==0) result.pop_back();
66     return result;
67 }

```

## 4.11 Fft Tourist

```

1 struct num{
2     ld x, y;
3     num() { x = y = 0; }
4     num(ld x, ld y) : x(x), y(y) {}
5 };
6
7 inline num operator+(num a, num b) { return num(a.x +
8     b.x, a.y + b.y); }
9 inline num operator-(num a, num b) { return num(a.x -
10    b.x, a.y - b.y); }
11 inline num operator*(num a, num b) { return num(a.x *
12    b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
13 inline num conj(num a) { return num(a.x, -a.y); }
14
15 int base = 1;
16 vector<num> roots = {{0, 0}, {1, 0}};
17 vi rev = {0, 1};
18
19 void ensure_base(int nbase){
20     if(nbase <= base)
21         return;
22
23     rev.resize(1 << nbase);
24     for(int i = 0; i < (1 << nbase); i++)
25         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
26             nbase - 1));
27
28     roots.resize(1 << nbase);
29
30     while(base < nbase){
31         ld angle = 2*PI / (1 << (base + 1));

```



```

28     for(int i = 1 << (base - 1); i < (1 << base); i++){
29         roots[i << 1] = roots[i];
30         ld angle_i = angle * (2 * i + 1 - (1 <<
base));
31         roots[(i << 1) + 1] = num(cos(angle_i),
sin(angle_i));
32     }
33     base++;
34 }
35 }
36
37 void fft(vector<num> &a, int n = -1){
38     if(n == -1)
39         n = a.size();
40
41     assert((n & (n-1)) == 0);
42     int zeros = __builtin_ctz(n);
43     ensure_base(zeros);
44     int shift = base - zeros;
45     for(int i = 0; i < n; i++){
46         if(i < (rev[i] >> shift))
47             swap(a[i], a[rev[i] >> shift]);
48
49     for(int k = 1; k < n; k <= 1)
50         for(int i = 0; i < n; i += 2 * k)
51             for(int j = 0; j < k; j++){
52                 num z = a[i+j+k] * roots[j+k];
53                 a[i+j+k] = a[i+j] - z;
54                 a[i+j] = a[i+j] + z;
55             }
56 }
57
58 vector<num> fa, fb;
59 vi multiply(vi &a, vi &b){
60     int need = a.size() + b.size() - 1;
61     int nbase = 0;
62     while((1 << nbase) < need) nbase++;
63     ensure_base(nbase);
64     int sz = 1 << nbase;
65     if(sz > (int) fa.size())
66         fa.resize(sz);
67
68     for(int i = 0; i < sz; i++){
69         int x = (i < (int) a.size() ? a[i] : 0);
70         int y = (i < (int) b.size() ? b[i] : 0);
71         fa[i] = num(x, y);
72     }
73     fft(fa, sz);
74     num r(0, -0.25 / sz);
75     for(int i = 0; i <= (sz >> 1); i++){
76         int j = (sz - i) & (sz - 1);
77         num z = (fa[j] * fa[j] - conj(fa[i] * fa[i]))
78             * r;
79         if(i != j) {
80             fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa
81 j])) * r;
82             fa[i] = z;
83         }
84         fft(fa, sz);
85         vi res(need);
86         for(int i = 0; i < need; i++)
87             res[i] = fa[i].x + 0.5;
88     }
89 }
90
91
92 vi multiply_mod(vi &a, vi &b, int m, int eq = 0){
93     int need = a.size() + b.size() - 1;
94     int nbase = 0;
95     while((1 << nbase) < need) nbase++;
96
97     ensure_base(nbase);
98     int sz = 1 << nbase;
99     if(sz > (int) fa.size())
100         fa.resize(sz);
101
102     for(int i=0;i<(int)a.size();i++){
103         int x = (a[i] % m + m) % m;
104         fa[i] = num(x & ((1 << 15) - 1), x >> 15);
105     }
106     fill(fa.begin() + a.size(), fa.begin() + sz, num
107 {0, 0});
108     fft(fa, sz);
109     if(sz > (int) fb.size())
110         fb.resize(sz);
111     if(eq)
112         copy(fa.begin(), fa.begin() + sz, fb.begin())
113 ;
114     else{
115         for(int i = 0; i < (int) b.size(); i++){
116             int x = (b[i] % m + m) % m;
117             fb[i] = num(x & ((1 << 15) - 1), x >> 15)
118 ;
119         }
120         fill(fb.begin() + b.size(), fb.begin() + sz,
121 num {0, 0});
122         fft(fb, sz);
123     }
124     ld ratio = 0.25 / sz;
125     num r2(0, -1);
126     num r3(ratio, 0);
127     num r4(0, -ratio);
128     num r5(0, 1);
129     for(int i=0;i<=(sz >> 1);i++) {
130         int j = (sz - i) & (sz - 1);
131         num a1 = (fa[i] + conj(fa[j]));
132         num a2 = (fa[i] - conj(fa[j])) * r2;
133         num b1 = (fb[i] + conj(fb[j])) * r3;
134         num b2 = (fb[i] - conj(fb[j])) * r4;
135         if(i != j){
136             num c1 = (fa[j] + conj(fa[i]));
137             num c2 = (fa[j] - conj(fa[i])) * r2;
138             num d1 = (fb[j] + conj(fb[i])) * r3;
139             num d2 = (fb[j] - conj(fb[i])) * r4;
140             fa[i] = c1 * d1 + c2 * d2 * r5;
141             fb[i] = c1 * d2 + c2 * d1;
142         }
143         fa[j] = a1 * b1 + a2 * b2 * r5;
144         fb[j] = a1 * b2 + a2 * b1;
145     }
146     fft(fa, sz);
147     fft(fb, sz);
148     vi res(need);
149     for(int i=0;i<need;i++){
150         ll aa = fa[i].x + 0.5;
151         ll bb = fb[i].x + 0.5;
152         ll cc = fa[i].y + 0.5;
153         res[i] = (aa + ((bb % m) << 15) + ((cc % m)
154 << 30)) % m;
155     }
156     return res;
157 }
158
159 int main()
160 {sws;
161
162     //FFT
163     vi fx{1, 2, 3}; // 1+2x+3x^2
164     vi gx{4, 5}; // 4+5x
165     vi res;

```

```

164     res = multiply(fx,gx); //4 + 13x + 22x^2 + 15x^3
165
166     return 0;
167
168 }

```

## 4.12 Matrix Exponentiation

```

1 struct Matrix {
2     vector<vl> m;
3     int r, c;
4
5     Matrix(vector<vl> mat) {
6         m = mat;
7         r = mat.size();
8         c = mat[0].size();
9     }
10
11     Matrix(int row, int col, bool ident=false) {
12         r = row; c = col;
13         m = vector<vl>(r, vl(c, 0));
14         if(ident) {
15             for(int i = 0; i < min(r, c); i++) {
16                 m[i][i] = 1;
17             }
18         }
19     }
20
21     Matrix operator*(const Matrix &o) const {
22         assert(c == o.r); // garantir que da pra
multiplicar
23         vector<vl> res(r, vl(o.c, 0));
24
25         for(int i = 0; i < r; i++) {
26             for(int k = 0; k < c; k++) {
27                 for(int j = 0; j < o.c; j++) {
28                     res[i][j] = (res[i][j] + m[i][k]*
29 o.m[k][j]) % MOD;
30                 }
31             }
32
33             return Matrix(res);
34         }
35 };
36
37 Matrix fexp(Matrix b, int e, int n) {
38     if(e == 0) return Matrix(n, n, true); //
identidade
39     Matrix res = fexp(b, e/2, n);
40     res = (res * res);
41     if(e%2) res = (res * b);
42
43     return res;
44 }

```

## 4.13 Mulmod

```

1 ll mulmod(ll a, ll b) {
2     if(a == 0) {
3         return 0LL;
4     }
5     if(a%2 == 0) {
6         ll val = mulmod(a/2, b);
7         return (val + val) % MOD;
8     }
9     else {
10         ll val = mulmod((a-1)/2, b);
11         val = (val + val) % MOD;
12         return (val + b) % MOD;
13     }
14 }

```

## 4.14 Raiz Primitiva

```

1 ll fexp(ll b, ll e, ll mod) {
2     if(e == 0) return 1LL;
3     ll res = fexp(b, e/2LL, mod);
4     res = (res*res)%mod;
5     if(e%2LL)
6         res = (res*b)%mod;
7
8     return res%mod;
9 }
10
11 vl fatorar(ll n) { // fatora em primos
12     vl fat;
13     for(int i = 2; i*i <= n; i++) {
14         if(n%i == 0) {
15             fat.pb(i);
16             while(n%i == 0)
17                 n /= i;
18         }
19     }
20     return fat;
21 }
22
23 // O(log(n) ^ 2)
24 bool raiz_prim(ll a, ll mod, ll phi, vl fat) {
25     if(__gcd(a, mod) != 1 or fexp(a, phi/2, mod) ==
26 1) // phi de euler sempre eh PAR
27         return false;
28
29     for(auto f : fat) {
30         if(fexp(a, phi/f, mod) == 1)
31             return false;
32     }
33     return true;
34 }
35
36 // mods com raizes primitivas: 2, 4, p^k, 2*p^k, p eh
primo impar, k inteiro --- O(n log^2(n))
37 ll achar_raiz(ll mod, ll phi) {
38     if(mod == 2) return 1;
39     vl fat, elementos;
40     fat = fatorar(phi);
41
42     for(ll i = 2; i <= mod-1; i++) {
43         if(raiz_prim(i, mod, phi, fat))
44             return i;
45     }
46
47     return -1; // retorna -1 se nao existe
48 }
49
50 vl todas_raizes(ll mod, ll phi, ll raiz) {
51     vl raizes;
52     if(raiz == -1) return raizes;
53     ll r = raiz;
54     for(ll i = 1; i <= phi-1; i++) {
55         if(__gcd(i, phi) == 1) {
56             raizes.pb(r);
57             r = (r * raiz) % mod;
58         }
59     }
60
61     return raizes;
62 }

```

## 4.15 Bigmod

```

1 ll mod(string a, ll p) {
2     ll res = 0, b = 1;
3     reverse(all(a));

```

```

4
5     for(auto c : a) {
6         ll tmp = (((ll)c-'0')*b) % p;
7         res = (res + tmp) % p;
8
9         b = (b * 10) % p;
10    }
11
12    return res;
13 }

```

## 4.16 Berlekamp Massey

