



Notebook - Maratona de Programação

Posso mandar um WA?

Contents

1 Geometria	2	4.10 Fft Tourist	12
1.1 Linear Transformation	2	4.11 Matrix Exponentiation	14
1.2 Inside Polygon	2	4.12 Mulmod	14
1.3 Convex Hull	2	4.13 Raiz Primitiva	14
1.4 Sort By Angle	2	4.14 Bigmod	14
1.5 Minkowski Sum	2	4.15 Berlekamp Massey	15
1.6 Tetrahedron Distance3d	3	4.16 Double Gcd	15
1.7 Numintersectionline	3	4.17 Totient	15
1.8 Polygon Diameter	4	4.18 Kitamasa	16
1.9 Polygon Cut Length	4	4.19 Mobius	16
1.10 Mindistpair	4	5 Grafos	16
1.11 Rotating Callipers	4	5.1 Ford	16
1.12 Half Plane Intersect	5	5.2 2sat	16
1.13 2d	5	5.3 Kahn	17
1.14 Intersect Polygon	7	5.4 Hungarian	17
1.15 3d	7	5.5 Dfs Tree	17
2 Algoritmos	8	5.6 Lca	18
2.1 Cdq	8	5.7 Hld Aresta	18
2.2 Histogram Rectangle	9	5.8 Memf	19
3 Misc	9	5.9 Centroid	19
3.1 Rand	9	5.10 Kosaraju	20
3.2 Bitwise	9	5.11 Dinic	20
3.3 Template	10	5.12 Hld Vertice	20
3.4 Ordered Set	10	6 Numeric	21
4 Math	10	6.1 Lagrange Interpolation	21
4.1 Randommod	10	6.2 Newton Raphson	21
4.2 Inverso Mult	10	6.3 Simpson's Formula	21
4.3 Crt	10	7 DP	22
4.4 Gaussxor	11	7.1 Largest Ksubmatrix	22
4.5 Pollard Rho	11	7.2 Dp Digitos	22
4.6 Fast Exponentiaion	11	7.3 Partition Problem	22
4.7 Linear Diophantine Equation	11	7.4 Aliens	22
4.8 Miller Habin	11	7.5 Lis	23
4.9 Fft Simple	12		

8	Strings	23
8.1	Manacher	23
8.2	Suffix Automaton	23
8.3	Edit Distance	23
8.4	Suffix Array	24
8.5	Lcs	24
8.6	Eertree	24
8.7	Aho Corasick	24
8.8	Kmp	25
8.9	Z Func	25
8.10	Lcsubseq	25
8.11	Hash	25
8.12	Suffix Array Radix	25
9	ED	26
9.1	Dsu	26
9.2	Sparse Table	26
9.3	Color Update	27
9.4	Segtree Pa	27
9.5	Segtree Iterative Lazy	28
9.6	Segtree Recursive	28
9.7	Segtree Maxsubarray	28
9.8	Segtree Implicita Lazy	29
9.9	Segtree Iterative	29
9.10	Segtree Implicita	29
9.11	Mergesorttree	30
9.12	Segpersistente Mkthnum	30
9.13	Cht	31
9.14	Bit Kth	31
9.15	Bit	31
9.16	Minqueue	32

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
30 int inside(vp &p, point pp){
31     // 1 - inside / 0 - boundary / -1 - outside
32     int n = p.size();
33     for(int i=0; i<n; i++){
34         int j = (i+1)%n;
35         if(line({p[i], p[j]}).inside_seg(pp))
36             return 0;
37     }
38     int inter = 0;
39     for(int i=0; i<n; i++){
40         int j = (i+1)%n;
41         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
   [i], p[j], pp)==1)
42             inter++; // up
43         else if(p[j].x <= pp.x and pp.x < p[i].x and
   ccw(p[i], p[j], pp)==-1)
44             inter++; // down
45     }
46
47     if(inter%2==0) return -1; // outside
48     else return 1; // inside
49 }
```

1.3 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.
   back(), p)!=1)
7             L.pop_back();
8         L.pb(p);
9     }
10    reverse(P.begin(), P.end());
11    for(auto p: P){
12        while(U.size()>=2 and ccw(U[U.size()-2], U.
   back(), p)!=1)
13            U.pop_back();
14        U.pb(p);
15    }
16    L.pop_back();
17    L.insert(L.end(), U.begin(), U.end()-1);
18    return L;
19 }
```

1.4 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);
```

1.5 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         i = k;
5     for(int k = 0; k < (int)b.size(); k++){if(b[k]<b[j
   ])
6         j = k;
7
8     vp c;
9     c.reserve(a.size() + b.size());
10    for(int k = 0; k < int(a.size()+b.size()); k++){
11        point pt{a[i] + b[j]};
12        if((int)c.size() >= 2 and !ccw(c[c.size()-2],
   c.back(), pt))
13            c.pop_back();
14        c.pb(pt);
15        int q = i+1, w = j+1;
16        if(q == int(a.size())) q = 0;
17        if(w == int(b.size())) w = 0;
18        if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
   = q;
19        else j = w;
```

```

20     }
21
22     if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()-2]))
23         c.pop_back();
24     if(!ccw(c.back(), c[0], c[1])){
25         c[0]=c.back();
26         c.pop_back();
27     }
28     c.shrink_to_fit();
29
30     return c;
31 }

