ACM ICPC Team Reference - Black flames

Contents

1 basic

1.1 fast

```
#pragma GCC optimize("Ofast,inline,unroll-loops")
#pragma GCC target("bmi,bmi2,lzcnt,popcnt,avx2")
#include <stdio.h>
#define windows_system
#ifdef windows_system
/* For Windows */
inline int getchar_unlocked() { return getchar(); }
inline void putchar_unlocked(char _c) { putchar(_c); }
#endif
inline int in() {
  int re = 0;
  char c = getchar_unlocked();
  while (c == ' ' || c == '\n') c = getchar_unlocked();
  while (c >= '0' && c <= '9') re = (re << 3) + (re << 1) + c - '0', c = getchar_unlocked();
  return re;
}
inline void out(int x) {
  char str[20];
  int pos = 0;
  do {
    str[pos++] = x % 10 + '0';
    x /= 10;
  } while (x);
  for (int i = pos - 1; i >= 0; i--) putchar_unlocked(str[i]);
  putchar_unlocked('\n');
}
```

1.2 random

```
mt19937 seed(chrono::steady_clock::now().time_since_epoch().count());
inline int rnd(int l, int r) {
  return uniform_int_distribution<int>(l, r)(seed);
}
inline double drnd(double l, double r) {
  return uniform_real_distribution<double>(l, r)(seed);
}
```

1.3 debug

```
void debug() { cout << endl; }
template <typename T, typename ...U>
void debug(T i, U ...j) { cout << i << ' ', debug(j...); }
#define test(x...) debug("[" + string(x) + "]", x)
```

2 data structure

2.1 sparse table

```
const int N = 100005;
int a[N];
struct Sparse_table {
    int n;
    vector<vector<int>> st;
    Sparse_table(int _n) : n(_n), st(_n + 1, vector<int>(__lg(_n) + 1)) {
        for (int i = 1; i <= n; i++) st[i][0] = a[i];
        for (int lg = 1; lg <= __lg(n); lg++) {
            int len = 1 << (lg - 1);
            for (int i = 1; i + len <= n; i++)
                st[i][lg] = max(st[i][lg - 1], st[i + len][lg - 1]);
        }
    }
    int query(int l, int r) {
        int lg = __lg(r - l + 1);
        int len = 1 << lg;
        return max(st[l][lg], st[r - len + 1][lg]);
    }
};
```

2.2 range set

```
struct RangeSet { // [l, r)
  set<pii> st;
  void cut(int x) {
    auto it = st.lower_bound({x + 1, -1});
    if (it == st.begin()) return;
    auto [l, r] = *prev(it);
    if (l >= x || x >= r) return;
    st.erase(prev(it));
    st.insert({l, x});
    st.insert({x, r});
  }
  vector<pii> split(int l, int r) {
    // remove and return ranges in [l, r)
    cut(l), cut(r);
    vector<pii> res;
    while (true) {
      auto it = st.lower_bound({l, -1});
      if (it == st.end() || r <= it->first) break;
      res.push_back(*it), st.erase(it);
    }
    return res;
  }
  void insert(int l, int r) {
    // add a range [l, r), [l, r) not in st
    auto it = st.lower_bound({l, r});
    if (it != st.begin() && prev(it)->second == l)
      l = prev(it)->first, st.erase(prev(it));
    if (it != st.end() && r == it->first)
      r = it->second, st.erase(it);
    st.insert({l, r});
  }
  bool count(int x) {
    auto it = st.lower_bound({x + 1, -1});
    return it != st.begin() && prev(it)->first <= x && x < prev(it)->second;
  }
};
```

2.3 pbds

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define int long long
using namespace std;
void solve() {
  tree<int, null_type, greater<int>, rb_tree_tag, tree_order_statistics_node_update> bst;
  bst.insert(1);
  int a = *bst.find_by_order(2); /// 第 k + 1 大的數
  int b = bst.order_of_key(2);   /// 有幾個數 > k
  // ----------------------------------------------------
  tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> bst;
  bst.insert(1);
  int a = *bst.find_by_order(2); /// 第 k + 1 小的數
  int b = bst.order_of_key(2);   /// 有幾個數 < k
}
```

2.4 treap

```
struct node {
  int data, sz;
  node *l, *r;
  node(int k) : data(k), sz(1), l(0), r(0) {}
  void up() {
    sz = 1;
    if (l) sz += l->sz;
    if (r) sz += r->sz;
  }
  void down() {}
};
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
  if (!a || !b) return a ? a : b;
  if (rand() % (sz(a) + sz(b)) < sz(a))
    return a->down(), a->r = merge(a->r, b), a->up(), a;
  return b->down(), b->l = merge(a, b->l), b->up(), b;
}
void split1(node *o, node *&a, node *&b, int k) {
  if (!o) return a = b = 0, void();
  o->down();
  if (o->data <= k)
    a = o, split1(o->r, a->r, b, k), a->up();
  else
    b = o, split1(o->l, a, b->l, k), b->up();
}
void split2(node *o, node *&a, node *&b, int k) {
  if (sz(o) <= k) return a = o, b = 0, void();
  o->down();
  if (sz(o->l) + 1 <= k)
    a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
  else
    b = o, split2(o->l, a, b->l, k);
  o->up();
}
node *kth(node *o, int k) {
  if (k <= sz(o->l)) return kth(o->l, k);
  if (k == sz(o->l) + 1) return o;
  return kth(o->r, k - sz(o->l) - 1);
}
int Rank(node *o, int key) {
  if (!o) return 0;
  if (o->data < key)
    return sz(o->l) + 1 + Rank(o->r, key);
  else
    return Rank(o->l, key);
}
bool erase(node *&o, int k) {
  if (!o) return 0;
  if (o->data == k) {
    node *t = o;
    o->down(), o = merge(o->l, o->r);
    delete t;
    return 1;
  }
  node *&t = k < o->data ? o->l : o->r;
  return erase(t, k) ? o->up(), 1 : 0;
}
void insert(node *&o, int k) {
  node *a, *b;
  split(o, a, b, k),
    o = merge(a, merge(new node(k), b));
}
void interval(node *&o, int l, int r) {
  node *a, *b, *c;
  split2(o, a, b, l - 1), split2(b, b, c, r);
  // operate
  o = merge(a, merge(b, c));
}
```

2.5 bitset