```

1
2 #define SZ 233333
3
4 ll qp(ll a,ll b)
5 {
6     ll x=1; a%=MOD;
7     while(b)
8     {
9         if(b&1) x=x*a%MOD;
10        a=a*a%MOD; b>>=1;
11    }
12    return x;
13 }
14 namespace linear_seq {
15
16 inline vector<int> BM(vector<int> x)
17 {
18     //ls: (shortest) relation sequence (after filling
19     zeroes) so far
20     //cur: current relation sequence
21     vector<int> ls,cur;
22     //lf: the position of ls (t')
23     //ldt: delta of ls (v')
24     int lf=0,ldt=0;
25     for(int i=0;i<int(x.size());++i)
26     {
27         ll t=0;
28         //evaluate at position i
29         for(int j=0;j<int(cur.size());++j)
30             t=(t+x[i-j-1]*(ll)cur[j])%MOD;
31         if((t-x[i])%MOD==0) continue; //good so far
32         //first non-zero position
33         if(!cur.size())
34         {
35             cur.resize(i+1);
36             lf=i; ldt=(t-x[i])%MOD;
37             continue;
38         }
39         //cur=cur-c/ldt*(x[i]-t)
40         ll k=-x[i]-t)*qp(ldt,MOD-2)%MOD/*1/ldt*/;
41         vector<int> c(i-lf-1); //add zeroes in front
42         c.pb(k);
43         for(int j=0;j<int(ls.size());++j)
44             c.pb(-ls[j]*k%MOD);
45         if(c.size()<cur.size()) c.resize(cur.size());
46         for(int j=0;j<int(cur.size());++j)
47             c[j]=(c[j]+cur[j])%MOD;
48         //if cur is better than ls, change ls to cur
49         if(i-lf+(int)ls.size()>=(int)cur.size())
50             ls=cur,lf=i,ldt=(t-x[i])%MOD;
51         cur=c;
52     }
53     for(int i=0;i<int(cur.size());++i)
54         cur[i]=(cur[i]%MOD+MOD)%MOD;
55     return cur;
56 }
57 int m; //length of recurrence
58 //a: first terms
59 //h: relation
60 ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];

```

```

60 //calculate p*q mod f
61 inline void mull(ll*p,ll*q)
62 {
63     for(int i=0;i<m+m;++i) t_[i]=0;
64     for(int i=0;i<m;++i) if(p[i])
65         for(int j=0;j<m;++j)
66             t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
67     for(int i=m+m-1;i>=m;--i) if(t_[i])
68         //miuns t_[i]x^{i-m}(x^{m-\sum_{j=0}^{m-1}} x^{
69         m-j-1}h_j)
70         for(int j=m-1;~j;--j)
71             t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
72     for(int i=0;i<m;++i) p[i]=t_[i];
73 }
74 inline ll calc(ll K)
75 {
76     for(int i=m;~i;--i)
77         s[i]=t[i]=0;
78     //init
79     s[0]=1; if(m!=1) t[1]=1; else t[0]=h[0];
80     //binary-exponentiation
81     while(K)
82     {
83         if(K&1) mull(s,t);
84         mull(t,t); K>>=1;
85     }
86     ll su=0;
87     for(int i=0;i<m;++i) su=(su+s[i]*a[i])%MOD;
88     return (su%MOD+MOD)%MOD;
89 }
90 inline int work(vector<int> x,ll n)
91 {
92     if(n<int(x.size())) return x[n];
93     vector<int> v=BM(x); m=v.size(); if(!m) return 0;
94     for(int i=0;i<m;++i) h[i]=v[i],a[i]=x[i];
95     return calc(n);
96 }
97 }
98 using linear_seq::work;

```

## 4.17 Double Gcd

```

1 ld gcdf(ld a, ld b){
2     if(a<b) return gcdf(b, a);
3
4     if(fabs(b)<EPS)
5         return a;
6     else
7         return (gcdf(b, a - floor(a/b)*b));
8 }

```

## 4.18 Totient

```

1 // phi(p^k) = (p^{k-1})*(p-1) com p primo
2 // 0(sqrt(m))
3 ll phi(ll m){
4     ll res = m;
5     for(ll d=2;d*d<=m;d++){
6         if(m % d == 0){
7             res = (res/d)*(d-1);
8             while(m%d == 0)
9                 m /= d;
10        }
11    }
12    if(m > 1) {
13        res /= m;
14        res *= (m-1);
15    }
16    return res;
17 }
18

```

```

19 // modificacao do crivo, O(n*log(log(n)))
20 vl phi_to_n(ll n){
21     vector<bool> isprime(n+1, true);
22     vl tot(n+1);
23     tot[0] = 0; tot[1] = 1;
24     for(ll i=1;i<=n; i++){
25         tot[i] = i;
26     }
27
28     for(ll p=2;p<=n;p++){
29         if(isprime[p]){
30             tot[p] = p-1;
31             for(ll i=p+1;i<=n;i+=p){
32                 isprime[i] = false;
33                 tot[i] = (tot[i]/p)*(p-1);
34             }
35         }
36     }
37     return tot;
38 }

```

## 4.19 Kitamasa

```

1 using poly = vector<mint>; // mint = int mod P with
  operators +, - and *
2 inline int len(const poly& a) { return a.size(); } //
  get rid of the annoying "hey a.size() is
  unsigned" warning
3
4 poly pmul(const poly& a, const poly& b) {
5     poly c(len(a) + len(b) - 1, 0);
6     for (int i = 0; i < len(a); i++)
7         for (int j = 0; j < len(b); j++)
8             c[i+j] = c[i+j] + a[i] * b[j];
9     return c;
10 }
11
12 // only works if b.back() == 1
13 poly pmod(const poly& a, const poly& b) {
14     poly c(a.begin(), a.end());
15     for (int i = len(c) - 1; i >= len(b) - 1; i--) {
16         int k = i - (len(b) - 1); // index of the
  quotient term
17         for (int j = 0; j < len(b); j++)
18             c[j+k] = c[j+k] - c[i] * b[j];
19     }
20     c.resize(len(b) - 1);
21     return c;
22 }
23
24 poly ppwr(poly x, ll e, poly f) {
25     poly ans = { 1 };
26     for (; e > 0; e /= 2) {
27         if (e & 1) ans = pmod(pmul(ans, x), f);
28         x = pmod(pmul(x, x), f);
29     }
30     return ans;
31 }
32
33 // values = { A0, A1, ..., An }. recurrence = C0 * A0
  + C1 * A1 + ... + Cn * An generates A{n+1}
34 mint kitamasa(const poly& values, const poly&
  recurrence, ll n) {
35     poly f(len(recurrence) + 1);
36     f.back() = 1;
37     for (int i = 0; i < len(recurrence); i++)
38         f[i] = mint(0) - recurrence[i];
39
40     auto d = ppwr(poly{0, 1}, n, f); // x^N mod f(x)
41
42     mint ans = 0;
43     for (int i = 0; i < len(values); i++)
44         ans = ans + d[i] * values[i];

```

```

45     return ans;
46 }

```

## 4.20 Mobius

```

1 vi mobius(int n) {
2     // g(n) = sum{f(d)} => f(n) = sum{mu(d)*g(n/d)}
3     vi mu(n+1);
4     mu[1] = 1; mu[0] = 0;
5     for(int i = 1; i <= n; i++)
6         for(int j = i + i; j <= n; j += i)
7             mu[j] -= mu[i];
8
9     return mu;
10 }

```

## 5 Grafos

### 5.1 Ford

```

1 const int N = 2000010;
2
3 struct Ford {
4     struct Edge {
5         int to, f, c;
6     };
7
8     int vis[N];
9     vector<int> adj[N];
10    vector<Edge> edges;
11    int cur = 0;
12
13    void addEdge(int a, int b, int cap, int rcap) {
14        Edge e;
15        e.to = b; e.c = cap; e.f = 0;
16        edges.pb(e);
17        adj[a].pb(cur++);
18
19        e = Edge();
20        e.to = a; e.c = rcap; e.f = 0;
21        edges.pb(e);
22        adj[b].pb(cur++);
23    }
24
25    int dfs(int s, int t, int f, int tempo) {
26        if(s == t)
27            return f;
28        vis[s] = tempo;
29
30        for(int e : adj[s]) {
31            if(vis[edges[e].to] < tempo and (edges[e]
  ].c - edges[e].f) > 0) {
32                if(int a = dfs(edges[e].to, t, min(f,
  edges[e].c-edges[e].f), tempo)) {
33                    edges[e].f += a;
34                    edges[e^1].f -= a;
35                    return a;
36                }
37            }
38        }
39        return 0;
40    }
41
42    int flow(int s, int t) {
43        int mflow = 0, tempo = 1;
44        while(int a = dfs(s, t, INF, tempo)) {
45            mflow += a;
46            tempo++;
47        }
48        return mflow;
49    }
50 };

```

## 5.2 2sat

```
1 vector<int> g[MAX], gt[MAX], S; int vis[MAX], cor[MAX];
2
3 int val(int n, bool tvalue) {
4     if(tvalue) return 2*n;
5     return 2*n + 1;
6 }
7
8 void dfs(int u) {
9     vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
10    S.push_back(u);
11 }
12
13 void dfst(int u, int e) {
14     cor[u] = e;
15     for(int v : gt[u]) if(!cor[v]) dfst(v, e);
16 }
17
18 void kosaraju(int n) {
19     for(int i = 0; i <= n; i++) if(!vis[i]) dfs(i);
20     for(int i = 0; i <= n; i++) for(int j : g[i])
21         gt[j].push_back(i);
22     int e = 0; reverse(S.begin(), S.end());
23     for(int u : S) if(!cor[u]) dfst(u, ++e);
24 }
25
26 // antes de chamar essa funcao, colocar as arestas do
27 // grafo
28 bool solve(int n, vi &res) {
29     kosaraju(2*n); // MAX > 2*N
30     vi r;
31     forn(i, n) {
32         int t = val(i, true), f = val(i, false);
33         if(cor[t] == cor[f]) {
34             return false;
35         }
36         else {
37             if(cor[t] > cor[f])
38                 r.pb(1);
39             else
40                 r.pb(0);
41         }
42     }
43     swap(r, res);
44     return true;
45 }
```

## 5.3 Kahn

```
1 vi g[MAX];
2 int in[MAX], cor[MAX];
3 void kahn(int n) {
4     int label = 1;
5     priority_queue<int, vector<int>, greater<int>> pq
6     ; // trocar por queue para O(n)
7     for(int i = 1; i <= n; i++) {
8         if(in[i] == 0) {
9             pq.push(i);
10        }
11    }
12    while(pq.size()) {
13        int u = pq.top(); pq.pop();
14        cor[u] = label++;
15        for(auto prox : g[u]) {
16            in[prox]--;
17            if(in[prox] == 0) {
18                pq.push(prox);
19            }
20        }
21    }
```

```
20    }
21 }
22 }
```

## 5.4 Hungarian

```
1 template<typename T> struct hungarian {
2     int n, m;
3     vector<vector<T>> a;
4     vector<T> u, v;
5     vector<int> p, way;
6     T inf;
7
8     hungarian(int n_, int m_) : n(n_), m(m_), u(m+1),
9         v(m+1), p(m+1), way(m+1) {
10        a = vector<vector<T>>(n, vector<T>(m));
11        inf = numeric_limits<T>::max();
12    }
13    pair<T, vector<int>> assignment() {
14        for (int i = 1; i <= n; i++) {
15            p[0] = i;
16            int j0 = 0;
17            vector<T> minv(m+1, inf);
18            vector<int> used(m+1, 0);
19            do {
20                used[j0] = true;
21                int i0 = p[j0], j1 = -1;
22                T delta = inf;
23                for (int j = 1; j <= m; j++) if (!
24                    used[j]) {
25                    T cur = a[i0-1][j-1] - u[i0] - v[
26                        j];
27                    if (cur < minv[j]) minv[j] = cur,
28                        way[j] = j0;
29                    if (minv[j] < delta) delta = minv
30                        [j], j1 = j;
31                }
32                for (int j = 0; j <= m; j++)
33                    if (used[j]) u[p[j]] += delta, v[
34                        j] -= delta;
35                else minv[j] -= delta;
36                j0 = j1;
37            } while (p[j0] != 0);
38            do {
39                int j1 = way[j0];
40                p[j0] = p[j1];
41                j0 = j1;
42            } while (j0);
43        }
44        vector<int> ans(m);
45        for (int j = 1; j <= n; j++) ans[p[j]-1] = j
46            - 1;
47        return make_pair(-v[0], ans);
48    }
49 };
50 }
```

## 5.5 Dfs Tree

