Codebook

November 6, 2023

Contents

1 Setup

1.1 Template

```cpp
#include <bits/stdc++.h>
#include <bits/extc++.h>
#define F first
#define S second
#define pb push_back
#define pob pop_back
#define pf push_front
#define pof pop_front
#define mp make_pair
#define mt make_tuple
#define all(x) (x).begin(),(x).end()
using namespace std;
//using namespace __gnu_pbds;
using pii = pair<long long,long long>;
using ld = long double;
using ll = long long;
mt19937 mtrd(chrono::steady_clock::now() \
.time_since_epoch().count());
const int mod = 1000000007;
const int mod2 = 998244353;
const ld PI = acos(-1);
#define Bint __int128
#define int long long
template <typename T>
inline void printv(T l, T r){
  cerr << "[ ";
  for(; l != r; l++)
    cerr << *l << ", ";
  cerr << "]" << endl;
}
#define TEST
#ifdef TEST
#define de(x) cerr << #x << '=' << x << ", "
#define ed cerr << '\n';
#else
```

```
#define de(x) void(0)
#define ed void(0)
#define printv(...) void(0)
#endif
/* --------------------------------------- */
void solve(){
}
signed main(){
  ios::sync_with_stdio(0);
  cin.tie(0);
  int t = 1;
  // cin >> t;
  while(t--)
    solve();
}
```

1.2 Template$_ruru$

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
#define V vector
#define sz(a) ((int)a.size())
#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()
#define pb push_back
#define rsz resize
#define mp make_pair
#define mt make_tuple
#define ff first
#define ss second
#define FOR(i,j,k) for (int i=(j); i<=(k); i++)
#define FOR(i,j,k) for (int i=(j); i<(k); i++)
#define REP(i) FOR(_,1,i)
#define foreach(a,x) for (auto& a: x)
template<class T> bool cmin(T& a, const T& b) {
    return b < a ? a = b, 1 : 0; } // set a = min(a,b)
template<class T> bool cmax(T& a, const T& b) {
    return a < b ? a = b, 1 : 0; } // set a = max(a,b)
ll cdiv(ll a, ll b) { return a/b+((a^b)>0&&a%b); }
ll fdiv(ll a, ll b) { return a/b-((a^b)<0&&a%b); }
#define roadroller ios::sync_with_stdio(0), cin.tie(0);
#define de(x) cerr << #x << '=' << x << ", "
#define dd cerr << '\n';
```

1.3 vimrc

```
syntax on
set mouse=a
set nu
set tabstop=4
set softtabstop=4
set shiftwidth=4
set autoindent
set cursorline
imap kj <Esc>
imap {}} {<CR>}<Esc>ko<Tab>
imap [] []<Esc>i
imap () ()<Esc>i
imap <> <><Esc>i
```

2 Data-structure

2.1 PBDS

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

2.2 SparseTable

```
template <class T> struct SparseTable{
  // idx: [0, n - 1]
  int n;
  T id;
  vector<vector<T>>tbl;
  T op(T lhs, T rhs){
    // write your mege function
  }
  T query(int l, int r){
    int lg = __lg(r - l + 1);
    return op(tbl[lg][l], tbl[lg][r - (1 << lg) + 1]);
  }
  SparseTable (): n(0) {}
  template<typename iter_t>
  SparseTable (int _n, iter_t l, iter_t r, T _id) {
    n = _n;
    id = _id;
    int lg = __lg(n) + 2;
    tbl.resize(lg, vector<T>(n + 5, id));
    iter_t ptr = l;
    for(int i = 0; i < n; i++, ptr++){
      assert(ptr != r);
      tbl[0][i] = *ptr;
    }
    for(int i = 1; i <= lg; i++)
      for(int j = 0; j + (1 << (i - 1)) < n; j++)
        tbl[i][j] = op(tbl[i - 1][j], tbl[i - 1][j + (1 << (i - 1))]);
  }
};
```

2.3 SegmentTree

```
template <class T> struct Segment_tree{
  int L, R;
  T id;
  vector<T>seg;
```

```
  T op(T lhs, T rhs){
    // write your merge function
  }
  void _modify(int p, T v, int l, int r, int idx = 1){
    assert(p <= r && p >= l);
    if(l == r){
      seg[idx] = v;
      return;
    }
    int mid = (l + r) >> 1;
    if(p <= mid)
      _modify(p, v, l, mid, idx << 1);
    else
      _modify(p, v, mid + 1, r, idx << 1 | 1);
    seg[idx] = op(seg[idx << 1], seg[idx << 1 | 1]);
  }
  T _query(int ql, int qr, int l, int r, int idx = 1){
    if(ql == l && qr == r)
      return seg[idx];
    int mid = (l + r) >> 1;
    if(qr <= mid)
      return _query(ql, qr, l, mid, idx << 1);
    else if(ql > mid)
      return _query(ql, qr, mid + 1, r, idx << 1 | 1);
    return op(_query(ql, mid, l, mid, idx << 1), _query(mid + 1, qr, mid + 1, r, idx << 1 | 1));
  }
  void modify(int p, T v){ _modify(p, v, L, R, 1); }
  T query(int l, int r){ return _query(l, r, L, R, 1); }
  Segment_tree(): Segment_tree(0, 0, 0) {}
  Segment_tree(int l, int r, T _id): L(l), R(r) {
    id = _id;
    seg.resize(4 * (r - l + 10));
    fill(seg.begin(), seg.end(), id);
  }
};
```

2.4 LazyTagSegtree

```
template<class T, int SZ> struct LazySeg { // SZ must be power of 2
  // depends
  T tID, ID;
  T seg[SZ * 2], lazy[SZ * 2];
  T cmb(T a, T b) {
    return max(a, b);
  }
  LazySeg(T id, T tid): ID(id), tID(tid) {
    for(int i = 0; i < SZ * 2; i++)
      seg[i] = ID, lazy[id] = tID;
  }
  void addtag(int l, int r, int ind, int v){
    if(lazy[ind] == tID)
      lazy[ind] = v;
    else
      lazy[ind] += v;
  }
  /// modify values for current node
  void push(int ind, int L, int R) {
    // dependent on operation
    if(lazy[ind] == tID)
      return;
    seg[ind] += lazy[ind];
    if(L != R){
      int mid = (L + R) >> 1;
      addtag(L, mid, ind << 1, lazy[ind]);
      addtag(mid + 1, R, ind << 1 | 1, lazy[ind]);
    }
    lazy[ind] = tID;
  }
  void pull(int ind){
    seg[ind] = cmb(seg[ind << 1], seg[ind << 1 | 1]);
  }
  void upd(int lo, int hi, T v, int ind = 1, int L = 0, int R = SZ - 1) {
    push(ind, L, R);
    if (hi < L || R < lo) return;
    if (lo <= L && R <= hi) {
      addtag(L, R, ind, v);
      push(ind, L, R); return;
    }
    int mid = (L + R) >> 1;
    upd(lo, hi, v, ind << 1, L, mid);
    upd(lo, hi, v, ind << 1 | 1, mid + 1, R);
    pull(ind);
  }
  T query(int lo, int hi, int ind = 1, int L = 0, int R = SZ - 1) {
    push(ind, L, R);
    if (lo > R || L > hi) return ID;
    if (lo <= L && R <= hi) return seg[ind];
    int mid = (L + R) >> 1;
    return cmb(query(lo, hi, ind << 1, L, mid),
      query(lo, hi, ind << 1 | 1, mid + 1, R));
  }
};
```

2.5 LiChaoTree

