CPK14 – Codebase

Boilerplate

```
#include<bits/stdc++.h>
using namespace std;
#define MOD 1000000007
#define nl <<'\n'
#define sp <<" "<<
#define fast_io ios_base::sync_with_stdio(false); cin.tie(NULL)
#define ll long long int
bool solve(){
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0;i < n;i++) {
        cin >> a[i];
    }
    return true;
}

int main(){
    fast_io;
#ifdef __linux__
    string path = "/media/gakshat468/New Volume/CP/";
#elif _WIN32
    string path = "D:/CP/";
#endif
#ifndef ONLINE_JUDGE
    freopen((path + "input.txt").c_str(), "r", stdin);
    freopen((path + "output.txt").c_str(), "w", stdout);
    freopen((path + "error.txt").c_str(), "w", stderr);
#endif
    int t = 1;
    cin >> t;
    while (t--) {
        solve();
        // cout << (solve() ? "YES\n" : "NO\n");
        // cout << (solve() ? "" : "-1\n");
    }
}
```

Graphs

Bridges(Offline)

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        }
        else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
```

```
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

Bridges(Online)

Articulation Points

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children = 0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        }
        else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p != -1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if (p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

Kosaraju SCC

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;
```

```
    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);

    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);

    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}
int main() {
//get adj and adj_rev
    used.assign(n, false);

    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);
    used.assign(n, false);
    reverse(order.begin(), order.end());
    for (auto v : order)
        if (!used[v]) {
            dfs2(v);
            // ... processing next component ...
            component.clear();
        }
}
```

```
// continuing from previous code

vector<int> roots(n, 0);
vector<int> root_nodes;
vector<vector<int>> adj_scc(n);

for (auto v : order)
if (!used[v]) {
    dfs2(v);
    int root = component.front();
    for (auto u : component) roots[u] = root;
    root_nodes.push_back(root);

    component.clear();
}
for (int v = 0; v < n; v++)
    for (auto u : adj[v]) {
        int root_v = roots[v],
            root_u = roots[u];

        if (root_u != root_v)
            adj_scc[root_v].push_back(root_u);
    }
```

Dijkistra

Bellmann Ford

```
void solve()
{
  //get n,m,adj,radj
```

```
pair<ll, ll> inf = { 1e18,0 };
vector<pair<ll, ll>> dis(n + 1, inf);
dis[1] = { 0,0 };
for (int k = 0;k < n;k++) {
    for (int u = 1;u <= n;u++) {
        for (auto& [v, c] : adj[u]) {
            if (dis[v].first > dis[u].first + c) {
                dis[v].first = dis[u].first + c;
                dis[v].second = u;
            }
        }
    }
}
ll pos = -1;
for (int u = 1;u <= n;u++) {
    for (auto& [v, c] : adj[u]) {
        if (dis[v].first > dis[u].first + c) {
            pos = v;
            break;
        }
    }
}
if (pos == -1) {
    cout << "NO\n";
    return;
}
vector<bool> vis(n + 1, false);
vector<ll> nodelist = { dis[pos].second };
while (!vis[pos]) {
    vis[pos] = true;
    nodelist.push_back(dis[pos].second);
    pos = dis[pos].second;
}
nodelist.push_back(dis[pos].second);
reverse(nodelist.begin(), nodelist.end());
}
```

BinaryUplifting

```
class binruplift {
public:
    int n;
    vector<vector<int>> anc;
    vector<int> height;
    int sz;
    binruplift(int n, vector<int> p): n(n) {
        height.resize(n + 1, -1);
        anc_precomp(p);
        sz = anc[0].size() - 1;
        height[0] = 0;
        for (int i = 1;i <= n;i++) {
            if (height[i] < 0)calht(i, height, p);
        }
    }
    binruplift() {}

    int kthancestor(int k, int x) {
        int j = 0;
        while (k > 0) {
            if (k & 1) x = anc[x][j];
            k >>= 1;
            j++;
        }
        return x;
```

