Contents

1. DataStructure

1.1. Treap

```cpp
#define pii pair<int, int>
struct node {
  int tag = 0;
  int sum = 0;
  int prio = rand();
  int lson = 0;
  int rson = 0;
  int si = 0;
  int val = 0;
};
node treap[400005];
int cnt = 0;
int root = 0;

void update(int index) {
  int lson = treap[index].lson;
  int rson = treap[index].rson;
  treap[index].si = treap[lson].si + treap[rson].si + 1;
  treap[index].sum = treap[lson].sum;
  treap[index].sum += treap[rson].sum;
  treap[index].sum += treap[index].val;
}
void push(int index) {
  if (!treap[index].tag)
    return;
  swap(treap[index].lson, treap[index].rson);
  int lson = treap[index].lson;
  int rson = treap[index].rson;
  treap[lson].tag ^= 1;
  treap[rson].tag ^= 1;
  treap[index].tag = 0;
}

pii split(int rk, int index) {
  if (!index)
    return {0, 0};
  push(index);
  int lson = treap[index].lson;
  int rson = treap[index].rson;
  if (rk <= treap[lson].si) {
    pii temp = split(rk, lson);
    treap[index].lson = temp.second;
    update(index);
    return {temp.first, index};
  } else {
    pii temp = split(rk - treap[lson].si - 1, rson);
    treap[index].rson = temp.first;
    update(index);
    return {index, temp.second};
  }
}

int merge(int x, int y) {
  if (!x && !y)
    return 0;
  if (!x && y)
    return y;
  if (x && !y)
    return x;
  push(x);
  push(y);
  if (treap[x].prio < treap[y].prio) {
    treap[x].rson = merge(treap[x].rson, y);
    update(x);
    return x;
  } else {
    treap[y].lson = merge(x, treap[y].lson);
    update(y);
    return y;
  }
}

void insert(int x, int v) {
  pii temp = split(x - 1, root);
  cnt++;
  treap[cnt].val = v;
  update(cnt);
  temp.first = merge(temp.first, cnt);
  root = merge(temp.first, temp.second);
}

int query(int l, int r) {
  pii R = split(r, root);
  pii L = split(l - 1, R.first);
  int ret = treap[L.second].sum;
  R.first = merge(L.first, L.second);
  root = merge(R.first, R.second);
  return ret;
}

void modify(int l, int r) {
  pii R = split(r, root);
  pii L = split(l - 1, R.first);
  treap[L.second].tag ^= 1;
  R.first = merge(L.first, L.second);
  root = merge(R.first, R.second);
}
```

1.2. Dynamic Segment Tree

```cpp

#define int long long
using namespace std;

int n, q;
struct node {
  int data, lson, rson, tag;
  int rv() { return data + tag; }
};

node tree[20000005];
int a[200005];
int now = 1;
int mx = 1000000005;

void push(int index) {
  if (!tree[index].lson) {
    tree[index].lson = ++now;
  }
  if (!tree[index].rson) {
    tree[index].rson = ++now;
  }
  int lson = tree[index].lson;
  int rson = tree[index].rson;
  tree[lson].tag += tree[index].tag;
  tree[rson].tag += tree[index].tag;
  tree[index].data = tree[index].rv();
  tree[index].tag = 0;
}

void modify(int l, int r, int L, int R, int val, int index) {
  if (l == L && r == R) {
    tree[index].tag += val;
    return;
  }
  int mid = (l + r) >> 1;
  push(index);
  int lson = tree[index].lson;
  int rson = tree[index].rson;
  if (R <= mid) {
    modify(l, mid, L, R, val, lson);
  } else if (L > mid) {
    modify(mid + 1, r, L, R, val, rson);
  } else {
    modify(l, mid, L, mid, val, lson);
    modify(mid + 1, r, mid + 1, R, val, rson);
  }
  tree[index].data = tree[lson].rv() + tree[rson].rv();
}

int query(int l, int r, int L, int R, int index) {
  // cout << L << " " << R << "\n";
  if (l == L && r == R) {
    return tree[index].rv();
  }
  int mid = (l + r) >> 1;
  push(index);
  int lson = tree[index].lson;
  int rson = tree[index].rson;
  if (R <= mid) {
    return query(l, mid, L, R, lson);
  }
  if (L > mid) {
    return query(mid + 1, r, L, R, rson);
  }
  return query(l, mid, L, mid, lson) + query(mid + 1, r, mid + 1, R, rson);
}

signed main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  cin >> n >> q;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
    modify(1, mx, a[i], a[i], 1, 1);
  }
  while (q--) {
    char mode;
    int x, y;
    cin >> mode;
    if (mode == '?') {
      cin >> x >> y;
      cout << query(1, mx, x, y, 1) << "\n";
    } else {
      cin >> x >> y;
      modify(1, mx, a[x], a[x], -1, 1);
      a[x] = y;
      modify(1, mx, a[x], a[x], 1, 1);
    }
  }
}
```

