

Codebook

September 1, 2024

Contents

1 Setup	1	8 Math	13
1.1 Template	1	8.1 EXGCD	13
1.2 TemplateRuru	2	8.2 DiscreteLog	13
1.3 vimrc	2	8.3 EXCRT	13
1.4 vimrc2	2	8.4 FFT	14
2 Data-structure	2	8.5 NTT	14
2.1 PBDS	2	8.6 MillerRain	15
2.2 SparseTable	2	8.7 PollardRho	15
2.3 SegmentTree	3	8.8 XorBasis	15
2.4 LazyTagSegtree	3	8.9 XorGaussianElimination	16
2.5 LiChaoTree	4	8.10 GeneratingFunctions	16
2.6 Treap	4	8.11 Numbers	17
2.7 DSU	5	8.12 Theorem	17
2.8 RollbackDSU	5		
3 Graph	5	1 Setup	
3.1 RoundSquareTree	5	1.1 Template	
3.2 SCC	6		
3.3 2SAT	6		
3.4 Bridge	6		
3.5 BronKerboschAlgorithm	6		
3.6 Theorem	7		
4 Tree	7		
4.1 HLD	7		
4.2 LCA	8		
5 Geometry	8		
5.1 Point	8		
5.2 Geometry	8		
5.3 ConvexHull	9		
5.4 MaximumDistance	9		
5.5 Theorem	10		
6 String	10		
6.1 RollingHash	10		
6.2 SuffixArray	10		
6.3 KMP	11		
6.4 Trie	11		
6.5 Zvalue	11		
7 Flow	11		
7.1 Dinic	11		
7.2 MCMF	12		

```

1 #include <bits/stdc++.h>
2 #include <bits/extc++.h>
3 #define F first
4 #define S second
5 #define pb push_back
6 #define pob pop_back
7 #define pf push_front
8 #define pof pop_front
9 #define mp make_pair
10 #define mt make_tuple
11 #define all(x) (x).begin(), (x).end()
12 using namespace std;
13 //using namespace __gnu_pbds;
14 using pii = pair<long long, long long>;
15 using ld = long double;
16 using ll = long long;
17 mt19937 mtrd(chrono::steady_clock::now() \
18 .time_since_epoch().count());
19 const int mod = 1000000007;
20 const int mod2 = 998244353;
21 const ld PI = acos(-1);
22 #define Bint __int128
23 #define int long long
24 template <typename T>
25 inline void printv(T l, T r){
26     cerr << "[ ";
27     for(; l != r; l++){
28         cerr << *l << ", ";
29     }
30 }
31 #define TEST
32 #ifdef TEST

```

```

33 #define de(x) cerr << #x << '=' << x << ", "
34 #define ed cerr << '\n';
35 #else
36 #define de(x) void(0)
37 #define ed void(0)
38 #define printv(...) void(0)
39 #endif
40 /* ----- */
41 void solve(){
42 }
43 signed main(){
44     ios::sync_with_stdio(0);
45     cin.tie(0);
46     int t = 1;
47     // cin >> t;
48     while(t--){
49         solve();
50 }

```

1.2 TemplateRuru

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 using namespace std;
4 using namespace __gnu_pbds;
5 typedef long long ll;
6 typedef pair<int, int> pii;
7 typedef vector<int> vi;
8 #define V vector
9 #define sz(a) ((int)a.size())
10 #define all(v) (v).begin(), (v).end()
11 #define rall(v) (v).rbegin(), (v).rend()
12 #define pb push_back
13 #define rsz resize
14 #define mp make_pair
15 #define mt make_tuple
16 #define ff first
17 #define ss second
18 #define FOR(i,j,k) for (int i=(j); i<=(k); i++)
19 #define FOR(i,j,k) for (int i=(j); i<(k); i++)
20 #define REP(i) FOR(_,1,i)
21 #define foreach(a,x) for (auto& a: x)
22 template<class T> bool cmin(T& a, const T& b) {
23     return b < a ? a = b, 1 : 0; } // set a =
24     ↪ min(a,b)
25 template<class T> bool cmax(T& a, const T& b) {
26     return a < b ? a = b, 1 : 0; } // set a =
27     ↪ max(a,b)
28 ll cdiv(ll a, ll b) { return a/b+((a^b)>0&&a%b); }
29 ll fddiv(ll a, ll b) { return a/b-((a^b)<0&&a%b); }
30 #define roadroller ios::sync_with_stdio(0),
31     ↪ cin.tie(0);
32 #define de(x) cerr << #x << '=' << x << ", "
33 #define dd cerr << '\n';

```

1.3 vimrc

```

1 syntax on
2 set mouse=a
3 set nu
4 set tabstop=4

```

```

5 set softtabstop=4
6 set shiftwidth=4
7 set autoindent
8 set cursorline
9 imap kj <Esc>
10 imap {} {<CR><Esc>ko<Tab>
11 imap [] []<Esc>i
12 imap () ()<Esc>i
13 imap <> <><Esc>i

```

1.4 vimrc2

```

1 se nu ai hls et ru ic is sc cul
2 se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
3 syntax on
4 hi cursorline cterm=none ctermbg=89
5 set bg=dark
6 map <F5> :w <CR> :!clear ; g++ -g --std=c++17 % &&
  ↪ echo Compiled successfully. && ./a.out; <CR>

```

2 Data-structure

2.1 PBDS

```

1 gp_hash_table<T, T> h;
2 tree<T, null_type, less<T>, rb_tree_tag,
  ↪ tree_order_statistics_node_update> tr;
3 tr.order_of_key(x); // find x's ranking
4 tr.find_by_order(k); // find k-th minimum, return
  ↪ iterator

```

2.2 SparseTable

```

1 template <class T> struct SparseTable{
2     // idx: [0, n - 1]
3     int n;
4     T id;
5     vector<vector<T>>tbl;
6     T op(T lhs, T rhs){
7         // write your mege function
8     }
9     T query(int l, int r){
10         int lg = __lg(r - l + 1);
11         return op(tbl[lg][l], tbl[lg][r - (1 << lg) +
12             ↪ 1]);
13     }
14     SparseTable (int n): n(0) {}
15     template<typename iter_t>
16     SparseTable (int _n, iter_t l, iter_t r, T _id) {
17         n = _n;
18         id = _id;
19         int lg = __lg(n) + 2;
20         tbl.resize(lg, vector<T>(n + 5, id));
21         iter_t ptr = l;
22         for(int i = 0; i < n; i++, ptr++){
23             tbl[0][i] = *ptr;
24         }

```

```

25     for(int i = 1; i <= lg; i++)
26         for(int j = 0; j + (1 << (i - 1)) < n; j++)
27             tbl[i][j] = op(tbl[i - 1][j], tbl[i - 1][j
                ↪ + (1 << (i - 1))]);
28     }
29 };

```

2.3 SegmentTree

```

1  template <class T> struct Segment_tree{
2      int L, R;
3      T id;
4      vector<T>seg;
5      T op(T lhs, T rhs){
6          // write your merge function
7      }
8      void _modify(int p, T v, int l, int r, int idx =
        ↪ 1){
9          assert(p <= r && p >= 1);
10         if(l == r){
11             seg[idx] = v;
12             return;
13         }
14         int mid = (l + r) >> 1;
15         if(p <= mid)
16             _modify(p, v, l, mid, idx << 1);
17         else
18             _modify(p, v, mid + 1, r, idx << 1 | 1);
19         seg[idx] = op(seg[idx << 1], seg[idx << 1 |
        ↪ 1]);
20     }
21     T _query(int ql, int qr, int l, int r, int idx =
        ↪ 1){
22         if(ql == l && qr == r)
23             return seg[idx];
24         int mid = (l + r) >> 1;
25         if(qr <= mid)
26             return _query(ql, qr, l, mid, idx << 1);
27         else if(ql > mid)
28             return _query(ql, qr, mid + 1, r, idx << 1 |
        ↪ 1);
29         return op(_query(ql, mid, l, mid, idx << 1),
        ↪ _query(mid + 1, qr, mid + 1, r, idx << 1 |
        ↪ 1));
30     }
31     void modify(int p, T v){ _modify(p, v, L, R, 1);
        ↪ }
32     T query(int l, int r){ return _query(l, r, L, R,
        ↪ 1); }
33     Segment_tree(): Segment_tree(0, 0, 0) {}
34     Segment_tree(int l, int r, T _id): L(l), R(r) {
35         id = _id;
36         seg.resize(4 * (r - l + 10));
37         fill(seg.begin(), seg.end(), id);
38     }
39 };

```

2.4 LazyTagSegtree

