

# UAP\_Nightjars CP Reference

University of Asia Pacific

Team: UAP\_Nightjars (Maruf,Rhythm,Arafat)

## Contents

<b>1 Data Structure</b>	<b>1</b>	4.8 Euler Phi . . . . .	9	<b>7 Mathematical Formulas</b>	<b>18</b>
1.1 BIT . . . . .	1	4.9 Fibonacci Number Faster . . . . .	9	7.1 Math Formula . . . . .	18
1.2 BST . . . . .	2	4.10 GCD LCM . . . . .	10	7.2 Matrices and Determinants . . . . .	18
1.3 Bit Binary Search . . . . .	2	4.11 Josephus . . . . .	10	7.3 Vectors . . . . .	19
1.4 DSU . . . . .	2	4.12 Large Number GCD . . . . .	10	7.4 Straight Lines . . . . .	19
1.5 Merge Sort Tree . . . . .	2	4.13 Lengenders Formula . . . . .	10		
1.6 Monotonous Queue . . . . .	2	4.14 Modular for Subtraction & Multiplication . . . . .	10		
1.7 Ordered Set . . . . .	2	4.15 Optimized Sieve . . . . .	10		
1.8 Segment Tree All . . . . .	3	4.16 Lower Prime Factorization . . . . .	10		
1.9 Sparse Table . . . . .	4	4.17 Prime or Not . . . . .	10		
<b>2 Dynamic Programming</b>	<b>4</b>	4.18 Number of Divisor A Number Have . . . . .	10		
2.1 Longest Common Subsequence print using next state . . . . .	4	4.19 Prime Factors . . . . .	11		
2.2 Longest Increasing Subsequence Print . . . . .	4	4.20 Formulas Using Prime Factorization Math . . . . .	11		
<b>3 Graphs</b>	<b>4</b>	4.21 Sum of Divisor . . . . .	11		
3.1 BFS . . . . .	4	4.22 Sieve . . . . .	11		
3.2 Bellman Ford . . . . .	5	4.23 n! Trailing Zero . . . . .	11		
3.3 Bipartite Garphs . . . . .	5	4.24 Number of Divisors 1 to n . . . . .	11		
3.4 Cycle or not Print Cycle . . . . .	5	<b>5 Mixed</b>	<b>11</b>		
3.5 Cycle or not in Directed Graph . . . . .	5	5.1 Run Code Technique . . . . .	11		
3.6 DFS . . . . .	6	5.2 i128 Bit . . . . .	11		
3.7 Dijkstra . . . . .	6	5.3 Base Conberstion . . . . .	11		
3.8 Floyd Warshall . . . . .	6	5.4 Bitset . . . . .	12		
3.9 Krushkals MST . . . . .	6	5.5 Bit Operation . . . . .	12		
3.10 LCA . . . . .	7	5.6 Merge Sort . . . . .	12		
3.11 Prims MST . . . . .	7	5.7 STL . . . . .	12		
3.12 Strongly Connected Components . . . . .	8	5.8 Sqrt Decomposition . . . . .	12		
3.13 Topological Sorting . . . . .	8	5.9 Binary Search . . . . .	13		
3.14 Zero One BFS . . . . .	8	5.10 Ternary Search . . . . .	13		
<b>4 Mathematics</b>	<b>8</b>	5.11 python . . . . .	13		
4.1 2D Prefix Sum . . . . .	8	<b>6 Strings</b>	<b>13</b>		
4.2 Binary Exponentiation . . . . .	9	6.1 All String Operation . . . . .	13		
4.3 Binary Multiplication with Mod . . . . .	9	6.2 Single Hashing . . . . .	13		
4.4 Modular Arithmetic . . . . .	9	6.3 Single Hashing . . . . .	14		
4.5 Binomial Coefficients nCr . . . . .	9	6.4 Largest Substring More than K . . . . .	14		
4.6 How Many Digits in n! . . . . .	9	6.5 Longest Common Prefix Of Two Substrings . . . . .	14		
4.7 Derangement . . . . .	9	6.6 Longest Common Substring . . . . .	14		
		6.7 Pattern Matching . . . . .	15		
		6.8 Two Strings Are Equal Or Not . . . . .	16		
		6.9 Lines . . . . .	16		
		6.10 Polygons . . . . .	17		

## 1. Data Structure

### 1.1. BIT

```
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;
struct BIT{ // 1-indexed
    int n;
    vector<int> t;
    BIT() {}
    BIT(int _n){
        n = _n;
        t.assign(n + 5, 0);
    }
    int qry(int i){
        int ans = 0;
        for (; i >= 1; i -= (i & -i))
            ans += t[i];
        return ans;
    }
    void upd(int i, int val){
        if (i <= 0)
            return;
        for (; i <= n; i += (i & -i))
            t[i] += val;
    }
    void upd(int l, int r, int val){
        upd(l, val);
        upd(r + 1, -val);
    }
    int qry(int l, int r)
    {
        return qry(r) - qry(l - 1);
    }
};
int32_t main(){
    int n, q;
    cin >> n >> q;
    BIT bit(n);
    bit.upd(x, 1); // demo
    bit.query(10); // dsemo, change this
    return 0;}
```

## 1.2. BST

```
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;

// the code returns a BST which will create if we
// add the values one by one
// here nodes are indicated by values and every node
// must be distinct
set<int> se;
map<int, int> l, r; // l contains the left child of
// the node, r contains right child of the node
int main() {
    int n;
    cin >> n;
    int k;
    cin >> k; // root of the tree
    se.insert(k);
    for (int i = 1; i < n; i++) {
        int k;
        cin >> k;
        auto it = se.upper_bound(k);
        if (it != se.end() && l.find(*it) == l.end())
            l[*it] = k;
        else
            --it, r[*it] = k;
        se.insert(k);
    }
    for (int i = 1; i <= n; i++)
        cout << l[i] << ' ' << r[i] << '\n';
    return 0;}
```

## 1.3. Bit Binary Search

```
// --- Bit Binary Search in o(log(n)) ---
const int M = 20, const int N = 1 << M, int
lower_bound(int val) { int ans = 0, sum = 0;
    for (int i = M - 1; i >= 0; i--) {
        int x = ans + (1 << i);
        if (sum + bit[x] < val) ans = x, sum += bit[x];
    } return ans + 1; }
```

## 1.4. DSU

```
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;
struct DSU {
    vector<int> par, rnk, sz;
    int c;

    DSU(int n) : par(n + 1), rnk(n + 1, 0), sz(n + 1,
        1), c(n) {
```

```
        for (int i = 1; i <= n; ++i)
            par[i] = i;
    }
    int find(int i) {
        return (par[i] == i ? i : (par[i] = find(par[i]
            )));
    }
    bool same(int i, int j) {
        return find(i) == find(j);
    }
    int get_size(int i) {
        return sz[find(i)];
    }
    int count() {
        return c; // connected components
    }
    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j)))
            return -1;
        else
            --c;
        if (rnk[i] > rnk[j])
            swap(i, j);
        par[i] = j;
        sz[j] += sz[i];
        if (rnk[i] == rnk[j])
            rnk[j]++;
        return j;
    }
};
```

