



Notebook - Maratona de Programação

Heladito??

Contents

1 Algoritmos	2	5	22
1.1 Cdq	2	5.1 2sat	22
1.2 Histogram Rectangle	2	5.2 Block Cut Tree	22
1.3 Mst Xor	3	5.3 Centroid Decomp	23
1.4 Ternary Search	4	5.4 Dfs Tree	23
2 DP	4	5.5 Dinic	23
2.1 Aliens	4	5.6 Dominator Tree	24
2.2 Divide Conquer	4	5.7 Ford	24
2.3 Dp Digitos	5	5.8 Hld Aresta	25
2.4 Knuth	5	5.9 Hld Vertice	25
2.5 Largest Ksubmatrix	5	5.10 Hungarian	26
2.6 Lis	5	5.11 Kosaraju	26
2.7 Partition Problem	5	5.12 Lca	26
3 ED	6	5.13 Mcmf	27
3.1 Bit	6	5.14 Mcmf Quirino	28
3.2 Bit Kth	6	6 Math	29
3.3 Cht	6	6.1 Berlekamp Massey	29
3.4 Color Update	6	6.2 Bigmod	29
3.5 Dsu Queue	7	6.3 Crt	29
3.6 Minqueue	7	6.4 Division Trick	30
3.7 Segtree Implicita	8	6.5 Fft Mod Tfg	30
3.8 Segtree Implicita Lazy	8	6.6 Fft Simple	30
3.9 Segtree Iterative	8		
3.10 Segtree Maxsubarray	9		
3.11 Segtree Pa	9		
3.12 Segtree Recursive	9		
3.13 Sparse Table	10		
3.14 Treap	10		
3.15 Virtual Tree	11		
4 Geometria	12		
4.1 2d	12		
4.2 3d	14		
4.3 Convex Hull	15		
4.4 Delaunay	15		
4.5 Halfplane Inter	16		
4.6 Inside Polygon	17		
4.7 Intersect Polygon	17		
4.8 Kdtree	17		
4.9 Lichao	18		
4.10 Linear Transformation	18		
4.11 Mindistpair	18		
4.12 Minkowski Sum	19		
4.13 Numintersectionline	19		
4.14 Polygon Cut Length	19		
4.15 Polygon Diameter	19		
4.16 Rotating Callipers	20		
4.17 Sort By Angle	20		
4.18 Tetrahedron Distance3d	20		
4.19 Voronoi	21		

6.7	Fft Tourist	31
6.8	Frac	32
6.9	Fwht	32
6.10	Gaussxor	32
6.11	Inverso Mult	32
6.12	Kitamasa	32
6.13	Linear Diophantine Equation	33
6.14	Matrix Exponentiation	33
6.15	Miller Habin	33
6.16	Mint	34
6.17	Mobius	34
6.18	Mulmod	34
6.19	Pollard Rho	34
6.20	Poly	35
6.21	Primitiveroot	36
6.22	Raiz Primitiva	36
6.23	Randommod	37
6.24	Totient	37
7	Misc	37
7.1	Bitwise	37
7.2	Ordered Set	37
7.3	Rand	37
7.4	Submask	37
7.5	Template	38
8	Numeric	38
8.1	Lagrange Interpolation	38
8.2	Newton Raphson	38
8.3	Simpson's Formula	38
9	Strings	39
9.1	Aho Corasick	39
9.2	Edit Distance	39
9.3	Eertree	39
9.4	Hash	39
9.5	Kmp	39
9.6	Lcs	40
9.7	Lcsubseq	40
9.8	Manacher	40
9.9	Suffix Array	40
9.10	Suffix Array Radix	41
9.11	Suffix Automaton	41
9.12	Z Func	42

1 Algoritmos

1.1 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            left = false;
42        }
43        bool operator<(point &o){
44            if(x != o.x) return x < o.x;
45            if(y != o.y) return y > o.y;
46            return z < o.z;
47        }
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
```

```
68            tmp.back().z = idz[tmp.back().z];
69            if(i<=mid)
70                tmp.back().left = true;
71        }
72
73        Segtree st(id);
74
75        sort(tmp.rbegin(), tmp.rend());
76
77        for(auto t: tmp){
78            if(t.left){
79                st.update(t.z, dp[t.id]);
80            }else{
81                dp[t.id] = max(dp[t.id], st.query(0, t.z
82                    -1)+1);
83            }
84        }
85        cdq(mid+1, r, a, dp);
86    }
87
88    int32_t main()
89    {sws;
90
91
92        int n; cin >> n;
93
94        vector<point> vet(n);
95        for(int i=0;i<n;i++){
96            cin >> vet[i].x >> vet[i].y >> vet[i].z;
97        }
98
99        sort(vet.begin(), vet.end());
100
101        for(int i=0;i<n;i++)
102            vet[i].id = i;
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        cout << ans << endl;
113
114        return 0;
115    }
116 }
```

1.2 Histogram Rectangle

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

```

19     for(int j=0;j<n;j++){
20         ll area = hist[j]*(right_smaller[j]-
left_smaller[j]-1);
21         ans = max(ans, area);
22     }
23     return ans;
24 }

```

1.3 Mst Xor

```

1 // omg why just 2 seconds
2 #include <bits/stdc++.h>
3 // #define int long long
4 #define ff first
5 #define ss second
6 #define ll long long
7 #define ld long double
8 #define pb push_back
9 #define eb emplace_back
10 #define pii pair<int, int>
11 #define pll pair<ll, ll>
12 #define ti tuple<int, int, int>
13 #define vi vector<int>
14 #define vl vector<ll>
15 #define vii vector<pii>
16 #define sws ios_base::sync_with_stdio(false);cin.tie(
NULL);cout.tie(NULL);
17 #define endl '\n'
18 #define teto(a, b) (((a)+(b)-1)/(b))
19 #define all(x) x.begin(), x.end()
20 #define forn(i, n) for(int i = 0; i < (int)n; i++)
21 #define forne(i, a, b) for(int i = a; i <= b; i++)
22 #define dbg(msg, var) cerr << msg << " " << var <<
endl;
23
24 using namespace std;
25
26 const int MAX = 6e6+10;
27 const ll MOD = 1e9+7;
28 const int INF = 0x3f3f3f3f;
29 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
30 const ld EPS = 1e-6;
31 const ld PI = acos(-1);
32
33 // End Template //
34
35 const int N = 2e5+10;
36
37 struct DSU {
38     int n;
39     map<int, int> parent;
40     map<int, vi> comp;
41
42     int find(int v) {
43         if(v==parent[v])
44             return v;
45         return parent[v]=find(parent[v]);
46     }
47
48     void join(int a, int b) {
49         a = find(a);
50         b = find(b);
51         if(a!=b) {
52             if((int)comp[a].size()<(int)comp[b].size
53             ())
54                 swap(a, b);
55             for(auto v: comp[b])
56                 comp[a].pb(v);
57             comp[b].clear();
58             parent[b]=a;
59         }
60

```

```

61     }
62 };
63
64 int trie[MAX][2];
65 set<int> idx[MAX];
66 int finish[MAX];
67 int nxt = 1;
68
69 void add(int s){
70     int node = 0;
71     for(int i=30;i>=0;i--){
72         bool c = (s & (1<<i));
73         if(trie[node][c] == 0)
74             node = trie[node][c] = nxt++;
75         else
76             node = trie[node][c];
77         finish[node]++;
78     }
79 }
80
81 void remove(int s){
82     int node = 0;
83     for(int i=30;i>=0;i--){
84         bool c = (s & (1<<i));
85         node = trie[node][c];
86         finish[node]--;
87     }
88 }
89
90 int min_xor(int s){
91     int node = 0;
92     int ans = 0;
93     for(int i=30;i>=0;i--){
94         bool c = (s & (1<<i));
95         if(finish[trie[node][c]] != 0)
96             node = trie[node][c];
97         else{
98             ans ^= 1 << i;
99             node = trie[node][!c];
100         }
101     }
102     return ans;
103 }
104
105
106 int32_t main()
107 {sws;
108
109     int n;
110     cin >> n;
111     vi x(n);
112     for(int i=0;i<n;i++)
113         cin >> x[i];
114
115     sort(x.begin(), x.end());
116     x.erase(unique(x.begin(), x.end()), x.end());
117     n = x.size();
118
119     DSU dsu;
120
121     ll mstsum = 0;
122
123     vi pais;
124     for(int i=0;i<n;i++){
125         add(x[i]);
126         dsu.parent[x[i]] = x[i];
127         dsu.comp[x[i]].pb(x[i]);
128         pais.pb(x[i]);
129     }
130
131     while((int)pais.size()!=1){
132         vector<ti> edges;
133         for(auto p: pais){

```

```

134     vi &nodes = dsu.comp[p];
135     // erase
136     for(auto u: nodes) remove(u);
137
138     // query
139     ti ed = {LLINF, 0, 0};
140     for(auto u: nodes){
141         int xr = min_xor(u);
142         ed = min(ed, {xr, u, xr^u});
143     }
144     edges.pb(ed);
145
146     // add back
147     for(auto u: nodes) add(u);
148 }
149
150 for(auto [xr, u, v]: edges){
151     if(dsu.find(u)!=dsu.find(v)){
152         // u, v -> mst
153         // cout << "mst = " << u << " " << v
154         mstsum += xr;
155         dsu.join(u, v);
156     }
157 }
158 vi pais2;
159 for(auto p: pais)
160     if(p==dsu.find(p))
161         pais2.pb(p);
162 swap(pais, pais2);
163 }
164
165 cout << mstsum << endl;
166
167 return 0;
168 }

```

1.4 Ternary Search

```

1 // Ternary
2 ld l = -1e4, r = 1e4;
3 int iter = 100;
4 while(iter--){
5     ld m1 = (2*l + r) / 3;
6     ld m2 = (l + 2*r) / 3;
7     if(check(m1) > check(m2))
8         l = m1;
9     else
10        r = m2;
11 }

```

2 DP

2.1 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 éat 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         for(int i=0;i<n;i++){
57             if('a'<= s[i] and s[i] <= 'z')
58                 lower[i+1] = 1;
59
60         cout << min(solve(lower),
61                     solve(upper)) << endl;
62
63         return 0;
64 }

```

2.2 Divide Conquer

```

1 ll cost(int l, int r) {
2     return ?;
3 }
4
5 void process(int l, int r, int optl, int optr) {
6     if (l > r) return;
7     int opt = optl;
8     int mid = (l + r) / 2;
9     for (int i=optl;i<=min(mid-1, optr);i++) {
10         if (dp[i] + cost(i+1, mid) < dp2[mid]) {
11             opt = i;
12             dp2[mid] = dp[i] + cost(i+1, mid);
13         }
14     }
15     process(l, mid-1, optl, opt);
16     process(mid+1, r, opt, optr);

```

```

17 }
18
19 int main() {
20     for (int i=0;i<n;i++) {
21         dp[i] = cost(0, i);
22         dp2[i] = LLINF;
23     }
24
25     for (int i=0;i<k-1;i++) {
26         process(0, n-1, 0, n-1);
27         swap(dp, dp2);
28         dp2.assign(N, LLINF);
29     }
30 }

```

2.3 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 }

```

2.4 Knuth

```

1 for (int i=1;i<=n;i++) {
2     opt[i][i] = i;
3     dp[i][i] = ?; // initialize
4 }
5 auto cost = [&](int l, int r) {
6     return ?;
7 };
8
9 for (int l=n-1;l>=1;l--) {
10    for (int r=l+1;r<=n;r++) {
11        ll ans = LLINF;
12        for (int k=opt[l][r-1]; k<=min(r-1, opt[l+1][
13            r]); k++) {
14            ll best = dp[l][k] + dp[k+1][r];
15            if (ans > best) {
16                ans = best;
17                opt[l][r] = k;
18            }
19            dp[l][r] = ans + cost(l, r);
20        }
21    }
22 }
23 cout << dp[1][n] << endl;

```

2.5 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][
17                    j-1]),
18                                dp[i-1][j-1]) + 1;
19            else dp[i][j] = 1;
20
21            result = max(result, dp[i][j]);
22        }
23    }
24    return result;
25 }

```

2.6 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
  -180208
13
14 vi LIS(const vi &elements){
15     auto compare = [&](int x, int y) {
16         return elements[x] < elements[y];
17     };
18     set< int, decltype(compare) > S(compare);
19
20     vi previous( elements.size(), -1 );
21     for(int i=0; i<int( elements.size() ); ++i){
22         auto it = S.insert(i).first;
23         if(it != S.begin())
24             previous[i] = *prev(it);
25         if(*it == i and next(it) != S.end())
26             S.erase(next(it));
27     }
28
29     vi answer;
30     answer.push_back( *S.rbegin() );
31     while ( previous[answer.back()] != -1 )
32         answer.push_back( previous[answer.back()] );
33     reverse( answer.begin(), answer.end() );
34     return answer;
35 }

```

2.7 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     return part[sum / 2][n];

```

3 ED

3.1 Bit

```

1 struct FT {
2     vi bit; // indexado em 1
3     int n;
4
5     FT(int sz) {
6         this->n = n;
7         bit.assign(n+1, 0);
8     }
9
10    int sum(int idx) {
11        int ret = 0;
12        for(; idx >= 1; 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 <= n; idx += idx & -idx)
23            bit[idx] += delta;
24    }
25 };

```

3.2 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 };

```

3.3 Cht

```

1 const ll is_query = -LLINF;
2 struct Line{
3     ll m, b;
4     mutable function<const Line*> succ;
5     bool operator<(const Line& rhs) const{
6         if(rhs.b != is_query) return m < rhs.m;
7         const Line* s = succ();
8         if(!s) return 0;
9         ll x = rhs.m;
10        return b - s->b < (s->m - m) * x;
11    }
12 };
13 struct Cht : public multiset<Line>{ // maintain max m
14     *x+b
15     bool bad(iterator y){
16         auto z = next(y);
17         if(y == begin()){
18             if(z == end()) return 0;
19             return y->m == z->m && y->b <= z->b;
20         }
21         auto x = prev(y);
22         if(z == end()) return y->m == x->m && y->b <=
23 x->b;
24         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)
25 (y->b - z->b)*(y->m - x->m);
26     }
27     void insert_line(ll m, ll b){ // min -> insert (-
28 m, -b) -> -eval()
29         auto y = insert({ m, b });
30         y->succ = [=]{ return next(y) == end() ? 0 :
31 &*next(y); };
32         if(bad(y)){ erase(y); return; }
33         while(next(y) != end() && bad(next(y))) erase
34 (next(y));
35         while(y != begin() && bad(prev(y))) erase(
36 prev(y));
37     }
38     ll eval(ll x){
39         auto l = *lower_bound((Line) { x, is_query })
40 ;
41         return l.m * x + l.b;
42     }
43 };

```

3.4 Color Update

```

1 #define ti tuple<int, int, int>
2 struct Color{
3     set<ti> inter; // l, r, color
4     vector<ti> update(int l, int r, int c){
5         if(inter.empty()){ inter.insert({l, r, c});
6         return {}; }
7         vector<ti> removed;
8         auto it = inter.lower_bound({l+1, 0, 0});
9         it = prev(it);
10        while(it != inter.end()){
11            auto [l1, r1, c1] = *it;
12            if((l1<=l and l1<=r) or (l1<=r1 and r1<=r)
13 or (l1<=l and r<=r1)){

```

```

12         removed.pb({l1, r1, c1});
13     }else if(l1 > r)
14         break;
15     it = next(it);
16 }
17 for(auto [l1, r1, c1]: removed){
18     inter.erase({l1, r1, c1});
19     if(l1<l) inter.insert({l1, min(r1, l-1),
20 c1});
21     if(r<r1) inter.insert({max(l1, r+1), r1,
22 c1});
23     }
24     if(c != 0) inter.insert({l, r, c});
25     return removed;
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 };

```

3.5 Dsu Queue

```

1 // DSU with queue rollback
2 // Normal DSU implementation with queue-like rollback
3 // find(x) - O(logn)
4 // join(a, b) - O(logn)
5 // pop() - (log^2n) amortized
6
7 struct event {
8     int a, b; // original operation
9     int fa, fb; // fa turned into fb's father
10    bool type; // 1 = inverted, 0 = normal
11 };
12
13 struct DSU {
14     int n;
15     vector<int> parent, size;
16     vector<event> st; int qnt_inv;
17     DSU(int n): n(n), parent(n), size(n, 1), qnt_inv
18 (0) {
19         for (int i=0;i<n;i++) parent[i] = i;
20     }
21     int find(int a) {
22         if (parent[a] == a) return a;
23         return find(parent[a]);
24     }
25     void join(int a, int b, bool inverted=false) {
26         int fa = find(a), fb = find(b);
27         if (size[fa] < size[fb]) swap(fa, fb);
28         st.push_back({a, b, fa, fb, inverted});
29         if (inverted == 1) qnt_inv++;
30         if (fa != fb) {
31             parent[fb] = fa;
32             size[fa] += size[fb];
33         }
34     }
35     void roll_back() {
36         auto [a, b, fa, fb, type] = st.back(); st.
37 pop_back();
38         if (type == 1) qnt_inv--;
39         if (fa != fb) {
40             parent[fb] = fb;
41             size[fa] -= size[fb];
42         }
43     }
44     void pop() {

```