```
}

void anc_precomp(vector<int>& p) {
    anc.resize(n + 1);
    anc[0].push_back(0);
    anc[1].push_back(0);
    for (int i = 2;i <= n;i++) {
        anc[i].push_back(p[i]);
    }
    int k = 2;
    for (int j = 1;k <= n;j++, k <<= 1) {
        anc[0].push_back(0);
        for (int i = 1;i <= n;i++) {
            anc[i].push_back(anc[anc[i][j - 1]][j - 1]);
        }
    }
}
int calht(int x, vector<int>& height, vector<int>& p) {
    if (height[x] >= 0) return height[x];
    height[x] = calht(p[x], height, p) + 1;
    return height[x];
}
int findlca(int a, int b) {
    if (height[a] > height[b]) swap(a, b);
    b = kthancestor(height[b] - height[a], b);
    return findlcarec(a, b, sz);
}
int findlcarec(int a, int b, int r) {
    if (a == b) return a;
    if (r == 0) return anc[a][0];
    if (anc[a][r - 1] == anc[b][r - 1]) {
        return findlcarec(a, b, r - 1);
    }
    else return findlcarec(anc[a][r - 1], anc[b][r - 1], r - 1);
}
int getDistance(int u, int v) {
    int l = findlca(u, v);
    return height[u] + height[v] - 2 * height[l];
}
};
```

Dinic's

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap): v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t): n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }
```

```
    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow <
    1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap -
    edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

Kuhn's O(nm)

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;
bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    mt.assign(k, -1);
    vector<bool> used1(n, false);
    for (int v = 0; v < n; ++v) {
        for (int to : g[v]) {
            if (mt[to] == -1) {
                mt[to] = v;
                used1[v] = true;
                break;
            }
        }
    }
    for (int v = 0; v < n; ++v) {
        if (used1[v])
            continue;
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

Heavy-Light Decomposition

```
vector<int> parent, depth, heavy, head, pos;
int cur_pos;

int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c, adj);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;
        }
    }
    return size;
}

void decompose(int v, int h, vector<vector<int>> const& adj) {
    head[v] = h, pos[v] = cur_pos++;
```

```
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);
    }
}


void init(vector<vector<int>> const& adj) {
    int n = adj.size();
    parent = vector<int>(n);
    depth = vector<int>(n);
    heavy = vector<int>(n, -1);
    head = vector<int>(n);
    pos = vector<int>(n);
    cur_pos = 0;

    dfs(0, adj);
    decompose(0, 0, adj);
}
int query(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path_max = segment_tree_query(pos[head[b]],
pos[b]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path_max = segment_tree_query(pos[a], pos[b]);
    res = max(res, last_heavy_path_max);
    return res;
}
```

Centroid Decomposition

```
struct node {
    int val;
    node(int val = 1e9):val(val) {}
};

struct upd {
    int pos;
    upd(int pos = 0):pos(pos) {};
};

node combine(const node& l, const node& r) {
    return min(l.val, r.val);
}

class centroidDecomposition {
public:
    int n;
    vector<vector<int>> adj;
    vector<int> sz;
    vector<int> decHead;
    vector<int> parent;
    vector<bool> removed;
    binruplift B;

    vector<node> tree;
    void resolve(int u, upd q) {
```

```
        tree[u].val = min(tree[u].val, B.getDistance(u, q.pos));
    }
    centroidDecomposition(int n, vector<vector<int>> adj):n(n),
adj(adj) {
        sz.resize(n + 1, 1);
        decHead.resize(n + 1);
        removed.resize(n + 1, false);
        parent.resize(n + 1);
        tree.resize(n + 1);
        getsz(1, 0);
        build(1, 0);
        B = binruplift(n, parent);
    }

    void getsz(int u, int p) {
        parent[u] = p;
        for (auto& v : adj[u])
            if (v != p) {
                getsz(v, u);
                sz[u] += sz[v];
            }
    };