```

1  template<class T, int SZ> struct LazySeg { // SZ
        ↪ must be power of 2
2      // depends
3      T tID, ID;
4      T seg[SZ * 2], lazy[SZ * 2];
5      T cmb(T a, T b) {
6          return max(a, b);
7      }
8      LazySeg(T id, T tid): ID(id), tID(tid) {
9          for(int i = 0; i < SZ * 2; i++)
10             seg[i] = ID, lazy[i] = tID;
11     }
12     void addtag(int l, int r, int ind, int v){
13         if(lazy[ind] == tID)
14             lazy[ind] = v;
15         else
16             lazy[ind] += v;
17     }
18     /// modify values for current node
19     void push(int ind, int L, int R) {
20         // dependent on operation
21         if(lazy[ind] == tID)
22             return;
23         seg[ind] += lazy[ind];
24         if(L != R){
25             int mid = (L + R) >> 1;
26             addtag(L, mid, ind << 1, lazy[ind]);
27             addtag(mid + 1, R, ind << 1 | 1, lazy[ind]);
28         }
29         lazy[ind] = tID;
30     }
31     void pull(int ind){
32         seg[ind] = cmb(seg[ind << 1], seg[ind << 1 |
        ↪ 1]);
33     }
34     void upd(int lo, int hi, T v, int ind = 1, int L
        ↪ = 0, int R = SZ - 1) {
35         push(ind, L, R);
36         if (hi < L || R < lo) return;
37         if (lo <= L && R <= hi) {
38             addtag(L, R, ind, v);
39             push(ind, L, R); return;
40         }
41         int mid = (L + R) >> 1;
42         upd(lo, hi, v, ind << 1, L, mid);
43         upd(lo, hi, v, ind << 1 | 1, mid + 1, R);
44         pull(ind);
45     }
46     T query(int lo, int hi, int ind = 1, int L = 0,
        ↪ int R = SZ - 1) {
47         push(ind, L, R);
48         if (lo > R || L > hi) return ID;
49         if (lo <= L && R <= hi) return seg[ind];
50         int mid = (L + R) >> 1;
51         return cmb(query(lo, hi, ind << 1, L, mid),
        ↪ query(lo, hi, ind << 1 | 1, mid + 1, R));
52     }
53 }
54 };

```

2.5 LiChaoTree

```

1 struct line{
2     int m, c;
3     int val(int x){
4         return m * x + c;
5     }
6     line(): m(_id), c(0) {} // _id is the identity
7     ↪ element
8     line(int _m, int _c): m(_m), c(_c) {}
9 };
10 struct Li_Chao_Tree{
11     line seg[N << 2];
12     void ins(int l, int r, int idx, line x){
13         if(l == r){
14             if(x.val(l) > seg[idx].val(l))
15                 seg[idx] = x; // change > to < when get min
16             return;
17         }
18         int mid = (l + r) >> 1;
19         if(x.m < seg[idx].m) // change < to > when get
20             ↪ min
21             swap(x, seg[idx]);
22         if(seg[idx].val(mid) <= x.val(mid)){
23             // change <= to >= when get min
24             swap(x, seg[idx]);
25             ins(l, mid, idx << 1, x);
26         }
27         else
28             ins(mid + 1, r, idx << 1 | 1, x);
29     }
30     int query(int l, int r, int p, int idx){
31         if(l == r)
32             return seg[idx].val(l);
33         int mid = (l + r) >> 1;
34         // change max to min when get min
35         if(p <= mid)
36             return max(seg[idx].val(p), query(l, mid, p,
37                 ↪ idx << 1));
38         else
39             return max(seg[idx].val(p), query(mid + 1, r,
40                 ↪ p, idx << 1 | 1));
41     }
42 }

```

2.6 Treap

```

1 struct Treap{
2     Treap *l, *r;
3     int pri, key, sz;
4     Treap(){
5         Treap(int _v){
6             l = r = NULL;
7             pri = mtrd();
8             key = _v;
9             sz = 1;
10         }
11     ~Treap(){
12         if ( l )
13             delete l;
14         if ( r )
15             delete r;

```

```

16     }
17     void push(){
18         for(auto ch : {l, r}){
19             if(ch){
20                 // do something
21             }
22         }
23     }
24 };
25 int getSize(Treap *t){
26     return t ? t->sz : 0;
27 }
28 void pull(Treap *t){
29     t->sz = getSize(t->l) + getSize(t->r) + 1;
30 }
31 Treap* merge(Treap* a, Treap* b){
32     if(!a || !b)
33         return a ? a : b;
34     if(a->pri > b->pri){
35         a->push();
36         a->r = merge(a->r, b);
37         pull(a);
38         return a;
39     }
40     else{
41         b->push();
42         b->l = merge(a, b->l);
43         pull(b);
44         return b;
45     }
46 }
47 void splitBySize(Treap *t, Treap *&a, Treap *&b,
48     ↪ int k){
49     if(!t)
50         a = b = NULL;
51     else if(getSize(t->l) + 1 <= k){
52         a = t;
53         a->push();
54         splitBySize(t->r, a->r, b, k - getSize(t->l) -
55             ↪ 1);
56         pull(a);
57     }
58     else{
59         b = t;
60         b->push();
61         splitBySize(t->l, a, b->l, k);
62         pull(b);
63     }
64 }
65 void splitByKey(Treap *t, Treap *&a, Treap *&b, int
66     ↪ k){
67     if(!t)
68         a = b = NULL;
69     else if(t->key <= k){
70         a = t;
71         a->push();
72         splitByKey(t->r, a->r, b, k);
73         pull(a);
74     }
75     else{
76         b = t;
77         b->push();
78         splitByKey(t->l, a, b->l, k);
79         pull(b);
80     }
81 }

```

```

78 }
79 // O(n) build treap with sorted key nodes
80 void traverse(Treap *t){
81     if(t->l)
82         traverse(t->l);
83     if(t->r)
84         traverse(t->r);
85     pull(t);
86 }
87 Treap *build(int n){
88     vector<Treap*>st(n);
89     int tp = 0;
90     for(int i = 0, x; i < n; i++){
91         cin >> x;
92         Treap *nd = new Treap(x);
93         while(tp && st[tp - 1]->pri < nd->pri)
94             nd->l = st[tp - 1], tp--;
95         if(tp)
96             st[tp - 1]->r = nd;
97         st[tp++] = nd;
98     }
99     if(!tp){
100         st[0] = NULL;
101         return st[0];
102     }
103     traverse(st[0]);
104     return st[0];
105 }

```

2.7 DSU

```

1 struct Disjoint_set{
2     int n;
3     vector<int>sz, p;
4     int fp(int x){
5         return (p[x] == -1 ? x : p[x] = fp(p[x]));
6     }
7     bool U(int x, int y){
8         x = fp(x), y = fp(y);
9         if(x == y)
10             return false;
11         if(sz[x] > sz[y])
12             swap(x, y);
13         p[x] = y;
14         sz[y] += sz[x];
15         return true;
16     }
17     Disjoint_set() {}
18     Disjoint_set(int _n){
19         n = _n;
20         sz.resize(n + 5, 1);
21         p.resize(n + 5, -1);
22     }
23 };

```

2.8 RollbackDSU

```

1 struct Rollback_DSU{
2     vector<int>p, sz;
3     vector<pair<int, int>>history;
4     int fp(int x){

```

```

5         while(p[x] != -1)
6             x = p[x];
7         return x;
8     }
9     bool U(int x, int y){
10         x = fp(x), y = fp(y);
11         if(x == y){
12             history.push_back(make_pair(-1, -1));
13             return false;
14         }
15         if(sz[x] > sz[y])
16             swap(x, y);
17         p[x] = y;
18         sz[y] += sz[x];
19         history.push_back(make_pair(x, y));
20         return true;
21     }
22     void undo(){
23         if(history.empty() || history.back().first ==
24             ↪ -1){
25             if(!history.empty())
26                 history.pop_back();
27             return;
28         }
29         auto [x, y] = history.back();
30         history.pop_back();
31         p[x] = -1;
32         sz[y] -= sz[x];
33     }
34     Rollback_DSU(): Rollback_DSU(0) {}
35     Rollback_DSU(int n): p(n + 5), sz(n + 5) {
36         fill(p.begin(), p.end(), -1);
37         fill(sz.begin(), sz.end(), 1);
38     };

```

3 Graph

3.1 RoundSquareTree