## 1.5. Merge Sort Tree

```
// Mergesort Tree - Time <O(nlogn), O(log^2n)> -
// Memory O(nlogn)
// Mergesort Tree is a segment tree that stores the
// sorted subarray
// on each node.
```

```
vi st[4 * N];
void build(int p, int l, int r) {
    if (l == r) {
        st[p].pb(s[l]);
        return;
    }
    build(2 * p, l, (l + r) / 2);
    build(2 * p + 1, (l + r) / 2 + 1, r);
    st[p].resize(r - l + 1);
    merge(st[2 * p].begin(), st[2 * p].end(),
        st[2 * p + 1].begin(), st[2 * p + 1].end(),
        st[p].begin());
}

int query(int p, int l, int r, int i, int j, int a,
    int b) {
    if (j < l or i > r)
        return 0;
    if (i <= l and j >= r)
```

```
        return upper_bound(st[p].begin(), st[p].end(),
            b) -
            lower_bound(st[p].begin(), st[p].end(), a
            );
    return query(2 * p, l, (l + r) / 2, i, j, a, b) +
        query(2 * p + 1, (l + r) / 2 + 1, r, i, j, a
            , b);
}
```

## 1.6. Monotonous Queue

```
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;
struct monotonous_queue { // max, stores strictly
    // decreasing sequence of the current queue
    int a[N + 10], b[N + 10], l = 0, r = -1;

    void push(int val) {
        int cnt = 0;
        while (l <= r && a[r] <= val) {
            cnt += b[r] + 1;
            r--;
        }
        a[++r] = val;
        b[r] = cnt;
    }
    int top() {
        return a[l];
    }

    void pop() {
        if (l > r)
            return;
        if (b[l] > 0) {
            b[l]--;
            return;
        }
        l++;
    }
};
```

## 1.7. Ordered Set

```
// Find by Order (find_by_order) ->O(log(n));
// Order of Key (order_of_key)->O(log(n));
// Deletion (erase)->O(log(n));
```

```
#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; //additional line

// Define an indexed set
```

```

typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> indexed_set;

//greater(x ar the koto gula boro value ache)
//greater_equal(for multiset)
// typedef tree<int, null_type, less_equal<int>,
rb_tree_tag, tree_order_statistics_node_update>
indexed_set;

//for multiset or same value count
// typedef tree<pair<int, int>, null_type, less<pair
<int, int>>, rb_tree_tag,
tree_order_statistics_node_update> indexed_set;

// s.find_by_order(1)->first
// s.order_of_key(make_pair(20, 2))

int main() {
    indexed_set s;
    // Insert elements
    s.insert(10);

    // Find the 2nd smallest element (0-based
    indexing)
    cout << "2nd smallest: " << *s.find_by_order(1)
    << "\n";

    // Find the number of elements less than 20
    cout << "Number of elements < 20: " << s.
    order_of_key(20) << "\n";

    // Check if an element exists
    if (s.find(15) != s.end()) {
        cout << "15 is present in the set.\n";
    }

    // Erase an element
    s.erase(15);
    return 0;
}

```

## 1.8. Segment Tree All

```

#include <bits/stdc++.h>
#define int long long
using namespace std;
const long long inf = 1e18;
struct SegmentTree {
    vector< int > stree, lazy;
    int sz;
    vector< int > v;
    SegmentTree (vector< int > &ara) {
        v = ara;
        sz = ara.size();
        stree = vector< int > (4 * sz, inf);
        lazy = vector< int > (4 * sz, 0);
        build(1, 0, sz - 1);
    }
}

```

```

int merge(int v1, int v2) {
    return min(v1, v2);
}

int size() {
    return sz;
}

void push (int u) {
    int val = lazy[u];
    lazy[u] = 0;
    stree[u * 2] += val;
    stree[u * 2 + 1] += val;
    lazy[u * 2] += val;
    lazy[u * 2 + 1] += val;
}

// Definition : lazy propagation for range update
(SUM update)
void push(int u, int tl, int tr) {
    if(lazy[u] == 0 || tl == tr) return;
    int mid = (tl + tr) / 2;
    stree[u * 2] += lazy[u] * (mid - tl + 1);
    lazy[u * 2] += lazy[u];
    stree[u * 2 + 1] += lazy[u] * (tr - mid);
    lazy[u * 2 + 1] += lazy[u];
    lazy[u] = 0;
}

void build (int u, int tl, int tr) {
    if (tl > tr) return;
    if (tl == tr) {
        if (tl < v.size()) stree[u] = v[tl];
        return;
    }
    int mid = (tl + tr) / 2;
    build(u * 2, tl, mid);
    build(u * 2 + 1, mid + 1, tr);
    stree[u] = merge(stree[u * 2], stree[u * 2 +
    1]);
}

int query(int l, int r) {
    return query(1, 0, sz - 1, l, r);
}

void update(int l, int r, int val) {
    update(1, 0, sz - 1, l, r, val);
}

void update (int id, int val) {
    update(1, 0, sz - 1, id, val);
}

// Definition : return minimum of the range that
overlaps with l, r
int query(int u, int tl, int tr, int l, int r) {
    if (tl > tr) return inf;
    if (l > r) return inf;
    if (tr < l || tl > r) return inf;
    if (tl >= l && tr <= r) return stree[u];
    int mid = (tl + tr) / 2;
    push(u);
    int lc = query(u * 2, tl, mid, l, r);
    int rc = query(u * 2 + 1, mid + 1, tr, l, r);
    return merge(lc, rc);
}

```

```

void update (int u, int tl, int tr, int id, int
val) {
    if (id < tl || id > tr) return;
    if (tl == tr) {
        stree[u] = val;
        return;
    }
    if (tl > tr) return;
    int mid = (tl + tr) / 2;
    push(u);
    update(u * 2, tl, mid, id, val);
    update(u * 2 + 1, mid + 1, tr, id, val);
    stree[u] = merge(stree[u * 2], stree[u * 2 +
    1]);
}

void update (int u, int tl, int tr, int l, int r,
int val) {
    if (tl > tr) return;
    if (l > r) return;
    if (tr < l || tl > r) return;
    if (l <= tl && tr <= r) {
        stree[u] += val; //if range add -> stree[u]
        += (val * (tr - tl + 1));
        lazy[u] += val;
        return;
    }
    int mid = (tl + tr) / 2;
    push(u);
    update(u * 2, tl, mid, l, r, val);
    update(u * 2 + 1, mid + 1, tr, l, r, val);
    stree[u] = merge(stree[u * 2], stree[u * 2 +
    1]);
}

void printTree() {
    printTree(1, 0, sz - 1);
}

void printTree(int u, int tl, int tr) {
    cout << "Node " << u << " -> [" << tl << ", "
    << tr << "] = " << stree[u] << "\n";
    if (tl == tr) return; // leaf node

    int mid = (tl + tr) / 2;
    printTree(u * 2, tl, mid); // left child
    printTree(u * 2 + 1, mid + 1, tr); // right
    child
}

// using alive array
int Kth_one(int k) {
    return Kth_one(1, 0, sz - 1, k);
}

int Kth_one(int u, int tl, int tr, int k) {
    if(k > stree[u] || k <= 0) return -1; // not
    enough 1's
    if (tl == tr) {
        return tl;
    }
    // push(u);
    int mid = (tl + tr) / 2;
    if (stree[u * 2] >= k) {

```

```

        return Kth_one(u * 2, tl, mid, k);
    } else {
        return Kth_one(u * 2 + 1, mid + 1, tr, k -
            stree[u * 2]);
    }
}
};
signed main () {

return 0;
}

```

## 1.9. Sparse Table

```

#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 9;
int t[N][18], a[N];

void build(int n) {
    for (int i = 1; i <= n; ++i)
        t[i][0] = a[i];
    for (int k = 1; k < 18; ++k) {
        for (int i = 1; i + (1 << k) - 1 <= n; ++i) {
            t[i][k] = min(t[i][k - 1], t[i + (1 << (k -
                1))][k - 1]);
        }
    }
}

int query(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    return min(t[l][k], t[r - (1 << k) + 1][k]);
}

int32_t main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    build(n);
    int q;
    cin >> q;
    while (q--) {
        int l, r;
        cin >> l >> r;
        ++l;
        ++r;
        cout << query(l, r) << '\n';
    }
    return 0;
}