```
#pragma GCC target("popcnt")
/// 15 = 1111
bitset <10> bit(15); /// 宣告 長度 = 10 的 bitset
cout << bit.count() << "\n"; /// 回傳 幾個 1
cout << bit.none() << "\n";  /// 回傳是否全為 0
cout << bit.any() << "\n";   /// 回傳是否有 1
cout << bit._Find_first() << "\n"; /// 回傳 第一個 bit = 1 的位置
int x = 2;
cout << bit._Find_next(x) << "\n"; /// 回傳 x 後 第一個 bit = 1 的位置
```

3 graph

3.1 boruvka

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

using Graph = vector<vector<int>>;

struct DSU {
    int cc;
    vector<int> par, sz;
    vector<set<int>> S;

    DSU(int n = 0) : cc(n), par(n), sz(n, 1), S(n) {
        for (int i = 0; i < n; i++) {
            par[i] = i;
        }
    }
    int find(int x) {
        if (par[x] == x) return x;
        return par[x] = find(par[x]);
    }
    bool merge(int u, int v) {
        u = find(u), v = find(v);
        if (u == v) return false;
        if (sz[u] < sz[v]) swap(u, v);
        par[v] = u;
        sz[u] += sz[v];
        for (int x : S[v]) {
            S[u].insert(x);
        }
        S[v].clear();
        cc--;
        return true;
    }
};

struct Edge {
    int u, v;
    int cost;
};

bool operator<(const Edge &a, const Edge &b) {
    return a.cost < b.cost;
}

const int INF = 2e18;

// 對於目前選到的每個集合，選他周圍的最小邊
int MST(int n, vector<Edge> edges) {
    int m = edges.size();

    DSU dsu(n);
    vector<Edge> nei(n);

    int mst_ans = 0;

    int conti = true;
    while (conti) {
        conti = false;
        fill(nei.begin(), nei.end(), Edge{-1, -1, INF});

        for (auto [u, v, cost] : edges) {
            int fu = dsu.find(u), fv = dsu.find(v);
            if (fu == fv) continue;

            nei[fu] = min(nei[fu], {u, v, cost});
            nei[fv] = min(nei[fv], {v, u, cost});
        }

        for (int i = 0; i < n; i++) {
            auto e = nei[i];
            if (e.u == -1) continue;
            if (dsu.find(e.u) != dsu.find(e.v)) {
                dsu.merge (e.u, e.v);
                mst_ans += e.cost;
                conti = true;
            }
        }
    }

    return mst_ans;
}

signed main() {
    int n, m;
    cin >> n >> m;
    vector<Edge> edges;

    int u, v, w;
    for (int i = 0; i < m; i++) {
        cin >> u >> v >> w;
        u--, v--;
        edges.push_back({u, v, w});
    }

    cout << MST(n, edges) << "\n";
}
```

3.2 cut bcc

```
const int N = 200005;
vector <int> G[N];
int low[N], depth[N];
bool vis[N];
vector <vector <int>> bcc;
stack <int> stk;
void dfs(int v, int p) {
  stk.push(v);
  vis[v] = true;
  low[v] = depth[v] = (p == -1 ? 1 : depth[p] + 1);
  for (int u : G[v]) {
    if (u == p) continue;
    if (!vis[u]) {
      /// (v, u) 是樹邊
      dfs(u, v);
      low[v] = min(low[v], low[u]);
      /// u 無法在不經過父邊的情況走到 v 的祖先
      if (low[u] >= depth[v]) {
        bcc.emplace_back();
        while (stk.top() != u) {
          bcc.back().push_back(stk.top());
          stk.pop();
        }
        bcc.back().push_back(stk.top());
        stk.pop();
        bcc.back().push_back(v);
      }
    } else {
      /// (v, u) 是回邊
      low[v] = min(low[v], depth[u]);
    }
  }
}
```

3.3 spfa

```
bool SPFA(int s) {
    vector<int> dis(n, INF);
    vector<bool> inq(n);
    vector<int> cnt(n);

    queue<int> q;
    q.push(s);
    dis[s] = 0;
    inq[s] = true;

    while (q.size()) {
        int u = q.front();
        q.pop();
        cnt[u]++;

        if (cnt[u] == n) {
            // negative cycle
            return true;
        }

        inq[u] = false;

        for (auto [v, w] : G[u]) {
            if (dis[u] + w < dis[v]) {
                dis[v] = dis[u] + w;

                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }

    return false;
}
```

3.4 tarjan

```
const int N = 5e5 + 5;
int n, m, stamp;
vector<int> G[N];
int dfn[N], low[N];
vector<vector<int>> bcc;
stack<int> stk;

void dfs (int u, int par) {
    dfn[u] = low[u] = ++stamp;
    stk.push(u);
    int cnt = 0; // 兒子個數
    for (auto v : G[u]) {
        if (v == par) continue;
        if (!dfn[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            cnt++;
            if (low[v] >= dfn[u]) { // 若 u 為割點
                int now = 0;
                bcc.push_back({});
                do {
                    now = stk.top();
                    stk.pop();
                    bcc.back().push_back(now);
                } while (now != v);
                bcc.back().push_back(u);
            }
        } else {
            low[u] = min(low[u], dfn[v]);
        }
    }
    // 特判孤立點
    if (par == 0 && cnt == 0) {
        bcc.push_back({u});
        return;
    }
}
```