```

1 int cnt;
2 int dep[N], low[N]; // dep == -1 -> unvisited
3 vector<int>G[N], rstree[2 * N]; // 1 ~ n: round, n
4     ↪ + 1 ~ 2n: square
5 vector<int>stk;
6 void init(){
7     cnt = n;
8     for(int i = 1; i <= n; i++){
9         G[i].clear();
10        rstree[i].clear();
11        rstree[i + n].clear();
12        dep[i] = low[i] = -1;
13    }
14    dep[1] = low[1] = 0;
15 }
16 void tarjan(int x, int px){
17     stk.push_back(x);
18     for(auto i : G[x]){
19         if(dep[i] == -1){
20             dep[i] = low[i] = dep[x] + 1;
21             tarjan(i, x);
22             low[x] = min(low[x], low[i]);

```

```

22         if(dep[x] <= low[i]){
23             int z;
24             cnt++;
25             do{
26                 z = stk.back();
27                 rstree[cnt].push_back(z);
28                 rstree[z].push_back(cnt);
29                 stk.pop_back();
30             }while(z != i);
31             rstree[cnt].push_back(x);
32             rstree[x].push_back(cnt);
33         }
34     }
35     else if(i != px)
36         low[x] = min(low[x], dep[i]);
37 }
38 }

```

3.2 SCC

```

1 struct SCC{
2     int n;
3     int cnt;
4     vector<vector<int>>>G, revG;
5     vector<int>stk, sccid;
6     vector<bool>vis;
7     SCC(): SCC(0) {}
8     SCC(int _n): n(_n), G(_n + 1), revG(_n + 1),
9         ↪ sccid(_n + 1), vis(_n + 1), cnt(0) {}
10    void addEdge(int u, int v){
11        // u -> v
12        assert(u > 0 && u <= n);
13        assert(v > 0 && v <= n);
14        G[u].push_back(v);
15        revG[v].push_back(u);
16    }
17    void dfs1(int u){
18        vis[u] = 1;
19        for(int v : G[u]){
20            if(!vis[v])
21                dfs1(v);
22        }
23        stk.push_back(u);
24    }
25    void dfs2(int u, int k){
26        vis[u] = 1;
27        sccid[u] = k;
28        for(int v : revG[u]){
29            if(!vis[v])
30                dfs2(v, k);
31        }
32    }
33    void Kosaraju(){
34        for(int i = 1; i <= n; i++)
35            if(!vis[i])
36                dfs1(i);
37        fill(vis.begin(), vis.end(), 0);
38        while(!stk.empty()){
39            if(!vis[stk.back()])
40                dfs2(stk.back(), ++cnt);
41            stk.pop_back();
42        }
43    }

```

```

43 };

```

3.3 2SAT

```

1 struct two_sat{
2     int n;
3     SCC G; // u: u, u + n: ~u
4     vector<int>ans;
5     two_sat(): two_sat(0) {}
6     two_sat(int _n): n(_n), G(2 * _n), ans(_n + 1) {}
7     void disjunction(int a, int b){
8         G.addEdge((a > n ? a - n : a + n), b);
9         G.addEdge((b > n ? b - n : b + n), a);
10    }
11    bool solve(){
12        G.Kosaraju();
13        for(int i = 1; i <= n; i++){
14            if(G.sccid[i] == G.sccid[i + n])
15                return false;
16            ans[i] = (G.sccid[i] > G.sccid[i + n]);
17        }
18        return true;
19    }
20 };

```

3.4 Bridge

```

1 int dep[N], low[N];
2 vector<int>G[N];
3 vector<pair<int, int>>bridge;
4 void init(){
5     for(int i = 1; i <= n; i++){
6         G[i].clear();
7         dep[i] = low[i] = -1;
8     }
9     dep[1] = low[1] = 0;
10 }
11 void tarjan(int x, int px){
12     for(auto i : G[x]){
13         if(dep[i] == -1){
14             dep[i] = low[i] = dep[x] + 1;
15             tarjan(i, x);
16             low[x] = min(low[x], low[i]);
17             if(low[i] > dep[x])
18                 bridge.push_back(make_pair(i, x));
19         }
20         else if(i != px)
21             low[x] = min(low[x], dep[i]);
22     }
23 }

```

3.5 BronKerboschAlgorithm

```

1 vector<vector<int>>>maximal_clique;
2 int cnt, G[N][N], all[N][N], some[N][N],
3 ↪ none[N][N];
4 void dfs(int d, int an, int sn, int nn)
5 {
6     if(sn == 0 && nn == 0){

```

```

6 vector<int>v;
7 for(int i = 0; i < an; i++)
8     v.push_back(all[d][i]);
9 maximal_clique.push_back(v);
10 cnt++;
11 }
12 int u = sn > 0 ? some[d][0] : none[d][0];
13 for(int i = 0; i < sn; i++)
14 {
15     int v = some[d][i];
16     if(G[u][v])
17         continue;
18     int tsu = 0, tnn = 0;
19     for(int j = 0; j < an; j++)
20         all[d + 1][j] = all[d][j];
21     all[d + 1][an] = v;
22     for(int j = 0; j < sn; j++)
23         if(g[v][some[d][j]])
24             some[d + 1][tsu++] = some[d][j];
25     for(int j = 0; j < nn; j++)
26         if(g[v][none[d][j]])
27             none[d + 1][tnn++] = none[d][j];
28     dfs(d + 1, an + 1, tsu, tnn);
29     some[d][i] = 0, none[d][nn++] = v;
30 }
31 }
32 void process(){
33     cnt = 0;
34     for(int i = 0; i < n; i++)
35         some[0][i] = i + 1;
36     dfs(0, 0, n, 0);
37 }

```

3.6 Theorem

- Kosaraju's algorithm visit the strong connected components in topological order at second dfs.
- Euler's formula on planar graph: $V - E + F = C + 1$
- Kuratowski's theorem: A simple graph G is a planar graph iff G doesn't has a subgraph H such that H is homeomorphic to K_5 or $K_{3,3}$
- A complement set of every vertex cover correspond to a independent set. \Rightarrow Number of vertex of maximum independent set + Number of vertex of minimum vertex cover = V
- Maximum independent set of G = Maximum clique of the complement graph of G .
- A planar graph G colored with three colors iff there exist a maximal clique I such that $G - I$ is a bipartite.

4 Tree

4.1 HLD

```

1 /**
2  * Description: Heavy-Light Decomposition, add val
  ↳ to verts
3  * and query sum in path/subtree.
4  * Time: any tree path is split into  $O(\log N)$  parts

```

```

5  */
6  // #include "LazySeg.h"
7  template<int SZ, bool VALS_IN_EDGES> struct HLD {
8      int N; vi adj[SZ];
9      int par[SZ], root[SZ], depth[SZ], sz[SZ], ti;
10     int pos[SZ]; vi rpos;
11     // rpos not used but could be useful
12     void ae(int x, int y) {
13         adj[x].pb(y), adj[y].pb(x);
14     }
15     void dfsSz(int x) {
16         sz[x] = 1;
17         foreach(y, adj[x]) {
18             par[y] = x; depth[y] = depth[x] + 1;
19             adj[y].erase(find(all(adj[y]), x));
20             // remove parent from adj list
21             dfsSz(y); sz[x] += sz[y];
22             if (sz[y] > sz[adj[x][0]])
23                 swap(y, adj[x][0]);
24         }
25     }
26     void dfsHld(int x) {
27         pos[x] = ti++; rpos.pb(x);
28         foreach(y, adj[x]) {
29             root[y] =
30                 (y == adj[x][0] ? root[x] : y);
31             dfsHld(y); }
32     }
33     void init(int _N, int R = 0) { N = _N;
34         par[R] = depth[R] = ti = 0; dfsSz(R);
35         root[R] = R; dfsHld(R);
36     }
37     int lca(int x, int y) {
38         for (; root[x] != root[y]; y = par[root[y]])
39             if (depth[root[x]] > depth[root[y]])
40                 ↳ swap(x, y);
41         return depth[x] < depth[y] ? x : y;
42     }
43     // int dist(int x, int y) { // # edges on path
44     //     return depth[x] + depth[y] - 2 * depth[lca(x, y)];
45     //     ↳ }
46     LazySeg<ll, SZ> tree; // segtree for sum
47     template <class BinaryOp>
48     void processPath(int x, int y, BinaryOp op) {
49         for (; root[x] != root[y]; y = par[root[y]]) {
50             if (depth[root[x]] > depth[root[y]])
51                 ↳ swap(x, y);
52             op(pos[root[y]], pos[y]); }
53         if (depth[x] > depth[y]) swap(x, y);
54         op(pos[x] + VALS_IN_EDGES, pos[y]);
55     }
56     void modifyPath(int x, int y, int v) {
57         processPath(x, y, [this, &v](int l, int r) {
58             tree.upd(l, r, v); });
59     }
60     ll queryPath(int x, int y) {
61         ll res = 0;
62         processPath(x, y, [this, &res](int l, int r) {
63             res += tree.query(l, r); });
64         return res;
65     }
66     void modifySubtree(int x, int v) {
67         ↳ tree.upd(pos[x] + VALS_IN_EDGES, pos[x] + sz[x] - 1, v)
68     }

```



