Contents

1 Basic

1.1 .vimrc

```
"This file should be placed at ~/.vimrc"
se nu ai hls et ru ic is sc cul
se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
syntax on
hi cursorline cterm=none ctermbg=89
set bg=dark
inoremap {<CR> {<CR>}<Esc>ko<tab>
```

1.2 Default Bear

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define int ll
typedef pair<int,int> pii;
#define X first
#define Y second
#define pb push_back
#define All(a) a.begin(), a.end()
#define SZ(a) ((int)a.size())
#define endl '\n'
```

1.3 Default Ken

```
#include <bits/stdc++.h>
#define F first
#define S second
#define pb push_back
#define pob pop_back
#define SZ(x) (int)(x.size())
#define all(x) begin(x), end(x)
#ifdef LOCAL
#define HEHE freopen("in.txt", "r", stdin);
#define debug(...)
     {cout << #__VA_ARGS__ << " = "; dbg(__VA_ARGS__);}
#else
#define HEHE ios_base::sync_with_stdio(0), cin.tie(0);
#define debug(...) 7122;
#endif
using namespace std;

#define chmax(a, b) (a) = (a) > (b) ? (a) : (b)
#define chmin(a, b) (a) = (a) < (b) ? (a) : (b)

#define FOR(i, a, b) for (int i = (a); i <= (b); i++)
void dbg() { cerr << '\n'; }
template<typename T, typename ...U>
void dbg(T t, U ...u) { cerr << t << ' '; dbg(u...); }

#define int long long

signed main() {
  HEHE
}
```

1.4 IO Optimize

```
bool rit(auto& x) {
  x = 0; char c = cin.rdbuf()->sbumpc(); bool neg = 0;
  while (!isdigit(c)) {
    if (c == EOF) return 0;
    if (c == '-') neg = 1;
    c = cin.rdbuf()->sbumpc();
  }
  while (isdigit(c))
    x = x * 10 + c - '0', c = cin.rdbuf()->sbumpc();
  return x = neg ? -x : x, 1;
}
void wit(auto x) {
  if (x < 0) cout.rdbuf()->sputc('-'), x = -x;
  char s[20], len = 0;
  do s[len++] = x % 10 + '0'; while (x /= 10);
  while (len) cout.rdbuf()->sputc(s[--len]);
}
```

1.5 PBDS

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
// #include <bits/extc++.h>
#include <bits/stdc++.h>
using namespace __gnu_pbds;
using namespace std;
template <typename T>
```

```
using rbtree = tree<T, null_type, less<T
    >, rb_tree_tag, tree_order_statistics_node_update>;
// less<T> : increasing, greater<T> : decreasing
// rb_tree_tag, splay_tree_tag, ov_tree_tag

int main() {
  int x;
  rbtree<int> t, rhs, rhs2;
  t.insert(x);
  t.erase(x); // return 1 or 0
  cout << t.order_of_key(x) << '\n'; // rank
  cout << *t.find_by_order(x) << '\n'; // x-th
  cout << *t.lower_bound(x) << '\n'; // iterator >= x
  cout << *t.upper_bound(x) << '\n'; // iterator > x
  t.join(rhs
      ); // merge // same type, no duplicate elements
  t.split(x, rhs2
      ); // tree : elements <= x, rhs : elements > x
}
```

1.6 Set Comperator

```
auto cmp = [](int a, int b) {
        return a > b;
};
set<int, decltype(cmp)> s = {1, 2, 3, 4, 5};
cout << *s.begin() << '\n';
```

1.7 Random

```
#include <random>
#include <chrono>
#include <algorithm>
mt19937 rng(chrono
    ::system_clock::now().time_since_epoch().count());
int randint(int lb, int ub) {
  return uniform_int_distribution<int>(lb, ub)(rng);
}
// shuffle(v.begin(), v.end(), rng);
```

1.8 Python

```
from decimal import *
setcontext(Context(prec
    =MAX_PREC, Emax=MAX_EMAX, rounding=ROUND_FLOOR))
print(Decimal(input()) * Decimal(input()))
from fractions import Fraction
Fraction
    ('3.14159').limit_denominator(10).numerator # 22
```

2 Graph

2.1 2 SAT

```
struct TwoSAT {
  int n;
  Scc scc;
  void init(int _n) {
    // (0,1),(2,3),...
    n = _n; scc.init(n * 2);
  }
  void add_disjunction(int a, int na, int b, int nb) {
    a = 2 * a ^ na, b = 2 * b ^ nb;
    scc.addEdge(a ^ 1, b);
    scc.addEdge(b ^ 1, a);
  }
  vector<int> solve() {
    scc.solve();
    vector<int> assignment(n, 0);
    for (int i = 0; i < n; i++) {
      if (scc
          .bln[2 * i] == scc.bln[2 * i ^ 1]) return {};
      assignment
          [i] = scc.bln[2 * i] > scc.bln[2 * i ^ 1];
    }
    return assignment;
  }
};
```

2.2 Bellman Ford

```
struct edge{
  int u, v;
  int cost;
};
vector<int> d(n, inf);
bool bellman_ford(vector<edge> &ee, int n, int s){
  d[s] = 0;
  auto relax = [&](edge e){
    if(d[e.v] > d[e.u] + e.cost){
      d[e.v] = d[e.u] + e.cost;
      return 1;
    }
    return 0;
  }
  for(int t = 1; t <= n; ++t){
    bool update = 0;
    for(auto &e: ee)
      update |= relax(e);
    if(t == n && update) return 0;
  }
  return 1;
}
```

2.3 Biconnected Component

```
// beware of multiple inputs
#define ep emplace
#define eb emplace_back
const int N = 2e5 + 5;

int d[N], low[N];
vector<int> g[N];
vector<vector<int>> bcc;
stack<int> st;

void dfs(int x, int p) {
  d[x] = ~p ? d[p] + 1 : 1, low[x] = d[x];
  st.ep(x);
  for (const auto& i : g[x]) {
    if (i == p) continue;
    if (!d[i]) {
      dfs(i, x);
      low[x] = min(low[x], low[i]);
      if (d[x] <= low[i]) {
        int tmp;
        bcc.eb();
        do tmp = st.top(), st.pop
            (), bcc.back().eb(tmp); while (tmp != x);
        st.ep(x);
      }
    }
    low[x] = min(low[x], d[i]);
  }
}
```

2.4 Bridge Connected Component

```
#define ep emplace
constexpr int N = 2e5 + 1;

int d[N], low[N], bcc[N], nbcc;
vector<int> g[N];
stack<int> st;

void dfs(int x, int p) {
  d[x] = ~p ? d[p] + 1 : 1, low[x] = d[x];
  st.ep(x);
  for (const auto& i : g[x]) {
    if (i == p) continue;
    if (!d[i]) {
      dfs(i, x);
      low[x] = min(low[x], low[i]);
    }
    low[x] = min(low[x], d[i]);
  }
  if (low[x] == d[x]) {
    nbcc++;
    int tmp;
    do tmp = st.top()
        , st.pop(), bcc[tmp] = nbcc; while (tmp != x);
  }
}
```

2.5 Bridge

```
#define eb emplace_back
using pii = pair<int, int>;
const int N = 2e5 + 5;

int d[N], low[N];
vector<int> g[N];
vector<int> ap; // articulation point
vector<pii> bridge;
```

```
void dfs(int x, int p) {
  d[x] = ~p ? d[p] + 1 : 1, low[x] = d[x];
  int cnt = 0;
  bool isap = 0;
  for (const auto& i : g[x]) {
    if (i == p) continue;
    if (!d[i]) {
      dfs(i, x), cnt++;
      if (d[x] <= low[i]) isap = 1;
      if (d[x] < low[i]) bridge.eb(x, i);
      low[x] = min(low[x], low[i]);
    }
    low[x] = min(low[x], d[i]);
  }
  if (p == -1 && cnt < 2) isap = 0;
  if (isap) ap.eb(x);
}
```

2.6 C3C4

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

signed main() {
  cin.tie(0)->sync_with_stdio(0);

  int N, M;
  cin >> N >> M;

  vector<int> deg(N);
  vector<array<int, 2>> e(M);
  for (auto &[u, v] : e) {
    cin >> u >> v;
    --u, --v;
    ++deg[u], ++deg[v];
  }

  vector<int> ord(N), rk(N);
  iota(all(ord), 0);
  sort(all(ord)
      , [&](int x, int y) { return deg[x] > deg[y]; });
  for (int i = 0; i < N; i++) rk[ord[i]] = i;

  vector<vector<int>> D(N), adj(N);
  for (auto [u, v] : e) {
    if (rk[u] > rk[v]) swap(u, v);
    D[u].emplace_back(v);
    adj[u].emplace_back(v);
    adj[v].emplace_back(u);
  }

  vector<int> vis(N);

  int64_t c3 = 0, c4 = 0;
  // ord = sort by deg decreasing, rk[ord[i]] = i
  // D[i] = edge point from rk small to rk big
  for (int x : ord) { // c3
    for (int y : D[x]) vis[y] = 1;
    for (int y : D[x]) for (int z : D[y]) c3 += vis[z];
    for (int y : D[x]) vis[y] = 0;
  }
  for (int x : ord) { // c4
    for (int y : D[x]) for (int z : adj[y])
      if (rk[z] > rk[x]) c4 += vis[z]++;
    for (int y : D[x]) for (int z : adj[y])
      if (rk[z] > rk[x]) --vis[z];
  } //
      both are O(M*sqrt(M)), test @ 2022 CCPC guangzhou

  cout << c4 * 8 << '\n';
}
```

2.7 Centroid Decomposition

```
const int MAXN = 1e5 + 5;
int n, q, vis[MAXN], sz[MAXN];
vector<int> adj[MAXN], pa[MAXN], mx[MAXN], dis[MAXN];

void dfs_sz(int x, int p) {
    sz[x] = 1;
    for (int i : adj[x]) {
        if (i == p or vis[i]) continue;
        dfs_sz(i, x);
        sz[x] += sz[i];
    }
}
int cen;
void dfs_cen(int x, int p, int all) {
    int tmp = all - sz[x];
    for (int i : adj[x]) {
        if (i == p or vis[i]) continue;
        dfs_cen(i, x, all);
        chmax(tmp, sz[i]);
    }
    if (tmp * 2 <= all) cen = x;
}
void dfs(int x, int p, int d) {
    pa[x].pb(cen);
    dis[x].pb(d);
    if (d >= mx[cen].size()) mx[cen].pb(x);
    else chmax(mx[cen][d], x);
    for (int i : adj[x]) {
        if (i == p or vis[i]) continue;
        dfs(i, x, d + 1);
    }
}
void deco(int x, int d) {
    dfs_sz(x, x);
    dfs_cen(x, x, sz[x]);
    vis[cen] = 1;
    dfs(cen, cen, 0);
    for (int i = 1; i < mx[cen].size(); i++) {
        chmax(mx[cen][i], mx[cen][i - 1]);
    }
    for (int i : adj[cen]) {
        if (vis[i]) continue;
        deco(i, d + 1);
    }
}
int get(int x, int k) {
    if (!mx[x].size() or k < 0) return 0;
    return k >= mx[x].size() ? mx[x].back() : mx[x][k];
}
int query(int x, int k) {
    int res = get(x, k);
    for (int i = 0; i < pa[x].size(); i++) {
        int p = pa[x][i];
        int d = dis[x][i];
        chmax(res, get(p, k - d));
    }
    return res;
}

signed main() {
    WOSHAOJI
    cin >> n >> q;
    for (int i = 1, u, v; i < n; i++) {
        cin >> u >> v;
        adj[u].pb(v);
        adj[v].pb(u);
    }
    deco(1, 0);
    while (q--) {
        int x, k; cin >> x >> k;
        cout << query(x, k) << '\n';
    }
}
```

2.8 Close Vertices

```
#include <iostream>
#include <vector>
#include <bitset>
#include <algorithm>
#include <cstring>
using namespace std;
int l, w;
vector<pair<int, short>> tree[100000];
bitset<100000> removed;
int current_centroid, BIT[100000];
// Return subtree size internally
// and
     place the discovered centroid in current_centroid
int find_centroid
    (const int n, const int u, const int p = -1) {
  if (n == 1) { current_centroid = u; return 0; }
  int subtree_sum = 0;
  for (const auto
        &[v, w] : tree[u]) if (v != p && !removed[v]) {
      subtree_sum += find_centroid(n, v, u);
      if (current_centroid > -1) return 0;
      if (subtree_sum >=
            n >> 1) { current_centroid = u; return 0; }
    }
```