```

1.6 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
22         //prisma (va^vb eh nulo),
23         //ele esta dentro do prisma (poderia
24         //desconsiderar pois distancia
25         //vai ser a msm da distancia do ponto ao
26         //segmento)
27         if(!nulo(va^vb) and (v[(i+2)%3]*ve>d) ^ (p*ve
28         >d)) return LLINF;
29     }
30
31     //se ponto for coplanar ao triangulo (e dentro do
32     //triangulo)
33     //vai retornar zero corretamente
34     return fabs(misto(p-v[0],v1,v2)/norm(n));
35 }
36
37 ld dist_pt_seg(point p, vp li){
38     return norm((li[1]-li[0])^(p-li[0]))/norm(li[1]-
39     li[0]);
40 }
41
42 ld dist_line(vp l1, vp l2){
43     point n = (l1[1]-l1[0])^(l2[1]-l2[0]);
44     if(nulo(n)) //retas paralelas - dist ponto a reta
45     return dist_pt_seg(l2[0],l1);
46
47     point o1o2 = l2[0]-l1[0];
48     return fabs((o1o2*n)/norm(n));
49 }
50 // retas paralelas e intersecao nao nula
51 ld dist_seg(vp l1, vp l2){
52     assert(l2.size()==2);
53     assert(l1.size()==2);
54
55     //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
57         //ponto interno dos segs
58         for(int t=0;t<2;t++){
59             for(int i=0;i<2;i++){
60                 bool c=true;
61                 for(int k=0;k<2;k++){
62                     point va = l1[i]-l2[k];
63                     point vb = l2[k]-l2[k];
64                     ld ang = atan2(norm((vb^va)), vb*va);
65                     if(ang>PI/2) c = false;
66                 }
67                 if(c)
68                     ans = min(ans,dist_pt_seg(l1[i],l2));
69             }
70             swap(l1,l2);
71         }
72
73         //ponto interno com ponto interno dos segmentos
74         point v1 = l1[1]-l1[0], v2 = l2[1]-l2[0];
75         point n = v1^v2;
76         if(!nulo(n)){
77             bool ok = true;
78             for(int t=0;t<2;t++){
79                 point n2 = v2^n;
80                 point o1o2 = l2[0]-l1[0];
81                 ld escalar = (o1o2*n2)/(v1*n2);
82                 if(escalar<0 or escalar>1) ok = false;
83                 swap(l1,l2);
84                 swap(v1,v2);
85             }
86             if(ok) ans = min(ans,dist_line(l1,l2));
87         }
88     }
89     return ans;
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
102                 ], v));
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][
111                             i1], vet[0][j1]},
112                             {vet[1][
113                             i2], vet[1][j2]}));
114                         }
115                     }
116                 }
117             }
118         }
119     }
120     return ans;
121 }

```

1.7 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         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 }

```

1.8 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[j]) >
16             area(h[i], h[(i + 1) % m], h[(j + 1) % m]))
17             ++j;
18     }
19     return res;
20 }

```

1.9 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[i], p[j]}[0]));
19        } else if(signi > 0 and signj <= 0){
20            ans += param(a, b, inter_line({a, b}, {p[i], p[j]}[0]));
21        }
22    }
23
24    return abs(ans * norm(b-a));
25 }

```

1.10 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].x});
16        auto it2 = s.upper_bound({vet[i].y + d, vet[i].x});
17
18        for(auto it=it1; it!=it2; it++){
19            ll dx = vet[i].x - it->x;
20            ll dy = vet[i].y - it->y;
21            if(best_dist > dx*dx + dy*dy){
22                best_dist = dx*dx + dy*dy;
23                // vet[i] e inv(it)
24            }
25        }
26
27        s.insert(point(vet[i].y, vet[i].x));
28    }
29    return best_dist;
30 }

```

1.11 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
31    return ans;
32 }

```

1.12 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 }

```

1.13 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
14    point operator+(const point &o) const{
15        return {x+o.x, y+o.y};
16    }
17    point operator-(const point &o) const{
18        return {x-o.x, y-o.y};
19    }

```

```

20    point operator*(cod t) const{
21        return {x*t, y*t};
22    }
23    point operator/(cod t) const{
24        return {x/t, y/t};
25    }
26    cod operator*(const point &o) const{ // dot
27        return x * o.x + y * o.y;
28    }
29    cod operator^(const point &o) const{ // cross
30        return x * o.y - y * o.x;
31    }
32    bool operator<(const point &o) const{
33        if(!eq(x, o.x)) return x < o.x;
34        return y < o.y;
35    }
36    bool operator==(const point &o) const{
37        return eq(x, o.x) and eq(y, o.y);
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
52 bool nulo(point a){
53     return (eq(a.x, 0) and eq(a.y, 0));
54 }
55 point rotccw(point p, ld a){
56     // a = PI*a/180; // graus
57     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
58         +p.x*sin(a)));
59 }
60 point rot90cw(point a) { return point(a.y, -a.x); };
61 point rot90ccw(point a) { return point(-a.y, a.x); };
62
63 ld proj(point a, point b){ // a sobre b
64     return a*b/norm(b);
65 }
66
67 ld angle(point a, point b){ // em radianos
68     ld ang = a*b / norm(a) / norm(b);
69     return acos(max(min(ang, (ld)1), (ld)-1));
70 }
71
72 ld angle_vec(point v){
73     // return 180/PI*atan2(v.x, v.y); // graus
74     return atan2(v.x, v.y);
75 }
76
77 ld order_angle(point a, point b){ // from a to b ccw
78     (a in front of b)
79     ld aux = angle(a,b)*180/PI;
80     return ((a^b)<=0 ? aux:360-aux);
81 }
82
83 bool angle_less(point a1, point b1, point a2, point
84     b2){ // ang(a1,b1) <= ang(a2,b2)
85     point p1((a1*b1), abs((a1^b1)));
86     point p2((a2*b2), abs((a2^b2)));
87     return (p1^p2) <= 0;
88 }
89
90 ld area(vp &p){ // (points sorted)
91     ld ret = 0;
92     for(int i=2; i<(int)p.size(); i++){
93         ret += (p[i]-p[0])^(p[i-1]-p[0]);
94     }
95     return abs(ret/2);
96 }
97
98 ld areaT(point &a, point &b, point &c){

```

```

89     return abs((b-a)^(c-a))/2.0;
90 }
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 // be careful with precision error
177 vp inter_line(line l1, line l2){
178     ld det = l1.a*l2.b - l1.b*l2.a;
179     if(det==0) return {};
180     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
181     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
182     return {point(x, y)};
183 }
184
185 // segments not collinear
186 vp inter_seg(line l1, line l2){
187     vp ans = inter_line(l1, l2);
188     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.inside_seg(ans[0]))
189         return {};
190     return ans;
191 }
192
193 ld dist_seg(point p, point a, point b){ // point - seg
194     if(((p-a)*(b-a)) < EPS) return norm(p-a);
195     if(((p-b)*(a-b)) < EPS) return norm(p-b);
196     return abs((p-a)^(b-a))/norm(b-a);
197 }
198
199 ld dist_line(point p, line l){ // point - line
200     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
201 }
202
203 line bisector(point a, point b){
204     point d = (b-a)*2;
205     return line(d.x, d.y, a*a - b*b);
206 }
207
208 line perpendicular(line l, point p){ // passes through p
209     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
210 }
211
212
213 ///////////////
214 // Circle //
215 ///////////////
216
217 struct circle{
218     point c; cod r;
219     circle() : c(0, 0), r(0){}
220     circle(const point o) : c(o), r(0){}
221     circle(const point a, const point b){
222         c = (a+b)/2;
223         r = norm(a-c);
224     }
225
226     circle(const point a, const point b, const point cc){
227         c = inter_line(bisector(a, b), bisector(b, cc));
228     }