```
1 int desce[MAX], sobe[MAX], vis[MAX], h[MAX];
2 int backedges[MAX], pai[MAX];
3
4 // backedges[u] = backedges que comecam embaixo de (
5 // ou =) u e sobem pra cima de u; backedges[u] == 0
6 // => u eh ponte
7 void dfs(int u, int p) {
8     if(vis[u]) return;
9     pai[u] = p;
10    h[u] = h[p]+1;
11    vis[u] = 1;
12    for(auto v : g[u]) {
13        if(p == v or vis[v]) continue;
14    }
```

```

13     dfs(v, u);
14     backedges[u] += backedges[v];
15 }
16 for(auto v : g[u]) {
17     if(h[v] > h[u]+1)
18         desce[u]++;
19     else if(h[v] < h[u]-1)
20         sobe[u]++;
21 }
22 backedges[u] += sobe[u] - desce[u];
23 }

```

## 5.6 Lca

```

1 template<typename T> struct rmq {
2     vector<T> v;
3     int n; static const int b = 30;
4     vector<int> mask, t;
5
6     int op(int x, int y) { return v[x] < v[y] ? x : y; }
7     int msb(int x) { return __builtin_clz(1) -
8         __builtin_clz(x); }
9     rmq(const vector<T>& v_) : v(v_), n(v.size()),
10        mask(n), t(n) {
11         for (int i = 0, at = 0; i < n; mask[i++] = at
12             |= 1) {
13             at = (at<<1)&((1<<b)-1);
14             while (at and op(i, i-msb(at&-at)) == i)
15                 at ^= at&-at;
16             for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-
17                 msb(mask[b*i+b-1]);
18             for (int j = 1; (1<<j) <= n/b; j++) for (int
19                 i = 0; i+(1<<j) <= n/b; i++)
20                 t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j
21                     -1)+i+(1<<(j-1))]);
22             int small(int r, int sz = b) { return r-msb(mask[
23                 r]&((1<<sz)-1)); }
24             T query(int l, int r) {
25                 if (r-l+1 <= b) return small(r, r-l+1);
26                 int ans = op(small(l+b-1), small(r));
27                 int x = l/b+1, y = r/b-1;
28                 if (x <= y) {
29                     int j = msb(y-x+1);
30                     ans = op(ans, op(t[n/b*j+x], t[n/b*j+y
31                         -(1<<j)+1]));
32                 }
33                 return ans;
34             }
35         }
36     };
37 };
38 namespace lca {
39     vector<int> g[MAX];
40     int v[2*MAX], pos[MAX], dep[2*MAX];
41     int t;
42     rmq<int> RMQ;
43
44     void dfs(int i, int d = 0, int p = -1) {
45         v[t] = i, pos[i] = t, dep[t++] = d;
46         for (int j : g[i]) if (j != p) {
47             dfs(j, d+1, i);
48             v[t] = i, dep[t++] = d;
49         }
50     }
51     void build(int n, int root) {
52         t = 0;
53         dfs(root);
54         RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
55     }
56     int lca(int a, int b) {

```

```

50         a = pos[a], b = pos[b];
51         return v[RMQ.query(min(a, b), max(a, b))];
52     }
53     int dist(int a, int b) {
54         return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[
55             lca(a, b)]];
56     }

```

## 5.7 Hld Aresta

```

1 vector<vector<pair<int, int>>> g(MAX, vector<pair<int
2     ,int>>());
3 vi in(MAX), inv(MAX), sz(MAX);
4 vi peso(MAX), pai(MAX);
5 vi head(MAX), tail(MAX), h(MAX);
6 int tin;
7
8 void dfs(int u, int p=-1, int depth=0){
9     sz[u] = 1; h[u] = depth;
10    for(auto &i: g[u]) if(i.ff != p){
11        auto [v, w] = i;
12        dfs(v, u, depth+1);
13        pai[v] = u; sz[u] += sz[v]; peso[v] = w;
14        if (sz[v] > sz[g[u][0].ff) or g[u][0].ff == p
15            swap(i, g[u][0]);
16    }
17 void build_hld(int u, int p = -1) {
18     v[in[u] = tin++] = peso[u]; tail[u] = u;
19     inv[tin-1] = u;
20     for(auto &i: g[u]) if(i.ff != p) {
21         int v = i.ff;
22         head[v] = (i == g[u][0] ? head[u] : v);
23         build_hld(v, u);
24     }
25     if(g[u].size() > 1) tail[u] = tail[g[u][0].ff];
26 }
27 void init_hld(int root = 0) {
28     dfs(root);
29     tin = 0;
30     build_hld(root);
31     build();
32 }
33 void reset(){
34     g.assign(MAX, vector<pair<int,int>>());
35     in.assign(MAX, 0), sz.assign(MAX, 0);
36     peso.assign(MAX, 0), pai.assign(MAX, 0);
37     head.assign(MAX, 0); tail.assign(MAX, 0);
38     h.assign(MAX, 0); inv.assign(MAX, 0);
39
40     t.assign(4*MAX, 0); v.assign(MAX, 0);
41     lazy.assign(4*MAX, 0);
42 }
43 ll query_path(int a, int b) {
44     if (a == b) return 0;
45     if(in[a] < in[b]) swap(a, b);
46
47     if(head[a] == head[b]) return query(in[b]+1, in[a
48     ]);
49     return merge(query(in[head[a]], in[a]),
50         query_path(pai[head[a]], b));
51 }
52 void update_path(int a, int b, int x) {
53     if (a == b) return;
54     if(in[a] < in[b]) swap(a, b);
55
56     if(head[a] == head[b]) return (void)update(in[b
57     ]+1, in[a], x);
58     update(in[head[a]], in[a], x); update_path(pai[
59     head[a]], b, x);

```

```

57 ll query_subtree(int a) {
58     if(sz[a] == 1) return 0;
59     return query(in[a]+1, in[a]+sz[a]-1);
60 }
61 void update_subtree(int a, int x) {
62     if(sz[a] == 1) return;
63     update(in[a]+1, in[a]+sz[a]-1, x);
64 }
65 int lca(int a, int b) {
66     if(in[a] < in[b]) swap(a, b);
67     return head[a] == head[b] ? b : lca(pai[head[a]],
68     b);
69 }

```

## 5.8 MCMF

```

1  template <class T = int>
2  class MCMF {
3  public:
4      struct Edge {
5          Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
6          int to;
7          T cap, cost;
8      };
9
10     MCMF(int size) {
11         n = size;
12         edges.resize(n);
13         pot.assign(n, 0);
14         dist.resize(n);
15         visit.assign(n, false);
16     }
17
18     std::pair<T, T> mcmf(int src, int sink) {
19         std::pair<T, T> ans(0, 0);
20         if(!SPFA(src, sink)) return ans;
21         fixPot();
22         // can use dijkstra to speed up depending on
23         the graph
24         while(SPFA(src, sink)) {
25             auto flow = augment(src, sink);
26             ans.first += flow.first;
27             ans.second += flow.first * flow.second;
28             fixPot();
29         }
30         return ans;
31     }
32
33     void addEdge(int from, int to, T cap, T cost) {
34         edges[from].push_back(list.size());
35         list.push_back(Edge(to, cap, cost));
36         edges[to].push_back(list.size());
37         list.push_back(Edge(from, 0, -cost));
38     }
39 private:
40     int n;
41     std::vector<std::vector<int>>> edges;
42     std::vector<Edge> list;
43     std::vector<int> from;
44     std::vector<T> dist, pot;
45     std::vector<bool> visit;
46
47     /*bool dij(int src, int sink) {
48         T INF = std::numeric_limits<T>::max();
49         dist.assign(n, INF);
50         from.assign(n, -1);
51         visit.assign(n, false);
52         dist[src] = 0;
53         for(int i = 0; i < n; i++) {
54             int best = -1;
55             for(int j = 0; j < n; j++) {

```

```

56         if(best == -1 || dist[best] > dist[j]) best = j;
57     }
58     if(dist[best] >= INF) break;
59     visit[best] = true;
60     for(auto e : edges[best]) {
61         auto ed = list[e];
62         if(ed.cap == 0) continue;
63         T toDist = dist[best] + ed.cost + pot
64         [best] - pot[ed.to];
65         assert(toDist >= dist[best]);
66         if(toDist < dist[ed.to]) {
67             dist[ed.to] = toDist;
68             from[ed.to] = e;
69         }
70     }
71     return dist[sink] < INF;
72 }*/
73
74 std::pair<T, T> augment(int src, int sink) {
75     std::pair<T, T> flow = {list[from[sink]].cap,
76     0};
77     for(int v = sink; v != src; v = list[from[v]
78     ^1].to) {
79         flow.first = std::min(flow.first, list[
80         from[v]].cap);
81         flow.second += list[from[v]].cost;
82     }
83     for(int v = sink; v != src; v = list[from[v]
84     ^1].to) {
85         list[from[v]].cap -= flow.first;
86         list[from[v]^1].cap += flow.first;
87     }
88     return flow;
89 }
90
91 std::queue<int> q;
92 bool SPFA(int src, int sink) {
93     T INF = std::numeric_limits<T>::max();
94     dist.assign(n, INF);
95     from.assign(n, -1);
96     q.push(src);
97     dist[src] = 0;
98     while(!q.empty()) {
99         int on = q.front();
100         q.pop();
101         visit[on] = false;
102         for(auto e : edges[on]) {
103             auto ed = list[e];
104             if(ed.cap == 0) continue;
105             T toDist = dist[on] + ed.cost + pot[
106             on] - pot[ed.to];
107             if(toDist < dist[ed.to]) {
108                 dist[ed.to] = toDist;
109                 from[ed.to] = e;
110                 if(!visit[ed.to]) {
111                     visit[ed.to] = true;
112                     q.push(ed.to);
113                 }
114             }
115         }
116     }
117     return dist[sink] < INF;
118 }
119
120 void fixPot() {
121     T INF = std::numeric_limits<T>::max();
122     for(int i = 0; i < n; i++) {
123         if(dist[i] < INF) pot[i] += dist[i];
124     }
125 }

```

## 5.9 Centroid

```
1 int sz[MAX];
2 bool erased[MAX];
3 vi grafo[MAX];
4
5 void dfs(int u, int p=-1){
6     sz[u] = 1;
7     for(int v: grafo[u]) if(v!=p and !erased[v]){
8         dfs(v, u);
9         sz[u] += sz[v];
10    }
11 }
12
13 int centroid(int u, int p=-1, int size=-1){
14     if(size==-1) size = sz[u];
15     for(int v: grafo[u])
16         if(v!=p and !erased[v] and sz[v]>size/2)
17             return centroid(v, u, size);
18     return u;
19 }
20
21 pii centroids(int u=1){ // idx 1
22     dfs(u);
23     int c1=centroid(u), c2=c1;
24     for(int v: grafo[c1]) if(2*sz[v]==sz[u]) c2=v;
25     return {c1, c2};
26 }
```

## 5.10 Kosaraju