```

47     auto lsb = [](int x) { return x&-x; };
48     if (qnt_inv == 0) { // invert all elements
49         vector<event> normal;
50         while (!st.empty()) {
51             normal.push_back(st.back());
52             roll_back();
53         }
54         for (auto [a, b, fa, fb, type]: normal) {
55             join(a, b, true);
56         }
57     } else if (st.back().type == 0) { // need to
58 reallocate
59         int qnt = lsb(qnt_inv);
60         vector<event> normal, inverted;
61         while (qnt > 0) {
62             event e = st.back();
63             if (e.type == 1) {
64                 inverted.push_back(e);
65                 qnt --;
66             } else {
67                 normal.push_back(e);
68             }
69             roll_back();
70         }
71         while (!normal.empty()) {
72             auto [a, b, fa, fb, type] = normal.
73 back(); normal.pop_back();
74             join(a, b);
75         }
76         while (!inverted.empty()) {
77             auto [a, b, fa, fb, type] = inverted.
78 back(); inverted.pop_back();
79             join(a, b, true);
80         }
81     }
82     // remove the last element
83     roll_back();
84 }
85 };

```

3.6 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({val, minimum});
9     }
10    ll pop() {
11        if(out.empty()) {
12            while(!in.empty()) {
13                ll val = in.top().ff;
14                in.pop();
15                ll minimum = out.empty() ? val : min(
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;

```



```

28         else
29             minimum = min((ll)in.top().ss, (ll)out.
30 top().ss);
31         return minimum;
32     }
33
34     ll size() {
35         return in.size() + out.size();
36     }
37 };

```

3.7 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
20 =1){
21     if(i==j){
22         t[no].val+=x;
23         return;
24     }
25     int meio = (i+j)/2;
26
27     if(pos<=meio){
28         if(t[no].l==0) t[no].l=idx++;
29         update(pos, x, i, meio, t[no].l);
30     }
31     else{
32         if(t[no].r==0) t[no].r=idx++;
33         update(pos, x, meio+1, j, t[no].r);
34     }
35
36     t[no].val=merge(t[t[no].l].val, t[t[no].r].val);
37 }
38
39 int query(int A, int B, int i=1, int j=N, int no=1){
40     if(B<i or j<A)
41         return 0;
42     if(A<=i and j<=B)
43         return t[no].val;
44
45     int mid = (i+j)/2;
46
47     int ans1 = 0, ansr = 0;
48
49     if(t[no].l!=0) ans1 = query(A, B, i, mid, t[no].l
50 );
51     if(t[no].r!=0) ansr = query(A, B, mid+1, j, t[no]
52 ].r);
53
54     return merge(ans1, ansr);
55 }

```

3.8 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
38 no=1){
39     prop(l, r, no);
40     if(a<=l and r<=b){
41         tree[no].lazy += x;
42         prop(l, r, no);
43         return;
44     }
45     if(r<a or b<l) return;
46     int m = (l+r)/2;
47     update(a, b, x, l, m, tree[no].l);
48     update(a, b, x, m+1, r, tree[no].r);
49
50     tree[no].val = merge(tree[tree[no].l].val, tree[
51 tree[no].r].val);
52 }
53
54 pll query(int a, int b, int l=0, int r=2*N, int no=1)
55 {
56     prop(l, r, no);
57     if(a<=l and r<=b) return tree[no].val;
58     if(r<a or b<l) return {INF, 0};
59     int m = (l+r)/2;
60     int left = tree[no].l, right = tree[no].r;
61
62     return tree[no].val = merge(query(a, b, l, m,
63 left),
64 query(a, b, m+1, r,
65 right));

```

3.9 Segtree Iterative

```

1 struct Segtree{
2     int n; vector<int> t;
3     Segtree(int n): n(n), t(2*n, 0) {}
4 }

```

```

5   int f(int a, int b) { return max(a, b); }
6
7   void build(){
8       for(int i=n-1; i>0; i--)
9           t[i] = f(t[i<<1], t[i<<1|1]);
10  }
11
12  int query(int l, int r) { // [l, r]
13      int resl = -INF, resr = -INF;
14      for(l+=n, r+=n+1; l<r; l>>=1, r>>=1) {
15          if(l&1) resl = f(resl, t[l++]);
16          if(r&1) resr = f(t[--r], resr);
17      }
18      return f(resl, resr);
19  }
20
21  void update(int p, int value) {
22      for(t[p+=n]=value; p >>= 1;)
23          t[p] = f(t[p<<1], t[p<<1|1]);
24  }
25 };

```

3.10 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 };

```

3.11 Segtree Pa

```

1 int N;
2 vl t(4*MAX, 0);
3 vl v(MAX, 0);
4 vector<pll> 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 pll sum(pll a, pll 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-1
31             +1)*y, y});
32     }
33     lazy[no] = {0,0};
34 }
35
36 ll query(int a, int b, int l=0, int r=N-1, int no=1){
37     prop(l, r, no);
38     if(r<a or b<l) return 0;
39     if(a<=l and r<=b) return t[no];
40     int mid = (l + r) / 2;
41     return merge(
42         query(a, b, l, mid, 2*no),
43         query(a, b, mid+1, r, 2*no+1)
44     );
45 }
46
47 void update(int a, int b, ll x, ll y, int l=0, int r=
48     N-1, int no=1){
49     prop(l, r, no);
50     if(r<a or b<l) return;
51     if(a<=l and r<=b){
52         lazy[no] = {x, y};
53         prop(l, r, no);
54         return;
55     }
56     int mid = (l + r) / 2;
57     update(a, b, x, y, l, mid, 2*no);
58     update(a, b, x + max((mid-max(l, a)+1)*y, OLL), y
59         , mid+1, r, 2*no+1);
60     t[no] = merge(t[2*no], t[2*no+1]);
61 }

```

3.12 Segtree Recursive

```

1 vector<ll> t(4*N, 0);
2 vector<ll> lazy(4*N, 0);
3
4 inline ll f(ll a, ll b) {
5     return a + b;
6 }
7
8 void build(vector<int> &v, int lx=0, int rx=N-1, int
9     x=1) {

```

```

9     if (lx == rx) { if (lx < v.size()) t[x] = v[lx];
10    return; }
11    int mid = (lx + rx) / 2;
12    build(v, lx, mid, 2*x);
13    build(v, mid+1, rx, 2*x+1);
14    t[x] = f(t[2*x], t[2*x+1]);
15 }
16 void prop(int lx, int rx, int x) {
17     if (lazy[x] != 0) {
18         t[x] += lazy[x] * (rx-lx+1);
19         if (lx != rx) {
20             lazy[2*x] += lazy[x];
21             lazy[2*x+1] += lazy[x];
22         }
23         lazy[x] = 0;
24     }
25 }
26
27 ll query(int l, int r, int lx=0, int rx=N-1, int x=1)
28 {
29     prop(lx, rx, x);
30     if (r < lx or rx < l) return 0;
31     if (l <= lx and rx <= r) return t[x];
32     int mid = (lx + rx) / 2;
33     return f(
34         query(l, r, lx, mid, 2*x),
35         query(l, r, mid+1, rx, 2*x+1)
36     );
37 }
38 void update(int l, int r, ll val, int lx=0, int rx=N-1, int x=1) {
39     prop(lx, rx, x);
40     if (r < lx or rx < l) return;
41     if (l <= lx and rx <= r) {
42         lazy[x] += val;
43         prop(lx, rx, x);
44         return;
45     }
46     int mid = (lx + rx) / 2;
47     update(l, r, val, lx, mid, 2*x);
48     update(l, r, val, mid+1, rx, 2*x+1);
49     t[x] = f(t[2*x], t[2*x+1]);
50 }

```

3.13 Sparse Table

```

1 int logv[N+1];
2 void make_log() {
3     logv[1] = 0; // pre-computar tabela de log
4     for (int i = 2; i <= N; i++)
5         logv[i] = logv[i/2] + 1;
6 }
7 struct Sparse {
8     int n;
9     vector<vector<int>> st;
10
11     Sparse(vector<int>& v) {
12         n = v.size();
13         int k = logv[n];
14         st.assign(n+1, vector<int>(k+1, 0));
15
16         for (int i=0; i<n; i++) {
17             st[i][0] = v[i];
18         }
19
20         for (int j = 1; j <= k; j++) {
21             for (int i = 0; i + (1 << j) <= n; i++) {
22                 st[i][j] = f(st[i][j-1], st[i + (1 <<
23                 (j-1))][j-1]);
24             }
25         }
26     }

```

```

25 }
26
27 int f(int a, int b) {
28     return min(a, b);
29 }
30
31 int query(int l, int r) {
32     int k = logv[r-l+1];
33     return f(st[l][k], st[r - (1 << k) + 1][k]);
34 }
35 };
36
37 struct Sparse2d {
38     int n, m;
39     vector<vector<vector<int>>> st;
40
41     Sparse2d(vector<vector<int>> mat) {
42         n = mat.size();
43         m = mat[0].size();
44         int k = logv[min(n, m)];
45
46         st.assign(n+1, vector<vector<int>>(m+1,
47         vector<int>(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+delta][y1+delta][j-1]);
64                 }
65             }
66         }
67
68         // so funciona para quadrados
69         int query(int x1, int y1, int x2, int y2) {
70             assert(x2-x1+1 == y2-y1+1);
71             int k = logv[x2-x1+1];
72             int delta = (1 << k);
73
74             int res = st[x1][y1][k];
75             res = f(res, st[x2 - delta+1][y1][k]);
76             res = f(res, st[x1][y2 - delta+1][k]);
77             res = f(res, st[x2 - delta+1][y2 - delta+1][k]);
78         }
79         return res;
80     }
81
82     int f(int a, int b) {
83         return a | b;
84     }
85 };

```

3.14 Treap

```

1 mt19937 rng(chrono::steady_clock::now().
2     time_since_epoch().count()); // mt19937_64
3 uniform_int_distribution<int> distribution(1, INF);

```

```

4  const int N = 2e5+10;
5  int nxt = 0;
6  int X[N], Y[N], L[N], R[N], sz[N], idx[N];
7  bool flip[N];
8
9  //! Call this before anything else
10 void build() {
11     iota(Y+1, Y+N, 1);
12     shuffle(Y+1, Y+N, rng); // rng :: mt19937
13 }
14
15 int new_node(int x, int id) {
16     int u = ++nxt;
17     idx[u] = id;
18     sz[u] = 1;
19     X[u] = x;
20     return u;
21 }
22
23 void push(int u) { // also known as unlaze
24     if(!u) return;
25     if (flip[u]) {
26         flip[u] = false;
27         flip[L[u]] ^= 1;
28         flip[R[u]] ^= 1;
29         swap(L[u], R[u]);
30     }
31 }
32
33 void pull(int u) { // also known as fix
34     if (!u) return;
35     sz[u] = sz[L[u]] + 1 + sz[R[u]];
36 }
37
38 // root = merge(l, r);
39 int merge(int l, int r) {
40     push(l); push(r);
41     int u;
42     if (!l || !r) {
43         u = l ? l : r;
44     } else if (Y[l] < Y[r]) {
45         u = l;
46         R[u] = merge(R[u], r);
47     } else {
48         u = r;
49         L[u] = merge(l, L[u]);
50     }
51     pull(u);
52     return u;
53 }
54
55 // (s elements, N - s elements)
56 pair<int, int> splitsz(int u, int s) {
57     if (!u) return {0, 0};
58     push(u);
59     if (sz[L[u]] >= s) {
60         auto [l, r] = splitsz(L[u], s);
61         L[u] = r;
62         pull(u);
63         return {l, u};
64     } else {
65         auto [l, r] = splitsz(R[u], s - sz[L[u]] - 1);
66         R[u] = l;
67         pull(u);
68         return {u, r};
69     }
70 }
71
72 // (<= x, > x)
73 pair<int, int> splitval(int u, int x) {
74     if (!u) return {0, 0};
75     push(u);

```

```

76     if (X[u] > x) {
77         auto [l, r] = splitval(L[u], x);
78         L[u] = r;
79         pull(u);
80         return {l, u};
81     } else {
82         auto [l, r] = splitval(R[u], x);
83         R[u] = l;
84         pull(u);
85         return {u, r};
86     }
87 }
88
89 int insert(int u, int node) {
90     push(u);
91     if (!u) return node;
92     if (Y[node] < Y[u]) {
93         tie(L[node], R[node]) = splitval(u, X[node]);
94         u = node;
95     }
96     else if (X[node] < X[u]) L[u] = insert(L[u], node);
97     else R[u] = insert(R[u], node);
98     pull(u);
99     return u;
100 }
101
102 int find(int u, int x) {
103     return u == 0 ? 0 :
104         x == X[u] ? u :
105         x < X[u] ? find(L[u], x) :
106             find(R[u], x);
107 }
108
109 void free(int u) { /* node u can be deleted, maybe
110     put in a pool of free IDs */ }
111
112 int erase(int u, int key) {
113     push(u);
114     if (!u) return 0;
115     if (X[u] == key) {
116         int v = merge(L[u], R[u]);
117         free(u);
118         u = v;
119     } else u = erase(key < X[u] ? L[u] : R[u], key);
120     pull(u);
121     return u;

```

3.15 Virtual Tree

```

1  bool initialized = false;
2  int original_root = 1;
3  const int E = 2 * N;
4  vector<int> vt[N]; // virtual tree edges
5  int in[N], out[N], T, t[E<<1];
6  void dfs_time(int u, int p = 0) {
7      in[u] = ++T;
8      t[T + E] = u;
9      for (int v : g[u]) if (v != p) {
10         dfs_time(v, u);
11         t[++T + E] = u;
12     }
13     out[u] = T;
14 }
15
16 int take(int u, int v) { return in[u] < in[v] ? u : v; }
17
18 bool cmp_in(int u, int v) { return in[u] < in[v]; }
19 void build_st() {
20     in[0] = 0x3f3f3f3f;
21     for (int i = E-1; i > 0; i--)
22         t[i] = take(t[i<<1], t[i<<1+1]);

```

```

22 }
23
24 int query(int l, int r) {
25     int ans = 0;
26     for (l+=E, r+=E; l < r; l>>=1, r>>=1) {
27         if (l&1) ans = take(ans, t[l++]);
28         if (r&1) ans = take(ans, t[--r]);
29     }
30     return ans;
31 }
32
33 int get_lca(int u, int v) {
34     if (in[u] > in[v]) swap(u, v);
35     return query(in[u], out[v]+1);
36 }
37
38 int covers(int u, int v) { // does u cover v?
39     return in[u] <= in[v] && out[u] >= out[v];
40 }
41
42 int build_vt(vector<int>& vnodes) {
43     assert(initialized);
44
45     sort(all(vnodes), cmp_in);
46     int n = vnodes.size();
47     for (int i = 0; i < n-1; i++) {
48         int u = vnodes[i], v = vnodes[i+1];
49         vnodes.push_back(get_lca(u, v));
50     }
51     sort(all(vnodes), cmp_in);
52     vnodes.erase(unique(all(vnodes)), vnodes.end());
53
54     for (int u : vnodes)
55         vt[u].clear();
56
57     stack<int> s;
58     for (int u : vnodes) {
59         while (!s.empty() && !covers(s.top(), u))
60             s.pop();
61         if (!s.empty()) vt[s.top()].push_back(u);
62         s.push(u);
63     }
64     return vnodes[0]; // root
65 }
66
67 void initialize() {
68     initialized = true;
69     dfs_time(original_root);
70     build_st();
71 }

```

4 Geometria

4.1 2d

```

1 #define vp vector<point>
2 #define ld long double
3 const ld EPS = 1e-6;
4 const ld PI = acos(-1);
5
6 typedef ld T;
7 bool eq(T a, T b){ return abs(a - b) <= EPS; }
8
9 struct point{
10     T x, y;
11     int id;
12     point(T x=0, T y=0): x(x), y(y){}
13
14     point operator+(const point &o) const{ return {x
15         + o.x, y + o.y}; }
16     point operator-(const point &o) const{ return {x
17         - o.x, y - o.y}; }

```