```

```

228     r = norm(a-c);
229 }
230 bool inside(const point &a) const{
231     return norm(a - c) <= r + EPS;
232 }
233 };
234
235 pair<point, point> getTangentPoint(circle cr, point p)
236 {
237     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
238     point p1 = rotccw(cr.c-p, -theta);
239     point p2 = rotccw(cr.c-p, theta);
240     assert(d1 >= cr.r);
241     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
242     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
243     return {p1, p2};
244 }
245
246 circle incircle(point p1, point p2, point p3){
247     ld m1 = norm(p2-p3);
248     ld m2 = norm(p1-p3);
249     ld m3 = norm(p1-p2);
250     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
251     ld s = 0.5*(m1+m2+m3);
252     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^v)/(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
295         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
296             ans = circle(v[i], v[j]);
297         }
298         for(int k=0;k<j;k++) if(!ans.inside(v[k])){
299             ans = circle(v[i], v[j], v[k]);
300         }
301     }
302 }
303
304 }
305 return ans;
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
1000 }

```

1.14 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 }

```

1.15 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 }

```



```

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 dot(a, b^c); // Area do paralelepipedo
57 }
58
59
60 struct plane{
61     point p1, p2, p3;
62     plane(point p1=0, point p2=0, point p3=0): p1(p1)
63     , p2(p2), p3(p3){}
64
65     point aux = (p1-p3)^(p2-p3);
66     cod a = aux.x, b = aux.y, c = aux.z;
67     cod d = -a*p1.x - b*p1.y - c*p1.z;
68     // ax+by+cz+d = 0;
69     cod eval(point &p){
70         return a*p.x + b*p.y + c*p.z + d;
71     };
72
73     cod dist(plane pl, point p){
74         return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d
75         ) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
76     }
77
78     point rotate(point v, point k, ld theta){
79         // Rotaciona o vetor v theta graus em torno do
80         eixo k
81         // theta *= PI/180; // graus
82         return rotated = (v*cos(theta)) +
83         ((k^v)*sin(theta)) +
84         (k*(k^v))*(1-cos(theta));
85     }

```

2 Algoritmos

2.1 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 void cdq(int l, int r, vector<point> &a, vi &dp){
51     if(l==r) return;
52
53     int mid = (l+r) / 2;
54
55     cdq(l, mid, a, dp);
56
57     // compress z
58     set<int> uz; map<int, int> idz;
59     for(int i=l;i<=r;i++) uz.insert(a[i].z);
60     int id = 0;
61     for(auto z: uz) idz[z] = id++;
62
63     vector<point> tmp;
64     for(int i=l;i<=r;i++){
65         tmp.pb(a[i]);
66         tmp.back().x = 0;
67         tmp.back().z = idz[tmp.back().z];
68         if(i<=mid)
69             tmp.back().left = true;
70     }
71
72     Segtree st(id);
73
74     sort(tmp.rbegin(), tmp.rend());
75
76     for(auto t: tmp){
77         if(t.left){
78             st.update(t.z, dp[t.id]);
79         }else{
80             dp[t.id] = max(dp[t.id], st.query(0, t.z
81             -1)+1);
82         }
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 }

```

2.2 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 T = S & (1<<j); // T = 0
51
52 // Least significant bit (lsb)
53
54 int lsb(int x){ return x&-x; }
55
56 // Exchange o j-th element
57
58 S ^= (1<<j)
59
60 // Position of the first bit on
61
62 T = (S & (-S))
63 T -> 4 bit ligado //(1000)
64
65 // Most significant digit of N

```

```

59     double K = log10(N);
60     K = K - floor(K);
61     int X = pow(10, K);
62
63     // Number of digits in N
64
65     X = floor(log10(N)) + 1;
66
67     // Power of two
68
69     bool isPowerOfTwo(int x){ return x && (!(x&(x
70 -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
94     int flog2ll(ll x){ return 64-1-
95     __builtin_clzll(x); }

```

3.3 Template

```

1  #include <bits/stdc++.h>
2  #define ff first
3  #define ss second
4  #define ll long long
5  #define ld long double
6  #define pb push_back
7  #define eb emplace_back
8  #define mp make_pair
9  #define mt make_tuple
10 #define pii pair<int, int>
11 #define vi vector<int>
12 #define vl vector<ll>
13 #define vii vector<pii>
14 #define sws ios_base::sync_with_stdio(false);cin.tie(
15     NULL);cout.tie(NULL);
16 #define endl '\n'
17 #define teto(a, b) ((a+b-1)/(b))
18 #define all(x) x.begin(), x.end()
19 #define forn(i, n) for(int i = 0; i < (int)n; i++)
20 #define forne(i, a, b) for(int i = a; i <= b; i++)
21 #define dbg(msg, var) cerr << msg << " " << var <<
22     endl;
23
24 using namespace std;
25
26 const int MAX = 200010;
27 const int MOD = 1000000007;
28 const int INF = 1e8;
29 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
30 const ld EPS = 1e-7;
31
32 // End Template //

```

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,
10     tree_order_statistics_node_update>;
11 // 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
9      (1000000007, 15000000000);
10     int num = distribution(rng);
11     while(!primo(num)) num++;
12     return num;
13 }

```

4.2 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
10     entao phi(m) = p-1
11     ll e = phim-1;
12     return fexp(a, e);
13 }

```

4.3 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 || 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 };

```

4.4 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 }
```

4.5 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
40     ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
41     while (t % 40 != 0 or gcd(prd, n) == 1) {
42         if (x==y) x = ++x0, y = f(x);
43         q = mul(prd, abs(x-y), n);
44         if (q != 0) prd = q;
45     }
46 }
```

```
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 }
```

4.6 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 }
```