```

## 2. Dynamic Programming

### 2.1. Longest Common Subsequence print using next state

```

// Longest Common Subsequence print using next state
#include <bits/stdc++.h>
#define ll long long
using namespace std;
int n, m;
vector<int> a,b;
vector<vector<int>>> dp;
vector<vector<pair<int , int>>> next_state;

int lcs(int i,int j){
    if(dp[i][j]!= -1) return dp[i][j];
    if(i==n || j==m) return 0;
    if(a[i]==b[j]){
        next_state[i][j] = {i + 1, j + 1};
        dp[i][j] = 1 + lcs(i+1,j+1);
    }
    else{
        int x = lcs(i+1,j);
        int y = lcs(i,j+1);
        if(x>y){
            next_state[i][j] = {i + 1, j};
        }
        else{
            next_state[i][j] = {i, j + 1};
        }
        dp[i][j] = max(dp[i][j],x);
        dp[i][j] = max(dp[i][j],y);
    }
    return dp[i][j];
}

void I_Am_Here() {
    cin >> n >> m;
    a = vector<int>(n);
    b = vector<int>(m);
    dp = vector<vector<int>>>(n + 1, vector<int>(m +
        1, -1));
    next_state = vector<vector<pair<int, int>>>(n +
        1, vector<pair<int, int>>(m + 1, {0, 0}));

    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    for (int i = 0; i < m; i++) {
        cin >> b[i];
    }
    cout << lcs(0,0) << endl;

    int i=0 , j=0;
    while(i<n && j<m){
        int nxPosI=next_state[i][j].ff;
        int nxPosJ=next_state[i][j].ss;
        if(a[i]==b[j] && nxPosI==i+1 && nxPosJ==j+1){

```

```

        cout << a[i] << " ";
    }
    i = nxPosI;
    j = nxPosJ;
}
cout << endl;
}

```

```

int32_t main() {
    I_Am_Here();
}

```

### 2.2. Longest Increasing Subsequence Print

```

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

int lengthOfLIS(vector<int>& arr) {
    vector<int> lis;
    for(int x : arr) {
        auto it = lower_bound(lis.begin(), lis.end(),
            x);
        if(it == lis.end()) lis.push_back(x);
        else *it = x;
    }
    return lis.size();
}

int main() {
    vector<int> arr = {10, 22, 9, 33, 21, 50, 41,
        60};
    cout << lengthOfLIS(arr) << "\n";
    return 0;
}

```

## 3. Graphs

### 3.1. BFS

```

const int N = 1e5 + 10; // Maximum num of nodes
int dist[N], par[N];
vector<int> adj[N];
queue<int> q;
void bfs(int s) {
    memset(dist, 63, sizeof(dist));
    dist[s] = 0;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (auto v : adj[u])
            if (dist[v] > dist[u] + 1) {
                par[v] = u;
                dist[v] = dist[u] + 1;
            }
    }
}

```

```

        q.push(v);
    }
}

```

### 3.2. Bellman Ford

```

const int N = 3e5 + 9;
struct st {
    int a, b, cost;
} e[N];
const int INF = 2e9;
int32_t main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < m; i++)
        cin >> e[i].a >> e[i].b >> e[i].cost;
    int s;
    cin >> s; // is there any negative cycle which is
               // reachable from s ?

    vector<int> d(n, INF); // for finding any cycle(
                           // not necessarily from s) set d[i] = 0 for all
                           // i
    d[s] = 0;
    vector<int> p(n, -1);
    int x;

    for (int i = 0; i < n; ++i) {
        x = -1;
        for (int j = 0; j < m; ++j) {
            if (d[e[j].a] < INF) {
                if (d[e[j].b] > d[e[j].a] + e[j].cost) {
                    d[e[j].b] = max(-INF, d[e[j].a] + e[j].cost); // for overflow
                    p[e[j].b] = e[j].a;
                    x = e[j].b;
                }
            }
        }
    }

    if (x == -1)
        cout << "No negative cycle from " << s;
    else {
        int y = x; // x can be on any cycle or
                   // reachable from some cycle
        for (int i = 0; i < n; ++i)
            y = p[y];

        vector<int> path;
        for (int cur = y; cur = p[cur]) {
            path.push_back(cur);
            if (cur == y && path.size() > 1)
                break;
        }

        reverse(path.begin(), path.end());
        cout << "Negative cycle: ";
        for (int i = 0; i < path.size(); ++i)
            cout << path[i] << ' ';
    }
}

```

```

}
return 0;
}

```

### 3.3. Bipartite Graphs

```

const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
int col[N];
bool ok;
void dfs(int u) {
    vis[u] = true;
    for (auto v : g[u]) {
        if (!vis[v]) {
            col[v] = col[u] ^ 1;
            dfs(v);
        } else {
            if (col[u] == col[v])
                ok = false;
        }
    }
}

int32_t main() {
    int n, m;
    cin >> n >> m;
    while (m--) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    ok = true;
    for (int i = 1; i <= n; i++)
        if (!vis[i])
            dfs(i);

    if (ok) cout << "YES\n";
    else cout << "NO\n";
}

```

### 3.4. Cycle or not Print Cycle

```

vector < vector < int > > adj;
vector < bool > vis;
vector < int > parent;
bool isCycle = 0;
int stop;
int start;
void dfs(int s, int p) {
    vis[s] = 1;
    parent[s] = p;
    for (auto x : adj[s]) {
        if (vis[x]) {
            if (parent[s] == x) {
                continue; // agar tar thake aysi
            }
        }
    }
}

```

```

    else {
        isCycle = 1;
        start = x;
        stop = s;
        return;
    }
}
else {
    dfs(x, s);
}
}

void I_Am_Here() {
    int n;
    cin >> n;
    int m;
    cin >> m;
    adj = vector < vector < int > > (n + 1, vector <
        int > ());
    vis = vector < bool > (n + 1, false);
    parent = vector < int > (n + 1, 0);

    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    for (int i = 1; i <= n; i++) {
        if (!vis[i]) {
            dfs(i, 0);
            if (isCycle) break;
        }
    }

    if (!isCycle) {
        cout << "IMPOSSIBLE" << endl;
        return;
    }

    int t = start;
    vector<int> ans;
    ans.push_back(start);
    while (start != stop) {
        start = parent[start];
        ans.push_back(start);
    }
    ans.push_back(t);
    cout << ans.size() << endl;
    for (auto x : ans) cout << x << " "; cout << endl;
}

```

### 3.5. Cycle or not in Directed Graph

```

vector<vector<int>>> adj;
vector<int> vis;
vector<int> parent;
int n, m;
bool isCycle = 0;
int start, End;
void dfs(int s) {

```

```

vis[s]=1;
for(auto child : adj[s]){
    if(!vis[child]){
        parent[child]=s;
        dfs(child);
        if(isCycle) return;
    }
    else if (vis[child]==1){
        isCycle=1;
        start = child;
        End = s;
        return;
    }
}
vis[s]=2;
}

void I_Am_Here() {
    cin>>n>>m;
    adj = vector<vector<int>>(n+1,vector<int>());
    vis = vector<int>(n+1,0);
    parent = vector<int>(n+1,0);

    for(int i=0,x,y ; i<m ; i++){
        cin>>x>>y;
        adj[x].push_back(y);
    }
    for(int i=1 ; i<=n ; i++){
        if(!vis[i]){
            dfs(i);
        }
        if(isCycle){
            break;
        }
    }
    if(!isCycle){
        cout<<"IMPOSSIBLE"<<endl;
        return ;
    }
    vector<int>ans;
    ans.push_back(start);
    ans.push_back(End);
    while(start != End){
        End = parent[End];
        ans.push_back(End);
    }
    reverse(full(ans));
    cout<<ans.size()<<endl;
    for(auto i:ans){
        cout<<i<<' ';
    }
    cout<<endl;
}