```

16     point operator*(T t) const{ return {x * t, y * t
17 }; }
18     point operator/(T t) const{ return {x / t, y / t
19 }; }
20     T operator*(const point &o) const{ return x * o.x
21         + y * o.y; }
22     T operator^(const point &o) const{ return x * o.y
23         - y * o.x; }
24     bool operator<(const point &o) const{
25         return (eq(x, o.x) ? y < o.y : x < o.x);
26     }
27     bool operator==(const point &o) const{
28         return eq(x, o.x) and eq(y, o.y);
29     }
30     friend ostream& operator<<(ostream& os, point p)
31     {
32         return os << "(" << p.x << "," << p.y << ")";
33     }
34 };
35
36 int ccw(point a, point b, point e){ // -1=dir; 0=
37     collinear; 1=esq;
38     T tmp = (b-a) ^ (e-a); // vector from a to b
39     return (tmp > EPS) - (tmp < -EPS);
40 }
41
42 ld norm(point a){ // Modulo
43     return sqrt(a * a);
44 }
45
46 T norm2(point a){
47     return a * a;
48 }
49
50 bool nulo(point a){
51     return (eq(a.x, 0) and eq(a.y, 0));
52 }
53
54 point rotccw(point p, ld a){
55     // a = PI*a/180; // graus
56     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
57         +p.x*sin(a)));
58 }
59
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     return abs(ret/2);
95 }
96
97 ld areaT(point &a, point &b, point &c){

```

```

79     return abs((b-a)^(c-a))/2.0;
80 }
81
82 point center(vp &A){
83     point c = point();
84     int len = A.size();
85     for(int i=0;i<len;i++)
86         c=c+A[i];
87     return c/len;
88 }
89
90 point forca_mod(point p, ld m){
91     ld cm = norm(p);
92     if(cm<EPS) return point();
93     return point(p.x*m/cm,p.y*m/cm);
94 }
95
96 ld param(point a, point b, point v){
97     // v = t*(b-a) + a // return t;
98     // assert(line(a, b).inside_seg(v));
99     return ((v-a) * (b-a)) / ((b-a) * (b-a));
100 }
101
102 bool simetric(vp &a){ //ordered
103     int n = a.size();
104     point c = center(a);
105     if(n&1) return false;
106     for(int i=0;i<n/2;i++)
107         if(ccw(a[i], a[i+n/2], c) != 0)
108             return false;
109     return true;
110 }
111
112 point mirror(point m1, point m2, point p){
113     // mirror point p around segment m1m2
114     point seg = m2-m1;
115     ld t0 = ((p-m1)*seg) / (seg*seg);
116     point ort = m1 + seg*t0;
117     point pm = ort-(p-ort);
118     return pm;
119 }
120
121
122 ////////////////
123 // Line //
124 ////////////////
125
126 struct line{
127     point p1, p2;
128     T a, b, c; // ax+by+c = 0;
129     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
130     line(point p1=0, point p2=0): p1(p1), p2(p2){
131         a = p1.y - p2.y;
132         b = p2.x - p1.x;
133         c = p1 ^ p2;
134     }
135     line(T a=0, T b=0, T c=0): a(a), b(b), c(c){
136         // Gera os pontos p1 p2 dados os coeficientes
137         // isso aqui eh um lixo mas quebra um galho
138         kkkkkk
139         if(b==0){
140             p1 = point(1, -c/a);
141             p2 = point(0, -c/a);
142         }else{
143             p1 = point(1, (-c-a*1)/b);
144             p2 = point(0, -c/b);
145         }
146     }
147     T eval(point p){
148         return a*p.x+b*p.y+c;
149     }
150     bool inside(point p){

```

```

151         return eq(eval(p), 0);
152     }
153     point normal(){
154         return point(a, b);
155     }
156
157     bool inside_seg(point p){
158         return (
159             ((p1-p) ^ (p2-p)) == 0 and
160             ((p1-p) * (p2-p)) <= 0
161         );
162     }
163 };
164
165 // be careful with precision error
166 vp inter_line(line l1, line l2){
167     ld det = l1.a*l2.b - l1.b*l2.a;
168     if(det==0) return {};
169     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
170     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
171     return {point(x, y)};
172 }
173
174 // segments not collinear
175 vp inter_seg(line l1, line l2){
176     vp ans = inter_line(l1, l2);
177     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
178         inside_seg(ans[0]))
179         return {};
180     return ans;
181 }
182 bool seg_has_inter(line l1, line l2){
183     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.
184         p2, l2.p2) < 0 and
185         ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.
186         p2, l1.p2) < 0;
187 }
188 ld dist_seg(point p, point a, point b){ // point -
189     seg
190     if((p-a)*(b-a) < EPS) return norm(p-a);
191     if((p-b)*(a-b) < EPS) return norm(p-b);
192     return abs((p-a)^(b-a)) / norm(b-a);
193 }
194
195 ld dist_line(point p, line l){ // point - line
196     return abs(l.eval(p))/sqrt(1.a*1.a + 1.b*1.b);
197 }
198 line bisector(point a, point b){
199     point d = (b-a)*2;
200     return line(d.x, d.y, a*a - b*b);
201 }
202 line perpendicular(line l, point p){ // passes
203     through p
204     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
205 }
206
207 // Circle //
208 // Circle //
209
210 struct circle{
211     point c; T r;
212     circle() : c(0, 0), r(0){}
213     circle(const point o) : c(o), r(0){}
214     circle(const point a, const point b){
215         c = (a+b)/2;
216         r = norm(a-c);
217     }
218 }

```

```

219 circle(const point a, const point b, const point cc){
220     assert(ccw(a, b, cc) != 0);
221     c = inter_line(bisector(a, b), bisector(b, c))[0];
222     r = norm(a-c);
223 }
224 bool inside(const point &a) const{
225     return norm(a - c) <= r + EPS;
226 }
227 };
228
229 pair<point, point> tangent_points(circle cr, point p){
230     {
231         ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
232         point p1 = rotccw(cr.c-p, -theta);
233         point p2 = rotccw(cr.c-p, theta);
234         assert(d1 >= cr.r);
235         p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
236         p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
237     }
238     return {p1, p2};
239 }
240 circle incircle(point p1, point p2, point p3){
241     ld m1 = norm(p2-p3);
242     ld m2 = norm(p1-p3);
243     ld m3 = norm(p1-p2);
244     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
245     ld s = 0.5*(m1+m2+m3);
246     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
247     return circle(c, r);
248 }
249
250 circle circumcircle(point a, point b, point c) {
251     circle ans;
252     point u = point((b-a).y, -(b-a).x);
253     point v = point((c-a).y, -(c-a).x);
254     point n = (c-b)*0.5;
255     ld t = (u^v)/(v^u);
256     ans.c = ((a+c)*0.5) + (v*t);
257     ans.r = norm(ans.c-a);
258     return ans;
259 }
260
261 vp inter_circle_line(circle C, line L){
262     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)*(ab) / (ab*ab));
263     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s / (ab*ab);
264     if (h2 < -EPS) return {};
265     if (eq(h2, 0)) return {p};
266     point h = (ab/norm(ab)) * sqrt(h2);
267     return {p - h, p + h};
268 }
269
270 vp inter_circle(circle c1, circle c2){
271     if (c1.c == c2.c) { assert(c1.r != c2.r); return {}; }
272     point vec = c2.c - c1.c;
273     ld d2 = vec * vec, sum = c1.r + c2.r, dif = c1.r - c2.r;
274     ld p = (d2 + c1.r * c1.r - c2.r * c2.r) / (2 * d2);
275     ld h2 = c1.r * c1.r - p * p * d2;
276     if (sum * sum < d2 or dif * dif > d2) return {};
277     point mid = c1.c + vec * p, per = point(-vec.y, vec.x) * sqrt(fmax(0, h2) / d2);
278     if (eq(per.x, 0) and eq(per.y, 0)) return {mid};
279     return {mid + per, mid - per};
280 }
281
282 // minimum circle cover O(n) amortizado

```

```

283 circle min_circle_cover(vp v){
284     random_shuffle(v.begin(), v.end());
285     circle ans;
286     int n = v.size();
287     for(int i=0;i<n;i++) if(!ans.inside(v[i])){
288         ans = circle(v[i]);
289         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
290             ans = circle(v[i], v[j]);
291             for(int k=0;k<j;k++) if(!ans.inside(v[k])){
292                 ans = circle(v[i], v[j], v[k]);
293             }
294         }
295     }
296     return ans;
297 }

```

4.2 3d

```

1 // typedef ll cod;
2 // bool eq(cod a, cod b){ return (a==b); }
3
4 const ld EPS = 1e-6;
5 #define vp vector<point>
6 typedef ld cod;
7 bool eq(cod a, cod b){ return fabs(a - b) <= EPS; }
8
9 struct point
10 {
11     cod x, y, z;
12     point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z) {}
13
14     point operator+(const point &o) const {
15         return {x+o.x, y+o.y, z+o.z};
16     }
17     point operator-(const point &o) const {
18         return {x-o.x, y-o.y, z-o.z};
19     }
20     point operator*(cod t) const {
21         return {x*t, y*t, z*t};
22     }
23     point operator/(cod t) const {
24         return {x/t, y/t, z/t};
25     }
26     bool operator==(const point &o) const {
27         return eq(x, o.x) and eq(y, o.y) and eq(z, o.z);
28     }
29     cod operator*(const point &o) const { // dot
30         return x*o.x + y*o.y + z*o.z;
31     }
32     point operator^(const point &o) const { // cross
33         return point(y*o.z - z*o.y,
34                     z*o.x - x*o.z,
35                     x*o.y - y*o.x);
36     }
37 };
38
39 ld norm(point a) { // Modulo
40     return sqrt(a * a);
41 }
42 cod norm2(point a) {
43     return a * a;
44 }
45 bool nulo(point a) {
46     return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0));
47 }
48 ld proj(point a, point b) { // a sobre b
49     return (a*b)/norm(b);
50 }
51 ld angle(point a, point b) { // em radianos

```

```

52     return acos((a*b) / norm(a) / norm(b));
53 }
54
55 cod triple(point a, point b, point c) {
56     return (a * (b^c)); // Area do paralelepipedo
57 }
58
59 point normilize(point a) {
60     return a/norm(a);
61 }
62
63 struct plane {
64     cod a, b, c, d;
65     point p1, p2, p3;
66     plane(point p1=0, point p2=0, point p3=0): p1(p1)
67     , p2(p2), p3(p3) {
68         point aux = (p1-p3)^(p2-p3);
69         a = aux.x; b = aux.y; c = aux.z;
70         d = -a*p1.x - b*p1.y - c*p1.z;
71     }
72     plane(point p, point normal) {
73         normal = normilize(normal);
74         a = normal.x; b = normal.y; c = normal.z;
75         d = -(p*normal);
76     }
77     // ax+by+cz+d = 0;
78     cod eval(point &p) {
79         return a*p.x + b*p.y + c*p.z + d;
80     }
81 };
82
83 cod dist(plane pl, point p) {
84     return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d
85 ) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
86 }
87
88 point rotate(point v, point k, ld theta) {
89     // Rotaciona o vetor v theta graus em torno do
90     eixo k
91     // theta *= PI/180; // graus
92     return (
93         v*cos(theta)) +
94         ((k^v)*sin(theta)) +
95         (k*(k^v))*(1-cos(theta))
96 );
97
98 // 3d line inter / mindistance
99 cod d(point p1, point p2, point p3, point p4) {
100     return (p2-p1) * (p4-p3);
101 }
102
103 vector<point> inter3d(point p1, point p2, point p3,
104 point p4) {
105     cod mua = ( d(p1, p3, p4, p3) * d(p4, p3, p2, p1)
106 - d(p1, p3, p2, p1) * d(p4, p3, p4, p3) )
107 / ( d(p2, p1, p2, p1) * d(p4, p3, p4, p3)
108 - d(p4, p3, p2, p1) * d(p4, p3, p2, p1) );
109     cod mub = ( d(p1, p3, p4, p3) + mua * d(p4, p3,
110 p2, p1) ) / d(p4, p3, p4, p3);
111     point pa = p1 + (p2-p1) * mua;
112     point pb = p3 + (p4-p3) * mub;
113     if (pa == pb) return {pa};
114     return {};
115 }

```

4.3 Convex Hull

```

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

```

4.4 Delaunay

```

1 cod areaT2(point &a, point &b, point &c){
2     return abs((b-a)^(c-a));
3 }
4
5 typedef struct QuadEdge* Q;
6 struct QuadEdge {
7     int id;
8     point o;
9     Q rot, nxt;
10    bool used;
11
12    QuadEdge(int id_ = -1, point o_ = point(INF, INF)
13    ) :
14        id(id_), o(o_), rot(nullptr), nxt(nullptr),
15        used(false) {}
16
17    Q rev() const { return rot->rot; }
18    Q next() const { return rot->nxt; }
19    Q prev() const { return rot->nxt->rot; }
20    point dest() const { return rev()->o; }
21 };
22
23 Q edge(point from, point to, int id_from, int id_to)
24 {
25     Q e1 = new QuadEdge(id_from, from);
26     Q e2 = new QuadEdge(id_to, to);
27     Q e3 = new QuadEdge;
28     Q e4 = new QuadEdge;
29     tie(e1->rot, e2->rot, e3->rot, e4->rot) = {e3, e4
30     , e2, e1};
31     tie(e1->nxt, e2->nxt, e3->nxt, e4->nxt) = {e1, e2
32     , e4, e3};
33     return e1;
34 }
35
36 void splice(Q a, Q b) {
37     swap(a->nxt->rot->nxt, b->nxt->rot->nxt);
38     swap(a->nxt, b->nxt);
39 }
40
41 void del_edge(Q& e, Q ne) { // delete e and assign e
42     <- ne
43     splice(e, e->prev());
44     splice(e->rev(), e->rev()->prev());
45     delete e->rev()->rot, delete e->rev();
46     delete e->rot; delete e;
47     e = ne;
48 }
49
50 Q conn(Q a, Q b) {
51     Q e = edge(a->dest(), b->o, a->rev()->id, b->id);
52     splice(e, a->rev()->prev());
53     splice(e->rev(), b);
54     return e;
55 }

```