4.7 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
```

4.8 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 }
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     // para n <= 2^64
25     // funciona para n <= 3*10^24 com os primos ate
26     // 41
27     for (int i : {2, 325, 9375, 28178, 450775,
28         9780504, 795265022}) {
29         if (i >= n) break;
30         ll x = expo(i, d, n);
31         if (x == 1 or x == n - 1) continue;
32
33         bool deu = 1;
34         for (int j = 0; j < r - 1; j++) {
35             x = mul(x, x, n);
36             if (x == n - 1) {
37                 deu = 0;
38                 break;
39             }
40         }
41         if (deu) return 0;
42     }
43     return 1;
44 }

```

4.9 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 vl multiply(vl const& a, vl const& b){
49     vector<num> fa(a.begin(), a.end());
50     vector<num> fb(b.begin(), b.end());
51     int n = 1;
52     while(n < int(a.size() + b.size()) )
53         n <= 1;
54     fa.resize(n);
55     fb.resize(n);
56     fft(fa, false);
57     fft(fb, false);
58     for(int i=0;i<n;i++)
59         fa[i] = fa[i]*fb[i];
60     fft(fa, true);
61     vl result(n);
62     for(int i=0;i<n;i++)
63         result[i] = round(fa[i].a);
64     while(result.back()==0) result.pop_back();
65     return result;
66 }
67 }

```

4.10 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     rev.resize(1 << nbase);
23     for(int i = 0; i < (1 << nbase); i++)
24         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
25             nbase - 1));
26     roots.resize(1 << nbase);
27     while(base < nbase){
28         ld angle = 2*PI / (1 << (base + 1));
29         for(int i = 1 << (base - 1); i < (1 << base);
30             i++){
31             roots[i << 1] = roots[i];
32             ld angle_i = angle * (2 * i + 1 - (1 <<
33                 base));

```

```

31         roots[(i << 1) + 1] = num(cos(angle_i),
32         sin(angle_i));
33     }
34     base++;
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 vi multiply_mod(vi &a, vi &b, int m, int eq = 0){
92     int need = a.size() + b.size() - 1;
93     int nbase = 0;
94     while((1 << nbase) < need) nbase++;
95     ensure_base(nbase);
96     int sz = 1 << nbase;
97     if(sz > (int) fa.size())
98         fa.resize(sz);
99
100
101     for(int i=0;i<(int)a.size();i++){
102         int x = (a[i] % m + m) % m;
103         fa[i] = num(x & ((1 << 15) - 1), x >> 15);
104     }
105     fill(fa.begin() + a.size(), fa.begin() + sz, num
106 {0, 0});
107     fft(fa, sz);
108     if(sz > (int) fb.size())
109         fb.resize(sz);
110     if(eq)
111         copy(fa.begin(), fa.begin() + sz, fb.begin())
112 ;
113     else{
114         for(int i = 0; i < (int) b.size(); i++){
115             int x = (b[i] % m + m) % m;
116             fb[i] = num(x & ((1 << 15) - 1), x >> 15)
117 ;
118         }
119         fill(fb.begin() + b.size(), fb.begin() + sz,
120 num {0, 0});
121         fft(fb, sz);
122     }
123     ld ratio = 0.25 / sz;
124     num r2(0, -1);
125     num r3(ratio, 0);
126     num r4(0, -ratio);
127     num r5(0, 1);
128     for(int i=0;i<=(sz >> 1);i++) {
129         int j = (sz - i) & (sz - 1);
130         num a1 = (fa[i] + conj(fa[j]));
131         num a2 = (fa[i] - conj(fa[j])) * r2;
132         num b1 = (fb[i] + conj(fb[j])) * r3;
133         num b2 = (fb[i] - conj(fb[j])) * r4;
134         if(i != j){
135             num c1 = (fa[j] + conj(fa[i]));
136             num c2 = (fa[j] - conj(fa[i])) * r2;
137             num d1 = (fb[j] + conj(fb[i])) * r3;
138             num d2 = (fb[j] - conj(fb[i])) * r4;
139             fa[i] = c1 * d1 + c2 * d2 * r5;
140             fb[i] = c1 * d2 + c2 * d1;
141         }
142         fa[j] = a1 * b1 + a2 * b2 * r5;
143         fb[j] = a1 * b2 + a2 * b1;
144     }
145     fft(fa, sz);
146     fft(fb, sz);
147     vi res(need);
148     for(int i=0;i<need;i++){
149         ll aa = fa[i].x + 0.5;
150         ll bb = fb[i].x + 0.5;
151         ll cc = fa[i].y + 0.5;
152         res[i] = (aa + ((bb % m) << 15) + ((cc % m)
153 << 30)) % m;
154     }
155     return res;
156 }
157
158 int main()
159 {sws;
160
161     //FFT
162     vi fx{1, 2, 3}; // 1+2x+3x^2
163     vi gx{4, 5}; // 4+5x
164     vi res;
165
166     res = multiply(fx,gx); //4 + 13x + 22x^2 + 15x^3
167
168     return 0;
169 }

```

4.11 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
23         multiplicar
24         vector<vl> res(r, vl(o.c, 0));
25
26         for(int i = 0; i < r; i++) {
27             for(int k = 0; k < c; k++) {
28                 for(int j = 0; j < o.c; j++) {
29                     res[i][j] = (res[i][j] + m[i][k]*
30                     o.m[k][j]) % MOD;
31                 }
32             }
33         }
34         return Matrix(res);
35     };
36
37     Matrix fexp(Matrix b, int e, int n) {
38         if(e == 0) return Matrix(n, n, true); //
39         identidade
40         Matrix res = fexp(b, e/2, n);
41         res = (res * res);
42         if(e%2) res = (res * b);
43
44         return res;
45     }
46 }
```

4.12 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 }
```

4.13 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 || 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,  $p^k$ ,  $2p^k$ , p eh
37 primo impar, k inteiro ---  $O(n \log^2(n))$ 
38 ll achar_raiz(ll mod, ll phi) {
39     if(mod == 2) return 1;
40     vl fat, elementos;
41     fat = fatorar(phi);
42
43     for(ll i = 2; i <= mod-1; i++) {
44         if(raiz_prim(i, mod, phi, fat))
45             return i;
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     return raizes;
61 }
62 }
```

4.14 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 }