2. Math

2.1. Mu

```
vector<int> prime;
bitset<1000005> vis;
int n;
int mu[1000005];

void init() {
  for (int i = 2; i <= n; i++) {
    if (!vis[i]) {
      prime.push_back(i);
      mu[i] = -1;
    }
    for (int p : prime) {
      if (i * p > n)
        break;
      vis[i * p] = 1;
      if (i % p == 0) {
        mu[i * p] = 0;
        break;
      } else {
        mu[i * p] = mu[i] * mu[p];
      }
    }
  }
}
```

2.2. Lucas

```
int fact[100005];
int p;

void init() {
  fact[0] = 1;
  for (int i = 1; i <= p; i++) {
    fact[i] = fact[i - 1] * i % p;
  }
}

int inv(int x, int p) {
  if (x == 1)
    return 1;
  return (p - p / x) * inv(p % x, p) % p;
}

int c(int x, int y, int p) {
  if (x < y)
    return 0;
  int k = fact[x] * inv(fact[y], p) % p;
  return k * inv(fact[x - y], p) % p;
}

int lucas(int x, int y, int p) {
  if (x == 0)
    return 1;
  return lucas(x / p, y / p, p) % p * c(x % p, y % p, p) % p;
}
```

2.3. Inv

```
int exgcd(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int d = exgcd(b, a % b, y, x);
  y -= x * (a / b);
  return d;
}

int inv(int a, int p) {
  int x, y;
  exgcd(a, p, x, y);
  return (x % p + p) % p;
}
```

2.4. Formula

2.4.1. Dirichlet Convolution

$\varepsilon = \mu * 1$
$\varphi = \mu * \mathrm{Id}$

2.4.2. Burnside's Lemma

Let X be a set and G be a group that acts on X. For $g \in G$, denote by X^g the elements fixed by g:

$$X^g = \{x \in X \mid gx \in X\}$$

Then

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|.$$

2.4.3. Pick Theorem

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

3. String

3.1. KMP

```
string s, t;
int pmt[1000005];

void init() {
  for (int i = 1, j = 0; i < t.size(); i++) {
    while (j && t[j] ^ t[i]) {
      j = pmt[j - 1];
    }
    if (t[j] == t[i])
      j++;
    pmt[i] = j;
  }
}

int kmp(string s) {
  int ret = 0;
  for (int i = 0, j = 0; i < s.size(); i++) {
    while (j && s[i] ^ t[j]) {
      j = pmt[j - 1];
    }
    if (s[i] == t[j]) {
      j++;
    }
    if (j == t.size()) {
      ret++;
      j = pmt[j - 1];
    }
  }
  return ret;
}
```