```

49 }
50
51 bool in_c(point a, point b, point c, point p) { // p
    ta na circunf. (a, b, c) ?
52 __int128 p2 = p*p, A = a*a - p2, B = b*b - p2, C
    = c*c - p2;
53 return areaT2(p, a, b) * C + areaT2(p, b, c) * A
    + areaT2(p, c, a) * B > 0;
54 }
55
56 pair<Q, Q> build_tr(vector<point>& p, int l, int r) {
57     if (r-l+1 <= 3) {
58         Q a = edge(p[l], p[l+1], l, l+1), b = edge(p[
59             l+1], p[r], l+1, r);
60         if (r-l+1 == 2) return {a, a->rev()};
61         splice(a->rev(), b);
62         ll ar = areaT2(p[l], p[l+1], p[r]);
63         Q c = ar ? conn(b, a) : 0;
64         if (ar >= 0) return {a, b->rev()};
65         return {c->rev(), c};
66     }
67     int m = (l+r)/2;
68     auto [la, ra] = build_tr(p, l, m);
69     auto [lb, rb] = build_tr(p, m+1, r);
70     while (true) {
71         if (ccw(lb->o, ra->o, ra->dest())) ra = ra->
72             rev()->prev();
73         else if (ccw(lb->o, ra->o, lb->dest())) lb =
74             lb->rev()->next();
75         else break;
76     }
77     Q b = conn(lb->rev(), ra);
78     auto valid = [&](Q e) { return ccw(e->dest(), b->
79         dest(), b->o); };
80     if (ra->o == la->o) la = b->rev();
81     if (lb->o == rb->o) rb = b;
82     while (true) {
83         Q L = b->rev()->next();
84         if (valid(L)) while (in_c(b->dest(), b->o, L
85             ->dest(), L->next()->dest()))
86             del_edge(L, L->next());
87         Q R = b->prev();
88         if (valid(R)) while (in_c(b->dest(), b->o, R
89             ->dest(), R->prev()->dest()))
90             del_edge(R, R->prev());
91         if (!valid(L) and !valid(R)) break;
92         if (!valid(L) or (valid(R) and in_c(L->dest()
93             , L->o, R->o, R->dest())))
94             b = conn(R, b->rev());
95         else b = conn(b->rev(), L->rev());
96     }
97     return {la, rb};
98 }
99
100 vector<vector<int>>> delaunay(vp v) {
101     int n = v.size();
102     auto tmp = v;
103     vector<int> idx(n);
104     iota(idx.begin(), idx.end(), 0);
105     sort(idx.begin(), idx.end(), [&](int l, int r) {
106         return v[l] < v[r]; });
107     for (int i = 0; i < n; i++) v[i] = tmp[idx[i]];
108     assert(unique(v.begin(), v.end()) == v.end());
109     vector<vector<int>>> g(n);
110     bool col = true;
111     for (int i = 2; i < n; i++) if (areaT2(v[i], v[i
112         -1], v[i-2])) col = false;
113     if (col) {
114         for (int i = 1; i < n; i++)
115             g[idx[i-1]].push_back(idx[i]), g[idx[i]].
116             push_back(idx[i-1]);
117         return g;
118     }
119 }
120
121 Q e = build_tr(v, 0, n-1).first;
122 vector<Q> edg = {e};
123 for (int i = 0; i < edg.size(); i = edg[i++]) {
124     for (Q at = e; !at->used; at = at->next()) {
125         at->used = true;
126         g[idx[at->id]].push_back(idx[at->rev()->
127             id]);
128         edg.push_back(at->rev());
129     }
130 }
131 return g;
132
133 4.5 Halfplane Inter
134
135 struct Halfplane {
136     point p, pq;
137     ld angle;
138     Halfplane() {}
139     Halfplane(const point &a, const point &b) : p(a),
140         pq(b - a) {
141         angle = atan2l(pq.y, pq.x);
142     }
143
144     bool out(const point &r) { return (pq ^ (r - p))
145         < -EPS; }
146     bool operator<(const Halfplane &e) const { return
147         angle < e.angle; }
148
149     friend point inter(const Halfplane &s, const
150         Halfplane &t) {
151         ld alpha = ((t.p - s.p) ^ t.pq) / (s.pq ^ t.
152             pq);
153         return s.p + (s.pq * alpha);
154     }
155 }
156
157 vp hp_intersect(vector<Halfplane> &H) {
158     point box[4] = {
159         point(LLINF, LLINF),
160         point(-LLINF, LLINF),
161         point(-LLINF, -LLINF),
162         point(LLINF, -LLINF)
163     };
164     for(int i = 0; i < 4; i++) {
165         Halfplane aux(box[i], box[(i+1) % 4]);
166         H.push_back(aux);
167     }
168
169     sort(H.begin(), H.end());
170     deque<Halfplane> dq;
171     int len = 0;
172     for(int i = 0; i < (int)H.size(); i++) {
173         while (len > 1 && H[i].out(inter(dq[len-1],
174             dq[len-2]))) {
175             dq.pop_back();
176             --len;
177         }
178         while (len > 1 && H[i].out(inter(dq[0], dq
179             [1]))) {
180             dq.pop_front();
181             --len;
182         }
183         if (len > 0 && fabs1((H[i].pq ^ dq[len-1].pq)
184             ) < EPS) {
185             if ((H[i].pq * dq[len-1].pq) < 0.0)
186                 return vp();
187         }
188     }
189 }

```

```

51         if (H[i].out(dq[len-1].p)) {
52             dq.pop_back();
53             --len;
54         }
55         else continue;
56     }
57
58     dq.push_back(H[i]);
59     ++len;
60 }
61
62 while (len > 2 && dq[0].out(inter(dq[len-1], dq[
len-2]))) {
63     dq.pop_back();
64     --len;
65 }
66
67 while (len > 2 && dq[len-1].out(inter(dq[0], dq
[1]))) {
68     dq.pop_front();
69     --len;
70 }
71
72 if (len < 3) return vp();
73
74 vp ret(len);
75 for(int i = 0; i+1 < len; i++) {
76     ret[i] = inter(dq[i], dq[i+1]);
77 }
78 ret.back() = inter(dq[len-1], dq[0]);
79 return ret;
80 }
81
82 // O(n3)
83 vp half_plane_intersect(vector<line> &v){
84     vp ret;
85     int n = v.size();
86     for(int i=0; i<n; i++){
87         for(int j=i+1; j<n; j++){
88             point crs = inter(v[i], v[j]);
89             if(crs.x == INF) continue;
90             bool bad = 0;
91             for(int k=0; k<n; k++){
92                 if(v[k].eval(crs) < -EPS){
93                     bad = 1;
94                     break;
95                 }
96             }
97             if(!bad) ret.push_back(crs);
98         }
99     }
100     return ret;
101 }

```

4.6 Inside Polygon

```

1 // Convex O(logn)
2
3 bool insideT(point a, point b, point c, point e){
4     int x = ccw(a, b, e);
5     int y = ccw(b, c, e);
6     int z = ccw(c, a, e);
7     return !((x==1 or y==1 or z==1) and (x==-1 or y
==-1 or z==-1));
8 }
9
10 bool inside(vp &p, point e){ // ccw
11     int l=2, r=(int)p.size()-1;
12     while(l<r){
13         int mid = (l+r)/2;
14         if(ccw(p[0], p[mid], e) == 1)
15             l=mid+1;
16         else{

```

```

17             r=mid;
18         }
19     }
20     // bordo
21     // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)
==0) return false;
22     // if(r==2 and ccw(p[0], p[1], e)==0) return
false;
23     // if(ccw(p[r], p[r-1], e)==0) return false;
24     return insideT(p[0], p[r-1], p[r], e);
25 }
26
27 // Any O(n)
28
29 int inside(vp &p, point pp){
30     // 1 - inside / 0 - boundary / -1 - outside
31     int n = p.size();
32     for(int i=0; i<n; i++){
33         int j = (i+1)%n;
34         if(line({p[i], p[j]}).inside_seg(pp))
35             return 0;
36     }
37
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 }

```

4.7 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 }

```

4.8 Kdtree

```

1 bool on_x(const point& a, const point& b) { return a.
x < b.x; }
2 bool on_y(const point& a, const point& b) { return a.
y < b.y; }
3 bool on_z(const point& a, const point& b) { return a.
z < b.z; }
4
5 struct Node {
6     point pt; // if this is a leaf, the single point
in it
7     cod x0 = LLINF, x1 = -LLINF, y0 = LLINF, y1 = -
LLINF, z0 = LLINF, z1 = -LLINF; // bounds
8     Node *first = 0, *second = 0;
9 }

```

```

10     cod distance(const point &p) { // min squared
distance to a point
11         cod x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x)
;
12         cod y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y)
;
13         cod z = (p.z < z0 ? z0 : p.z > z1 ? z1 : p.z)
;
14         return norm(point(x,y,z) - p);
15     }
16
Node(vp&& p) : pt(p[0]) {
17     for (point pi : p) {
18         x0 = min(x0, pi.x); x1 = max(x1, pi.x);
19         y0 = min(y0, pi.y); y1 = max(y1, pi.y);
20         z0 = min(z0, pi.z); z1 = max(z1, pi.z);
21     }
22     if (p.size() > 1) {
23         auto cmp = (x1-x0 >= y1-y0 and x1-x0 >=
z1-z0 ? on_x : (y1-y0 >= z1-z0 ? on_y : on_z));
24         sort(p.begin(), p.end(), cmp);
25         // divide by taking half the array for
each child (not
26         // best performance with many duplicates
in the middle)
27         int half = p.size() / 2;
28         first = new Node({p.begin(), p.begin() +
half});
29         second = new Node({p.begin() + half, p.
end()});
30     }
31 }
32
33 };
34
35 struct KDTree {
36     Node* root;
37     KDTree(const vp& p) : root(new Node({p.begin(), p
.end()})) {}
38
39     pair<cod, point> search(Node *node, const point&
p) {
40         if (!node->first) {
41             // uncomment if we should not find the
point itself:
42             if (p == node->pt) return {LLINF, point()}
;
43             return make_pair(norm(p - node->pt), node
->pt);
44         }
45
46         Node *f = node->first, *s = node->second;
47         cod bfirst = f->distance(p), bsec = s->
distance(p);
48         if (bfirst > bsec) swap(bsec, bfirst), swap(f
, s);
49
50         auto best = search(f, p);
51         if (bsec < best.first)
52             best = min(best, search(s, p));
53         return best;
54     }
55
56     // find nearest point to a point, and its squared
distance
57     // (requires an arbitrary operator< for Point)
58     pair<cod, point> nearest(const point& p) {
59         return search(root, p);
60     }
61 };

```

4.9 Lichao

```

1 struct Lichao { // min

```

```

struct line {
    ll a, b;
    array<int, 2> ch;
    line(ll a_ = 0, ll b_ = LLINF) : a(a_), b(b_)
, ch({-1, -1}) {}
    ll operator()(ll x) { return a * x + b; }
};
vector<line> ln;

int ch(int p, int d) {
    if (ln[p].ch[d] == -1) {
        ln[p].ch[d] = ln.size();
        ln.emplace_back();
    }
    return ln[p].ch[d];
}
Lichao() { ln.emplace_back(); }

void add(line s, ll l=-N, ll r=N, int p=0) {
    ll m = (l+r)/2;
    bool L = s(l) < ln[p](l);
    bool M = s(m) < ln[p](m);
    bool R = s(r) < ln[p](r);
    if (M) swap(ln[p], s), swap(ln[p].ch, s.ch);
    if (s.b == LLINF) return;
    if (L != M) add(s, l, m-1, ch(p, 0));
    else if (R != M) add(s, m+1, r, ch(p, 1));
}
ll query(int x, ll l=-N, ll r=N, int p=0) {
    ll m = (l + r) / 2, ret = ln[p](x);
    if (ret == LLINF) return ret;
    if (x < m) return min(ret, query(x, l, m-1,
ch(p, 0)));
    return min(ret, query(x, m+1, r, ch(p, 1)));
}
};

```

4.10 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 }

```

4.11 Mindistpair

```

1 ll MinDistPair(vp &vet){
2     int n = vet.size();
3     sort(vet.begin(), vet.end());
4     set<point> s;
5
6     ll best_dist = LLINF;
7     int j=0;
8     for(int i=0;i<n;i++){
9         ll d = ceil(sqrt(best_dist));
10        while(j<n and vet[i].x-vet[j].x >= d){
11            s.erase(point(vet[j].y, vet[j].x));
12            j++;
13        }
14
15        auto it1 = s.lower_bound({vet[i].y - d, vet[i
].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->y;
20            ll dy = vet[i].y - it->x;

```

```

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 }

```

4.12 Minkowski Sum

```

1 vp minkowski(vp p, vp q){
2     int n = p.size(), m = q.size();
3     auto reorder = [&](vp &p) {
4         // set the first vertex must be the lowest
5         int id = 0;
6         for(int i=1;i<p.size();i++){
7             if(p[i].y < p[id].y or (p[i].y == p[id].y
8             and p[i].x < p[id].x))
9                 id = i;
10        }
11        rotate(p.begin(), p.begin() + id, p.end());
12    };
13    reorder(p); reorder(q);
14    p.push_back(p[0]);
15    q.push_back(q[0]);
16    vp ans; int i = 0, j = 0;
17    while(i < n or j < m){
18        ans.push_back(p[i] + q[j]);
19        cod cross = (p[i+1] - p[i]) ^ (q[j+1] - q[j])
20        ;
21        if(cross >= 0) i ++;
22        if(cross <= 0) j ++;
23    }
24    return ans;
25 }

```

4.13 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         }
32     }
33 }

```

```

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 }
47 }

```

4.14 Polygon Cut Length

```

1 // Polygon Cut length
2 ld solve(vp &p, point a, point b){ // ccw
3     int n = p.size();
4     ld ans = 0;
5
6     for(int i=0;i<n;i++){
7         int j = (i+1) % n;
8
9         int signi = ccw(a, b, p[i]);
10        int signj = ccw(a, b, p[j]);
11
12        if(signi == 0 and signj == 0){
13            if((b-a) * (p[j]-p[i]) > 0){
14                ans += param(a, b, p[j]);
15                ans -= param(a, b, p[i]);
16            }
17        }else if(signi <= 0 and signj > 0){
18            ans -= param(a, b, inter_line({a, b}, {p[
19            i], p[j]}[0]));
20        }else if(signi > 0 and signj <= 0){
21            ans += param(a, b, inter_line({a, b}, {p[
22            i], p[j]}[0]));
23        }
24    }
25    return abs(ans * norm(b-a));
26 }

```

4.15 Polygon Diameter

```

1 pair<point, point> polygon_diameter(vp p) {
2     p = convex_hull(p);
3     int n = p.size(), j = n<2 ? 0:1;
4     pair<ll, vp> res({0, {p[0], p[0]}});
5     for (int i=0;i<j;i++){
6         for (; j = (j+1) % n) {
7             res = max(res, {norm2(p[i] - p[j]), {p[i]
8             ], p[j]}});
9             if ((p[(j + 1) % n] - p[j]) ^ (p[i + 1] -
10             p[i]) >= 0)
11                 break;
12         }
13     }
14     return res.second;
15 }
16
17 double diameter(const vector<point> &p) {
18     vector<point> h = convexHull(p);
19     int m = h.size();
20     if (m == 1)
21         return 0;
22     if (m == 2)
23         return h[1] - h[0];
24 }

```

```

21     return dist(h[0], h[1]);
22     int k = 1;
23     while (area(h[m - 1], h[0], h[(k + 1) % m]) >
24            area(h[m - 1], h[0], h[k]))
25         ++k;
26     double res = 0;
27     for (int i = 0, j = k; i <= k && j < m; i++) {
28         res = max(res, dist(h[i], h[j]));
29         while (j < m && area(h[i], h[(i + 1) % m], h
30            [(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j]))
31             j = (j + 1) % m;
32         res = max(res, dist(h[i], h[(j + 1) % m]));
33     }
34     return res;

```

4.16 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             ans = max(ans, area(vet[i], vet[p2], vet[
26                j], vet[p4]));
27         }
28     }
29     return ans;
30 }

```

4.17 Sort By Angle

```

1  // Comparator function for sorting points by angle
2
3  int ret[2][2] = {{3, 2},{4, 1}};
4  inline int quad(point p) {
5      return ret[p.x >= 0][p.y >= 0];
6  }
7
8  bool comp(point a, point b) { // ccw
9      int qa = quad(a), qb = quad(b);
10     return (qa == qb ? (a ^ b) > 0 : qa < qb);
11 }
12
13 // only vectors in range [x+0, x+180)
14 bool comp(point a, point b){
15     return (a ^ b) > 0; // ccw
16     // return (a ^ b) < 0; // cw

```

```

17 }

```

4.18 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
51 // retas paralelas e intersecao nao nula
52 ld dist_seg(vp l1, vp l2){
53     assert(l2.size()==2);
54     assert(l1.size()==2);
55
56     //pontos extremos do segmento
57     ld ans = LLINF;
58     for(int i=0;i<2;i++)
59         for(int j=0;j<2;j++){
60             ans = min(ans, norm(l1[i]-l2[j]));
61         }
62
63     //verificando distancia de ponto extremo com
64     //ponto interno dos segs
65     for(int t=0;t<2;t++){
66         for(int i=0;i<2;i++){
67             bool c=true;
68             for(int k=0;k<2;k++){
69                 point va = l1[i]-l2[k];

```

```

62         point vb = l2[!k]-l2[k];
63         ld ang = atan2(norm((vb~va)), vb*va);
64         if(ang>PI/2) c = false;
65     }
66     if(c)
67         ans = min(ans,dist_pt_seg(l1[i],l2));
68 }
69 swap(l1,l2);
70 }
71
72 //ponto interno com ponto interno dos segmentos
73 point v1 = l1[1]-l1[0], v2 = l2[1]-l2[0];
74 point n = v1~v2;
75 if(!nulo(n)){
76     bool ok = true;
77     for(int t=0;t<2;t++){
78         point n2 = v2~n;
79         point oio2 = l2[0]-l1[0];
80         ld escalar = (oio2*n2)/(v1*n2);
81         if(escalar<0 or escalar>1) ok = false;
82         swap(l1,l2);
83         swap(v1,v2);
84     }
85     if(ok) ans = min(ans,dist_line(l1,l2));
86 }
87
88 return ans;
89 }
90
91 ld ver(vector<vp> &vet){
92     ld ans = LLINF;
93     // vertice - face
94     for(int k=0;k<2;k++){
95         for(int pt=0;pt<4;pt++){
96             for(int i=0;i<4;i++){
97                 vp v;
98                 for(int j=0;j<4;j++){
99                     if(i!=j) v.pb(vet[!k][j]);
100             }
101             ans = min(ans, dist_pt_face(vet[k][pt
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 return ans;
117 }
118 }