```

4.15 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];
61 //calculate p*q mod f
62 inline void mull(ll*p,ll*q)
63 {
64     for(int i=0;i<m+m;++i) t_[i]=0;
65     for(int i=0;i<m;++i) if(p[i])
66         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;

```

4.16 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 }

```

4.17 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, 0(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+p;i<=n;i+=p){
32             isprime[i] = false;
33             tot[i] = (tot[i]/p)*(p-1);
34         }
35     }
36 }
37 return tot;
38 }

```

4.18 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
17         quotient term
18         for (int j = 0; j < len(b); j++)
19             c[j+k] = c[j+k] - c[i] * b[j];
20     }
21     c.resize(len(b) - 1);
22     return c;
23 }
24
25 poly ppwr(poly x, ll e, poly f) {
26     poly ans = { 1 };
27     for (; e > 0; e /= 2) {
28         if (e & 1) ans = pmod(pmul(ans, x), f);
29         x = pmod(pmul(x, x), f);
30     }
31     return ans;
32 }
33
34 // values = { A0, A1, ..., An }. recurrence = C0 * A0
35 // + C1 * A1 + ... + Cn * An generates A{n+1}
36 mint kitamasa(const poly& values, const poly&
37 recurrence, ll n) {
38     poly f(len(recurrence) + 1);
39     f.back() = 1;
40     for (int i = 0; i < len(recurrence); i++)
41         f[i] = mint(0) - recurrence[i];
42
43     auto d = ppwr(poly{0, 1}, n, f); // x^N mod f(x)
44
45     mint ans = 0;
46     for (int i = 0; i < len(values); i++)
47         ans = ans + d[i] * values[i];
48     return ans;
49 }

```

4.19 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]
32            ].c - edges[e].f) > 0) {
33                if(int a = dfs(edges[e].to, t, min(f,
34                edges[e].c-edges[e].f), tempo)) {
35                    edges[e].f += a;
36                    edges[e^1].f -= a;
37                    return a;
38                }
39            }
40        }
41        return 0;
42    }
43
44    int flow(int s, int t) {
45        int mflow = 0, tempo = 1;
46        while(int a = dfs(s, t, INF, tempo)) {
47            mflow += a;
48            tempo++;
49        }
50        return mflow;
51    }
52 }

```

5.2 2sat

```

1 vector<int> g[MAX], gt[MAX], S; int vis[MAX], cor[MAX]
2 ];

```

```

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    }
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 };

```

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
13    for(auto v : g[u]) {
14        if(p == v or vis[v]) continue;
15        dfs(v, u);
16        backedges[u] += backedges[v];
17    }
18    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         }
23         int small(int r, int sz = b) { return r-msb(mask[
24             r]&((1<<sz)-1)); }
25         T query(int l, int r) {
26             if (r-l+1 <= b) return small(r, r-l+1);
27             int ans = op(small(l+b-1), small(r));
28             int x = l/b+1, y = r/b-1;
29             if (x <= y) {
30                 int j = msb(y-x+1);
31                 ans = op(ans, op(t[n/b*j+x], t[n/b*j+y
32                 -(1<<j)+1]));
33             }
34             return ans;
35         }
36     };
37 };
38
39 namespace lca {
40     vector<int> g[MAX];
41     int v[2*MAX], pos[MAX], dep[2*MAX];
42     int t;
43     rmq<int> RMQ;
44
45     void dfs(int i, int d = 0, int p = -1) {
46         v[t] = i, pos[i] = t, dep[t++] = d;
47         for (int j : g[i]) if (j != p) {
48             dfs(j, d+1, i);
49             v[t] = i, dep[t++] = d;
50         }
51     }
52     void build(int n, int root) {
53         t = 0;
54         dfs(root);
55         RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
56     }
57     int lca(int a, int b) {
58         a = pos[a], b = pos[b];
59         return v[RMQ.query(min(a, b), max(a, b))];
60     }
61     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 struct Hld {
2     Segtree st;
3     int n;
4     vector<vii> g;
5     vi pos, sz, sobe, pai, h, v;
6     int t;
7
8     Hld(int n){
9         this->n=n;
10        st = Segtree(n);
11        g.assign(n, vii());
12        pos.assign(n, 0);sz.assign(n, 0);
13        sobe.assign(n, 0);pai.assign(n, 0);
14        h.assign(n, 0);v.assign(n, 0);
15    }
16
17    void build_hld(int k, int p = -1, int f = 1){
18        v[pos[k] = t++] = sobe[k]; sz[k] = 1;
19        for(auto &i: g[k]) if(i.ff != p){
20            sobe[i.ff] = i.ss; pai[i.ff] = k;
21            h[i.ff] = (i==g[k][0] ? h[k]:i.ff);
22            build_hld(i.ff, k, f); sz[k]+=sz[i.ff];
23
24            if(sz[i.ff]>sz[g[k][0].ff] or g[k][0].ff
25                ==p) swap(i, g[k][0]);
26            if(p*f == -1) build_hld(h[k] = k, -1, t = 0);
27        }
28    }
29    void build(int root = 0){
30        t = 0;
31        build_hld(root);
32        for(int i=0;i<n;i++) st.seg[i+n]=v[i];
33        st.build();
34    }
35    ll query_path(int a, int b){
36        if(a==b) return 0;
37        if(pos[a]<pos[b]) swap(a, b);
38
39        if(h[a]==h[b]) return st.query(pos[b]+1, pos[
40            a]);
41        return st.query(pos[h[a]], pos[a]) +
42            query_path(pai[h[a]], b);
43    }
44    void update_path(int a, int b, int x){
45        if(a==b) return;
46        if(pos[a]<pos[b]) swap(a, b);
47
48        if(h[a]==h[b]) return (void)st.update(pos[b
49            ]+1, pos[a], x);
50        st.update(pos[h[a]], pos[a], x); update_path(
51            pai[h[a]], b, x);
52    }
53    ll query_subtree(int a){
54        if(sz[a]==1) return 0;
55        return st.query(pos[a]+1, pos[a]+sz[a]-1);
56    }
57    void update_subtree(int a, int x){
58        if(sz[a]==1) return;
59        st.update(pos[a]+1, pos[a]+sz[a]-1, x);
60    }
61    int lca(int a, int b){
62        if(pos[a] < pos[b]) swap(a, b);
63        return (h[a]==h[b] ? b:lca(pai[h[a]], b));
64    }
65 };