```
struct line{
  int m, c;
  int val(int x){
    return m * x + c;
  }
  line(): m(_id), c(0) {} // _id is the identity element
  line(int _m, int _c): m(_m), c(_c) {}
};
struct Li_Chao_Tree{
  line seg[N << 2];
  void ins(int l, int r, int idx, line x){
    if(l == r){
      if(x.val(l) > seg[idx].val(l))
        seg[idx] = x; // change > to < when get min
      return;
    }
    int mid = (l + r) >> 1;
    if(x.m < seg[idx].m) // change < to > when get min
```

```
      swap(x, seg[idx]);
    if(seg[idx].val(mid) <= x.val(mid)){
      // change <= to >= when get min
      swap(x, seg[idx]);
      ins(l, mid, idx << 1, x);
    }
    else
      ins(mid + 1, r, idx << 1 | 1, x);
  }
  int query(int l, int r, int p, int idx){
    if(l == r)
      return seg[idx].val(l);
    int mid = (l + r) >> 1;
    // change max to min when get min
    if(p <= mid)
      return max(seg[idx].val(p), query(l, mid, p, idx << 1));
    else
      return max(seg[idx].val(p), query(mid + 1, r, p, idx << 1 | 1));
  }
}
```

2.6 Treap

```
struct Treap{
  Treap *l, *r;
  int pri, key, sz;
  Treap(){}
  Treap(int _v){
    l = r = NULL;
    pri = mtrd();
    key = _v;
    sz = 1;
  }
  ~Treap(){
        if ( l )
            delete l;
        if ( r )
            delete r;
    }
  void push(){
    for(auto ch : {l, r}){
      if(ch){
        // do something
      }
    }
  }
};
int getSize(Treap *t){
  return t ? t->sz : 0;
}
void pull(Treap *t){
  t->sz = getSize(t->l) + getSize(t->r) + 1;
}
Treap* merge(Treap* a, Treap* b){
  if(!a || !b)
    return a ? a : b;
  if(a->pri > b->pri){
    a->push();
    a->r = merge(a->r, b);
    pull(a);
    return a;
  }
  else{
    b->push();
    b->l = merge(a, b->l);
    pull(b);
    return b;
  }
}
void splitBySize(Treap *t, Treap *&a, Treap *&b, int k){
  if(!t)
    a = b = NULL;
  else if(getSize(t->l) + 1 <= k){
    a = t;
    a->push();
    splitBySize(t->r, a->r, b, k - getSize(t->l) - 1);
    pull(a);
  }
  else{
    b = t;
    b->push();
    splitBySize(t->l, a, b->l, k);
    pull(b);
  }
}
void splitByKey(Treap *t, Treap *&a, Treap *&b, int k){
    if(!t)
        a = b = NULL;
    else if(t->key <= k){
        a = t;
        a->push();
        splitByKey(t->r, a->r, b, k);
        pull(a);
    }
    else{
        b = t;
        b->push();
        splitByKey(t->l, a, b->l, k);
        pull(b);
    }
}
// O(n) build treap with sorted key nodes
void traverse(Treap *t){
  if(t->l)
    traverse(t->l);
  if(t->r)
    traverse(t->r);
  pull(t);
}
Treap *build(int n){
  vector<Treap*>st(n);
  int tp = 0;
  for(int i = 0, x; i < n; i++){
    cin >> x;
    Treap *nd = new Treap(x);
    while(tp && st[tp - 1]->pri < nd->pri)
      nd->l = st[tp - 1], tp--;
    if(tp)
      st[tp - 1]->r = nd;
    st[tp++] = nd;
  }
  if(!tp){
    st[0] = NULL;
```

```
    return st[0];
  }
  traverse(st[0]);
  return st[0];
}
```

2.7 DSU

```
struct Disjoint_set{
  int n;
  vector<int>sz, p;
  int fp(int x){
    return (p[x] == -1 ? x : p[x] = fp(p[x]));
  }
  bool U(int x, int y){
    x = fp(x), y = fp(y);
    if(x == y)
      return false;
    if(sz[x] > sz[y])
      swap(x, y);
    p[x] = y;
    sz[y] += sz[x];
    return true;
  }
  Disjoint_set() {}
  Disjoint_set(int _n){
    n = _n;
    sz.resize(n + 5, 1);
    p.resize(n + 5, -1);
  }
};
```

2.8 RollbackDSU

```
struct Rollback_DSU{
  vector<int>p, sz;
  vector<pair<int, int>>history;
  int fp(int x){
    while(p[x] != -1)
      x = p[x];
    return x;
  }
  bool U(int x, int y){
    x = fp(x), y = fp(y);
    if(x == y){
      history.push_back(make_pair(-1, -1));
      return false;
    }
    if(sz[x] > sz[y])
      swap(x, y);
    p[x] = y;
    sz[y] += sz[x];
    history.push_back(make_pair(x, y));
    return true;
  }
  void undo(){
    if(history.empty() || history.back().first == -1){
      if(!history.empty())
        history.pop_back();
      return;
    }
    auto [x, y] = history.back();
    history.pop_back();
    p[x] = -1;
    sz[y] -= sz[x];
  }
  Rollback_DSU(): Rollback_DSU(0) {}
  Rollback_DSU(int n): p(n + 5), sz(n + 5) {
    fill(p.begin(), p.end(), -1);
    fill(sz.begin(), sz.end(), 1);
  }
};
```

3 Graph

3.1 RoundSquareTree

