JAW Codebook

Contents

1 Basic

1.1 vimrc

```
sy on
se sw=4 ts=4 sts=4 et nu sc hls
filet plugin indent on
no <F4> :vs input.txt<CR>
no <F5> :!./a.out<CR>
no <F6> :!./a.out < input.txt<CR>
no <F9> :!g++ -static -O2 -std=gnu++14 -lm % -g
    -fsanitize=undefined -Wall -Wextra -Wshadow
    -Wno-unused-result<CR>
no <expr> <silent> <Home> col('.') ==
    match(getline('.'),'\S') + 1 ? '0' : '^'
im <silent> <Home> <C-O><Home>
```

2 Data Structure

2.1 Undo Disjoint Set

```
int par[maxn], sz[maxn];
vector<pair<int*, int>> h;
vector<int> sp;
void init(int n) {
    iota(par, par + n, 0);
```

```
    fill_n(sz, n, 1);
    sp.clear();
    h.clear();
}
void assign(int *k, int v) {
    h.emplace_back(k, *k);
    *k = v;
}
void save() {
    sp.push_back(h.size());
}
void undo() {
    while (h.size() != sp.back()) {
        auto x = h.back();
        h.pop_back();
        *x.first = x.second;
    }
    sp.pop_back();
}
int find(int x) {
    while (x != par[x])
        x = par[x];
    return x;
}
void merge(int x, int y) {
    x = find(x), y = find(y);
    if (x != y) {
        if (sz[x] < sz[y])
            swap(x, y);
        assign(sz + x, sz[x] + sz[y]);
        assign(par + y, x);
    }
}
```

2.2 Range Disjoint Set

```
int par[maxn][20];
void init(int n) {
    for (int i = 0; i < n; i++)
        fill_n(par[i], 20, i);
}
int find(int x, int y = 0) {
    return (par[x][y] == x) ? x : (par[x][y] =
        find(par[x][y], y));
}
bool merge(int x, int y, int k) {
    x = find(x, k);
    y = find(y, k);
    if (x == y)
        return false;
    par[y][k] = x;
    if (k--) {
        merge(x, y, k);
        merge(x + (1 << k), y + (1 << k), k);
    }
    return true;
}
```

2.3 Treap

```
const int MEM = 16000004;
struct Treap {
  static Treap nil, mem[MEM], *pmem;
  Treap *l, *r;
  char val;
  int size;
  Treap () : l(&nil), r(&nil), size(0) {}
  Treap (char _val) :
    l(&nil), r(&nil), val(_val), size(1) {}
} Treap::nil, Treap::mem[MEM], *Treap::pmem =
    Treap::mem;

int size(const Treap *t) { return t->size; }
void pull(Treap *t) {
  if (!size(t)) return;
  t->size = size(t->l) + size(t->r) + 1;
}
Treap* merge(Treap *a, Treap *b) {
  if (!size(a)) return b;
  if (!size(b)) return a;
  Treap *t;
  if (rand() % (size(a) + size(b)) < size(a)) {
    t = new (Treap::pmem++) Treap(*a);
    t->r = merge(a->r, b);
  } else {
    t = new (Treap::pmem++) Treap(*b);
    t->l = merge(a, b->l);
  }
  pull(t);
  return t;
}
void split(Treap *t, int k, Treap *&a, Treap *&b) {
  if (!size(t)) a = b = &Treap::nil;
  else if (size(t->l) + 1 <= k) {
    a = new (Treap::pmem++) Treap(*t);
    split(t->r, k - size(t->l) - 1, a->r, b);
    pull(a);
  } else {
    b = new (Treap::pmem++) Treap(*t);
    split(t->l, k, a, b->l);
    pull(b);
  }
}

int nv;
Treap *rt[50005];

void print(const Treap *t) {
  if (!size(t)) return;
  print(t->l);
  cout << t->val;
  print(t->r);
}

int main(int argc, char** argv) {
  IOS;
  rt[nv=0] = &Treap::nil;
  Treap::pmem = Treap::mem;
  int Q, cmd, p, c, v;
  string s;
  cin >> Q;
  while (Q--) {
    cin >> cmd;
    if (cmd == 1) {
      // insert string s after position p
      cin >> p >> s;
      Treap *tl, *tr;
      split(rt[nv], p, tl, tr);
      for (int i=0; i<SZ(s); i++)
        tl = merge(tl, new (Treap::pmem++)
            Treap(s[i]));
      rt[++nv] = merge(tl, tr);
    } else if (cmd == 2) {
      // remove c characters starting at position
      Treap *tl, *tm, *tr;
      cin >> p >> c;
      split(rt[nv], p-1, tl, tm);
      split(tm, c, tm, tr);
      rt[++nv] = merge(tl, tr);
    } else if (cmd == 3) {
      // print c characters starting at position p,
          in version v
      Treap *tl, *tm, *tr;
      cin >> v >> p >> c;
      split(rt[v], p-1, tl, tm);
      split(tm, c, tm, tr);
      print(tm);
      cout << "\n";
    }
  }
  return 0;
}
```

2.4 Heavy Light Decomposition

```
// only one segment tree / 0-base
// should call init after input N
// getPathSeg return the segment in order u->v
// fa[root] = root

typedef pair<int,int> pii;

int N,fa[MXN],belong[MXN],dep[MXN],sz[MXN],que[MXN];
int step,line[MXN],stPt[MXN],edPt[MXN];
vector<int> E[MXN], chain[MXN];

void init() {
  REP(i,N) {
    E[i].clear();
    chain[i].clear();
  }
}
void DFS(int u){
  vector<int> &c = chain[belong[u]];
  for (int i=c.size()-1; i>=0; i--){
    int v = c[i];
    stPt[v] = step;
    line[step++] = v;
  }
  for (int i=0; i<(int)c.size(); i++){
    u = c[i];
    for (auto v : E[u]){
      if (fa[u] == v || (i && v == c[i-1])) continue;
      DFS(v);
    }
    edPt[u] = step-1;
  }
}
void build_chain(int st){
  int fr,bk;
  fr=bk=0; que[bk++]=st; fa[st]=st; dep[st]=0;
  while (fr < bk){
    int u=que[fr++];
    for (auto v : E[u]){
      if (v == fa[u]) continue;
      que[bk++] = v;
      dep[v] = dep[u]+1;
      fa[v] = u;
    }
  }
  for (int i=bk-1,u,pos; i>=0; i--){
    u = que[i]; sz[u] = 1; pos = -1;
    for (auto v : E[u]){
      if (v == fa[u]) continue;
      sz[u] += sz[v];
      if (pos==-1 || sz[v]>sz[pos]) pos=v;
    }
    if (pos == -1) belong[u] = u;
    else belong[u] = belong[pos];
    chain[belong[u]].PB(u);
  }
  step = 0;
  DFS(st);
}
int getLCA(int u, int v){
  while (belong[u] != belong[v]){
    int a = chain[belong[u]].back();
    int b = chain[belong[v]].back();
    if (dep[a] > dep[b]) u = fa[a];
    else v = fa[b];
  }
  return sz[u] >= sz[v] ? u : v;
}
vector<pii> getPathSeg(int u, int v){
  vector<pii> ret1,ret2;
  while (belong[u] != belong[v]){
    int a = chain[belong[u]].back();
    int b = chain[belong[v]].back();
    if (dep[a] > dep[b]){
      ret1.PB({stPt[a],stPt[u]});
      u = fa[a];
    } else {
      ret2.PB({stPt[b],stPt[v]});
      v = fa[b];
    }
  }
  if (dep[u] > dep[v]) swap(u,v);
  ret1.PB({stPt[u],stPt[v]});
  reverse(ret2.begin(), ret2.end());
  ret1.insert(ret1.end(),ret2.begin(),ret2.end());
  return ret1;
}
// Usage
void build(){
  build_chain(0); //change root
  init(0,step,0); //init segment tree
}
int get_answer(int u, int v){
  int ret = -2147483647;
  vector<pii> vec = getPathSeg(u,v);
  for (auto it : vec)
    ; // check answer with segment [it.F, it.S]
  return ret;
}
```

2.5 Link Cut Tree

```
struct Node {
    Node *par, *ch[2], *mx;
    int id, sz, rev_tag, val;
    Node(int _id = 0, int _val = 0): par(), ch(),
        mx(this), id(_id), sz(1), rev_tag(),
        val(_val) {}
};
struct Edge {
    int x, y, a, b;
    Edge() {}
    Edge(int _x, int _y, int _a, int _b): x(_x),
        y(_y), a(_a), b(_b) {}
    bool operator < (const Edge &rhs) const {
        return a < rhs.a;
    }
};

Node *tr[maxn];
vector<Edge> edges;
void rev(Node *o) {
    swap(o->ch[0], o->ch[1]);
    o->rev_tag ^= 1;
}
int sz(Node *o) {
    return o ? o->sz : 0;
}
void push(Node *o) {
    if (o->rev_tag) {
        for (auto ch : o->ch)
            if (ch)
                rev(ch);
        o->rev_tag ^= 1;
    }
}
void pull(Node *o) {
    o->sz = sz(o->ch[0]) + 1 + sz(o->ch[1]);
    o->mx = o;
    for (auto ch : o->ch)
        if (ch && ch->mx->val > o->mx->val)
            o->mx = ch->mx;
}
int get_ch_id(Node *p, Node *o) {
    for (int i = 0; i < 2; i++)
        if (p->ch[i] == o)
            return i;
    return -1;
}
void rotate(Node *o, int d) {
    push(o);
    push(o->ch[d]);
```

```
    Node *u = o;
    o = o->ch[d];
    Node *p = u->par;
    int t;
    if (p && (t = get_ch_id(p, u)) != -1)
        p->ch[t] = o;
    o->par = p;
    u->par = o;
    if (o->ch[d^1])
        o->ch[d^1]->par = u;
    u->ch[d] = o->ch[d^1];
    o->ch[d^1] = u;
    pull(u);
    pull(o);
}
void rotate(Node *o) {
    if (sz(o->ch[0]) > sz(o->ch[1]))
        rotate(o, 0);
    else if (sz(o->ch[0]) < sz(o->ch[1]))
        rotate(o, 1);
}
void all_push(Node *o) {
    if (o->par && get_ch_id(o->par, o) != -1)
        all_push(o->par);
    push(o);
}
void splay(Node *o) {
    all_push(o);
    Node *p;
    for (int d; (p = o->par) && (d = get_ch_id(p,
        o)) != -1; ) {
        rotate(p, d);
        rotate(p);
    }
}
Node* access(Node *o) {
    Node *last = 0;
    while (o) {
        splay(o);
        o->ch[1] = last;
        pull(o);
        last = o;
        o = o->par;
    }
    return last;
}
void make_root(Node *o) {
    rev(access(o));
    splay(o);
}
void link(Node *a, Node *b) {
    make_root(b);
    b->par = a;
}
void cut(Node *a, Node *b) {
    make_root(a);
    access(b);
    splay(b);
    b->ch[0] = 0;
    a->par = 0;
    pull(b);
}
Node* find_root(Node *o) {
    o = access(o);
    while (o->ch[0])
        o = o->ch[0];
    splay(o);
    return o;
}
void add_edge(int n, int i, int x, int y, int v) {
    tr[n + i] = new Node(n + i, v);
    if (find_root(tr[x]) == find_root(tr[y])) {
        make_root(tr[x]);
        access(tr[y]);
        splay(tr[x]);
        int id = tr[x]->mx->id - n;
        if (edges[id].b > v) {
            cut(tr[edges[id].x], tr[n + id]);
            cut(tr[edges[id].y], tr[n + id]);
            link(tr[x], tr[n + i]);
            link(tr[y], tr[n + i]);
        }
    } else {
        link(tr[x], tr[n + i]);
        link(tr[y], tr[n + i]);
    }
}
```