```
1 int n;
2 vi g[MAX], gi[MAX]; // grafo invertido
3 int vis[MAX], comp[MAX]; // componente conexo de cada
   vertice
4 stack<int> S;
5
6 void dfs(int u){
7     vis[u] = 1;
8     for(auto v: g[u]) if(!vis[v]) dfs(v);
9     S.push(u);
10 }
11
12 void scc(int u, int c){
13     vis[u] = 1; comp[u] = c;
14     for(auto v: gi[u]) if(!vis[v]) scc(v, c);
15 }
16
17 void kosaraju(){
18     for(int i=0;i<n;i++) vis[i] = 0;
19     for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
20     for(int i=0;i<n;i++) vis[i] = 0;
21     while(S.size()){
22         int u = S.top();
23         S.pop();
24         if(!vis[u]) scc(u, u);
25     }
26 }
```

## 5.11 Dinic

```
1 const int N = 300;
2
3 struct Dinic {
4     struct Edge{
5         int from, to; ll flow, cap;
6     };
7     vector<Edge> edge;
8
9     vector<int> g[N];
10    int ne = 0;
11    int lvl[N], vis[N], pass;
```

```
12    int qu[N], px[N], qt;
13
14    ll run(int s, int sink, ll minE) {
15        if(s == sink) return minE;
16
17        ll ans = 0;
18
19        for(; px[s] < (int)g[s].size(); px[s]++) {
20            int e = g[s][ px[s] ];
21            auto &v = edge[e], &rev = edge[e^1];
22            if(lvl[v.to] != lvl[s]+1 || v.flow >= v.
23                cap) continue; // v.cap - v.flow
24            < lim
25            ll tmp = run(v.to, sink, min(minE, v.cap-v
26                .flow));
27            v.flow += tmp, rev.flow -= tmp;
28            ans += tmp, minE -= tmp;
29            if(minE == 0) break;
30        }
31        return ans;
32    }
33
34    bool bfs(int source, int sink) {
35        qt = 0;
36        qu[qt++] = source;
37        lvl[source] = 1;
38        vis[source] = ++pass;
39        for(int i = 0; i < qt; i++) {
40            int u = qu[i];
41            px[u] = 0;
42            if(u == sink) return true;
43            for(auto& ed : g[u]) {
44                auto v = edge[ed];
45                if(v.flow >= v.cap || vis[v.to] ==
46                    pass) continue; // v.cap - v.flow < lim
47                vis[v.to] = pass;
48                lvl[v.to] = lvl[u]+1;
49                qu[qt++] = v.to;
50            }
51        }
52        return false;
53    }
54
55    ll flow(int source, int sink) {
56        reset_flow();
57        ll ans = 0;
58        //for(lim = (1LL << 62); lim >= 1; lim /= 2)
59        while(bfs(source, sink))
60            ans += run(source, sink, LLINF);
61        return ans;
62    }
63
64    void addEdge(int u, int v, ll c, ll rc) {
65        Edge e = {u, v, 0, c};
66        edge.pb(e);
67        g[u].push_back(ne++);
68
69        e = {v, u, 0, rc};
70        edge.pb(e);
71        g[v].push_back(ne++);
72    }
73
74    void reset_flow() {
75        for(int i = 0; i < ne; i++)
76            edge[i].flow = 0;
77        memset(lvl, 0, sizeof(lvl));
78        memset(vis, 0, sizeof(vis));
79        memset(qu, 0, sizeof(qu));
80        memset(px, 0, sizeof(px));
81        qt = 0; pass = 0;
82    }
83 }
```

## 5.12 Hld Vertice



```

1 // Use it together with recursive_segtree
2 vector<vi> g(MAX, vi());
3 vi in(MAX), inv(MAX), sz(MAX);
4 vi peso(MAX), pai(MAX);
5 vi head(MAX), tail(MAX), h(MAX);
6
7 int tin;
8
9 void dfs(int u, int p=-1, int depth=0){
10     sz[u] = 1; h[u] = depth;
11     for(auto &v: g[u]) if(v != p){
12         dfs(v, u, depth+1);
13         pai[v] = u; sz[u] += sz[v];
14         if (sz[v] > sz[g[u][0]] or g[u][0] == p) swap
            (v, g[u][0]);
15     }
16 }
17 void build_hld(int u, int p = -1) {
18     v[in[u] = tin++] = peso[u]; tail[u] = u;
19     inv[tin-1] = u;
20     for(auto &v: g[u]) if(v != p) {
21         head[v] = (v == g[u][0] ? head[u] : v);
22         build_hld(v, u);
23     }
24     if(g[u].size() > 1) tail[u] = tail[g[u][0]];
25 }
26 void init_hld(int root = 0) {
27     dfs(root);
28     tin = 0;
29     build_hld(root);
30     build();
31 }
32 void reset(){
33     g.assign(MAX, vi());
34     in.assign(MAX, 0), sz.assign(MAX, 0);
35     peso.assign(MAX, 0), pai.assign(MAX, 0);
36     head.assign(MAX, 0); tail.assign(MAX, 0);
37     h.assign(MAX, 0); inv.assign(MAX, 0);
38
39     t.assign(4*MAX, 0); v.assign(MAX, 0);
40     lazy.assign(4*MAX, 0);
41 }
42 ll query_path(int a, int b) {
43     if(in[a] < in[b]) swap(a, b);
44
45     if(head[a] == head[b]) return query(in[b], in[a]);
46     return merge(query(in[head[a]], in[a]),
47         query_path(pai[head[a]], b));
48 }
49 void update_path(int a, int b, int x) {
50     if(in[a] < in[b]) swap(a, b);
51
52     if(head[a] == head[b]) return (void)update(in[b],
53         in[a], x);
54     update(in[head[a]], in[a], x); update_path(pai[
55         head[a]], b, x);
56 }
57 void update_subtree(int a, int x) {
58     update(in[a], in[a]+sz[a]-1, x);
59 }
60 int lca(int a, int b) {
61     if(in[a] < in[b]) swap(a, b);
62     return head[a] == head[b] ? b : lca(pai[head[a]],
63         b);
64 }

```

## 6 Numeric

### 6.1 Lagrange Interpolation

```

1 // Lagrange's interpolation O(n^2)
2 ld interpolate(vii d, ld x){
3     ld y = 0;
4     int n = d.size();
5     for(int i=0;i<n;i++){
6         ld yi = d[i].ss;
7         for(int j=0;j<n;j++){
8             if(j!=i)
9                 yi = yi*(x - d[j].ff)/(ld)(d[i].ff - d
10                     [j].ff);
11         }
12         y += yi;
13     }
14     return y;
15 }
16 // O(n)
17
18 template<typename T = mint>
19 struct Lagrange {
20     vector<T> y, den, l, r;
21     int n;
22     Lagrange(const vector<T>& _y) : y(_y), n(_y.size
23         ()) {
24         den.resize(n, 0);
25         l.resize(n, 0); r.resize(n, 0);
26
27         for (int i = 0; i < n; i++) {
28             den[i] = ifac[i] * ifac[n - 1 - i];
29             if ((n - 1 - i) % 2 == 1) den[i] = -den[i]
30         };
31     }
32
33     T eval(T x) {
34         l[0] = 1;
35         for (int i = 1; i < n; i++)
36             l[i] = l[i-1] * (x + -T(i-1));
37
38         r[n - 1] = 1;
39         for (int i = n - 2; i >= 0; i--)
40             r[i] = r[i+1] * (x + -T(i+1));
41
42         T ans = 0;
43         for (int i = 0; i < n; i++) {
44             T num = l[i] * r[i];
45             ans = ans + y[i] * num * den[i];
46         }
47         return ans;
48     }
49 };

```

### 6.2 Newton Raphson

```

1 // Newton Raphson
2
3 ld f(x){ return x*2 + 2; }
4 ld fd(x){ return 2; } // derivada
5
6 ld root(ld x){
7     // while(f(x)>EPS)
8     for(int i=0;i<20;i++){
9         if(fd(x)<EPS)
10             x = LLINF;
11         else
12             x = x - f(x)/fd(x);
13     }
14     return x;
15 }

```

```
15 }
```

### 6.3 Simpson's Formula

```
1 inline ld simpson(ld fl, ld fr, ld fmid, ld l, ld r){
2     return (fl+fr+4*fmid)*(r-l)/6;
3 }
4
5 ld rsimpson(ld slr, ld fl, ld fr, ld fmid, ld l, ld r
6 )
7 {
8     ld mid = (l+r)/2;
9     ld fml = f((l+mid)/2), fmr = f((mid+r)/2);
10    ld slm = simpson(fl,fmid,fml,l,mid);
11    ld smr = simpson(fmid,fr,fmr,mid,r);
12    if(fabs(slr-slm-smr) < EPS) return slm+smr; //
    aprox. good enough
13    return rsimpson(slm,fl,fmid,fml,l,mid)+rsimpson(
14        smr,fmid,fr,fmr,mid,r);
15 }
16
17 ld integrate(ld l, ld r)
18 {
19     ld mid = (l+r)/2;
20     ld fl = f(l), fr = f(r);
21     ld fmid = f(mid);
22     return rsimpson(simpson(fl,fr,fmid,l,r),fl,fr,
23         fmid,l,r);
24 }
```

## 7 DP

### 7.1 Largest Ksubmatrix

```
1 int n, m;
2 int a[MAX][MAX];
3 // Largest K such that exists a block K*K with equal
4 // numbers
5 int largestKSubmatrix(){
6     int dp[n][m];
7     memset(dp, 0, sizeof(dp));
8
9     int result = 0;
10    for(int i = 0 ; i < n ; i++){
11        for(int j = 0 ; j < m ; j++){
12            if(!i or !j)
13                dp[i][j] = 1;
14            else if(a[i][j] == a[i-1][j] and
15                a[i][j] == a[i][j-1] and
16                a[i][j] == a[i-1][j-1])
17                dp[i][j] = min(min(dp[i-1][j], dp[i][j-1]),
18                    dp[i-1][j-1]) + 1;
19            else dp[i][j] = 1;
20
21            result = max(result, dp[i][j]);
22        }
23    }
24    return result;
25 }
```

### 7.2 Dp Digitos

```
1 // dp de quantidade de numeros <= r com ate qt
2 // digitos diferentes de 0
3 ll dp(int idx, string& r, bool menor, int qt, vector<
4 vector<vi>>& tab) {
5     if(qt > 3) return 0;
6     if(idx >= r.size()) {
7         return 1;
8     }
```

```
9 }
10 if(tab[idx][menor][qt] != -1)
11     return tab[idx][menor][qt];
12
13 ll res = 0;
14 for(int i = 0; i <= 9; i++) {
15     if(menor or i <= r[idx]-'0') {
16         res += dp(idx+1, r, menor or i < (r[idx]-
17             '0'), qt+(i>0), tab);
18     }
19 }
20
21 return tab[idx][menor][qt] = res;
22 }
```

### 7.3 Partition Problem

```
1 // Partition Problem DP O(n2)
2 bool findPartition(vi &arr){
3     int sum = 0;
4     int n = arr.size();
5
6     for(int i=0;i<n;i++)
7         sum += arr[i];
8
9     if(sum&1) return false;
10
11    bool part[sum/2+1][n+1];
12
13    for(int i=0;i<=n;i++){
14        part[0][i] = true;
15    }
16
17    for(int i=1;i<=sum/2;i++){
18        part[i][0] = false;
19
20        for(int j=1;j<=n;j++){
21            part[i][j] = part[i][j-1];
22            if(i >= arr[j-1])
23                part[i][j] |= part[i - arr[j-1]][j-1];
24        }
25    }
26    return part[sum / 2][n];
27 }
```

### 7.4 Aliens