3.2. Longest Palindrome

```

#define int long long
using namespace std;

string s;
string t;
int n;
int d[2000005];
int ans = 0;

signed main() {
  cin >> t;
  n = t.size();
  for (int i = 0; i < 2 * n + 1; i++) {
    if (i & 1 ^ 1) {
      s += '0';
    } else {
      s += t[i / 2];
    }
  }
  n = s.size();
  d[0] = 1;
  for (int i = 0, l = 0, r = 0; i < n; i++) {
    if (i > r) {
      d[i] = 1;
      bool a = i + d[i] < n;
      bool b = i - d[i] >= 0;
      bool c = (s[i + d[i]] == s[i - d[i]];
      while (a && b && c) {
        d[i]++;
        a = i + d[i] < n;
        b = i - d[i] >= 0;
        c = ([i + d[i]] == s[i - d[i]]);
      }
      l = i - d[i] + 1;
      r = i + d[i] - 1;
    } else {
      int j = l + r - i;
      if (j - d[j] + 1 > l) {
        d[i] = d[j];
      } else {
        d[i] = r - i + 1;
        a = i + d[i] < n;
        b = i - d[i] >= 0;
        c = (s[i + d[i]] == s[i - d[i]]);
        while (a && b && c) {
          d[i]++;
          a = i + d[i] < n;
          b = i - d[i] >= 0;
          c = (s[i + d[i]] == s[i - d[i]]);
        }
        l = i - d[i] + 1;
        r = i + d[i] - 1;
      }
    }
    // cout << d[i] << " ";
    if (d[i] > d[ans]) {
      ans = i;
    }
  }
  for (int i = ans - d[ans] + 1; i < ans + d[ans]; i++) {
    if (s[i] ^ '0') {
      cout << s[i];
    }
  }
}
```

4. Graph

4.1. Dijkstra

```
int n, m;
vector<pair<int, int>> v[100005];
bitset<100005> vis;
int dis[100005];

void dijkstra(int x) {
  priority_queue<pair<int, int>, vector<pair<int, int>>,
                 greater<pair<int, int>>>
      pq;
  memset(dis, 0x3f, sizeof(dis));
  dis[x] = 0;
  pq.push({0, x});
  while (!pq.empty()) {
    pair<int, int> now = pq.top();
    pq.pop();
    if (vis[now.second])
      continue;
    vis[now.second] = 1;
    for (auto [i, w] : v[now.second]) {
      if (vis[i])
        continue;
      if (dis[now.second] + w < dis[i]) {
        dis[i] = dis[now.second] + w;
        pq.push({dis[i], i});
      }
    }
  }
}
```

4.2. MaximumFlow

```

#define int long long
using namespace std;

int n, m;
vector<int> v[1005];
int head[1005];
int c[1005][1005];
int lv[1005];
int ans = 0;

bool bfs() {
  memset(head, 0, sizeof(head));
  memset(lv, 0, sizeof(lv));
  queue<int> q;
  q.push(1);
  while (!q.empty()) {
    int now = q.front();
    q.pop();
    if (now == n)
      continue;
    for (int i : v[now]) {
      if (i != 1 && c[now][i] && !lv[i]) {
        lv[i] = lv[now] + 1;
        q.push(i);
      }
    }
  }
  return lv[n];
}

int dfs(int x, int flow) {
  int ret = 0;
  if (x == n)
    return flow;
  for (int i = head[x]; i < v[x].size(); i++) {
    int y = v[x][i];
    head[x] = y;
    if (c[x][y] && lv[y] == lv[x] + 1) {
      int d = dfs(y, min(flow, c[x][y]));
      flow -= d;
      c[x][y] -= d;
      c[y][x] += d;
      ret += d;
    }
  }
  return ret;
}

signed main() {
  cin >> n >> m;
  while (m--) {
    int x, y, z;
    cin >> x >> y >> z;
    if (c[x][y] || c[y][x]) {
      c[x][y] += z;
      continue;
    }
    v[x].push_back(y);
    v[y].push_back(x);
    c[x][y] = z;
  }
  while (bfs()) {
    ans += dfs(1, INT_MAX);
  }
  cout << ans;
}
```