3 Graph

3.1 BCC Edge

```
struct BccEdge {
  static const int MXN = 100005;
  struct Edge { int v,eid; };
  int n,m,step,par[MXN],dfn[MXN],low[MXN];
  vector<Edge> E[MXN];
  DisjointSet djs;
  void init(int _n) {
    n = _n; m = 0;
    for (int i=0; i<n; i++) E[i].clear();
    djs.init(n);
  }
  void add_edge(int u, int v) {
    E[u].PB({v, m});
    E[v].PB({u, m});
    m++;
  }
  void DFS(int u, int f, int f_eid) {
    par[u] = f;
    dfn[u] = low[u] = step++;
    for (auto it:E[u]) {
      if (it.eid == f_eid) continue;
      int v = it.v;
      if (dfn[v] == -1) {
        DFS(v, u, it.eid);
        low[u] = min(low[u], low[v]);
      } else {
        low[u] = min(low[u], dfn[v]);
      }
    }
  }
  void solve() {
    step = 0;
    memset(dfn, -1, sizeof(int)*n);
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) DFS(i, i, -1);
    }
    djs.init(n);
    for (int i=0; i<n; i++) {
      if (low[i] < dfn[i]) djs.uni(i, par[i]);
    }
  }
}graph;
```

3.2 BCC Vertex

```
struct BccVertex {
  int n,nBcc,step,root,dfn[MXN],low[MXN];
  vector<int> E[MXN], ap;
  vector<pii> bcc[MXN];
  int top;
  pii stk[MXN];
  void init(int _n) {
    n = _n;
    nBcc = step = 0;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v) {
    E[u].PB(v);
    E[v].PB(u);
  }
```

```
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    int son = 0;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        son++;
        stk[top++] = {u,v};
        DFS(v,u);
        if (low[v] >= dfn[u]) {
          if(v != root) ap.PB(v);
          do {
            assert(top > 0);
            bcc[nBcc].PB(stk[--top]);
          } while (stk[top] != pii(u,v));
          nBcc++;
        }
        low[u] = min(low[u], low[v]);
      } else {
        if (dfn[v] < dfn[u]) stk[top++] = pii(u,v);
        low[u] = min(low[u],dfn[v]);
      }
    }
    if (u == root && son > 1) ap.PB(u);
  }
  // return the edges of each bcc;
  vector<vector<pii>> solve() {
    vector<vector<pii>> res;
    for (int i=0; i<n; i++) {
      dfn[i] = low[i] = -1;
    }
    ap.clear();
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) {
        top = 0;
        root = i;
        DFS(i,i);
      }
    }
    REP(i,nBcc) res.PB(bcc[i]);
    return res;
  }
}graph;
```

3.3 Strongly Connected Components

```
struct Scc{
  int n, nScc, vst[MXN], bln[MXN];
  vector<int> E[MXN], rE[MXN], vec;
  void init(int _n){
    n = _n;
    for (int i=0; i<n; i++){
      E[i].clear();
      rE[i].clear();
    }
  }
  void add_edge(int u, int v){
    E[u].PB(v);
    rE[v].PB(u);
  }
  void DFS(int u){
    vst[u]=1;
    for (auto v : E[u])
      if (!vst[v]) DFS(v);
    vec.PB(u);
  }
  void rDFS(int u){
    vst[u] = 1;
    bln[u] = nScc;
    for (auto v : rE[u])
      if (!vst[v]) rDFS(v);
  }
  void solve(){
    nScc = 0;
    vec.clear();
    for (int i=0; i<n; i++) vst[i] = 0;
    for (int i=0; i<n; i++)
      if (!vst[i]) DFS(i);
    reverse(vec.begin(),vec.end());
    for (int i=0; i<n; i++) vst[i] = 0;
    for (auto v : vec){
      if (!vst[v]){
        rDFS(v);
        nScc++;
      }
    }
  }
};
```

3.4 DMST_with_sol

```
const int INF = 1029384756;

struct edge_t{
  int u,v,w;
  set< pair<int,int> > add, sub;
  edge_t() : u(-1), v(-1), w(0) {}
  edge_t(int _u, int _v, int _w) {
    u = _u; v = _v; w = _w;
    add.insert({u, v});
  }
  edge_t& operator += (const edge_t& obj) {
    w += obj.w;
    FOR (it, obj.add) {
      if (!sub.count(*it)) add.insert(*it);
      else sub.erase(*it);
    }
    FOR (it, obj.sub) {
      if (!add.count(*it)) sub.insert(*it);
      else add.erase(*it);
    }
    return *this;
  }
  edge_t& operator -= (const edge_t& obj) {
    w -= obj.w;
    FOR (it, obj.sub) {
      if (!sub.count(*it)) add.insert(*it);
      else sub.erase(*it);
    }
    for (auto it : obj.add) {
      if (!add.count(it)) sub.insert(it);
      else add.erase(it);
    }
    return *this;
  }
}eg[MXN*MXN],prv[MXN],EDGE_INF(-1,-1,INF);
int N,M;
int cid,incyc[MXN],contracted[MXN];
vector<int> E[MXN];

edge_t dmst(int rt){
  edge_t cost;
  for (int i=0; i<N; i++){
    contracted[i] = incyc[i] = 0;
    prv[i] = EDGE_INF;
  }
  cid = 0;
  int u,v;
  while (true){
    for (v=0; v<N; v++){
      if (v != rt && !contracted[v] && prv[v].w ==
          INF) break;
    }
    if (v >= N) break; // end
    for (int i=0; i<M; i++){
      if (eg[i].v == v && eg[i].w < prv[v].w)
        prv[v] = eg[i];
    }
    if (prv[v].w == INF) // not connected
      return EDGE_INF;
    cost += prv[v];
    for (u=prv[v].u; u!=v && u!=-1; u=prv[u].u);
    if (u == -1) continue;
    incyc[v] = ++cid;
```

```
    for (u=prv[v].u; u!=v; u=prv[u].u){
      contracted[u] = 1;
      incyc[u] = cid;
    }
    for (int i=0; i<M; i++){
      if (incyc[eg[i].u] != cid && incyc[eg[i].v] ==
          cid){
        eg[i] -= prv[eg[i].v];
      }
    }
    for (int i=0; i<M; i++){
      if (incyc[eg[i].u] == cid) eg[i].u = v;
      if (incyc[eg[i].v] == cid) eg[i].v = v;
      if (eg[i].u == eg[i].v) eg[i--] = eg[--M];
    }
    for (int i=0; i<N; i++){
      if (contracted[i]) continue;
      if (prv[i].u>=0 && incyc[prv[i].u] == cid)
        prv[i].u = v;
    }
    prv[v] = EDGE_INF;
  }
  return cost;
}

void solve(){
  edge_t cost = dmst(0);
  for (auto it : cost.add){ // find a solution
    E[it.F].PB(it.S);
    prv[it.S] = edge_t(it.F,it.S,0);
  }
}
```

3.5 Dominator Tree

```
// idom[n] is the unique node that strictly
    dominates n but does
// not strictly dominate any other node that
    strictly dominates n.
// idom[n] = 0 if n is entry or the entry cannot
    reach n.
struct DominatorTree{
  static const int MAXN = 200010;
  int n,s;
  vector<int> g[MAXN],pred[MAXN];
  vector<int> cov[MAXN];
  int dfn[MAXN],nfd[MAXN],ts;
  int par[MAXN];
  int sdom[MAXN],idom[MAXN];
  int mom[MAXN],mn[MAXN];

  inline bool cmp(int u,int v) { return dfn[u] <
      dfn[v]; }

  int eval(int u) {
    if(mom[u] == u) return u;
    int res = eval(mom[u]);
    if(cmp(sdom[mn[mom[u]]],sdom[mn[u]]))
      mn[u] = mn[mom[u]];
    return mom[u] = res;
  }

  void init(int _n, int _s) {
    n = _n;
    s = _s;
    REP1(i,1,n) {
      g[i].clear();
      pred[i].clear();
      idom[i] = 0;
    }
  }
  void add_edge(int u, int v) {
    g[u].push_back(v);
    pred[v].push_back(u);
  }
  void DFS(int u) {
    ts++;
    dfn[u] = ts;
    nfd[ts] = u;
    for(int v:g[u]) if(dfn[v] == 0) {
      par[v] = u;
      DFS(v);
    }
  }
  void build() {
    ts = 0;
    REP1(i,1,n) {
      dfn[i] = nfd[i] = 0;
      cov[i].clear();
      mom[i] = mn[i] = sdom[i] = i;
    }
    DFS(s);
    for (int i=ts; i>=2; i--) {
      int u = nfd[i];
      if(u == 0) continue ;
      for(int v:pred[u]) if(dfn[v]) {
        eval(v);
        if(cmp(sdom[mn[v]],sdom[u])) sdom[u] =
            sdom[mn[v]];
      }
      cov[sdom[u]].push_back(u);
      mom[u] = par[u];
      for(int w:cov[par[u]]) {
        eval(w);
        if(cmp(sdom[mn[w]],par[u])) idom[w] = mn[w];
        else idom[w] = par[u];
      }
      cov[par[u]].clear();
    }
    REP1(i,2,ts) {
      int u = nfd[i];
      if(u == 0) continue ;
      if(idom[u] != sdom[u]) idom[u] = idom[idom[u]];
    }
  }
}dom;
```

3.6 Maximum Clique

```
int ans;
bitset<128> adj[128];
int ctz(bitset<128> x) {
  bitset<128> y = x & std::bitset<128>(~0ULL);
  if (y.any())
    return __builtin_ctzll(y.to_ullong());
  return __builtin_ctzll((x >> 64).to_ullong()) + 64;
}
void dfs(bitset<128> r, bitset<128> p, bitset<128>
    x) {
  ans = max(ans, (int) r.count());
  int t = r.count() + p.count();
  if (t <= ans || !p.any()) return;
  int pivot = ctz(p | x);
  bitset<128> tp = p & ~adj[pivot];
  while (tp.any() && t > ans) {
    int v = ctz(tp);
    auto bv = bitset<128>(1) << v;
    dfs(r | bv, p & adj[v], x & adj[v]);
    p ^= bv;
    x |= bv;
    t--;
    if (t <= ans)
      return;
    tp ^= bv;
  }
}
```