```
66 };
```

4.2 LCA

```
1 int anc[20][N];
2 int dis[20][N];
3 int dep[N];
4 vector<pair<int, int>>G[N]; // weighted(edge) tree
5 void dfs(int u, int pu = 0){
6     for(int i = 1; i < 20; i++){
7         anc[i][u] = anc[i - 1][anc[i - 1][u]];
8         dis[i][u] = dis[i - 1][u] + dis[i - 1][anc[i - 1][u]];
9     }
10    for(auto [v, c] : G[u]){
11        if(v == pu)
12            continue;
13        dep[v] = dep[u] + 1;
14        anc[0][v] = u;
15        dis[0][v] = c;
16        dfs(v, u);
17    }
18 }
19 int LCA(int x, int y){
20     if(dep[x] < dep[y])
21         swap(x, y);
22     int diff = dep[x] - dep[y];
23     for(int i = 19; i >= 0; i--){
24         if(diff - (1 << i) >= 0)
25             x = anc[i][x], diff -= (1 << i);
26     }
27     if(x == y)
28         return x;
29     for(int i = 19; i >= 0; i--){
30         if(anc[i][x] != anc[i][y]){
31             x = anc[i][x];
32             y = anc[i][y];
33         }
34     }
35     return anc[0][x];
36 }
```

5 Geometry

5.1 Point

```
1 template<class T> struct Point {
2     T x, y;
3     Point(): x(0), y(0) {};
4     Point(T a, T b): x(a), y(b) {};
5     Point(pair<T, T>p): x(p.first), y(p.second) {};
6     Point operator + (const Point& rhs){ return
7         ↪ Point(x + rhs.x, y + rhs.y); }
8     Point operator - (const Point& rhs){ return
9         ↪ Point(x - rhs.x, y - rhs.y); }
10    Point operator * (const T& rhs){ return Point(x *
11        ↪ rhs, y * rhs); }
12    Point operator / (const T& rhs){ return Point(x /
13        ↪ rhs, y / rhs); }
```

```
10 T cross(Point rhs){ return x * rhs.y - y * rhs.x;
11     ↪ }
12 T dot(Point rhs){ return x * rhs.x + y * rhs.y; }
13 T cross2(Point a, Point b){ // (a - this) cross
14     ↪ (b - this)
15     return (a - *this).cross(b - *this);
16 }
17 T dot2(Point a, Point b){ // (a - this) dot (b -
18     ↪ this)
19     return (a - *this).dot(b - *this);
20 }
21 };
22 struct Circle {
23     Point<double>O;
24     double R;
25     Circle(): O(), R(0) {}
26     Circle(double _R): O(), R(_R) {}
27     Circle(double _x, double _y, double _R): O(_x,
28         ↪ _y), R(_R) {}
29 };
30 
```

5.2 Geometry

```
1 template<class T> int ori(Point<T>a, Point<T>b,
2     ↪ Point<T>c){
3     // sign of (b - a) cross(c - a)
4     auto res = a.cross2(b, c);
5     // if type is double
6     // if(abs(res) <= eps)
7     if(res == 0)
8         return 0;
9     return res > 0 ? 1 : -1;
10 }
11 template<class T> bool collinearity(Point<T>a,
12     ↪ Point<T>b, Point<T>c){
13     // if type is double
14     // return abs(c.cross2(a,b)) <= eps;
15     return c.cross2(a, b) == 0;
16 }
17 template<class T> bool between(Point<T>a,
18     ↪ Point<T>b, Point<T>c){
19     // check if c is between a, b
20     return collinearity(a, b, c) && c.dot2(a, b) <=
21         ↪ 0;
22 }
23 template<class T> bool seg_intersect(Point<T>p1,
24     ↪ Point<T>p2, Point<T>p3, Point<T>p4){
25     // seg (p1, p2), seg(p3, p4)
26     int a123 = ori(p1, p2, p3);
27     int a124 = ori(p1, p2, p4);
28     int a341 = ori(p3, p4, p1);
29     int a342 = ori(p3, p4, p2);
30     if(a123 == 0 && a124 == 0)
31         return between(p1, p2, p3) || between(p1, p2,
32             ↪ p4) || between(p3, p4, p1) || between(p3,
33             ↪ p4, p2);
34     return a123 * a124 <= 0 && a341 * a342 <= 0;
35 }
36 template<class T> Point<T>
37     ↪ point2point_intersect_at(Point<T> a, Point<T>
38     ↪ b, Point<T> c, Point<T> d) {
39     // line(a, b), line(c, d)
40     T a123 = a.cross(b, c);
```



```

32 T a124 = a.cross(b, d);
33 return (d * a123 - c * a124) / (a123 - a124);
34 }
35 bool circle2circle_intersect_at(Circle c1, Circle
↪ c2, Point<double>&p1, Point<double>&p2){
36 // return 1 if has intersect points
37 Point<double>o1 = c1.O, o2 = c2.O;
38 Point<double>od = o1 - o2;
39 double r1 = a.R, r2 = b.R, d2 = od.dot(od), d =
↪ sqrt(d2);
40 if(d < max(r1, r2) - min(r1, r2) || d > r1 + r2)
↪ return 0;
41 Point<double> u = (o1 + o2) * 0.5 + (o1 - o2) *
↪ ((r2 * r2 - r1 * r1) / (2 * d2));
42 double A = sqrt((r1 + r2 + d) * (r1 - r2 + d) *
↪ (r1 + r2 - d) * (-r1 + r2 + d));
43 Point<double> v = Point(o1.y - o2.y, -o1.x +
↪ o2.x) * A / (2 * d2);
44 p1 = u + v, p2 = u - v;
45 return 1;
46 }
47 template<class T> int
↪ point_in_convex_polygon(vector<Point<T>>& a,
↪ Point<T>p){
48 // 1: IN
49 // 0: OUT
50 // -1: ON
51 // the points of convex polygon must sort in
↪ counter-clockwise order
52 int n = a.size();
53 if(between(a[0], a[1], p) || between(a[0], a[n -
↪ 1], p))
54 return -1;
55 int l = 0, r = n - 1;
56 while(l <= r){
57 int mid = (l + r) >> 1;
58 auto a1 = a[0].cross2(a[mid], p);
59 auto a2 = a[0].cross2(a[(mid + 1) % n], p);
60 if(a1 >= 0 && a2 <= 0){
61 auto res = a[mid].cross2(a[(mid + 1) % n],
↪ p);
62 return res > 0 ? 1 : (res >= 0 ? -1 : 0);
63 }
64 else if(a1 < 0)
65 r = mid - 1;
66 else
67 l = mid + 1;
68 }
69 return 0;
70 }
71 template<class T> int
↪ point_in_simple_polygon(vector<Point<T>>&a,
↪ Point<T>p, Point<T>INF_point){
72 // 1: IN
73 // 0: ON
74 // -1: OUT
75 // a[i] must adjacent to a[(i + 1) % n] for all i
76 // collinearity(a[i], p, INF_point) must be false
↪ for all i
77 // we can let the slope of line(p, INF_point) be
↪ irrational (e.g. PI)
78 int ans = -1;
79 for(auto l = prev(a.end()), r = a.begin(); r !=
↪ a.end(); l = r++){
80 if(between(*l, *r, p))

```

```

81 return 0;
82 if(seg_intersect(*l, *r, p, INF_point)){
83 ans *= -1;
84 if(collinearity(*l, p, INF_point))
85 assert(0);
86 }
87 }
88 return ans;
89 }
90 template<class T> T area(vector<Point<T>>&a){
91 // remember to divide 2 after calling this
↪ function
92 if(a.size() <= 1)
93 return 0;
94 T ans = 0;
95 for(auto l = prev(a.end()), r = a.begin(); r !=
↪ a.end(); l = r++){
96 ans += l->cross(*r);
97 return ans;
98 }

```

5.3 ConvexHull