```
1 // Solves https://codeforces.com/contest/1279/problem
2 // F
3 // dado um vetor de inteiros, escolha k subsegmentos
4 // disjuntos de soma máxima
5 // em vez de rodar a dp[i][k] = melhor soma até i
6 // usando k segmentos,
7 // vc roda uma dp[i] adicionando um custo W toda vez
8 // que usa um novo subsegmento,
9 // e faz busca binária nesse W pra achar o custo
10 // mínimo que usa exatamente K intervalos
11
12 ll n, k, L;
13 pll check(ll w, vl& v){
14     vector<pll> dp(n+1);
15     dp[0] = {0,0};
16     for(int i=1;i<=n;i++){
17         dp[i] = dp[i-1];
18         dp[i].ff += v[i];
19         if(i-L>=0){
20             pll t = {dp[i-L].ff + w, dp[i-L].ss + 1};
21             dp[i] = min(dp[i], t);
22         }
23     }
24 }
```

```

20
21     return dp[n];
22 }
23
24 ll solve(vl v){
25     ll l=-1, r=n+1, ans=-1;
26     while(l<=r){
27         ll mid = (l+r)/2;
28         pll c = check(mid, v);
29         if(c.ss <= k){
30             r = mid - 1;
31             ans = mid;
32         }else{
33             l = mid + 1;
34         }
35     }
36
37     pll c = check(ans, v);
38
39     if(ans < 0) return 0;
40
41     // we can simply use k insted of c.ss ~magic~
42     return c.ff - ans*k;
43 }
44
45 int32_t main()
46 {sws;
47
48     string s;
49     cin >> n >> k >> L;
50     cin >> s;
51
52     vl upper(n+1, 0), lower(n+1, 0);
53     for(int i=0;i<n;i++){
54         if('A'<= s[i] and s[i] <= 'Z')
55             upper[i+1] = 1;
56     }
57     for(int i=0;i<n;i++){
58         if('a'<= s[i] and s[i] <= 'z')
59             lower[i+1] = 1;
60
61     cout << min(solve(lower),
62                 solve(upper)) << endl;
63
64     return 0;
65 }

```

## 7.5 Lis

```

1 multiset<int> S;
2 for(int i=0;i<n;i++){
3     auto it = S.upper_bound(vet[i]); // lowfor inc
4     if(it != S.end())
5         S.erase(it);
6     S.insert(vet[i]);
7 }
8 // size of the lis
9 int ans = S.size();
10
11 // see that later
12 // https://codeforces.com/blog/entry/13225?comment=180208
13
14 vi LIS(const vi &elements){
15     auto compare = [&](int x, int y) {
16         return elements[x] < elements[y];
17     };
18     set< int, decltype(compare) > S(compare);
19
20     vi previous( elements.size(), -1 );
21     for(int i=0; i<int( elements.size() ); ++i){
22         auto it = S.insert(i).first;
23         if(it != S.begin())
24             previous[i] = *prev(it);

```

```

25         if(*it == i and next(it) != S.end())
26             S.erase(next(it));
27     }
28
29     vi answer;
30     answer.push_back( *S.rbegin() );
31     while ( previous[answer.back()] != -1 )
32         answer.push_back( previous[answer.back()] );
33     reverse( answer.begin(), answer.end() );
34     return answer;
35 }

```

## 8 Strings

### 8.1 Manacher

```

1 // 0(n), d1 -> palindromo impar, d2 -> palindromo par
   (centro da direita)
2 void manacher(string &s, vi &d1, vi &d2) {
3     int n = s.size();
4     for(int i = 0, l = 0, r = -1; i < n; i++) {
5         int k = (i > r) ? 1 : min(d1[l + r - i], r -
6             i + 1);
7         while(0 <= i - k && i + k < n && s[i - k] ==
8             s[i + k]) {
9             k++;
10            d1[i] = k--;
11            if(i + k > r) {
12                l = i - k;
13                r = i + k;
14            }
15        }
16        for(int i = 0, l = 0, r = -1; i < n; i++) {
17            int k = (i > r) ? 0 : min(d2[l + r - i + 1],
18                r - i + 1);
19            while(0 <= i - k - 1 && i + k < n && s[i - k
20                - 1] == s[i + k]) {
21                k++;
22            }
23            d2[i] = k--;
24            if(i + k > r) {
25                l = i - k - 1;
26                r = i + k ;
27            }
28        }
29    }
30 }

```

### 8.2 Suffix Automaton

```

1 const int SA = 2*N; // Node 1 is the initial node of
   the automaton
2 int last = 1;
3 int len[SA], link[SA];
4 array<int, 26> to[SA]; // maybe map<int, int>
5 int lastID = 1;
6 void push(int c) {
7     int u = ++lastID;
8     len[u] = len[last] + 1;
9
10    int p = last;
11    last = u; // update last immediately
12    for (; p > 0 && !to[p][c]; p = link[p])
13        to[p][c] = u;
14
15    if (p == 0) { link[u] = 1; return; }
16
17    int q = to[p][c];
18    if (len[q] == len[p] + 1) { link[u] = q; return; }

```

```

19
20 int clone = ++lastID;
21 len[clone] = len[p] + 1;
22 link[clone] = link[q];
23 link[q] = link[u] = clone;
24 to[clone] = to[q];
25 for (int pp = p; to[pp][c] == q; pp = link[pp])
26     to[pp][c] = clone;
27 }

```

## 8.3 Edit Distance

```

1 int edit_distance(int a, int b, string& s, string& t)
2 {
3     // indexado em 0, transforma s em t
4     if(a == -1) return b+1;
5     if(b == -1) return a+1;
6     if(tab[a][b] != -1) return tab[a][b];
7
8     int ins = INF, del = INF, mod = INF;
9     ins = edit_distance(a-1, b, s, t) + 1;
10    del = edit_distance(a, b-1, s, t) + 1;
11    mod = edit_distance(a-1, b-1, s, t) + (s[a] != t[
12    b]);
13
14    return tab[a][b] = min(ins, min(del, mod));
15 }

```

## 8.4 Suffix Array

```

1 vi suffix_array(string s){
2     s.pb('$');
3     int n = s.size();
4
5     vi p(n), c(n);
6     vector< pair<char, int> > a(n);
7     for(int i=0;i<n;i++) a[i] = {s[i], i};
8     sort(a.begin(), a.end());
9
10    for(int i=0;i<n;i++) p[i] = a[i].ss;
11    c[p[0]]=0;
12    for(int i=1;i<n;i++)
13        c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff);
14
15    int k=0;
16    while((1<k) < n){
17        vector< pair<pii, int> > a(n);
18        for(int i=0;i<n;i++)
19            a[i] = {{c[i], c[(i+(1<k))%n]}, i};
20        sort(a.begin(), a.end());
21
22        for(int i=0;i<n;i++) p[i] = a[i].ss;
23        c[p[0]]=0;
24        for(int i=1;i<n;i++)
25            c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff);
26    }
27    k++;
28    return p;
29 }

```

## 8.5 Lcs

```

1 string LCSUBSTR(string X, string Y)
2 {
3     int m = X.size();
4     int n = Y.size();
5
6     int result = 0, end;
7     int len[2][n];
8     int currRow = 0;
9

```

```

10     for(int i=0;i<=m;i++){
11         for(int j=0;j<=n;j++){
12             if(i==0 || j==0)
13                 len[currRow][j] = 0;
14             else if(X[i-1] == Y[j-1]){
15                 len[currRow][j] = len[1-currRow][j-1]
16                 + 1;
17                 if(len[currRow][j] > result){
18                     result = len[currRow][j];
19                     end = i - 1;
20                 }
21             }
22             else
23                 len[currRow][j] = 0;
24         }
25         currRow = 1 - currRow;
26     }
27
28     if(result==0)
29         return string();
30
31     return X.substr(end - result + 1, result);
32 }

```

## 8.6 Eertree

```

1 // heavily based on https://ideone.com/YQX9jv,
2 // which adamant cites here https://codeforces.com/
3 // blog/entry/13959?comment=196033
4 struct Eertree {
5     int s[N];
6     int n, last, sz;
7
8     int len[N], link[N];
9     int to[N][A];
10
11    Eertree() {
12        s[n++] = -1;
13        len[1] = -1, link[1] = 1; // "backspace" root
14        // is 1
15        len[0] = 0, link[0] = 1; // empty root is 0
16        (to[backspace root][any char] = empty root)
17        last = 2;
18        sz = 2;
19    }
20
21    int get_link(int u) {
22        while (s[n - len[u] - 2] != s[n - 1]) u =
23        link[u];
24        return u;
25    }
26
27    void push(int c) {
28        s[n++] = c;
29        int p = get_link(last);
30        if (!to[p][c]) {
31            int u = ++sz;
32            len[u] = len[p] + 2;
33            link[u] = to[get_link(link[p])][c]; //
34            // may be 0 (empty), but never 1 (backspace)
35            to[p][c] = u;
36        }
37        last = to[p][c];
38    }
39 }

```

## 8.7 Aho Corasick

```

1 // https://github.com/joseleite19/icpc-notebook/blob/
2 // master/code/string/aho_corasick.cpp
3 int to[N][A];

```

```

3 int ne = 2, fail[N], term[N];
4 void add_string(const char *str, int id){
5     int p = 1;
6     for(int i = 0; str[i]; i++){
7         int ch = str[i] - 'a'; // !
8         if(!to[p][ch]) to[p][ch] = ne++;
9         p = to[p][ch];
10    }
11    term[p]++;
12 }
13 void init(){
14     for(int i = 0; i < ne; i++) fail[i] = 1;
15     queue<int> q; q.push(1);
16     int u, v; char c;
17     while(!q.empty()){
18         u = q.front(); q.pop();
19         for(int i = 0; i < A; i++){
20             if(to[u][i]){
21                 v = to[u][i]; q.push(v);
22                 if(u != 1){
23                     fail[v] = to[ fail[u] ][i];
24                     term[v] += term[ fail[v] ];
25                 }
26             }
27             else if(u != 1) to[u][i] = to[ fail[u] ][i];
28         }
29     }
30 }
31 }

```

## 8.8 Kmp

```

1 string p;
2 int neighbor[N];
3 int walk(int u, char c) { // leader after inputting '
4     while (u != -1 && (u+1) >= (int)p.size() || p[u+1] != c) // leader doesn't match
5         u = neighbor[u];
6     return p[u+1] == c ? u+1 : u;
7 }
8 void build() {
9     neighbor[0] = -1; // -1 is the leftmost state
10    for (int i = 1; i < (int)p.size(); i++)
11        neighbor[i] = walk(neighbor[i-1], p[i]);
12 }

```

## 8.9 Z Func

```

1 vi Z(string s) {
2     int n = s.size();
3     vi z(n);
4     int x = 0, y = 0;
5     for (int i = 1; i < n; i++) {
6         z[i] = max(0, min(z[i-x], y-i+1));
7         while (i+z[i] < n && s[z[i]] == s[i+z[i]]) {
8             x = i; y = i+z[i]; z[i]++;
9         }
10    }
11    return z;
12 }

```

## 8.10 Lcsubseq

```

1 // Longest Common Subsequence
2 string lcs(string x, string y){
3     int n = x.size(), m = y.size();
4     vector<vi> dp(n+1, vi(m+1, 0));
5
6     for(int i=0;i<=n;i++){
7         for(int j=0;j<=m;j++){

```