```
int cnt;
int dep[N], low[N]; // dep == -1 -> unvisited
vector<int>G[N], rstree[2 * N]; // 1 ~ n: round, n + 1 ~ 2n: square
vector<int>stk;
void init(){
    cnt = n;
    for(int i = 1; i <= n; i++){
        G[i].clear();
        rstree[i].clear();
        rstree[i + n].clear();
        dep[i] = low[i] = -1;
    }
    dep[1] = low[1] = 0;
}
void tarjan(int x, int px){
    stk.push_back(x);
    for(auto i : G[x]){
        if(dep[i] == -1){
            dep[i] = low[i] = dep[x] + 1;
            tarjan(i, x);
            low[x] = min(low[x], low[i]);
            if(dep[x] <= low[i]){
                int z;
        cnt++;
                do{
                    z = stk.back();
                    rstree[cnt].push_back(z);
                    rstree[z].push_back(cnt);
                    stk.pop_back();
                }while(z != i);
                rstree[cnt].push_back(x);
                rstree[x].push_back(cnt);
            }
        }
        else if(i != px)
            low[x] = min(low[x], dep[i]);
    }
}
```

3.2 SCC

```
struct SCC{
  int n;
  int cnt;
  vector<vector<int>>G, revG;
  vector<int>stk, sccid;
  vector<bool>vis;
  SCC(): SCC(0) {}
  SCC(int _n): n(_n), G(_n + 1), revG(_n + 1), sccid(_n + 1), vis(_n + 1), cnt(0) {}
  void addEdge(int u, int v){
    // u -> v
    assert(u > 0 && u <= n);
    assert(v > 0 && v <= n);
    G[u].push_back(v);
    revG[v].push_back(u);
  }
  void dfs1(int u){
    vis[u] = 1;
    for(int v : G[u]){
      if(!vis[v])
        dfs1(v);
    }
    stk.push_back(u);
  }
  void dfs2(int u, int k){
    vis[u] = 1;
    sccid[u] = k;
    for(int v : revG[u]){
      if(!vis[v])
        dfs2(v, k);
    }
  }
  void Kosaraju(){
    for(int i = 1; i <= n; i++)
      if(!vis[i])
        dfs1(i);
    fill(vis.begin(), vis.end(), 0);
    while(!stk.empty()){
      if(!vis[stk.back()])
        dfs2(stk.back(), ++cnt);
      stk.pop_back();
    }
  }
};
```

3.3 2SAT

```
struct two_sat{
  int n;
  SCC G; // u: u, u + n: ~u
  vector<int>ans;
  two_sat(): two_sat(0) {}
  two_sat(int _n): n(_n), G(2 * _n), ans(_n + 1) {}
  void disjunction(int a, int b){
    G.addEdge((a > n ? a - n : a + n), b);
    G.addEdge((b > n ? b - n : b + n), a);
  }
  bool solve(){
    G.Kosaraju();
    for(int i = 1; i <= n; i++){
      if(G.sccid[i] == G.sccid[i + n])
        return false;
      ans[i] = (G.sccid[i] > G.sccid[i + n]);
    }
    return true;
  }
};
```

3.4 Bridge

```
int dep[N], low[N];
vector<int>G[N];
vector<pair<int, int>>bridge;
void init(){
  for(int i = 1; i <= n; i++){
    G[i].clear();
    dep[i] = low[i] = -1;
  }
  dep[1] = low[1] = 0;
}
void tarjan(int x, int px){
  for(auto i : G[x]){
    if(dep[i] == -1){
      dep[i] = low[i] = dep[x] + 1;
      tarjan(i, x);
      low[x] = min(low[x], low[i]);
      if(low[i] > dep[x])
        bridge.push_back(make_pair(i, x));
    }
    else if(i != px)
      low[x] = min(low[x], dep[i]);
  }
}
```

3.5 BronKerboschAlgorithm

```
vector<vector<int>>maximal_clique;
int cnt, G[N][N], all[N][N], some[N][N], none[N][N];
void dfs(int d, int an, int sn, int nn)
{
    if(sn == 0 && nn == 0){
    vector<int>v;
    for(int i = 0; i < an; i++)
      v.push_back(all[d][i]);
    maximal_clique.push_back(v);
    cnt++;
    }
  int u = sn > 0 ? some[d][0] : none[d][0];
    for(int i = 0; i < sn; i ++)
    {
        int v = some[d][i];
        if(G[u][v])
      continue;
        int tsn = 0, tnn = 0;
        for(int j = 0; j < an; j ++)
      all[d + 1][j] = all[d][j];
        all[d + 1][an] = v;
        for(int j = 0; j < sn; j ++)
            if(g[v][some[d][j]])
        some[d + 1][tsn ++] = some[d][j];
```

```
        for(int j = 0; j < nn; j ++)
            if(g[v][none[d][j]])
        none[d + 1][tnn ++] = none[d][j];
        dfs(d + 1, an + 1, tsn, tnn);
        some[d][i] = 0, none[d][nn ++] = v;
    }
}
void process(){
    cnt = 0;
    for(int i = 0; i < n; i ++)
    some[0][i] = i + 1;
    dfs(0, 0, n, 0);
}
```

3.6 Theorem

- Kosaraju's algorithm visit the strong connected components in topolocical 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

```
/**
 * Description: Heavy-Light Decomposition, add val to verts
   * and query sum in path/subtree.
 * Time: any tree path is split into O(log N) parts
 */
// #include "LazySeg.h"
template<int SZ, bool VALS_IN_EDGES> struct HLD {
  int N; vi adj[SZ];
  int par[SZ], root[SZ], depth[SZ], sz[SZ], ti;
  int pos[SZ]; vi rpos;
  // rpos not used but could be useful
  void ae(int x, int y) {
    adj[x].pb(y), adj[y].pb(x);
  }
  void dfsSz(int x) {
    sz[x] = 1;
    foreach(y, adj[x]) {
      par[y] = x; depth[y] = depth[x]+1;
      adj[y].erase(find(all(adj[y]),x));
      /// remove parent from adj list
      dfsSz(y); sz[x] += sz[y];
      if (sz[y] > sz[adj[x][0]])
        swap(y,adj[x][0]);
    }
  }
  void dfsHld(int x) {
    pos[x] = ti++; rpos.pb(x);
    foreach(y,adj[x]) {
      root[y] =
        (y == adj[x][0] ? root[x] : y);
      dfsHld(y); }
  }
  void init(int _N, int R = 0) { N = _N;
    par[R] = depth[R] = ti = 0; dfsSz(R);
    root[R] = R; dfsHld(R);
  }
  int lca(int x, int y) {
    for (; root[x] != root[y]; y = par[root[y]])
      if (depth[root[x]] > depth[root[y]]) swap(x,y);
    return depth[x] < depth[y] ? x : y;
  }
  /// int dist(int x, int y) { // # edges on path
  ///   return depth[x]+depth[y]-2*depth[lca(x,y)]; }
  LazySeg<ll,SZ> tree; // segtree for sum
  template <class BinaryOp>
  void processPath(int x, int y, BinaryOp op) {
    for (; root[x] != root[y]; y = par[root[y]]) {
      if (depth[root[x]] > depth[root[y]]) swap(x,y);
      op(pos[root[y]],pos[y]); }
    if (depth[x] > depth[y]) swap(x,y);
    op(pos[x]+VALS_IN_EDGES,pos[y]);
  }
  void modifyPath(int x, int y, int v) {
    processPath(x,y,[this,&v](int l, int r) {
      tree.upd(l,r,v); });
  }
  ll queryPath(int x, int y) {
    ll res = 0;
    processPath(x,y,[this,&res](int l, int r) {
      res += tree.query(l,r); });
    return res;
  }
  void modifySubtree(int x, int v) {
    tree.upd(pos[x]+VALS_IN_EDGES,pos[x]+sz[x]-1,v);
  }
};
```

4.2 LCA