```

1 template<class T> vector<Point<T>>
↪ convex_hull(vector<Point<T>>&a){
2 int n = a.size();
3 sort(a.begin(), a.end(), [](Point<T>p1,
↪ Point<T>p2){
4 if(p1.x == p2.x)
5 return p1.y < p2.y;
6 return p1.x < p2.x;
7 });
8 int m = 0, t = 1;
9 vector<Point<T>>ans;
10 auto addPoint = [&](const Point<T>p) {
11 while(m > t && ans[m - 2].cross2(ans[m - 1], p)
↪ <= 0)
12 ans.pop_back(), m--;
13 ans.push_back(p);
14 m++;
15 };
16 for(int i = 0; i < n; i++)
17 addPoint(a[i]);
18 t = m;
19 for(int i = n - 2; ~i; i--)
20 addPoint(a[i]);
21 if(a.size() > 1)
22 ans.pop_back();
23 return ans;
24 }

```

5.4 MaximumDistance

```

1 template<class T>
2 T MaximumDistance(vector<Point<T>>&p){
3 vector<Point<T>>C = convex_hull(p);
4 int n = C.size(), t = 2;
5 T ans = 0;
6 for(int i = 0; i < n; i++){

```

```

7   while(((C[i] - C[t]) ^ (C[(i+1)%n] - C[t])) <
   ↪ ((C[i] - C[(t+1)%n]) ^ (C[(i+1)%n] -
   ↪ C[(t+1)%n]))) t = (t + 1)%n;
8   ans = max({ans, abs2(C[i] - C[t]),
   ↪ abs2(C[(i+1)%n] - C[t])});
9   }
10  return ans;
11 }

```

5.5 Theorem

- Pick's theorem: Suppose that a polygon has integer coordinates for all of its vertices. Let i be the number of integer points interior to the polygon, b be the number of integer points on its boundary (including both vertices and points along the sides). Then the area A of this polygon is:

$$A = i + \frac{b}{2} - 1$$

6 String

6.1 RollingHash

```

1 struct Rolling_Hash{
2   int n;
3   const int P[5] = {146672737, 204924373,
   ↪ 585761567, 484547929, 116508269};
4   const int M[5] = {922722049, 952311013,
   ↪ 955873937, 901981687, 993179543};
5   vector<int>PW[5], pre[5], suf[5];
6   Rolling_Hash(): Rolling_Hash("") {}
7   Rolling_Hash(string s): n(s.size()){
8     for(int i = 0; i < 5; i++){
9       PW[i].resize(n), pre[i].resize(n),
   ↪ suf[i].resize(n);
10    PW[i][0] = 1, pre[i][0] = s[0];
11    suf[i][n - 1] = s[n - 1];
12  }
13  for(int i = 1; i < n; i++){
14    for(int j = 0; j < 5; j++){
15      PW[j][i] = PW[j][i - 1] * P[j] % M[j];
16      pre[j][i] = (pre[j][i - 1] * P[j] + s[i]) %
   ↪ M[j];
17    }
18  }
19  for(int i = n - 2; i >= 0; i--){
20    for(int j = 0; j < 5; j++){
21      suf[j][i] = (suf[j][i + 1] * P[j] + s[i]) %
   ↪ M[j];
22    }
23  }
24  int _substr(int k, int l, int r) {
25    int res = pre[k][r];
26    if(l > 0)
27      res -= 1LL * pre[k][l - 1] * PW[k][r - l + 1]
   ↪ % M[k];
28    if(res < 0)
29      res += M[k];
30    return res;
31  }
32  vector<int>substr(int l, int r){
33    vector<int>res(5);

```

```

34    for(int i = 0; i < 5; ++i)
35      res[i] = _substr(i, l, r);
36    return res;
37  }
38 };

```

6.2 SuffixArray

```

1 struct Suffix_Array{
2   int n, m; // m is the range of s
3   string s;
4   vector<int>sa, rk, lcp;
5   // sa[i]: the i-th smallest suffix
6   // rk[i]: the rank of suffix i (i.e. s[i, n - 1])
7   // lcp[i]: the longest common prefix of sa[i] and
   ↪ sa[i - 1]
8   Suffix_Array(): Suffix_Array(0, 0, "") {};
9   Suffix_Array(int _n, int _m, string _s): n(_n),
   ↪ m(_m), sa(_n), rk(_n), lcp(_n), s(_s) {}
10  void Sort(int k, vector<int>&bucket,
   ↪ vector<int>&idx, vector<int>&lst){
11    for(int i = 0; i < m; i++)
12      bucket[i] = 0;
13    for(int i = 0; i < n; i++)
14      bucket[lst[i]]++;
15    for(int i = 1; i < m; i++)
16      bucket[i] += bucket[i - 1];
17    int p = 0;
18    // update index
19    for(int i = n - k; i < n; i++)
20      idx[p++] = i;
21    for(int i = 0; i < n; i++)
22      if(sa[i] >= k)
23        idx[p++] = sa[i] - k;
24    for(int i = n - 1; i >= 0; i--)
25      sa[--bucket[lst[idx[i]]]] = idx[i];
26  }
27  void build(){
28    vector<int>idx(n), lst(n), bucket(max(n, m));
29    for(int i = 0; i < n; i++)
30      bucket[lst[i] = (s[i] - 'a')]++; // may
   ↪ change
31    for(int i = 1; i < m; i++)
32      bucket[i] += bucket[i - 1];
33    for(int i = n - 1; i >= 0; i--)
34      sa[--bucket[lst[i]]] = i;
35    for(int k = 1; k < n; k <= 1){
36      Sort(k, bucket, idx, lst);
37      // update rank
38      int p = 0;
39      idx[sa[0]] = 0;
40      for(int i = 1; i < n; i++){
41        int a = sa[i], b = sa[i - 1];
42        if(lst[a] == lst[b] && a + k < n && b + k <
   ↪ n && lst[a + k] == lst[b + k]);
43        else
44          p++;
45        idx[sa[i]] = p;
46      }
47      if(p == n - 1)
48        break;
49      for(int i = 0; i < n; i++)
50        lst[i] = idx[i];

```

```

51     m = p + 1;
52 }
53 for(int i = 0; i < n; i++)
54     rk[sa[i]] = i;
55 buildLCP();
56 }
57 void buildLCP(){
58     // lcp[rk[i]] >= lcp[rk[i - 1]] - 1
59     int v = 0;
60     for(int i = 0; i < n; i++){
61         if(!rk[i])
62             lcp[rk[i]] = 0;
63         else{
64             if(v)
65                 v--;
66             int p = sa[rk[i] - 1];
67             while(i + v < n && p + v < n && s[i + v] ==
68                 ↪ s[p + v])
69                 v++;
70             lcp[rk[i]] = v;
71         }
72     }
73 };

```

6.3 KMP

```

1 struct KMP {
2     int n;
3     string s;
4     vector<int> fail;
5     // s: pattern, t: text => find s in t
6     int match(string &t){
7         int ans = 0, m = t.size(), j = -1;
8         for(int i = 0; i < m; i++){
9             while(j != -1 && t[i] != s[j + 1])
10                 j = fail[j];
11             if(t[i] == s[j + 1])
12                 j++;
13             if(j == n - 1){
14                 ans++;
15                 j = fail[j];
16             }
17         }
18         return ans;
19     }
20     KMP(string &s){
21         s = _s;
22         n = s.size();
23         fail = vector<int>(n, -1);
24         int j = -1;
25         for(int i = 1; i < n; i++){
26             while(j != -1 && s[i] != s[j + 1])
27                 j = fail[j];
28             if(s[i] == s[j + 1])
29                 j++;
30             fail[i] = j;
31         }
32     }
33 };

```

6.4 Trie

```

1 struct Node {
2     int hit = 0;
3     Node *next[26];
4     // 26 is the size of the set of characters
5     // a - z
6     Node(){
7         for(int i = 0; i < 26; i++)
8             next[i] = NULL;
9     }
10 };
11 void insert(string &s, Node *node){
12     // node cannot be null
13     for(char v : s){
14         if(node->next[v - 'a'] == NULL)
15             node->next[v - 'a'] = new Node;
16         node = node->next[v - 'a'];
17     }
18     node->hit++;
19 }

```

6.5 Zvalue