3.7 MinimumMeanCycle

```
/* minimum mean cycle */
const int MAXE = 1805;
const int MAXN = 35;
const double inf = 1029384756;
```

```
const double eps = 1e-6;
struct Edge {
  int v,u;
  double c;
};
int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN];
Edge e[MAXE];
vector<int> edgeID, cycle, rho;
double d[MAXN][MAXN];
inline void bellman_ford() {
  for(int i=0; i<n; i++) d[0][i]=0;
  for(int i=0; i<n; i++) {
    fill(d[i+1], d[i+1]+n, inf);
    for(int j=0; j<m; j++) {
      int v = e[j].v, u = e[j].u;
      if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
        d[i+1][u] = d[i][v]+e[j].c;
        prv[i+1][u] = v;
        prve[i+1][u] = j;
      }
    }
  }
}
double karp_mmc() {
  // returns inf if no cycle, mmc otherwise
  double mmc=inf;
  int st = -1;
  bellman_ford();
  for(int i=0; i<n; i++) {
    double avg=-inf;
    for(int k=0; k<n; k++) {
      if(d[n][i]<inf-eps)
          avg=max(avg,(d[n][i]-d[k][i])/(n-k));
      else avg=max(avg,inf);
    }
    if (avg < mmc) tie(mmc, st) = tie(avg, i);
  }
  for(int i=0; i<n; i++) vst[i] = 0;
  edgeID.clear(); cycle.clear(); rho.clear();
  for (int i=n; !vst[st]; st=prv[i--][st]) {
    vst[st]++;
    edgeID.PB(prve[i][st]);
    rho.PB(st);
  }
  while (vst[st] != 2) {
    int v = rho.back(); rho.pop_back();
    cycle.PB(v);
    vst[v]++;
  }
  reverse(ALL(edgeID));
  edgeID.resize(SZ(cycle));
  return mmc;
}
```

4 Flow

4.1 Push-relabel

```
#include <algorithm>
#include <list>
constexpr int maxn = 604;
int c[maxn][maxn], f[maxn][maxn], h[maxn], e[maxn],
    g[2 * maxn + 1];
int max_flow(int s, int t, int n) {
  for (int i = 0; i < n; i++)
    fill_n(f[i], n, 0);
  fill_n(h, n, 0);
  fill_n(e, n, 0);
  fill_n(g, 2 * n + 1, 0);

  for (int i = 0; i < n; i++) {
    f[s][i] = e[i] = c[s][i];
    f[i][s] = -c[s][i];
  }
  h[s] = n;
  e[s]++, e[t]++;
  g[0] = n - 1;
  g[n] = 1;

  list<int> fifo;
  for (int i = 0; i < n; i++)
    if (i != s && i != t && e[i])
      fifo.push_back(i);
  while (!fifo.empty()) {
    int u = fifo.front();
    fifo.pop_front();
    while (e[u]) {
      for (int v = 0; e[u] && v < n; v++) {
        if (h[u] == h[v] + 1 && f[u][v] < c[u][v]) {
          if (e[v] == 0)
            fifo.push_back(v);
          int x = min(e[u], c[u][v] - f[u][v]);
          e[u] -= x;
          e[v] += x;
          f[u][v] += x;
          f[v][u] -= x;
        }
      }
      if (e[u]) {
        if (--g[h[u]] == 0 && h[u] < n)
          for (int i = 0; i < n; i++)
            if (h[i] > h[u] && h[i] < n)
              h[i] = n + 1;
        h[u] = 2 * n;
        for (int v = 0; v < n; v++)
          if (f[u][v] < c[u][v])
            h[u] = min(h[u], h[v] + 1);
        g[h[u]]++;
      }
    }
  }
  return e[t] - 1;
}
```

4.2 Dinic

```
struct Dinic{
  static const int MXN = 10000;
  struct Edge{ int v,f,re; };
  int n,s,t,level[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t){
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u].PB({v,f,SZ(E[v])});
    E[v].PB({u,0,SZ(E[u])-1});
  }
  bool BFS(){
    for (int i=0; i<n; i++) level[i] = -1;
    queue<int> que;
    que.push(s);
    level[s] = 0;
    while (!que.empty()){
      int u = que.front(); que.pop();
      for (auto it : E[u]){
        if (it.f > 0 && level[it.v] == -1){
          level[it.v] = level[u]+1;
          que.push(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
      if (it.f > 0 && level[it.v] == level[u]+1){
        int tf = DFS(it.v, min(nf,it.f));
        res += tf; nf -= tf; it.f -= tf;
        E[it.v][it.re].f += tf;
```

```
      if (nf == 0) return res;
    }
  }
  if (!res) level[u] = -1;
  return res;
 }
 int flow(int res=0){
  while ( BFS() )
    res += DFS(s,2147483647);
  return res;
 }
}flow;
```

4.3 Cost Flow

```
typedef pair<long long, long long> pll;
struct CostFlow {
  static const int MXN = 205;
  static const long long INF = 102938475610293847LL;
  struct Edge {
    int v, r;
    long long f, c;
  };
  int n, s, t, prv[MXN], prvL[MXN], inq[MXN];
  long long dis[MXN], fl, cost;
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t) {
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
    fl = cost = 0;
  }
  void add_edge(int u, int v, long long f, long long
      c) {
    E[u].PB({v, SZ(E[v])  , f,  c});
    E[v].PB({u, SZ(E[u])-1, 0, -c});
  }
  pll flow() {
    while (true) {
      for (int i=0; i<n; i++) {
        dis[i] = INF;
        inq[i] = 0;
      }
      dis[s] = 0;
      queue<int> que;
      que.push(s);
      while (!que.empty()) {
        int u = que.front(); que.pop();
        inq[u] = 0;
        for (int i=0; i<SZ(E[u]); i++) {
          int v = E[u][i].v;
          long long w = E[u][i].c;
          if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
            prv[v] = u; prvL[v] = i;
            dis[v] = dis[u] + w;
            if (!inq[v]) {
              inq[v] = 1;
              que.push(v);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      long long tf = INF;
      for (int v=t, u, l; v!=s; v=u) {
        u=prv[v]; l=prvL[v];
        tf = min(tf, E[u][l].f);
      }
      for (int v=t, u, l; v!=s; v=u) {
        u=prv[v]; l=prvL[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
}flow;
```

4.4 Kuhn Munkres

```
int nl, nr, pre[maxn], mx[maxn], my[maxn];
ll slack[maxn], w[maxn][maxn], lx[maxn], ly[maxn];
bool vx[maxn], vy[maxn];
void augment(int u) {
  if (!u) return;
  augment(mx[pre[u]]);
  mx[pre[u]] = u;
  my[u] = pre[u];
}
void match(int x) {
  queue<int> que;
  que.push(x);
  while (true) {
    while (!que.empty()) {
      x = que.front();
      que.pop();
      vx[x] = 1;
      for (int y = 0; y < nr; y++) {
        if (vy[y]) continue;
        ll t = lx[x] + ly[y] - w[x][y];
        if (t > 0) {
          if (slack[y] >= t)
            slack[y] = t, pre[y] = x;
          continue;
        }
        pre[y] = x;
        if (!my[y]) {
          augment(y);
          return;
        }
        vy[y] = 1;
        que.push(my[y]);
      }
    }
    ll t = inf;
    for (int y = 0; y < nr; y++)
      if (!vy[y])
        t = min(t, slack[y]);
    for (int x = 0; x < nl; x++)
      if (vx[x])
        lx[x] -= t;
    for (int y = 0; y < nr; y++) {
      if (vy[y]) ly[y] += t;
      else slack[y] -= t;
    }
    for (int y = 0; y < nr; y++) {
      if (vy[y] || slack[y]) continue;
      if (!my[y]) {
        augment(y);
        return;
      }
      vy[y] = 1;
      que.push(my[y]);
    }
  }
}
int main() {
  for(int i = 0; i < nl; i++)
    lx[i] = *max_element(w[i], w[i] + nr);
  for(int i = 0; i < nl; i++) {
    fill_n(vx, nl, 0);
    fill_n(vy, nr, 0);
    fill_n(slack, nr, inf);
    match(i);
  }
}
```

4.5 SW-Mincut

```
struct SW{ // O(V^3) 0-base
  static const int MXN = 514;
  int n,vst[MXN],del[MXN];
```

```
  int edge[MXN][MXN],wei[MXN];
  void init(int _n){
    n = _n;
    for (int i=0; i<n; i++) {
      for (int j=0; j<n; j++)
        edge[i][j] = 0;
      del[i] = 0;
    }
  }
  void add_edge(int u, int v, int w){
    edge[u][v] += w;
    edge[v][u] += w;
  }
  void search(int &s, int &t){
    for (int i=0; i<n; i++)
      vst[i] = wei[i] = 0;
    s = t = -1;
    while (true){
      int mx=-1, cur=0;
      for (int i=0; i<n; i++)
        if (!del[i] && !vst[i] && mx<wei[i])
          cur = i, mx = wei[i];
      if (mx == -1) break;
      vst[cur] = 1;
      s = t;
      t = cur;
      for (int i=0; i<n; i++)
        if (!vst[i] && !del[i]) wei[i] +=
            edge[cur][i];
    }
  }
  int solve(){
    int res = 2147483647;
    for (int i=0,x,y; i<n-1; i++){
      search(x,y);
      res = min(res,wei[y]);
      del[y] = 1;
      for (int j=0; j<n; j++)
        edge[x][j] = (edge[j][x] += edge[y][j]);
    }
    return res;
  }
}graph;
```

4.6 Maximum Matching

```
int n;
vector<int> adj[maxn];
int pa[maxn], match[maxn], st[maxn], color[maxn],
    vis[maxn];
int lca(int u, int v) {
  static int t = 0;
  for (++t; ; swap(u, v)) {
    if (u == 0) continue;
    if (vis[u] == t) return u;
    vis[u] = t;
    u = st[pa[match[u]]];
  }
}
void flower(int u, int v, int l, queue<int> &q) {
  while (st[u] != l) {
    pa[u] = v;
    v = match[u];
    if (color[v] == 1)
      q.push(v), color[v] = 0;
    st[u] = st[v] = l;
    u = pa[v];
  }
}
bool augment(int u, int v) {
  for (int lst; u; v = lst, u = pa[v]) {
    lst = match[u];
    match[u] = v;
    match[v] = u;
  }
}
bool bfs(int u) {
  iota(st, n, 0);
  fill_n(color, n, -1);
  queue<int> q;
  q.push(u), color[u] = 0;
  while (!q.empty()) {
    u = q.front(), q.pop();
    for (int v : adj[u]) {
      if (color[v] == -1){
        pa[v] = u;
        color[v] = 1;
        if (!match[v]) {
          augment(u, v);
          return true;
        }
        q.push(match[v]), color[match[v]] = 0;
      } else if (!color[v] && st[v] != st[u]) {
        int l = lca(v, u);
        flower(v, u, l, q);
        flower(u, v, l, q);
      }
    }
  }
  return false;
}
int blossom() {
  fill_n(pa, n, 0);
  fill_n(match, n, 0);
  int ans = 0;
  for(int i = 0; i < n; ++i)
    if (!match[i] && bfs(i))
      ++ans;
  return ans;
}
```