```
  return subtree_sum + 1;
}
void DFS(const int u, const int p, const int length,
    const int weight, vector<pair<int, int>> &record) {
  record.emplace_back(weight, length);
  for (const auto
       &[v, w] : tree[u]) if (v != p && !removed[v])
      DFS(v, u, length + 1, weight + w, record);
}
bool greater_size(const vector<pair
    <int, int>> &v, const vector<pair<int, int>> &w) {
  return v.size() > w.size();
}
long long centroid_decomposition(const int n, int u) {
  long long ans = 0;
  // Step 1: find the centroid
  current_centroid = -1; find_centroid(n, u);
  removed[u = current_centroid] = true;
  // Step 2: DFS from the centroid (again)
  // and continue the centroid decomposition
  vector<vector<pair<int, int>>> root2subtree_paths;
  for (const auto &[v, w] : tree[u]) if (!removed[v]) {
      root2subtree_paths.emplace_back();
      DFS(v, u, 1, w, root2subtree_paths.back());
      // Sort mainly according to weight
      ranges::sort(root2subtree_paths.back());
      ans += centroid_decomposition
          (root2subtree_paths.back().size(), v);
    }
  for (const auto &v : root2subtree_paths)
    for (const auto &[weight, length] : v)
      if (length <= l && weight <= w) ++ans;
  // Step 3: optimal merging
  ranges::make_heap(root2subtree_paths, greater_size);
  while (root2subtree_paths.size() > 1) {
    ranges::pop_heap(root2subtree_paths, greater_size);
    // Merge
         front() (with maybe larger size) and back()
    // Count cross-centroid paths
    memset(BIT, 0, root2subtree_paths
        .back().size() * sizeof(int));
    auto p = root2subtree_paths.front().crbegin();
    for (auto q = root2subtree_paths.back().cbegin()
        ; q != root2subtree_paths.back().cend(); ++q) {
      int L;
      while (p != root2subtree_paths.front().crend()
          && p->first + q->first > w) {
        L = min(l - p->second,
                static_cast<int>(
                    root2subtree_paths.back().size()));
        while
            (L > 0) { ans += BIT[L - 1]; L -= L & -L; }
        ++p;
      }
      L = q->second;
      while (L <= static_cast
          <int>(root2subtree_paths.back().size()))
      {
        ++BIT[L - 1]; L += L & -L;
      }
    }
    while (p != root2subtree_paths.front().crend()) {
      int L = min(l - p++->second, static_cast
          <int>(root2subtree_paths.back().size()));
      while (L > 0) { ans += BIT[L - 1]; L -= L & -L; }
    }
    // Actually merge the lists
    vector<pair<int, int>> buffer;
    buffer.reserve(root2subtree_paths.front
        ().size() + root2subtree_paths.back().size());
    ranges::merge
        (root2subtree_paths.front(), root2subtree_paths
        .back(), back_inserter(buffer));

    root2subtree_paths.pop_back();
    ranges::pop_heap(root2subtree_paths, greater_size);
    root2subtree_paths.back() = move(buffer);
    ranges
        ::push_heap(root2subtree_paths, greater_size);
  }
  return ans;
}
int main() {
  ios_base::sync_with_stdio(false);
  int n; cin >> n >> l >> w;
  for (int i = 1; i < n; ++i) {
    int p; short w; cin >> p >> w;
    tree[--p].emplace_back(i, w);
    tree[i].emplace_back(p, w);
  }
  cout << centroid_decomposition(n, 0) << endl;
}
```

2.9 Disjoint Set

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

struct disjoint_set {
  static const int maxn = (int)5e5 + 5;
  int n, fa[maxn], sz[maxn];
  vector<pair<int*, int>> h;
  vector<int> sp;
  void init(int _n) {
    n = _n;
    for (int i = 0 ; i < n ; ++i)
      fa[i] = i, sz[i] = 1;
    sp.clear(); h.clear();
  }
  void assign(int *k, int v) {
    h.push_back({k, *k});
    *k = v;
  }
  void save() { sp.push_back((int)h.size()); }
  void undo() {
    assert(!sp.empty());
    int last = sp.back(), cnt = 0; sp.pop_back();
    while (h.size() > last) {
      auto x = h.back(); h.pop_back();
      *x.first = x.second;
      cnt++;
    }
    n += cnt / 2;
  }
  int f(int x) {
    while (fa[x] != x) x = fa[x];
    return x;
  }
  bool merge(int x, int y) {
    x = f(x); y = f(y);
    if (x == y) return 0;
    if (sz[x] < sz[y]) swap(x, y);
    assign(&sz[x], sz[x] + sz[y]);
    assign(&fa[y], x);
    n--;
    return 1;
  }
} djs;
```

2.10 Heavy Light Decomposition

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5 + 5;
#define eb emplace_back
int t, n, q, seg[N << 1]; // t := time-stamp
int sz[N], fa[N], dep[N], to[N], fr[N], dfn[N], arr[N];
// size, father, depth
    , to-heavy-child, from-head, dfs-order, a_i value
vector<int> g[N];
void upd(int x, int v) {
  for (seg[x += n] = v; x > 1; x >>= 1)
    seg[x >> 1] = max(seg[x], seg[x ^ 1]);
}
int qry(int l, int r) { // [l, r]
  int ret = -1e9; // -max
  for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
    if (l & 1) ret = max(ret, seg[l++]);
    if (r & 1) ret = max(ret, seg[--r]);
  }
  return ret;
}
void dfs(int x, int p) {
  sz[x] = 1, fa[
      x] = p, to[x] = -1, dep[x] = ~p ? dep[p] + 1 : 0;
  for (auto i : g[x])
    if (i != p) {
      dfs(i, x);
      if (to[x] == -1 || sz[i] > sz[to[x]]) to[x] = i;
      sz[x] += sz[i];
    }
}
void dfs2(int x, int f) {
```

```
  fr[x] = f, dfn[x] = ++t, upd(dfn[x], arr[x]);
  if (to[x] != -1) dfs2(to[x], f);
  for (auto i : g[x])
    if (i != fa[x] && i != to[x]) dfs2(i, i);
}
int qry2(int u, int v) { // query on tree
  int fu = fr[u], fv = fr[v], ret = -1e9;
  while (fu != fv) {
    if (dep[fu] < dep[fv]) swap(fu, fv), swap(u, v);
    ret = max(ret, qry(dfn
        [fu], dfn[u])); // interval: [dfn[fu], dfn[u]]
    u = fa[fu], fu = fr[u];
  }
  if (dep[u] > dep[v]) swap(u, v);
  // u is the LCA
  ret = max(ret, qry(dfn[u], dfn[v]));
  return ret;
}
int main() {
  ios::sync_with_stdio(false), cin.tie(nullptr);
  cin >> n >> q;
  for (int i = 1; i <= n; i++) cin >> arr[i];
  for (int i = 1, a, b; i < n; i++)
    cin >> a >> b, g[a].eb(b), g[b].eb(a);
  dfs(1, -1), dfs2(1, 1);
  while (q--) {
    int op; cin >> op;
    if (op == 1) {
      int x,
          v; cin >> x >> v, arr[x] = v, upd(dfn[x], v);
    }
    else {
      int a, b; cin >> a >> b;
      cout << qry2(a, b) << '\n';
    }
  }
}
```

2.11 KSP

```
// from CRyptoGRapheR
// time: O(|E| \lg |E|+|V| \lg |V|+K)
// memory: O(|E| \lg |E|+|V|)
struct KSP { // 1-base
  struct nd {
    int u, v; ll d;
    nd(int ui = 0, int vi
        = 0, ll di = INF) { u = ui; v = vi; d = di; }
  };
  struct heap { nd* edge; int dep; heap* chd[4]; };
  static int cmp(heap
      * a, heap* b) { return a->edge->d > b->edge->d; }
  struct node {
    int v; ll d; heap* H; nd* E;
    node() {}
    node(ll
        _d, int _v, nd* _E) { d = _d; v = _v; E = _E; }
    node(heap* _H, ll _d) { H = _H; d = _d; }
    friend bool operator<(node a, node b)
    { return a.d > b.d; }
  };
  int n, k, s, t, dst[N]; nd *nxt[N];
  vector<nd*> g[N], rg[N]; heap *nullNd, *head[N];
  void init(int _n, int _k, int _s, int _t) {
    n = _n; k = _k; s = _s; t = _t;
    for (int i = 1; i <= n; i++) {
      g[i].clear(); rg[i].clear();
      nxt[i] = NULL; head[i] = NULL; dst[i] = -1;
    }
  }
  void addEdge(int ui, int vi, ll di) {
    nd* e = new nd(ui, vi, di);
    g[ui].push_back(e); rg[vi].push_back(e);
  }
  queue<int> dfsQ;
  void dijkstra() {
    while (dfsQ.size()) dfsQ.pop();
    priority_queue<node> Q; Q.push(node(0, t, NULL));
    while (!Q.empty()) {
      node p = Q
          .top(); Q.pop(); if (dst[p.v] != -1)continue;
      dst[p.v] = p.d; nxt[p.v] = p.E; dfsQ.push(p.v);
      for (auto e
          : rg[p.v]) Q.push(node(p.d + e->d, e->u, e));
    }
  }
  heap* merge(heap* curNd, heap* newNd) {
    if (curNd == nullNd) return newNd;
    heap* root
        = new heap; memcpy(root, curNd, sizeof(heap));
    if (newNd->edge->d < curNd->edge->d) {
      root->edge = newNd->edge;
      root->chd[2] = newNd->chd[2];
      root->chd[3] = newNd->chd[3];
      newNd->edge = curNd->edge;
      newNd->chd[2] = curNd->chd[2];
      newNd->chd[3] = curNd->chd[3];
    }
    if (root->chd[0]->dep < root->chd[1]->dep)
      root->chd[0] = merge(root->chd[0], newNd);
    else root->chd[1] = merge(root->chd[1], newNd);
    root->dep = max(root->chd[0]->dep,
                    root->chd[1]->dep) + 1;
    return root;
  }
  vector<heap*> V;
  void build() {
    nullNd = new
        heap; nullNd->dep = 0; nullNd->edge = new nd;
    fill(nullNd->chd, nullNd->chd + 4, nullNd);
    while (not dfsQ.empty()) {
      int u = dfsQ.front(); dfsQ.pop();
      if (!nxt[u]) head[u] = nullNd;
      else head[u] = head[nxt[u]->v];
      V.clear();
      for (auto && e : g[u]) {
        int v = e->v;
        if (dst[v] == -1) continue;
        e->d += dst[v] - dst[u];
        if (nxt[u] != e) {
          heap* p = new
              heap; fill(p->chd, p->chd + 4, nullNd);
          p->dep = 1; p->edge = e; V.push_back(p);
        }
      }
      if (V.empty()) continue;
      make_heap(V.begin(), V.end(), cmp);
#define L(X) ((X<<1)+1)
#define R(X) ((X<<1)+2)
      for (size_t i = 0; i < V.size(); i++) {
        if (L(i) < V.size()) V[i]->chd[2] = V[L(i)];
        else V[i]->chd[2] = nullNd;
        if (R(i) < V.size()) V[i]->chd[3] = V[R(i)];
        else V[i]->chd[3] = nullNd;
      }
      head[u] = merge(head[u], V.front());
    }
  }
  vector<ll> ans;
  void first_K() {
    ans.clear(); priority_queue<node> Q;
    if (dst[s] == -1) return;
    ans.push_back(dst[s]);
    if (head[s] != nullNd)
      Q.push(node(head[s], dst[s] + head[s]->edge->d));
    for (int _ = 1; _ < k and not Q.empty(); _++) {
      node p = Q.top(), q; Q.pop(); ans.push_back(p.d);
      if (head[p.H->edge->v] != nullNd) {
        q.H = head
            [p.H->edge->v]; q.d = p.d + q.H->edge->d;
        Q.push(q);
      }
      for (int i = 0; i < 4; i++)
        if (p.H->chd[i] != nullNd) {
          q.H = p.H->chd[i];
          q.d = p
              .d - p.H->edge->d + p.H->chd[i]->edge->d;
          Q.push(q);
        }
    }
  }
  void
      solve() { // ans[i] stores the i-th shortest path
    dijkstra(); build();
    first_K(); // ans.size() might less than k
  }
} solver;
```

2.12 LCA

```
#define eb emplace_back
const int N = 2e5 + 5, logN = __lg(N) + 1, inf = 1e9;
int n, q, logn;
int dep[N], fa[N][logN];
```

```
vector<int> g[N];

void dfs(int x, int p) {
  dep[x] = ~p ? dep[p] + 1 : 0;
  fa[x][0] = p;
  for (int i = 1; (1 << i) <= dep[x]; i++)
    fa[x][i] = fa[fa[x][i - 1]][i - 1];
  for (const auto& u : g[x])
    if (u != p) dfs(u, x);
}

int LCA(int u, int v) {
  if (dep[u] > dep[v]) swap(u, v);
  for (int i = 0; i < logn; i++)
    if ((dep[v] - dep[u]) >> i & 1) v = fa[v][i];
  if (u == v) return u;
  for (int i = logn - 1; i >= 0; i--)
    if (fa[u][i] != fa[v][i])
      u = fa[u][i], v = fa[v][i];
  return fa[u][0];
}
// logn = __lg(n) + 1
// g[a].eb(b)
// dfs(root, -1)
// query -> LCA(u, v)
// distance
     of (u, v) = dep[u] + dep[v] - 2 * dep[LCA(u, v)]
```

2.13 Maximum Clique

```
struct Maximum_Clique {
  typedef bitset<MAXN> bst;
  bst N[MAXN], empty;
  int p[MAXN], n, ans;
  void BronKerbosch2(bst R, bst P, bst X) {
    if (P == empty && X == empty)
      return ans = max(ans, (int)R.count()), void();
    bst tmp = P | X;
    int u;
    if ((R | P | X).count() <= ans) return;
    for (int uu = 0; uu < n; ++uu) {
      u = p[uu];
      if (tmp[u] == 1) break;
    }
    // if (double(clock())/CLOCKS_PER_SEC > .999)
    // return;
    bst now2 = P & ~N[u];
    for (int vv = 0; vv < n; ++vv) {
      int v = p[vv];
      if (now2[v] == 1) {
        R[v] = 1;
        BronKerbosch2(R, P & N[v], X & N[v]);
        R[v] = 0, P[v] = 0, X[v] = 1;
      }
    }
  }
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) N[i].reset();
  }
  void add_edge(int u, int v) {
    N[u][v] = N[v][u] = 1;
  }
  int solve() { // remember srand
    bst R, P, X;
    ans = 0, P.flip();
    for (int i = 0; i < n; ++i) p[i] = i;
    random_shuffle(p, p + n), BronKerbosch2(R, P, X);
    return ans;
  }
};
```

2.14 SCC Kosaraju

```
#define eb emplace_back
const int N = 2e5 + 5;
vector<int> g[N], rg[N], ord;
int scc[N];
bool v[N];
void rdfs(int x) {
  v[x] = 1;
  for (const auto& i : rg[x])
    if (!v[i]) rdfs(i);
  ord.eb(x);
}
void dfs(int x, int nscc) {
  scc[x] = nscc;
  for (const auto& i : g[x])
    if (scc[i] == -1) dfs(i, nscc);
}
void kosaraju(int n) {
  memset(v, 0, sizeof(v));
  memset(scc, -1, sizeof(scc));
  for (int i = 0; i < n; i++)
    if (!v[i]) rdfs(i);
  int nscc = 0;
  for (int i = n - 1; i >= 0; i--) {
    int x = ord[i];
    if (scc[x] == -1)
      dfs(x, nscc++);
  }
}
```

2.15 SCC Tarjan