```
int anc[20][N];
int dis[20][N];
int dep[N];
vector<pair<int, int>>G[N]; // weighted(edge) tree
void dfs(int u, int pu = 0){
  for(int i = 1; i < 20; i++){
    anc[i][u] = anc[i - 1][anc[i - 1][u]];
    dis[i][u] = dis[i - 1][u] + dis[i - 1][anc[i - 1][u]];
  }
  for(auto [v, c] : G[u]){
    if(v == pu)
      continue;
```

```
    dep[v] = dep[u] + 1;
    anc[0][v] = u;
    dis[0][v] = c;
    dfs(v, u);
  }
}
int LCA(int x, int y){
  if(dep[x] < dep[y])
    swap(x, y);
  int diff = dep[x] - dep[y];
  for(int i = 19; i >= 0; i--){
    if(diff - (1 << i) >= 0)
      x = anc[i][x], diff -= (1 << i);
  }
  if(x == y)
    return x;
  for(int i = 19; i >= 0; i--){
    if(anc[i][x] != anc[i][y]){
      x = anc[i][x];
      y = anc[i][y];
    }
  }
  return anc[0][x];
}
```

5 Geometry

5.1 Point

```
template<class T> struct Point {
  T x, y;
  Point(): x(0), y(0) {};
  Point(T a, T b): x(a), y(b) {};
  Point(pair<T, T>p): x(p.first), y(p.second) {};
  Point operator + (const Point& rhs){ return Point(x + rhs.x, y + rhs.y); }
  Point operator - (const Point& rhs){ return Point(x - rhs.x, y - rhs.y); }
  Point operator * (const int& rhs){ return Point(x * rhs, y * rhs); }
  Point operator / (const int& rhs){ return Point(x / rhs, y / rhs); }
  T cross(Point rhs){ return x * rhs.y - y * rhs.x; }
  T dot(Point rhs){ return x * rhs.x + y * rhs.y; }
  T cross2(Point a, Point b){ // (a - this) cross (b - this)
    return (a - *this).cross(b - *this);
  }
  T dot2(Point a, Point b){ // (a - this) dot (b - this)
    return (a - *this).dot(b - *this);
  }
};
```

5.2 Geometry

```
template<class T> int ori(Point<T>a, Point<T>b, Point<T>c){
  // sign of (b - a) cross(c - a)
  auto res = a.cross2(b, c);
  // if type if double
  // if(abs(res) <= eps)
  if(res == 0)
    return 0;
  return res > 0 ? 1 : -1;
}
template<class T> bool collinearity(Point<T>a, Point<T>b, Point<T>c){
  // if type is double
  // return abs(c.cross2(a,b)) <= eps;
  return c.cross2(a, b) == 0;
}
template<class T> bool between(Point<T>a, Point<T>b, Point<T>c){
  // check if c is between a, b
  return collinearity(a, b, c) && c.dot2(a, b) <= 0;
}
template<class T> bool seg_intersect(Point<T>p1, Point<T>p2, Point<T>p3, Point<T>p4){
  // seg (p1, p2), seg(p3, p4)
  int a123 = ori(p1, p2, p3);
  int a124 = ori(p1, p2, p4);
  int a341 = ori(p3, p4, p1);
  int a342 = ori(p3, p4, p2);
  if(a123 == 0 && a124 == 0)
    return between(p1, p2, p3) || between(p1, p2, p4) || between(p3, p4, p1) || between(p3, p4, p2);
  return a123 * a124 <= 0 && a341 * a342 <= 0;
}
template<class T> Point<T> intersect_at(Point<T> a, Point<T> b, Point<T> c, Point<T> d) {
  // line(a, b), line(c, d)
  T a123 = a.cross(b, c);
  T a124 = a.cross(b, d);
  return (d * a123 - c * a124) / (a123 - a124);
}
template<class T> int point_in_convex_polygon(vector<Point<T>>& a, Point<T>p){
  // 1: IN
  // 0: OUT
  // -1: ON
  // the points of convex polygon must sort in counter-clockwise order
  int n = a.size();
  if(between(a[0], a[1], p) || between(a[0], a[n - 1], p))
    return -1;
  int l = 0, r = n - 1;
  while(l <= r){
    int mid = (l + r) >> 1;
    auto a1 = a[0].cross2(a[mid], p);
    auto a2 = a[0].cross2(a[(mid + 1) % n], p);
    if(a1 >= 0 && a2 <= 0){
      auto res = a[mid].cross2(a[(mid + 1) % n], p);
      return res > 0 ? 1 : (res >= 0 ? -1 : 0);
    }
    else if(a1 < 0)
      r = mid - 1;
    else
      l = mid + 1;
```

```
  }
  return 0;
}
template<class T> int point_in_simple_polygon(vector<Point<T>>&a, Point<T>p, Point<T>INF_point){
  // 1: IN
  // 0: ON
  // -1: OUT
  // a[i] must adjacent to a[(i + 1) % n] for all i
  // collinearity(a[i], p, INF_point) must be false for all i
  // we can let the slope of line(p, INF_point) be irrational (e.g. PI)
  int ans = -1;
  for(auto l = prev(a.end()), r = a.begin(); r != a.end(); l = r++){
    if(between(*l, *r, p))
      return 0;
    if(seg_intersect(*l, *r, p, INF_point)){
      ans *= -1;
      if(collinearity(*l, p, INF_point))
        assert(0);
    }
  }
  return ans;
}
template<class T> T area(vector<Point<T>>&a){
  // remember to divide 2 after calling this function
  if(a.size() <= 1)
    return 0;
  T ans = 0;
  for(auto  l = prev(a.end()), r = a.begin(); r != a.end(); l = r++)
    ans += l->cross(*r);
  return abs(ans);
}
```

5.3 ConvexHull