```

8             if(!i or !j)
9                 dp[i][j]=0;
10            else if(x[i-1] == y[j-1])
11                dp[i][j]=dp[i-1][j-1]+1;
12            else
13                dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
14        }
15    }
16
17    // int len = dp[n][m];
18    string ans="";
19
20    // recover string
21    int i = n-1, j = m-1;
22    while(i>=0 and j>=0){
23        if(x[i] == y[j]){
24            ans.pb(x[i]);
25            i--; j--;
26        }else if(dp[i][j+1]>dp[i+1][j])
27            i--;
28        else
29            j--;
30    }
31    reverse(ans.begin(), ans.end());
32
33    return ans;
34 }
35 }

```

## 8.11 Hash

```

1 struct Hash {
2     vector<unordered_set<ll>> h;
3     vector<ll> mods = {
4
5         1000000009,10000000021,10000000033,10000000087,10000000093,
6
7         10000000123,10000000181,10000000207,10000000223,10000000241,
8
9     };
10    ll p = 31;
11    int num;
12
13    Hash(int qt) {
14        srand(time(0));
15        num = qt;
16        h.assign(num, unordered_set<ll>());
17        random_shuffle(all(mods));
18    }
19
20    ll compute_hash(string const& s, ll p, ll m) {
21        ll res = 0, p_pow = 1;
22
23        for(char c : s) {
24            res = (res + (c-'a'+1) * p_pow) % m;
25            p_pow = (p_pow * p) % m;
26        }
27        return res;
28    }
29
30    void add(string const& s) {
31        forn(i, num) {
32            ll value = compute_hash(s, p, mods[i]);
33            h[i].insert(value);
34        }
35    }
36
37    bool query(string const& s) {
38        forn(i, num) {
39            ll val = compute_hash(s, p, mods[i]);
40            if(!h[i].count(val))
41                return false;

```

```

39     }
40     return true;
41 }
42 };

8.12 Suffix Array Radix

1 void radix_sort(vector<pii>& rnk, vi& ind) {
2     auto counting_sort = [](vector<pii>& rnk, vi& ind
3     ) {
4         int n = ind.size(), maxx = -1;
5         for(auto p : rnk) maxx = max(maxx, p.ff);
6
7         vi cnt(maxx+1, 0), pos(maxx+1), ind_new(n);
8         for(auto p : rnk) cnt[p.ff]++;
9         pos[0] = 0;
10
11         for(int i = 1; i <= maxx; i++) {
12             pos[i] = pos[i-1] + cnt[i-1];
13         }
14
15         for(auto idx : ind) {
16             int val = rnk[idx].ff;
17             ind_new[pos[val]] = idx;
18             pos[val]++;
19         }
20
21         swap(ind, ind_new);
22     };
23
24     for(int i = 0; i < (int)rnk.size(); i++) swap(rnk
25     [i].ff, rnk[i].ss);
26     counting_sort(rnk, ind);
27 }
28
29 vi suffix_array(const string& s) {
30     int n = s.size();
31     vector<pii> rnk(n, mp(0, 0));
32     vi ind(n);
33     forn(i, n) {
34         rnk[i].ff = (s[i] == '$') ? 0 : s[i] - 'a' + 1;
35         // manter '$' como 0
36         ind[i] = i;
37     }
38
39     for(int k = 1; k <= n; k = (k << 1)) {
40         for(int i = 0; i < n; i++) {
41             if(ind[i]+k >= n) {
42                 rnk[ind[i]].ss = 0;
43             }
44             else {
45                 rnk[ind[i]].ss = rnk[ind[i]+k].ff;
46             }
47         }
48         radix_sort(rnk, ind); // sort(all(rnk), cmp)
49         pra n*log(n), cmp com rnk[i] < rnk[j]
50
51         vector<pii> tmp = rnk;
52         tmp[ind[0]] = mp(1, 0); // rnk.ff comecar em
53         1 pois '$' eh o 0
54         for(int i = 1; i < n; i++) {
55             tmp[ind[i]].ff = tmp[ind[i-1]].ff;
56             if(rnk[ind[i]] != rnk[ind[i-1]]) {
57                 tmp[ind[i]].ff++;
58             }
59         }
60         swap(rnk, tmp);
61     }
62     return ind;
63 }

```

```

61 vi lcp_array(const string& s, const vi& sarray) {
62     vi inv(s.size());
63     for(int i = 0; i < (int)s.size(); i++) {
64         inv[sarray[i]] = i;
65     }
66     vi lcp(s.size());
67     int k = 0;
68     for(int i = 0; i < (int)s.size()-1; i++) {
69         int pi = inv[i];
70         if(pi-1 < 0) continue;
71         int j = sarray[pi-1];
72
73         while(s[i+k] == s[j+k]) k++;
74         lcp[pi] = k;
75         k = max(k-1, 0);
76     }
77
78     return vi(lcp.begin()+1, lcp.end()); // LCP(i, j)
79     = min(lcp[i], ..., lcp[j-1])
80 }

```

## 9 ED

### 9.1 Sparse Table

```

1 int logv[MAX+1];
2 void make_log() {
3     logv[1] = 0; // pre-computar tabela de log
4     for (int i = 2; i <= MAX; i++)
5         logv[i] = logv[i/2] + 1;
6 }
7 struct Sparse {
8     int n;
9     vector<vi> st;
10
11     Sparse(vi& v) {
12         n = v.size();
13         int k = logv[n];
14         st.assign(n+1, vi(k+1, 0));
15
16         forn(i, n) {
17             st[i][0] = v[i];
18         }
19
20         for(int j = 1; j <= k; j++) {
21             for(int i = 0; i + (1 << j) <= n; i++) {
22                 st[i][j] = f(st[i][j-1], st[i + (1 <<
23                 (j-1))][j-1]);
24             }
25         }
26
27         int f(int a, int b) {
28             return min(a, b);
29         }
30
31         int query(int l, int r) {
32             int k = logv[r-l+1];
33             return f(st[l][k], st[r - (1 << k) + 1][k]);
34         }
35 };
36
37 struct Sparse2d {
38     int n, m;
39     vector<vector<vi>> st;
40
41     Sparse2d(vector<vi> mat) {
42         n = mat.size();
43         m = mat[0].size();
44         int k = logv[min(n, m)];
45

```

```

46     st.assign(n+1, vector<vi>(m+1, vi(k+1)));
47     for(int i = 0; i < n; i++)
48         for(int j = 0; j < m; j++)
49             st[i][j][0] = mat[i][j];
50
51     for(int j = 1; j <= k; j++) {
52         for(int x1 = 0; x1 < n; x1++) {
53             for(int y1 = 0; y1 < m; y1++) {
54                 int delta = (1 << (j-1));
55                 if(x1+delta >= n or y1+delta >= m
56 ) continue;
57
58                 st[x1][y1][j] = st[x1][y1][j-1];
59                 st[x1][y1][j] = f(st[x1][y1][j],
60 st[x1+delta][y1][j-1]);
61                 st[x1][y1][j] = f(st[x1][y1][j],
62 st[x1][y1+delta][j-1]);
63                 st[x1][y1][j] = f(st[x1][y1][j],
64 st[x1+delta][y1+delta][j-1]);
65             }
66         }
67
68         // so funciona para quadrados
69         int query(int x1, int y1, int x2, int y2) {
70             assert(x2-x1+1 == y2-y1+1);
71             int k = logv[x2-x1+1];
72             int delta = (1 << k);
73
74             int res = st[x1][y1][k];
75             res = f(res, st[x2 - delta+1][y1][k]);
76             res = f(res, st[x1][y2 - delta+1][k]);
77             res = f(res, st[x2 - delta+1][y2 - delta+1][k]);
78         }
79
80         return res;
81     }
82
83     int f(int a, int b) {
84         return a | b;
85     }
86
87     };

```

## 9.2 Color Update

```

1 struct Color{
2     set<ti> inter; // l, r, color
3     vector<ti> update(int l, int r, int c){
4         if(inter.empty()){ inter.insert({l,r, c});
5         return {}; }
6         vector<ti> removed;
7         auto it = inter.lower_bound({l+1, 0, 0});
8         it = prev(it);
9         while(it != inter.end()){
10             auto [l1, r1, c1] = *it;
11             if((l1<=l and l1<=r) or (l<=r1 and r1<=r)
12 or (l1<=l and r<=r1)){
13                 removed.pb({l1, r1, c1});
14             }else if(l1 > r)
15                 break;
16             it = next(it);
17         }
18         for(auto [l1, r1, c1]: removed){
19             inter.erase({l1, r1, c1});
20             if(l1<l) inter.insert({l1, min(r1, l-1),
21 c1});
22             if(r<r1) inter.insert({max(l1, r+1), r1,
23 c1});
24         }
25         if(c != 0) inter.insert({l, r, c});
26         return removed;
27     }

```

```

23     }
24
25     ti query(int i){
26         if(inter.empty()) return {INF, INF, INF};
27         return *prev(inter.lower_bound({i+1, 0, 0}));
28     }
29 };

```

## 9.3 Segtree Pa

```

1 int N;
2 vl t(4*MAX, 0);
3 vl v(MAX, 0);
4 vector<p11> lazy(4*MAX, {0,0});
5 // [x, x+y, x+2y...] //
6
7 inline ll merge(ll a, ll b){
8     return a + b;
9 }
10
11 void build(int l=0, int r=N-1, int no=1){
12     if(l == r){ t[no] = v[l]; return; }
13     int mid = (l + r) / 2;
14     build(l, mid, 2*no);
15     build(mid+1, r, 2*no+1);
16     t[no] = merge(t[2*no], t[2*no+1]);
17 }
18
19 inline p11 sum(p11 a, p11 b){ return {a.ff+b.ff, a.ss
20 +b.ss}; }
21
22 inline void prop(int l, int r, int no){
23     auto [x, y] = lazy[no];
24     if(x==0 and y==0) return;
25     ll len = (r-l+1);
26     t[no] += (x + x + y*(len-1))*len / 2;
27     if(l != r){
28         int mid = (l + r) / 2;
29         lazy[2*no] = sum(lazy[2*no], lazy[no]);
30         lazy[2*no+1] = sum(lazy[2*no+1], {x + (mid-l
31 +1)*y, y});
32     }
33     lazy[no] = {0,0};
34 }
35
36 ll query(int a, int b, int l=0, int r=N-1, int no=1){
37     prop(l, r, no);
38     if(r<a or b<l) return 0;
39     if(a<=l and r<=b) return t[no];
40     int mid = (l + r) / 2;
41     return merge(
42         query(a, b, l, mid, 2*no),
43         query(a, b, mid+1, r, 2*no+1)
44     );
45 }
46
47 void update(int a, int b, ll x, ll y, int l=0, int r=
48 N-1, int no=1){
49     prop(l, r, no);
50     if(r<a or b<l) return;
51     if(a<=l and r<=b){
52         lazy[no] = {x, y};
53         prop(l, r, no);
54         return;
55     }
56     int mid = (l + r) / 2;
57     update(a, b, x, y, l, mid, 2*no);
58     update(a, b, x + max((mid-max(l, a)+1)*y, 0LL), y
59 , mid+1, r, 2*no+1);
60     t[no] = merge(t[2*no], t[2*no+1]);
61 }

```

## 9.4 Segtree Iterative Lazy