4.7 Minimum Weight Matching (Clique version)

```
struct Graph {
  // Minimum General Weighted Matching (Perfect
      Match) 0-base
  static const int MXN = 105;

  int n, edge[MXN][MXN];
  int match[MXN],dis[MXN],onstk[MXN];
  vector<int> stk;

  void init(int _n) {
    n = _n;
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        edge[i][j] = 0;
  }
  void add_edge(int u, int v, int w) {
    edge[u][v] = edge[v][u] = w;
  }
  bool SPFA(int u){
    if (onstk[u]) return true;
    stk.PB(u);
    onstk[u] = 1;
    for (int v=0; v<n; v++){
      if (u != v && match[u] != v && !onstk[v]){
        int m = match[v];
        if (dis[m] > dis[u] - edge[v][m] +
            edge[u][v]){
          dis[m] = dis[u] - edge[v][m] + edge[u][v];
          onstk[v] = 1;
          stk.PB(v);
          if (SPFA(m)) return true;
          stk.pop_back();
          onstk[v] = 0;
        }
      }
    }
    onstk[u] = 0;
    stk.pop_back();
    return false;
```

```
  }

  int solve() {
    // find a match
    for (int i=0; i<n; i+=2){
      match[i] = i+1;
      match[i+1] = i;
    }
    while (true){
      int found = 0;
      for (int i=0; i<n; i++)
        dis[i] = onstk[i] = 0;
      for (int i=0; i<n; i++){
        stk.clear();
        if (!onstk[i] && SPFA(i)){
          found = 1;
          while (SZ(stk)>=2){
            int u = stk.back(); stk.pop_back();
            int v = stk.back(); stk.pop_back();
            match[u] = v;
            match[v] = u;
          }
        }
      }
      if (!found) break;
    }
    int ret = 0;
    for (int i=0; i<n; i++)
      ret += edge[i][match[i]];
    ret /= 2;
    return ret;
  }
}graph;
```

4.8 (+1) SW-mincut $O(NM)$

```
// {{{ StoerWagner
const int inf=1000000000;
// should be larger than max.possible mincut
class StoerWagner {
  public:
    int n,mc; // node id in [0,n-1]
    vector<int> adj[MAXN];
    int cost[MAXN][MAXN];
    int cs[MAXN];
    bool merged[MAXN],sel[MAXN];
    // --8<-- include only if cut is explicitly
        needed
      DisjointSet djs;
    vector<int> cut;
    //--8<----------
      StoerWagner(int _n):n(_n),mc(inf),djs(_n) {
        for(int i=0;i<n;i++)
          merged[i]=0;
        for(int i=0;i<n;i++)
          for(int j=0;j<n;j++)
            cost[i][j]=cost[j][i]=0;
      }
    void append(int v,int u,int c) {
      if(v==u) return;
      if(!cost[v][u]&&c) {
        adj[v].PB(u);
        adj[u].PB(v);
      }
      cost[v][u]+=c;
      cost[u][v]+=c;
    }
    void merge(int v,int u) {
      merged[u]=1;
      for(int i=0;i<n;i++)
        append(v,i,cost[u][i]);
      // --8<-- include only if cut is explicitly
          needed
        djs.merge(v,u);
      //--8<----------
    }
    void phase() {
      priority_queue<pii> pq;
      for(int v=0;v<n;v++) {
        if(merged[v]) continue;
        cs[v]=0;
        sel[v]=0;
        pq.push({0,v});
      }
      int v,s,pv;
      while(pq.size()) {
        if(cs[pq.top().S]>pq.top().F) {
          pq.pop();
          continue;
        }
        pv=v;
        v=pq.top().S;
        s=pq.top().F;
        pq.pop();
        sel[v]=1;
        for(int i=0;i<adj[v].size();i++) {
          int u=adj[v][i];
          if(merged[u]||sel[u]) continue;
          cs[u]+=cost[v][u];
          pq.push({cs[u],u});
        }
      }
      if(s<mc) {
        mc=s;
        // --8<-- include only if cut is explicitly
            needed ------
        cut.clear();
        for(int i=0;i<n;i++)
          if(djs.getrep(i)==djs.getrep(v)) cut.PB(i);
        //--8<----------
      }
      merge(v,pv);
    }
    int mincut() {
      if(mc==inf) {
        for(int t=0;t<n-1;t++)
          phase();

        return mc;
      }
      // --8<-- include only if cut is explicitly
          needed
        vector<int> getcut() { // return one side of
            the cut
          mincut();
          return cut;
        }
      //--8<----------
};
// }}}
```

5 Math

5.1 Linear Inverse Table

```
int inv[maxn + 1];
void build(int n) {
  inv[1] = 1;
  for (int i = 2; i <= n; i++)
    inv[i] = (long long) (mod - mod / i) * inv[mod %
        i] % mod;
}
```

5.2 ax+by=gcd

```
typedef pair<int, int> pii;

pii gcd(int a, int b){
  if(b == 0) return make_pair(1, 0);
  else{
    int p = a / b;
    pii q = gcd(b, a % b);
```

```
    return make_pair(q.second, q.first - q.second *
        p);
  }
}
```

5.3 Fast Fourier Transform

```
using cplx = complex<double>;
constexpr double pi = acos(-1.0);
void fft(cplx *a, int n, bool inverse) {
  static cplx tmp[maxn];
  if (n == 1) return;
  copy_n(a, n, tmp);
  for (int i = 0; i < n; i++)
    a[(i & 1) ? (n >> 1) + (i >> 1) : (i >> 1)] =
        tmp[i];
  cplx *a1 = a, *a2 = a + (n >> 1);
  fft(a1, n >> 1, inverse);
  fft(a2, n >> 1, inverse);
  cplx w_base = polar(1.0, 2.0 * pi / n);
  if (inverse)
    w_base = conj(w_base);
  cplx w(1.0);
  for (int i = 0; (i << 1) < n; i++, w *= w_base) {
    tmp[i] = a1[i] + w * a2[i];
    tmp[(n >> 1) + i] = a1[i] - w * a2[i];
  }
  copy_n(tmp, n, a);
}
int mult(cplx *a, int la, cplx *b, int lb, cplx *c) {
  int n = 2;
  while (n < la + lb) n <<= 1;
  fill(a + la, a + n, cplx());
  fill(b + lb, b + n, cplx());
  fft(a, n, false);
  fft(b, n, false);
  for (int i = 0; i < n; i++) c[i] = a[i] * b[i];
  fft(c, n, true);
  for (int i = 0; i < n; i++) c[i] /= n;
  return la + lb - 1;
}
```

5.4 Fast Linear Recurrence

```
int p[maxn], e[maxn], dp[2 * maxn];
vector<int> mul(const vector<int>& v1, const
    vector<int>& v2) {
  int n = v1.size();
  vector<int> v(2 * n);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      v[i + j + 1] = (v[i + j + 1] + (ll) v1[i] *
          v2[j]) % mod;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      v[i + j + 1] = (v[i + j + 1] + (ll) v[i] *
          e[j]) % mod;
  v.erase(v.begin(), v.begin() + n);
  return v;
}
void pre_dp(int n) {
  copy_n(p, n, dp);
  for (int i = n; i < 2 * n; i++) {
    dp[i] = 0;
    for (int j = 0; j < n; j++)
      dp[i] = (dp[i] + (ll) e[j] * dp[i - j - 1]) %
          mod;
  }
}
int solve(int n, ll m) {
  if (m < 2 * n) return dp[m];
  vector<int> vi(e, e + n), va = vi;
  ll dlt = (m - n) / n, rdlt = dlt * n;
  while (dlt) {
    if (dlt & 1)
      vi = mul(vi, va);
    va = mul(va, va);
    dlt >>= 1;
  }
  int ans = 0;
  for (int i = 0; i < n; i++)
    ans = (ans + (ll) vi[i] * dp[m - i - 1 - rdlt])
        % mod;
  return ans;
}
```

5.5 (+1) ntt

```
int P=605028353,root=3,MAXNUM=262144;
// Remember coefficient are mod P
/*
p=a*2^n+1
n   2^n         p             a     root
5   32          97            3     5
6   64          193           3     5
7   128         257           2     3
8   256         257           1     3
9   512         7681          15    17
10  1024        12289         12    11
11  2048        12289         6     11
12  4096        12289         3     11
13  8192        40961         5     3
14  16384       65537         4     3
15  32768       65537         2     3
16  65536       65537         1     3
17  131072      786433        6     10
18  262144      786433        3     10 (605028353,
    2308, 3)
19  524288      5767169       11    3
20  1048576     7340033       7     3
21  2097152     23068673      11    3
22  4194304     104857601     25    3
23  8388608     167772161     20    3
24  16777216    167772161     10    3
25  33554432    167772161     5     3 (1107296257,
    33, 10)
26  67108864    469762049     7     3
27  134217728   2013265921    15    31
*/
int bigmod(long long a,int b){
  if(b==0)return 1;
  return (bigmod((a*a)%P,b/2)*(b%2?a:1ll))%P;
}
int inv(int a,int b){
  if(a==1)return 1;
  return (((long long)(a-inv(b%a,a))*b+1)/a)%b;
}
std::vector<long long> ps(MAXNUM);
std::vector<int> rev(MAXNUM);
struct poly{
  std::vector<unsigned int> co;
  int n;//polynomial degree = n
  poly(int d){n=d;co.resize(n+1,0);}
  void trans2(int NN){
    int r=0,st,N;
    unsigned int a,b;
    while((1<<r)<(NN>>1))++r;
    for(N=2;N<=NN;N<<=1,--r){
      for(st=0;st<NN;st+=N){
        int i,ss=st+(N>>1);
        for(i=(N>>1)-1;i>=0;--i){
          a=co[st+i]; b=(ps[i<<r]*co[ss+i])%P;
          co[st+i]=a+b; if(co[st+i]>=P)co[st+i]-=P;
          co[ss+i]=a+P-b; if(co[ss+i]>=P)co[ss+i]-=P;
        }
      }
    }
  }
  void trans1(int NN){
    int r=0,st,N;
    unsigned int a,b;
    for(N=NN;N>1;N>>=1,++r){
      for(st=0;st<NN;st+=N){
```

```
        int i,ss=st+(N>>1);
        for(i=(N>>1)-1;i>=0;--i){
          a=co[st+i]; b=co[ss+i];
          co[st+i]=a+b; if(co[st+i]>=P)co[st+i]-=P;
          co[ss+i]=((a+P-b)*ps[i<<r])%P;
        }
      }
    }
  }
  poly operator*(const poly& _b)const{
    poly a=*this,b=_b;
    int k=n+b.n,i,N=1;
    while(N<=k)N*=2;
    a.co.resize(N,0); b.co.resize(N,0);
    int r=bigmod(root,(P-1)/N),Ni=inv(N,P);
    ps[0]=1;
    for(i=1;i<N;++i)ps[i]=(ps[i-1]*r)%P;
    a.trans1(N);b.trans1(N);
    for(i=0;i<N;++i)a.co[i]=((long
        long)a.co[i]*b.co[i])%P
      ;
    r=inv(r,P);
    for(i=1;i<N/2;++i)std::swap(ps[i],ps[N-i]);
    a.trans2(N);
    for(i=0;i<N;++i)a.co[i]=((long
        long)a.co[i]*Ni)%P;
    a.n=n+_b.n; return a;
  }
};
```

5.6 Mod

```
/// _fd(a,b)  floor(a/b).
/// _rd(a,m)  a-floor(a/m)*m.
/// _pv(a,m,r) largest x s.t x<=a && x%m == r.
/// _nx(a,m,r) smallest x s.t x>=a && x%m == r.
/// _ct(a,b,m,r) |A| , A = { x : a<=x<=b && x%m == r
    }.

int _fd(int a,int b){ return a<0?(-~a/b-1):a/b; }
int _rd(int a,int m){ return a-_fd(a,m)*m; }
int _pv(int a,int m,int r)
{
    r=(r%m+m)%m;
    return _fd(a-r,m)*m+r;
}
int _nt(int a,int m,int r)
{
    m=abs(m);
    r=(r%m+m)%m;
    return _fd(a-r-1,m)*m+r+m;
}
int _ct(int a,int b,int m,int r)
{
    m=abs(m);
    a=_nt(a,m,r);
    b=_pv(b,m,r);
    return (a>b)?0:((b-a+m)/m);
}
```