```
#define ep emplace
const int N = 2e5 + 5;
int d[N], low[N], scc[N], ins[N], nscc;
vector<int> g[N];
stack<int> st;
void dfs(int x, int p) {
  d[x] = ~p ? d[p] + 1 : 1, low[x] = d[x];
  st.ep(x), ins[x] = 1;
  for (const auto& i : g[x]) {
    if (!d[i]) dfs(i, x), low[x] = min(low[x], low[i]);
    else if (ins[i]) low[x] = min(low[x], d[i]);
  }
  if (d[x] == low[x]) {
    nscc++;
    int tmp;
    do tmp = st.top(), st.pop(), scc
        [tmp] = nscc, ins[tmp] = 0; while (tmp != x);
  }
}
```

2.16 Tree Centroid

```
const int N = 2e5 + 5, inf = 1e9;

vector<int> g[N];
int n, sz[N], center, csize; // set csize = inf

void dfs(int x, int p) {
  int mxsub = 0;
  sz[x] = 1;
  for (const auto& i : g[x])
    if (i != p) dfs(i,
         x), sz[x] += sz[i], mxsub = max(mxsub, sz[i]);
  mxsub = max(mxsub, n - sz[x]);
  if (mxsub < csize) center = x, csize = mxsub;
}
```

2.17 Virtual Tree

```
vector<int> vG[N];
int top, st[N];

void insert(int u) {
  if (top == -1) return st[++top] = u, void();
  int p = LCA(st[top], u);
  if (p == st[top]) return st[++top] = u, void();
  while (top >= 1 && dep[st[top - 1]] >= dep[p])
    vG[st[top - 1]].pb(st[top]), --top;
  if (st[top] != p)
    vG[p].pb(st[top]), --top, st[++top] = p;
  st[++top] = u;
}

void reset(int u) {
  for (int i : vG[u]) reset(i);
  vG[u].clear();
}

void solve(vector<int> &v) {
  top = -1;
  sort(ALL(v),
    [&](int a, int b) { return dfn[a] < dfn[b]; });
  for (int i : v) insert(i);
  while (top > 0) vG[st[top - 1]].pb(st[top]), --top;
  // do something
  reset(v[0]);
}
```

3 Data Structure

3.1 2D BIT

```
const int N = 1000 + 5;
int a[N][N];

struct BIT { // 1-based
  ll bit[N][N];
  int n, m;
  void init(int _n, int _m) { // O(nm)
    n = _n, m = _m;
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= m; j++)
        bit[i][j] = a[i][j];
    for (int b = 1; b << 1 <= max(n, m); b <<= 1) {
      for (int i = b; i + b <= n; i += b << 1)
        for (int j = 1; j <= m; j++)
          bit[i + b][j] += bit[i][j];
      for (int i = 1; i <= n; i++)
        for (int j = b; j + b <= m; j += b << 1)
          bit[i][j + b] += bit[i][j];
    }
  }
  void upd(int x, int y, int v) {
    for (int i = x; i <= n; i += i & -i)
      for (int j = y; j <= m; j += j & -j)
        bit[i][j] += v;
  }
  ll qry(int x, int y) {
    ll ret = 0;
    for (int i = x; i; i -= i & -i)
      for (int j = y; j; j -= j & -j)
        ret += bit[i][j];
    return ret;
  }
  ll qry(int
      x1, int y1, int x2, int y2) { // closed-interval
    return qry(x2, y2) - qry(x1 -
        1, y2) - qry(x2, y1 - 1) + qry(x1 - 1, y1 - 1);
  }
} tree;
// tree.init(n, m)
```

3.2 2D Segment Tree

```
const int inf = 1e9;
#define lc(x) (x << 1)
#define rc(x) (x << 1 | 1)
int N, M; // N : row max, M : col max
struct seg {
  vector<int> st;
  void pull(int);
  void merge(const seg&, const seg&, int, int, int);
  void build(int, int, int);
  void upd(int, int, int, int, int);
  int qry(int, int, int, int, int);
  seg(int size): st(size << 2 | 1) {}
};
void seg::pull(int id) {
  st[id] = max(st[lc(id)], st[rc(id)]);
}
void seg::merge(const seg& a
    , const seg& b, int id = 1, int l = 1, int r = M) {
  st[id] = max(a.st[id], b.st[id]);
  if (l == r) return;
  int m = (l + r) >> 1;
  merge(a,
        b, lc(id), l, m), merge(a, b, rc(id), m + 1, r);
}
void seg::build(int id = 1, int l = 1, int r = M) {
  if (l == r) {cin >> st[id]; return;}
  int m = (l + r) >> 1;
  build(lc(id), l, m), build(rc(id), m + 1, r);
  pull(id);
}
void seg::upd
    (int x, int v, int id = 1, int l = 1, int r = M) {
  if (l == r) {st[id] = v; return;}
  int m = (l + r) >> 1;
  if (x <= m) upd(x, v, lc(id), l, m);
  else upd(x, v, rc(id), m + 1, r);
  pull(id);
}
int seg::qry(
    int ql, int qr, int id = 1, int l = 1, int r = M) {
  if (ql <= l && r <= qr) return st[id];
  int m = (l + r) >> 1, ret = -inf;
  if (ql
      <= m) ret = max(ret, qry(ql, qr, lc(id), l, m));
  if (qr >
      m) ret = max(ret, qry(ql, qr, rc(id), m + 1, r));
  return ret;
}

struct segseg {
  vector<seg> st;
  void pull(int, int);
  void build(int, int, int);
  void upd(int, int, int, int, int, int);
  int qry(int, int, int, int, int, int, int);
  segseg(int n, int m): st(n << 2 | 1, seg(m)) {}
};
void segseg::pull(int id, int x) {
  st[id].upd(x,
      max(st[lc(id)].qry(x, x), st[rc(id)].qry(x, x)));
}
void segseg::build(int id = 1, int l = 1, int r = N) {
  if (l == r) {st[id].build(); return;}
  int m = (l + r) >> 1;
  build(lc(id), l, m), build(rc(id), m + 1, r);
  st[id].merge(st[lc(id)], st[rc(id)]);
}
void segseg::upd(int y
    , int x, int v, int id = 1, int l = 1, int r = N) {
  if (l == r) {st[id].upd(x, v); return;}
  int m = (l + r) >> 1;
  if (y <= m) upd(y, x, v, lc(id), l, m);
  else upd(y, x, v, rc(id), m + 1, r);
  pull(id, x);
}
int segseg::qry(int y1, int y2,
    int x1, int x2, int id = 1, int l = 1, int r = N) {
  if (y1 <= l && r <= y2) return st[id].qry(x1, x2);
  int m = (l + r) >> 1, ret = -inf;
  if (y1 <= m) ret
      = max(ret, qry(y1, y2, x1, x2, lc(id), l, m));
  if (y2 > m) ret =
      max(ret, qry(y1, y2, x1, x2, rc(id), m + 1, r));
  return ret;
}
```

3.3 BIT

```
const int N = 2e5 + 5;
int n, a[N];

struct BIT { // 1-based
  ll bit1[N], bit2[N];
  ll sum(ll* bit, int x) {
    ll ret = 0;
    for (; x; x -= x & -x) ret += bit[x];
    return ret;
  }
  void upd(ll* bit, int x, ll v) {
    for (; x <= n; x += x & -x) bit[x] += v;
  }
  ll qry(int x) {
    return (x + 1) * sum(bit1, x) - sum(bit2, x);
  }
  ll qry(int l, int r) { // [l, r]
    return qry(r) - qry(l - 1);
  }
  void upd(int l, int r, ll v) { // [l, r]
    upd(bit1, l, v), upd(bit2, l, l * v);
    upd(bit1
        , r + 1, -v), upd(bit2, r + 1, (r + 1) * -v);
  }
  BIT() {
    fill_n(bit1, N, 0), fill_n(bit2, N, 0);
  }
  BIT(int* a) { // O(n) build
    fill_n(bit1, N, 0), fill_n(bit2, N, 0);
    for (int i = 1;
         i <= n; i++) bit1[i] = a[i] - a[i - (i & -i)];
    for (int i = n; i; i--) a[i] -= a[i - 1];
    for (int
        i = 1; i <= n; i++) a[i] = a[i - 1] + a[i] * i;
    for (int i = 1;
         i <= n; i++) bit2[i] = a[i] - a[i - (i & -i)];
  }
};
```

3.4 chtholly tree

```
// 存 {x, v} , 從 x 開始到下一個位置前都是v
map<int, int> s;
// [l, r)
void ins(int l, int r, int i) {
    auto it1 = s.find(l);
    auto it2 = s.find(r);
    for (auto it = it1; it != it2; it++) {

    }
    s.erase(it1, it2); // [it`, it2)
    s[l] = ;
}
void split(int pos) {
    auto it = s.lower_bound(pos);
    if (it == s.end() or it->F != pos) {
        s[pos] = prev(it)->S;
    }
}
```

3.5 LiChaoST

```
struct LiChao_min {
  struct line {
    LL m, c;
    line(LL _m = 0, LL _c = 0) {
      m = _m;
      c = _c;
    }
    LL eval(LL x) { return m * x + c; }
  };
  struct node {
    node *l, *r;
    line f;
    node(line v) {
      f = v;
      l = r = NULL;
    }
  };
  typedef node *pnode;
  pnode root;
  int sz;
#define mid ((l + r) >> 1)
  void insert(line &v, int l, int r, pnode &nd) {
    if (!nd) {
      nd = new node(v);
      return;
    }
    LL trl = nd->f.eval(l), trr = nd->f.eval(r);
    LL vl = v.eval(l), vr = v.eval(r);
    if (trl <= vl && trr <= vr) return;
    if (trl > vl && trr > vr) {
      nd->f = v;
      return;
    }
    if (trl > vl) swap(nd->f, v);
    if (nd->f.eval(mid) < v.eval(mid))
      insert(v, mid + 1, r, nd->r);
    else swap(nd->f, v), insert(v, l, mid, nd->l);
  }
  LL query(int x, int l, int r, pnode &nd) {
    if (!nd) return LLONG_MAX;
    if (l == r) return nd->f.eval(x);
    if (mid >= x)
      return min(
        nd->f.eval(x), query(x, l, mid, nd->l));
    return min(
      nd->f.eval(x), query(x, mid + 1, r, nd->r));
  }
  /* -sz <= query_x <= sz */
  void init(int _sz) {
    sz = _sz + 1;
    root = NULL;
  }
  void add_line(LL m, LL c) {
    line v(m, c);
    insert(v, -sz, sz, root);
  }
  LL query(LL x) { return query(x, -sz, sz, root); }
};
```

3.6 persistent

```
const int MAXN = 2e5 + 5;
int a[MAXN];
int sum[MAXN * 25], lc[MAXN * 25], rc[MAXN * 25];
int add_node() {
    static int now = 0;
    return ++now;
}
void pull(int x) {
    sum[x] = sum[lc[x]] + sum[rc[x]];
}
void init(int &x, int lx, int rx) {
    if (!x) x = add_node();
    if (lx + 1 == rx) return;
    int mid = (lx + rx) / 2;
    init(lc[x], lx, mid);
    init(rc[x], mid, rx);
}
void update(int fa, int &x, int lx, int rx, int i) {
    if (!x) x = add_node();
    if (lx + 1 == rx) return sum[x]++, void();
    int mid = (lx + rx) / 2;
    if (i < mid) {
        rc[x] = rc[fa];
        update(lc[fa] , lc[x], lx, mid, i);
    }
    else {
        lc[x] = lc[fa];
        update(rc[fa], rc[x], mid, rx, i);
    }
    pull(x);
}
int query(int x, int lx, int rx, int l, int r) {
    if (lx >= r or rx <= l) return 0;
    if (lx >= l and rx <= r) return sum[x];
    int mid = (lx + rx) / 2;
    return query(lc[x],
         lx, mid, l, r) + query(rc[x], mid, rx, l, r);
}
```

3.7 Sparse Table

```
const int N = 5e5 + 5, logN = __lg(N) + 1;
int a[N];
struct sparse_table { // 0-based
  int st[logN][N];
  void init(int n) {
    copy(a, a + n, st[0]);
    for (int i = 1; (1 << i) <= n; i++)
      for (int j = 0; j + (1 << i) - 1 <= n; j++)
        st[i][j] = max(st
            [i - 1][j], st[i - 1][j + (1 << (i - 1))]);
  }
  int qry(int l, int r) {
    int k = __lg(r - l + 1);
    return max(st[k][l], st[k][r - (1 << k) + 1]);
  }
} st;
// st.init(n)
// st.qry(l - 1, r - 1)
```

3.8 Treap

```
#include <bits/stdc++.h>
using namespace std;
mt19937 rng;
struct node {
  node *l, *r;
  int v, p, s; bool t; // val, pri, size, tag
  void pull() {
    s = 1;
    for (auto x : {l, r})
      if (x) s += x->s;
  }
  void push() {
    if (t) {
      swap(l, r), t = 0;
      for (auto& x : {l, r})
        if (x) x->t ^= 1;
    }
  }
  node(int _v
       = 0): v(_v), p(rng()), s(1), t(0), l(0), r(0) {}
};
int sz(node* o) {return o ? o->s : 0;}
node* merge(node* a, node* b) {
  if (!a || !b) return a ? : b;
  if (a->p < b->p) return
       a->push(), a->r = merge(a->r, b), a->pull(), a;
  else return
       b->push(), b->l = merge(a, b->l), b->pull(), b;
```

```
}
void split(node
    * o, node*& a, node*& b, int k) { // a < k, b >= k
  if (!o) return a = b = nullptr, void();
  o->push();
  if (o->v < k) a = o, split(o->r, a->r, b, k);
  else b = o, split(o->l, a, b->l, k);
  o->pull();
}
void insert(node*& o, int k) {
  node *a, *b;
  split(
      o, a, b, k), o = merge(a, merge(new node(k), b));
}
void ssplit(node* o, node
    *& a, node*& b, int k) { // split first k things
  if (!o) return a = b = nullptr, void();
  o->push();
  if (sz(o->l) + 1 <= k
      ) a = o, ssplit(o->r, a->r, b, k - sz(o->l) - 1);
  else b = o, ssplit(o->l, a, b->l, k);
  o->pull();
}
void reverse(node* o, int l, int r) { // [l, r]
  node *a, *b, *c;
  ssplit(o, a, b, l - 1), ssplit(b, b, c, r - l + 1);
  b->t ^= 1, o = merge(a, merge(b, c));
}
/*
node* root = nullptr;
for (int i = 0; i < n; i++)
  root = merge(root, new node(x));
*/
```

3.9 ZKW Segment Tree