```

### 3.6. DFS

```

const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
void dfs(int u) {

```

```

    vis[u] = true;
    for (auto v : g[u])
        if (!vis[v])
            dfs(v);
}

int32_t main() {
    int n, m;
    cin >> n >> m;
    while (m--) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(u);
}

```

### 3.7. Dijkstra

```

const int N = 3e5 + 9, mod = 998244353;
int n, m;
vector<pair<int, int>> g[N], r[N];

vector<long long> dijkstra(int s, int t, vector<int>
    &cnt) {
    const long long inf = 1e18;
    priority_queue<pair<long long, int>, vector<pair<
        long long, int>>, greater<pair<long long, int>
        >>> q;
    vector<long long> d(n + 1, inf);
    vector<bool> vis(n + 1, 0);

    q.push({0, s});
    d[s] = 0;
    cnt.resize(n + 1, 0); // number of shortest paths
    cnt[s] = 1;

    while (!q.empty()) {
        auto x = q.top();
        q.pop();
        int u = x.second;
        if (vis[u]) continue;
        vis[u] = 1;

        for (auto y : g[u]) {
            int v = y.first;
            long long w = y.second;
            if (d[u] + w < d[v]) {
                d[v] = d[u] + w;
                q.push({d[v], v});
                cnt[v] = cnt[u];
            } else if (d[u] + w == d[v]) {
                cnt[v] = (cnt[v] + cnt[u]) % mod;
            }
        }
    }
    return d;
}

int u[N], v[N], w[N];

```

```

int32_t main() {
    int s, t;
    cin >> n >> m >> s >> t;
    for (int i = 1; i <= m; i++) {
        cin >> u[i] >> v[i] >> w[i];
        g[u[i]].push_back({v[i], w[i]});
        r[v[i]].push_back({u[i], w[i]});
    }
    vector<int> cnt1, cnt2;
    auto d1 = dijkstra(s, t, cnt1);
    auto d2 = dijkstra(t, s, cnt2);
    long long ans = d1[t];

    for (int i = 1; i <= m; i++) {
        int x = u[i], y = v[i];
        long long nw = d1[x] + w[i] + d2[y];
        if (nw == ans && 1LL * cnt1[x] * cnt2[y] % mod
            == cnt1[t])
            cout << "YES\n";
        else if (nw - ans + 1 < w[i]) cout << "CAN " <<
            nw - ans + 1 << '\n';

        else cout << "NO\n";
    }
    return 0;
}

```

### 3.8. Floyd Warshall

```

const int N = 105;
int d[N][N];
int main() {
    int n = 10;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i != j)
                d[i][j] = 1e9;
        }
    }

    for (int k = 1; k <= n; ++k) {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }

    return 0;
}

```

### 3.9. Krushkals MST

```

const int N = 3e5 + 9, mod = 1e9;
struct dsu {
    vector<int> par, rnk, size;
    int c;

```

```

dsu(int n) : par(n + 1), rnk(n + 1, 0), size(n + 1, 1), c(n) {
    for (int i = 1; i <= n; ++i)
        par[i] = i;
}

int find(int i) {
    return (par[i] == i ? i : (par[i] = find(par[i])));
}

bool same(int i, int j) {
    return find(i) == find(j);
}

int get_size(int i) {
    return size[find(i)];
}

int count() {
    return c; // connected components
}

int merge(int i, int j) {
    if ((i = find(i)) == (j = find(j)))
        return -1;
    else
        --c;
    if (rnk[i] > rnk[j])
        swap(i, j);
    par[i] = j;
    size[j] += size[i];
    if (rnk[i] == rnk[j])
        rnk[j]++;
    return j;
}

};

int32_t main() {
    int n, m;
    cin >> n >> m;
    vector<array<int, 3>> ed;
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        ed.push_back({w, u, v});
    }

    sort(ed.begin(), ed.end());
    long long ans = 0;
    dsu d(n);

    for (auto e : ed) {
        int u = e[1], v = e[2], w = e[0];
        if (d.same(u, v))
            continue;
        ans += w;
        d.merge(u, v);
    }
    cout << ans << '\n';
}

```

```
return 0;}
```

### 3.10. LCA

```

const int N = 3e5 + 9, LG = 18;
vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
void dfs(int u, int p = 0) {
    par[u][0] = p;
    dep[u] = dep[p] + 1;
    sz[u] = 1;

    for (int i = 1; i <= LG; i++)
        par[u][i] = par[par[u][i - 1]][i - 1];

    for (auto v : g[u])
        if (v != p) {
            dfs(v, u);
            sz[u] += sz[v];
        }
}

int lca(int u, int v) {
    if (dep[u] < dep[v])
        swap(u, v);
    for (int k = LG; k >= 0; k--)
        if (dep[par[u][k]] >= dep[v])
            u = par[u][k];
    if (u == v)
        return u;
    for (int k = LG; k >= 0; k--)
        if (par[u][k] != par[v][k])
            u = par[u][k], v = par[v][k];
    return par[u][0];
}

int kth(int u, int k) {
    assert(k >= 0);
    for (int i = 0; i <= LG; i++)
        if (k & (1 << i))
            u = par[u][i];
    return u;
}

int dist(int u, int v) {
    int l = lca(u, v);
    return dep[u] + dep[v] - (dep[l] << 1);
}

// kth node from u to v, 0th node is u
int go(int u, int v, int k) {
    int l = lca(u, v);
    int d = dep[u] + dep[v] - (dep[l] << 1);
    assert(k <= d);
    if (dep[l] + k <= dep[u])
        return kth(u, k);
    k -= dep[u] - dep[l];
    return kth(v, dep[v] - dep[l] - k);
}

```

```

}

int32_t main() {
    int n;
    cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(1);
    int q;
    cin >> q;
    while (q--) {
        int u, v;
        cin >> u >> v;
        cout << dist(u, v) << '\n';
    }
    return 0;
}

```

### 3.11. Prims MST

```

const int N = 2020;
int g[N][N], w[N], to[N], selected[N];

long long Prims(int n, vector<pair<int, int>> &edges) {
    long long ans = 0;
    for (int i = 1; i <= n; i++)
        w[i] = 1e9, selected[i] = 0, to[i] = -1;
    w[1] = 0;

    for (int i = 1; i <= n; i++) {
        int u = -1;
        for (int j = 1; j <= n; j++)
            if (!selected[j] && (u == -1 || w[j] < w[u]))
                u = j;

        if (w[u] == 1e9)
            return -1; // NO MST

        selected[u] = 1;
        ans += w[u];

        if (to[u] != -1)
            edges.emplace_back(u, to[u]); // order of
            the edges may be changed

        for (int v = 1; v <= n; v++)
            if (g[u][v] < w[v])
                w[v] = g[u][v], to[v] = u;
    }

    return ans;
}

```

```

string s[N];
int main() {
    int n, m;
    cin >> n >> m;

    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            g[i][j] = 1e9;

    for (int i = 1; i <= n; i++)
        cin >> s[i];

    for (int i = 1; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
            int w = 0;
            for (int k = 0; k < m; k++)
                w = max(w, (int)abs(s[i][k] - s[j][k]));
            g[i][j] = min(g[i][j], w);
            g[j][i] = min(g[j][i], w);
        }
    }

    vector<pair<int, int>> ed;
    long long ans = Prims(n, ed);
    int res = 0;
    for (auto e : ed) res = max(res, g[e.first][e.second]);
    cout << res << '\n';
    return 0;
}