5.7 Miller Rabin

```
ll modpow(ll a, ll b, ll m) {
    ll r = 1;
    while (b) {
        if (b & 1) r = (__int128) r * a % m;
        a = (__int128) a * a % m;
        b >>= 1;
    }
    return r;
}
bool miller_rabin(ll n, ll a) {
    if (__gcd(a, n) == n) return true;
    if (__gcd(a, n) != 1) return false;
    ll d = n - 1, r = 0;
    while (~d & 1) r++, d >>= 1;
    ll res = modpow(a, d, n);
    if (res == 1 || res == n - 1) return true;
    while (r--) {
        res = (__int128) res * res % n;
        if(res == n - 1) return true;
    }
    return false;
}
bool is_prime(ll n) {
    // n < 4,759,123,141        {2, 7, 61}
    // n < 1,122,004,669,633    {2, 13, 23, 1662803}
    // n < 3,474,749,660,383    {primes <= 13}
    for (ll x : {2, 325, 9375, 28178, 450775,
        9780504, 1795265022})
        if (!miller_rabin(n, x))
            return false;
    return true;
}
```

5.8 Pollard Rho

```
// does not work when n is prime
ll pollard_rho(ll n) {
  if (!(n & 1)) return 2;
  while (true) {
    ll y = 2, x = rand() % (n - 1) + 1, res = 1;
    for(int sz = 2; res == 1; sz *= 2) {
      for(int i = 0; i < sz && res <= 1; i++) {
        x = ((__int128) x * x + 1) % n;
        res = __gcd(abs(x - y), n);
      }
      y = x;
    }
    if (res != 0 && res != n) return res;
  }
}
```

5.9 Algorithms about Primes

12721, 13331, 14341, 75577, 123457, 222557, 556679, 999983, 98789101, 100102021, 987654361, 987777733, 999888733, 999991231, 999991921, 999997771, 1000512343, 1001010013, 1010101333, 1010102101, 1076767633, 1097774749, 1000000000039, 1000000000000037, 2305843009213693951, 4611686018427387847, 9223372036854775783, 18446744073709551557

```
int p_tbl[MX];
vector<int> primes;
void sieve() {
  p_tbl[1] = 1;
  for (int i=2; i<MX; i++) {
    if (!p_tbl[i]) {
      p_tbl[i] = i;
      primes.PB(i);
    }
    for (auto p : primes) {
      if (i*p >= M) break;
      p_tbl[i*p] = p;
      if (i%p==0)
        break;
    }
  }
}

vector<int> factor(int x) {
  vector<int> fac{1};
  while (x > 1) {
    int fn=SZ(fac), p=p_tbl[x], pos=0;
    while (x%p == 0) {
      x /= p;
      for (int i=0; i<fn; i++)
        fac.PB(fac[pos++]*p);
    }
  }
```

```
  return fac;
}
```

5.10 Count Coprime Pairs

```
// # {(x, y) in [1, n] * [1, m] | gcd(x, y) = 1}
int mu[maxn + 1];
ll sum[maxn + 1];
vector<int> prime;
bool is_prime[maxn + 1];
void preprocess() {
    fill(is_prime + 2, is_prime + maxn + 1, true);
    mu[1] = 1;
    for (int i = 2; i <= maxn; i++) {
        if (is_prime[i]) {
            prime.push_back(i);
            mu[i] = -1;
        }
        for (int p : prime) {
            if (p * i > maxn)
                break;
            is_prime[p * i] = false;
            if (i % p == 0) {
                mu[p * i] = 0;
                break;
            }
            mu[p * i] = -mu[i];
        }
    }
    partial_sum(mu + 1, mu + maxn + 1, sum + 1);
}
ll count_coprime_pairs(int n, int m) {
    ll ans = 0;
    for(int i = 1, p; i <= min(n, m); i = p + 1) {
        p = min(n / (n / i), m / (m / i));
        ans += (sum[p] - sum[i - 1]) * (n / i) * (m
            / i);
    }
    return ans;
}
```

5.11 (+1) PolynomialGenerator

```
class PolynomialGenerator {
  /* for a nth-order polynomial f(x), *
   * given f(0), f(1), ..., f(n) *
   * express f(x) as sigma_i{c_i*C(x,i)} */
  public:
    int n;
    vector<long long> coef;
    // initialize and calculate f(x), vector _fx
        should be
    // filled with f(0) to f(n)
      PolynomialGenerator(int _n,vector<long long>
          _fx):n(_n
          ),coef(_fx) {
        for(int i=0;i<n;i++)
          for(int j=n;j>i;j--)
            coef[j]-=coef[j-1];
      }
    // evaluate f(x), runs in O(n)
    long long eval(int x) {
      long long m=1,ret=0;
      for(int i=0;i<=n;i++) {
        ret+=coef[i]*m;
        m=m*(x-i)/(i+1);
      }
      return ret;
    }
};
```

5.12 Pseudoinverse of Square matrix

```
Mat pinv(Mat m)
{
  Mat res = I;

  FZ(used);
  for(int i=0; i<W; i++)
  {
    int piv = -1;
    for(int j=0; j<W; j++)
    {
      if(used[j]) continue;
      if(abs(m.v[j][i]) > EPS)
      {
        piv = j;
        break;
      }
    }
    if(piv == -1)
      continue;
    used[i] = true;
    swap(m.v[piv], m.v[i]);
    swap(res.v[piv], res.v[i]);

    ld rat = m.v[i][i];
    for(int j=0; j<W; j++)
    {
      m.v[i][j] /= rat;
      res.v[i][j] /= rat;
    }

    for(int j=0; j<W; j++)
    {
      if(j == i) continue;
      rat = m.v[j][i];
      for(int k=0; k<W; k++)
      {
        m.v[j][k] -= rat * m.v[i][k];
        res.v[j][k] -= rat * res.v[i][k];
      }
    }
  }

  for(int i=0; i<W; i++)
  {
    if(used[i]) continue;
    for(int j=0; j<W; j++)
      res.v[i][j] = 0;
  }

  return res;
}
```

5.13 Simplex

```
const int maxn = 111;
const int maxm = 111;
const double eps = 1E-10;

double a[maxn][maxm], b[maxn], c[maxm],
    d[maxn][maxm];
double x[maxm];
int ix[maxn + maxm]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
//
// usage :
// value = simplex(a, b, c, N, M);
double simplex(double a[maxn][maxm], double b[maxn],
    double c[maxm], int n, int m) {
    ++m;
    int r = n, s = m - 1;
    memset(d, 0, sizeof(d));
    for (int i = 0; i < n + m; ++i) ix[i] = i;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j)
            d[i][j] = -a[i][j];
```

```
        d[i][m - 1] = 1;
        d[i][m] = b[i];
        if (d[r][m] > d[i][m]) r = i;
    }
    for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
    d[n + 1][m - 1] = -1;
    for (double dd;; ) {
        if (r < n) {
            int t = ix[s];
            ix[s] = ix[r + m]; ix[r + m] = t;
            d[r][s] = 1.0 / d[r][s];
            for (int j = 0; j <= m; ++j)
                if (j != s) d[r][j] *= -d[r][s];
            for (int i = 0; i <= n + 1; ++i)
                if (i != r) {
                    for (int j = 0; j <= m; ++j)
                        if (j != s)
                            d[i][j] +=
                                d[r][j]*d[i][s];
                    d[i][s] *= d[r][s];
                }
        }
        r = -1; s = -1;
        for (int j = 0; j < m; ++j)
            if (s < 0 || ix[s] > ix[j]) {
                if (d[n + 1][j] > eps || (d[n +
                    1][j] > -eps && d[n][j] > eps))
                    s = j;
            }
        if (s < 0) break;
        for (int i=0; i<n; ++i) if (d[i][s] < -eps) {
            if (r < 0 || (dd = d[r][m] / d[r][s] -
                d[i][m] / d[i][s]) < -eps || (dd <
                eps && ix[r + m] > ix[i + m])) r = i;
        }
        if (r < 0) return -1; // not bounded
    }
    if (d[n + 1][m] < -eps) return -1; // not
        executable
    double ans = 0;
    for(int i=0; i<m; i++) x[i] = 0;
    for (int i = m; i < n + m; ++i) { // the missing
        enumerated x[i] = 0
        if (ix[i] < m - 1)
        {
            ans += d[i - m][m] * c[ix[i]];
            x[ix[i]] = d[i-m][m];
        }
    }
    return ans;
}
```

5.14 Lucas's Theorem

For non-negative integers n, m and a prime p,

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod{p},$$

where m_i and n_i are base p expansions of m and n, respectively.

5.15 Pick's Theorem

Given a simple polygon of which all vertices are lattice points with area A, i interior lattice points, and b boundary lattice points, we have

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

5.16 Kirchhoff's Theorem

Given a simple graph G with n vertices. Let

$$L_{i,j} \coloneqq \begin{cases} deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}.$$

Then for any i and j, # of spanning trees of G is

$$\det \widehat{L_{i,j}}.$$

6 Geometry

6.1 Point operators

```
#define x first
#define y second

#define cpdd const pdd
struct pdd : pair<double, double> {
    using pair<double, double>::pair;

    pdd operator + (cpdd &p) const {
        return {x+p.x, y+p.y};
    }

    pdd operator - () const {
        return {-x, -y};
    }

    pdd operator - (cpdd &p) const {
        return (*this) + (-p);
    }

    pdd operator * (double f) const {
        return {f*x, f*y};
    }

    double operator * (cpdd &p) const {
        return x*p.x + y*p.y;
    }
};

double abs(cpdd &p) { return hypot(p.x, p.y); }
double arg(cpdd &p) { return atan2(p.y, p.x); }
double cross(cpdd &p, cpdd &q) { return p.x*q.y -
    p.y*q.x; }
double cross(cpdd &p, cpdd &q, cpdd &o) { return
    cross(p-o, q-o); }
pdd operator * (double f, cpdd &p) { return p*f; }
    // !! Not f*p !!
```

6.2 Intersection of two circles

```
using ld = double;
vector<pdd> interCircle(pdd o1, double r1, pdd o2,
    double r2) {
  ld d2 = (o1 - o2) * (o1 - o2);
  ld d = sqrt(d2);
  if (d < abs(r1-r2)) return {};
  if (d > r1+r2) return {};
  pdd u = 0.5*(o1+o2) +
      ((r2*r2-r1*r1)/(2*d2))*(o1-o2);
  double A = sqrt((r1+r2+d) * (r1-r2+d) * (r1+r2-d)
      * (-r1+r2+d));
  pdd v = A / (2*d2) * pdd(o1.S-o2.S, -o1.F+o2.F);
  return {u+v, u-v};
}
```

6.3 Intersection of two lines

```
const double EPS = 1e-9;

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2, bool
    &res){
  double f1 = cross(p2, q1, p1);
  double f2 = -cross(p2, q2, p1);
  double f = (f1 + f2);

  if(fabs(f) < EPS) {
    res = false;
    return {};
    }

  res = true;
  return (f2 / f) * q1 + (f1 / f) * q2;
}
```