```
const int N = 5e5 + 5;
int a[N];

struct seg_tree { // 0-based
  int seg[N << 1], n;
  void upd(int x, int v) {
    for (seg[x += n] = v; x > 1; x >>= 1)
      seg[x >> 1] = max(seg[x], seg[x ^ 1]);
  }
  int qry(int l, int r) { // [ql, qr]
    int ret = -1e9;
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
      if (l & 1) ret = max(ret, seg[l++]);
      if (r & 1) ret = max(ret, seg[--r]);
    }
    return ret;
  }
  void init(int _n) {
    n = _n;
    copy(a, a + n, seg + n);
    for (int i = n - 1; i >= 0; i--)
      seg[i] = max(seg[i << 1], seg[i << 1 | 1]);
  }
} tree;
// tree.init(n)
// tree.qry(l - 1, r - 1)
```

4 Flow

4.1 Bipartite Matching

```
// O(E * sqrt(V))
struct Bipartite_Matching { // 0-base
  int l, r;
  int mp[MAXN], mq[MAXN];
  int dis[MAXN], cur[MAXN];
  vector<int> G[MAXN];
  bool dfs(int u) {
    for (int &i = cur[u]; i < SZ(G[u]); ++i) {
      int e = G[u][i];
      if (!~mq[e]
          || (dis[mq[e]] == dis[u] + 1 && dfs(mq[e])))
        return mp[mq[e] = u] = e, 1;
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    int rt = 0;
    queue<int> q;
    fill_n(dis, l, -1);
    for (int i = 0; i < l; ++i)
      if (!~mp[i])
        q.push(i), dis[i] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int e : G[u])
        if (!~mq[e])
          rt = 1;
        else if (!~dis[mq[e]]) {
          q.push(mq[e]);
          dis[mq[e]] = dis[u] + 1;
        }
    }
    return rt;
  }
  int matching() {
    int rt = 0;
    fill_n(mp, l, -1);
    fill_n(mq, r, -1);
    while (bfs()) {
      fill_n(cur, l, 0);
      for (int i = 0; i < l; ++i)
        if (!~mp[i] && dfs(i))
          ++rt;
    }
    return rt;
  }
  void add_edge(int s, int t) {
    G[s].pb(t);
  }
  void init(int _l, int _r) {
    l = _l, r = _r;
    for (int i = 0; i < l; ++i)
      G[i].clear();
  }
};
```

4.2 Bounded Flow

```
// time complexity: same as Dinic
struct BoundedFlow { // 0-base
  struct edge {
    int to, cap, flow, rev;
  };
  vector<edge> G[N];
  int n, s, t, dis[N], cur[N], cnt[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n + 2; ++i)
      G[i].clear(), cnt[i] = 0;
  }
  void add_edge(int u, int v, int lcap, int rcap) {
    cnt[u] -= lcap, cnt[v] += lcap;
    G[u].pb(edge{v, rcap, lcap, SZ(G[v])});
    G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
  }
  void add_edge(int u, int v, int cap) {
    G[u].pb(edge{v, cap, 0, SZ(G[v])});
    G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
  }
  int dfs(int u, int cap) {
    if (u == t || !cap) return cap;
    for (int &i = cur[u]; i < SZ(G[u]); ++i) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u] + 1 && e.cap != e.flow) {
        int df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
          e.flow += df, G[e.to][e.rev].flow -= df;
          return df;
        }
      }
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    fill_n(dis, n + 3, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (edge &e : G[u])
        if (!~dis[e.to] && e.flow != e.cap)
          q.push(e.to), dis[e.to] = dis[u] + 1;
    }
```

```
    return dis[t] != -1;
  }
  int maxflow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, df;
    while (bfs()) {
      fill_n(cur, n + 3, 0);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
  bool solve() {
    int sum = 0;
    for (int i = 0; i < n; ++i)
      if (cnt[i] > 0)
        add_edge(n + 1, i, cnt[i]), sum += cnt[i];
      else if (cnt[i] < 0) add_edge(i, n + 2, -cnt[i]);
    if (sum != maxflow(n + 1, n + 2)) sum = -1;
    for (int i = 0; i < n; ++i)
      if (cnt[i] > 0)
        G[n + 1].pop_back(), G[i].pop_back();
      else if (cnt[i] < 0)
        G[i].pop_back(), G[n + 2].pop_back();
    return sum != -1;
  }
  int solve(int _s, int _t) {
    add_edge(_t, _s, INF);
    if (!solve()) return -1; // invalid flow
    int x = G[_t].back().flow;
    return G[_t].pop_back(), G[_s].pop_back(), x;
  }
};
```

4.3 Dinic

```
// O(V^2 * E)
// O(min(V^(2/3)
     , E^(1/2)) * E) for unit graph (all cap are same)
// O(E * sqrt(V)) for bipartite matching
struct MaxFlow { // 0-base
  struct edge {
    int to, cap, flow, rev;
  };
  vector<edge> G[MAXN];
  int s, t, dis[MAXN], cur[MAXN], n;
  int dfs(int u, int cap) {
    if (u == t || !cap) return cap;
    for (int &i = cur[u]; i < (int)G[u].size(); ++i) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
        int df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
          e.flow += df;
          G[e.to][e.rev].flow -= df;
          return df;
        }
      }
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    fill_n(dis, n, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
      int tmp = q.front();
      q.pop();
      for (auto &u : G[tmp])
        if (!~dis[u.to] && u.flow != u.cap) {
          q.push(u.to);
          dis[u.to] = dis[tmp] + 1;
        }
    }
    return dis[t] != -1;
  }
  int maxflow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, df;
    while (bfs()) {
      fill_n(cur, n, 0);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void reset() {
    for (int i = 0; i < n; ++i)
      for (auto &j : G[i]) j.flow = 0;
  }
  void add_edge(int u, int v, int cap) {
    G[u].pb(edge{v, cap, 0, (int)G[v].size()});
    G[v].pb(edge{u, 0, 0, (int)G[u].size() - 1});
  }
};
```

4.4 KM

```
// O(n^3), where n is the number
     of vertices on one side of the bipartite graph
// Finds
     the maximum weight matching in a bipartite graph
struct KM { // 0-base
  int w[MAXN][MAXN], hl[MAXN], hr[MAXN], slk[MAXN], n;
  int fl[MAXN], fr[MAXN], pre[MAXN], qu[MAXN], ql, qr;
  bool vl[MAXN], vr[MAXN];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j) w[i][j] = -INF;
  }
  void add_edge(int a, int b, int wei) {
    w[a][b] = wei;
  }
  bool Check(int x) {
    if (vl[x] = 1, ~fl[x])
      return vr[qu[qr++] = fl[x]] = 1;
    while (~x) swap(x, fr[fl[x] = pre[x]]);
    return 0;
  }
  void Bfs(int s) {
    fill(slk, slk + n, INF);
    fill(vl, vl + n, 0), fill(vr, vr + n, 0);
    ql = qr = 0, qu[qr++] = s, vr[s] = 1;
    while (1) {
      int d;
      while (ql < qr)
        for (int x = 0, y = qu[ql++]; x < n; ++x)
          if (!vl[x] &&
              slk[x] >= (d = hl[x] + hr[y] - w[x][y]))
            if (pre[x] = y, d) slk[x] = d;
            else if (!Check(x)) return;
      d = INF;
      for (int x = 0; x < n; ++x)
        if (!vl[x] && d > slk[x]) d = slk[x];
      for (int x = 0; x < n; ++x) {
        if (vl[x]) hl[x] += d;
        else slk[x] -= d;
        if (vr[x]) hr[x] -= d;
      }
      for (int x = 0; x < n; ++x)
        if (!vl[x] && !slk[x] && !Check(x)) return;
    }
  }
  int Solve() {
    fill(fl, fl + n, -1), fill(fr, fr + n, -1),
         fill(hr, hr + n, 0);
    for (int i = 0; i < n; ++i)
      hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) Bfs(i);
    int res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
  }
};
```

4.5 Maximum Simple Graph Matching

```
//  O(V^3) , where V is the number of vertices
struct Matching { // 0-base
  queue<int> q; int n;
  vector<int> fa, s, vis, pre, match;
  vector<vector<int>> G;
  int Find(int u)
  { return u == fa[u] ? u : fa[u] = Find(fa[u]); }
  int LCA(int x, int y) {
    static int tk = 0; tk++; x = Find(x); y = Find(y);
    for (;; swap(x, y)) if (x != n) {
        if (vis[x] == tk) return x;
        vis[x] = tk;
        x = Find(pre[match[x]]);
```

```
    }
  }
  void Blossom(int x, int y, int l) {
    for (; Find(x) != l; x = pre[y]) {
      pre[x] = y, y = match[x];
      if (s[y] == 1) q.push(y), s[y] = 0;
      for (int z : {x, y}) if (fa[z] == z) fa[z] = l;
    }
  }
  bool Bfs(int r) {
    iota(ALL(fa), 0); fill(ALL(s), -1);
    q = queue<int>(); q.push(r); s[r] = 0;
    for (; !q.empty(); q.pop()) {
      for (int x = q.front(); int u : G[x])
        if (s[u] == -1) {
          if (pre[u] = x, s[u] = 1, match[u] == n) {
            for (int a = u, b = x, last;
                 b != n; a = last, b = pre[a])
              last =
                  match[b], match[b] = a, match[a] = b;
            return true;
          }
          q.push(match[u]); s[match[u]] = 0;
        } else if (!s[u] && Find(u) != Find(x)) {
          int l = LCA(u, x);
          Blossom(x, u, l); Blossom(u, x, l);
        }
    }
    return false;
  }
  Matching(int _n) : n(_n), fa(n + 1), s(n + 1), vis
      (n + 1), pre(n + 1, n), match(n + 1, n), G(n) {}
  void add_edge(int u, int v)
  { G[u].pb(v), G[v].pb(u); }
  int solve() {
    int ans = 0;
    for (int x = 0; x < n; ++x)
      if (match[x] == n) ans += Bfs(x);
    return ans;
  } // match[x] == n means not matched
};
```

4.6 MCMF

```
// O(FE * logV), where F is
     the maximum flow,  E is edges, and V is vertices.
struct MinCostMaxFlow { // 0-base
  struct Edge {
    ll from, to, cap, flow, cost, rev;
  } *past[N];
  vector<Edge> G[N];
  int inq[N], n, s, t;
  ll dis[N], up[N], pot[N];
  bool BellmanFord() {
    fill_n(dis, n, INF), fill_n(inq, n, 0);
    queue<int> q;
    auto relax = [&](int u, ll d, ll cap, Edge * e) {
      if (cap > 0 && dis[u] > d) {
        dis[u] = d, up[u] = cap, past[u] = e;
        if (!inq[u]) inq[u] = 1, q.push(u);
      }
    };
    relax(s, 0, INF, 0);
    while (!q.empty()) {
      int u = q.front();
      q.pop(), inq[u] = 0;
      for (auto &e : G[u]) {
        ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
        relax
            (e.to, d2, min(up[u], e.cap - e.flow), &e);
      }
    }
    return dis[t] != INF;
  }
  void solve(int _s
      , int _t, ll &flow, ll &cost, bool neg = true) {
    s = _s, t = _t, flow = 0, cost = 0;
    if (neg) BellmanFord(), copy_n(dis, n, pot);
    for (; BellmanFord(); copy_n(dis, n, pot)) {
      for (int
           i = 0; i < n; ++i) dis[i] += pot[i] - pot[s];
      flow += up[t], cost += up[t] * dis[t];
      for (int i = t; past[i]; i = past[i]->from) {
        auto &e = *past[i];
        e.flow += up[t], G[e.to][e.rev].flow -= up[t];
      }
    }
  }
  void init(int _n) {
    n = _n, fill_n(pot, n, 0);
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void add_edge(ll a, ll b, ll cap, ll cost) {
    G[a].pb(Edge{a, b, cap, 0, cost, SZ(G[b])});
    G[b].pb(Edge{b, a, 0, 0, -cost, SZ(G[a]) - 1});
  }
};
```

4.7 Mimum Vertex Cover

```
// O(VE)
struct Maximum_cardinality_matching {
  int n, k;
  int match[1005]; //right
  int vis[1005]; // left
  vector<int> adj[1005]; // left
  int dfs(int x) {
    vis[x] = 1;
    for (int i : adj[x]) {
      if (match[i] ==
           -1 or (!vis[match[i]] and dfs(match[i]))) {
        match[i] = x;
        return true;
      }
    }
    return false;
  }
  int paired[1005];
  int vis2[1005];
  void dfs2(int x) {
    vis[x] = 1;
    for (int i : adj[x]) {
      if (!vis2[i]) {
        vis2[i] = 1;
        dfs2(match[i]);
      }
    }
  }
  void matching() {
    fill(match + 1, match + 1 + k, -1);
    int res = 0;
    FOR (i, 1, k) {
      fill(vis + 1, vis + 1 + k, 0);
      res += dfs(i);
    }
    FOR (i, 1, k) {
      if (match[i] != -1) {
        paired[match[i]] = 1;
      }
    }
    fill(vis + 1, vis + 1 + k, 0);
    fill(vis2 + 1, vis2 + 1 + k, 0);
    FOR (i, 1, k) {
      if (!paired[i]) {
        dfs2(i);
      }
    }
    vector<int> a, b;
    FOR (i, 1, k) {
      if (!vis[i]) a.pb(i);
      if (vis2[i]) b.pb(i);
    }
    cout << SZ(a) << ' ' << SZ(b) << '\n';
    for (int i : a) cout << i << ' '; cout << '\n';
    for (int i : b) cout << i << ' '; cout << '\n';
    assert(SZ(a) + SZ(b) == res);
  }
```

4.8 Theorem

- Maximum Independent Set: A largest set of non-adjacent vertices.
- Maximum Matching: A largest set of edges with no shared vertices.
- Minimum Vertex Cover: A smallest set of vertices that covers all edges.
- Minimum Edge Cover: A smallest set of edges that covers all vertices.
- Maximum Clique: A largest complete subgraph.
- Properties:
  - |Maximum Matching| = |Minimum Vertex Cover|
  - |Maximum Matching| + |Minimum Edge Cover| = $|V|$
  - |Maximum Independent Set| + |Minimum Vertex Cover| = $|V|$
  - |Maximum Independent Set| = $|V|$ − |Maximum Matching|
  - |Maximum Clique| = |Maximum Independent Set in the Complement Graph|

5 Geometry

5.1 Basic 2D