```

/\*<https://www.codechef.com/ICL2016/problems/ICL16A>\*/

### 3.12. Strongly Connected Components

**const int** N = 3e5 + 9;  
*// given a directed graph return the minimum number of edges to be added so that the whole graph become an SCC*

```

bool vis[N];
vector<int> g[N], r[N], G[N], vec; // G is the condensed graph
void dfs1(int u) {
    vis[u] = 1;
    for (auto v : g[u])
        if (!vis[v])
            dfs1(v);
    vec.push_back(u);
}

```

```
vector<int> comp;
```

```

void dfs2(int u) {
    comp.push_back(u);
    vis[u] = 1;
    for (auto v : r[u])
        if (!vis[v])
            dfs2(v);
}

```

```

}

int idx[N], in[N], out[N];

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        r[v].push_back(u);
    }

    for (int i = 1; i <= n; i++)
        if (!vis[i])
            dfs1(i);

    reverse(vec.begin(), vec.end());
    memset(vis, 0, sizeof vis);
    int scc = 0;

    for (auto u : vec) {
        if (!vis[u]) {
            comp.clear();
            dfs2(u);
            scc++;
            for (auto x : comp)
                idx[x] = scc;
        }
    }

    for (int u = 1; u <= n; u++) {
        for (auto v : g[u]) {
            if (idx[u] != idx[v]) {
                in[idx[v]]++;
                out[idx[u]]++;
                G[idx[u]].push_back(idx[v]);
            }
        }
    }

    int needed_in = 0, needed_out = 0;
    for (int i = 1; i <= scc; i++) {
        if (!in[i]) needed_in++;
        if (!out[i]) needed_out++;
    }
    int ans = max(needed_in, needed_out);
    if (scc == 1) ans = 0;
    cout << ans << '\n';
    return 0;
}

```

### 3.13. Topological Sorting

```

vector<int> adj[100005];
vector<int> ans;
vector<int> vis;
void dfs(int u) {
    vis[u] = 1;
    for (int v : adj[u]) {

```

```

        if (!vis[v]) dfs(v);
    }
    ans.push_back(u);
}

vector<int> topSort(int n) {
    vis.assign(n + 1, 0);
    ans.clear();
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) dfs(i);
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
// https://cses.fi/problemset/task/1679

```

### 3.14. Zero One BFS

```

// 0-1 BFS - O(V+E)
const int N = 1e5 + 5;
int dist[N];
vector<pii> adj[N];
deque<pii> dq;

void zero_one_bfs(int x) {
    cl(dist, 63);
    dist[x] = 0;
    dq.push_back({x, 0});

    while (!dq.empty()) {
        int u = dq.front().st;
        int ud = dq.front().nd;
        dq.pop_front();

        if (dist[u] < ud)
            continue;

        for (auto x : adj[u]) {
            int v = x.st;
            int w = x.nd;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                if (w)
                    dq.push_back({v, dist[v]});
                else
                    dq.push_front({v, dist[v]});
            }
        }
    }
}

```

## 4. Mathematics

### 4.1. 2D Prefix Sum

```

for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {

```



```

    prefixSum[i][j] = prefixSum[i-1][j] + prefixSum[
        i][j-1] - prefixSum[i-1][j-1] + grid[i-1][j
            -1];
    }}
cin>>x1>>y1>>x2>>y2;
cout<<prefixSum[x2][y2] + prefixSum[x1-1][y1-1] -
    prefixSum[x1-1][y2] - prefixSum[x2][y1-1]<<endl;

```

## 4.2. Binary Exponentiation

```

long long binpow(long long a, long long b, long long
    m) {
    a %= m;
    long long res = 1;
    while (b > 0) {
        if (b & 1) res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}

```

## 4.3. Binary Multiplication with Mod

```

long long binmul(long long a, long long b, long long
    m) {
    long long res = 0;
    a %= m;
    while (b > 0) {
        if (b & 1) res = (res + a) % m;
        a = (a + a) % m;
        b >>= 1;
    }
    return res;
}

```

## 4.4. Modular Arithmetic

```

// Greatest Common Divisor & Lowest Common Multiple
ll gcd(ll a, ll b) {
    return b ? gcd(b, a % b) : a;
}

ll lcm(ll a, ll b) {
    return a / gcd(a, b) * b;
}

// Multiply caring overflow
ll mulmod(ll a, ll b, ll m = MOD) {
    ll r = 0;
    for (a %= m; b; b >>= 1, a = (a * 2) % m)
        if (b & 1)
            r = (r + a) % m;
    return r;
}

```

```

// Another option for mulmod is using long double
ull mulmod(ull a, ull b, ull m = MOD) {
    ull q = (ld)a * (ld)b / (ld)m;
    ull r = a * b - q * m;
    return (r + m) % m;
}

// Fast exponential
ll fexp(ll a, ll b, ll m = MOD) {
    ll r = 1;
    for (a %= m; b; b >>= 1, a = (a * a) % m)
        if (b & 1)
            r = (r * a) % m;
    return r;
}

```

## 4.5. Binomial Coefficients nCr

```

const int N = 2e5 + 9, mod = 998244353;
long long fact[N], finv[N], inv[N];
void precal_factorial(int n) {
    inv[0] = inv[1] = finv[0] = finv[1] = fact[0] =
        fact[1] = 1;
    for (int i = 2; i <= n; i++) {
        inv[i] = inv[mod % i] * (mod - mod / i) % mod;
        finv[i] = (finv[i - 1] * inv[i]) % mod;
        fact[i] = (fact[i - 1] * i) % mod;
    }
}

long long ncr(long long n, long long r) {
    if (r > n) return 0;
    long long a = (finv[r] * finv[n - r]) % mod;
    return (a * fact[n]) % mod;
}

```

## 4.6. How Many Digits in n!

```

#include <bits/stdc++.h>
using namespace std;
int findDigits(int n) { if (n < 0) return 0; if (n <=
    1) return 1; double digits = 0;
    for (int i = 2; i <= n; i++) digits += log10(i);
    return floor(digits) + 1; }

```

## 4.7. Derangement

```

const int N = 1e6 + 9, mod = 1e9 + 7;
int d[N];
int32_t main() {
    d[0] = 1;
    d[1] = 0;
}

```

```

for (int i = 1; i < N; i++)
    d[i] = 1LL * (i - 1) * (d[i - 1] + d[i - 2]) %
        mod;

int n;
cin >> n;
cout << d[n] << '\n';
return 0;
}

```

*/\* There are n children at a Christmas party, and each of them has brought a gift. The idea is that everybody will get a gift brought by someone else. In how many ways can the gifts be distributed? \*/*

## 4.8. Euler Phi

```

// Euler phi (totient)
int ind = 0, pf = primes[0], ans = n;
while (1ll * pf * pf <= n) {
    if (n % pf == 0)
        ans -= ans / pf;
    while (n % pf == 0)
        n /= pf;
    pf = primes[++ind];
}
if (n != 1) ans -= ans / n;

// Euler Totient Function 1 to n in O(nloglog(n))
const int N = 1e5 + 9;
int phi[N];

void totient() {
    for (int i = 1; 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;
    }
}

```

## 4.9. Fibonacci Number Faster

```

int fib(long long n, int mod) {
    assert(n >= 0);
    if (n <= 1) return n;
    int a = 0, b = 1;
    long long i = 1ll << (63 - __builtin_clzll(n) -
        1);

    for (; i; i >>= 1) {
        int na = (a * (long long)a + b * (long long)b)
            % mod;
        int nb = (2ll * a + b) * b % mod;
        a = na;
        b = nb;
    }
}