```

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

```

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.
cap)
                continue;          // v.cap - v.flow
< lim

```

```

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

```

5.12 Hld Vertice

```

1 struct Hld {
2     Segtree st;
3     int n;
4     vector<vi> g;
5     vi pos, sz, peso, pai, h, v;
6     int t;
7
8     Hld(int n){
9         this->n=n;
10        st = Segtree(n);
11        g.assign(n, vi());
12        pos.assign(n, 0);sz.assign(n, 0);
13        peso.assign(n, 0);pai.assign(n, 0);
14        h.assign(n, 0);v.assign(n, 0);

```

```

15     }
16
17 void build_hld(int k, int p = -1, int f = 1){
18     v[pos[k] = t++] = peso[k]; sz[k] = 1;
19     for(auto &i: g[k]) if(i!=p){
20         pai[i] = k;
21         h[i] = (i==g[k][0] ? h[k]:i);
22         build_hld(i, k, f); sz[k]+=sz[i];
23
24         if(sz[i]>sz[g[k][0]] or g[k][0]==p) swap(
25             i, g[k][0]);
26     }
27     if(p*f == -1) build_hld(h[k] = k, -1, t = 0);
28 }
29 void build(int root = 0){
30     t = 0;
31     build_hld(root);
32     for(int i=0;i<n;i++) st.seg[i+n]=v[i];
33     st.build();
34 }
35 ll query_path(int a, int b){
36     if(pos[a]<pos[b]) swap(a, b);
37
38     if(h[a]==h[b]) return st.query(pos[b], pos[a
39 ]);
40     return st.query(pos[h[a]], pos[a]) +
41     query_path(pai[h[a]], b);
42 }
43 void update_path(int a, int b, int x){
44     if(pos[a]<pos[b]) swap(a, b);
45
46     if(h[a]==h[b]) return (void)st.update(pos[b],
47     pos[a], x);
48     st.update(pos[h[a]], pos[a], x); update_path(
49     pai[h[a]], b, x);
50 }
51 ll query_subtree(int a){
52     return st.query(pos[a], pos[a]+sz[a]-1);
53 }
54 void update_subtree(int a, int x){
55     st.update(pos[a], pos[a]+sz[a]-1, x);
56 }
57 int lca(int a, int b){
58     if(pos[a]<pos[b]) swap(a, b);
59     return (h[a]==h[b] ? b:lca(pai[h[a]], b));
60 }
61 };

```

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         y += yi;
12     }
13     return y;
14 }
15
16 // O(n)
17 template<typename T = mint>
18 struct Lagrange {
19

```

```

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
34     T eval(T x) {
35         l[0] = 1;
36         for (int i = 1; i < n; i++)
37             l[i] = l[i-1] * (x + -T(i-1));
38
39         r[n - 1] = 1;
40         for (int i = n - 2; i >= 0; i--)
41             r[i] = r[i+1] * (x + -T(i+1));
42
43         T ans = 0;
44         for (int i = 0; i < n; i++) {
45             T num = l[i] * r[i];
46             ans = ans + y[i] * num * den[i];
47         }
48         return ans;
49     }
50 };

```

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 }

```

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; //
13    aprox. good enough
14    return rsimpson(slm,fl,fmid,fml,l,mid)+rsimpson(
15    smr,fmid,fr,fmr,mid,r);
16 }
17
18 ld integrate(ld l, ld r)
19 {
20     ld mid = (l+r)/2;

```

```

18     ld fl = f(l), fr = f(r);
19     ld fmid = f(mid);
20     return rsimpson(simpson(fl, fr, fmid, l, r), fl, fr,
21         fmid, l, r);

```

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

```

7.2 Dp Digitos

```

1 // dp de quantidade de numeros <= r com ate qt
   digitos diferentes de 0
2 ll dp(int idx, string& r, bool menor, int qt, vector<
   vector<vi>>& tab) {
3     if(qt > 3) return 0;
4     if(idx >= r.size()) {
5         return 1;
6     }
7     if(tab[idx][menor][qt] != -1)
8         return tab[idx][menor][qt];
9
10    ll res = 0;
11    for(int i = 0; i <= 9; i++) {
12        if(menor or i <= r[idx] - '0') {
13            res += dp(idx+1, r, menor or i < (r[idx] -
14                '0'), qt+(i>0), tab);
15        }
16    }
17    return tab[idx][menor][qt] = res;
18 }

```

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        for(int i=1; i<=sum/2; i++){
17            part[i][0] = false;
18
19            for(int i=1; i<=sum/2; i++){
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
24                        -1];
25                }
26            }
27    }

```

7.4 Aliens

```

1 // Solves https://codeforces.com/contest/1279/problem
   /F
2
3 // dado um vetor de inteiros, escolha k subsegmentos
   disjuntos de soma máxima
4 // em vez de rodar a dp[i][k] = melhor soma até i
   usando k segmentos,
5 // vc roda uma dp[i] adicionando um custo W toda vez
   que usa um novo subsegmento,
6 // e faz busca binária nesse W pra achar o custo
   mínimo que usa exatamente K intervalos
7
8 ll n, k, L;
9 pll check(ll w, vl& v){
10    vector<pll> dp(n+1);
11    dp[0] = {0,0};
12    for(int i=1; i<=n; i++){
13        dp[i] = dp[i-1];
14        dp[i].ff += v[i];
15        if(i-L>=0){
16            pll t = {dp[i-L].ff + w, dp[i-L].ss + 1};
17            dp[i] = min(dp[i], t);
18        }
19    }
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]); // low for 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
13 // -180208
14
15 vi LIS(const vi &elements){
16     auto compare = [&](int x, int y) {
17         return elements[x] < elements[y];
18     };
19     set< int, decltype(compare) > S(compare);
20
21     vi previous( elements.size(), -1 );
22     for(int i=0; i<int( elements.size() ); ++i){
23         auto it = S.insert(i).first;
24         if(it != S.begin())
25             previous[i] = *prev(it);
26         if(*it == i and next(it) != S.end())
27             S.erase(next(it));
28     }
29
30     vi answer;
31     answer.push_back( *S.rbegin() );
32     while ( previous[answer.back()] != -1 )
33         answer.push_back( previous[answer.back()] );
34     reverse( answer.begin(), answer.end() );
35     return answer;
36 }

```

8 Strings

8.1 Manacher