6.4 Half Plane Intersection

```
struct Point {
  double x, y;
  Point (double _x = 0, double _y = 0) : x(_x),
      y(_y) {}
  bool operator < (const Point &rhs) const {
    return x != rhs.x ? x < rhs.x : y < rhs.y;
  }
};
using Vector = Point;
Point operator + (Point p, Vector v) {
  return Point(p.x + v.x, p.y + v.y);
}
Vector operator - (Point a, Point b) {
  return Vector(a.x - b.x, a.y - b.y);
}
Vector operator * (Vector v, double p) {
  return Vector(v.x * p, v.y * p);
}
Vector operator / (Vector v, double p) {
  return Vector(v.x / p, v.y / p);
}
double cross(Vector a, Vector b) {
  return a.x * b.y - a.y * b.x;
}
double dot(Vector a, Vector b) {
  return a.x * b.x + a.y * b.y;
}
int dcmp(double x) {
  return fabs(x) < 1e-10 ? 0 : x > 0 ? 1 : -1;
}
struct Line {
  Point p;
  Vector v;
  double ang;
  Line() {}
  Line(Point _p, Vector _v) : p(_p), v(_v) {
    ang = atan2(_v.y, _v.x);
  }
  bool operator < (const Line &rhs) const {
    return ang < rhs.ang;
  }
};
bool on_left(Line l, Point p) {
  return cross(l.v, p - l.p) > 0;
}
Point get_line_intersec(Line a, Line b) {
  return a.p + a.v * (cross(b.v, a.p - b.p) /
      cross(a.v, b.v));
}
vector<Point> half_plane_intersec(vector<Line> ls) {
  int n = ls.size();
  sort(ls.begin(), ls.end());
  int f, r;
  vector<Point> p(n), ans;
  vector<Line> q(n);
  q[f = r = 0] = ls[0];
  for (int i = 1; i < n; i++) {
    while (f < r && !on_left(ls[i], p[r - 1])) r--;
    while (f < r && !on_left(ls[i], p[f])) f++;
    q[++r] = ls[i];
    if (dcmp(cross(q[r].v, q[r - 1].v)) == 0) {
      r--;
      if (on_left(q[r], ls[i].p)) q[r] = ls[i];
    }
    if (f < r)
      p[r - 1] = get_line_intersec(q[r - 1], q[r]);
  }
  while (f < r && !on_left(q[f], p[r - 1])) r--;
  if (r - f <= 1) return ans;
  p[r] = get_line_intersec(q[r], q[f]);
  for (int i = f; i <= r; i++) ans.push_back(p[i]);
  return ans;
}
```

6.5 2D Convex Hull

```
vector<pdd> convex_hull(vector<pdd> pt){
  sort(pt.begin(),pt.end());
  int top=0;
  vector<pdd> stk(2*pt.size());
  for (int i=0; i<(int)pt.size(); i++){
    while (top >= 2 &&
        cross(stk[top-1],pt[i],stk[top-2]) <= 0)
      top--;
    stk[top++] = pt[i];
  }
  for (int i=pt.size()-2, t=top+1; i>=0; i--){
    while (top >= t && cross(stk[top-1],pt[i],
        stk[top-2]) <= 0)
      top--;
    stk[top++] = pt[i];
  }
  stk.resize(top-1);
  return stk;
}
```

6.6 3D Convex Hull

```
// return the faces with pt indexes
int flag[MXN][MXN];
struct Point{
  ld x,y,z;
  Point operator - (const Point &b) const {
    return (Point){x-b.x,y-b.y,z-b.z};
  }
  Point operator * (const ld &b) const {
    return (Point){x*b,y*b,z*b};
  }
  ld len() const { return sqrtl(x*x+y*y+z*z); }
  ld dot(const Point &a) const {
    return x*a.x+y*a.y+z*a.z;
  }
  Point operator * (const Point &b) const {
    return
        (Point){y*b.z-b.y*z,z*b.x-b.z*x,x*b.y-b.x*y};
  }
};
Point ver(Point a, Point b, Point c) {
  return (b - a) * (c - a);
}
vector<Face> convex_hull_3D(const vector<Point> pt) {
  int n = SZ(pt);
  REP(i,n) REP(j,n)
    flag[i][j] = 0;

  vector<Face> now;
  now.push_back((Face){0,1,2});
  now.push_back((Face){2,1,0});
  int ftop = 0;
  for (int i=3; i<n; i++){
    ftop++;
    vector<Face> next;
    REP(j, SZ(now)) {
      Face& f=now[j];
```

```
      ld d=(pt[i]-pt[f.a]).dot(ver(pt[f.a], pt[f.b],
          pt[f.c]));
      if (d <= 0) next.push_back(f);
      int ff = 0;
      if (d > 0) ff=ftop;
      else if (d < 0) ff=-ftop;
      flag[f.a][f.b] = flag[f.b][f.c] =
          flag[f.c][f.a] = ff;
    }
    REP(j, SZ(now)) {
      Face& f=now[j];
      if (flag[f.a][f.b] > 0 and flag[f.a][f.b] !=
          flag[f.b][f.a])
        next.push_back((Face){f.a,f.b,i});
      if (flag[f.b][f.c] > 0 and flag[f.b][f.c] !=
          flag[f.c][f.b])
        next.push_back((Face){f.b,f.c,i});
      if (flag[f.c][f.a] > 0 and flag[f.c][f.a] !=
          flag[f.a][f.c])
        next.push_back((Face){f.c,f.a,i});
    }
    now=next;
  }
  return now;
}
```

6.7 Minimum Covering Circle

```
struct Mcc{
  // return pair of center and r^2
  static const int MAXN = 1000100;
  int n;
  pdd p[MAXN],cen;
  double r2;

  void init(int _n, pdd _p[]){
    n = _n;
    memcpy(p,_p,sizeof(pdd)*n);
  }
  double sqr(double a){ return a*a; }
  double abs2(pdd a){ return a*a; }
  pdd center(pdd p0, pdd p1, pdd p2) {
    pdd a = p1-p0;
    pdd b = p2-p0;
    double c1=abs2(a)*0.5;
    double c2=abs2(b)*0.5;
    double d = a % b;
    double x = p0.x + (c1 * b.y - c2 * a.y) / d;
    double y = p0.y + (a.x * c2 - b.x * c1) / d;
    return pdd(x,y);
  }

  pair<pdd,double> solve(){
    random_shuffle(p,p+n);
    r2=0;
    for (int i=0; i<n; i++){
      if (abs2(cen-p[i]) <= r2) continue;
      cen = p[i];
      r2 = 0;
      for (int j=0; j<i; j++){
        if (abs2(cen-p[j]) <= r2) continue;
        cen = 0.5 * (p[i]+p[j]);
        r2 = abs2(cen-p[j]);
        for (int k=0; k<j; k++){
          if (abs2(cen-p[k]) <= r2) continue;
          cen = center(p[i],p[j],p[k]);
          r2 = abs2(cen-p[k]);
        }
      }
    }
    return {cen,r2};
  }
}mcc;
```

6.8 KDTree (Nearest Point)

```
const int MXN = 100005;

struct KDTree {
  struct Node {
    int x,y,x1,y1,x2,y2;
    int id,f;
    Node *L, *R;
  }tree[MXN];
  int n;
  Node *root;

  long long dis2(int x1, int y1, int x2, int y2) {
    long long dx = x1-x2;
    long long dy = y1-y2;
    return dx*dx+dy*dy;
  }
  static bool cmpx(Node& a, Node& b){ return
      a.x<b.x; }
  static bool cmpy(Node& a, Node& b){ return
      a.y<b.y; }
  void init(vector<pair<int,int>> ip) {
    n = ip.size();
    for (int i=0; i<n; i++) {
      tree[i].id = i;
      tree[i].x = ip[i].first;
      tree[i].y = ip[i].second;
    }
    root = build_tree(0, n-1, 0);
  }
  Node* build_tree(int L, int R, int dep) {
    if (L>R) return nullptr;
    int M = (L+R)/2;
    tree[M].f = dep%2;
    nth_element(tree+L, tree+M, tree+R+1, tree[M].f
        ? cmpy : cmpx);
    tree[M].x1 = tree[M].x2 = tree[M].x;
    tree[M].y1 = tree[M].y2 = tree[M].y;

    tree[M].L = build_tree(L, M-1, dep+1);
    if (tree[M].L) {
      tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
      tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
      tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
      tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
    }

    tree[M].R = build_tree(M+1, R, dep+1);
    if (tree[M].R) {
      tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
      tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
      tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
      tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
    }

    return tree+M;
  }
  int touch(Node* r, int x, int y, long long d2){
    long long dis = sqrt(d2)+1;
    if (x<r->x1-dis || x>r->x2+dis || y<r->y1-dis ||
        y>r->y2+dis)
      return 0;
    return 1;
  }
  void nearest(Node* r, int x, int y, int &mID, long
      long &md2) {
    if (!r || !touch(r, x, y, md2)) return;
    long long d2 = dis2(r->x, r->y, x, y);
    if (d2 < md2 || (d2 == md2 && mID < r->id)) {
      mID = r->id;
      md2 = d2;
    }
    // search order depends on split dim
    if ((r->f == 0 && x < r->x) ||
        (r->f == 1 && y < r->y)) {
      nearest(r->L, x, y, mID, md2);
      nearest(r->R, x, y, mID, md2);
    } else {
```

```
      nearest(r->R, x, y, mID, md2);
      nearest(r->L, x, y, mID, md2);
    }
  }
  int query(int x, int y) {
    int id = 1029384756;
    long long d2 = 102938475612345678LL;
    nearest(root, x, y, id, d2);
    return id;
  }
}tree;
```