4.3. SCC

```
int n, m;
vector<int> v[100005];
int d[100005];
int low[100005];
int cnt = 0;
stack<int> s;
int scc[100005];
int now = 0;

void dfs(int x) {
  d[x] = low[x] = ++cnt;
  s.push(x);
  for (int i : v[x]) {
    if (scc[i])
      continue;
    if (d[i]) {
      low[x] = min(low[x], d[i]);
    } else {
      dfs(i);
      low[x] = min(low[x], low[i]);
    }
  }
  if (d[x] == low[x]) {
    now++;
    while (!s.empty()) {
      int k = s.top();
      s.pop();
      scc[k] = now;
      if (k == x)
        break;
    }
  }
}
```

4.4. 2-SAT(CSES Giant Pizza)

```

#define int long long
using namespace std;

int n, m;
vector<int> v[200005];
int d[200005];
int low[200005];
int cnt = 0;
int now = 0;
int scc[200005];
stack<int> s;
int op[200005];
vector<int> v2[200005];
int ind[200005];
queue<int> q;
int ans[200005];

int no(int x) {
  if (x > m)
    return x - m;
  return x + m;
}

void dfs(int x) {
  d[x] = low[x] = ++cnt;
  s.push(x);
  for (int i : v[x]) {
    if (scc[i])
      continue;
    if (d[i]) {
      low[x] = min(low[x], d[i]);
    } else {
      dfs(i);
      low[x] = min(low[x], low[i]);
    }
  }
  if (d[x] == low[x]) {
    now++;
    while (!s.empty()) {
      int k = s.top();
      s.pop();
      scc[k] = now;
      if (k == x)
        break;
    }
  }
}

signed main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  cin >> n >> m;
  while (n--) {
    char a, b;
    int x, y;
    cin >> a >> x >> b >> y;
    if (a == '-')
      x = no(x);
    if (b == '-')
      y = no(y);
    v[no(x)].push_back(y);
    v[no(y)].push_back(x);
  }
  for (int i = 1; i <= 2 * m; i++) {
    if (!d[i]) {
      dfs(i);
    }
  }
  for (int i = 1; i <= m; i++) {
    if (scc[i] ^ scc[i + m]) {
      op[scc[i]] = scc[i + m];
      op[scc[i + m]] = scc[i];
    } else {
      cout << "IMPOSSIBLE";
      exit(0);
    }
  }
  for (int i = 1; i <= 2 * m; i++) {
    for (int j : v[i]) {
      if (scc[i] ^ scc[j]) {
        v2[scc[j]].push_back(scc[i]);
        ind[scc[i]]++;
      }
    }
  }
  for (int i = 1; i <= now; i++) {
    if (!ind[i]) {
      q.push(i);
    }
  }
  while (!q.empty()) {
    int k = q.front();
    q.pop();
    if (!ans[k]) {
      ans[k] = 1;
      ans[op[k]] = 2;
    }
    for (int i : v2[k]) {
      ind[i]--;
      if (!ind[i]) {
        q.push(i);
      }
    }
  }
  for (int i = 1; i <= m; i++) {
    if (ans[scc[i]] == 1) {
      cout << "+ ";
    } else {
      cout << "- ";
    }
  }
}
```

5. DP

5.1. Li-Chao Segment Tree

```
struct line {
  int a, b = 1000000000000000;
  int y(int x) { return a * x + b; }
};

line tree[4000005];
int n, x;
int s[200005];
int f[200005];
int dp[200005];

void update(line ins, int l = 1, int r = 1e6, int index = 1) {
  if (l == r) {
    if (ins.y(l) < tree[index].y(l)) {
      tree[index] = ins;
    }
    return;
  }
  int mid = (l + r) >> 1;
  if (tree[index].a < ins.a)
    swap(tree[index], ins);
  if (tree[index].y(mid) > ins.y(mid)) {
    swap(tree[index], ins);
    update(ins, l, mid, index << 1);
  } else {
    update(ins, mid + 1, r, index << 1 | 1);
  }
}

int query(int x, int l = 1, int r = 1000000, int index = 1) {
  int cur = tree[index].y(x);
  if (l == r) {
    return cur;
  }
  int mid = (l + r) >> 1;
  if (x <= mid) {
    return min(cur, query(x, l, mid, index << 1));
  } else {
    return min(cur, query(x, mid + 1, r, index << 1 | 1));
  }
}
```