3.5 cut

```
const int N = 200005;
vector <int> G[N];
int low[N], depth[N];
bool vis[N], cut[N];
void dfs(int v, int p) {
  vis[v] = true;
  int child = 0;
  low[v] = depth[v] = (p == -1 ? 1 : depth[p] + 1);
  for (int u : G[v]) {
    if (u == p) continue;
    if (!vis[u]) {
      /// (v, u) 是樹邊
      dfs(u, v);
      child++;
      low[v] = min(low[v], low[u]);
      /// u 無法在不經過父邊的情況走到 v 的祖先
      if (low[u] >= depth[v] && p != -1)
        cut[v] = true;
    } else {
      /// (v, u) 是回邊
      low[v] = min(low[v], depth[u]);
    }
  }
  /// 根節點有超過 2 個子節點
  if (p == -1 && child >= 2)
    cut[v] = true;
}
```

3.6 2SAT

```
struct TWO_SAT {
  int n, N;
  vector<vector<int>> G, rev_G;
  deque<bool> used;
  vector<int> order, comp;
  deque<bool> assignment;
  void init(int _n) {
    n = _n;
    N = _n * 2;
    G.resize(N + 5);
    rev_G.resize(N + 5);
  }
  void dfs1(int v) {
    used[v] = true;
    for (int u : G[v]) {
        if (!used[u])
            dfs1(u);
    }
    order.push_back(v);
  }
  void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : rev_G[v]) {
        if (comp[u] == -1)
            dfs2(u, cl);
    }
  }
  bool solve() {
    order.clear();
    used.assign(N, false);
    for (int i = 0; i < N; ++i) {
        if (!used[i])
            dfs1(i);
    }
    comp.assign(N, -1);
    for (int i = 0, j = 0; i < N; ++i) {
        int v = order[N - i - 1];
        if (comp[v] == -1)
            dfs2(v, j++);
    }
    assignment.assign(n, false);
    for (int i = 0; i < N; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        assignment[i / 2] = (comp[i] > comp[i + 1]);
    }
    return true;
  }
  void add_disjunction(int a, bool na, int b, bool nb) { // A or B
    // na means whether a is negative or not
    // nb means whether b is negative or not
    a = 2 * a ^ na;
    b = 2 * b ^ nb;
    int neg_a = a ^ 1;
    int neg_b = b ^ 1;
    G[neg_a].push_back(b);
    G[neg_b].push_back(a);
    rev_G[b].push_back(neg_a);
    rev_G[a].push_back(neg_b);
    return;
  }
  void get_result(vector<int>& res) {
    res.clear();
    for (int i = 0; i < n; i++)
        res.push_back(assignment[i]);
  }
};
/* CSES Giant Pizza
3 5
+ 1 + 2
- 1 + 3
+ 4 - 2
- + + + -
*/
int main() {
  int n, m;
  cin >> n >> m;
  TWO_SAT E;
  E.init(m);
  char c1, c2;
  int inp1, inp2;
  for (int i = 0; i < n; i++) {
    cin >> c1 >> inp1;
    cin >> c2 >> inp2;
    E.add_disjunction(inp1 - 1, c1 == '-', inp2 - 1, c2 == '-');
  }
  bool able = E.solve();
  if (able) {
    vector <int> ans;
    E.get_result(ans);
    for (int i : ans)
      cout << (i == true ? '+' : '-') << ' ';
    cout << '\n';
  } else {
    cout << "IMPOSSIBLE\n";
  }
  return 0;
}
```

3.7 bridge

```
const int N = 200005;
vector <int> G[N];
int low[N], depth[N];
bool vis[N];
vector <pair <int, int>> bridge;
void dfs(int v, int p) {
    vis[v] = true;
    low[v] = depth[v] = (p == -1 ? 1 : depth[p] + 1);
    for (int u : G[v]) {
        if (u == p) continue;
        if (!vis[u]) {
            /// (v, u) 是樹邊
            dfs(u, v);
            low[v] = min(low[v], low[u]);
        } else {
            /// (v, u) 是回邊
            low[v] = min(low[v], depth[u]);
        }
    }
    /// v 在不依靠父邊的情況下永遠沒辦法走到它的祖先
    /// (root, -1) 不算
    if (low[v] == depth[v] && p != -1)
        bridge.push_back({v, p});
}
```

3.8 prim

```
const int N = 2005;
int dis[N][N]; // input
int distfromtree[N];

for (int i = 1; i <= n; i++)
  distfromtree[i] = dis[1][i];
int res = 0;
for (int i = 2; i <= n; i++) {
  int mindist = INT_MAX, next = -1;
  for (int j = 1; j <= n; j++) {
    if (distfromtree[j] != 0 && distfromtree[j] < mindist) {
      mindist = distfromtree[j];
      next = j;
    }
  }
  res += mindist;
  for (int j = 1; j <= n; j++)
    distfromtree[j] = min(distfromtree[j], dis[next][j]);
}
```

3.9 euler-cycle

```
void dfs(int u) {
    while(G[u].size()) {
        auto [v, eid] = G[u].back();
        G[u].pop_back();

        if (vis[eid]) continue;

        vis[eid] = 1;
        dfs(v);
        ans.pb(id);
    }
}
```