```cpp
// Courtesy of Jinkela
const double PI = atan2(0.0, -1.0);
template<typename T>
struct point {
  T x, y;
  point() {}
  point(const T&x, const T&y): x(x), y(y) {}
  point operator+(const point &b)const {
    return point(x + b.x, y + b.y);
  }
  point operator-(const point &b)const {
    return point(x - b.x, y - b.y);
  }
  point operator*(const T &b)const {
    return point(x * b, y * b);
  }
  point operator/(const T &b)const {
    return point(x / b, y / b);
  }
  bool operator==(const point &b)const {
    return x == b.x && y == b.y;
  }
  T dot(const
        point &b)const { return x * b.x + y * b.y; }
  T cross(const
        point &b)const { return x * b.y - y * b.x; }
  point normal()const { //求法向量
    return point(-y, x);
  }
  T abs2()const { return dot(*this); }
  T rad(const point &b)const { //兩向量的弧度
    return fabs(atan2(fabs(cross(b)), dot(b)));
  }
  T getA()const { //對x軸的弧度
    T A = atan2(y, x); //超過180度會變負的
    if (A <= -PI / 2)A += PI * 2;
    return A;
  }
};
template<typename T>
struct line {
  line() {}
  point<T> p1, p2;
  T a, b, c; //ax+by+c=0
  line(const
        point<T>&x, const point<T>&y): p1(x), p2(y) {}
  void pton() { //轉成一般式
    a = p1.y - p2
        .y; b = p2.x - p1.x; c = -a * p1.x - b * p1.y;
  }
  T ori(const point<T> &p)const
      { //點和有向直□的關□, >0左邊、=0在□上<0右邊
    return (p2 - p1).cross(p - p1);
  }
  T btw(const point<T> &p)const { //點投影落在□段上<=0
    return (p1 - p).dot(p2 - p);
  }
  bool point_on_segment(const point<T>&p)const {
    return ori(p) == 0 && btw(p) <= 0;
  }
  T dis2(const point<T> &p, bool
        is_segment = 0)const { //點跟直□/□段的距離平方
    point<T> v = p2 - p1, v1 = p - p1;
    if (is_segment) {
      point<T> v2 = p - p2;
      if (v.dot(v1) <= 0)return v1.abs2();
      if (v.dot(v2) >= 0)return v2.abs2();
    }
    T tmp = v.cross(v1); return tmp * tmp / v.abs2();
  }
  T seg_dis2(const line<T> &l)const { //兩□段距離平方
    return min({dis2(l.p1, 1),
         dis2(l.p2, 1), l.dis2(p1, 1), l.dis2(p2, 1)});
  }
  point<T> projection
      (const point<T> &p)const { //點對直□的投影
    point<T> n = (p2 - p1).normal();
    return p - n * (p - p1).dot(n) / n.abs2();
  }
  point<T> mirror(const point<T> &p)const {
    //點對直□的鏡射，要先呼叫pton轉成一般式
    point<T> R; T d = a * a + b * b;
    R.x = (b * b * p.x -
          a * a * p.x - 2 * a * b * p.y - 2 * a * c) / d;
    R.y = (a * a * p.y -
          b * b * p.y - 2 * a * b * p.x - 2 * b * c) / d;
    return R;
  }
  bool parallel(const line &l)const {
    return (p1 - p2).cross(l.p1 - l.p2) == 0;
  }
};
template<typename T>
struct polygon {
  polygon() {}
  vector<point<T> > p;//逆時針順序
  T double_signed_area()const {
    T ans = 0;
    for (int i = p
          .size() - 1, j = 0; j < (int)p.size(); i = j++)
      ans += p[i].cross(p[j]);
    return ans;
  }
  point<T> center_of_mass()const {
    T cx = 0, cy = 0, w = 0;
    for (int i = p.size
          () - 1, j = 0; j < (int)p.size(); i = j++) {
      T a = p[i].cross(p[j]);
      cx += (p[i].
           x + p[j].x) * a; cy += (p[i].y + p[j].y) * a;
      w += a;
    } return point<T>(cx / 3 / w, cy / 3 / w);
  }
  int ahas(const point<T>& t)const { //點是否在簡
      單多邊形□，是的話回傳1、在邊上回傳-1、否則回傳0
    int c = 0; //Works for clockwise input as well
    for (int i
          = 0, j = p.size() - 1; i < p.size(); j = i++) {
      if (line<
           T>(p[i], p[j]).point_on_segment(t))return -1;
      if ((p[i].y > t.y) != (p[j].y > t.y)) {
        T L = (t.x - p[i].x) * (p[j].y - p[i].y);
        T R = (p[j].x - p[i].x) * (t.y - p[i].y);
        if (p[j].y < p[i].y) {L = -L; R = -R;}
        if (L < R)c = !c;
      }
    } return c;
  }
  int point_in_convex(const point<T>&x)const {
    int l = 1, r = (int)p.size() - 2;
    while (l <= r) { //點是否在凸
        多邊形□，是的話回傳1、在邊上回傳-1、否則回傳0
      int mid = (l + r) / 2;
      T a1 = (p[mid] - p[0]).cross(x - p[0]);
      T a2 = (p[mid + 1] - p[0]).cross(x - p[0]);
      if (a1 >= 0 && a2 <= 0) {
        T res
            = (p[mid + 1] - p[mid]).cross(x - p[mid]);
        return res > 0 ? 1 : (res >= 0 ? -1 : 0);
      }
      if (a1 < 0)r = mid - 1; else l = mid + 1;
    } return 0;
  }
  vector<T> getA()const { //凸包邊對x軸的夾角
    vector<T>res;//一定是遞增的
    for (size_t i = 0; i < p.size(); ++i)
      res.push_back
          ((p[(i + 1) % p.size()] - p[i]).getA());
    return res;
  }
  bool line_intersect(const
       vector<T>&A, const line<T> &l)const { //O(logN)
    int f1 = upper_bound(A.begin
          (), A.end(), (l.p1 - l.p2).getA()) - A.begin();
    int f2 = upper_bound(A.begin
          (), A.end(), (l.p2 - l.p1).getA()) - A.begin();
    return l.cross_seg(line<T>(p[f1], p[f2]));
  }
  T diam() {
    int n = p.size(), t = 1;
    T ans = 0; p.push_back(p[0]);
    for (int i = 0; i < n; i++) {
      point<T> now = p[i + 1] - p[i];
      while (now.cross(p[t + 1] - p[
            i]) > now.cross(p[t] - p[i]))t = (t + 1) % n;
      ans = max(ans, (p[i] - p[t]).abs2());
```

```
    } return p.pop_back(), ans;
  }
  T min_cover_rectangle() {
    int n = p.size(), t = 1, r = 1, l;
    if (n < 3)return 0; //也可以做最小周長矩形
    T ans = 1e99; p.push_back(p[0]);
    for (int i = 0; i < n; i++) {
      point<T> now = p[i + 1] - p[i];
      while (now.cross(p[t + 1] - p[
          i]) > now.cross(p[t] - p[i]))t = (t + 1) % n;
      while (now.dot(p[r + 1] -
          p[i]) > now.dot(p[r] - p[i]))r = (r + 1) % n;
      if (!i)l = r;
      while (now.dot(p[l + 1] - p
          [i]) <= now.dot(p[l] - p[i]))l = (l + 1) % n;
      T d = now.abs2();
      T tmp = now.cross(p[t] - p[i]) * (now.
          dot(p[r] - p[i]) - now.dot(p[l] - p[i])) / d;
      ans = min(ans, tmp);
    } return p.pop_back(), ans;
  }
  T dis2(polygon &pl) { //凸包最近距離平方
    vector<point<T> > &P = p, &Q = pl.p;
    int n = P.size(), m = Q.size(), l = 0, r = 0;
    for (int
         i = 0; i < n; ++i)if (P[i].y < P[l].y)l = i;
    for (int
         i = 0; i < m; ++i)if (Q[i].y < Q[r].y)r = i;
    P.push_back(P[0]), Q.push_back(Q[0]);
    T ans = 1e99;
    for (int i = 0; i < n; ++i) {
      while ((P[l] - P[l + 1])
          .cross(Q[r + 1] - Q[r]) < 0)r = (r + 1) % m;
      ans = min(ans, line<T>(P[l],
          P[l + 1]).seg_dis2(line<T>(Q[r], Q[r + 1])));
      l = (l + 1) % n;
    } return P.pop_back(), Q.pop_back(), ans;
  }
  static int sign(const point<T>&t) {
    return (t.y ? t.y : t.x) < 0;
  }
  static bool
       angle_cmp(const line<T>& A, const line<T>& B) {
    point<T> a = A.p2 - A.p1, b = B.p2 - B.p1;
    return sign(a) < sign
        (b) || (sign(a) == sign(b) && a.cross(b) > 0);
  }
  int halfplane_intersection(vector<line<T> > &s) {
    sort(s.begin()
        , s.end(), angle_cmp); //□段左側□該□段半平面
    int L, R, n = s.size();
    vector<point<T> > px(n);
    vector<line<T> > q(n);
    q[L = R = 0] = s[0];
    for (int i = 1; i < n; ++i) {
      while (L < R && s[i].ori(px[R - 1]) <= 0)--R;
      while (L < R && s[i].ori(px[L]) <= 0)++L;
      q[++R] = s[i];
      if (q[R].parallel(q[R
          - 1]) && q[--R].ori(s[i].p1) > 0)q[R] = s[i];
      if (L < R)
          px[R - 1] = q[R - 1].line_intersection(q[R]);
    }
    while (L < R && q[L].ori(px[R - 1]) <= 0)--R;
    p.clear();
    if (R - L <= 1)return 0;
    px[R] = q[R].line_intersection(q[L]);
    for (int i = L; i <= R; ++i)p.push_back(px[i]);
    return R - L + 1;
  }
};
```

5.2 Convex Hull

```
#define f first
#define s second
#define ALL(x) (x).begin(), (x).end()
template <typename T>
pair<T, T> operator
    -(const pair<T, T>& a, const pair<T, T>& b) {
  return {a.f - b.f, a.s - b.s};
}
template <typename T>
int cross(const pair<T,
     T>& o, const pair<T, T>& a, const pair<T, T>& b) {
  auto p = a - o, q = b - o;
  return p.f * q.s - q.f * p.s;
}
template <typename T>
vector
    <pair<T, T>> convex_hull(vector<pair<T, T>> hull) {
  if (hull.size() <= 2) return hull;
  sort(ALL(hull));
  vector<pair<T, T>> stk;
  int n = hull.size();
  for (int i = 0; i < n; i++) {
    while (stk.size() >= 2 && cross
        (stk.end()[-2], stk.end()[-1], hull[i]) <= 0)
      stk.pop_back();
    stk.push_back(hull[i]);
  }
  for (
      int i = n - 2, t = stk.size() + 1; i >= 0; i--) {
    while ((int)stk.size() >= t && cross
        (stk.end()[-2], stk.end()[-1], hull[i]) <= 0)
      stk.pop_back();
    stk.push_back(hull[i]);
  }
  return stk.pop_back(), stk;
}
```

5.3 Dynamic Convex Hull

```
struct Line {
  ll a, b, l = MIN, r = MAX;
  Line(ll a, ll b): a(a), b(b) {}
  ll operator()(ll x) const {
    return a * x + b;
  }
  bool operator<(Line b) const {
    return a < b.a;
  }
  bool operator<(ll b) const {
    return r < b;
  }
};

ll iceil(ll a, ll b) {
  if (b < 0) a *= -1, b *= -1;
  if (a > 0) return (a + b - 1) / b;
  else return a / b;
}

ll intersect(Line a, Line b) {
  return iceil(a.b - b.b, b.a - a.a);
}

struct DynamicConvexHull {
  multiset<Line, less<>> ch;

  void add(Line ln) {
    auto it = ch.lower_bound(ln);
    while (it != ch.end()) {
      Line tl = *it;
      if (tl(tl.r) <= ln(tl.r)) {
        it = ch.erase(it);
      }
      else break;
    }
    auto it2 = ch.lower_bound(ln);
    while (it2 != ch.begin()) {
      Line tl = *prev(it2);
      if (tl(tl.l) <= ln(tl.l)) {
        it2 = ch.erase(prev(it2));
      }
      else break;
    }
    it = ch.lower_bound(ln);
    if (it != ch.end()) {
      Line tl = *it;
      if (tl(tl.l) >= ln(tl.l)) ln.r = tl.l - 1;
      else {
        ll pos = intersect(ln, tl);
        tl.l = pos;
        ln.r = pos - 1;
        ch.erase(it);
        ch.insert(tl);
      }
    }
    it2 = ch.lower_bound(ln);
    if (it2 != ch.begin()) {
      Line tl = *prev(it2);
      if (tl(tl.r) >= ln(tl.r)) ln.l = tl.r + 1;
```

```
      else {
        ll pos = intersect(tl, ln);
        tl.r = pos - 1;
        ln.l = pos;
        ch.erase(prev(it2));
        ch.insert(tl);
      }
    }
    if (ln.l <= ln.r) ch.insert(ln);
  }

  ll query(ll pos) {
    auto it = ch.lower_bound(pos);
    if (it == ch.end()) return 0;
    return (*it)(pos);
  }
};
```

5.4 Segmentation Intersection