6.9 Triangulation

```
bool inCircle(pdd a, pdd b, pdd c, pdd d) {
    b = b - a;
    c = c - a;
    d = d - a;
    if (cross(b, c) < 0) swap(b, c);
    double m[3][3] = {
        {b.x, b.y, b*b},
        {c.x, c.y, c*c},
        {d.x, d.y, d*d}
    };

    double det = m[0][0] * (m[1][1]*m[2][2] -
        m[1][2]*m[2][1])
               + m[0][1] * (m[1][2]*m[2][0] -
                   m[1][0]*m[2][2])
               + m[0][2] * (m[1][0]*m[2][1] -
                   m[1][1]*m[2][0]);
    return det < 0;
}

bool intersect(pdd a, pdd b, pdd c, pdd d) {
    return cross(b, c, a) * cross(b, d, a) < 0 and
        cross(d, a, c) * cross(d, b, c) < 0;
}


const double EPS = 1e-12;
struct Triangulation {
    static const int MXN = 1e5+5;
    int N;
    vector<int> ord;
    vector<pdd> pts;
    set<int> E[MXN];
    vector<vector<int>> solve(vector<pdd> p) {
        N = SZ(p);
        ord.resize(N);
        for (int i=0; i<N; i++) {
            E[i].clear();
            ord[i] = i;
        }
        sort(ALL(ord), [&p](int i, int j) {
            return p[i] < p[j];
        });

        pts.resize(N);
        for (int i=0; i<N; i++) pts[i] = p[ord[i]];

        go(0, N);

        vector<vector<int>> res(N);
        for (int i=0; i<N; i++) {
            int o = ord[i];
            for (auto x: E[i]) {
                res[o].PB(ord[x]);
            }
        }
        return res;
    }

    void add_edge(int u, int v) {
        E[u].insert(v);
        E[v].insert(u);
    }

    void remove_edge(int u, int v) {
        E[u].erase(v);
        E[v].erase(u);
    }

    void go(int l, int r) {
        int n = r - l;

        if (n <= 3) {
            for (int i=l; i<r; i++)
                for (int j=i+1; j<r; j++)
                    add_edge(i, j);
            return;
        }
        int md = (l+r)/2;

        go(l, md);
        go(md, r);

        int il = l, ir = r-1;
        while (1) {
            int nx = -1;
            for (auto i: E[il]) {
                double cs = cross(pts[il], pts[i],
                    pts[ir]);
                if (cs > EPS ||
                    (abs(cs) < EPS and
                        abs(pts[i]-pts[ir]) <
                        abs(pts[il]-pts[ir]))) {
                    nx = i;
                    break;
                }
            }
            if (nx != -1) {
                il = nx;
                continue;
            }
            for (auto i: E[ir]) {
                double cs = cross(pts[ir], pts[i],
                    pts[il]);
                if (cs < -EPS ||
                    (abs(cs) < EPS and
                        abs(pts[i]-pts[il]) <
                        abs(pts[ir]-pts[il]))) {
                    nx = i;
                    break;
                }
            }

            if (nx != -1) {
                ir = nx;
            } else break;
        }

        add_edge(il, ir);

        while (1) {
            int nx = -1;
            bool is2 = false;

            for (int i: E[il]) {
                if (cross(pts[il], pts[i], pts[ir])
                    < -EPS and
                    (nx == -1 or inCircle(pts[il],
                        pts[ir], pts[nx], pts[i])))
                        nx = i;
            }

            for (int i: E[ir]) {
                if (cross(pts[ir], pts[i], pts[il])
                    > EPS and
                    (nx == -1 or inCircle(pts[il],
                        pts[ir], pts[nx], pts[i])))
                        nx = i, is2 = 1;
            }
```

```
            if (nx == -1) break;

            int a = il, b = ir;
            if (is2) swap(a, b);

            for (auto i: E[a]) {
                if (intersect(pts[a], pts[i],
                    pts[b], pts[nx])) {
                    remove_edge(a, i);
                }
            }
            if (is2) {
                add_edge(il, nx);
                ir = nx;
            } else {
                add_edge(ir, nx);
                il = nx;
            }
        }
    }
} tri;
```

7 Stringology

7.1 Suffix Array

```
char str[maxn]; // need null character
// sa[0]="", ht[i+1]=lcp(sa[i],sa[i+1])
int sa[maxn], cnt[maxn], x[maxn], y[maxn], ht[maxn];
void build_sa(int n, int m) {
  copy_n(str, n + 1, x);
  fill_n(cnt, m, 0);
  for (int i = 0; i <= n; i++)
    cnt[x[i]]++;
  partial_sum(cnt, cnt + m, cnt);
  for (int i = n; i >= 0; i--)
    sa[--cnt[x[i]]] = i;
  for (int k = 1; ; k <<= 1, m = p) {
    int p = 0;
    for (int i = n - k + 1; i <= n; i++)
      y[p++] = i;
    for (int i = 0; i <= n; i++)
      if (sa[i] >= k)
        y[p++] = sa[i] - k;
    fill_n(cnt, m, 0);
    for (int i = 0; i <= n; i++)
      cnt[x[y[i]]]++;
    partial_sum(cnt, cnt + m, cnt);
    for (int i = n; i >= 0; i--)
      sa[--cnt[x[y[i]]]] = y[i];
    copy_n(y, n + 1, x);
    p = 1;
    x[sa[0]] = 0;
    for (int i = 1; i <= n; i++)
      x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] &&
          y[sa[i] + k] == y[sa[i - 1] + k] ? p - 1 :
          p++);
    if (p >= n + 1) break;
  }
}
void build_ht(int n) {
  for (int i = 1; i <= n; i++)
    x[sa[i]] = i;
  for (int i = 0, h = 0; i < n; i++) {
    int j = sa[x[i] - 1];
    if (h) h--;
    while (str[i + h] == str[j + h]) h++;
    ht[x[i]] = h;
  }
}
```

7.2 Suffix Array (SAIS TWT514)

```
struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
static const int MXN = 300010;
bool _t[MXN*2];
int _s[MXN*2], _sa[MXN*2], _c[MXN*2], x[MXN],
    _p[MXN], _q[MXN*2], hei[MXN], r[MXN];
int operator [] (int i){ return _sa[i]; }
void build(int *s, int n, int m){
  memcpy(_s, s, sizeof(int) * n);
  sais(_s, _sa, _p, _q, _t, _c, n, m);
  mkhei(n);
}
void mkhei(int n){
  REP(i,n) r[_sa[i]] = i;
  hei[0] = 0;
  REP(i,n) if(r[i]) {
    int ans = i>0 ? max(hei[r[i-1]] - 1, 0) : 0;
    while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans++;
    hei[r[i]] = ans;
  }
}
void sais(int *s, int *sa, int *p, int *q, bool *t,
    int *c, int n, int z){
  bool uniq = t[n-1] = true, neq;
  int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
      lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
  memcpy(x, c, sizeof(int) * z); \
  XD; \
  memcpy(x + 1, c, sizeof(int) * (z - 1)); \
  REP(i,n) if(sa[i] && !t[sa[i]-1]) \
      sa[x[s[sa[i]-1]]++] = sa[i]-1; \
  memcpy(x, c, sizeof(int) * z); \
  for(int i = n - 1; i >= 0; i--) if(sa[i] && \
      t[sa[i]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
  MS0(c, z);
  REP(i,n) uniq &= ++c[s[i]] < 2;
  REP(i,z-1) c[i+1] += c[i];
  if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return; }
  for(int i = n - 2; i >= 0; i--) t[i] =
      (s[i]==s[i+1] ? t[i+1] : s[i]<s[i+1]);
  MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1])
      sa[--x[s[i]]]=p[q[i]=nn++]=i);
  REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1]) {
    neq = lst < 0 || memcmp(s + sa[i], s + lst,
        (p[q[sa[i]] + 1] - sa[i]) * sizeof(int));
    ns[q[lst=sa[i]]]=nmxz+=neq;
  }
  sais(ns, nsa, p + nn, q + n, t + n, c + z, nn,
      nmxz + 1);
  MAGIC(for(int i = nn - 1; i >= 0; i--)
      sa[--x[s[p[nsa[i]]]]] = p[nsa[i]]);
}
}sa;

void suffix_array(int* ip, int len) {
// should padding a zero in the back
// s is int array, n is array length
// s[0..n-1] != 0, and s[n] = 0
// resulting SA will be length n+1
ip[len++] = 0;
sa.build(ip, len, 128);
// original 1-base
for (int i=0; i<l; i++) {
  hei[i] = sa.hei[i + 1];
  sa[i] = sa._sa[i + 1];
}
}
```

7.3 Aho-Corasick Algorithm

```
struct ACautomata{
  struct Node{
    int cnt,dp;
    Node *go[26], *fail;
    Node (){
```

```
      cnt = 0;
      dp = -1;
      memset(go,0,sizeof(go));
      fail = 0;
    }
  };

  Node *root, pool[1048576];
  int nMem;

  Node* new_Node(){
    pool[nMem] = Node();
    return &pool[nMem++];
  }
  void init(){
    nMem = 0;
    root = new_Node();
  }
  void add(const string &str){
    insert(root,str,0);
  }
  void insert(Node *cur, const string &str, int pos){
    if (pos >= (int)str.size()){
      cur->cnt++;
      return;
    }
    int c = str[pos]-'a';
    if (cur->go[c] == 0){
      cur->go[c] = new_Node();
    }
    insert(cur->go[c],str,pos+1);
  }
  void make_fail(){
    queue<Node*> que;
    que.push(root);
    while (!que.empty()){
      Node* fr=que.front();
      que.pop();
      for (int i=0; i<26; i++){
        if (fr->go[i]){
          Node *ptr = fr->fail;
          while (ptr && !ptr->go[i]) ptr = ptr->fail;
          if (!ptr) fr->go[i]->fail = root;
          else fr->go[i]->fail = ptr->go[i];
          que.push(fr->go[i]);
        }
      }
    }
  }
};
```

7.4 KMP

```
#include<bits/stdc++.h>
using namespace std;

void build_fail_function(string B, int *fail) {
    int len = B.length(), pos;
    pos = fail[0] = -1;
    for (int i = 1; i < len; i ++) {
        while (pos != -1 and B[pos + 1] != B[i])
            pos = fail[pos];
        if (B[pos + 1] == B[i]) pos ++;
        fail[i] = pos;
    }
}

void match(string A, string B, int *fail) {
    int lenA = A.length(), lenB = B.length();
    int pos = -1;
    for (int i = 0; i < lenA; i ++) {
        while (pos != -1 and B[pos + 1] != A[i])
            pos = fail[pos];

        if (B[pos + 1] == A[i]) pos ++;

        if (pos == lenB - 1) {
            // Match ! A[i - lenB + 1, i] = B
            pos = fail[pos];
        }
    }
}
```

7.5 Z value

```
void Zval(const char *s, int len, int *z) {
    z[0] = 0;
    for (int b=0, i=1; i<len; i++) {
        z[i] = max(min(z[i-b], z[b] + b - i), 0);
        while (s[i + z[i]] == s[z[i]]) z[i] ++;
        if (i+z[i] > b+z[b]) b=i;
    }
}
```

7.6 Z value (palindrome ver.)

```
void Zpal(const char *s, int len, int *z) {
    // Only odd palindrome len is considered
    // z[i] means that the longest odd palindrom
        centered at
    // i is [i-z[i] .. i+z[i]]
    z[0] = 0;
    for (int b=0, i=1; i<len; i++) {
        if (z[b] + b >= i) z[i] = min(z[2*b-i],
            b+z[b]-i);
        else z[i] = 0;
        while (i+z[i]+1 < len and i-z[i]-1 >= 0 and
               s[i+z[i]+1] == s[i-z[i]-1]) z[i] ++;
        if (z[i] + i > z[b] + b) b = i;
    }
}
```

7.7 Palindromic Tree

```
struct Node {
  int fail, ch[26], len;
  long long dp;
  Node(): fail(), ch(), len(), dp() {}
};
char s[maxn + 1];
Node mem[maxn + 2];
int pmem, last;
int new_node() {
  int id = pmem++;
  mem[id] = Node();
  return id;
}
void init() {
  pmem = 0;
  int a = new_node(), b = new_node();
  mem[a].fail = b;
  mem[b].len = -1;
  last = b;
}
void insert(int i, int c) {
  c -= 'a';
  int p = last, np;
  while (s[i - mem[p].len - 1] != s[i])
    p = mem[p].fail;
  if (!mem[p].ch[c]) {
    np = new_node();
    mem[np].len = mem[p].len + 2;
    int q = mem[p].fail;
    while (s[i - mem[q].len - 1] != s[i])
      q = mem[q].fail;
    mem[np].fail = mem[q].ch[c];
    mem[p].ch[c] = np;
  }
  else np = mem[p].ch[c];
  mem[np].dp++;
  last = np;
}
```

7.8 Lexicographically Smallest Rotation

```
string mcp(string s){
  int n = s.length();
  s += s;
  int i=0, j=1;
  while (i<n && j<n){
    int k = 0;
    while (k < n && s[i+k] == s[j+k]) k++;
    if (s[i+k] <= s[j+k]) j += k+1;
    else i += k+1;
    if (i == j) j++;
  }
  int ans = i < n ? i : j;
  return s.substr(ans, n);
}
```