5.2. CHO

```
struct line {
  int a, b;
  int y(int x) { return a * x + b; }
};

struct CHO {
  deque<line> dq;
  int intersect(line x, line y) {
    int d1 = x.b - y.b;
    int d2 = y.a - x.a;
    return d1 / d2;
  }
  bool check(line x, line y, line z) {
    int I12 = intersect(x, y);
    int I23 = intersect(y, z);
    return I12 < I23;
  }
  void insert(int a, int b) {
    if (!dq.empty() && a == dq.back().a)
      return;
    while (dq.size() >= 2 &&
           !check(dq[dq.size() - 2], dq[dq.size() - 1], {a, b})) {
      dq.pop_back();
    }
    dq.push_back({a, b});
  }
  void update(int x) {
    while (dq.size() >= 2 && dq[0].y(x) >= dq[1].y(x)) {
      dq.pop_front();
    }
  }
  int query(int x) {
    update(x);
    return dq.front().y(x);
  }
};
```

6. Geometry

6.1. Intersect

```
struct point {
  int x, y;
  point operator+(point b) { return {x + b.x, y + b.y}; }
  point operator-(point b) { return {x - b.x, y - b.y}; }
  int operator*(point b) { return x * b.x + y * b.y; }
  int operator^(point b) { return x * b.y - y * b.x; }
};

bool onseg(point x, point y, point z) {
  return ((x - z) ^ (y - z)) == 0 && (x - z) * (y - z) <= 0;
}

int dir(point x, point y) {
  int k = x ^ y;
  if (k == 0)
    return 0;
  if (k > 0)
    return 1;
  return -1;
}

bool intersect(point x, point y, point z, point w) {
  if (onseg(x, y, z) || onseg(x, y, w))
    return 1;
  if (onseg(z, w, x) || onseg(z, w, y))
    return 1;
  if (dir(y - x, z - x) * dir(y - x, w - x) == -1 &&
      dir(z - w, x - w) * dir(z - w, y - w) == -1) {
    return 1;
  }
  return 0;
}
```

6.2. Inside

```
int inside(point p) {
  int ans = 0;
  for (int i = 1; i <= n; i++) {
    if (onseg(a[i], a[i + 1], {p.x, p.y})) {
      return -1;
    }
    if (intersect({p.x, p.y}, {INF, p.y}, a[i], a[i + 1])) {
      ans ^= 1;
    }
    point temp = a[i].y > a[i + 1].y ? a[i] : a[i + 1];
    if (temp.y == p.y && temp.x > p.x) {
      ans ^= 1;
    }
  }
  return ans;
}
```

6.3. Minimum Euclidean Distance

```

#define int long long
#define pii pair<int, int>
using namespace std;

int n;
vector<pair<int, int>> v;
set<pair<int, int>> s;
int dd = LONG_LONG_MAX;

int dis(pii x, pii y) {
  return (x.first - y.first) * (x.first - y.first) +
         (x.second - y.second) * (x.second - y.second);
}

signed main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  cin >> n;
  for (int i = 0; i < n; i++) {
    int x, y;
    cin >> x >> y;
    x += 1000000000;
    v.push_back({x, y});
  }
  sort(v.begin(), v.end());
  int l = 0;
  for (int i = 0; i < n; i++) {
    int d = ceil(sqrt(dd));
    while (l < i && v[i].first - v[l].first > d) {
      s.erase({v[l].second, v[l].first});
      l++;
    }
    auto x = s.lower_bound({v[i].second - d, 0});
    auto y = s.upper_bound({v[i].second + d, 0});
    for (auto it = x; it != y; it++) {
      dd = min(dd, dis({it->second, it->first}, v[i]));
    }
    s.insert({v[i].second, v[i].first});
  }
  cout << dd;
}
```