```

4.19 Voronoi

```

1 bool polygonIntersection(line &seg, vp &p) {
2     long double l = -1e18, r = 1e18;
3     for(auto ps : p) {
4         long double z = seg.eval(ps);
5         l = max(l, z);
6         r = min(r, z);
7     }
8     return l - r > EPS;
9 }
10
11 int w, h;
12
13 line getBisector(point a, point b) {
14     line ans(a, b);
15     swap(ans.a, ans.b);

```

```

16     ans.b *= -1;
17     ans.c = ans.a * (a.x + b.x) * 0.5 + ans.b * (a.y
18 + b.y) * 0.5;
19     return ans;
20 }
21
22 vp cutPolygon(vp poly, line seg) {
23     int n = (int) poly.size();
24     vp ans;
25     for(int i = 0; i < n; i++) {
26         double z = seg.eval(poly[i]);
27         if(z > -EPS) {
28             ans.push_back(poly[i]);
29         }
30         double z2 = seg.eval(poly[(i + 1) % n]);
31         if((z > EPS && z2 < -EPS) || (z < -EPS && z2
32 > EPS)) {
33             ans.push_back(inter_line(seg, line(poly[i
34 ], poly[(i + 1) % n]))[0]);
35         }
36     }
37     return ans;
38 }
39
40 // BE CAREFUL!
41 // the first point may be any point
42 // O(N^3)
43 vp getCell(vp pts, int i) {
44     vp ans;
45     ans.emplace_back(0, 0);
46     ans.emplace_back(1e6, 0);
47     ans.emplace_back(1e6, 1e6);
48     ans.emplace_back(0, 1e6);
49     for(int j = 0; j < (int) pts.size(); j++) {
50         if(j != i) {
51             ans = cutPolygon(ans, getBisector(pts[i],
52 pts[j]));
53         }
54     }
55     return ans;
56 }
57
58 // O(N^2) expected time
59 vector<vp> getVoronoi(vp pts) {
60     // assert(pts.size() > 0);
61     int n = (int) pts.size();
62     vector<int> p(n, 0);
63     for(int i = 0; i < n; i++) {
64         p[i] = i;
65     }
66     shuffle(p.begin(), p.end(), rng);
67     vector<vp> ans(n);
68     ans[0].emplace_back(0, 0);
69     ans[0].emplace_back(w, 0);
70     ans[0].emplace_back(w, h);
71     ans[0].emplace_back(0, h);
72     for(int i = 1; i < n; i++) {
73         ans[i] = ans[0];
74     }
75     for(auto i : p) {
76         for(auto j : p) {
77             if(j == i) break;
78             auto bi = getBisector(pts[j], pts[i]);
79             if(!polygonIntersection(bi, ans[j]))
80                 continue;
81             ans[j] = cutPolygon(ans[j], getBisector(
82 pts[j], pts[i]));
83             ans[i] = cutPolygon(ans[i], getBisector(
84 pts[i], pts[j]));
85         }
86     }
87     return ans;
88 }

```

5 Grafos

5.1 2sat

```
1 #define rep(i,l,r) for (int i = (l); i < (r); i++)
2 struct TwoSat { // copied from kth-competitive-
   programming/kactl
3   int N;
4   vector<vi> gr;
5   vi values; // 0 = false, 1 = true
6   TwoSat(int n = 0) : N(n), gr(2*n) {}
7   int addVar() { // (optional)
8     gr.emplace_back();
9     gr.emplace_back();
10    return N++;
11  }
12  void either(int f, int j) {
13    f = max(2*f, -1-2*f);
14    j = max(2*j, -1-2*j);
15    gr[f].push_back(j^1);
16    gr[j].push_back(f^1);
17  }
18  void atMostOne(const vi& li) { // (optional)
19    if ((int)li.size() <= 1) return;
20    int cur = ~li[0];
21    rep(i,2,(int)li.size()) {
22      int next = addVar();
23      either(cur, ~li[i]);
24      either(cur, next);
25      either(~li[i], next);
26      cur = ~next;
27    }
28    either(cur, ~li[1]);
29  }
30  vi _val, comp, z; int time = 0;
31  int dfs(int i) {
32    int low = _val[i] = ++time, x; z.push_back(i)
33    ;
34    for(int e : gr[i]) if (!comp[e])
35      low = min(low, _val[e]?: dfs(e));
36    if (low == _val[i]) do {
37      x = z.back(); z.pop_back();
38      comp[x] = low;
39      if (values[x>>1] == -1)
40        values[x>>1] = x&1;
41    } while (x != i);
42    return _val[i] = low;
43  }
44  bool solve() {
45    values.assign(N, -1);
46    _val.assign(2*N, 0); comp = _val;
47    rep(i,0,2*N) if (!comp[i]) dfs(i);
48    rep(i,0,N) if (comp[2*i] == comp[2*i+1])
49      return 0;
50    return 1;
51  }
52 };
```

5.2 Block Cut Tree

```
1 // Block-Cut Tree do brunomaletta
2 // art[i] responde o numero de novas componentes
   conexas
3 // criadas apos a remocao de i do grafo g
4 // Se art[i] >= 1, i eh ponto de articulacao
5 //
6 // Para todo i <= blocks.size()
7 // blocks[i] eh uma componente 2-vertice-conexa
   maximal
8 // edgblocks[i] sao as arestas do bloco i
9 // tree[i] eh um vertice da arvore que corresponde ao
   bloco i
```

```
10 //
11 // pos[i] responde a qual vertice da arvore vertice i
   pertence
12 // Arvore tem no maximo 2n vertices
13
14 struct block_cut_tree {
15   vector<vector<int>> g, blocks, tree;
16   vector<vector<pair<int, int>>> edgblocks;
17   stack<int> s;
18   stack<pair<int, int>> s2;
19   vector<int> id, art, pos;
20
21   block_cut_tree(vector<vector<int>> g_) : g(g_) {
22     int n = g.size();
23     id.resize(n, -1), art.resize(n), pos.resize(n)
24   };
25   build();
26
27   int dfs(int i, int& t, int p = -1) {
28     int lo = id[i] = t++;
29     s.push(i);
30
31     if (p != -1) s2.emplace(i, p);
32     for (int j : g[i]) if (j != p and id[j] !=
33       -1) s2.emplace(i, j);
34
35     for (int j : g[i]) if (j != p) {
36       if (id[j] == -1) {
37         int val = dfs(j, t, i);
38         lo = min(lo, val);
39
40         if (val >= id[i]) {
41           art[i]++;
42           blocks.emplace_back(1, i);
43           while (blocks.back().back() != j)
44             blocks.back().push_back(s.top
45               ()), s.pop();
46
47           edgblocks.emplace_back(1, s2.top
48             ()), s2.pop();
49           while (edgblocks.back().back() !=
50             pair(j, i))
51             edgblocks.back().push_back(s2
52               .top()), s2.pop();
53         }
54         // if (val > id[i]) aresta i-j eh
55         ponte
56       }
57       else lo = min(lo, id[j]);
58     }
59
60     if (p == -1 and art[i]) art[i]--;
61     return lo;
62   }
63
64   void build() {
65     int t = 0;
66     for (int i = 0; i < g.size(); i++) if (id[i]
67       == -1) dfs(i, t, -1);
68
69     tree.resize(blocks.size());
70     for (int i = 0; i < g.size(); i++) if (art[i]
71       )
72       pos[i] = tree.size(), tree.emplace_back()
73     ;
74
75     for (int i = 0; i < blocks.size(); i++) for (
76       int j : blocks[i]) {
77       if (!art[j]) pos[j] = i;
78       else tree[i].push_back(pos[j]), tree[pos[
79         j]].push_back(i);
80     }
81   }
82 }
```

```

70     }
71 };

```

5.3 Centroid Decomp

```

1  vector<int> g[N];
2  int sz[N], rem[N];
3
4  void dfs(vector<int>& path, int u, int d=0, int p=-1)
5  {
6      path.push_back(d);
7      for (int v : g[u]) if (v != p and !rem[v]) dfs(
8          path, v, d+1, u);
9  }
10
11 int dfs_sz(int u, int p=-1) {
12     sz[u] = 1;
13     for (int v : g[u]) if (v != p and !rem[v]) sz[u]
14         += dfs_sz(v, u);
15     return sz[u];
16 }
17
18 int centroid(int u, int p, int size) {
19     for (int v : g[u]) if (v != p and !rem[v] and sz[
20         v] > size / 2)
21         return centroid(v, u, size);
22     return u;
23 }
24
25 ll decomp(int u, int k) {
26     int c = centroid(u, u, dfs_sz(u));
27     rem[c] = true;
28
29     ll ans = 0;
30     vector<int> cnt(sz[u]);
31     cnt[0] = 1;
32     for (int v : g[c]) if (!rem[v]) {
33         vector<int> path;
34         dfs(path, v);
35         // d1 + d2 + 1 == k
36         for (int d : path) if (0 <= k-d-1 and k-d-1 <
37             sz[u])
38             ans += cnt[k-d-1];
39         for (int d : path) cnt[d+1]++;
40     }
41
42     for (int v : g[c]) if (!rem[v]) ans += decomp(v,
43         k);
44     return ans;
45 }

```

5.4 Dfs Tree

```

1  int desce[N], sobe[N], vis[N], h[N];
2  int backedges[N], pai[N];
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
19     for(auto v : g[u]) {
20         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 }

```

5.5 Dinic

```

1  const int N = 300;
2
3  struct Dinic {
4      struct Edge{
5          int from, to; ll flow, cap;
6      };
7      vector<Edge> edge;
8
9      vector<int> g[N];
10     int ne = 0;
11     int lvl[N], vis[N], pass;
12     int qu[N], px[N], qt;
13
14     ll run(int s, int sink, ll minE) {
15         if(s == sink) return minE;
16
17         ll ans = 0;
18
19         for(; px[s] < (int)g[s].size(); px[s]++) {
20             int e = g[s][ px[s] ];
21             auto &v = edge[e], &rev = edge[e^1];
22             if(lvl[v.to] != lvl[s]+1 || v.flow >= v.
23                 cap)
24                 continue; // v.cap - v.flow
25
26             < lim
27             ll tmp = run(v.to, sink, min(minE, v.cap-v
28                 .flow));
29             v.flow += tmp, rev.flow -= tmp;
30             ans += tmp, minE -= tmp;
31             if(minE == 0) break;
32         }
33         return ans;
34     }
35
36     bool bfs(int source, int sink) {
37         qt = 0;
38         qu[qt++] = source;
39         lvl[source] = 1;
40         vis[source] = ++pass;
41         for(int i = 0; i < qt; i++) {
42             int u = qu[i];
43             px[u] = 0;
44             if(u == sink) return true;
45             for(auto& ed : g[u]) {
46                 auto v = edge[ed];
47                 if(v.flow >= v.cap || vis[v.to] ==
48                     pass)
49                     continue; // v.cap - v.flow < lim
50                 vis[v.to] = pass;
51                 lvl[v.to] = lvl[u]+1;
52                 qu[qt++] = v.to;
53             }
54         }
55         return false;
56     }
57
58     ll flow(int source, int sink) {
59         reset_flow();
60         ll ans = 0;
61         //for(lim = (1LL << 62); lim >= 1; lim /= 2)
62         while(bfs(source, sink))
63             ans += run(source, sink, LLINF);
64         return ans;
65     }
66
67     void addEdge(int u, int v, ll c, ll rc) {
68         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 vector<pair<int, int>> cut() {
78     vector<pair<int, int>> cuts;
79     for (auto [from, to, flow, cap]: edge) {
80         if (flow == cap and vis[from] == pass and
81             vis[to] < pass and cap > 0) {
82             cuts.pb({from, to});
83         }
84     }
85     return cuts;
86 };

```

5.6 Dominator Tree

```

1 // Dominator Tree
2 // idom[x] = immediate dominator of x
3
4 vector<int> g[N], gt[N], T[N];
5 vector<int> S;
6 int dsu[N], label[N];
7 int sdom[N], idom[N], dfs_time, id[N];
8
9 vector<int> bucket[N];
10 vector<int> down[N];
11
12 void prep(int u){
13     S.push_back(u);
14     id[u] = ++dfs_time;
15     label[u] = sdom[u] = dsu[u] = u;
16
17     for(int v : g[u]){
18         if(!id[v])
19             prep(v), down[u].push_back(v);
20         gt[v].push_back(u);
21     }
22 }
23
24 int fnd(int u, int flag = 0){
25     if(u == dsu[u]) return u;
26     int v = fnd(dsu[u], 1), b = label[ dsu[u] ];
27     if(id[ sdom[b] ] < id[ sdom[ label[u] ] ])
28         label[u] = b;
29     dsu[u] = v;
30     return flag ? v : label[u];
31 }
32
33 void build_dominator_tree(int root, int sz){
34     // memset(id, 0, sizeof(int) * (sz + 1));
35     // for(int i = 0; i <= sz; i++) T[i].clear();
36     prep(root);
37     reverse(S.begin(), S.end());
38
39     int w;
40     for(int u : S){
41         for(int v : gt[u]){
42             w = fnd(v);
43             if(id[ sdom[w] ] < id[ sdom[u] ])

```

```

44         sdom[u] = sdom[w];
45     }
46     gt[u].clear();
47
48     if(u != root) bucket[ sdom[u] ].push_back(u);
49
50     for(int v : bucket[u]){
51         w = fnd(v);
52         if(sdom[w] == sdom[v]) idom[v] = sdom[v];
53         else idom[v] = w;
54     }
55     bucket[u].clear();
56
57     for(int v : down[u]) dsu[v] = u;
58     down[u].clear();
59 }
60
61 reverse(S.begin(), S.end());
62 for(int u : S) if(u != root){
63     if(idom[u] != sdom[u]) idom[u] = idom[ idom[u]
64 ] ];
65     T[ idom[u] ].push_back(u);
66 }
67 S.clear();
68 }

```

5.7 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;

```

```

44         while(int a = dfs(s, t, INF, tempo)) {
45             mflow += a;
46             tempo++;
47         }
48         return mflow;
49     }
50 };

```

5.8 Hld Aresta

```

1 // Use it together with recursive_segtree
2 const int N = 3e5+10;
3 vector<vector<pair<int, int>>> g(N, vector<pair<int,
4     int>>());
5 vector<int> in(N), inv(N), sz(N);
6 vector<int> peso(N), pai(N);
7 vector<int> head(N), tail(N), h(N);
8 int tin;
9
10 void dfs(int u, int p=-1, int depth=0){
11     sz[u] = 1; h[u] = depth;
12     for(auto &i: g[u]) if(i.ff != p){
13         auto [v, w] = i;
14         dfs(v, u, depth+1);
15         pai[v] = u; sz[u] += sz[v]; peso[v] = w;
16         if (sz[v] > sz[g[u][0].ff) or g[u][0].ff == p swap(i, g[u][0]);
17     }
18 }
19 void build_hld(int u, int p = -1) {
20     v[in[u] = tin++] = peso[u]; tail[u] = u;
21     inv[tin-1] = u;
22     for(auto &i: g[u]) if(i.ff != p) {
23         int v = i.ff;
24         head[v] = (i == g[u][0] ? head[u] : v);
25         build_hld(v, u);
26     }
27     if(g[u].size() > 1) tail[u] = tail[g[u][0].ff];
28 }
29 void init_hld(int root = 0) {
30     dfs(root);
31     tin = 0;
32     build_hld(root);
33     build();
34 }
35 void reset(){
36     g.assign(N, vector<pair<int,int>>());
37     in.assign(N, 0), sz.assign(N, 0);
38     peso.assign(N, 0), pai.assign(N, 0);
39     head.assign(N, 0); tail.assign(N, 0);
40     h.assign(N, 0); inv.assign(N, 0);
41
42     t.assign(4*N, 0); v.assign(N, 0);
43     lazy.assign(4*N, 0);
44 }
45 ll query_path(int a, int b) {
46     if (a == b) return 0;
47     if(in[a] < in[b]) swap(a, b);
48
49     if(head[a] == head[b]) return query(in[b]+1, in[a]
50 ];
51     return merge(query(in[head[a]], in[a]),
52         query_path(pai[head[a]], b));
53 }
54 void update_path(int a, int b, int x) {
55     if (a == b) return;
56     if(in[a] < in[b]) swap(a, b);
57
58     if(head[a] == head[b]) return (void)update(in[b]
59 +1, in[a], x);
60     update(in[head[a]], in[a], x); update_path(pai[
61 head[a]], b, x);

```