    int getcentroid(int u, int n) {
        for (auto& v : adj[u]) {
            if (sz[v] > n / 2) {
                sz[u] -= sz[v];
                sz[v] += sz[u];
                return getcentroid(v, n);
            }
        }
        return u;
    };

    void build(int u, int p) {
        u = getcentroid(u, sz[u]);
        decHead[u] = p;
        removed[u] = true;
        sz[u] = 0;
        for (auto& v : adj[u]) {
            if (!removed[v])
                build(v, u);
        }
    };

    node getmin(int q) {
        node res;
        for (int u = q;u != 0;u = decHead[u]) {
            res = combine(res, tree[u].val + B.getDistance(q, u));
        }
        return res;
    }

    void update(int u, upd q) {
        for (;u != 0;u = decHead[u]) {
            resolve(u, q);
        }
    }
};
```

Geometry

Convex-Hull

```
struct pt {
   double x, y;
};

int orientation(pt a, pt b, pt c) {
   double v = a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y);
   if (v < 0) return -1; // clockwise
   if (v > 0) return +1; // counter-clockwise
   return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
   int o = orientation(a, b, c);
   return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<pt>& a, bool include_collinear = false) {
   pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
      return make_pair(a.y, a.x) < make_pair(b.y, b.x);
      });
   sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
      int o = orientation(p0, a, b);
      if (o == 0)
         return (p0.x - a.x) * (p0.x - a.x) + (p0.y - a.y) * (p0.y - a.y)
         < (p0.x - b.x) * (p0.x - b.x) + (p0.y - b.y) * (p0.y - b.y);
      return o < 0;
      });

   if (include_collinear) {
      int i = (int)a.size() - 1;
      while (i >= 0 && collinear(p0, a[i], a.back())) i--;
      reverse(a.begin() + i + 1, a.end());
   }

   vector<pt> st;
   for (int i = 0; i < (int)a.size(); i++) {
      while (st.size() > 1 && !cw(st[st.size() - 2], st.back(), a[i],
include_collinear))
         st.pop_back();
      st.push_back(a[i]);
   }

   a = st;
}
```

Area of Polygon

```
double area(const vector<point>& fig) {
   double res = 0;
   for (unsigned i = 0; i < fig.size(); i++) {
      point p = i ? fig[i - 1] : fig.back();
      point q = fig[i];
      res += (p.x - q.x) * (p.y + q.y);
   }
   return fabs(res) / 2;
}
```

Strings

Trie

```
const int k = 26;
```

```
class Trie {
    struct node {
        int count = 0;
        vector<int> next;
        node() {
            next.assign(k, -1);
        }
    };
    vector<node> tree;
    Trie() {
        tree.resize(1);
    }
public:
    void add_string(string s) {
        int x = 0;
        for (int i = 0;i < s.length();i++) {
            if (tree[x].next[s[i] - 'a'] == -1) {
                tree[x].next[s[i] - 'a'] = tree.size();
                tree.emplace_back();
            }
            x = tree[x].next[s[i] - 'a'];
        }
        tree[x].count++;
    }

    int countString(string s) {
        int x = 0;
        for (int i = 0;i < s.length();i++) {
            if (tree[x].next[s[i] - 'a'] == -1)return false;
            x = tree[x].next[s[i] - 'a'];
        }
        return tree[x].count;
    }
};
```

KMP

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j])
            j = pi[j - 1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

Z-Array

```
vector<int> getzarray(string a) {
    int n = a.size();
    vector<int> z(n);
    int l = 1, r = 1;
    for (int i = 1;i < n;i++) {
        if (r - i >= 0)z[i] = min(z[i - l], r - i);
        while (i + z[i] < n && a[i + z[i]] == a[z[i]])z[i]++;
        if (i + z[i] > r)l = i, r = i + z[i];
    }
    return z;
}
```

Suffix-Array