```

1 // 0(n), d1 -> palindromo impar, d2 -> palindromo par
2 // (centro da direita)
3 void manacher(string &s, vi &d1, vi &d2) {
4     int n = s.size();
5     for(int i = 0, l = 0, r = -1; i < n; i++) {
6         int k = (i > r) ? 1 : min(d1[l + r - i], r -
7         i + 1);

```

```

6         while(0 <= i - k && i + k < n && s[i - k] ==
7         s[i + k]) {
8             k++;
9         }
10        d1[i] = k--;
11        if(i + k > r) {
12            l = i - k;
13            r = i + k;
14        }
15    }
16
17    for(int i = 0, l = 0, r = -1; i < n; i++) {
18        int k = (i > r) ? 0 : min(d2[l + r - i + 1],
19        r - i + 1);
20        while(0 <= i - k - 1 && i + k < n && s[i - k
21        - 1] == s[i + k]) {
22            k++;
23        }
24        d2[i] = k--;
25        if(i + k > r) {
26            l = i - k - 1;
27            r = i + k ;
28        }
29    }
30 }

```

8.2 Suffix Automaton

```

1 const int SA = 2*N; // Node 1 is the initial node of
2 // the automaton
3 int last = 1;
4 int len[SA], link[SA];
5 array<int, 26> to[SA]; // maybe map<int, int>
6 int lastID = 1;
7 void push(int c) {
8     int u = ++lastID;
9     len[u] = len[last] + 1;
10
11     int p = last;
12     last = u; // update last immediately
13     for (; p > 0 && !to[p][c]; p = link[p])
14         to[p][c] = u;
15
16     if (p == 0) { link[u] = 1; return; }
17
18     int q = to[p][c];
19     if (len[q] == len[p] + 1) { link[u] = q; return; }
20
21     int clone = ++lastID;
22     len[clone] = len[p] + 1;
23     link[clone] = link[q];
24     link[q] = link[u] = clone;
25     to[clone] = to[q];
26     for (int pp = p; to[pp][c] == q; pp = link[pp])
27         to[pp][c] = clone;

```

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]);

```



```

11     return tab[a][b] = min(ins, min(del, mod));
12 }
13 }

```

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
16    int k=0;
17    while((1<<k) < n){
18        vector< pair<pii, int> > a(n);
19        for(int i=0;i<n;i++){
20            a[i] = {{c[i], c[(i+(1<<k))%n]}}, i};
21            sort(a.begin(), a.end());
22
23            for(int i=0;i<n;i++) p[i] = a[i].ss;
24            c[p[0]]=0;
25            for(int i=1;i<n;i++){
26                c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff);
27            }
28            k++;
29        }
30    }
31    return p;
32 }

```

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[currRow-1][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];
4 int ne = 2, fail[N], term[N];
5 void add_string(const char *str, int id){
6     int p = 1;
7     for(int i = 0; str[i]; i++){
8         int ch = str[i] - 'a'; // !
9         if(!to[p][ch]) to[p][ch] = ne++;
10        p = to[p][ch];
11    }
12    term[p]++;
13
14    void init(){
15        for(int i = 0; i < ne; i++) fail[i] = 1;
16        queue<int> q; q.push(1);
17        int u, v; char c;
18        while(!q.empty()){
19            u = q.front(); q.pop();
20            for(int i = 0; i < A; i++){
21                if(to[u][i]){
22                    v = to[u][i]; q.push(v);
23                    if(u != 1){
24                        fail[v] = to[fail[u]][i];
25                        term[v] += term[fail[v]];
26                    }
27                }
28            }
29            else if(u != 1) to[u][i] = to[fail[u]][i];
30        }
31    }
32 }

```

```

28         else to[u][i] = 1;
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 +
5         1] != c) // leader doesn't match
6         u = neighbor[u];
7     return p[u + 1] == c ? u+1 : u;
8 }
9 void build() {
10     neighbor[0] = -1; // -1 is the leftmost state
11     for (int i = 1; i < (int)p.size(); i++)
12         neighbor[i] = walk(neighbor[i-1], p[i]);
13 }

```

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 || !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 }

```

```

34     return ans;
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;
42        }
43        return true;
44    }
45 }

```

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 }

```

```

18     }
19
20     swap(ind, ind_new);
21 };
22
23 for(int i = 0; i < (int)rnk.size(); i++) swap(rnk
[i].ff, rnk[i].ss);
24 counting_sort(rnk, ind);
25 for(int i = 0; i < (int)rnk.size(); i++) swap(rnk
[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     for(int k = 1; k <= n; k = (k << 1)) {
39         for(int i = 0; i < n; i++) {
40             if(ind[i] + k >= n) {
41                 rnk[ind[i]].ss = 0;
42             }
43             else {
44                 rnk[ind[i]].ss = rnk[ind[i] + k].ff;
45             }
46         }
47         radix_sort(rnk, ind); // sort(all(rnk), cmp)
48         pra n*log(n), cmp com rnk[i] < rnk[j]
49
50         vector<pii> tmp = rnk;
51         tmp[ind[0]] = mp(1, 0); // rnk.ff comecar em
52         1 pois '$' eh o 0
53         for(int i = 1; i < n; i++) {
54             tmp[ind[i]].ff = tmp[ind[i-1]].ff;
55             if(rnk[ind[i]] != rnk[ind[i-1]]) {
56                 tmp[ind[i]].ff++;
57             }
58         }
59         swap(rnk, tmp);
60     }
61     return ind;
62 }
63
64 vi lcp_array(const string& s, const vi& sarray) {
65     vi inv(s.size());
66     for(int i = 0; i < (int)s.size(); i++) {
67         inv[sarray[i]] = i;
68     }
69     vi lcp(s.size());
70     int k = 0;
71     for(int i = 0; i < (int)s.size() - 1; i++) {
72         int pi = inv[i];
73         if(pi - 1 < 0) continue;
74         int j = sarray[pi - 1];
75
76         while(s[i + k] == s[j + k]) k++;
77         lcp[pi] = k;
78         k = max(k - 1, 0);
79     }
80     return vi(lcp.begin() + 1, lcp.end()); // LCP(i, j)
81     = min(lcp[i], ..., lcp[j - 1])