```

1 struct Zvalue {
2     const string inf = "$"; // character that has
3     ↪ never used
4     vector<int> z;
5     // s: pattern, t: text => find s in t
6     int match(string &s, string &t){
7         string fin = s + inf + t;
8         build(fin);
9         int n = s.size(), m = t.size();
10        int ans = 0;
11        for(int i = n + 1; i < n + m + 1; i++){
12            if(z[i] == n)
13                ans++;
14        }
15        return ans;
16    }
17    void build(string &s){
18        int n = s.size();
19        z = vector<int>(n, 0);
20        int l = 0, r = 0;
21        for(int i = 0; i < n; i++){
22            z[i] = max(min(z[i - 1], r - i), 0LL);
23            while(i + z[i] < n && s[z[i]] == s[i + z[i]])
24                l = i, r = i + z[i], z[i]++;
25        }
26    }
27 };

```

7 Flow

7.1 Dinic

```

1 /**
2  * After computing flow, edges {u,v} s.t
3  * lev[u] ≠ -1, lev[v] = -1 are part of min cut.

```

```

4  * Use \texttt{reset} and \texttt{rcap} for
   ↪ Gomory-Hu.
5  * Time:  $O(N^2M)$  flow
6  *  $O(M\sqrt{N})$  bipartite matching
7  *  $O(NM\sqrt{N})$  or  $O(NM\sqrt{M})$  on unit graph.
8  */
9  struct Dinic {
10     using F = long long; // flow type
11     struct Edge { int to; F flo, cap; };
12     int N;
13     vector<Edge> eds;
14     vector<vector<int>>> adj;
15     void init(int _N) {
16         N = _N; adj.resize(N), cur.resize(N);
17     }
18     void reset() {
19         for (auto &e: eds) e.flo = 0;
20     }
21     void ae(int u, int v, F cap, F rcap = 0) {
22         assert(min(cap,rcap) >= 0);
23         adj[u].pb((int)eds.size());
24         eds.pb({v, 0, cap});
25         adj[v].pb((int)eds.size());
26         eds.pb({u, 0, rcap});
27     }
28     vector<int> lev;
29     vector<vector<int>::iterator> cur;
30     // level = shortest distance from source
31     bool bfs(int s, int t) {
32         lev = vector<int>(N, -1);
33         for(int i = 0; i < N; i++) cur[i] =
34             ↪ begin(adj[i]);
35         queue<int> q({s}); lev[s] = 0;
36         while (!q.empty()) {
37             int u = q.front(); q.pop();
38             for (auto &e: adj[u]) {
39                 const Edge& E = eds[e];
40                 int v = E.to;
41                 if (lev[v] < 0 && E.flo < E.cap)
42                     q.push(v), lev[v] = lev[u]+1;
43             }
44         }
45         return lev[t] >= 0;
46     }
47     F dfs(int v, int t, F flo) {
48         if (v == t) return flo;
49         for (; cur[v] != end(adj[v]); cur[v]++) {
50             Edge& E = eds[*cur[v]];
51             if (lev[E.to] != lev[v]+1 || E.flo == E.cap)
52                 ↪ continue;
53             F df =
54                 ↪ dfs(E.to, t, min(flo, E.cap-E.flo));
55             if (df) {
56                 E.flo += df;
57                 eds[*cur[v]^1].flo -= df;
58                 return df;
59             } // saturated >=1 one edge
60         }
61         return 0;
62     }
63     F maxFlow(int s, int t) {
64         F tot = 0;
65         while (bfs(s,t)) while (F df =
66             ↪ dfs(s,t,numeric_limits<F>::max()))
67             tot += df;

```

```

65         return tot;
66     }
67     int fp(int u, int t, F f, vector<int> &path,
68         ↪ vector<F> &flo, vector<int> &vis) {
69         vis[u] = 1;
70         if (u == t) {
71             path.pb(u);
72             return f;
73         }
74         for (auto eid: adj[u]) {
75             auto &e = eds[eid];
76             F w = e.flo - flo[eid];
77             if (w <= 0 || vis[e.to]) continue;
78             w = fp(e.to, t,
79                 min(w, f), path, flo, vis);
80             if (w) {
81                 flo[eid] += w, path.pb(u);
82                 return w;
83             }
84         }
85         return 0;
86     }
87     // return collection of {bottleneck, path[]}
88     vector<pair<F, vector<int>>> allPath(int s, int
89         ↪ t) {
90         vector<pair<F, vector<int>>> res; vector<F>
91             ↪ flo((int)eds.size());
92         vector<int> vis;
93         do res.pb(mp(0, vector<int>()));
94         while (res.back().first =
95             ↪ fp(s, t, numeric_limits<F>::max(),
96                 res.back().second, flo, vis=vector<int>(N))
97         );
98         for (auto &p: res) reverse(all(p.second));
99         return res.pop_back(), res;
100     }

```

7.2 MCMF

```

1  struct MCMF{
2      struct Edge{
3          int from, to;
4          int cap, cost;
5          Edge(int f, int t, int ca, int co): from(f),
6              ↪ to(t), cap(ca), cost(co) {}
7      };
8      int n, s, t;
9      vector<Edge> edges;
10     vector<vector<int>>> G;
11     vector<int> d;
12     vector<int> in_queue, prev_edge;
13     MCMF(){}
14     MCMF(int _n, int _s, int _t): n(_n), G(_n + 1),
15         ↪ d(_n + 1), in_queue(_n + 1), prev_edge(_n +
16         ↪ 1), s(_s), t(_t) {}
17     void addEdge(int u, int v, int cap, int cost){
18         G[u].push_back(edges.size());
19         edges.push_back(Edge(u, v, cap, cost));
20         G[v].push_back(edges.size());
21         edges.push_back(Edge(v, u, 0, -cost));
22     }
23     bool bfs(){

```

```

21 bool found = false;
22 fill(d.begin(), d.end(), (int)1e18+10);
23 fill(in_queue.begin(), in_queue.end(), false);
24 d[s] = 0;
25 in_queue[s] = true;
26 queue<int>q;
27 q.push(s);
28 while(!q.empty()){
29     int u = q.front();
30     q.pop();
31     if(u == t)
32         found = true;
33     in_queue[u] = false;
34     for(auto &id : G[u]){
35         Edge e = edges[id];
36         if(e.cap > 0 && d[u] + e.cost < d[e.to]){
37             d[e.to] = d[u] + e.cost;
38             prev_edge[e.to] = id;
39             if(!in_queue[e.to]){
40                 in_queue[e.to] = true;
41                 q.push(e.to);
42             }
43         }
44     }
45 }
46 return found;
47 }
48 pair<int, int>flow(){
49     // return (cap, cost)
50     int cap = 0, cost = 0;
51     while(bfs()){
52         int send = (int)1e18 + 10;
53         int u = t;
54         while(u != s){
55             Edge e = edges[prev_edge[u]];
56             send = min(send, e.cap);
57             u = e.from;
58         }
59         u = t;
60         while(u != s){
61             Edge &e = edges[prev_edge[u]];
62             e.cap -= send;
63             Edge &e2 = edges[prev_edge[u] ^ 1];
64             e2.cap += send;
65             u = e.from;
66         }
67         cap += send;
68         cost += send * d[t];
69     }
70     return make_pair(cap, cost);
71 }
72 };

```

8 Math

8.1 EXGCD

```

1 // ax + by = c
2 // return (gcd(a, b), x, y)
3 tuple<long long, long long, long long>exgcd(long
    ↪ long a, long long b){
4     if(b == 0)

```

```

5     return make_tuple(a, 1, 0);
6     auto [g, x, y] = exgcd(b, a % b);
7     return make_tuple(g, y, x - (a / b) * y);

```

8.2 DiscreteLog

```

1 int DiscreteLog(int s, int x, int y, int m) {
2     constexpr int kStep = 32000;
3     unordered_map<int, int> p;
4     int b = 1;
5     for (int i = 0; i < kStep; ++i) {
6         p[y] = i;
7         y = 1LL * y * x % m;
8         b = 1LL * b * x % m;
9     }
10    for (int i = 0; i < m + 10; i += kStep) {
11        s = 1LL * s * b % m;
12        if (p.find(s) != p.end()) return i + kStep -
            ↪ p[s];
13    }
14    return -1;
15 }
16 int DiscreteLog(int x, int y, int m) {
17     if (m == 1) return 0;
18     int s = 1;
19     for (int i = 0; i < 100; ++i) {
20         if (s == y) return i;
21         s = 1LL * s * x % m;
22     }
23     if (s == y) return 100;
24     int p = 100 + DiscreteLog(s, x, y, m);
25     if (fpow(x, p, m) != y) return -1;
26     return p;
27 }

```

8.3 EXCRT