```

58 }
59 ll query_subtree(int a) {
60     if(sz[a] == 1) return 0;
61     return query(in[a]+1, in[a]+sz[a]-1);
62 }
63 void update_subtree(int a, int x) {
64     if(sz[a] == 1) return;
65     update(in[a]+1, in[a]+sz[a]-1, x);
66 }
67 int lca(int a, int b) {
68     if(in[a] < in[b]) swap(a, b);
69     return head[a] == head[b] ? b : lca(pai[head[a]],
70 b);
71 }

```

5.9 Hld Vertice

```

1 // Use it together with recursive_segtree
2 const int N = 3e5+10;
3 vector<vector<int>> g(N, vector<int>());
4 vector<int> in(N), inv(N), sz(N);
5 vector<int> peso(N), pai(N);
6 vector<int> head(N), tail(N), h(N);
7 int tin;
8
9 void dfs(int u, int p=-1, int depth=0){
10     sz[u] = 1; h[u] = depth;
11     for(auto &v: g[u]) if(v != p){
12         dfs(v, u, depth+1);
13         pai[v] = u; sz[u] += sz[v];
14         if (sz[v] > sz[g[u][0]] or g[u][0] == p) swap
15 (v, g[u][0]);
16     }
17 }
18 void build_hld(int u, int p = -1) {
19     v[in[u] = tin++] = peso[u]; tail[u] = u;
20     inv[tin-1] = u;
21     for(auto &v: g[u]) if(v != p) {
22         head[v] = (v == g[u][0] ? head[u] : v);
23         build_hld(v, u);
24     }
25     if(g[u].size() > 1) tail[u] = tail[g[u][0]];
26 }
27 void init_hld(int root = 0) {
28     dfs(root);
29     tin = 0;
30     build_hld(root);
31     build();
32 }
33 void reset(){
34     g.assign(N, vector<int>());
35     in.assign(N, 0), sz.assign(N, 0);
36     peso.assign(N, 0), pai.assign(N, 0);
37     head.assign(N, 0); tail.assign(N, 0);
38     h.assign(N, 0); inv.assign(N, 0);
39
40     t.assign(4*N, 0); v.assign(N, 0);
41     lazy.assign(4*N, 0);
42 }
43 ll query_path(int a, int b) {
44     if(in[a] < in[b]) swap(a, b);
45
46     if(head[a] == head[b]) return query(in[b], in[a])
47 ;
48     return merge(query(in[head[a]], in[a]),
49         query_path(pai[head[a]], b));
50 }
51 void update_path(int a, int b, int x) {
52     if(in[a] < in[b]) swap(a, b);
53
54     if(head[a] == head[b]) return (void)update(in[b],
55 in[a], x);
56     update(in[head[a]], in[a], x); update_path(pai[
57 head[a]], b, x);

```

```

53     update(in[head[a]], in[a], x); update_path(pai[
54     head[a]], b, x);
55 }
56 ll query_subtree(int a) {
57     return query(in[a], in[a]+sz[a]-1);
58 }
59 void update_subtree(int a, int x) {
60     update(in[a], in[a]+sz[a]-1, x);
61 }
62 int lca(int a, int b) {
63     if(in[a] < in[b]) swap(a, b);
64     return head[a] == head[b] ? b : lca(pai[head[a]],
65     b);
66 }

```

5.10 Hungarian

```

1 // Hungarian Algorithm
2 //
3 // Assignment problem
4 // Put the edges in the 'a' matrix (negative or
5 // positive)
6 // assignment() returns a pair with the min
7 // assignment,
8 // and the column chosen by each row
9 // assignment() - O(n^3)
10
11 template<typename T>
12 struct hungarian {
13     int n, m;
14     vector<vector<T>> a;
15     vector<T> u, v;
16     vector<int> p, way;
17     T inf;
18
19     hungarian(int n_, int m_) : n(n_), m(m_), u(m+1),
20     v(m+1), p(m+1), way(m+1) {
21         a = vector<vector<T>>(n, vector<T>(m));
22         inf = numeric_limits<T>::max();
23     }
24     pair<T, vector<int>> assignment() {
25         for (int i = 1; i <= n; i++) {
26             p[0] = i;
27             int j0 = 0;
28             vector<T> minv(m+1, inf);
29             vector<int> used(m+1, 0);
30             do {
31                 used[j0] = true;
32                 int i0 = p[j0], j1 = -1;
33                 T delta = inf;
34                 for (int j = 1; j <= m; j++) if (!
35                 used[j]) {
36                     T cur = a[i0-1][j-1] - u[i0] - v[
37                     j];
38                     if (cur < minv[j]) minv[j] = cur,
39                     j1 = j;
40                     if (minv[j] < delta) delta = minv
41                     [j], j1 = j;
42                 }
43                 for (int j = 0; j <= m; j++)
44                     if (used[j]) u[p[j]] += delta, v[
45                     j] -= delta;
46                 else minv[j] -= delta;
47                 j0 = j1;
48             } while (p[j0] != 0);
49             do {
50                 int j1 = way[j0];
51                 p[j0] = p[j1];
52                 j0 = j1;
53             } while (j0);
54             vector<int> ans(m);

```

```

48         for (int j = 1; j <= m; j++) ans[p[j]-1] = j
49         -1;
50         return make_pair(-v[0], ans);
51     }
52 };

```

5.11 Kosaraju

```

1 vector<int> g[N], gi[N]; // grafo invertido
2 int vis[N], comp[N]; // componente conexo de cada
3     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 void scc(int u, int c){
12     vis[u] = 1; comp[u] = c;
13     for(auto v: gi[u]) if(!vis[v]) scc(v, c);
14 }
15
16 void kosaraju(int n){
17     for(int i=0;i<n;i++) vis[i] = 0;
18     for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
19     for(int i=0;i<n;i++) vis[i] = 0;
20     while(S.size()){
21         int u = S.top();
22         S.pop();
23         if(!vis[u]) scc(u, u);
24     }
25 }

```

5.12 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     ; }
8     int msb(int x) { return __builtin_clz(1)-
9     __builtin_clz(x); }
10    rmq() {}
11    rmq(const vector<T>& v_) : v(v_), n(v.size()),
12    mask(n), t(n) {
13        for (int i = 0, at = 0; i < n; mask[i++] = at
14        |= 1) {
15            at = (at<<1)&((1<<b)-1);
16            while (at and op(i, i-msb(at&-at)) == i)
17                at ^= at&-at;
18            for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-
19            msb(mask[b*i+b-1]);
20            for (int j = 1; (1<<j) <= n/b; j++) for (int
21            i = 0; i+(1<<j) <= n/b; i++)
22                t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j
23                -1)+i+(1<<(j-1))]);
24        }
25        int small(int r, int sz = b) { return r-msb(mask[
26        r]&((1<<sz)-1)); }
27        T query(int l, int r) {
28            if (r-l+1 <= b) return small(r, r-l+1);
29            int ans = op(small(l+b-1), small(r));
30            int x = l/b+1, y = r/b-1;
31            if (x <= y) {
32                int j = msb(y-x+1);
33                ans = op(ans, op(t[n/b*j+x], t[n/b*j+y
34                -(1<<j)+1]));
35            }
36        }
37    };
38 }

```

```

26     }
27     return ans;
28 }
29 };
30
31 namespace lca {
32     vector<int> g[MAX];
33     int v[2*MAX], pos[MAX], dep[2*MAX];
34     int t;
35     rmq<int> RMQ;
36
37     void dfs(int i, int d = 0, int p = -1) {
38         v[t] = i, pos[i] = t, dep[t++] = d;
39         for (int j : g[i]) if (j != p) {
40             dfs(j, d+1, i);
41             v[t] = i, dep[t++] = d;
42         }
43     }
44     void build(int n, int root) {
45         t = 0;
46         dfs(root);
47         RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
48     }
49     int lca(int a, int b) {
50         a = pos[a], b = pos[b];
51         return v[RMQ.query(min(a, b), max(a, b))];
52     }
53     int dist(int a, int b) {
54         return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[lca(a, b)]];
55     }
56 }
57
58 // binary lift
59
60 const int LOG = 22;
61 vector<vector<int>> g(N);
62 int t, n;
63 vector<int> in(N), height(N);
64 vector<vector<int>> up(LOG, vector<int>(N));
65 void dfs(int u, int h=0, int p=-1) {
66     up[0][u] = p;
67     in[u] = t++;
68     height[u] = h;
69     for (auto v : g[u]) if (v != p) dfs(v, h+1, u);
70 }
71
72 void blift() {
73     up[0][0] = 0;
74     for (int i=1; i<LOG; i++) {
75         for (int j=0; j<n; j++) {
76             up[i][j] = up[i-1][up[i-1][j]];
77         }
78     }
79 }
80
81 int lca(int u, int v) {
82     if (u == v) return u;
83     if (in[u] < in[v]) swap(u, v);
84     for (int i=LOG-1; i>=0; i--) {
85         int u2 = up[i][u];
86         if (in[u2] > in[v])
87             u = u2;
88     }
89     return up[0][u];
90 }
91
92 t = 0;
93 dfs(0);
94 blift();

```

5.13 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
the graph
23        while (SPFA(src, sink)) {
24            auto flow = augment(src, sink);
25            ans.first += flow.first;
26            ans.second += flow.first * flow.second;
27            fixPot();
28        }
29        return ans;
30    }
31
32    void addEdge(int from, int to, T cap, T cost) {
33        edges[from].push_back(list.size());
34        list.push_back(Edge(to, cap, cost));
35        edges[to].push_back(list.size());
36        list.push_back(Edge(from, 0, -cost));
37    }
38 private:
39     int n;
40     std::vector<std::vector<int>> edges;
41     std::vector<Edge> list;
42     std::vector<int> from;
43     std::vector<T> dist, pot;
44     std::vector<bool> visit;
45
46     /*bool dij(int src, int sink) {
47         T INF = std::numeric_limits<T>::max();
48         dist.assign(n, INF);
49         from.assign(n, -1);
50         visit.assign(n, false);
51         dist[src] = 0;
52         for (int i = 0; i < n; i++) {
53             int best = -1;
54             for (int j = 0; j < n; j++) {
55                 if (visit[j]) continue;
56                 if (best == -1 || dist[best] > dist[j])
57                     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[best]);
66                 if (toDist < dist[ed.to]) {
67                     dist[ed.to] = toDist;
68                     from[ed.to] = e;
69                 }

```

```

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

```

5.14 Mcmf Quirino

```

1 struct Dinitz {
2     struct Edge {
3         int v, u, cap, flow=0, cost;
4         Edge(int v, int u, int cap, int cost) : v(v), u(u
5         ), cap(cap), cost(cost) {}
6     };
7
8     int n, s, t;
9     Dinitz(int n, int s, int t) : n(n), s(s), t(t) {
10         adj.resize(n);
11     }
12
13     vector<Edge> edges;

```

```

13     vector<vector<int>> adj;
14     void add_edge(int v, int u, int cap, int cost) {
15         edges.eb(v, u, cap, cost);
16         adj[v].pb(sz(edges)-1);
17         edges.eb(u, v, 0, -cost);
18         adj[u].pb(sz(edges)-1);
19     }
20
21     vector<int> dist;
22     bool spfa() {
23         dist.assign(n, LLINF);
24
25         queue<int> Q;
26         vector<bool> inqueue(n, false);
27
28         dist[s] = 0;
29         Q.push(s);
30         inqueue[s] = true;
31
32         vector<int> cnt(n);
33
34         while (!Q.empty()) {
35             int v = Q.front(); Q.pop();
36             inqueue[v] = false;
37
38             for (auto eid : adj[v]) {
39                 auto const& e = edges[eid];
40                 if (e.cap - e.flow <= 0) continue;
41                 if (dist[e.u] > dist[e.v] + e.cost) {
42                     dist[e.u] = dist[e.v] + e.cost;
43                     if (!inqueue[e.u]) {
44                         Q.push(e.u);
45                         inqueue[e.u] = true;
46                     }
47                 }
48             }
49         }
50
51         return dist[t] != LLINF;
52     }
53
54     int cost = 0;
55     vector<int> ptr;
56     int dfs(int v, int f) {
57         if (v == t || f == 0) return f;
58         for (auto &cid = ptr[v]; cid < sz(adj[v]);) {
59             auto eid = adj[v][cid];
60             auto &e = edges[eid];
61             cid++;
62             if (e.cap - e.flow <= 0) continue;
63             if (dist[e.v] + e.cost != dist[e.u]) continue;
64             int newf = dfs(e.u, min(f, e.cap-e.flow));
65             if (newf == 0) continue;
66             e.flow += newf;
67             edges[eid^1].flow -= newf;
68             cost += e.cost * newf;
69             return newf;
70         }
71         return 0;
72     }
73
74     int total_flow = 0;
75     int flow() {
76         while (spfa()) {
77             ptr.assign(n, 0);
78             while (int newf = dfs(s, LLINF))
79                 total_flow += newf;
80         }
81         return total_flow;
82     }
83 };

```

6 Math

6.1 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)
67             t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
68     for(int i=m+m-1; i>=m; --i) if(t_[i])
```

```
68         //miuns t_[i]x^{i-m}(x^{m-1}\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     }
73     inline ll calc(ll K)
74     {
75         for(int i=m; ~i; --i)
76             s[i]=t[i]=0;
77         //init
78         s[0]=1; if(m!=1) t[1]=1; else t[0]=h[0];
79         //binary-exponentiation
80         while(K)
81         {
82             if(K&1) mull(s, t);
83             mull(t, t); K>>=1;
84         }
85         ll su=0;
86         for(int i=0; i<m; ++i) su=(su+s[i]*a[i])%MOD;
87         return (su%MOD+MOD)%MOD;
88     }
89     inline int work(vector<int> x, ll n)
90     {
91         if(n<int(x.size())) return x[n];
92         vector<int> v=BM(x); m=v.size(); if(!m) return 0;
93         for(int i=0; i<m; ++i) h[i]=v[i], a[i]=x[i];
94         return calc(n);
95     }
96 }
97 }
98 using linear_seq::work;
```

6.2 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 }
```

6.3 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 };
```

6.4 Division Trick

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

6.5 Fft Mod Tfg

```
1 // usar vector<int> p(ms, 0);
2
3 const int me = 20;
4 const int ms = 1 << me;
5
6 ll fexp(ll x, ll e, ll mod = MOD) {
7     ll ans = 1;
8     x %= mod;
9     for(; e > 0; e /= 2) {
10         if(e & 1) {
11             ans = ans * x % mod;
12         }
13         x = x * x % mod;
14     }
15     return ans;
16 }
17
18 //is n primitive root of p ?
19 bool test(ll x, ll p) {
20     ll m = p - 1;
21     for(int i = 2; i * i <= m; ++i) if(m % i == 0) {
22         if(fexp(x, i, p) == 1) return false;
23         if(fexp(x, m / i, p) == 1) return false;
24     }
25     return true;
26 }
27
28 //find the largest primitive root for p
29 int search(int p) {
30     for(int i = p - 1; i >= 2; --i) if(test(i, p))
31         return i;
32     return -1;
33 }
34
35 #define add(x, y, mod) (x+y>=mod?x+y-mod:x+y)
36
37 const int gen = search(MOD);
38 int bits[ms], r[ms + 1];
39
40 void pre(int n) {
41     int LOG = 0;
42     while(1 << (LOG + 1) < n) {
43         LOG++;
44     }
45     for(int i = 1; i < n; i++) {
46         bits[i] = (bits[i >> 1] >> 1) | ((i & 1) << LOG);
47     }
48 }
49
50 void pre(int n, int root, int mod) {
51     pre(n);
52     r[0] = 1;
53     for(int i = 1; i <= n; i++) {
54         r[i] = (ll) r[i - 1] * root % mod;
55     }
56 }
57
58 vector<int> fft(vector<int> a, int mod, bool inv =
59     false) {
60     int root = gen;
61     if(inv) {
62         root = fexp(root, mod - 2, mod);
63     }
```

```
61 }
62 int n = a.size();
63 root = fexp(root, (mod - 1) / n, mod);
64 pre(n, root, mod);
65 for(int i = 0; i < n; i++) {
66     int to = bits[i];
67     if(i < to) {
68         swap(a[i], a[to]);
69     }
70 }
71 for(int len = 1; len < n; len *= 2) {
72     for(int i = 0; i < n; i += len * 2) {
73         int cur_root = 0;
74         int delta = n / (2 * len);
75         for(int j = 0; j < len; j++) {
76             int u = a[i + j], v = (ll) a[i + j +
77 len] * r[cur_root] % mod;
78             a[i + j] = add(u, v, mod);
79             a[i + j + len] = add(u, mod - v, mod);
80             cur_root += delta;
81         }
82     }
83 }
84 if(inv) {
85     int rev = fexp(n, mod - 2, mod);
86     for(int i = 0; i < n; i++)
87         a[i] = (ll) a[i] * rev % mod;
88 }
89 return a;
90 }
```