7.9 Suffix Automaton

```
// par : fail link
// val : a topological order ( useful for DP )
// go[x] : automata edge ( x is integer in [0,26) )
struct State {
  int par, go[26], val;
  State(): par(), go(), val() {}
  State(int _val): par(), go(), val(_val) {}
};
vector<State> vec;
int root, tail;
void init(int arr[], int len) {
  vec.resize(2);
  vec[0] = vec[1] = State(0);
  root = tail = 1;
  for (int i = 0; i < len; i++)
    extend(arr[i]);
}
void extend(int w) {
  int p = tail, np = vec.size();
  vec.push_back(State(vec[p].val + 1));
  for (; p && vec[p].go[w] == 0; p = vec[p].par)
    vec[p].go[w] = np;
  if (p == 0) {
    vec[np].par = root;
  } else {
    if (vec[vec[p].go[w]].val == vec[p].val + 1) {
      vec[np].par = vec[p].go[w];
    } else {
      int q = vec[p].go[w], r = vec.size();
      vec.push_back(vec[q]);
      vec[r].val = vec[p].val + 1;
      vec[q].par = vec[np].par = r;
      for (; p && vec[p].go[w] == q; p = vec[p].par)
        vec[p].go[w] = r;
    }
  }
  tail = np;
}
```

8 Problems

8.1 Mo's Algorithm on Tree

```
constexpr int MAX_N = 50000 + 10;
constexpr int MAX_M = 50000 + 10;
constexpr int MAX_LOG_N = 17 + 1;
constexpr int MAX_SQRT_N = 1000 + 10;
int bk[MAX_N];
struct Query {
  int id, x, y, t;
  Query() {}
  Query(int id0, int x0, int y0, int t0): id(id0),
      x(x0), y(y0), t(t0) {}
  bool operator < (const Query &rhs) const {
    return bk[x] != bk[rhs.x] ? bk[x] < bk[rhs.x] :
        bk[y] != bk[rhs.y] ? bk[y] < bk[rhs.y] : t <
        rhs.t;
  }
};
struct Modify {
  int p, v, pre_v;
  Modify() {}
  Modify(int p0, int v0, int pre_v0): p(p0), v(v0),
      pre_v(pre_v0) {}
};
stack<int> stk;
vector<Query> qs;
vector<Modify> ms;
vector<int> G[MAX_N];
int depth[MAX_N], dfn[MAX_N], dfs_clock, N, bk_cnt,
    pre_v[MAX_N], A[MAX_N], cnt[MAX_N], ans[MAX_M],
    state[MAX_N], anc[MAX_N][MAX_LOG_N], bk_sz,
    bk_sz2, bk2[MAX_SQRT_N];
int dfs(int u) {
  int sz = 0;
  dfn[u] = ++dfs_clock;
  for (int i = 1; i < MAX_LOG_N; i++) {
    if (depth[u] < (1 << i))
      break;
    anc[u][i] = anc[anc[u][i-1]][i-1];
  }
  REP(i, SZ(G[u])) {
    int v = G[u][i];
    if (v == anc[u][0]) continue;
    depth[v] = depth[u] + 1;
    anc[v][0] = u;
    sz += dfs(v);
    if (sz >= bk_sz) {
      bk_cnt++;
      while (sz--) {
        int x = stk.top();
        stk.pop();
        bk[x] = bk_cnt;
      }
      sz = 0;
    }
  }
  stk.push(u);
  return sz + 1;
}
int lca(int u, int v) {
  if (depth[u] < depth[v]) swap(u, v);
  int t = depth[u] - depth[v];
  for (int i = MAX_LOG_N - 1; i >= 0; i--) {
    if (t&(1<<i)) u = anc[u][i];
  }
  for (int i = MAX_LOG_N - 1; i >= 0; i--) {
    if (anc[u][i] != anc[v][i]) {
      u = anc[u][i];
      v = anc[v][i];
    }
  }
  return (u == v) ? u : anc[u][0];
}
void reverse(int u) {
  if (state[u]) {
    if (A[u] <= N && !--cnt[A[u]])
      bk2[A[u] / bk_sz2]--;
  } else {
    if (A[u] <= N && !cnt[A[u]]++)
      bk2[A[u] / bk_sz2]++;
  }
  state[u] ^= 1;
}
void change(int u, int x) {
  if (state[u]) {
    reverse(u);
    A[u] = x;
    reverse(u);
  } else
    A[u] = x;
```

```
}
void solve(int u, int v) {
  while (u != v) {
    if (depth[u] > depth[v]) {
      reverse(u);
      u = anc[u][0];
    } else {
      reverse(v);
      v = anc[v][0];
    }
  }
}
int get_mex() {
  for (int i = 0; ; i++)
    if (bk2[i] != bk_sz2)
      for (int j = 0; j < bk_sz2; j++)
        if (!cnt[bk_sz2 * i + j])
          return bk_sz2 * i + j;
}
int main() {
  int M;
  scanf("%d%d", &N, &M);
  bk_sz = (int) pow((double) N, 2.0 / 3.0);
  bk_sz2 = (int) sqrt((double) N);
  REP1 (i, 1, N) {
    scanf("%d",&A[i]);
    pre_v[i] = A[i];
  }
  REP (i, N - 1) {
    int x, y;
    scanf("%d%d", &x, &y);
    G[x].PB(y);
    G[y].PB(x);
  }
  dfs(1);
  while (!stk.empty()) {
    int x = stk.top();
    stk.pop();
    bk[x] = bk_cnt;
  }
  for (int i = 0; i < M; i++) {
    int op, a, b;
    scanf("%d%d%d", &op, &a, &b);
    if (op == 0) {
      ms.PB(Modify(a, b, pre_v[a]));
      pre_v[a] = b;
    } else {
      if (dfn[a] > dfn[b]) swap(a, b);
      qs.PB(Query(SZ(qs), a, b, SZ(ms) - 1));
    }
  }
  if (qs.empty()) return 0;
  sort(ALL(qs));
  for (int i = 0; i <= qs[0].t; i++)
    change(ms[i].p, ms[i].v);
  solve(qs[0].x, qs[0].y);
  int t = lca(qs[0].x, qs[0].y);
  reverse(t);
  ans[qs[0].id] = get_mex();
  reverse(t);
  for (int i = 1; i < (int) qs.size(); i++) {
    for (int j = qs[i-1].t + 1; j <= qs[i].t; j++)
      change(ms[j].p, ms[j].v);
    for (int j = qs[i-1].t; j > qs[i].t; j--)
      change(ms[j].p, ms[j].pre_v);
    solve(qs[i-1].x, qs[i].x);
    solve(qs[i-1].y, qs[i].y);
    t = lca(qs[i].x, qs[i].y);
    reverse(t);
    ans[qs[i].id] = get_mex();
    reverse(t);
  }
  REP (i, SZ(qs)) printf("%d\n", ans[i]);
  return 0;
}
```

8.2 Manhattan MST

```
#include<bits/stdc++.h>
#define REP(i,n) for(int i=0;i<n;i++)
using namespace std;
typedef long long LL;
const int N=200100;
int n,m;
struct PT {int x,y,z,w,id;}p[N];
inline int dis(const PT &a,const PT &b){return
    abs(a.x-b.x)+abs(a.y-b.y);}
inline bool cpx(const PT &a,const PT &b){return
    a.x!=b.x? a.x>b.x:a.y>b.y;}
inline bool cpz(const PT &a,const PT &b){return
    a.z<b.z;}
struct E{int a,b,c;}e[8*N];
bool operator<(const E&a,const E&b){return a.c<b.c;}
struct Node{
  int L,R,key;
}node[4*N];
int s[N];
int F(int x){return s[x]==x?x:s[x]=F(s[x]);}
void U(int a,int b){s[F(b)]=F(a);}
void init(int id,int L,int R) {
  node[id]=(Node){L,R,-1};
  if(L==R)return;
  init(id*2,L,(L+R)/2);
  init(id*2+1,(L+R)/2+1,R);
}
void ins(int id,int x) {
  if(node[id].key==-1 ||
      p[node[id].key].w>p[x].w)node[id].key=x;
  if(node[id].L==node[id].R)return;
  if(p[x].z<=(node[id].L+node[id].R)/2)ins(id*2,x);
  else ins(id*2+1,x);
}
int Q(int id,int L,int R){
  if(R<node[id].L || L>node[id].R)return -1;
  if(L<=node[id].L && node[id].R<=R)return
      node[id].key;
  int a=Q(id*2,L,R),b=Q(id*2+1,L,R);
  if(b==-1 || (a!=-1 && p[a].w<p[b].w)) return a;
  else return b;
}
void calc() {
  REP(i,n) {
    p[i].z=p[i].y-p[i].x;
    p[i].w=p[i].x+p[i].y;
  }
  sort(p,p+n,cpz);
  int cnt=0,j,k;
  for(int i=0;i<n;i=j){
    for(j=i+1;p[j].z==p[i].z && j<n;j++);
    for(k=i,cnt++;k<j;k++)p[k].z=cnt;
  }
  init(1,1,cnt);
  sort(p,p+n,cpx);
  REP(i,n) {
    j=Q(1,p[i].z,cnt);
    if (j != -1)
      e[m++] = (E){p[i].id, p[j].id, dis(p[i],
          p[j])};
    ins(1,i);
  }
}
LL MST() {
  LL r=0;
  sort(e,e+m);
  REP(i,m) {
    if(F(e[i].a)==F(e[i].b))continue;
    U(e[i].a,e[i].b);
    r+=e[i].c;
  }
  return r;
}
int main(){
  int ts;
```

```
  scanf("%d", &ts);
  while (ts--) {
    m = 0;
    scanf("%d",&n);
    REP(i,n) {
      scanf("%d%d",&p[i].x,&p[i].y);
      p[i].id=s[i]=i;
    }
    calc();
    REP(i,n)p[i].y= -p[i].y;
    calc();
    REP(i,n)swap(p[i].x,p[i].y);
    calc();
    REP(i,n)p[i].x=-p[i].x;
    calc();
    printf("%lld\n",MST()*2);
  }
  return 0;
}
```

9 Miscellany

9.1 tabi no hidarite saihate no migite

tabi no hidarite saihate no migite
旅の左手、最果ての右手

sora ni ukannderu hikaru nami o
空に浮かんでる光る波を
tabanete niji no hasi wo kakeyou
束ねて虹の橋をかけよう
ayaui asiba suberu suro-pu
危うい足場　滑るスロープ
kako to mirai no mitisirube
過去と未来の道標

kimi no hidarite boku no migite wo
君の左手　僕の右手を
tunaide tunagete hazimeyou
繋いで繋げてはじめよう
itumo soba ni iru yo
いつも　そばに　いるよ
kako mo ima mo mirai mo
過去も　今も　未来も
dakara mayowazu ni aruite ikou
だから迷わずに歩いていこう

irotoridori no yume kasanete
色とりどりの夢　重ねて
asita ha doko ni mukau
明日はどこに向かう
kimi to issyo ni iretara sekai ha kagayakidasu yo
君と一緒にいれたら　世界は輝きだすよ
saihate no ti ni saku hana wo sagasou
最果ての地に咲く花を探そう

9.2 Made in Abyss