```
template<class T> vector<Point<T>> convex_hull(vector<Point<T>>&a){
  int n  = a.size();
  sort(a.begin(), a.end(), [](Point<T>p1, Point<T>p2){
    if(p1.x == p2.x)
      return p1.y < p2.y;
    return p1.x < p2.x;
  });
  int m = 0, t = 1;
  vector<Point<T>>ans;
  auto addPoint = [&](const Point<T>p) {
    while(m > t && ans[m - 2].cross2(ans[m - 1], p) <= 0)
      ans.pop_back(), m--;
    ans.push_back(p);
    m++;
  };
  for(int i = 0; i < n; i++)
    addPoint(a[i]);
  t = m;
  for(int i = n - 2; ~i; i--)
    addPoint(a[i]);
  if(a.size() > 1)
    ans.pop_back();
  return ans;
}
```

5.4 MaximumDistance

```
template<class T>
T MaximumDistance(vector<Point<T>>&p){
  vector<Point<T>>C = convex_hull(p);
  int n = C.size(),t = 2;
  T ans = 0;
  for(int i = 0;i<n;i++){
    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;
    ans = max({ans, abs2(C[i] - C[t]), abs2(C[(i+1)%n] - C[t])});
  }
  return ans;
}
```

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

```
struct Rolling_Hash{
  int n;
  const int P[5] = {146672737, 204924373, 585761567, 484547929, 116508269};
  const int M[5] = {922722049, 952311013, 955873937, 901981687, 993179543};
  vector<int>PW[5], pre[5], suf[5];
  Rolling_Hash(): Rolling_Hash("") {}
  Rolling_Hash(string s): n(s.size()){
    for(int i = 0; i < 5; i++){
      PW[i].resize(n), pre[i].resize(n), suf[i].resize(n);
      PW[i][0] = 1, pre[i][0] = s[0];
      suf[i][n - 1] = s[n - 1];
    }
    for(int i = 1; i < n; i++){
      for(int j = 0; j < 5; j++){
        PW[j][i] = PW[j][i - 1] * P[j] % M[j];
        pre[j][i] = (pre[j][i - 1] * P[j] + s[i]) % M[j];
      }
    }
    for(int i = n - 2; i >= 0; i--){
```

```
      for(int j = 0; j < 5; j++)
        suf[j][i] = (suf[j][i + 1] * P[j] + s[i]) % M[j];
    }
  }
  int _substr(int k, int l, int r) {
    int res = pre[k][r];
    if(l > 0)
      res -= 1LL * pre[k][l - 1] * PW[k][r - l + 1] % M[k];
    if(res < 0)
      res += M[k];
    return res;
  }
  vector<int>substr(int l, int r){
    vector<int>res(5);
    for(int i = 0; i < 5; ++i)
      res[i] = _substr(i, l, r);
    return res;
  }
};
```

6.2 SuffixArray

```
struct Suffix_Array{
  int n, m; // m is the range of s
  string s;
  vector<int>sa, rk, lcp;
  // sa[i]: the i-th smallest suffix
  // rk[i]: the rank of suffix i (i.e. s[i, n - 1])
  // lcp[i]: the longest common prefix of sa[i] and sa[i - 1]
  Suffix_Array(): Suffix_Array(0, 0, "") {};
  Suffix_Array(int _n, int _m, string _s): n(_n), m(_m), sa(_n), rk(_n), lcp(_n), s(_s) {}
  void Sort(int k, vector<int>&bucket, vector<int>&idx, vector<int>&lst){
    for(int i = 0; i < m; i++)
      bucket[i] = 0;
    for(int i = 0; i < n; i++)
      bucket[lst[i]]++;
    for(int i = 1; i < m; i++)
      bucket[i] += bucket[i-1];
    int p = 0;
    // update index
    for(int i = n - k; i < n; i++)
      idx[p++] = i;
    for(int i = 0; i < n; i++)
      if(sa[i] >= k)
        idx[p++] = sa[i] - k;
    for(int i = n - 1; i >= 0; i--)
      sa[--bucket[lst[idx[i]]]] = idx[i];
  }
  void build(){
    vector<int>idx(n), lst(n), bucket(max(n, m));
    for(int i = 0; i < n; i++)
      bucket[lst[i] = (s[i] - 'a')]++; // may change
    for(int i = 1; i < m; i++)
      bucket[i] += bucket[i - 1];
    for(int i = n - 1; i >= 0; i--)
      sa[--bucket[lst[i]]] = i;
    for(int k = 1; k < n; k <<= 1){
      Sort(k, bucket, idx, lst);
      // update rank
      int p = 0;
      idx[sa[0]] = 0;
      for(int i = 1; i < n; i++){
        int a = sa[i], b = sa[i - 1];
        if(lst[a] == lst[b] && a + k < n && b + k < n && lst[a + k] == lst[b + k]);
        else
          p++;
        idx[sa[i]] = p;
      }
      if(p == n - 1)
        break;
      for(int i = 0; i < n; i++)
        lst[i] = idx[i];
      m = p + 1;
    }
    for(int i = 0; i < n; i++)
      rk[sa[i]] = i;
    buildLCP();
  }
  void buildLCP(){
    // lcp[rk[i]] >= lcp[rk[i - 1]] - 1
    int v = 0;
    for(int i = 0; i < n; i++){
      if(!rk[i])
        lcp[rk[i]] = 0;
      else{
        if(v)
          v--;
        int p = sa[rk[i] - 1];
        while(i + v < n && p + v < n && s[i + v] == s[p + v])
          v++;
        lcp[rk[i]] = v;
      }
    }
  }
};
```

6.3 KMP

```
struct KMP {
  int n;
  string s;
  vector<int>fail;
  // s: pattern, t: text => find s in t
  int match(string &t){
    int ans = 0, m = t.size(), j = -1;
    for(int i = 0; i < m; i++){
      while(j != -1 && t[i] != s[j + 1])
        j = fail[j];
      if(t[i] == s[j + 1])
        j++;
      if(j == n - 1){
        ans++;
        j = fail[j];
      }
    }
    return ans;
  }
  KMP(string &_s){
```

```
    s = _s;
    n = s.size();
    fail = vector<int>(n, -1);
    int j = -1;
    for(int i = 1; i < n; i++){
      while(j != -1 && s[i] != s[j + 1])
        j = fail[j];
      if(s[i] == s[j + 1])
        j++;
      fail[i] = j;
    }
  }
};
```

6.4 Trie