```

```

    if (n & i) {
        int c = a + b;
        if (c >= mod) c -= mod;
        a = b;
        b = c;
    }
    return b;
}

```

#### 4.10. GCD LCM

```

int gcd(int a, int b){if (a == 0) return b; return
gcd(b % a, a);}
long long lcm(long long a, long long b){ return (a /
__gcd(a, b)) * b;}

```

#### 4.11. Josephus

```

/* Josephus Problem - It returns the position to be,
   in order to not die. O(n) */
/* With k=2, for instance, the game begins with 2
   being killed and then n+2, n+4, */
ll josephus(ll n, ll k) {
    if (n == 1)
        return 1;
    else
        return (josephus(n - 1, k) + k - 1) % n + 1;
}

/* Another Way to compute the last position to be
   killed - O(d * log n) */
ll josephus(ll n, ll d) {
    ll K = 1;
    while (K <= (d - 1) * n)
        K = (d * K + d - 2) / (d - 1);
    return d * n + 1 - K;
}

```

#### 4.12. Large Number GCD

```

ll gcd(ll a, ll b) {
    if (!a)
        return b;
    return gcd(b % a, a);
}

ll reduceB(ll a, char b[]) {
    ll mod = 0;
    for (int i = 0; i < strlen(b); i++)
        mod = (mod * 10 + b[i] - '0') % a;
    return mod; // return modulo
}

ll gcdLarge(ll a, char b[]) {

```

```

    ll num = reduceB(a, b);
    return gcd(a, num);
}

int main() {
    ll a = 1221;
    char b[] = "
1234567891011121314151617181920212223242526272829
";
    if (a == 0)
        cout << b << endl;
    else
        cout << gcdLarge(a, b) << endl;
    return 0;
}

```

#### 4.13. Lengenders Formula

```

// n and a prime number p, find the largest x such
// that px divides n !(factorial)in O(logn).
#include <bits/stdc++.h>
using namespace std;
int legendre(long long n, long long p){ int ans = 0;
    while (n){ans += n / p; n /= p;}
    return ans;}

```

#### 4.14. Modular for Subtraction & Multiplication

```

int vagsesh = (a % m - b % m + m) % m; // For
Substraction
long long res = 1; // For Multiplicaion
for (int i = 1; i <= n; i++)res = (res * a) % m;
cout << res << endl;

```

#### 4.15. Optimized Sieve

```

// (Fast Sieve, Using bit set, Works till 10^8 in
// less than 1s, Memory Complexity: O (n/64))

const int N = 1e8 + 9;
bitset<N> f;
int32_t main() {
    int n = N - 9;
    vector<int> primes;
    f[1] = true;

    for (int i = 2; i * i <= n; i++)
        if (!f[i])
            for (int j = i * i; j <= n; j += i)
                f[j] = true;

    for (int i = 2; i <= n; i++)
        if (!f[i])

```

```

        primes.push_back(i);

    cout << primes.size() << '\n';
    return 0;
}

```

#### 4.16. Lower Prime Factorization

```

// Linear Sieve for prime factorization
//time complexity: O(n)
//space complexity: O(n)
const int N = 10000000; // up to 1e7
vector<int> lp; // lowest prime factor
vector<int> primes; // list of primes
void linear_sieve(int n) {
    lp = vector<int>(n + 1, 0);
    for (int i = 2; i <= n; i++) {
        if (lp[i] == 0) { // i is prime
            lp[i] = i;
            primes.push_back(i);
        }
        for (int p : primes) {
            if (p > lp[i] || 1LL * i * p > n) break;
            lp[i * p] = p;
        }
    }
}

```

#### 4.17. Prime or Not

```

bool prime(int n) {
    if(n < 2) return false;
    if(n == 2) return true;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0)
            return false;
    }
    return true;}

```

#### 4.18. Number of Divisor A Number Have

```

long long numDivisors[1000005] = {};
void calculateDivisors(long long numDivisors[])
{ //calculate number of divisor
    for (int i = 1; i < 1000005; i++) {
        for (int j = i; j < 1000005; j += i) {
            numDivisors[j]++;
        }
    }
}

```

## 4.19. Prime Factors

```
void calcFp(int n) {
    fp = vector<int>(n + 1, 0);
    for (int i = 2; i <= n; i++) {
        if (fp[i] == 0) { // i is prime
            for (int j = i; j <= n; j += i) {
                if (fp[j] == 0) fp[j] = i;
            }
        }
    }

    int main () {
        calcFp((int)1e6);
        while (1) {
            int x;
            cin >> x;
            if (x == 0) break;
            vector<int> primes, pw;
            while (x > 1) {
                int p = fp[x];
                int cnt = 0;
                while (x % p == 0) {
                    x /= p;
                    cnt++;
                }
                primes.push_back(p);
                pw.push_back(cnt);
            }
            for (int i = 0; i < primes.size(); i++) {
                cout << primes[i] << "^" << pw[i] << endl;
            }
        }
        return 0;
    }
}
```

## 4.20. Formulas Using Prime Factorization Math

```
// if a number prime factor = p1^x1 * p2^x2 * p3^x3
// ...pn^xn

// then number of divisors = (x1+1)*(x2+1)*(x3+1)
// ...*(xn+1)

// total divisor sum = (p1^(x1+1)-1/p1-1) * (p2^(x2
// +1)-1/p2-1) * (p3^(x3+1)-1/p3-1) * ... * (pn^(xn
// +1)-1/pn-1)
// -----sum = sum-N

//product of divisor = n^(number of divisors/2)
// b/a = b*a^(-1);
// a^(1)==a^(MOD2)%MOD;

// product of divisor
int div_prod = expo(div_prod, k[i] + 1) *
    expo(expo(p[i], (k[i] * (k[i] + 1) / 2)),
        div_cnt2) % MOD;
int div_cnt2 = div_cnt2 * (k[i] + 1) % (MOD - 1);
```

## 4.21. Sum of Divisor

```
// Sum of divisors using the prime factorization of
// n.
long long SumOfDivisors(long long num){ long long
    total = 1;
    for (int i = 2; (long long)i * i <= num; i++){
        if (num % i == 0){ int e = 0;
            do{ e++; num /= i;} while (num%i == 0);
            long long sum = 0, pow = 1;
            do{sum += pow; pow *= i;}while(e-- > 0);
            total *= sum;}}
    if (num > 1) total *= (1 + num); return total;}
```

## 4.22. Sieve

```
// primes which are less than n in O(nloglog(n))
const int N = 1e7 + 9;
bool f[N];

int32_t main() {
    int n = N - 9;
    vector<int> primes;
    f[1] = true;
    for (int i = 2; i <= n; i++) {
        if (!f[i]) {
            primes.push_back(i);
            for (int j = i + i; j <= n; j += i)
                f[j] = true;
        }
    }
    cout << primes.size() << '\n';
    return 0;}
```

## 4.23. n! Trailing Zero

```
int findTrailingZeros(int n){
    if (n < 0) // Negative Number Edge Case
        return -1;
    int count = 0;
    for (int i=5;n/i >= 1; i *= 5) count += n / i;
    return count;}
```

## 4.24. Number of Divisors 1 to n

```
int d[104];
int32_t main(){
    int n = 100;
    for (int i = 1; i <= n; i++){for (int j = i; j <=
        n; j += i)d[j]++; // d[j] += i // for sum of
        divisors
```

```
for (int i = 1; i <= n; i++)cout << d[i] << " ";
return 0;}
```

# 5. Mixed

## 5.1. Run Code Technique

```
//C++ Shell for run
g++ file_name.cpp -o a
.\a

//Python shell for run
python -u "g:\foldername\filename.py"
```

```
//first io
ios::sync_with_stdio(false);
cin.tie(nullptr);
```

```
//input,output in file
#ifdef ONLINE_JUDGE
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
#endif
```