```
int sign(ll x) {
  return (x > 0 ? 1 : (x < 0 ? -1 : 0));
}

ll cross
    (pair<ll, ll> o, pair<ll, ll> a, pair<ll, ll> b) {
  return (a.first - o.first) * (b.second - o.second
      ) - (a.second - o.second) * (b.first - o.first);
}

bool intersect1D(ll a, ll b, ll c, ll d) {
  if (a > b) swap(a, b);
  if (c > d) swap(c, d);
  return max(a, c) <= min(b, d);
}

bool intersect2D(pair<ll, ll> a
    , pair<ll, ll> b, pair<ll, ll> c, pair<ll, ll> d) {
  return
       intersect1D(a.first, b.first, c.first, d.first)
         && intersect1D
              (a.second, b.second, c.second, d.second)
         && sign(cross
              (a, b, c)) * sign(cross(a, b, d)) <= 0
         && sign(cross
              (c, d, a)) * sign(cross(c, d, b)) <= 0;
}
```

5.5 Theorem

- Pick's Theorem:
  - If a polygon has vertices with integer coordinates (lattice points), then the area is given by:

$$\text{Area}(P) = i + \frac{1}{2}p - 1$$

    where i is the number of lattice points inside the polygon, and p is the number of lattice points on the perimeter of the polygon.

6 Math

6.1 Big Int

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

template<typename T>
inline string to_string(const T& x) {
  stringstream ss;
  return ss << x, ss.str();
}
using ll = long long;
struct bigN: vector<ll> {
  const static
       int base = 1000000000, width = log10(base);
  bool negative;
  bigN(const_iterator
       a, const_iterator b): vector<ll>(a, b) {}
  bigN(string s) {
    if (s.empty()) return;
    if (s[0] == '-')negative = 1, s = s.substr(1);
    else negative = 0;
    for (int
          i = int(s.size()) - 1; i >= 0; i -= width) {
      ll t = 0;
      for (int j = max(0, i - width + 1); j <= i; ++j)
        t = t * 10 + s[j] - '0';
      push_back(t);
    }
    trim();
  }
  template<typename T>
  bigN(const T &x): bigN(to_string(x)) {}
  bigN(): negative(0) {}
  void trim() {
    while (size() && !back())pop_back();
    if (empty()) negative = 0;
  }
  void carry(int _base = base) {
    for (size_t i = 0; i < size(); ++i) {
      if (at(i) >= 0 && at(i) < _base) continue;
      if (i + 1u == size())push_back(0);
      int r = at(i) % _base;
      if (r < 0)r += _base;
      at(i + 1) += (at(i) - r) / _base;
      at(i) = r;
    }
  }
  int abscmp(const bigN &b) const {
    if (size() > b.size()) return 1;
    if (size() < b.size()) return -1;
    for (int i = int(size()) - 1; i >= 0; --i) {
      if (at(i) > b[i]) return 1;
      if (at(i) < b[i]) return -1;
    }
    return 0;
  }
  int cmp(const bigN &b) const {
    if (negative
         != b.negative) return negative ? -1 : 1;
    return negative ? -abscmp(b) : abscmp(b);
  }
  bool operator
      <(const bigN&b) const {return cmp(b) < 0;}
  bool operator
      >(const bigN&b) const {return cmp(b) > 0;}
  bool operator
      <=(const bigN&b) const {return cmp(b) <= 0;}
  bool operator
      >=(const bigN&b) const {return cmp(b) >= 0;}
  bool operator==(const bigN&b) const {return !cmp(b);}
  bool operator
      !=(const bigN&b) const {return cmp(b) != 0;}
  bigN abs() const {
    bigN res = *this;
    return res.negative = 0, res;
  }
  bigN operator-() const {
    bigN res = *this;
    return res.negative = !negative, res.trim(), res;
  }
  bigN operator+(const bigN &b) const {
    if (negative) return -(-(*this) + (-b));
    if (b.negative) return *this - (-b);
    bigN res = *this;
    if (b.size() > size()) res.resize(b.size());
    for (size_t
          i = 0; i < b.size(); ++i) res[i] += b[i];
    return res.carry(), res.trim(), res;
  }
  bigN operator-(const bigN &b) const {
    if (negative) return -(-(*this) - (-b));
    if (b.negative) return *this + (-b);
    if (abscmp(b) < 0) return -(b - (*this));
    bigN res = *this;
    if (b.size() > size()) res.resize(b.size());
    for (size_t
          i = 0; i < b.size(); ++i) res[i] -= b[i];
    return res.carry(), res.trim(), res;
  }
  bigN convert_base
      (int old_width, int new_width) const {
    vector<
        long long> p(max(old_width, new_width) + 1, 1);
    for (size_t
          i = 1; i < p.size(); ++i)p[i] = p[i - 1] * 10;
    bigN ans;
    long long cur = 0;
    int cur_id = 0;
    for (size_t i = 0; i < size(); ++i) {
      cur += at(i) * p[cur_id];
      cur_id += old_width;
      while (cur_id >= new_width) {
        ans.push_back(cur % p[new_width]);
```

```
      cur /= p[new_width];
      cur_id -= new_width;
    }
  }
  return ans.push_back(cur), ans.trim(), ans;
}
bigN karatsuba(const bigN &b) const {
  bigN res; res.resize(size() * 2);
  if (size() <= 32) {
    for (size_t i = 0; i < size(); ++i)
      for (size_t j = 0; j < size(); ++j)
        res[i + j] += at(i) * b[j];
    return res;
  }
  size_t k = size() / 2;
  bigN a1(begin(), begin() + k);
  bigN a2(begin() + k, end());
  bigN b1(b.begin(), b.begin() + k);
  bigN b2(b.begin() + k, b.end());
  bigN a1b1 = a1.karatsuba(b1);
  bigN a2b2 = a2.karatsuba(b2);
  for (size_t i = 0; i < k; ++i)a2[i] += a1[i];
  for (size_t i = 0; i < k; ++i)b2[i] += b1[i];
  bigN r = a2.karatsuba(b2);
  for (size_t
       i = 0; i < a1b1.size(); ++i)r[i] -= a1b1[i];
  for (size_t
       i = 0; i < a2b2.size(); ++i)r[i] -= a2b2[i];
  for (size_t
       i = 0; i < r.size(); ++i)res[i + k] += r[i];
  for (size_t
       i = 0; i < a1b1.size(); ++i)res[i] += a1b1[i];
  for (size_t i = 0; i
       < a2b2.size(); ++i)res[i + size()] += a2b2[i];
  return res;
}
bigN operator*(const bigN &b) const {
  const static int mul_base
       = 1000000, mul_width = log10(mul_base);
  bigN A = convert_base(width, mul_width);
  bigN B = b.convert_base(width, mul_width);
  int n = max(A.size(), B.size());
  while (n & (n - 1))++n;
  A.resize(n), B.resize(n);
  bigN res = A.karatsuba(B);
  res.negative = negative != b.negative;
  res.carry(mul_base);
  res = res.convert_base(mul_width, width);
  return res.trim(), res;
}
bigN operator*(long long b) const {
  bigN res = *this;
  if (b < 0)res.negative = !negative, b = -b;
  for (size_t
       i = 0, is = 0; i < res.size() || is; ++i) {
    if (i == res.size()) res.push_back(0);
    long long a = res[i] * b + is;
    is = a / base;
    res[i] = a % base;
  }
  return res.trim(), res;
}
bigN operator/(const bigN &b) const {
  int norm = base / (b.back() + 1);
  bigN x = abs() * norm;
  bigN y = b.abs() * norm;
  bigN q, r;
  q.resize(x.size());
  for (int i = int(x.size()) - 1; i >= 0; --i) {
    r = r * base + x[i];
    int s1 = r.size() <= y.size() ? 0 : r[y.size()];
    int s2
         = r.size() < y.size() ? 0 : r[y.size() - 1];
    int d = (ll(base) * s1 + s2) / y.back();
    r = r - y * d;
    while (r.negative) r = r + y, --d;
    q[i] = d;
  }
  q.negative = negative != b.negative;
  return q.trim(), q;
}
bigN operator%(const bigN &b) const {
  return *this - (*this / b) * b;
}
friend istream& operator>>(istream &ss, bigN &b) {
  string s;
    return ss >> s, b = s, ss;
  }
  friend
      ostream& operator<<(ostream &ss, const bigN &b) {
    if (b.negative) ss << '-';
    ss << (b.empty() ? 0 : b.back());
    for (int i = int(b.size()) - 2; i >= 0; --i)
      ss << setw(width) << setfill('0') << b[i];
    return ss;
  }
  template<typename T>
  operator T() {
    stringstream ss;
    ss << *this;
    T res;
    return ss >> res, res;
  }
};
```

6.2 Chinese Remainder

```
int solve(int n, vector<int> &a, vector<int> &m){
    int M = 1;
    for(auto i : m) M *= i;
    int ans = 0;
    for(int i = 0; i < n; i++){
        int m1 = M / m[i], m2 = extgcd(m1, m[i]).X;
        ans += (a[i] * m1 * m2) % M;
    }
    ans = ans % M + M;
    ans %= M;
    return ans;
}
```

6.3 Extgcd

```
pair<ll, ll> extgcd(ll a, ll b) {
  if (b == 0) return {1, 0};
  auto [xp, yp] = extgcd(b, a % b);
  return {yp, xp - a / b * yp};
}
```

6.4 FFT

```
// Remember not to output -0
/*
   polynomial multiply:
   DFT(a, len); DFT(b, len);
   for(int i=0;i<len;i++) c[i] = a[i]*b[i];
   iDFT(c, len);
   (len must be 2^k and >= 2*(max(a, b)))
   Hand written Cplx would be 2x faster
 */
Cplx omega[2][N];
void init_omega(int n) {
  static constexpr llf PI = acos(-1);
  const llf arg = (PI + PI) / n;
  for (int i = 0; i < n; ++i)
    omega[0][i] = {cos(arg * i), sin(arg * i)};
  for (int i = 0; i < n; ++i)
    omega[1][i] = conj(omega[0][i]);
}
void tran(Cplx arr[], int n, Cplx omg[]) {
  for (int i = 0, j = 0; i < n; ++i) {
    if (i > j)swap(arr[i], arr[j]);
    for (int l = n >> 1; (j ^= l) < l; l >>= 1);
  }
  for (int l = 2; l <= n; l <<= 1) {
    int m = l >> 1;
    for (auto p = arr; p != arr + n; p += l) {
      for (int i = 0; i < m; ++i) {
        Cplx t = omg[n / l * i] * p[m + i];
        p[m + i] = p[i] - t; p[i] += t;
      }
    }
  }
}
void DFT(Cplx arr[], int n) {tran(arr, n, omega[0]);}
void iDFT(Cplx arr[], int n) {
  tran(arr, n, omega[1]);
  for (int i = 0; i < n; ++i) arr[i] /= n;
}
```

6.5 Gauss Elimination

```
#include <bits/stdc++.h>

std::bitset<1000> a[500];
```

```
int main() {
  int n; std::cin >> n;
  for (int i = 0; i < n; ++i) {
    for (int j = 0, t; j < n; ++j)
      std::cin >> t, a[i][j] = t;
    a[i][i + n] = 1;
  }
  for (int i = 0; i < n; ++i) {
    int t;
    for (t = i; t < n; ++t) if (a[t][i]) break;
    if (t == n) return std::cout << "-1\n", 0;
    std::swap(a[i], a[t]);
    for (int j
         = i + 1; j < n; ++j) if (a[j][i]) a[j] ^= a[i];
  }
  for (int i = n - 1; i >= 0; --i)
    for (int j = i - 1; j >= 0; --j)
      if (a[j][i]) a[j] ^= a[i];
  for (int i = 0; i < n; ++i) {
    std::vector<int> ans;
    for (int j = n; j < 2 *
         n; ++j) if (a[i][j]) ans.push_back(j - n + 1);
    for (size_t j = 0; j < ans.size(); ++j)
      std::cout << ans[j] << " \n"[j == ans.size()];
  }
  return 0;
}
```

6.6 Gauss Elimination2

```
using ll = long long;
const ll mod = 998244353;
ll fp(ll a, ll b) {
  ll ret = 1;
  for (; b; b >>= 1, a = a * a % mod)
    if (b & 1) ret = ret * a % mod;
  return ret;
}
vector<ll> gauss_elimination
    (vector<vector<ll>>& a) { // n * (n+1)
  // if a[i][j] < 0, a[i][j] += mod
  int n = a.size();
  bool swp = 0;
  for (int i = 0; i < n; i++) {
    for (int k = i; k < n; k++) {
      if (a[i][i] == 0 && a[k][i] != 0) {
        swap(a[i], a[k]), swp ^= 1; // det = -det
        break;
      }
    }
    if (a[i][i] == 0) return {}; // 0
    ll inv = fp(a[i][i], mod - 2);
    for (int j = 0; j < n; j++) {
      if (i != j) {
        ll tmp = a[j][i] * inv % mod;
        for (int k = i; k <= n; k++)
          a[j][k] = (a[
              j][k] - tmp * a[i][k] % mod + mod) % mod;
      }
    }
  }
  // general solution
  vector<ll> ans(n);
  for (int i = 0; i < n; i++)
    ans[i] = a[i][n] * fp(a[i][i], mod - 2) % mod;
  return ans;
  // det
  // ll ret = 1;
  // for (
     int i = 0; i < n; i++) ret = ret * a[i][i] % mod;
  // return swp ? mod - ret : ret;
}
```

6.7 Karatsuba

```
const ll base = 10000000;
void karatsuba(const vector<ll
    >& f, const vector<ll>& g, vector<ll>& c, int n) {
  if (n <= 32) {
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        c[i + j] += f[i] * g[j];
    return;
  }
  vector
      <ll> f1(n / 2), f2(n / 2), g1(n / 2), g2(n / 2);
  copy(f.begin(), f.begin() + n / 2, f1.begin()
      ), copy(f.begin() + n / 2, f.end(), f2.begin());
  copy(g.begin(), g.begin() + n / 2, g1.begin()
      ), copy(g.begin() + n / 2, g.end(), g2.begin());
  vector<ll> t1(n), t2(n), t3(n);
  karatsuba(
      f1, g1, t1, n / 2), karatsuba(f2, g2, t2, n / 2);
  for (int i = 0; i < n / 2; i++) f1[i] += f2[i];
  for (int i = 0; i < n / 2; i++) g1[i] += g2[i];
  karatsuba(f1, g1, t3, n / 2);
  for (int i = 0; i < n; i++) t3[i] -= t1[i] + t2[i];
  for (int i = 0; i < n; i++)
    c[i] += t1
        [i], c[i + n] += t2[i], c[i + n / 2] += t3[i];
}
void mul(const vector
    <ll>& a, const vector<ll>& b, vector<ll>& c) {
  int n = a.size(), m = b.size(), t = max(n, m), p = 1;
  while (p < t) p <<= 1;
  vector<ll> aa(p), bb(p);
  copy(a.begin(), a.end(), aa
      .begin()), copy(b.begin(), b.end(), bb.begin());
  c.assign(p << 1, 0), karatsuba(aa, bb, c, p);
  p = n + m - 1;
  for (int i = 0; i < p; i++)
    c[i + 1] += c[i] / base, c[i] %= base;
  if (c[p]) p++;
  c.resize(p);
}
```