```
struct Node {
  int hit = 0;
  Node *next[26];
  // 26 is the size of the set of characters
  // a - z
  Node(){
    for(int i = 0; i < 26; i++)
      next[i] = NULL;
  }
};
void insert(string &s, Node *node){
  // node cannot be null
  for(char v : s){
    if(node->next[v - 'a'] == NULL)
      node->next[v - 'a'] = new Node;
    node = node->next[v - 'a'];
  }
  node->hit++;
}
```

6.5 Zvalue

```
struct Zvalue {
  const string inf = "$"; // character that has never used
  vector<int>z;
  // s: pattern, t: text => find s in t
  int match(string &s, string &t){
    string fin = s + inf + t;
    build(fin);
    int n = s.size(), m = t.size();
    int ans = 0;
    for(int i = n + 1; i < n + m + 1; i++)
      if(z[i] == n)
        ans++;
    return ans;
  }
  void build(string &s){
    int n = s.size();
    z = vector<int>(n, 0);
    int l = 0, r = 0;
    for(int i = 0; i < n; i++){
      z[i] = max(min(z[i - l], r - i), 0LL);
      while(i + z[i] < n && s[z[i]] == s[i + z[i]])
        l = i, r = i + z[i], z[i]++;
    }
  }
};
```

7 Flow

7.1 Dinic

```
/**
* After computing flow, edges {u,v} s.t
* lev[u] ≠ −1, lev[v] = −1 are part of min cut.
* Use \texttt{reset} and \texttt{rcap} for Gomory-Hu.
* Time: O(N^2M) flow
* O(M√N) bipartite matching
* O(NM√N)orO(NM\sqrtM) on unit graph.
 */
struct Dinic {
    using F = long long; // flow type
    struct Edge { int to; F flo, cap; };
    int N;
  vector<Edge> eds;
  vector<vector<int>> adj;
    void init(int _N) {
        N = _N; adj.resize(N), cur.resize(N);
    }
    void reset() {
        for (auto &e: eds) e.flo = 0;
    }
    void ae(int u, int v, F cap, F rcap = 0) {
        assert(min(cap,rcap) >= 0);
        adj[u].pb((int)eds.size());
    eds.pb({v, 0, cap});
        adj[v].pb((int)eds.size());
    eds.pb({u, 0, rcap});
    }
    vector<int>lev;
  vector<vector<int>::iterator> cur;
    // level = shortest distance from source
    bool bfs(int s, int t) {
        lev = vector<int>(N,-1);
        for(int i = 0; i < N; i++) cur[i] = begin(adj[i]);
        queue<int> q({s}); lev[s] = 0;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto &e: adj[u]) {
                const Edge& E = eds[e];
                int v = E.to;
                if (lev[v] < 0 && E.flo < E.cap)
                    q.push(v), lev[v] = lev[u]+1;
            }
        }
        return lev[t] >= 0;
    }
    F dfs(int v, int t, F flo) {
        if (v == t) return flo;
        for (; cur[v] != end(adj[v]); cur[v]++) {
            Edge& E = eds[*cur[v]];
            if (lev[E.to]!=lev[v]+1||E.flo==E.cap) continue;
```

```
            F df = dfs(E.to,t,min(flo,E.cap-E.flo));
            if (df) {
                E.flo += df;
                eds[*cur[v]^1].flo -= df;
                return df;
            } // saturated >=1 one edge
        }
        return 0;
    }
    F maxFlow(int s, int t) {
        F tot = 0;
        while (bfs(s,t)) while (F df =
      dfs(s,t,numeric_limits<F>::max()))
        tot += df;
        return tot;
    }
    int fp(int u, int t,F f, vector<int> &path, vector<F> &flo, vector<int> &vis) {
        vis[u] = 1;
        if (u == t) {
            path.pb(u);
            return f;
        }
        for (auto eid: adj[u]) {
            auto &e = eds[eid];
            F w = e.flo - flo[eid];
            if (w <= 0 || vis[e.to]) continue;
            w = fp(e.to, t,
        min(w, f), path, flo, vis);
            if (w) {
                flo[eid] += w, path.pb(u);
                return w;
            }
        }
        return 0;
    }
  // return collection of {bottleneck, path[]}
    vector<pair<F, vector<int>>> allPath(int s, int t) {
        vector<pair<F, vector<int>>> res; vector<F> flo((int)eds.size());
    vector<int> vis;
        do res.pb(mp(0, vector<int>()));
        while (res.back().first =
      fp(s, t, numeric_limits<F>::max(),
      res.back().second, flo, vis=vector<int>(N))
    );
        for (auto &p: res) reverse(all(p.second));
        return res.pop_back(), res;
    }
};
```

7.2 MCMF

```
struct MCMF{
  struct Edge{
    int from, to;
    int cap, cost;
    Edge(int f, int t, int ca, int co): from(f), to(t), cap(ca), cost(co) {}
  };
  int n, s, t;
  vector<Edge>edges;
  vector<vector<int>>G;
  vector<int>d;
  vector<int>in_queue, prev_edge;
  MCMF(){}
  MCMF(int _n, int _s, int _t): n(_n), G(_n + 1), d(_n + 1), in_queue(_n + 1), prev_edge(_n + 1), s(_s), t(_t) {}
  void addEdge(int u, int v, int cap, int cost){
    G[u].push_back(edges.size());
    edges.push_back(Edge(u, v, cap, cost));
    G[v].push_back(edges.size());
    edges.push_back(Edge(v, u, 0, -cost));
  }
  bool bfs(){
    bool found = false;
    fill(d.begin(), d.end(), (int)1e18+10);
    fill(in_queue.begin(), in_queue.end(), false);
    d[s] = 0;
    in_queue[s] = true;
    queue<int>q;
    q.push(s);
    while(!q.empty()){
      int u = q.front();
      q.pop();
      if(u == t)
        found = true;
      in_queue[u] = false;
      for(auto &id : G[u]){
        Edge e = edges[id];
        if(e.cap > 0 && d[u] + e.cost < d[e.to]){
          d[e.to] = d[u] + e.cost;
          prev_edge[e.to] = id;
          if(!in_queue[e.to]){
            in_queue[e.to] = true;
            q.push(e.to);
          }
        }
      }
    }
    return found;
  }
  pair<int, int>flow(){
    // return (cap, cost)
    int cap = 0, cost = 0;
    while(bfs()){
      int send = (int)1e18 + 10;
      int u = t;
      while(u != s){
        Edge e = edges[prev_edge[u]];
        send = min(send, e.cap);
        u = e.from;
      }
      u = t;
      while(u != s){
        Edge &e = edges[prev_edge[u]];
        e.cap -= send;
        Edge &e2 = edges[prev_edge[u] ^ 1];
        e2.cap += send;
        u = e.from;
      }
      cap += send;
      cost += send * d[t];
    }
    return make_pair(cap, cost);
```