```

1 long long inv(long long x){ return qpow(x, mod - 2,
    ↪ mod); }
2 long long mul(long long x, long long y, long long
    ↪ m){
3     x = ((x % m) + m) % m, y = ((y % m) + m) % m;
4     long long ans = 0;
5     while(y){
6         if(y & 1)
7             ans = (ans + x) % m;
8         x = x * 2 % m;
9         y >>= 1;
10    }
11    return ans;
12 }
13 pii ExCRT(long long r1, long long m1, long long r2,
    ↪ long long m2){
14     long long g, x, y;
15     tie(g, x, y) = exgcd(m1, m2);
16     if((r1 - r2) % g)
17         return {-1, -1};
18     long long lcm = (m1 / g) * m2;
19     long long res = (mul(mul(m1, x, lcm), ((r2 - r1)
    ↪ / g), lcm) + r1) % lcm;
20     res = (res + lcm) % lcm;

```

```

21 return {res, lcm};
22 }
23 void solve(){
24     long long n, r, m;
25     cin >> n;
26     cin >> m >> r; // x == r (mod m)
27     for(long long i = 1; i < n; i++){
28         long long r1, m1;
29         cin >> m1 >> r1;
30         if(r != -1 && m != -1)
31             tie(r, m) = ExCRT(r, m, r1, m1);
32     }
33     if(r == -1 && m == -1)
34         cout << "no solution\n";
35     else
36         cout << r << '\n';
37 }

```

8.4 FFT

```

1 struct Polynomial{
2     int deg;
3     vector<int>x;
4     void FFT(vector<complex<double>>&a, bool invert){
5         int a_sz = a.size();
6         for(int len = 1; len < a_sz; len <= 1){
7             for(int st = 0; st < a_sz; st += 2 * len){
8                 double angle = PI / len * (invert ? -1 :
9                     ↪ 1);
10                complex<double>wnow(1), w(cos(angle),
11                    ↪ sin(angle));
12                for(int i = 0; i < len; i++){
13                    auto a0 = a[st + i], a1 = a[st + len +
14                        ↪ i];
15                    a[st + i] = a0 + wnow * a1;
16                    a[st + i + len] = a0 - wnow * a1;
17                    wnow *= w;
18                }
19            }
20        }
21        if(invert)
22            for(auto &i : a)
23                i /= a_sz;
24    }
25    void change(vector<complex<double>>&a){
26        int a_sz = a.size();
27        vector<int>rev(a_sz);
28        for(int i = 1; i < a_sz; i++){
29            rev[i] = rev[i / 2] / 2;
30            if(i & 1)
31                rev[i] += a_sz / 2;
32        }
33        for(int i = 0; i < a_sz; i++)
34            if(i < rev[i])
35                swap(a[i], a[rev[i]]);
36    }
37    Polynomial multiply(Polynomial const&b){
38        vector<complex<double>>A(x.begin(), x.end()),
39            ↪ B(b.x.begin(), b.x.end());
40        int mx_sz = 1;
41        while(mx_sz < A.size() + B.size())
42            mx_sz <= 1;
43        A.resize(mx_sz);

```

```

40 B.resize(mx_sz);
41 change(A);
42 change(B);
43 FFT(A, 0);
44 FFT(B, 0);
45 for(int i = 0; i < mx_sz; i++)
46     A[i] *= B[i];
47 change(A);
48 FFT(A, 1);
49 Polynomial res(mx_sz);
50 for(int i = 0; i < mx_sz; i++)
51     res.x[i] = round(A[i].real());
52 while(!res.x.empty() && res.x.back() == 0)
53     res.x.pop_back();
54 res.deg = res.x.size();
55 return res;
56 }
57 Polynomial(): Polynomial(0) {}
58 Polynomial(int Size): x(Size), deg(Size) {}
59 };

```

8.5 NTT

```

1 /*
2  $p = r * 2^k + 1$ 
3  $p \quad r \quad k \quad root$ 
4 998244353    119 23 3
5 2013265921    15 27 31
6 2061584302081    15 37 7
7 */
8 template<int MOD, int RT>
9 struct NTT {
10     #define OP(op) static int op(int x, int y)
11     OP(add) { return (x += y) >= MOD ? x - MOD : x; }
12     ↪ }
13     OP(sub) { return (x -= y) < 0 ? x + MOD : x; }
14     OP(mul) { return ll(x) * y % MOD; } // multiply
15     ↪ by bit if p * p > 9e18
16     static int mpow(int a, int n) {
17         int r = 1;
18         while (n) {
19             if (n % 2) r = mul(r, a);
20             n /= 2, a = mul(a, a);
21         }
22         return r;
23     }
24     static const int MAXN = 1 << 21;
25     static int minv(int a) { return mpow(a, MOD -
26         ↪ 2); }
27     int w[MAXN];
28     NTT() {
29         int s = MAXN / 2, dw = mpow(RT, (MOD - 1) /
30             ↪ MAXN);
31         for (; s; s >>= 1, dw = mul(dw, dw)) {
32             w[s] = 1;
33             for (int j = 1; j < s; ++j)
34                 w[s + j] = mul(w[s + j - 1], dw);
35         }
36     }
37     void apply(vector<int>&a, int n, bool inv = 0)
38         ↪ {
39         for (int i = 0, j = 1; j < n - 1; ++j) {

```

```

35     for (int k = n >> 1; (i ^ k) < k; k
    ↪ >>= 1);
36     if (j < i) swap(a[i], a[j]);
37 }
38 for (int s = 1; s < n; s <= 1) {
39     for (int i = 0; i < n; i += s * 2) {
40         for (int j = 0; j < s; ++j) {
41             int tmp = mul(a[i + s + j], w[s
    ↪ + j]);
42             a[i + s + j] = sub(a[i + j],
    ↪ tmp);
43             a[i + j] = add(a[i + j], tmp);
44         }
45     }
46 }
47 if(!inv)
48 return;
49 int iv = minv(n);
50 if(n > 1)
51     reverse(next(a.begin()), a.end());
52     for (int i = 0; i < n; ++i)
53         a[i] = mul(a[i], iv);
54 }
55 vector<int>convolution(vector<int>&a,
    ↪ vector<int>&b){
56     int sz = a.size() + b.size() - 1, n = 1;
57     while(n <= sz)
58         n <= 1; // check n <= MAXN
59     vector<int>res(n);
60     a.resize(n), b.resize(n);
61     apply(a, n);
62     apply(b, n);
63     for(int i = 0; i < n; i++)
64         res[i] = mul(a[i], b[i]);
65     apply(res, n, 1);
66     return res;
67 }
68 };

```

8.6 MillerRain

```

1 bool is_prime(long long n, vector<long long> x) {
2     long long d = n - 1;
3     d >>= __builtin_ctzll(d);
4     for(auto a : x) {
5         if(n <= a) break;
6         long long t = d, y = 1, b = t;
7         while(b) {
8             if(b & 1) y = __int128(y) * a % n;
9             a = __int128(a) * a % n;
10            b >>= 1;
11        }
12        while(t != n - 1 && y != 1 && y != n - 1) {
13            y = __int128(y) * y % n;
14            t <= 1;
15        }
16        if(y != n - 1 && t % 2 == 0) return 0;
17    }
18    return 1;
19 }
20 bool is_prime(long long n) {
21     if(n <= 1) return 0;
22     if(n % 2 == 0) return n == 2;

```

```

23     if(n < (1LL << 30)) return is_prime(n, {2, 7,
    ↪ 61});
24     return is_prime(n, {2, 325, 9375, 28178, 450775,
    ↪ 9780504, 1795265022});
25 }

```

8.7 PollardRho