3.10 bridge bcc

```
const int N = 200005;
vector <int> G[N];
int low[N], depth[N];
bool vis[N];
vector <vector <int>> bcc;
stack <int> stk;
void dfs(int v, int p) {
  stk.push(v);
  vis[v] = true;
  low[v] = depth[v] = (p == -1 ? 1 : depth[p] + 1);
  for (int u : G[v]) {
    if (u == p) continue;
    if (!vis[u]) {
      /// (v, u) 是樹邊
      dfs(u, v);
      low[v] = min(low[v], low[u]);
    } else {
      /// (v, u) 是回邊
      low[v] = min(low[v], depth[u]);
    }
  }
  /// v 在不依靠父邊的情況下永遠沒辦法走到它的祖先
  if (low[v] == depth[v]) {
    bcc.emplace_back();
    while (stk.top() != v) {
      bcc.back().push_back(stk.top());
      stk.pop();
    }
    bcc.back().push_back(stk.top());
    stk.pop();
  }
}
```

4 math

4.1 primes er

```
// Time Complexity : O(nloglogn)
const int N = 200005;
bool is_prime[N];
vector <int> primes;
void build() {
  fill(is_prime, is_prime + N, true);
  is_prime[0] = is_prime[1] = false;
  for (int i = 2; i <= 200000; i++) {
    if (is_prime[i]) {
      primes.push_back(i);
      if (1ll * i * i <= 200000)
        for (int j = i * i; j <= 200000; j += i)
          is_prime[j] = false;
    }
  }
}
```

4.2 factorize

```
using i64 = long long;
using i128 = __int128_t;
struct Factorize { // 質因數分解
  i64 fmul(i64 a, i64 b, i64 p) {
    return (i128)a * b % p;
  }
  i64 fpow(i64 a, i64 b, i64 p) {
    i64 res = 1;
    for (; b; b >>= 1, a = fmul(a, a, p))
```

```
      if (b & 1) res = fmul(res, a, p);
    return res;
  }
  bool check(i64 a, i64 u, i64 n, int t) {
    a = fpow(a, u, n);
    if (a == 0 or a == 1 or a == n - 1)
      return true;
    for (int i = 0; i < t; i++) {
      a = fmul(a, a, n);
      if (a == 1) return false;
      if (a == n - 1) return true;
    }
    return false;
  };
  bool isPrime(i64 n) {
    constexpr array<i64, 7> magic{2, 235, 9375, 28178, 450775, 9780504, 1795265022};
    // for int: {2, 7, 61}
    if (n < 2) return false;
    if (n % 2 == 0) return n == 2;
    i64 u = n - 1;
    int t = 0;
    while (u % 2 == 0) u >>= 1, t++;
    for (auto v : magic) if (!check(v, u, n, t)) return false;
    return true;
  }
  i64 PollardRho(i64 n) { // return non-trivial factor of n
    if (n % 2 == 0) return 2;
    i64 x = 2, y = 2, d = 1, p = 1;
    auto f = [](i64 x, i64 n, i64 p) -> i64 {
      return ((i128)x * x % n + p) % n;
    };
    while (true) {
      x = f(x, n, p);
      y = f(f(y, n, p), n, p);
      d = gcd(abs(x - y), n);
      if (d != n and d != 1) return d;
      if (d == n) ++p;
    }
  }
  i64 primeFactor(i64 n) {
    return isPrime(n) ? n : primeFactor(PollardRho(n));
  }
} ftr;
void solve() {
  i64 n;
  cin >> n;
  vector<i64> ans;
  while (n > 1) {
    i64 p = ftr.primeFactor(n);
    do {
      ans.push_back(p);
      n /= p;
    } while (n % p == 0);
  }
  sort(ans.begin(), ans.end());
  cout << ans.size() << ' ';
  for (int i : ans) cout << i << ' ';
  cout << '\n';
}
```

4.3 fraction

```
struct frac {
  int a, b;
  frac(int _a = 0, int _b = 1) : a(_a), b(_b) {
    int g = __gcd(a, b);
    a /= g; b /= g;
    if (b < 0) { a *= -1; b *= -1; }
  }
};
frac operator+(frac x, frac y) { return frac(x.a * y.b + y.a * x.b, x.b * y.b); }
frac operator-(frac x, frac y) { return frac(x.a * y.b - y.a * x.b, x.b * y.b); }
frac operator*(frac x, frac y) { return frac(x.a * y.a, x.b * y.b); }
frac operator/(frac x, frac y) { return frac(x.a * y.b, x.b * y.a); }
bool operator>(frac x, frac y) { return x.a * y.b > y.a * x.b; }
bool operator<(frac x, frac y) { return x.a * y.b < y.a * x.b; }
bool operator>=(frac x, frac y) { return x.a * y.b >= y.a * x.b; }
bool operator<=(frac x, frac y) { return x.a * y.b <= y.a * x.b; }
bool operator==(frac x, frac y) { return x.a * y.b == y.a * x.b; }
frac abs(frac x) { return frac(abs(x.a), abs(x.b)); }
ostream& operator<<(ostream &os, const frac& x) { os << x.a << "/" << x.b; return os; }
```

4.4 primes linear

```
// Time Complexity : O(n)
// if lpf(i) = i, then it means that i is a prime.
// else lpf(i) is the smallest prime factor.
// The 199999-th of the prime is 2750131
bool is_prime[2750135];
int lpf[2750135];
vector<int> primes;
void init() {
  fill(is_prime, is_prime + 2750135, true);
  for (int i = 2; i <= 2750131; i++) {
    if (is_prime[i]) {
      primes.push_back(i);
      lpf[i] = i;
    }
    for (int p : primes) {
      if (p * i > 2750131) break;
      is_prime[p * i] = false;
      lpf[p * i] = p;
      if (i % p == 0) break;
    }
  }
}
```