```
  }
};
```

8 Math

8.1 FastPow

```
long long qpow(long long x, long long powcnt, long long tomod){
  long long res = 1;
  for(; powcnt ; powcnt >>= 1 , x = (x * x) % tomod)
    if(1 & powcnt)
      res = (res * x) % tomod;
  return (res % tomod);
```

8.2 EXGCD

```
// ax + by = c
// return (gcd(a, b), x, y)
tuple<long long, long long, long long>exgcd(long long a, long long b){
  if(b == 0)
    return make_tuple(a, 1, 0);
  auto[g, x, y] = exgcd(b, a % b);
  return make_tuple(g, y, x - (a / b) * y);
```

8.3 EXCRT

```
long long inv(long long x){ return qpow(x, mod - 2, mod); }
long long mul(long long x, long long y, long long m){
   x = ((x % m) + m) % m, y = ((y % m) + m) % m;
  long long ans = 0;
  while(y){
    if(y & 1)
      ans = (ans + x) % m;
    x = x * 2 % m;
    y >>= 1;
  }
  return ans;
}
pii ExCRT(long long r1, long long m1, long long r2, long long m2){
  long long g, x, y;
  tie(g, x, y) = exgcd(m1, m2);
  if((r1 - r2) % g)
    return {-1, -1};
  long long lcm = (m1 / g) * m2;
  long long res = (mul(mul(m1, x, lcm), ((r2 - r1) / g), lcm) + r1) % lcm;
  res = (res + lcm) % lcm;
  return {res, lcm};
}
void solve(){
  long long n, r, m;
  cin >> n;
  cin >> m >> r; // x == r (mod m)
  for(long long i = 1 ; i < n ; i++){
    long long r1, m1;
    cin >> m1 >> r1;
    if(r != -1 && m != -1)
      tie(r, m) = ExCRT(r m, r1, m1);
  }
  if(r == -1 && m == -1)
    cout << "no solution\n";
  else
    cout << r << '\n';
}
```

8.4 FFT

```
struct Polynomial{
  int deg;
  vector<int>x;
  void FFT(vector<complex<double>>&a, bool invert){
    int a_sz = a.size();
    for(int len = 1; len < a_sz; len <<= 1){
      for(int st = 0; st < a_sz; st += 2 * len){
        double angle = PI / len * (invert ? -1 : 1);
        complex<double>wnow(1), w(cos(angle), sin(angle));
        for(int i = 0; i < len; i++){
          auto a0 = a[st + i], a1 = a[st + len + i];
          a[st + i] = a0 + wnow * a1;
          a[st + i + len] = a0 - wnow * a1;
          wnow *= w;
        }
      }
    }
    if(invert)
      for(auto &i : a)
        i /= a_sz;
  }
  void change(vector<complex<double>>&a){
    int a_sz = a.size();
    vector<int>rev(a_sz);
    for(int i = 1; i < a_sz; i++){
      rev[i] = rev[i / 2] / 2;
      if(i & 1)
        rev[i] += a_sz / 2;
    }
    for(int i = 0; i < a_sz; i++)
      if(i < rev[i])
        swap(a[i], a[rev[i]]);
  }
  Polynomial multiply(Polynomial const&b){
    vector<complex<double>>A(x.begin(), x.end()), B(b.x.begin(), b.x.end());
    int mx_sz = 1;
    while(mx_sz < A.size() + B.size())
      mx_sz <<= 1;
    A.resize(mx_sz);
    B.resize(mx_sz);
    change(A);
    change(B);
    FFT(A, 0);
    FFT(B, 0);
```

```
    for(int i = 0; i < mx_sz; i++)
      A[i] *= B[i];
    change(A);
    FFT(A, 1);
    Polynomial res(mx_sz);
    for(int i = 0; i < mx_sz; i++)
      res.x[i] = round(A[i].real());
    while(!res.x.empty() && res.x.back() == 0)
      res.x.pop_back();
    res.deg = res.x.size();
    return res;
  }
  Polynomial(): Polynomial(0) {}
  Polynomial(int Size): x(Size), deg(Size) {}
};
```

8.5 NTT

```
/*
p = r * 2^k + 1
p             r   k   root
998244353       119   23   3
2013265921       15   27   31
2061584302081     15   37   7
*/
template<int MOD, int RT>
struct NTT {
    #define OP(op) static int op(int x, int y)
    OP(add) { return (x += y) >= MOD ? x - MOD : x; }
    OP(sub) { return (x -= y) < 0 ? x + MOD : x; }
    OP(mul) { return ll(x) * y % MOD; } // multiply by bit if p * p > 9e18
    static int mpow(int a, int n) {
        int r = 1;
        while (n) {
            if (n % 2) r = mul(r, a);
            n /= 2, a = mul(a, a);
        }
        return r;
    }
  static const int MAXN = 1 << 21;
    static int minv(int a) { return mpow(a, MOD - 2); }
    int w[MAXN];
    NTT() {
        int s = MAXN / 2, dw = mpow(RT, (MOD - 1) / MAXN);
        for (; s; s >>= 1, dw = mul(dw, dw)) {
            w[s] = 1;
            for (int j = 1; j < s; ++j)
                w[s + j] = mul(w[s + j - 1], dw);
        }
    }
    void apply(vector<int>&a, int n, bool inv = 0) {
        for (int i = 0, j = 1; j < n - 1; ++j) {
            for (int k = n >> 1; (i ^= k) < k; k >>= 1);
            if (j < i) swap(a[i], a[j]);
        }
        for (int s = 1; s < n; s <<= 1) {
            for (int i = 0; i < n; i += s * 2) {
                for (int j = 0; j < s; ++j) {
                    int tmp = mul(a[i + s + j], w[s + j]);
                    a[i + s + j] = sub(a[i + j], tmp);
                    a[i + j] = add(a[i + j], tmp);
                }
            }
        }
        if(!inv)
      return;
        int iv = minv(n);
    if(n > 1)
      reverse(next(a.begin()), a.end());
        for (int i = 0; i < n; ++i)
      a[i] = mul(a[i], iv);
    }
  vector<int>convolution(vector<int>&a, vector<int>&b){
    int sz = a.size() + b.size() - 1, n = 1;
    while(n <= sz)
      n <<= 1; // check n <= MAXN
    vector<int>res(n);
    a.resize(n), b.resize(n);
    apply(a, n);
    apply(b, n);
    for(int i = 0; i < n; i++)
      res[i] = mul(a[i], b[i]);
    apply(res, n, 1);
    return res;
  }
};
```

8.6 MillerRain