```

1 struct Segtree {
2     vector<ll> seg, lazy;
3     int n, LOG;
4
5     Segtree(int n=0){
6         this->n = n;
7         LOG = ceil(log2(n));
8         seg.assign(2*n, 0);
9         lazy.assign(2*n, 0);
10    }
11
12    ll merge(ll a, ll b){
13        return a + b;
14    }
15    void poe(int p, ll x, int tam, bool prop=1){
16        seg[p] += x*tam;
17        if(prop and p < n) lazy[p] += x;
18    }
19    void sobe(int p){
20        for(int tam = 2; p /= 2; tam *= 2){
21            seg[p] = merge(seg[2*p], seg[2*p+1]);
22            if(lazy[p]!=0)
23                poe(p, lazy[p], tam, 0);
24        }
25    }
26    void prop(int p){
27        int tam = 1 << (LOG-1);
28        for(int s = LOG; s; s--, tam /= 2){
29            int i = p >> s;
30            if(lazy[i]){
31                poe(2*i, lazy[i], tam);
32                poe(2*i+1, lazy[i], tam);
33                lazy[i] = 0;
34            }
35        }
36    }
37    void build(){
38        for(int i = n-1; i; i--)
39            seg[i] = merge(seg[2*i], seg[2*i+1]);
40    }
41    ll query(int a, int b){
42        ll ret = 0;
43        for(prop(a+=n), prop(b+=n); a <= b; ++a/=2,
44            --b/=2) {
45            if(a%2 == 1) ret = merge(ret, seg[a]);
46            if(b%2 == 0) ret = merge(ret, seg[b]);
47        }
48        return ret;
49    }
50    void update(int a, int b, int x){
51        int a2 = a += n, b2 = b += n, tam = 1;
52        for(; a <= b; ++a/=2, --b/=2, tam *= 2){
53            if(a%2 == 1) poe(a, x, tam);
54            if(b%2 == 0) poe(b, x, tam);
55        }
56        sobe(a2), sobe(b2);
57    }
58 };

```

## 9.5 Segtree Recursive

```

1 int N;
2 vector<ll> t(4*MAX, 0);
3 vector<ll> v(MAX, 0);
4 vector<ll> lazy(4*MAX, 0);
5
6 inline ll merge(ll a, ll b){
7     return a + b;
8 }
9

```

```

10 void build(int l=0, int r=N-1, int no=1){
11     if(l == r){ t[no] = v[l]; return; }
12     int mid = (l + r) / 2;
13     build(l, mid, 2*no);
14     build(mid+1, r, 2*no+1);
15     t[no] = merge(t[2*no], t[2*no+1]);
16 }
17
18 void prop(int l, int r, int no){
19     if(lazy[no] != 0){
20         t[no] += lazy[no] * (r-l+1);
21         if(l != r){
22             lazy[2*no] += lazy[no];
23             lazy[2*no+1] += lazy[no];
24         }
25         lazy[no] = 0;
26     }
27 }
28
29 ll query(int a, int b, int l=0, int r=N-1, int no=1){
30     prop(l, r, no);
31     if(r<a or b<l) return 0;
32     if(a<=l and r<=b) return t[no];
33     int mid = (l + r) / 2;
34     return merge(
35         query(a, b, l, mid, 2*no),
36         query(a, b, mid+1, r, 2*no+1)
37     );
38 }
39
40 void update(int a, int b, ll x, int l=0, int r=N-1,
41     int no=1){
42     prop(l, r, no);
43     if(r<a or b<l) return;
44     if(a<=l and r<=b){
45         lazy[no] += x;
46         prop(l, r, no);
47         return;
48     }
49     int mid = (l + r) / 2;
50     update(a, b, x, l, mid, 2*no);
51     update(a, b, x, mid+1, r, 2*no+1);
52     t[no] = merge(t[2*no], t[2*no+1]);
53 }

```

## 9.6 Segtree Maxsubarray

```

1 // Subarray with maximum sum
2 struct no{
3     ll p, s, t, b; // prefix, suffix, total, best
4     no(ll x=0): p(x), s(x), t(x), b(x){}
5 };
6
7 struct Segtree{
8     vector<no> t;
9     int n;
10
11     Segtree(int n){
12         this->n = n;
13         t.assign(2*n, no(0));
14     }
15
16     no merge(no l, no r){
17         no ans;
18         ans.p = max(0LL, max(l.p, l.t+r.p));
19         ans.s = max(0LL, max(r.s, l.s+r.t));
20         ans.t = l.t+r.t;
21         ans.b = max(max(l.b, r.b), l.s+r.p);
22         return ans;
23     }
24
25     void build(){
26         for(int i=n-1; i>0; i--)

```



```

27         t[i]=merge(t[i<<1], t[i<<1|1]);
28     }
29
30     no query(int l, int r){ // idx 0
31         no a(0), b(0);
32         for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
33             if(l&1)
34                 a=merge(a, t[l++]);
35             if(r&1)
36                 b=merge(t[--r], b);
37         }
38         return merge(a, b);
39     }
40
41     void update(int p, int value){
42         for(t[p+=n] = no(value); p >>= 1;){
43             t[p] = merge(t[p<<1], t[p<<1|1]);
44         }
45     }
46 };

```

## 9.7 Segtree Implicita Lazy

```

1 struct node{
2     pll val;
3     ll lazy;
4     ll l, r;
5     node(){
6         l=-1; r=-1; val={0,0}; lazy=0;
7     }
8 };
9
10 node tree[40*MAX];
11 int id = 2;
12 ll N=1e9+10;
13
14 pll merge(pll A, pll B){
15     if(A.ff==B.ff) return {A.ff, A.ss+B.ss};
16     return (A.ff<B.ff ? A:B);
17 }
18
19 void prop(ll l, ll r, int no){
20     ll mid = (l+r)/2;
21     if(l!=r){
22         if(tree[no].l==-1){
23             tree[no].l = id++;
24             tree[tree[no].l].val = {0, mid-l+1};
25         }
26         if(tree[no].r==-1){
27             tree[no].r = id++;
28             tree[tree[no].r].val = {0, r-(mid+1)+1};
29         }
30         tree[tree[no].l].lazy += tree[no].lazy;
31         tree[tree[no].r].lazy += tree[no].lazy;
32     }
33     tree[no].val.ff += tree[no].lazy;
34     tree[no].lazy=0;
35 }
36
37 void update(int a, int b, int x, ll l=0, ll r=2*N, ll
    no=1){
38     prop(l, r, no);
39     if(a<=l and r<=b){
40         tree[no].lazy += x;
41         prop(l, r, no);
42         return;
43     }
44     if(r<a or b<l) return;
45     int m = (l+r)/2;
46     update(a, b, x, l, m, tree[no].l);
47     update(a, b, x, m+1, r, tree[no].r);
48 }

```

```

49     tree[no].val = merge(tree[tree[no].l].val, tree[
    tree[no].r].val);
50 }
51
52 pll query(int a, int b, int l=0, int r=2*N, int no=1)
    {
53     prop(l, r, no);
54     if(a<=l and r<=b) return tree[no].val;
55     if(r<a or b<l) return {INF, 0};
56     int m = (l+r)/2;
57     int left = tree[no].l, right = tree[no].r;
58
59     return tree[no].val = merge(query(a, b, l, m,
    left),
60                                 query(a, b, m+1, r,
    right));
61 }

```

## 9.8 Segtree Iterative

```

1 // Segment Tree Iterativa - Max
2
3 struct Segtree{
4     vi t;
5     int n;
6
7     Segtree(int n){
8         this->n = n;
9         t.assign(2*n, 0);
10    }
11
12    int merge(int a, int b){
13        return max(a, b);
14    }
15
16    void build(){
17        for(int i=n-1; i>0; i--){
18            t[i]=merge(t[i<<1], t[i<<1|1]);
19        }
20
21    int query(int l, int r){ // [l, r]
22        int resl=-INF, resr=-INF;
23        for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
24            if(l&1) resl = merge(resl, t[l++]);
25            if(r&1) resr = merge(t[--r], resr);
26        }
27        return merge(resl, resr);
28    }
29
30    void update(int p, int value){
31        for(t[p+=n]=value; p >>= 1;){
32            t[p] = merge(t[p<<1], t[p<<1|1]);
33        }
34    }
35 };

```

## 9.9 Segtree Implicita

```

1 // SegTree Implicita O(nlogMAX)
2
3 struct node{
4     int val;
5     int l, r;
6     node(int a=0, int b=0, int c=0){
7         l=a; r=b; val=c;
8     }
9 };
10
11 int idx=2; // 1-> root / 0-> zero element
12 node t[8600010];
13 int N;
14

```

```

15 int merge(int a, int b){
16     return a + b;
17 }
18
19 void update(int pos, int x, int i=1, int j=N, int no
=1){
20     if(i==j){
21         t[no].val+=x;
22         return;
23     }
24     int meio = (i+j)/2;
25
26     if(pos<=meio){
27         if(t[no].l==0) t[no].l=idx++;
28         update(pos, x, i, meio, t[no].l);
29     }
30     else{
31         if(t[no].r==0) t[no].r=idx++;
32         update(pos, x, meio+1, j, t[no].r);
33     }
34
35     t[no].val=merge(t[t[no].l].val, t[t[no].r].val);
36 }
37
38 int query(int A, int B, int i=1, int j=N, int no=1){
39     if(B<i or j<A)
40         return 0;
41     if(A<=i and j<=B)
42         return t[no].val;
43
44     int mid = (i+j)/2;
45
46     int ans1 = 0, ansr = 0;
47
48     if(t[no].l!=0) ans1 = query(A, B, i, mid, t[no].l
);
49     if(t[no].r!=0) ansr = query(A, B, mid+1, j, t[no
].r);
50
51     return merge(ans1, ansr);
52 }

```

## 9.10 Mergesorttree

```

1 struct ST { // indexado em 0, O(n * log^2(n) )
2     int size;
3     vector<vl> v;
4
5     vl f(vl a, vl& b) {
6         vl res = a;
7         for(auto val : b) {
8             res.pb(val);
9         }
10        sort(all(res));
11        return res;
12    }
13
14    ST(int n) {
15        size = n;
16        v.assign(4*size, vl());
17    }
18
19    void build(vector<ll>& a, int lx=0, int rx=size
-1, int x=1) {
20        if(lx==rx) {
21            v[x].pb(a[lx]);
22            return;
23        }
24        int m = (lx+rx)/2;
25        build(a, lx, m, 2*x);
26        build(a, m+1, rx, 2*x+1);
27        v[x] = f(v[2*x], v[2*x+1]);
28    }

```

```

29
30    ll greaterequal(int l, int r, int k, int lx=0,
int rx=size-1, int x=1) {
31        if(r < lx or l > rx) return 0;
32        if(l <= lx and rx <= r) {
33            auto it = lower_bound(all(v[x]), k);
34            return (v[x].end() - it);
35        }
36        int m = (lx + rx)/2;
37        ll s1 = greaterequal(l, r, k, lx, m, 2*x);
38        ll s2 = greaterequal(l, r, k, m+1, rx, 2*x+1)
;
39
40        return s1 + s2;
41    }
42
43 };

```

## 9.11 Segpersistente Mkthnum