6.8 Linear Sieve

```
vector<bool> isp;
vector<int> p;
void sieve(int n) {
  p.clear(), isp.assign(n + 1, 1);
  isp[0] = isp[1] = 0;
  for (int i = 2; i <= n; i++) {
    if (isp[i]) p.eb(i);
    for (const auto& x : p) {
      if (1LL * i * x > n) break;
      isp[i * x] = 0;
      if (i % x == 0) break;
    }
  }
}
```

6.9 Matrix

```
template <typename T> using vec = vector<T>;
template <typename T> using matrix = vec<vec<T>>;
constexpr int mod = 1e9 + 7;
template <typename T>
matrix<T>
    operator*(const matrix<T>& a, const matrix<T>& b) {
  int n = a.size(), r = b.size(), m = b.front().size();
  matrix<T> ret(n, vec<T>(m));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      for (int k = 0; k < r; k++)
        ret[i][j] += 1LL *
            a[i][k] * b[k][j] % mod, ret[i][j] %= mod;
  return ret;
}
```

6.10 Miller Rabin

```
using ll = ll;
ll mod_mul(ll a, ll b, ll m) {
  a %= m, b %= m;
  ll y = (ll)((
      double)a * b / m + 0.5); /* fast for m < 2^58 */
  ll r = (a * b - y * m) % m;
  return r < 0 ? r + m : r;
}
template<typename T>
T pow(T a, T b, T mod) { //a^b%mod
  T ans = 1;
  for (; b; a = mod_mul(a, a, mod), b >>= 1)
    if (b & 1) ans = mod_mul(ans, a, mod);
  return ans;
}
int sprp[3] = {2, 7, 61}; // range of int
int llsprp[7] = {2, 325, 9375, 28178, 450775,
     9780504, 1795265022}; // range of unsigned ll
template<typename T>
bool isprime(T n, int *sprp, int num) {
```

```
  if (n == 2)return 1;
  if (n < 2 || n % 2 == 0) return 0;
  int t = 0;
  T u = n - 1;
  for (; u % 2 == 0; ++t)u >>= 1;
  for (int i = 0; i < num; ++i) {
    T a = sprp[i] % n;
    if (a == 0 || a == 1 || a == n - 1) continue;
    T x = pow(a, u, n);
    if (x == 1 || x == n - 1) continue;
    for (int j = 1; j < t; ++j) {
      x = mod_mul(x, x, n);
      if (x == 1) return 0;
      if (x == n - 1) break;
    }
    if (x == n - 1) continue;
    return 0;
  }
  return 1;
}
```

6.11 Mobius

```
int mu[MAXN], lp[MAXN];
void build() {
  mu[1] = 1;
  FOR (i, 2, MAXN - 1) {
    if (!lp[i]) {
      for (int j = i; j < MAXN; j += i) {
        lp[j] = i;
      }
    }
    if (i / lp[i] % lp[i])
      mu[i] = -mu[i / lp[i]];
  }
}
```

6.12 NTT

```
const int G = 3, P = 998244353;
const int sval = 100, split = log10(sval);
int fpow(int x, int y) {
  int ret = 1;
  for (; y; y >>= 1, x = 1LL * x * x % P)
    if (y & 1) ret = 1LL * ret * x % P;
  return ret;
}
void ntt(vector<int>& x, int lim, int opt) {
  for (int i = 1, j = 0; i < lim; i++) {
    for (int k = lim >> 1; !((j ^= k) & k); k >>= 1);
    if (i < j) swap(x[i], x[j]);
  }
  for (int m = 2; m <= lim; m <<= 1) {
    int k = m >> 1;
    int gn = fpow(G, (P - 1) / m);
    for (int i = 0; i < lim; i += m) {
      int g = 1;
      for (int
            j = 0; j < k; ++j, g = 1LL * g * gn % P) {
        int tmp = 1LL * x[i + j + k] * g % P;
        x[i + j + k] = (x[i + j] - tmp + P) % P;
        x[i + j] = (x[i + j] + tmp) % P;
      }
    }
  }
  if (opt == -1) {
    reverse(x.begin() + 1, x.begin() + lim);
    int inv = fpow(lim, P - 2);
    for (int i = 0; i < lim; ++i)
      x[i] = 1LL * x[i] * inv % P;
  }
}
vector<int> mul(vector<int> a, vector<int> b) {
  int lim = 1, n = a.size(), m = b.size();
  while (lim < (n + m - 1)) lim <<= 1;
  a.resize(lim + 1), b.resize(lim + 1);
  ntt(a, lim, 1), ntt(b, lim, 1);
  for (int i = 0; i < lim; ++i)
    a[i] = 1LL * a[i] * b[i] % P;
  ntt(a, lim, -1);
  int len = 0;
  for (int i = 0; i < lim; ++i) {
    if (a[i] >= sval) len
         = i + 1, a[i + 1] += a[i] / sval, a[i] %= sval;
    if (a[i]) len = max(len, i);
  }
  while (a[len] >= sval) a[
      len + 1] += a[len] / sval, a[len] %= sval, len++;
  return a.resize(len + 1), a;
}
void print(const vector<int>& v) {
  if (!v.size()) return;
  cout << v.back();
  for (int i = v.size() - 2; ~i; --i)
    cout << setfill('0') << setw(split) << v[i];
  cout << '\n';
}
int main() {
  ios::sync_with_stdio(false), cin.tie(nullptr);
  string stra, strb;
  while (cin >> stra >> strb) {
    vector<int> a((stra.size() + split - 1) / split);
    vector<int> b((strb.size() + split - 1) / split);
    int tmp = stra.size();
    for (auto& i : a)
      tmp -= split, i = atoi(stra.substr(max
          (0, tmp), min(split, split + tmp)).data());
    tmp = strb.size();
    for (auto& i : b)
      tmp -= split, i = atoi(strb.substr(max
          (0, tmp), min(split, split + tmp)).data());
    print(mul(a, b));
  }
  return 0;
}
```

6.13 Pollard Rho

```
// does not work when n is prime
ll add(ll
     a, ll b, ll m) {return (a += b) > m ? a - m : a;}
ll mul(ll a, ll b, ll m) {
  a %= m, b %= m;
  ll y = (ll)((
      double)a * b / m + 0.5); /* fast for m < 2^58 */
  ll r = (a * b - y * m) % m;
  return r < 0 ? r + m : r;
}
ll f(ll
     x, ll mod) { return add(mul(x, x, mod), 1, mod); }
ll pollard_rho(ll n) {
  if (!(n & 1)) return 2;
  while (true) {
    ll y =
         2, x = rand() % (n - 1) + 1, res = 1, tmp = 1;
    for (int sz = 2; res == 1; sz *= 2, y = x) {
      for (int
            i = 0, t = 0; i < sz && res <= 1; i++, t++) {
        x = f(x, n); tmp = mul(tmp, abs(x - y), n);
        if (!(t & 31) ||
              i + 1 == sz) res = __gcd(tmp, n), tmp = 1;
      }
    }
    if (res != 0 && res != n) return res;
  }
}
```

6.14 Primes

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

6.15 Primitive Root

```
// g is O(log^6 n).
// Runtime is O(ans * log
     phi(n) * log n), which is approximately O(log^8 n).
// #define int long long
int fp(int a, int b, int p) {
  int ret = 1;
  for (; b; b >>= 1, a = a * a % m)
    if (b & 1) ret = ret * a % m;
  return ret;
}

int generator(int p) {
  vector<int> fact;
  int phi = p - 1,  n = phi;
  for (int i = 2; i * i <= n; ++i)
    if (n % i == 0) {
```

```
      fact.push_back(i);
      while (n % i == 0) n /= i;
    }
  if (n > 1) fact.push_back (n);

  for (int res = 2; res <= p; ++res) {
    bool ok = true;
    for (size_t i = 0; i < fact.size() && ok; ++i)
      ok &= fp(res, phi / fact[i], p) != 1;
    if (ok)  return res;
  }
  return -1;
}
```

7 String

7.1 AC

```
struct ACautomata {
  struct Node {
    int cnt;
    Node *go[26], *fail, *dic;
    Node () {
      cnt = 0, fail = 0, dic = 0;
      memset(go, 0, sizeof(go));
    }
  } pool[1048576], *root;
  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) {
    for (int i = pos; i < str.size(); i++) {
      if (!cur->go[str[i] - 'a'])
        cur->go[str[i] - 'a'] = new_Node();
      cur = cur->go[str[i] - 'a'];
    }
    cur->cnt++;
  }
  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;
          fr->go[i]->
              fail = ptr = (ptr ? ptr->go[i] : root);
          fr->go[i]->dic = (ptr->cnt ? ptr : ptr->dic);
          que.push(fr->go[i]);
        }
      }
    }
  }
} AC;
```

7.2 Hash

```
struct Hash {
  vector<ll> h;
  vector<ll> f;
  Hash(string s, int p = 127) {
    h.assign((int)s.size() + 1, 0);
    f.assign((int)s.size() + 1, 0);
    f[0] = 1;
    for (int i = 1; i <=
        (int)s.size(); ++ i) f[i] = f[i - 1] * p % MOD;
    for (int i = 1; i <= (int)s.size(); ++ i)
      h[i] = (h[i - 1] * p + s[i - 1]) % MOD;
  }
  int query(int l, int r) { // 0-based [l, r]
    if (r < l) return 0;
    return ((h[r
        + 1] - h[l] * f[r - l + 1]) % MOD + MOD) % MOD;
  }
};
```

7.3 KMP

```
#define pb push_back
const int N = 1e6 + 5;
int F[N];
vector<int> match(string A, string B) {
  vector<int> ans;
  F[0] = -1, F[1] = 0;
  for (int
        i = 1, j = 0; i < (int)B.size(); F[++i] = ++j) {
    if (B[i] == B[j]) F[i] = F[j]; // optimize
    while (j != -1 && B[i] != B[j]) j = F[j];
  }
  for (int i = 0, j = 0; i < (int)A.size(); ++i) {
    while (j != -1 && A[i] != B[j]) j = F[j];
    if (++j
          == (int)B.size()) ans.pb(i + 1 - j), j = F[j];
  }
  return ans;
}
```

7.4 Manacher

```
// P[2i] := max 2j+1: s[i-j, i+j] is palindromic
// P[2i-1] := max 2j: s[i-j, i+j) is palindromic
// maximal
     palindrome: s.substr((1 + i - P[i]) >> 1, P[i])
vector<unsigned> Manacher(const string &s) {
  unsigned L = 0, R = 1;
  vector<unsigned> P; P.reserve((s.size() << 1) - 1);
  P.push_back(1);
  for (unsigned i = 1; i < s.size(); ++i)
    for (int j = 0; j < 2; ++j) {
      if (i < R) {
        const int k = ((L + R - i) << 1) - j - 1;
        if (P[k] >> 1 <
            R - i - j) { P.push_back(P[k]); continue; }
        L = (i << 1) - R + j;
      }
      else R = (L = i) + j;
      while (L > 0 &&
          R < s.size() && s[L - 1] == s[R]) {--L; ++R;}
      P.push_back(R - L);
    }
  return P;
}
```

7.5 SA

```
const int N = 2e5 + 5;

string s;
int sa[N], tmp[2][N], c[N], rk[N], h[N];
// lcp(sa[i], sa[j]) = min{h[k]} where i <= k <= j

void suffix_array() {
  int *x = tmp[0], *y = tmp[1], m = 256, n = s.size();
  fill(c, c + m, 0);
  for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
  partial_sum(c, c + m, c);
  for (int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
  for (int k = 1; k < n; k <<= 1) {
    fill(c, c + m, 0);
    for (int i = 0; i < n; i++) c[x[i]]++;
    partial_sum(c, c + m, c);
    int p = 0;
    for (int i = n - k; i < n; i++) y[p++] = i;
    for (int i = 0; i < n; i++)
      if (sa[i] >= k) y[p++] = sa[i] - k;
    for (int i
          = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
    y[sa[0]] = p = 0;
    for (int i = 1; i < n; i++) {
      int a = sa[i], b = sa[i - 1];
      if (x[a] != x[b] || a + k >=
            n || b + k >= n || x[a + k] != x[b + k]) p++;
      y[sa[i]] = p;
    }
    if (n == p + 1) break;
    swap(x, y), m = p + 1;
  }
}

void LCP() {
  int n = s.size(), val = 0;
  for (int i = 0; i < n; i++) rk[sa[i]] = i;
  for (int i = 0; i < n; i++) {
    if (rk[i] == 0) h[rk[i]] = 0;
    else {
      if (val) val--;
      int p = sa[rk[i] - 1];
      while (val + i < n && val
            + p < n && s[val + i] == s[val + p]) val++;
```

```
      h[rk[i]] = val;
    }
  }
}
// cin >> s, suffix_array(), LCP();
```