4.5 big number

```
void init(string &a, string &b) {
  while (a.size() < b.size()) a = '0' + a;
  while (a.size() > b.size()) b = '0' + b;
}
int Compare(string &a, string &b) {
  if (a < b) {
    swap(a, b);
    return -1;
  }
  return 1;
}
bool del(string &a) {
  if (a[0] == '0') {
    a.erase(0, 1);
    return true;
  }
  return false;
}
void del_all_zero(string &a) {
  while (del(a)) ;
}
string add(string a, string b) {
  init(a, b);
  a = '0' + a; b = '0' + b;
  for (int i = a.size() - 1; i >= 0; i--) {
    int p = a[i] - '0', q = b[i] - '0';
    if (p + q >= 10) {
      a[i - 1] = '0' + (a[i - 1] - '0' + 1);
      a[i] = '0' + (p + q - 10);
    } else {
      a[i] = '0' + (p + q);
    }
  }
  del(a);
  return a;
}
string sub(string a, string b) {
  init(a, b);
  if (a == b) return "0";
  int flag = Compare(a, b);
  for (int i = a.size() - 1; i >= 0; i--) {
    int p = a[i] - '0', q = b[i] - '0';
    if (p < q) {
      a[i - 1] = '0' + (a[i - 1] - '0' - 1);
      a[i] = '0' + (p - q + 10);
    } else {
      a[i] = '0' + (p - q);
    }
  }
  del_all_zero(a);
  if (flag == -1) a = "-" + a;
  return a;
}
string mul(string a, string b) {
  string res = "0";
  init(a, b);
  Compare(a, b);
  del_all_zero(b);
  for (int i = b.size() - 1; i >= 0; i--) {
    int x = b[i] - '0';
    if (i != b.size() - 1) a = a + '0';
    for (int i = 0; i < x; i++)
      res = add(a, res);
  }
  return res;
}
```

```cpp
string div(string a, string b) {
  init(a, b);
  if (a < b) return "0";
  del_all_zero(b);
  string res = "0", restmp = "1", tmp = b;
  for (int i = 0; i < (a.size() - b.size()); i++) {
    restmp += '0';
    tmp += '0';
  }
  init(a, b);
  while (a >= b) {
    init(a, tmp);
    if (a >= tmp) {
      a = sub(a, tmp);
      res = add(res, restmp);
    } else {
      restmp.erase(restmp.size() - 1);
      tmp.erase(tmp.size() - 1);
    }
    init(a, b);
  }
  return res;
}
signed main() {
  string x, y, ch;
  cin >> x >> ch >> y;
  string ans;
  if (ch[0] == '+') ans = add(x, y);
  else if (ch[0] == '-') ans = sub(x, y);
  else if (ch[0] == '*') ans = mul(x, y);
  else ans = div(x, y);
  cout << ans << "\n";
  return 0;
}
```

4.6 miller rabin

```cpp
struct Miller_Rabin {
    static constexpr uint32_t ws32[3] = {2, 7, 61};
    static constexpr uint64_t ws64[7] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    template <class uint>
    static constexpr uint fast_pow(uint b, uint p, uint mod) {
        using Uint = conditional_t<is_same_v<uint, uint32_t>, uint64_t, __uint128_t>;
        uint ret{1};
        for (; p; p >>= 1) {
            if (p & 1) ret = Uint{ret} * b % mod;
            b = Uint{b} * b % mod;
        }
        return ret;
    }
    template <class uint, class = enable_if_t<is_unsigned_v<uint>>>
    static constexpr bool is_prime(uint n) {
        const auto& witness = []() -> const auto& {
            if constexpr (is_same_v<uint, uint32_t>) return ws32;
            else return ws64;
        }();
        using Uint = conditional_t<is_same_v<uint, uint32_t>, uint64_t, __uint128_t>;
        if (n < 3 || !(n & 1)) return n == 2;
        auto u = n - 1, t = (uint)__builtin_ctzll(u);
        u >>= t;
        for (auto x : witness) {
            auto a = x % n;
            if (a == 0 || a == 1 || a == n - 1) continue;
            auto v = fast_pow(a, u, n);
            uint i = 0;
            for (; v != 1 && v != n - 1 && i < t; i++) v = Uint{v} * v % n;
            if (v != n - 1 && i > 0) return false;
        }
        return true;
    }
    template <class I, class = enable_if_t<is_integral_v<I>>>
    constexpr bool operator()(I x) const {
        if constexpr (is_signed_v<I>) if (x < 0) return false;
        using U = make_unsigned_t<I>;
        return is_prime((U)x);
    }
} is_prime;
```

4.7 math block

```cpp
// the sum of floor(n / i) for i = 1 ~ n
int f(int n) {
  int ans = 0;
  int l = 1, r = 0;
  while (l <= n) {
    r = n / (n / l);
    ans += (r - l + 1) * (n / l);
    l = r + 1;
  }
  return ans;
}
// the sum of ceil(n / i) for i = 1 ~ n
int f(int n) {
  int ans = 0;
  int l = 1, r = 0;
  while (l < n) {
    r = (n - 1) / ((n - 1) / l);
    ans += (r - l + 1) * ((n + l - 1) / l);
    l = r + 1;
  }
  if (l == n) ans += (n + l - 1) / l;
  return ans;
}
```