```
vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
```

```
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = { c[p[i]], c[(p[i] + (1 << h)) % n] };
            pair<int, int> prev = { c[p[i - 1]], c[(p[i - 1] + (1 << h)) % n] };
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}
```

Aho-Corasick

```
const int K = 26;

struct Vertex {
    int next[K];
    bool leaf = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p = -1, char ch = '$'): p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);


void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
```

```
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf = true;
}
int go(int v, char ch);
int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}
int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

Data Structures

DSU

```
class dsu {
public:
    vector<int> head;
    vector<int> childs;
    int n;
    dsu(int n) {
        head.resize(n + 1);
        childs.resize(n + 1);
        this->n = n;
        for (int i = 1;i <= n;i++) {
            head[i] = i;
            childs[i] = 1;
        }
    }
    int find(int x) {
        while (head[x] != head[head[x]]) {
            head[x] = head[head[x]];
        }
        return head[x];
    }
    void unite(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y)return;
        if (childs[x] > childs[y])swap(x, y);
        head[x] = y;
        childs[y] += childs[x];
    }
};
```

SQRT Decomposition

```
// input data
int n;
vector<int> a(n);
```

```
// preprocessing
int len = (int)sqrt(n + .0) + 1; // size of the block and the number of
blocks
vector<int> b(len);
for (int i = 0; i < n; ++i)
    b[i / len] += a[i];

// answering the queries
for (;;) {
    int l, r;
    // read input data for the next query
    int sum = 0;
    for (int i = l; i <= r; )
        if (i % len == 0 && i + len - 1 <= r) {
            // if the whole block starting at i belongs to [l, r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}
```

```
int sum = 0;
int c_l = l / len, c_r = r / len;
if (c_l == c_r)
for (int i = l; i <= r; ++i)
sum += a[i];
else {
    for (int i = l, end = (c_l + 1) * len - 1; i <= end; ++i)
        sum += a[i];
    for (int i = c_l + 1; i <= c_r - 1; ++i)
        sum += b[i];
    for (int i = c_r * len; i <= r; ++i)
        sum += a[i];
}
```

```
// tips to improve complexity
bool cmp(pair<int, int> p, pair<int, int> q) {
    if (p.first / BLOCK_SIZE != q.first / BLOCK_SIZE)
        return p < q;
    return (p.first / BLOCK_SIZE & 1) ? (p.second < q.second) :
(p.second > q.second);
}
```