6.6 Fft Simple

```
1 #define ld long double
2 const ld PI = acos(-1);
3
4 struct num{
5     ld a {0.0}, b {0.0};
6     num(){}
7     num(ld na) : a{na}{}
8     num(ld na, ld nb) : a{na}, b{nb} {}
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);
14     }
15     const num operator*(const num &c) const{
16         return num(a*c.a - b*c.b, a*c.b + b*c.a);
17     }
18     const num operator/(const int &c) const{
19         return num(a/c, b/c);
20     }
21 };
22
23 void fft(vector<num> &a, bool invert){
24     int n = a.size();
25     for(int i=1,j=0;i<n;i++){
26         int bit = n>>1;
27         for(; j&bit; bit>>=1)
28             j^=bit;
29         j^=bit;
30         if(i<j)
31             swap(a[i], a[j]);
32     }
33     for(int len = 2; len <= n; len <= 1){
34         ld ang = 2 * PI / len * (invert ? -1 : 1);
35         num wlen(cos(ang), sin(ang));
36         for(int i=0;i<n;i+=len){
37             num w(1);
38             for (int j=0;j<len/2;j++){
39                 num u = a[i+j], v = a[i+j+len/2] * w;
```

```

40         a[i+j] = u + v;
41         a[i+j+len/2] = u - v;
42         w = w * wlen;
43     }
44 }
45 }
46 if(invert)
47     for(num &x: a)
48         x = x/n;
49 }
50 }
51
52 vector<ll> multiply(vector<int> const& a, vector<int>
53     const& b){
54     vector<num> fa(a.begin(), a.end());
55     vector<num> fb(b.begin(), b.end());
56     int n = 1;
57     while(n < int(a.size() + b.size()) )
58         n <= 1;
59     fa.resize(n);
60     fb.resize(n);
61     fft(fa, false);
62     fft(fb, false);
63     for(int i=0; i<n; i++)
64         fa[i] = fa[i]*fb[i];
65     fft(fa, true);
66     vector<ll> result(n);
67     for(int i=0; i<n; i++)
68         result[i] = round(fa[i].a);
69     while(result.back()==0) result.pop_back();
70     return result;
71 }

```

6.7 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 vector<int> rev = {0, 1};
18 const ld PI = acos(-1);
19
20 void ensure_base(int nbase){
21     if(nbase <= base)
22         return;
23     rev.resize(1 << nbase);
24     for(int i = 0; i < (1 << nbase); i++)
25         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
26             nbase - 1));
27
28     roots.resize(1 << nbase);
29
30     while(base < nbase){
31         ld angle = 2*PI / (1 << (base + 1));
32         for(int i = 1 << (base - 1); i < (1 << base);
33             i++){
34             roots[i << 1] = roots[i];
35             ld angle_i = angle * (2 * i + 1 - (1 <<
36                 base));

```

```

32         roots[(i << 1) + 1] = num(cos(angle_i),
33             sin(angle_i));
34     }
35     base++;
36 }
37
38 void fft(vector<num> &a, int n = -1){
39     if(n == -1)
40         n = a.size();
41
42     assert((n & (n-1)) == 0);
43     int zeros = __builtin_ctz(n);
44     ensure_base(zeros);
45     int shift = base - zeros;
46     for(int i = 0; i < n; i++)
47         if(i < (rev[i] >> shift))
48             swap(a[i], a[rev[i] >> shift]);
49
50     for(int k = 1; k < n; k <= 1)
51         for(int i = 0; i < n; i += 2 * k)
52             for(int j = 0; j < k; j++){
53                 num z = a[i+j+k] * roots[j+k];
54                 a[i+j+k] = a[i+j] - z;
55                 a[i+j] = a[i+j] + z;
56             }
57     }
58
59     vector<num> fa, fb;
60     vector<ll> multiply(vector<ll> &a, vector<ll> &b){
61         int need = a.size() + b.size() - 1;
62         int nbase = 0;
63         while((1 << nbase) < need) nbase++;
64         ensure_base(nbase);
65         int sz = 1 << nbase;
66         if(sz > (int) fa.size())
67             fa.resize(sz);
68
69         for(int i = 0; i < sz; i++){
70             int x = (i < (int) a.size() ? a[i] : 0);
71             int y = (i < (int) b.size() ? b[i] : 0);
72             fa[i] = num(x, y);
73         }
74         fft(fa, sz);
75         num r(0, -0.25 / sz);
76         for(int i = 0; i <= (sz >> 1); i++){
77             int j = (sz - i) & (sz - 1);
78             num z = (fa[j] * fa[j] - conj(fa[i] * fa[i]))
79                 * r;
80             if(i != j) {
81                 fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[
82                     j])) * r;
83             }
84             fa[i] = z;
85         }
86         fft(fa, sz);
87         vector<ll> res(need);
88         for(int i = 0; i < need; i++)
89             res[i] = round(fa[i].x);
90     }
91
92     return res;
93 }
94
95 vector<ll> multiply_mod(vector<ll> &a, vector<ll> &b,
96     int m, int eq = 0){
97     int need = a.size() + b.size() - 1;
98     int nbase = 0;
99     while((1 << nbase) < need) nbase++;
100     ensure_base(nbase);
101     int sz = 1 << nbase;
102     if(sz > (int) fa.size())
103         fa.resize(sz);

```



```

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 vector<ll> res(need);
148 for(int i=0;i<need;i++){
149     ll aa = round(fa[i].x);
150     ll bb = round(fb[i].x);
151     ll cc = round(fa[i].y);
152     res[i] = (aa + ((bb % m) << 15) + ((cc % m)
<< 30)) % m;
153 }
154 return res;
155 }

```

6.8 Frac

```

1 struct frac {
2     ll num, den;
3     frac(ll num=0, ll den=1) : num(num), den(den) {}
4     frac operator+(const frac &o) const { return {num
*o.den + o.num*den, den*o.den}; }
5     frac operator-(const frac &o) const { return {num
*o.den - o.num*den, den*o.den}; }
6     frac operator*(const frac &o) const { return {num
*o.num, den*o.den}; }
7     frac operator/(const frac &o) const { return {num
*o.den, den*o.num}; }
8     bool operator<(const frac &o) const { return num*
o.den < den*o.num; }

```

```
9 };
```

6.9 Fwht

```

1 // Fast Walsh Hadamard Transform
2 //
3 // FWHT<'|'>(f) eh SOS DP
4 // FWHT<'&'>(f) eh soma de superset DP
5 // Se chamar com ^, usar tamanho potencia de 2!!
6 //
7 // O(n log(n))
8
9 template<char op, class T> vector<T> FWHT(vector<T> f
, bool inv = false) {
10     int n = f.size();
11     for (int k = 0; (n-1)>>k; k++) for (int i = 0; i
< n; i++) if (i>>k&1) {
12         int j = i^(1<<k);
13         if (op == '^') f[j] += f[i], f[i] = f[j] - 2*
f[i];
14         if (op == '|') f[i] += (inv ? -1 : 1) * f[j];
15         if (op == '&') f[j] += (inv ? -1 : 1) * f[i];
16     }
17     if (op == '^' and inv) for (auto& i : f) i /= n;
18     return f;
19 }

```

6.10 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 &
(1LL << i)) {
13            if(vet[i] == 0) {
14                vet[i] = val;
15                size++;
16                return true;
17            }
18            val ^= vet[i];
19        }
20        return false;
21    }
22 };

```

6.11 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 phm) { // com phi(m), se m for primo
10     entao phi(m) = p-1
11     ll e = phm-1;
12     return fexp(a, e);

```

6.12 Kitamasa

```

1 using poly = vector<mint>; // mint = int mod P with
operators +, - and *

```

```

2 inline int len(const poly& a) { return a.size(); } // 20
   get rid of the annoying "hey a.size() is
   unsigned" warning
3
4 poly pmul(const poly& a, const poly& b) {
5     poly c(len(a) + len(b) - 1, 0);
6     for (int i = 0; i < len(a); i++)
7         for (int j = 0; j < len(b); j++)
8             c[i+j] = c[i+j] + a[i] * b[j];
9     return c;
10 }
11
12 // only works if b.back() == 1
13 poly pmod(const poly& a, const poly& b) {
14     poly c(a.begin(), a.end());
15     for (int i = len(c) - 1; i >= len(b) - 1; i--) {
16         int k = i - (len(b) - 1); // index of the
           quotient term
17         for (int j = 0; j < len(b); j++)
18             c[j+k] = c[j+k] - c[i] * b[j];
19     }
20     c.resize(len(b) - 1);
21     return c;
22 }
23
24 poly ppwr(poly x, ll e, poly f) {
25     poly ans = { 1 };
26     for (; e > 0; e /= 2) {
27         if (e & 1) ans = pmod(pmul(ans, x), f);
28         x = pmod(pmul(x, x), f);
29     }
30     return ans;
31 }
32
33 // values = { A0, A1, ..., An }. recurrence = C0 * A0
34 // + C1 * A1 + ... + Cn * An generates A{n+1}
35 mint kitamasa(const poly& values, const poly&
   recurrence, ll n) {
36     poly f(len(recurrence) + 1);
37     f.back() = 1;
38     for (int i = 0; i < len(recurrence); i++)
39         f[i] = mint(0) - recurrence[i];
40
41     auto d = ppwr(poly{0, 1}, n, f); // x^N mod f(x)
42
43     mint ans = 0;
44     for (int i = 0; i < len(values); i++)
45         ans = ans + d[i] * values[i];
46     return ans;
47 }

```

6.13 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,
   int &y0, int &g)
17 {
18     g = gcd(abs(a), abs(b), x0, y0);
19     if (c % g)

```

```

20         return false;
21
22     x0 *= c / g;
23     y0 *= c / g;
24     if (a < 0) x0 = -x0;
25     if (b < 0) y0 = -y0;
26     return true;
27 }
28
29 // All solutions
30 // x = x0 + k*b/g
31 // y = y0 - k*a/g

```

6.14 Matrix Exponentiation

```

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

```

6.15 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)%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 }

```

6.16 Mint

```

1 struct mint {
2     int x;
3     mint(int _x = 0) : x(_x) {}
4     mint operator +(const mint &o) const { return x +
5         o.x >= MOD ? x + o.x - MOD : x + o.x; }
6     mint operator *(const mint &o) const { return
7         mint((ll)x * o.x % MOD); }
8     mint operator -(const mint &o) const { return *
9         this + (MOD - o.x); }
10    mint inv() { return pwr(MOD - 2); }
11    mint pwr(ll e) {
12        mint ans = 1;
13        for (mint b=x; e; e >>= 1, b = b * b)
14            if (e & 1) ans = ans * b;
15        return ans;
16    }
17 };
18
19 mint fac[N], ifac[N];
20 void build_fac() {
21     fac[0] = 1;
22     for (int i=1; i<N; i++)
23         fac[i] = fac[i-1] * i;
24     ifac[N-1] = fac[N-1].inv();
25     for (int i=N-2; i>=0; i--)
26         ifac[i] = ifac[i+1] * (i+1);
27 }
28
29 mint c(ll n, ll k) {
30     if (k > n) return 0;
31     return fac[n] * ifac[k] * ifac[n-k];
32 }

```

6.17 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 }

```

6.18 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 }

```

6.19 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;

```

```

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

```

6.20 Poly

```

1  const int MOD = 998244353;
2  const int me = 15;
3  const int ms = 1 << me;
4
5  #define add(x, y) x+y>=MOD?x+y-MOD:x+y
6
7  const int gen = 3; // use search() from PrimitiveRoot
8  .cpp if MOD isn't 998244353
9  int bits[ms], root[ms];
10
11 void initFFT() {
12     root[1] = 1;
13     for(int len = 2; len < ms; len += len) {
14         int z = (int) fexp(gen, (MOD - 1) / len / 2);
15         for(int i = len / 2; i < len; i++) {
16             root[2 * i] = root[i];
17             root[2 * i + 1] = (int)((long long) root[i]
18                                     * z % MOD);
19         }
20     }
21 }
22
23 void pre(int n) {
24     int LOG = 0;
25     while(1 << (LOG + 1) < n) {
26         LOG++;
27     }
28     for(int i = 1; i < n; i++) {
29         bits[i] = (bits[i >> 1] >> 1) | ((i & 1) <<
30         LOG);
31     }
32 }
33
34 std::vector<int> fft(std::vector<int> a, bool inv =
35 false) {
36     int n = (int) a.size();
37     pre(n);
38     if(inv) {
39         std::reverse(a.begin() + 1, a.end());
40     }
41     for(int i = 0; i < n; i++) {
42         int to = bits[i];
43         if(i < to) { std::swap(a[i], a[to]); }
44     }
45     for(int len = 1; len < n; len *= 2) {
46         for(int i = 0; i < n; i += len * 2) {
47             for(int j = 0; j < len; j++) {
48                 int u = a[i + j], v = (int)((long
49                 long) a[i + j + len] * root[len + j] % MOD);
50                 a[i + j] = add(u, v);
51                 a[i + j + len] = add(u, MOD - v);
52             }
53         }
54     }
55 }

```

```

49 }
50 if(inv) {
51     long long rev = fexp(n, MOD-2, MOD);
52     for(int i = 0; i < n; i++)
53         a[i] = (int)(a[i] * rev % MOD);
54 }
55 return a;
56 }
57
58 std::vector<int> shift(const std::vector<int> &a, int
59 s) {
60     int n = std::max(0, s + (int) a.size());
61     std::vector<int> b(n, 0);
62     for(int i = std::max(-s, 0); i < (int) a.size();
63     i++) {
64         b[i + s] = a[i];
65     }
66     return b;
67 }
68
69 std::vector<int> cut(const std::vector<int> &a, int n
70 ) {
71     std::vector<int> b(n, 0);
72     for(int i = 0; i < (int) a.size() && i < n; i++)
73     {
74         b[i] = a[i];
75     }
76     return b;
77 }
78
79 std::vector<int> operator +(std::vector<int> a, const
80 std::vector<int> &b) {
81     int sz = (int) std::max(a.size(), b.size());
82     a.resize(sz, 0);
83     for(int i = 0; i < (int) b.size(); i++) {
84         a[i] = add(a[i], b[i]);
85     }
86     return a;
87 }
88
89 std::vector<int> operator -(std::vector<int> a, const
90 std::vector<int> &b) {
91     int sz = (int) std::max(a.size(), b.size());
92     a.resize(sz, 0);
93     for(int i = 0; i < (int) b.size(); i++) {
94         a[i] = add(a[i], MOD - b[i]);
95     }
96     return a;
97 }
98
99 std::vector<int> operator *(std::vector<int> a, std::
100 vector<int> b) {
101     while(!a.empty() && a.back() == 0) a.pop_back();
102     while(!b.empty() && b.back() == 0) b.pop_back();
103     if(a.empty() || b.empty()) return std::vector<int>
104     >(0, 0);
105     int n = 1;
106     while(n-1 < (int) a.size() + (int) b.size() - 2)
107     n += n;
108     a.resize(n, 0);
109     b.resize(n, 0);
110     a = fft(a, false);
111     b = fft(b, false);
112     for(int i = 0; i < n; i++) {
113         a[i] = (int) ((long long) a[i] * b[i] % MOD);
114     }
115     return fft(a, true);
116 }
117
118 std::vector<int> inverse(const std::vector<int> &a,
119 int k) {
120     assert(!a.empty() && a[0] != 0);
121     if(k == 0) {

```

```

112     return std::vector<int>(1, (int) fexp(a[0],
MOD - 2));
113 } else {
114     int n = 1 << k;
115     auto c = inverse(a, k-1);
116     return cut(c * cut(std::vector<int>(1, 2) -
cut(a, n) * c, n), n);
117 }
118 }
119
120 std::vector<int> operator /(std::vector<int> a, std::
vector<int> b) {
121     // NEED TO TEST!
122     while(!a.empty() && a.back() == 0) a.pop_back();
123     while(!b.empty() && b.back() == 0) b.pop_back();
124     assert(!b.empty());
125     if(a.size() < b.size()) return std::vector<int>
>(1, 0);
126     std::reverse(a.begin(), a.end());
127     std::reverse(b.begin(), b.end());
128     int n = (int) a.size() - (int) b.size() + 1;
129     int k = 0;
130     while((1 << k) - 1 < n) k++;
131     a = cut(a * inverse(b, k), (int) a.size() - (int)
b.size() + 1);
132     std::reverse(a.begin(), a.end());
133     return a;
134 }
135
136 std::vector<int> log(const std::vector<int> &a, int k
) {
137     assert(!a.empty() && a[0] != 0);
138     int n = 1 << k;
139     std::vector<int> b(n, 0);
140     for(int i = 0; i+1 < (int) a.size() && i < n; i
++) {
141         b[i] = (int)((i + 1LL) * a[i+1] % MOD);
142     }
143     b = cut(b * inverse(a, k), n);
144     assert((int) b.size() == n);
145     for(int i = n - 1; i > 0; i--) {
146         b[i] = (int) (b[i-1] * fexp(i, MOD - 2) % MOD
);
147     }
148     b[0] = 0;
149     return b;
150 }
151
152 std::vector<int> exp(const std::vector<int> &a, int k
) {
153     assert(!a.empty() && a[0] == 0);
154     if(k == 0) {
155         return std::vector<int>(1, 1);
156     } else {
157         auto b = exp(a, k-1);
158         int n = 1 << k;
159         return cut(b * cut(std::vector<int>(1, 1) +
cut(a, n) - log(b, k), n), n);
160     }
161 }

```

6.21 Primitiveroot

```

1 long long fexp(long long x, long long e, long long
mod = MOD) {
2     long long ans = 1;
3     x %= mod;
4     for(; e > 0; e /= 2, x = x * x % mod) {
5         if(e & 1) ans = ans * x % mod;
6     }
7     return ans;
8 }
9 //is n primitive root of p ?