## 5.2. i128 Bit

```
#include <bits/stdc++.h>
#define i128 __int128_t
using namespace std;
int32_t main() {
    i128 x = 1;
    int y = (int)(x+1);
    cout<<y<<endl;
    return 0;}
// i128 can be store not print
// range -2^127 to 2^127-1
```

## 5.3. Base Conberstion

```
string base_convert(int n, int b) {
    string s = "";
    while (n > 0) {
        s = to_string(n % b) + s;
        n /= b;
    }
    return s;
}

int convert_to_decimal(string s, int base) {
    int n = 0, power = 1;
    for (int i = (int)s.size() - 1; i >= 0; i--) {
        n += power * (s[i] - '0');
```

```

    power *= base;
}
return n;}
```

## 5.4. Bitset

```

// Declaration with size N
bitset<100005> bs; // all bits 0 initially
```

```

// Set all bits to 1
bs.set();
```

```

// Set i-th bit
bs.set(i);
```

```

// Reset i-th bit (make 0)
bs.reset(i);
```

```

// Reset all bits
bs.reset();
```

```

// Flip i-th bit
bs.flip(i);
```

```

// Flip entire bitset
bs.flip();
```

```

// Check if i-th bit is set
bool f = bs.test(i);
```

```

// Count number of 1s
int cnt = bs.count();
```

```

// Check if all bits are 1
bool ok1 = bs.all();
```

```

// Check if any bit is 1
bool ok2 = bs.any();
```

```

// Check if none bit is 1
bool ok3 = bs.none();
```

```

// Convert bitset to unsigned long long
unsigned long long x = bs.to_ullong();
```

```

// Convert bitset to string
string s = bs.to_string();
```

## 5.5. Bit Operation

```

const int inf = numeric_limits<int>::max() - 5;
#define int long long
```

```

struct ST {
```

```

    int right_shift(int a, int b) { return (a >> b); }
    int left_shift(int a, int b) { return (a << b); }
    int bitwise_not(int a) { return (~a); }
    int ON_BIT(int a) { return __builtin_popcount(a); }
    int leading_zero(int a) { return __builtin_clz(a); }
    int trailing_zero(int a) { return __builtin_ctz(a); }
    int LSB(int a) { return (a & 1); }
    bool kth_bit(int a, int k) { return (a & (1 << k)); }
    int msb(int a) { return a ? 32 - __builtin_clz(a) : 0; }
```

```

    string base_convert(int n, int b) {
        string s = "";
        while (n > 0) {
            s = to_string(n % b) + s;
            n /= b;
        }
        return s;
    }
```

```

    int convert_to_decimal(string s, int base) {
        int n = 0, power = 1;
        for (int i = (int)s.size() - 1; i >= 0; i--) {
            n += power * (s[i] - '0');
            power *= base;
        }
        return n;
    }
} bit;
```

```

// other
// Count number of set bits (1s)
int pc = __builtin_popcount(x); // for int
int pcLL = __builtin_popcountll(x); // for long long
```

```

// Count trailing zeros
int tz = __builtin_ctz(x);
int tzLL = __builtin_ctzll(x);
```

```

// Count leading zeros
int lz = __builtin_clz(x);
int lzLL = __builtin_clzll(x);
```

```

// Check power of two
bool isP2 = (x > 0) && ((x & (x - 1)) == 0);
```

## 5.6. Merge Sort

```

// Merge-sort with inversion count - O(nlog n)
int n, inv;
vector<int> v, ans;
```

```

void mergesort(int l, int r, vector<int> &v) {
```

```

    if (l == r) return;
    int mid = (l + r) / 2;
    mergesort(l, mid, v);
    mergesort(mid + 1, r, v);
```

```

    int i = l, j = mid + 1, k = l;
    while (i <= mid || j <= r) {
        if (i <= mid && (j > r || v[i] <= v[j]))
            ans[k++] = v[i++];
        else
            ans[k++] = v[j++], inv += j - k;
    }
```

```

    for (int i = l; i <= r; i++)
        v[i] = ans[i];
}
```

```

// in main: ans.resize(v.size());
```

## 5.7. STL

```

auto [a, b] = p; pair<int, int> p; pair<int, pair<
    int, int>> p3;
```

```

deque<int> dq; dq.push_front(1); dq.push_back(3);
pop_front(); pop_back();
```

```

stack<int> st; st.push(2); st.pop(); st.top(); st.
    size();
```

```

queue<int> q; q.push(5); q.pop(); q.front(); q.back();
    size(); empty();
```

```

// Max priority first (largest value removed first)
priority_queue<long long> pq_max;
```

```

// Min priority first (smallest value removed first)
priority_queue<long long, vector<long long>, greater<
    long long>> pq_min;
```

```

// Clear both
pq_max = priority_queue<long long>();
pq_min = priority_queue<long long, vector<long long>
    >, greater<long long>>();
```

```

map<int, int> mp1; map<int, pair<int, int>> mp2; mp2
    [0].first; mp2[0].second;
int index = lower_bound(v.begin(), v.end(), val) - v
    .begin();
int index = upper_bound(v.begin(), v.end(), val) - v
    .begin();
auto it = s.lower_bound(6); *it; it--;
auto it = s.upper_bound(6); *it; it--;
```

## 5.8. Sqrt Decomposition

```
// Square Root Decomposition (Mos Algorithm) - O(n^(3/2))
const int N = 1e5 + 1, SQ = 500;
int n, m, v[N];

void add(int p) { /* add value to aggregated data structure */ }
void rem(int p) { /* remove value from aggregated data structure */ }

struct query {
    int i, l, r, ans;
} qs[N];

bool c1(query a, query b) {
    if (a.l / SQ != b.l / SQ) return a.l < b.l;
    return a.l / SQ & l ? a.r > b.r : a.r < b.r;
}
bool c2(query a, query b) { return a.i < b.i; }
/* inside main */
int l = 0, r = -1;
sort(qs, qs + m, c1);

for (int i = 0; i < m; ++i) {
    query &q = qs[i];
    while (r < q.r) add(v[++r]);
    while (r > q.r) rem(v[r--]);
    while (l < q.l) rem(v[l++]);
    while (l > q.l) add(v[--l]);
    q.ans = /* calculate answer */;
}

sort(qs, qs + m, c2); // sort to original order
```

## 5.9. Binary Search

```
ll binary_search_last_true(ll lo, ll hi, Pred pred)
{
    while (lo < hi) {
        ll mid = lo + (hi - lo + 1) / 2; // upper mid
        if (pred(mid)) lo = mid;
        else hi = mid - 1;
    }
    return lo;
}
```

## 5.10. Ternary Search

```
// Ternary Search - O(log(n))
// Max version, for minimum version just change signals
ll ternary_search(ll l, ll r) {
    while (r - l > 3) {
        ll m1 = (l + r) / 2;
        ll m2 = (l + r) / 2 + 1;
```

```
ll f1 = f(m1), f2 = f(m2); // if(f1 > f2) l = m1;
if (f1 < f2)
    l = m1;
else
    r = m2;
}
ll ans = 0;
for (int i = l; i <= r; i++) {
    ll tmp = f(i); // ans = min(ans, tmp);
    ans = max(ans, tmp);
}
return ans;
}
// Faster version - 300 iterations up to 1e-6 precision
double ternary_search(double l, double r, int No = 300) {
    for(int i = 0; i < No; i++){
        while (r - l > EPS) {
            double m1 = l + (r - l) / 3;
            double m2 = r - (r - l) / 3;
            // if (f(m1) > f(m2))
            if (f(m1) < f(m2))
                l = m1;
            else
                r = m2;
        }
        return f(l);
    }
}
```