Segment Tree

```
enum lazytype {
    LAZY_NONE,
    LAZY_INCREASE,
    LAZY_SETVAL
};
```

```
struct  node
{
   ll val;
   node(ll tval) {
      val = tval;
   }
   node() {
      val = 0;
   }
};
node combine(const node& lval, const node& rval)
{
   return (lval.val + rval.val);
}
struct lazyobj {
   ll value = 0;
   lazytype cmdtype = LAZY_NONE;
};

class segmenttree
{
public:
   node nullval;
   bool inputisonebased;
   void propagatecommand(lazyobj& updatethis, const lazyobj&
refrlazy) {
      switch (refrlazy.cmdtype)
      {
      case LAZY_NONE:
         break;
      case LAZY_SETVAL:
         updatethis.value = refrlazy.value;
         updatethis.cmdtype = LAZY_SETVAL;
         break;
      case LAZY_INCREASE:
         updatethis.value += refrlazy.value;
         if (updatethis.cmdtype == LAZY_NONE)
            updatethis.cmdtype = refrlazy.cmdtype;
         break;
      }
   }
   void fixindexing(int& a, int& b) {
      a -= inputisonebased;
      b -= inputisonebased;
   }
   void fixindexing(int& k) {
      k -= inputisonebased;
   }
   int n;
   vector<node> tree;
   vector<node> arr;
   vector<lazyobj> lazytree;
   void resolve(int v, int l, int r) {
      switch (lazytree[v].cmdtype) {
      case LAZY_NONE:
         break;
      case LAZY_INCREASE:
         tree[v].val += lazytree[v].value * (r - l + 1);
         break;
      case LAZY_SETVAL:
         tree[v].val = lazytree[v].value * (r - l + 1);
         break;
      }
      if (l == r) {
```

```
        arr[l].val = tree[v].val;
    }
    else {
        propagatecommand(lazytree[2 * v], lazytree[v]);
        propagatecommand(lazytree[2 * v + 1], lazytree[v]);
    }
    lazytree[v].value = 0;
    lazytree[v].cmdtype = LAZY_NONE;
}

segmenttree(vector<node>& tarr, bool _inputisonebased)
{
    n = tarr.size();
    inputisonebased = _inputisonebased;
    tree.resize(4 * n + 1);
    lazytree.resize(4 * n + 1);
    arr = tarr;
    build(1, 0, n - 1);
}
void build(int v, int l, int r)
{
    if (l == r)
    {
        tree[v] = arr[l];
        return;
    }
    int mid = (l + r) / 2;
    build(2 * v, l, mid);
    build(2 * v + 1, mid + 1, r);
    tree[v] = combine(tree[2 * v], tree[2 * v + 1]);
}
void pointupdate(int k, lazyobj update) {
    fixindexing(k);
    recrangeupdate(0, n - 1, k, k, 1, update);
}
void rangeupdate(int a, int b, lazyobj update) {
    fixindexing(a, b);
    if (a > b)return;
    recrangeupdate(0, n - 1, a, b, 1, update);
}
void recrangeupdate(int l, int r, int a, int b, int v, lazyobj update) {
    resolve(v, l, r);
    if (r<a || l>b) return;
    if (a <= l && r <= b) {
        propagatecommand(lazytree[v], update);
        resolve(v, l, r);
        return;
    }
    int mid = (l + r) / 2;
    recrangeupdate(l, mid, a, b, 2 * v, update);
    recrangeupdate(mid + 1, r, a, b, 2 * v + 1, update);
    tree[v] = combine(tree[2 * v], tree[2 * v + 1]);
}

node rangequery(int a, int b)
{
    fixindexing(a, b);
    if (a > b) return nullval;
    return recrangequery(0, n - 1, a, b, 1);
}
node pointquery(int k) {
    fixindexing(k);
    return recrangequery(0, n - 1, k, k, 1);
}
```

```
    node recrangequery(int l, int r, int a, int b, int v)
    {
        resolve(v, l, r);
        if (r<a || l>b)
            return nullval;
        if (a <= l && r <= b)
            return tree[v];
        int mid = (l + r) / 2;
        return combine(recrangequery(l, mid, a, b, 2 * v),
recrangequery(mid + 1, r, a, b, 2 * v + 1));
    }
};
```

2D-RangeQuery

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

int bit[1001][1001];
int n;
void update(int x, int y, int val) {
    for (; x <= n; x += (x & (-x))) {
        for (int i = y; i <= n; i += (i & (-i))) {
            bit[x][i] += val;
        }
    }
}

int query(int x1, int y1, int x2, int y2) {
    int ans = 0;
    for (int i = x2; i; i -= (i & (-i))) {
        for (int j = y2; j; j -= (j & (-j))) {
            ans += bit[i][j];
        }
    }
    for (int i = x2; i; i -= (i & (-i))) {
        for (int j = y1 - 1; j; j -= (j & (-j))) {
            ans -= bit[i][j];
        }
    }
    for (int i = x1 - 1; i; i -= (i & (-i))) {
        for (int j = y2; j; j -= (j & (-j))) {
            ans -= bit[i][j];
        }
    }
    for (int i = x1 - 1; i; i -= (i & (-i))) {
        for (int j = y1 - 1; j; j -= (j & (-j))) {
            ans += bit[i][j];
        }
    }
    return ans;
}

int main() {
    iostream::sync_with_stdio(false);
    cin.tie(0);
    int q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++) {
        char c;
        cin >> c;
        if (c == '*') update(j, i, 1);
    }
    while (q--) {
```