```

1 void PollardRho(map<long long, int>& mp, long long
    ↪ n) {
2     if(n == 1) return;
3     if(is_prime(n)) return mp[n]++, void();
4     if(n % 2 == 0) {
5         mp[2] += 1;
6         PollardRho(mp, n / 2);
7         return;
8     }
9     ll x = 2, y = 2, d = 1, p = 1;
10    #define f(x, n, p) ((__int128(x) * x % n + p) %
    ↪ n)
11    while(1) {
12        if(d != 1 && d != n) {
13            PollardRho(mp, d);
14            PollardRho(mp, n / d);
15            return;
16        }
17        p += (d == n);
18        x = f(x, n, p), y = f(f(y, n, p), n, p);
19        d = __gcd(abs(x - y), n);
20    }
21    #undef f
22 }
23 vector<long long> get_divisors(long long n) {
24     if(n == 0) return {};
25     map<long long, int> mp;
26     PollardRho(mp, n);
27     vector<pair<long long, int>> v(mp.begin(),
    ↪ mp.end());
28     vector<long long> res;
29     auto f = [&](auto f, int i, long long x) -> void
    ↪ {
30         if(i == (int)v.size()) {
31             res.pb(x);
32             return;
33         }
34         for(int j = v[i].second; ; j--) {
35             f(f, i + 1, x);
36             if(j == 0) break;
37             x *= v[i].first;
38         }
39     };
40     f(f, 0, 1);
41     sort(res.begin(), res.end());
42     return res;
43 }

```

8.8 XorBasis

```

1 template<int LOG> struct XorBasis {
2     bool zero = false;
3     int cnt = 0;

```



```

4  ll p[LOG] = {};
5  vector<ll> d;
6  void insert(ll x) {
7      for(int i = LOG - 1; i >= 0; --i) {
8          if(x >> i & 1) {
9              if(!p[i]) {
10                 p[i] = x;
11                 cnt += 1;
12                 return;
13             } else x ^= p[i];
14         }
15     }
16     zero = true;
17 }
18 ll get_max() {
19     ll ans = 0;
20     for(int i = LOG - 1; i >= 0; --i) {
21         if((ans ^ p[i]) > ans) ans ^= p[i];
22     }
23     return ans;
24 }
25 ll get_min() {
26     if(zero) return 0;
27     for(int i = 0; i < LOG; ++i) {
28         if(p[i]) return p[i];
29     }
30 }
31 bool include(ll x) {
32     for(int i = LOG - 1; i >= 0; --i) {
33         if(x >> i & 1) x ^= p[i];
34     }
35     return x == 0;
36 }
37 void update() {
38     d.clear();
39     for(int j = 0; j < LOG; ++j) {
40         for(int i = j - 1; i >= 0; --i) {
41             if(p[j] >> i & 1) p[j] ^= p[i];
42         }
43     }
44     for(int i = 0; i < LOG; ++i) {
45         if(p[i]) d.PB(p[i]);
46     }
47 }
48 ll get_kth(ll k) {
49     if(k == 1 && zero) return 0;
50     if(zero) k -= 1;
51     if(k >= (1LL << cnt)) return -1;
52     update();
53     ll ans = 0;
54     for(int i = 0; i < SZ(d); ++i) {
55         if(k >> i & 1) ans ^= d[i];
56     }
57     return ans;
58 }
59 };

```

8.9 XorGaussianElimination

```

1 pair<int, vector<bool>> GaussElimination(int n, int
   ↪ m) {
2     // m = # of variable, n = # of equation, return
   ↪ solution of system

```

```

3     // X[0][0] + X[0][1] ... + X[0][m - 1] = X[0][m]
4     // ... to X[n - 1]
5     // has solution => return solution, no solution
   ↪ => return empty vector
6     int sol_num = 1;
7     vector<int> where(m, -1);
8     for(int col = 0, row = 0; col < m && row < n;
   ↪ col++){
9         for(int i = row; i < n; i++){
10             if(X[i][col]){
11                 swap(X[i], X[row]);
12                 break;
13             }
14         }
15         if(!X[row][col]){
16             sol_num = 2;
17             continue;
18         }
19         where[col] = row;
20         for(int i = 0; i < n; i++){
21             if(i != row && X[i][col])
22                 X[i] ^= X[row];
23         }
24         row++;
25     }
26     vector<bool> ans(m, 0);
27     for (int i = 0; i < m; i++){ //
28         if (where[i] != -1)
29             ans[i] = (X[where[i]][m] ? 1 : 0);
30     }
31     for (int i = 0; i < n; i++) {
32         bool sum = X[i][m];
33         for (int j = 0; j < m; j++)
34             sum ^= (X[i][j] && ans[j]);
35         if(sum)
36             return make_pair(0, vector<bool>(0));
37     }
38     for (int i = 0; i < m; i++)
39         if (where[i] == -1)
40             sol_num = 2;
41     return make_pair(sol_num, ans);
42 }

```

8.10 Generating Functions

- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$

$$\begin{aligned}
 & - A(rx) \Rightarrow r^n a_n \\
 & - A(x) + B(x) \Rightarrow a_n + b_n \\
 & - A(x)B(x) \Rightarrow \sum_{i=0}^n a_i b_{n-i} \\
 & - A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k} \\
 & - xA(x)' \Rightarrow n a_n \\
 & - \frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^n a_i
 \end{aligned}$$

- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$

$$\begin{aligned}
 & - A(x) + B(x) \Rightarrow a_n + b_n \\
 & - A^{(k)}(x) \Rightarrow a_{n+k} \\
 & - A(x)B(x) \Rightarrow \sum_{i=0}^n n i a_i b_{n-i} \\
 & - A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} n i_1 i_2 \dots i_k a_{i_1} a_{i_2} \dots a_{i_k} \\
 & - xA(x) \Rightarrow n a_n
 \end{aligned}$$

- Special Generating Function

$$\begin{aligned}
 & - (1+x)^n = \sum_{i \geq 0} n i x^i \\
 & - \frac{1}{(1-x)^n} = \sum_{i \geq 0} i^n - 1 x^i
 \end{aligned}$$

8.11 Numbers

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups. $S(n, k) = S(n-1, k-1) + kS(n-1, k)$, $S(n, 1) = S(n, n) = 1$ $S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$ $x^n = \sum_{i=0}^n S(n, i)(x)_i$
- Catalan numbers $C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$, $\forall n \geq 0$
 $C_{n+1} = \sum_{i=0}^n C_i C_{n-i} = \frac{2(2n+1)}{n+2} C_n$, $C_0 = 1$
- Number of triangle when the longest edge is x (if two triangles are considered the same if they are congruent)
 - if x is even, then $f(x) = \frac{x \times (x+2)}{4}$
 - if x is odd, then $f(x) = \frac{(x+1)^2 - 1}{4}$
- Hockey-stick identity: $\sum_{i=0}^n \binom{i}{k} = \sum_{i=k}^n \binom{i}{k} = \binom{n+1}{k+1}$
- Vandermonde's identity: $\sum_{k_1+\dots+k_p=m} \binom{n_1}{k_1} \binom{n_2}{k_2} \dots \binom{n_p}{k_p} = \binom{n_1+\dots+n_p}{m}$
- Ways to choosing k number from $[n]$ such that no consecutive number: $\binom{n+k-1}{k}$
- $\sum_{k=0}^n \binom{n}{k} = \binom{2n}{n}$

8.12 Theorem

- Cayley's Formula
 - Given a degree sequence d_1, d_2, \dots, d_n for each *labeled* vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$ spanning trees.
 - Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.
- Erdős–Gallai theorem A sequence of nonnegative integers $d_1 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \dots + d_n$ is even and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ holds for every $1 \leq k \leq n$.
- Gale–Ryser theorem A pair of sequences of nonnegative integers $a_1 \geq \dots \geq a_n$ and b_1, \dots, b_n is bigraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.
- Flooring and Ceiling function identity
 - $\lfloor \frac{\lfloor \frac{a}{b} \rfloor}{c} \rfloor = \lfloor \frac{a}{bc} \rfloor$
 - $\lceil \frac{\lceil \frac{a}{b} \rceil}{c} \rceil = \lceil \frac{a}{bc} \rceil$
 - $\lceil \frac{a}{b} \rceil \leq \frac{a+b-1}{b}$
 - $\lfloor \frac{a}{b} \rfloor \leq \frac{a-b+1}{b}$
- Möbius inversion formula
 - $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$
 - $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

$$\begin{aligned} - \sum_{d|n}^{n=1} \mu(d) &= 1 \\ - \sum_{d|n}^{n \neq 1} \mu(d) &= 0 \end{aligned}$$

- Spherical cap
 - A portion of a sphere cut off by a plane.
 - r : sphere radius, a : radius of the base of the cap, h : height of the cap, θ : $\arcsin(a/r)$.
 - Volume $= \pi h^2(3r-h)/3 = \pi h(3a^2+h^2)/6 = \pi r^3(2+\cos\theta)(1-\cos\theta)^2/3$.
 - Area $= 2\pi r h = \pi(a^2+h^2) = 2\pi r^2(1-\cos\theta)$.