4.8 count prime

```cpp
using namespace std;
// Count the number of primes not greater than n
int primeCount(const int n) {
  if (n <= 1) { return 0; }
  if (n == 2) { return 1; }
  const int v = sqrtl(n);
  int s = (v + 1) / 2;
  vector<int> smalls(s), roughs(s), skip(v + 1);
  vector<int> larges(s);
  iota(smalls.begin(), smalls.end(), 0);
  for (int i = 0; i < s; i++) {
    roughs[i] = 2 * i + 1;
    larges[i] = (n / roughs[i] - 1) / 2;
  }
  const auto half = [](int n) -> int { return (n - 1) >> 1; };
  int pc = 0;
  for (int p = 3; p <= v; p += 2) {
    if (skip[p]) { continue; }
    int q = p * p;
    if (1LL * q * q > n) { break; }
    skip[p] = true;
    for (int i = q; i <= v; i += 2 * p) skip[i] = true;
    int ns = 0;
    for (int k = 0; k < s; k++) {
      int i = roughs[k];
      if (skip[i]) { continue; }
      int d = 1LL * i * p;
      larges[ns] = larges[k] - (d <= v ? larges[smalls[d / 2] - pc] : smalls[half(n / d)]) + pc;
      roughs[ns++] = i;
    }
    s = ns;
    for (int i = half(v), j = v / p - 1 | 1; j >= p; j -= 2) {
      int c = smalls[j / 2] - pc;
      for (int e = j * p / 2; i >= e; i--) { smalls[i] -= c; }
    }
    pc++;
  }
  larges[0] += 1LL * (s + 2 * (pc - 1)) * (s - 1) / 2;
  for (int k = 1; k < s; k++) { larges[0] -= larges[k]; }
  for (int l = 1; l < s; l++) {
    int q = roughs[l];
    int M = n / q;
    int e = smalls[half(M / q)] - pc;
    if (e <= l) { break; }
    int t = 0;
    for (int k = l + 1; k <= e; k++) { t += smalls[half(M / roughs[k])]; }
    larges[0] += t - 1LL * (e - l) * (pc + l - 1);
  }
  return larges[0] + 1;
}
```

5 other

5.1 Geometry

```cpp
double eps = 0.000001;
pdd operator + (pdd a, pdd b) { return {a.X + b.X, a.Y + b.Y}; }
pdd operator - (pdd a, pdd b) { return {a.X - b.X, a.Y - b.Y}; }
double dot(pdd a, pdd b) { return a.X * b.X + a.Y * b.Y; }
double cross(pdd a, pdd b) { return a.X * b.Y - a.Y * b.X; }
int sign(double x) { return (fabs(x) < eps ? 0 : (x > 0 ? 1 : -1)); }
int ori(pdd a, pdd b, pdd c) { return sign(cross(b - a, c - a)); }
/// c is between a and b
```

```cpp
bool btw(pdd a, pdd b, pdd c) {
  return (sign(cross(a - c, b - c)) == 0 && sign(dot(a - c, b - c)) <= 0);
}
/// ab is touch cd
bool banana(pdd a, pdd b, pdd c, pdd d) {
  int a123 = ori(a, b, c), a124 = ori(a, b, d);
  int a341 = ori(c, d, a), a342 = ori(c, d, b);
  if (btw(a, b, c) || btw(a, b, d) || btw(c, d, a) || btw(c, d, b))
    return true;
  return (a123 * a124 < 0 && a341 * a342 < 0);
}
vector <pdd> convex_hull(vector <pdd> pts) {
  sort(pts.begin(), pts.end());
  pts.resize(unique(pts.begin(), pts.end()) - pts.begin());
  vector <pdd> hull(1, pts[0]);
  for (int t = 0; t < 2; t++) {
    int sz = hull.size();
    for (int i = 1; i < (int)pts.size(); i++) {
      while ((int)hull.size() > sz && ori(hull[(int)hull.size() - 2], hull.back(), pts[i]) < 0)
        hull.pop_back();
      hull.push_back(pts[i]);
    }
    reverse(pts.begin(), pts.end());
  }
  hull.pop_back();
  return hull;
}
double area(vector <pdd> pt) {
  double cnt = 0.0;
  for (int i = 0; i < (int)pt.size() - 1; i++)
    cnt += cross(pt[i], pt[i + 1]);
  cnt += cross(pt[(int)pt.size() - 1], pt[0]);
  return fabs(cnt);
}
void solve() {
  int n;
  cin >> n;
  vector <pdd> pts;
  pts.resize(n);
  for (int i = 0; i < n; i++)
    cin >> pts[i].X >> pts[i].Y;
  int ans = area(convex_hull(pts));
  if (ans % 2 == 1)
    cout << ans / 2 << ".5\n";
  else
    cout << ans / 2 << ".0\n";
}
```

5.2 subset

```cpp
for (int i = 0 ; i < (1 << n); i++){
  for (int m = i; m; m = (m - 1) & i){
    // m 是 i 的非空子集
  }
}
```

5.3 digit dp

```cpp
// 原題：計算 [l, r] 中兩兩相鄰數字不同的數字數量
// 注意邊界
int dp[20][10]; // i 位數，尾數是 j
void init() {
  for (int j = 0; j <= 9; j++) dp[1][j] = 1;
  for (int i = 2; i <= 18; i++)
    for (int now = 0; now <= 9; now++) // 這個位數的數字
      for (int pre = 0; pre <= 9; pre++) // 上個位數的數字
        if (now != pre)
          dp[i][now] += dp[i - 1][pre];
}
int calc(int x) { // 小於等於 x 的數量
  if (x == 0) return 0;
  int arr[20] = {}, len = 0;
  while (x > 0) arr[++len] = x % 10, x /= 10;
  // 計算 len 位數 的數量
  int cnt = 0, pre = -1;
  for (int idx = len; idx >= 1; idx--) {
    for (int now = (idx == len); now < arr[idx]; now++)
      if (now != pre)
        cnt += dp[idx][now];
    if (arr[idx] == pre) break;
    pre = arr[idx];
    if (idx == 1) cnt++; // 特判
  }
  // 計算 小於 len 位數 的數量
  for (int i = 1; i < len; i++)
    for (int j = 1; j <= 9; j++)
      cnt += dp[i][j];
  return cnt;
}
```