```
    int t;
    cin >> t;
    if (t == 1) {
        int x, y;
        cin >> y >> x;
        if (query(x, y, x, y)) update(x, y, -1);
        else update(x, y, 1);
    }
    else {
        int y1, x1, y2, x2;
        cin >> y1 >> x1 >> y2 >> x2;
        cout << query(x1, y1, x2, y2) << '\n';
    }
  }
  return 0;
}
```

Algebra

ModularArithmetic

```
long long binaryMultiply(long long, long long, long long);

int binaryExp(int a, int b, int M)
{
  // base a
  //power b
  //modular M
  //(a^b)%M

  int ans = 1;
  while (b > 0)
  {
    if (b & 1)
    {
      ans = (ans * 1ll * a) % M;
    }
    a = (a * 1ll * a) % M;
    b >>= 1;
  }
  return ans;
}

long long binaryMultiply(long long a, long long b, long long M)
{
  long long ans = 0;
  while (b > 0)
  {
    if (b & 1)
    {
      ans = (ans + a) % M;
    }
    a = (a + a) % M;
    b >>= 1;
  }
  return ans;
}

long long binaryExp_Big(long long a, long long b, long long M)
{
  //take b=b%(M-1); only valid if M is primes//ETF //Euler's theorem
  //take a=a%M

  ll ans = 1;
  while (b > 0){
```

```
        if (b & 1)
            ans = binaryMultiply(ans, a, M);

        a = binaryMultiply(a, a, M);
        b >>= 1;
    }
    return ans;
}

int factmod(int n, int p) {
    vector<int> f(p);
    f[0] = 1;
    for (int i = 1; i < p; i++)
        f[i] = f[i - 1] * i % p;

    int res = 1;
    while (n > 1) {
        if ((n / p) % 2)
            res = p - res;
        res = res * f[n % p] % p;
        n /= p;
    }
    return res;
}


int multiplicity_factorial(int n, int p) {
    int count = 0;
    do {
        n /= p;
        count += n;
    } while (n);
    return count;
}
```

Extended Euclidean Algorithm

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int& x0, int& y0, int& g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

Linear Sieve

```
vector<int> sieve(int size)
{
    vector<int> fac(size);
    int n = fac.size();
    for (int i = 0;i < n;i++) fac[i] = i;
    for (int i = 2;i * i <= n;i++)
    {
        for (int j = i * i;fac[i] == i && j < n;j += i)
            fac[j] = min(i, fac[j]);
    }
    return fac;
}
```

Primality Test

```
bool MillerRabin(u64 n) { // returns true if n is prime, else returns
false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

Totient Function

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
//O(root(n))

void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
```

```
}
//Calculate phi for all n
//O(nloglogn)
```

Discrete Log

```
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

FFT

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd>& a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;

    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++) {
        a0[i] = a[2 * i];
        a1[i] = a[2 * i + 1];
    }
    fft(a0, invert);
    fft(a1, invert);

    double ang = 2 * PI / n * (invert ? -1 : 1);
```

```
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n / 2] = a0[i] - w * a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n / 2] /= 2;
        }
        w *= wn;
    }
}



vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    int carry = 0;
    for (int i = 0; i < n; i++) {
        result[i] += carry;
        carry = result[i] / 10;
        result[i] %= 10;
    }

    return result;
}
```

Treap

```
typedef struct item* pitem;
struct item {
    int prior, value, cnt;
    bool rev;
    pitem l, r;
};

int cnt(pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt(pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
```