```
bool is_prime(long long n, vector<long long> x) {
  long long d = n - 1;
  d >>= __builtin_ctzll(d);
  for(auto a : x) {
    if(n <= a) break;
    long long t = d, y = 1, b = t;
    while(b) {
      if(b & 1) y = __int128(y) * a % n;
      a = __int128(a) * a % n;
      b >>= 1;
    }
    while(t != n - 1 && y != 1 && y != n - 1) {
      y = __int128(y) * y % n;
      t <<= 1;
    }
    if(y != n - 1 && t % 2 == 0) return 0;
  }
  return 1;
}
bool is_prime(long long n) {
  if(n <= 1) return 0;
  if(n % 2 == 0) return n == 2;
  if(n < (1LL << 30)) return is_prime(n, {2, 7, 61});
  return is_prime(n, {2, 325, 9375, 28178, 450775, 9780504, 1795265022});
}
```

8.7 PollardRho

```cpp
void PollardRho(map<long long, int>& mp, long long n) {
  if(n == 1) return;
  if(is_prime(n)) return mp[n]++, void();
  if(n % 2 == 0) {
    mp[2] += 1;
    PollardRho(mp, n / 2);
    return;
  }
  ll x = 2, y = 2, d = 1, p = 1;
  #define f(x, n, p) ((__int128(x) * x % n + p) % n)
  while(1) {
    if(d != 1 && d != n) {
      PollardRho(mp, d);
      PollardRho(mp, n / d);
      return;
    }
    p += (d == n);
    x = f(x, n, p), y = f(f(y, n, p), n, p);
    d = __gcd(abs(x - y), n);
  }
  #undef f
}
vector<long long> get_divisors(long long n) {
  if(n == 0) return {};
  map<long long, int> mp;
  PollardRho(mp, n);
  vector<pair<long long, int>> v(mp.begin(), mp.end());
  vector<long long> res;
  auto f = [&](auto f, int i, long long x) -> void {
    if(i == (int)v.size()) {
      res.pb(x);
      return;
    }
    for(int j = v[i].second; ; j--) {
      f(f, i + 1, x);
      if(j == 0) break;
      x *= v[i].first;
    }
  };
  f(f, 0, 1);
  sort(res.begin(), res.end());
  return res;
}
```

8.8 XorBasis

```cpp
template<int LOG> struct XorBasis {
  bool zero = false;
  int cnt = 0;
  ll p[LOG] = {};
  vector<ll> d;
  void insert(ll x) {
    for(int i = LOG - 1; i >= 0; --i) {
      if(x >> i & 1) {
        if(!p[i]) {
          p[i] = x;
          cnt += 1;
          return;
        } else x ^= p[i];
      }
    }
    zero = true;
  }
  ll get_max() {
    ll ans = 0;
    for(int i = LOG - 1; i >= 0; --i) {
      if((ans ^ p[i]) > ans) ans ^= p[i];
    }
    return ans;
  }
  ll get_min() {
    if(zero) return 0;
    for(int i = 0; i < LOG; ++i) {
      if(p[i]) return p[i];
    }
  }
  bool include(ll x) {
    for(int i = LOG - 1; i >= 0; --i) {
      if(x >> i & 1) x ^= p[i];
    }
    return x == 0;
  }
  void update() {
    d.clear();
    for(int j = 0; j < LOG; ++j) {
      for(int i = j - 1; i >= 0; --i) {
        if(p[j] >> i & 1) p[j] ^= p[i];
      }
    }
    for(int i = 0; i < LOG; ++i) {
      if(p[i]) d.PB(p[i]);
    }
  }
  ll get_kth(ll k) {
    if(k == 1 && zero) return 0;
    if(zero) k -= 1;
    if(k >= (1LL << cnt)) return -1;
    update();
    ll ans = 0;
    for(int i = 0; i < SZ(d); ++i) {
      if(k >> i & 1) ans ^= d[i];
    }
    return ans;
  }
};
```

8.9 GeneratingFunctions

- Ordinary Generating Function $A(x) = \sum_{i \ge 0} a_i x^i$
  - $A(rx) \Rightarrow r^n a_n$
  - $A(x) + B(x) \Rightarrow a_n + b_n$
  - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} a_i b_{n-i}$
  - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} a_{i_1} a_{i_2} \ldots a_{i_k}$
  - $xA(x)' \Rightarrow n a_n$
  - $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^{n} a_i$
- Exponential Generating Function $A(x) = \sum_{i \ge 0} \frac{a_i}{i!} x_i$
  - $A(x) + B(x) \Rightarrow a_n + b_n$
  - $A^{(k)}(x) \Rightarrow a_{n+k_n}$
  - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} n i a_i b_{n-i}$
  - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} n i_1, i_2, \ldots, i_k a_{i_1} a_{i_2} \ldots a_{i_k}$

- $xA(x) \Rightarrow na_n$

- Special Generating Function
  - $(1+x)^n = \sum_{i\geq 0} nix^i$
  - $\frac{1}{(1-x)^n} = \sum_{i\geq 0} in - 1x^i$

8.10 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 $C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$ $,\forall n \geq 0$ $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} = \frac{2(2n+1)}{n+2}C_n,\ C_0 = 1$

- Hockey-stick identity $\sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$

8.11 Theorem

- Cayley's Formula
  - Given a degree sequence $d_1, d_2, \ldots, d_n$ for each *labeled* vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees.
  - Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1, 2, \ldots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

- Erdős–Gallai theorem A sequence of nonnegative integers $d_1 \geq \cdots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1+\cdots+d_n$ is even and $\displaystyle\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$ holds for every $1 \leq k \leq n$.

- Gale–Ryser theorem A pair of sequences of nonnegative integers $a_1 \geq \cdots \geq a_n$ and $b_1, \ldots, b_n$ is bigraphic if and only if $\displaystyle\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\displaystyle\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{n} \min(b_i, k)$ 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
  - $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$
  - $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$
  - $\sum_{d|n}^{n=1} \mu(d) = 1$
  - $\sum_{d|n}^{n\neq 1} \mu(d) = 0$

- 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 $= 2\pi rh = \pi(a^2+h^2) = 2\pi r^2(1-\cos\theta)$.