5.4 mcmf

```cpp
struct Flow {
  struct Edge {
    int u, rc, k, rv;
  };
  vector<vector<Edge>> G;
  vector<int> par, par_eid;
  Flow(int n) : G(n + 1), par(n + 1), par_eid(n + 1) {}
  void add(int v, int u, int c, int k) {
    G[v].push_back({u, c, k, (int)G[u].size()});
    G[u].push_back({v, 0, -k, (int)G[v].size() - 1});
  }
  int spfa(int s, int t) {
    fill(par.begin(), par.end(), -1);
    vector<int> dis(par.size(), LONG_LONG_MAX);
    vector<bool> in_q(par.size(), false);
    queue<int> Q;
    dis[s] = 0; in_q[s] = true;
    Q.push(s);
    while (! Q.empty()) {
      int v = Q.front(); Q.pop();
      in_q[v] = false;
      for (int i = 0; i < (int)G[v].size(); i++) {
        auto [u, rc, k, rv] = G[v][i];
        if (rc > 0 && dis[v] + k < dis[u]) {
          dis[u] = dis[v] + k;
          par[u] = v;
          par_eid[u] = i;
          if (! in_q[u]) Q.push(u);
          in_q[u] = true;
        }
      }
    }
    return dis[t];
  }
  pair<int, int> flow(int s, int t) {
    int fl = 0, cost = 0, d;
    while ((d = spfa(s, t)) < LONG_LONG_MAX) {
      int cur = LONG_LONG_MAX;
      for (int v = t; v != s; v = par[v])
        cur = min(cur, G[par[v]][par_eid[v]].rc);
      fl += cur;
      cost += d * cur;
      for (int v = t; v != s; v = par[v]) {
        G[par[v]][par_eid[v]].rc -= cur;
        G[v][G[par[v]][par_eid[v]].rv].rc += cur;
      }
    }
    return {fl, cost};
  }
};
```

5.5 dinic

```cpp
struct Flow {
  struct Edge {
    // rc : residual capcity
    int u, rc, rev;
  };
  vector<vector<Edge>> G;
  vector<int> dis, it;
  Flow(int n) : G(n), dis(n), it(n) {}
  void add(int v, int u, int c) {
    G[v].push_back({u, c, sz(G[u])});
    G[u].push_back({v, 0, sz(G[v]) - 1});
  }
  int dfs(int v, int t, int f) {
    if (v == t || f == 0) return f;
    for (int &i = it[v]; i < sz(G[v]); i++) {
      auto &[u, rc, rev] = G[v][i];
      if (dis[u] != dis[v] + 1) continue;
      int res = dfs(u, t, min(f, rc));
      if (res <= 0) continue;
      rc -= res;
      G[u][rev].rc += res;
      return res;
    }
    return 0;
  }
  int flow(int s, int t) {
    int ans = 0;
    for (int l = 30; l >= 0; l--) while (true) {
      fill(all(dis), INT_MAX);
      queue<int> Q;
      Q.push(s);
      dis[s] = 0;
      while (! Q.empty()) {
        int v = Q.front(); Q.pop();
        for (auto [u, rc, rev] : G[v]) {
          if ((rc >> 1) <= 0 || dis[u] < INT_MAX) continue;
          dis[u] = dis[v] + 1;
          Q.push(u);
        }
      }
      if (dis[t] == INT_MAX) break;

      fill(all(it), 0);
      while (true) {
        int res = dfs(s, t, INT_MAX);
        if (res <= 0) break;
        ans += res;
      }
    }
    return ans;
  }
};
```

6 string

6.1 manacher

```cpp
string s;
int extend(int l, int r, int N) {
  int i = 0;
  while (l - i >= 0 && r + i < N && s[l - i] == s[r + i])
    i++;
  return i;
}
int Longest_Palindromic_Substring(string &t) {
  int N = t.length();
  s.resize(2 * N + 1, '$');
  for (int i = 0; i < N; i++)
    s[2 * i + 1] = t[i];
  N = 2 * N + 1;
  vector <int> res;
  res.resize(N, 0);
  res[0] = 1;
  for (int i = 1, mid = 0, R = 0; i < N; i++) {
    int j = mid - (i - mid);
    int lst = R + 1 - i;
    if (i > R) {
      res[i] = extend(i, i, N);
      mid = i;
      R = i + res[i] - 1;
    } else if (res[j] == lst) {
      res[i] = lst + extend(i - R, i + R, N);
      mid = i;
      R = i + res[i] - 1;
    } else {
      res[i] = min(res[j], lst);
    }
  }
  int mx = -1, idx = -1;
  for (int i = 0; i < N; i++) {
    if (res[i] > mx) {
      mx = res[i];
      idx = i;
    }
  }
  cout << "Longest Palindromic Substring Length is " << (mx + 1) / 2 << "\n";
  for (int i = idx - res[idx] + 1; i <= idx + res[idx] + 1; i++)
    if (i & 1)
      cout << s[i];
  cout << "\n";
}
```

6.2 string trie

```cpp
struct node {
  node* link[26] = {nullptr};
  bool flag = false;
  bool contain_key(char ch) { return (link[ch - 'a'] != nullptr); }
  void put(char ch, node* _node) { link[ch - 'a'] = _node; }
  node* get(char ch) { return link[ch - 'a']; }
  void set_end() { flag = true; }
  bool is_end() { return flag; }
};
struct Trie {
  node* root;
  Trie() { root = new node(); }
  void insert(string s) { /// insert a string
    node* now = root;
    for (int i = 0; i < (int)s.size(); i++) {
      if (!now->contain_key(s[i])) now->put(s[i], new node());
      now = now->get(s[i]);
    }
    now->set_end();
  }
  bool query(string s) {  /// return if there is a word of the given string
    node* now = root;
    for (int i = 0; i < (int)s.size(); i++) {
      if (!now->contain_key(s[i])) return false;
      now = now->get(s[i]);
    }
    return now->is_end();
  }
  /// return if there is any word that start with the given prefix
  bool query_prefix(string s) {
    node* now = root;
    for (int i = 0; i < (int)s.size(); i++) {
      if (!now->contain_key(s[i])) return false;
      now = now->get(s[i]);
    }
    return true;
  }
};
```