7.6 SA2

```
void counting_sort
    (vector<int> &dest, const vector<int> &src
    , int bucket_count, function<int(const int&)> f) {
  int *bucket_begin = new
       int[bucket_count], *buf = new int[src.size()];
  fill(bucket_begin, bucket_begin + bucket_count, 0);
  for (int i = 0; i < src.size(); ++i)
    if ((buf[i] = f(src[i])) + 1 < bucket_count)
      ++bucket_begin[buf[i] + 1];
  partial_sum(bucket_begin
      , bucket_begin + bucket_count, bucket_begin);
  dest.resize(src.size());
  for (int i = 0; i < src.size(); ++i)
    dest[bucket_begin[buf[i]]++] = src[i];
  delete[] bucket_begin; delete[] buf;
}
#define
     a 'a'  // The smallest character in the alphabet
#define sz 26  // The
     size of the alphabet. The alphabet is [a, a + sz)
vector<int> suffix_array(const string &s) {
  vector<int> SA, sa(s.size());
  SA.reserve(s.size()); iota(sa.begin(), sa.end(), 0);
  counting_sort(SA,
      sa, sz, [&](const int &i) { return s[i] - a; });
  int *R = new int[SA.size()], *r = new int[SA.size()];
  R[SA[0]]
       = 1;  // R = 0 is reserved for the empty string
  for (int i = 1; i < SA.size(); ++i)
    R[SA[i]] = s
        [SA[i]] == s[SA[i - 1]] ? R[SA[i - 1]] : i + 1;
  int L = 1;
  while (L < s.size()) {
    auto R2 = [&](const int &i) {
      if (i + L < SA.size()) return R[i + L];
      return 0;  // so
           that when L = 1, "a" is ordered before "aa"
    };
    counting_sort(sa, SA, SA.size() + 1, R2);
    counting_sort(SA, sa, SA.size
        (), [&](const int &i) { return R[i] - 1; });
    r[SA[0]] = 1;
    for (int i = 1; i < SA.size(); ++i)
      if (R[SA[i]] ==
           R[SA[i - 1]] && R2(SA[i]) == R2(SA[i - 1]))
        r[SA[i]] = r[SA[i - 1]];
      else r[SA[i]] = i + 1;
    swap(R, r); L <<= 1;
  }
  delete[] R; delete[] r; return SA;
}
#undef a
#undef sz
```

7.7 SAIS

```
const int N = 300010;
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++)
  bool _t[N * 2]; int _s[N * 2], _sa[N * 2];
  int _c[N * 2], x[N], _p[N], _q[N * 2], hei[N], r[N];
  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;
int H[N], SA[N], RA[N];
void suffix_array(int* ip, int len) {
  // should padding a zero in the back
  // ip is int array, len is array length
  // ip[0..n-1] != 0, and ip[len]=0
  ip[len++] = 0; sa.build(ip, len, 128);
  memcpy(H, sa.hei
      + 1, len << 2); memcpy(SA, sa._sa + 1, len << 2);
  for (int i = 0; i < len; i++) RA[i] = sa.r[i] - 1;
  // resulting height, sa array \in [0,len)
}
```

7.8 Suffix Automaton

```
// O(n)
// find all suffix substrings in lexicographical order
#include <bits/stdc++.h>
class SuffixAutomaton {
public:
  static const int MAXN = 500 << 1;
  static const int MAXC = 26;
  struct Node {
    Node *next[MAXC], *pre;
    int step;
    Node() {
      pre = NULL, step = 0;
      memset(next, 0, sizeof(next));
    }
  } _mem[MAXN];
  int size;
  Node *root, *tail;
  void init() {
    size = 0;
    root = tail = newNode();
  }
  Node* newNode() {
    Node *p = &_mem[size++];
    *p = Node();
    return p;
  }
  int toIndex(char c) { return c - 'A'; }
  char toChar(int c) { return c + 'A'; }
  void add(char c, int len) {
    c = toIndex(c);
    Node *p, *q, *np, *nq;
    p = tail, np = newNode();
    np->step = len;
    for (; p && p->next[c] == NULL; p = p->pre)
      p->next[c] = np;
    tail = np;
    if (p == NULL) {
      np->pre = root;
    } else {
      if (p->next[c]->step == p->step + 1) {
        np->pre = p->next[c];
      } else {
```

```
        q = p->next[c], nq = newNode();
        *nq = *q;
        nq->step = p->step + 1;
        q->pre = np->pre = nq;
        for (; p && p->next[c] == q; p = p->pre)
          p->next[c] = nq;
      }
    }
  }
  void build(const char *s) {
    init();
    for (int i = 0; s[i]; i++)
      add(s[i], i + 1);
  }
  void dfs(Node *u, int idx, char path[]) {
    for (int i = 0; i < MAXC; i++) {
      if (u->next[i]) {
        path[idx] = toChar(i);
        path[idx + 1] = '\0';
        puts(path);
        dfs(u->next[i], idx + 1, path);
      }
    }
  }
  void print() {
    char s[1024];
    dfs(root, 0, s);
  }
} SAM;
int main() {
  char s[1024];
  while (scanf("%s", s) == 1) {
    SAM.build(s);
    SAM.print();
  }
  return 0;
}
```

7.9 Trie

```
int trie[MAXN * 31][2], node;
int tag[MAXN * 31];

void add(int x) {
    int now = 0;
    for (int i = 30; i >= 0; i--) {
        if (!trie[now][x
             >> i & 1]) trie[now][x >> i & 1] = ++node;
        now = trie[now][x >> i & 1];
        tag[now]++;
    }
}
void del(int x) {
    int now = 0;
    for (int i = 30; i >= 0; i--) {
        now = trie[now][x >> i & 1];
        tag[now]--;
    }
}
int qry(int x) {
    int now = 0, res = 0;
    for (int i = 30; i >= 0; i--) {
        int id = (x >> i & 1) ^ 1;
        if (!tag[trie[now][id]]) id ^= 1;
        now = trie[now][id];
        res = res * 2 + id;
    }
    return res;
}
```

7.10 Z

```
void z_value(const char *s, int len, int *z) {
  z[0] = len;
  for (int i = 1, l = 0, r = 0; i < len; i++) {
    z[i] = i < r ? (i
         - l + z[i - l] < z[l] ? z[i - l] : r - i) : 0;
    while (i
         + z[i] < len && s[i + z[i]] == s[z[i]]) ++z[i];
    if (i + z[i] > r) l = i, r = i + z[i];
  }
}
```

8 Others

8.1 Aliens

```
/*
實際上如果這邊根本是平的，那我們只要讓二分艘找到最
     小的P讓他的切點不超過K，那就保證了這條F會貼在上面
ll mid = (l+r < 0 ? (l + r) / 2: (l + r + 1) / 2)
while(l < r){
    int m = (l + r) / 2;
    if(calc(m) <= K) r = m;
    else l = m + 1;
}
*/
#include <bits/stdc++.h>
#define F first
#define S second
#define int long long
using namespace std;

bool operator<(
    const pair<int, int> &a, const pair<int, int> &b) {
    return a.F < b.F or (a.F == b.F and a.S > b.S);
}
#define chmax(a, b) a = (a) < (b) ? (b) : (a)
int n, k;
int a[1000005];
pair<int, int> dp[1000005];
vector<int> last(100005, 0);

pair<int, int> DP(int penalty) {
    last.assign(100005, 0);
    pair<int, int> ans = {0, 0};
    int l = 0;
    for (int i = 1; i <= n; i++) {
        while (l < last[a[i]]) {
            l++;
            chmax(ans, dp[l]);
        }
        dp[i] = {ans.F + i - l - penalty, ans.S + 1};
        last[a[i]] = i;
    }
    while (l < n) {
        l++;
        chmax(ans, dp[l]);
    }
    return ans;
}

signed main() {
    ios_base::sync_with_stdio(0), cin.tie(0);
    cin >> n >> k;
    for (int i = 1; i <= n; i++) cin >> a[i];
    int l = -1, r = 2000000;
    while (l < r - 1) {
        int m = (l + r) / 2;
        pair<int, int> res = DP(m);
        if (res.S <= k) {
            r = m;
        } else
            l = m;
    }
    auto res = DP(r);
    cout << res.F + k * r << '\n';
}
```

8.2 Knapsack on Tree

```
#include <bits/stdc++.h>
#define F first
#define S second
#define pb push_back
#define all(x) begin(x), end(x)
#ifdef LOCAL
#define HEHE freopen("in.txt", "r", stdin);
#else
#define HEHE ios_base::sync_with_stdio(0), cin.tie(0);
#endif
using namespace std;

#define chmax(a, b) (a) = (a) < (b) ? (b) : (a)
#define chmin(a, b) (a) = (a) < (b) ? (a) : (b)

#define ll long long

#define FOR(i, a, b) for (int i = a; i <= b; i++)

int N, W, cur;
vector<int> w, v, sz;
vector<vector<int>> adj, dp;
```

```
void dfs(int x) {
    sz[x] = 1;
    for (int i : adj[x]) dfs(i), sz[x] += sz[i];
    cur++;
    // choose x
    FOR (i, w[x], W) {
        dp[cur][i] = dp[cur - 1][i - w[x]] + v[x];
    }
    // not choose x
    FOR (i, 0, W) {
        chmax(dp[cur][i], dp[cur - sz[x]][i]);
    }
}

signed main() {
    HEHE
    cin >> N >> W;
    adj.resize(N + 1);
    w.assign(N + 1, 0);
    v.assign(N + 1, 0);
    sz.assign(N + 1, 0);
    dp.assign(N + 2, vector<int>(W + 1, 0));
    FOR (i, 1, N) {
        int p; cin >> p;
        adj[p].pb(i);
    }
    FOR (i, 1, N) cin >> w[i];
    FOR (i, 1, N) cin >> v[i];
    dfs(0);
    cout << dp[N + 1][W] << '\n';
}
```

8.3 Mo

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

const int N = 2e5 + 5, sqN = sqrt(N) + 5;
int a[N], ans[N], n, q, sz; // maybe need blk[sqN];

struct Query {
  int ql, qr, id;
  bool operator<(const Query& b) const {
    int aa = ql / sz, bb = b.ql / sz;
    if (aa != bb) return aa < bb;
    else return qr < b.qr;
  }
} Q[N];

void add(int x) {}
void sub(int x) {}
int qry(int k) {}

int main() {
  ios::sync_with_stdio(false), cin.tie(nullptr);
  cin >> n >> q, sz = sqrt(n);
  for (int i = 0; i < n; i++) cin >> a[i];
  for (int i = 0, ql, qr; i < q; i++)
    cin >> ql >> qr, Q[i] = {ql - 1, qr - 1, i};
  // Mo's algorithm
  sort(Q, Q + q); /* remember initialize arrays */
  int l = 0, r = -1;
  for (int i = 0; i < q; i++) {
    auto [ql, qr, k, id] = Q[i];
    while (r < qr) add(a[++r]);
    while (r > qr) sub(a[r--]);
    while (l < ql) sub(a[l++]);
    while (l > ql) add(a[--l]);
    ans[id] = qry(k);
  }
  for (int i = 0; i < q; i++) cout << ans[i] << '\n';
}
```

8.4 Mono Slope

```
struct Line{
  ll a, b;
  ll l = MIN, r = MAX;
  Line(ll a, ll b): a(a), b(b) {}
  ll operator()(ll x){
    return a * x + b;
  }
};

deque<Line> dq;
ll iceil(ll a, ll b){
  if(b < 0) a *= -1, b *= -1;
  if(a > 0) return (a + b - 1) / b;
  else return a / b;
}

ll intersect(Line a, Line b){
  return iceil(a.b - b.b, b.a - a.a);
}

void add(Line ln){
  while(!dq.empty
      () && ln(dq.back().l) >= dq.back()(dq.back().l)){
    dq.pob;
  }
  if(dq.empty()){
    dq.eb(ln);
    return;
  }
  ll pos = intersect(ln, dq.back());
  if(pos > dq.back().r){
    if(dq.back().r != MAX){
      ln.l = dq.back().r + 1;
      dq.eb(ln);
    }
    return;
  }
  dq.back().r = pos - 1;
  ln.l = pos;
  dq.eb(ln);
 }

ll query(ll x){
  while(dq.front().r < x) dq.pof;
  return dq.front()(x);
}
```

8.5 Partial Ordering

```
// O(n log^2 n)
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1e5 + 5, M = 2e5 + 5;
int n, K, cnt, ans[N];
struct node {
  int x, y, z, v, ans, tag, id;
  node() { ans = tag = v = x = y = z = 0; }
  friend
       bool operator==(const node &a, const node &b) {
    return
         (a.x == b.x) && (a.y == b.y) && (a.z == b.z);
  }
} a[N], t[N];
bool cmp1(const node &a, const node &b) {
  if (a.x != b.x) return a.x < b.x;
  if (a.y != b.y) return a.y < b.y;
  return a.z < b.z;
}
bool cmp2(const node &a, const node &b) {
  if (a.y != b.y) return a.y < b.y;
  if (a.tag != b.tag) return a.tag < b.tag;
  return a.id < b.id;
}
#define lowbit(x) (x & -x)
int bit[M];
void add(int p, int x) {
  for (; p <= K; p += lowbit(p)) bit[p] += x;
}
int query(int p) {
  int ret = 0;
  for (; p; p -= lowbit(p)) ret += bit[p];
  return ret;
}
void CDQ(int l, int r) {
  if (l == r) return;
  int mid = (l + r) >> 1;
  CDQ(l, mid); CDQ(mid + 1, r);
  for (int i = l; i <= r; ++i) a[i].id = i;
  for (int i = l; i <= mid; ++i) a[i].tag = 0;
  for (int i = mid + 1; i <= r; ++i) a[i].tag = 1;
  sort(a + l, a + r + 1, cmp2);
  for (int i = l; i <= r; ++i) {
    if (!a[i].tag) add(a[i].z, a[i].v);
    else a[i].ans += query(a[i].z);
  }
  for (int i = l; i <= r; ++i)
```

```
    if (!a[i].tag) add(a[i].z, -a[i].v);
}
int main() {
  cin >> n >> K;
  for (int i = 1; i <= n; ++
      i) cin >> a[i].x >> a[i].y >> a[i].z, a[i].v = 1;
  sort(a + 1, a + n + 1, cmp1);
  cnt = 1;
  for (int i = 2; i <= n; ++i) {
    if (a[i] == a[cnt]) ++a[cnt].v;
    else a[++cnt] = a[i];
  }
  CDQ(1, cnt);
  // let ans[i] denote that the
      number of (aj<=ai && bj<=bi && cj<=ci) for i != j
  for (int i = 1; i <=
       cnt; ++i) ans[a[i].ans + a[i].v - 1] += a[i].v;
  for (int i = 0; i < n; ++i) cout << ans[i] << '\n';
  return 0;
}
```

8.6 Xor Basis

```
int basis[20]
bool add(int x) {
    for (int i = 19; i >= 0; i--) {
        if (!(x >> i & 1)) continue;
        if (!basis[i]) {
            basis[i] = x;
            return true;
        }
        else x ^= basis[i];
    }
    return false;
}
// 維持 basis[i] 的最高□是 i
```