```

9 ED

9.1 Dsu

```

1 struct DSU {
2     int n;
3     vi parent, size;
4
5     DSU(int n) {
6         this->n = n;
7         parent.assign(n + 1, 0);
8         size.assign(n + 1, 1);
9
10        for(int i = 0; i <= n; i++)
11            parent[i] = i;
12    }
13
14    int find(int v) {
15        if(v == parent[v])
16            return v;
17        return parent[v] = find(parent[v]);
18    }
19
20    void join(int a, int b) {
21        a = find(a);
22        b = find(b);
23        if(a != b) {
24            if(size[a] < size[b])
25                swap(a, b);
26
27            parent[b] = a;
28            size[a] += size[b];
29        }
30    }
31 };

```

9.2 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 <<
(j - 1))][j - 1]);
23             }
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 };

```

```

34     }
35 };
36
37
38 struct Sparse2d {
39     int n, m;
40     vector<vector<vi>> st;
41
42     Sparse2d(vector<vi> mat) {
43         n = mat.size();
44         m = mat[0].size();
45         int k = logv[min(n, m)];
46
47         st.assign(n+1, vector<vi>(m+1, vi(k+1)));
48         for(int i = 0; i < n; i++)
49             for(int j = 0; j < m; j++)
50                 st[i][j][0] = mat[i][j];
51
52         for(int j = 1; j <= k; j++) {
53             for(int x1 = 0; x1 < n; x1++) {
54                 for(int y1 = 0; y1 < m; y1++) {
55                     int delta = (1 << (j-1));
56                     if(x1+delta >= n or y1+delta >= m
57 ) continue;
58
59                     st[x1][y1][j] = st[x1][y1][j-1];
60                     st[x1][y1][j] = f(st[x1][y1][j],
61 st[x1+delta][y1][j-1]);
62                     st[x1][y1][j] = f(st[x1][y1][j],
63 st[x1][y1+delta][j-1]);
64                     st[x1][y1][j] = f(st[x1][y1][j],
65 st[x1+delta][y1+delta][j-1]);
66                 }
67             }
68         }
69
70 // so funciona para quadrados
71 int query(int x1, int y1, int x2, int y2) {
72     assert(x2-x1+1 == y2-y1+1);
73     int k = logv[x2-x1+1];
74     int delta = (1 << k);
75
76     int res = st[x1][y1][k];
77     res = f(res, st[x2 - delta+1][y1][k]);
78     res = f(res, st[x1][y2 - delta+1][k]);
79     res = f(res, st[x2 - delta+1][y2 - delta+1][k]);
80
81     return res;
82 }
83
84 int f(int a, int b) {
85     return a | b;
86 }
87
88 };

```

9.3 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((l<=l1 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)

```

```

13         break;
14         it = next(it);
15     }
16     for(auto [l1, r1, c1]: removed){
17         inter.erase({l1, r1, c1});
18         if(l1<l) inter.insert({l1, min(r1, l-1),
19 c1});
20         if(r<r1) inter.insert({max(l1, r+1), r1,
21 c1});
22     }
23     if(c != 0) inter.insert({l, r, c});
24     return removed;
25 }
26
27 ti query(int i){
28     if(inter.empty()) return {INF, INF, INF};
29     return *prev(inter.lower_bound({i+1, 0, 0}));
30 }
31 };

```

9.4 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){

```

```

49     lazy[no] = {x, y};
50     prop(1, r, no);
51     return;
52 }
53 int mid = (l + r) / 2;
54 update(a, b, x, y, l, mid, 2*no);
55 update(a, b, x + max((mid-max(l, a)+1)*y, 0LL), y
56 , mid+1, r, 2*no+1);
57 t[no] = merge(t[2*no], t[2*no+1]);
58 }

```

9.5 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 };

```

9.6 Segtree Recursive

```

1 int N;
2 vl t(4*MAX, 0);
3 vl v(MAX, 0);
4 vl 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 }

```

9.7 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(OLL, max(l.p, l.t+r.p));
19     ans.s = max(OLL, 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 };

```

9.8 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 }

```

9.9 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 };

```

9.10 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 }

```

9.11 Mergesorttree

```

1 struct ST { // indexado em 0, 0(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 };

```

9.12 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),
34     tree[root].val+1);
35     return idx++;
36 }
37
38 int query(int l, int r, int root1, int root2, int k){
39     while(l!=r)
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[
51 root1].l].val;
52         root1 = tree[root1].r;
53         root2 = tree[root2].r;
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     return 0;
91 }

```

9.13 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()){

```

```

17         if(z == end()) return 0;
18         return y->m == z->m && y->b <= z->b;
19     }
20     auto x = prev(y);
21     if(z == end()) return y->m == x->m && y->b <=
22     x->b;
23     return (ld)(x->b - y->b)*(z->m - y->m) >= (ld
24 ) (y->b - z->b)*(y->m - x->m);
25 }
26 void insert_line(ll m, ll b){ // min -> insert (-
27 m,-b) -> -eval()
28     auto y = insert({ m, b });
29     y->succ = [=]{ return next(y) == end() ? 0 :
30     &*next(y); };
31     if(bad(y)){ erase(y); return; }
32     while(next(y) != end() && bad(next(y))) erase
33     (next(y));
34     while(y != begin() && bad(prev(y))) erase(
35     prev(y));
36 }
37 ll eval(ll x){
38     auto l = *lower_bound((Line) { x, is_query })
39     ;
40     return l.m * x + l.b;
41 }
42 };

```

9.14 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 };

```

9.15 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 };

```

9.16 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.
7         top().ss);
8         in.push(mp(val, minimum));
9     }
10
11     ll pop() {
12         if(out.empty()) {
13             while(!in.empty()) {

```

```

13         ll val = in.top().ff;
14         in.pop();
15         ll minimum = out.empty() ? val : min(
16         val, out.top().ss);
17         out.push({val, minimum});
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 :
28         (ll)in.top().ss;
29     else
30         minimum = min((ll)in.top().ss, (ll)out.
31         top().ss);
32     return minimum;
33 }
34
35 ll size() {
36     return in.size() + out.size();
37 };

```