6.3 kmp

```cpp
vector <int> F;
void build_failure_function(string &s) {
    F.clear();
    F.resize(s.size(), -1);
    for (int i = 1; i < s.size(); i++) {
        int j = F[i - 1];
        while (j != -1 && s[i] != s[j + 1])
            j = F[j];
        if (s[i] == s[j + 1])
            F[i] = j + 1;
    }
}
void KMP_matching(string &a, string &b) {
    /// i -> a 的指針， j -> b 的指針
    for (int i = 0, j = -1; i < a.size(); i++) {
        while (j != -1 && a[i] != b[j + 1]) /// 匹配失敗
            j = F[j];
        if (a[i] == b[j + 1]) /// 匹配成功
            j++;
        if (j + 1 == b.size()) { /// 找到了，當作匹配失敗 重新匹配 找下一個
            cout << "found a matching start at " << i - j << "\n";
            j = F[j];
        }
    }
}
```

6.4 hash

```cpp
const int p = 1e6 + 3;
const int mod = 1e9 + 7;
int hash(string &s) {
  int res = 0;
  for (int i = 0; i < s.size(); i++) {
    res *= p;
    res += s[i] - 'a' + 1;
    res %= mod;
  }
  return res;
}
int pow_p[1000005];
vector <int> rh;
void build(string &s) {
  rh.resize(s.size() + 1);
  pow_p[0] = 1;
  for (int i = 1; i <= s.size(); i++)
    pow_p[i] = (pow_p[i - 1] * p) % mod;
  rh[0] = s[0] - 'a' + 1;
  for (int i = 1; i < s.size(); i++) {
    rh[i] = rh[i - 1] * p + (s[i] - 'a' + 1);
    rh[i] %= mod;
  }
}
int query(int l, int r) {
  int res = rh[r] - (l > 0 ? rh[l - 1] * pow_p[r - l + 1] : 0);
  res = (res % mod + mod) % mod;
  return res;
}
```

6.5 xor trie

```
struct node {
  node* link[2];
  bool contain_key(int val) { return (link[val] != nullptr); }
  node* get(int val) { return link[val]; }
  void put(int val, node* _node) { link[val] = _node; }
};
struct Trie {
  node* root;
  Trie() { root = new node(); }
  void update(int x) {
    node* now = root;
    for (int i = 30; i >= 0; i--) {
      if (x & (1 << i)) {
        if (!now->contain_key(1)) now->put(1, new node());
        now = now->get(1);
      } else {
        if (!now->contain_key(0)) now->put(0, new node());
        now = now->get(0);
      }
    }
  }
  int query(int x) { /// query MAX XOR with number x
    node* now = root;
    int mx = 0;
    for (int i = 30; i >= 0; i--) {
      int nowbit = (x >> i) & 1;
      int target = nowbit ^ 1;
      if (now->contain_key(target)) {
        mx += (1 << i);
        now = now->get(target);
      } else {
        now = now->get(nowbit);
      }
    }
    return mx;
  }
};
```

6.6 z

```
vector <int> Z;
void calculate_Z(string &s) {
  Z.clear();
  Z.resize(s.size(), 0);
  int l = 0, r = 0;
  for (int i = 1; i < s.size(); i++) {
    if (i <= r) /// 估算下界
      Z[i] = min(Z[i - l], r - i + 1);
    while (i + Z[i] < s.size() && s[i + Z[i]] == s[Z[i]]) ///
         暴力檢查 z(i) 是否可以變更大
      Z[i]++;
    if (i + Z[i] - 1 > r) { /// 更新 "看到最右邊的區間 [l, r]"
      l = i; r = i + Z[i] - 1;
    }
  }
}
void Z_matching(string &a, string &b) {
  string res = "";
  int i = 0;
  for (char ch : b)
    res[i++] = ch;
  res[i++] = '_';
  for (char ch : a)
    res[i++] = ch;
  calculate_Z(res);
  for (int i = 0; i < Z.size(); i++) {
    if (Z[i] == b.size())
      cout << "found a matching start at " << i - b.size() - 1
           << "\n";
  }
}
```

7 tree

7.1 hld

```
const int N = 100005;
vector <int> G[N];
struct HLD {
  vector<int> pa, sz, depth, mxson, topf, id;
  int n, idcnt = 0;
  HLD(int _n) : n(_n), pa(_n + 1), sz(_n + 1), depth(_n + 1),
       mxson(_n + 1), topf(_n + 1), id(_n + 1) {}
  void dfs1(int v = 1, int p = -1) {
    pa[v] = p; sz[v] = 1; mxson[v] = 0;
    depth[v] = (p == -1 ? 0 : depth[p] + 1);
    for (int u : G[v]) {
      if (u == p) continue;
      dfs1(u, v);
      sz[v] += sz[u];
      if (sz[u] > sz[mxson[v]]) mxson[v] = u;
    }
  }
  void dfs2(int v = 1, int top = 1) {
    id[v] = ++idcnt;
    topf[v] = top;
    if (mxson[v]) dfs2(mxson[v], top);
    for (int u : G[v]) {
      if (u == mxson[v] || u == pa[v]) continue;
      dfs2(u, u);
    }
  }
  /// query 為區間資料結構
  int path_query(int a, int b) {
    int res = 0;
    while (topf[a] != topf[b]) { /// 若不在同一條鍊上
      if (depth[topf[a]] < depth[topf[b]]) swap(a, b);
      res = max(res, 0ll); // query : l = id[topf[a]], r = id[a
         ]
      a = pa[topf[a]];
    }
    /// 此時已在同一條鍊上
    if (depth[a] < depth[b]) swap(a, b);
    res = max(res, 0ll); // query : l = id[b], r = id[a]
    return res;
  }
};
```