```

1 // kth number in range [l, r] if it was ordered
2 struct node{
3     int val;
4     int l, r;
5     node(int a=-1, int b=-1, int c=0){
6         val=c;l=a;r=b;
7     }
8 };
9
10 node tree[8600010]; // 4*nlog(4*n) space = 8600010
11 int idx=0;
12
13 int build(int l, int r){
14     if(l==r)
15         return idx++;
16
17     int mid = (l+r)/2;
18
19     tree[idx].l = build(l, mid);
20     tree[idx].r = build(mid+1, r);
21
22     return idx++;
23 }
24
25 int update(int l, int r, int root, int e){
26     if(l>e or r<e)
27         return root;
28     if(l==e and r==e){
29         tree[idx]=node(-1, -1, tree[root].val+1);
30         return idx++;
31     }
32     int mid = (l+r)/2;
33     tree[idx]=node(update(l, mid, tree[root].l, e),
update(mid+1, r, tree[root].r, e),
tree[root].val+1);
34     return idx++;
35 }
36
37 int query(int l, int r, int root1, int root2, int k){
38     while(l!=r)
39     {
40
41         int mid=(l+r)/2;
42         if(tree[tree[root2].l].val-tree[tree[root1].l
].val>=k)
43         {
44             r = mid;
45             root1 = tree[root1].l;
46             root2 = tree[root2].l;
47         }else
48         {
49             l = mid+1;
50             k-=tree[tree[root2].l].val-tree[tree[
root1].l].val;

```

```

51         root1 = tree[root1].r;
52         root2 = tree[root2].r;
53     }
54 }
55 return l;
56 }
57
58
59 int main()
60 {sws;
61
62     int n, m, a, b, k;
63     int v[MAX], aux[MAX];
64     int root[MAX];
65
66     cin >> n >> m;
67
68     for(int i=0; i<n; i++){
69         cin >> v[i]; aux[i]=v[i];
70     }
71
72     sort(v, v+n);
73
74     map<int, int> comp;
75     for(int i=0, j=0; i<n; i++){
76         if(i==0 or v[i]!=v[i-1])
77             comp[v[i]]=j++;
78
79     root[0]=build(0, n-1);
80
81     for(int i=1; i<=n; i++){
82         root[i] = update(0, n-1, root[i-1], comp[aux[
83 i-1]]);
84
85     for(int i=0; i<m; i++){
86         cin >> a >> b >> k;
87         cout << v[query(0, n-1, root[a-1], root[b], k
88 )] << endl;
89     }
90 }

```

## 9.12 Cht

```

1  const ll is_query = -LLINF;
2  struct Line{
3      ll m, b;
4      mutable function<const Line*> succ;
5      bool operator<(const Line& rhs) const{
6          if(rhs.b != is_query) return m < rhs.m;
7          const Line* s = succ();
8          if(!s) return 0;
9          ll x = rhs.m;
10         return b - s->b < (s->m - m) * x;
11     }
12 };
13 struct Cht : public multiset<Line>{ // maintain max m
14     *x+b
15     bool bad(iterator y){
16         auto z = next(y);
17         if(y == begin()){
18             if(z == end()) return 0;
19             return y->m == z->m && y->b <= z->b;
20         }
21         auto x = prev(y);
22         if(z == end()) return y->m == x->m && y->b <=
23 x->b;
24         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)
25 (y->b - z->b)*(y->m - x->m);
26     }
27     void insert_line(ll m, ll b){ // min -> insert (-
28 m, -b) -> -eval()

```

```

25         auto y = insert({ m, b });
26         y->succ = [=]{ return next(y) == end() ? 0 :
27 &*next(y); };
28         if(bad(y)){ erase(y); return; }
29         while(next(y) != end() && bad(next(y))) erase
30 (next(y));
31         while(y != begin() && bad(prev(y))) erase(
32 prev(y));
33     }
34     ll eval(ll x){
35         auto l = *lower_bound((Line) { x, is_query })
36 ;
37         return l.m * x + l.b;
38     }
39 };

```

## 9.13 Bit Kth

```

1  struct FT {
2      vector<int> bit; // indexado em 1
3      int n;
4
5      FT(int n) {
6          this->n = n + 1;
7          bit.assign(n + 1, 0);
8      }
9
10     int kth(int x){
11         int resp = 0;
12         x--;
13         for(int i=26; i>=0; i--){
14             if(resp + (1<<i) >= n) continue;
15             if(bit[resp + (1<<i)] <= x){
16                 x -= bit[resp + (1<<i)];
17                 resp += (1<<i);
18             }
19         }
20         return resp + 1;
21     }
22
23     void upd(int pos, int val){
24         for(int i = pos; i < n; i += (i&-i))
25             bit[i] += val;
26     }
27 };

```

## 9.14 Bit

```

1  struct FT {
2      vi bit; // indexado em 1
3      int n;
4
5      FT(int n) {
6          this->n = n+1;
7          bit.assign(n+2, 0);
8      }
9
10     int sum(int idx) {
11         int ret = 0;
12         for(++idx; idx > 0; idx -= idx & -idx)
13             ret += bit[idx];
14         return ret;
15     }
16
17     int sum(int l, int r) { // [l, r]
18         return sum(r) - sum(l - 1);
19     }
20
21     void add(int idx, int delta) {
22         for(++idx; idx < n; idx += idx & -idx)
23             bit[idx] += delta;
24     }
25 };

```

## 9.15 Virtual Tree

```

1 bool initialized = false;
2 int original_root = 1;
3 const int E = 2 * N;
4 vector<int> vt[N]; // virtual tree edges
5 int in[N], out[N], T, t[E<<1];
6 void dfs_time(int u, int p = 0) {
7     in[u] = ++T;
8     t[T + E] = u;
9     for (int v : g[u]) if (v != p) {
10         dfs_time(v, u);
11         t[++T + E] = u;
12     }
13     out[u] = T;
14 }
15
16 int take(int u, int v) { return in[u] < in[v] ? u : v; }
17 bool cmp_in(int u, int v) { return in[u] < in[v]; }
18 void build_st() {
19     in[0] = 0x3f3f3f3f;
20     for (int i = E-1; i > 0; i--)
21         t[i] = take(t[i<<1], t[i<<1|1]);
22 }
23
24 int query(int l, int r) {
25     int ans = 0;
26     for (l+=E, r+=E; l < r; l>>=1, r>>=1) {
27         if (l&1) ans = take(ans, t[l++]);
28         if (r&1) ans = take(ans, t[--r]);
29     }
30     return ans;
31 }
32
33 int get_lca(int u, int v) {
34     if (in[u] > in[v]) swap(u, v);
35     return query(in[u], out[v]+1);
36 }
37
38 int covers(int u, int v) { // does u cover v?
39     return in[u] <= in[v] && out[u] >= out[v];
40 }
41
42 int build_vt(vector<int>& vnodes) {
43     assert(initialized);
44
45     sort(all(vnodes), cmp_in);
46     int n = vnodes.size();
47     for (int i = 0; i < n-1; i++) {
48         int u = vnodes[i], v = vnodes[i+1];
49         vnodes.push_back(get_lca(u, v));
50     }
51     sort(all(vnodes), cmp_in);
52     vnodes.erase(unique(all(vnodes)), vnodes.end());
53
54     for (int u : vnodes)
55         vt[u].clear();
56
57     stack<int> s;
58     for (int u : vnodes) {
59         while (!s.empty() && !covers(s.top(), u))
60             s.pop();
61         if (!s.empty()) vt[s.top()].push_back(u);
62         s.push(u);
63     }
64     return vnodes[0]; // root
65 }
66
67 void initialize() {
68     initialized = true;
69     dfs_time(original_root);
70     build_st();

```

71 }

## 9.16 Treap

```

1 // source: https://github.com/victorsenam/caderno/
  blob/master/code/treap.cpp
2 //const int N = ; typedef int num;
3 num X[N]; int en = 1, Y[N], sz[N], L[N], R[N];
4 void calc (int u) { // update node given children
  info
5     if(!u) return;
6     sz[u] = sz[L[u]] + 1 + sz[R[u]];
7     // code here, no recursion
8 }
9 void unlaze (int u) {
10     if(!u) return;
11     // code here, no recursion
12 }
13 void split_val(int u, num x, int &l, int &r) { // l
  gets <= x, r gets > x
14     unlaze(u); if(!u) return (void) (l = r = 0);
15     if(X[u] <= x) { split_val(R[u], x, l, r); R[u] =
  l; l = u; }
16     else { split_val(L[u], x, l, r); L[u] = r; r = u;
  }
17     calc(u);
18 }
19 void split_sz(int u, int s, int &l, int &r) { // l
  gets first s, r gets remaining
20     unlaze(u); if(!u) return (void) (l = r = 0);
21     if(sz[L[u]] < s) { split_sz(R[u], s - sz[L[u]] -
  1, l, r); R[u] = l; l = u; }
22     else { split_sz(L[u], s, l, r); L[u] = r; r = u;
  }
23     calc(u);
24 }
25 int merge(int l, int r) { // els on l <= els on r
26     unlaze(l); unlaze(r); if(!l || !r) return l + r;
  int u;
27     if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
28     else { L[r] = merge(l, L[r]); u = r; }
29     calc(u); return u;
30 }
31 void init(int n=N-1) { // XXX call before using other
  funcs
32     for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i]
  = 1; L[i] = R[i] = 0; }
33     random_shuffle(Y + 1, Y + n + 1);
34 }
35 void insert(int &u, int it){
36     unlaze(u);
37     if(!u) u = it;
38     else if(Y[it] > Y[u]) split_val(u, X[it], L[it],
  R[it]), u = it;
39     else insert(X[it] < X[u] ? L[u] : R[u], it);
40     calc(u);
41 }
42 void erase(int &u, num key){
43     unlaze(u);
44     if(!u) return;
45     if(X[u] == key) u = merge(L[u], R[u]);
46     else erase(key < X[u] ? L[u] : R[u], key);
47     calc(u);
48 }
49 int create_node(num key){
50     X[en] = key;
51     sz[en] = 1;
52     L[en] = R[en] = 0;
53     return en++;
54 }
55 int query(int u, int l, int r){//0 index
56     unlaze(u);

```

```

57     if(u! or r < 0 or l >= sz[u]) return
identity_element;
58     if(l <= 0 and r >= sz[u] - 1) return subt_data[u
];
59     int ans = query(L[u], l, r);
60     if(l <= sz[ L[u] ] and sz[ L[u] ] <= r)
61         ans = max(ans, st[u]);
62     ans = max(ans, query(R[u], l-sz[L[u]]-1, r-sz[L[u]
]-1));
63     return ans;
64 }

```

## 9.17 Minqueue

```

1 struct MinQ {
2     stack<pair<ll,ll>> in;
3     stack<pair<ll,ll>> out;
4
5     void add(ll val) {
6         ll minimum = in.empty() ? val : min(val, in.
top().ss);
7         in.push(mp(val, minimum));
8     }
9
10    ll pop() {
11        if(out.empty()) {
12            while(!in.empty()) {
13                ll val = in.top().ff;

```

```

14            in.pop();
15            ll minimum = out.empty() ? val : min(
val, out.top().ss);
16            out.push({val, minimum});
17        }
18    }
19    ll res = out.top().ff;
20    out.pop();
21    return res;
22 }
23
24 ll minn() {
25     ll minimum = LLINF;
26     if(in.empty() || out.empty())
27         minimum = in.empty() ? (ll)out.top().ss :
(ll)in.top().ss;
28     else
29         minimum = min((ll)in.top().ss, (ll)out.
top().ss);
30
31     return minimum;
32 }
33
34 ll size() {
35     return in.size() + out.size();
36 }
37 };

```