```

```

10 bool test(long long x, long long p) {
11     long long m = p - 1;
12     for(int i = 2; i * i <= m; ++i if(!(m % i)) {
13         if(fexp(x, i, p) == 1) return false;
14         if(fexp(x, m / i, p) == 1) return false;
15     }
16     return true;
17 }
18 //find the smallest primitive root for p
19 int search(int p) {
20     for(int i = 2; i < p; i++) if(test(i, p)) return
i;
21     return -1;
22 }

```

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

```

```

55     if(__gcd(i, phi) == 1) {
56         raizes.pb(r);
57     }
58     r = (r * raiz) % mod;
59 }
60
61 return raizes;
62 }

```

6.23 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, 1500000000);
10    int num = distribution(rng);
11    while(!primo(num)) num++;
12    return num;
13 }

```

6.24 Totient

```

1 // phi(p^k) = (p^(k-1))*(p-1) com p primo
2 // O(sqrt(m))
3 ll phi(ll m){
4     ll res = m;
5     for(ll d=2; d*d<=m; d++){
6         if(m % d == 0){
7             res = (res/d)*(d-1);
8             while(m%d == 0)
9                 m /= d;
10        }
11    }
12    if(m > 1) {
13        res /= m;
14        res *= (m-1);
15    }
16    return res;
17 }
18
19 // modificacao do crivo, O(n*log(log(n)))
20 vector<ll> phi_to_n(ll n){
21     vector<bool> isprime(n+1, true);
22     vector<ll> 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 }

```

7 Misc

7.1 Bitwise

```

1 // Least significant bit (lsb)

```

```

2 int lsb(int x) { return x&-x; }
3 int lsb(int x) { return __builtin_ctz(x); } //
4 // Most significant bit (msb)
5 int msb(int x) { return 32-1-__builtin_clz(x); }
6 // bit position
7 // Power of two
8 bool isPowerOfTwo(int x){ return x && (!(x&(x-1)))
9 }; }
10 // floor(log2(x))
11 int flog2(int x) { return 32-1-__builtin_clz(x); }
12 int flog2ll(ll x) { return 64-1-__builtin_clzll(x); }
13
14 // Built-in functions
15 // Number of bits 1
16 __builtin_popcount()
17 __builtin_popcountll()
18
19 // Number of leading zeros
20 __builtin_clz()
21 __builtin_clzll()
22
23 // Number of trailing zeros
24 __builtin_ctz()
25 __builtin_ctzll()

```

7.2 Ordered Set

```

1 #include <bits/extc++.h>
2 using namespace __gnu_pbds; // or pb_ds;
3 template<typename T, typename B = null_type>
4 using ordered_set = tree<T, B, less<T>, rb_tree_tag,
5     tree_order_statistics_node_update>;
6
7 // order_of_key(k) : Number of items strictly
8 // smaller than k
9 // find_by_order(k) : K-th element in a set (counting
10 // from zero)
11
12 // to swap two sets, use a.swap(b);

```

7.3 Rand

```

1 mt19937 rng(chrono::steady_clock::now().
2     time_since_epoch().count()); // mt19937_64
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 using ull = unsigned long long;
9 ull mix(ull o){
10     o+=0x9e3779b97f4a7c15;
11     o=(o^(o>>30))*0xbf58476d1ce4e5b9;
12     o=(o^(o>>27))*0x94d049bb133111eb;
13     return o^(o>>31);
14 }
15 ull hash(pii a) {return mix(a.first ^ mix(a.second))
16 }; }

```

7.4 Submask

```

1 // O(3^n)
2 for (int m = 0; m < (1<<n); m++) {
3     for (int s = m; s; s = (s-1) & m) {
4         // s is every submask of m
5     }
6 }
7
8 // O(2^n * n) SOS dp like

```

```

9 for (int b = n-1; b >= 0; b--) {
10     for (int m = 0; m < (1 << n); m++) {
11         if (j & (1 << b)) {
12             // propagate info through submasks
13             amount[j ^ (1 << b)] += amount[j];
14         }
15     }
16 }

```

7.5 Template

```

1 #include <bits/stdc++.h>
2 #define ll long long
3 #define ff first
4 #define ss second
5 #define ld long double
6 #define pb push_back
7 #define sws cin.tie(0)->sync_with_stdio(false);
8 #define endl '\n'
9
10 using namespace std;
11
12 const int N = 0;
13 const ll MOD = 998244353;
14 const int INF = 0x3f3f3f3f;
15 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
16
17 int32_t main() {
18     #ifndef LOCAL
19         sws;
20     #endif
21
22     return 0;
23 }
24
25 // ulimit -s unlimited
26 // alias comp="g++ -std=c++20 -fsanitize=address -O2
27 // -o out"
28 // #pragma GCC optimize("O3,unroll-loops")
29 // #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

```

8 Numeric

8.1 Lagrange Interpolation

```

1 // Lagrange's interpolation O(n^2)
2 ld interpolate(vector<pair<int, int>> 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
18 template<typename T = mint>
19 struct Lagrange {
20     vector<T> y, den, l, r;
21     int n;
22     Lagrange(const vector<T>& _y) : y(_y), n(_y.size
23     ()) {
24         den.resize(n, 0);
25         l.resize(n, 0); r.resize(n, 0);

```

```

25
26         for (int i = 0; i < n; i++) {
27             den[i] = ifac[i] * ifac[n - 1 - i];
28             if ((n - 1 - i) % 2 == 1) den[i] = -den[i]
29         };
30     }
31
32     T eval(T x) {
33         l[0] = 1;
34         for (int i = 1; i < n; i++)
35             l[i] = l[i-1] * (x + -T(i-1));
36
37         r[n - 1] = 1;
38         for (int i = n - 2; i >= 0; i--)
39             r[i] = r[i+1] * (x + -T(i+1));
40
41         T ans = 0;
42         for (int i = 0; i < n; i++) {
43             T num = l[i] * r[i];
44             ans = ans + y[i] * num * den[i];
45         }
46         return ans;
47     }
48 };

```

8.2 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 }

```

8.3 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;
21     ld fl = f(l), fr = f(r);
22     ld fmid = f(mid);
23     return rsimpson(simpson(fl,fr,fmid,l,r),fl,fr,
24     fmid,l,r);
25 }

```

9 Strings

9.1 Aho Corasick

```
1 // https://github.com/joseleite19/icpc-notebook/blob/
  master/code/string/aho_corasick.cpp
2 const int A = 26;
3 int to[N][A];
4 int ne = 2, fail[N], term[N];
5 void add_string(string str, int id){
6     int p = 1;
7     for(auto c: str){
8         int ch = c - '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;
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             else if(u != 1) to[u][i] = to[ fail[u] ][i];
29         }
30     }
31 }
32 }
```

9.2 Edit Distance

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

9.3 Eertree

```
1 // heavily based on https://ideone.com/YQX9jv,
2 // which adamant cites here https://codeforces.com/
  blog/entry/139597#comment-196033
3 struct Eertree {
4     int s[N];
5     int n, last, sz;
6
7     int len[N], link[N];
8     int to[N][A];
9
10    Eertree() {
11        s[n++] = -1;
```

```
12        len[1] = -1, link[1] = 1; // "backspace" root
  is 1
13        len[0] = 0, link[0] = 1; // empty root is 0
  (to[backspace root][any char] = empty root)
14        last = 2;
15        sz = 2;
16    }
17
18    int get_link(int u) {
19        while (s[n - len[u] - 2] != s[n - 1]) u =
  link[u];
20        return u;
21    }
22
23    void push(int c) {
24        s[n++] = c;
25        int p = get_link(last);
26        if (!to[p][c]) {
27            int u = ++sz;
28            len[u] = len[p] + 2;
29            link[u] = to[get_link(link[p])][c]; //
  may be 0 (empty), but never 1 (backspace)
30            to[p][c] = u;
31        }
32        last = to[p][c];
33    }
34 };
```

9.4 Hash

```
1 // String Hash template
2 // constructor(s) - O(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
  from left to right - O(1)
4 // query_inv(l, r) from right to left - O(1)
5
6 struct Hash {
7     const ll P = 31;
8     int n; string s;
9     vector<ll> h, hi, p;
10    Hash() {}
11    Hash(string s): s(s), n(s.size()), h(n), hi(n), p
  (n) {
12        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
  % MOD;
13        for (int i=0;i<n;i++)
14            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
15        for (int i=n-1;i>=0;i--)
16            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
  % MOD;
17    }
18    int query(int l, int r) {
19        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
  0));
20        return hash < 0 ? hash + MOD : hash;
21    }
22    int query_inv(int l, int r) {
23        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-1
  +1] % MOD : 0));
24        return hash < 0 ? hash + MOD : hash;
25    }
26 };
```

9.5 Kmp

```
1 string p;
2 int neighbor[N];
3 int walk(int u, char c) { // leader after inputting '
  c'
4     while (u != -1 && (u+1 >= (int)p.size() || p[u +
  1] != c)) // leader doesn't match
5         u = neighbor[u];
```



```

6     return p[u + 1] == c ? u+1 : u;
7 }
8 void build() {
9     neighbor[0] = -1; // -1 is the leftmost state
10    for (int i = 1; i < (int)p.size(); i++)
11        neighbor[i] = walk(neighbor[i-1], p[i]);
12 }

```

9.6 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[i-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 }

```

9.7 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
32    reverse(ans.begin(), ans.end());
33
34    return ans;
35 }

```

9.8 Manacher

```

1 // 0(n), d1 -> palindromo impar, d2 -> palindromo par
   (centro da direita)
2 void manacher(string &s, vector<int> &d1, vector<int>
   &d2) {
3     int n = s.size();
4     for(int i = 0, l = 0, r = -1; i < n; i++) {
5         int k = (i > r) ? 1 : min(d1[l + r - i], r -
6         i + 1);
7         while(0 <= i - k && i + k < n && s[i - k] ==
8         s[i + k]) {
9             k++;
10        }
11        d1[i] = k--;
12        if(i + k > r) {
13            l = i - k;
14            r = i + k;
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    }
31 }

```

9.9 Suffix Array

```

1 vector<int> suffix_array(string s) {
2     s += "!";
3     int n = s.size(), N = max(n, 260);
4     vector<int> sa(n), ra(n);
5     for (int i = 0; i < n; i++) sa[i] = i, ra[i] = s[
6     i];
7
8     for (int k = 0; k < n; k ? k *= 2 : k++) {
9         vector<int> nsa(sa), nra(n), cnt(N);
10
11        for (int i = 0; i < n; i++) nsa[i] = (nsa[i] -
12        k+n)%n, cnt[ra[i]]++;
13        for (int i = 1; i < N; i++) cnt[i] += cnt[i
14        -1];
15        for (int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]
16        ]]] = nsa[i];
17
18        for (int i = 1, r = 0; i < n; i++) nra[sa[i]]
19        = r += ra[sa[i]] !=
20        ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[
21        i-1]+k)%n];
22        ra = nra;
23        if (ra[sa[n-1]] == n-1) break;
24    }
25 }

```

```

19     return vector<int>(sa.begin()+1, sa.end());
20 }
21
22 vector<int> kasai(string s, vector<int> sa) {
23     int n = s.size(), k = 0;
24     vector<int> ra(n), lcp(n);
25     for (int i = 0; i < n; i++) ra[sa[i]] = i;
26
27     for (int i = 0; i < n; i++, k -= !!k) {
28         if (ra[i] == n-1) { k = 0; continue; }
29         int j = sa[ra[i]+1];
30         while (i+k < n and j+k < n and s[i+k] == s[j+
31 k]) k++;
32         lcp[ra[i]] = k;
33     }
34     return lcp;

```

9.10 Suffix Array Radix

```

1 #define pii pair<int, int>
2
3 void radix_sort(vector<pii>& rnk, vi& ind) {
4     auto counting_sort = [](vector<pii>& rnk, vi& ind
5 ) {
6         int n = ind.size(), maxx = -1;
7         for(auto p : rnk) maxx = max(maxx, p.ff);
8
9         vi cnt(maxx+1, 0), pos(maxx+1), ind_new(n);
10        for(auto p : rnk) cnt[p.ff]++;
11        pos[0] = 0;
12
13        for(int i = 1; i <= maxx; i++) {
14            pos[i] = pos[i-1] + cnt[i-1];
15        }
16
17        for(auto idx : ind) {
18            int val = rnk[idx].ff;
19            ind_new[pos[val]] = idx;
20            pos[val]++;
21        }
22
23        swap(ind, ind_new);
24    };
25
26    for(int i = 0; i < (int)rnk.size(); i++) swap(rnk
27 [i].ff, rnk[i].ss);
28    counting_sort(rnk, ind);
29 }
30
31 vi suffix_array(const string& s) {
32     int n = s.size();
33     vector<pii> rnk(n, {0, 0});
34     vi ind(n);
35     for(int i=0; i<n; i++) {
36         rnk[i].ff = (s[i] == '$') ? 0 : s[i] - 'a' + 1;
37         // manter '$' como 0
38         ind[i] = i;
39     }
40
41     for(int k = 1; k <= n; k = (k << 1)) {
42         for(int i = 0; i < n; i++) {
43             if(ind[i]+k >= n) {
44                 rnk[ind[i]].ss = 0;
45             }
46             else {
47                 rnk[ind[i]].ss = rnk[ind[i]+k].ff;
48             }

```

```

49         radix_sort(rnk, ind); // sort(all(rnk), cmp)
50         pra n*log(n), cmp com rnk[i] < rnk[j]
51
52         vector<pii> tmp = rnk;
53         tmp[ind[0]] = {1, 0}; // rnk.ff começar em 1
54         pois '$' eh o 0
55         for(int i = 1; i < n; i++) {
56             tmp[ind[i]].ff = tmp[ind[i-1]].ff;
57             if(rnk[ind[i]] != rnk[ind[i-1]]) {
58                 tmp[ind[i]].ff++;
59             }
60             swap(rnk, tmp);
61         }
62         return ind;
63     }
64
65     vi lcp_array(const string& s, const vi& sarray) {
66         vi inv(s.size());
67         for(int i = 0; i < (int)s.size(); i++) {
68             inv[sarray[i]] = i;
69         }
70         vi lcp(s.size());
71         int k = 0;
72         for(int i = 0; i < (int)s.size()-1; i++) {
73             int pi = inv[i];
74             if(pi-1 < 0) continue;
75             int j = sarray[pi-1];
76
77             while(s[i+k] == s[j+k]) k++;
78             lcp[pi] = k;
79             k = max(k-1, 0);
80         }
81
82         return vi(lcp.begin()+1, lcp.end()); // LCP(i, j)
83         = min(lcp[i], ..., lcp[j-1])

```

9.11 Suffix Automaton

```

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

```

9.12 Z Func

```
1 vector<int> Z(string s) {
2     int n = s.size();
3     vector<int> 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 and s[z[i]] == s[i + z[i]
8     ]) {
9         x = i; y = i + z[i]; z[i]++;
10    }
11    return z;
12 }
```