```
}

void push(pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap(it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
    }
}

void merge(pitem& t, pitem l, pitem r) {
    push(l);
    push(r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge(l->r, l->r, r), t = l;
    else
        merge(r->l, l, r->l), t = r;
    upd_cnt(t);
}

void split(pitem t, pitem& l, pitem& r, int key, int add = 0) {
    if (!t)
        return void(l = r = 0);
    push(t);
    int cur_key = add + cnt(t->l);
    if (key <= cur_key)
        split(t->l, l, t->l, key, add), r = t;
    else
        split(t->r, t->r, r, key, add + 1 + cnt(t->l)), l = t;
    upd_cnt(t);
}

void reverse(pitem t, int l, int r) {
    pitem t1, t2, t3;
    split(t, t1, t2, l);
    split(t2, t2, t3, r - l + 1);
    t2->rev ^= true;
    merge(t, t1, t2);
    merge(t, t, t3);
}

void output(pitem t) {
    if (!t)  return;
    push(t);
    output(t->l);
    printf("%d ", t->value);
    output(t->r);
}
```

SmalltoLarge

```
int n, q;
cin >> n >> q;
vector<pair<int, int>> adj[n + 1];

for (int i = 0;i < n - 1;i++) {
    int u, v, w;
    cin >> u >> v >> w;
    adj[u].push_back({ v,w });
    adj[v].push_back({ u,w });
}
```

```
vector<int> subtreesize(n + 1, 1), torootXor(n + 1, 0), dscn[n + 1];
unordered_map<int, int> cnt;
function<void(int, int)> calToRootXor = [&](int u, int p) {
    for (auto& [v, w] : adj[u]) {
        if (v == p)continue;
        torootXor[v] = torootXor[u] ^ w;
        calToRootXor(v, u);
        subtreesize[u] += subtreesize[v];
    }
};

calToRootXor(1, 0);
for (auto& i : torootXor)
cnt[i] = 0;
vector<long> ans(n + 1);
function<void(int, int, bool)> dfs = [&](int u, int p, bool keep) {
    int bigchild = -1;
    for (auto& [v, w] : adj[u]) {
        if (v == p)continue;
        if (bigchild == -1 || subtreesize[bigchild] < subtreesize[v])
            bigchild = v;
    }

    for (auto& [v, w] : adj[u]) {
        if (v == p)continue;
        if (v != bigchild)
            dfs(v, u, false);
    }

    if (bigchild != -1) {
        dfs(bigchild, u, true);
        swap(dscn[u], dscn[bigchild]);
        ans[u] += ans[bigchild];
    }

    dscn[u].push_back(u);
    ans[u] += cnt[torootXor[u]];
    cnt[torootXor[u]]++;
    for (auto& [v, w] : adj[u]) {
        if (v == p || v == bigchild)continue;
        for (auto& i : dscn[v]) {
            ans[u] += cnt[torootXor[i]];
            cnt[torootXor[i]]++;
            dscn[u].push_back(i);
        }
    }
    if (!keep) {
        for (auto& v : dscn[u]) {
            cnt[torootXor[v]]--;
        }
    }
};

dfs(1, 0, true);
debug(ans);
while (q--) {
    int u;
    cin >> u;
    cout << ans[u] nl;
}
```

Pbds

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

#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
tree_order_statistics_node_update> pbds; // find_by_order,
order_of_key

int main() {
    pbds A; // declaration

    A.insert(3);
    A.insert(4);
    A.insert(5);
    A.insert(5);
    A.insert(7);
    A.insert(8);
    A.insert(9);

    // finding kth element - 4th query
    // cout << "0th element: " << *A.find_by_order(1) << endl;
    // A.erase();
    // A.erase(A.upper_bound(4));
    // cout << "0th element: " << *A.find_by_order(1) << endl;
    cout << "No. of elems smaller than 6: " << A.order_of_key(6) <<
endl; // 2
    // // lower bound -> Lower Bound of X = first element >= X in the
set
    // cout << "Lower Bound of 6: " << *A.lower_bound(6) << endl;
    // // Upper bound -> Upper Bound of X = first element > X in the
set
    // cout << "Upper Bound of 6: " << *A.upper_bound(6) << endl;
    // // // Remove elements - 2nd query
    // A.erase(1);
    // A.erase(11); // element that is not present is not affected


    // A contains
}
```