## 5.11. python

```
# reopen
import sys
sys.stdout = open(out, w)
sys.stdin = open(in, r)
//Dummy example
R = lambda: map(int, input().split())
n, k = R(),
v, t = [], [0]*n
for p, c, i in sorted(zip(R(), R(), range(n))):
    t[i] = sum(v)+c
    v += [c]
    v = sorted(v)[:k-1]
    if len(v) > k:
        v.pop()
print(' '.join(map(str, t)))
```

# 6. Strings

## 6.1. All String Operation

```
string s = "abcdef";
//Substring
```

```
string sub = s.substr(0, 5+1);

//Find substring or character
int pos = s.find("cd"); // -1 if not found
int posChar = s.find('e');

// Compare strings
bool eq = (s1 == s2);
bool lt = (s1 < s2); // lexicographical comparison

// Convert int <-> string
string numStr = to_string(45);
int x = stoi("123");
long long y = stoll("123456789"); // string to long long

// Convert float <-> string
double d = 3.14;
string ds = to_string(d);
double dd = stod("2.718");

// Uppercase / Lowercase
string t = "AbCd";
transform(t.begin(), t.end(), t.begin(), ::toupper);
transform(t.begin(), t.end(), t.begin(), ::tolower);

//Reverse string
reverse(rev.begin(), rev.end());

//Remove spaces
string str = "a b c d";
str.erase(remove(str.begin(), str.end(), ' '), str.end());
```

```
// Subsequence check
string small = "ace";
int i = 0;
for (char c : s) if (i < small.size() && small[i] == c) i++;
bool isSubseq = (i == small.size());
```

## 6.2. Single Hashing

```
//single string hashing
struct StringHash {
    const long long MOD = 1000000007; // large prime
    const long long P = 31; // base (for lowercase letters)
    vector<long long> prefix, power;

    StringHash(const string &s) {
        int n = s.size();
        prefix = vector<long long>(n+1, 0);
        power = vector<long long>(n+1, 1);

        for (int i = 1; i <= n; i++) {
            power[i] = (power[i-1] * P) % MOD;
```

```

        prefix[i] = (prefix[i-1] * P + (s[i-1] - 'a'
            + 1)) % MOD;
    }
}

// get hash of substring [l, r] (0-indexed
// inclusive)
long long getHash(int l, int r) {
    long long res = (prefix[r+1] - (prefix[l] *
        power[r-l+1]) % MOD + MOD) % MOD;
    return res;
}
};

```

### 6.3. Single Hashing

```

// double string hashing to further reduce collision
struct StringHash {
    // two large primes to reduce collision
    const long long MOD1 = 1000000007;
    const long long MOD2 = 1000000009;
    const long long P1 = 31;
    const long long P2 = 37;

    vector<long long> prefix1, prefix2;
    vector<long long> power1, power2;

    StringHash(const string &s) {
        int n = s.size();
        prefix1.resize(n+1, 0);
        prefix2.resize(n+1, 0);
        power1.resize(n+1, 1);
        power2.resize(n+1, 1);

        for (int i = 1; i <= n; i++) {
            int val = s[i-1]; // works for all ASCII
            // characters
            power1[i] = (power1[i-1] * P1) % MOD1;
            power2[i] = (power2[i-1] * P2) % MOD2;

            prefix1[i] = (prefix1[i-1] * P1 + val) %
                MOD1;
            prefix2[i] = (prefix2[i-1] * P2 + val) %
                MOD2;
        }
    }

    // get double hash of substring [l, r] inclusive
    pair<long long, long long> getHash(int l, int r) {
        long long h1 = (prefix1[r+1] - (prefix1[l] *
            power1[r-l+1]) % MOD1 + MOD1) % MOD1;
        long long h2 = (prefix2[r+1] - (prefix2[l] *
            power2[r-l+1]) % MOD2 + MOD2) % MOD2;
        return {h1, h2};
    }
}

```

```

// quick compare of substrings from two
// StringHash objects
bool isEqual(int l1, int r1, int l2, int r2,
    StringHash &other) {
    return getHash(l1, r1) == other.getHash(l2, r2);
}
};

```

### 6.4. Largest Substring More than K

```

int n;
int max_oc(int len) {
    map<pair<int, int>, int> mp;
    for (int i = 0; i + len - 1 < n; i++)
        mp[get_hash(i, i + len - 1)]++;

    int ans = 0;
    for (auto [x, y] : mp)
        ans = max(ans, y);

    return ans;
}

int32_t main() {
    prec();
    string s;
    cin >> s;
    build(s);

    int k;
    cin >> k;
    n = s.size();
    int l = 1, r = s.size(), ans = -1;
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (max_oc(mid) >= k)
            ans = mid, l = mid + 1;
        else
            r = mid - 1;
    }
    cout << ans << '\n';
    return 0;
}

```

### 6.5. Longest Common Prefix Of Two Substrings

```

int lcp(int i, int j, int x, int y) { // O(log n)
    int l = 1, r = min(j - i + 1, y - x + 1), ans = 0;
    while (l <= r) {
        int mid = l + r >> 1;
        if (get_hash(i, i + mid - 1) == get_hash(x, x
            + mid - 1)) {
            ans = mid;
            l = mid + 1;
        }
    }
    return ans;
}

```

```

    } else {
        r = mid - 1;
    }
}
return ans;}

/*
given a string s of size n and q queries of type i,
j, x, y.
find the LCP of Substring s[i...j] and s[x...y].
1 <= n, q <= 1e5
*/

```

### 6.6. Longest Common Substring

```

const int N = 1e5 + 9;
const int p1 = 137, mod1 = 127657753, p2 = 277, mod2 =
    987654319;

int power(long long n, long long k, int mod){
    int ans = 1 % mod; n %= mod;
    if (n < 0) n += mod;
    while (k){
        if (k & 1) ans = (long long)ans * n % mod;
        n = (long long)n * n % mod;
        k >>= 1;
    }
    return ans;
}

int ip1, ip2;
pair<int, int> pw[N], ipw[N];

void prec(){ // O(n)
    pw[0] = {1, 1};
    for (int i = 1; i < N; i++){
        pw[i].first = 1LL * pw[i-1].first * p1 %
            mod1;
        pw[i].second = 1LL * pw[i-1].second * p2 %
            mod2;
    }
    ip1 = power(p1, mod1 - 2, mod1);
    ip2 = power(p2, mod2 - 2, mod2);
    ipw[0] = {1, 1};
    for (int i = 1; i < N; i++){
        ipw[i].first = 1LL * ipw[i-1].first * ip1 %
            mod1;
        ipw[i].second = 1LL * ipw[i-1].second * ip2 %
            mod2;
    }
}

pair<int, int> string_hash(string s){ // O(n)
    int n = s.size();
    pair<int, int> hs({0, 0});
    for (int i = 0; i < n; i++){
        hs.first += 1LL * s[i] * pw[i].first % mod1;
        hs.first %= mod1;
        hs.second += 1LL * s[i] * pw[i].second % mod2;
    }
}

```

```

        hs.second %= mod2;
    }
    return hs;
}
struct Hashing{
    pair<int, int> pref[N];
    void build(string s){ // O(n)
        int n = s.size();
        for (int i = 0; i < n; i++){
            pref[i].first = 1LL * s[i] * pw[i].first %
            mod1;
            if (i) pref[i].first = (pref[i].first +
            pref[i - 1].first) % mod1;
            pref[i].second = 1LL * s[i] * pw[i].second
            % mod2;
            if (i) pref[i].second = (pref[i].second +
            pref[i - 1].second) % mod2;
        }
    }
    pair<int, int> get_hash(int i, int j){ // O(1)
        pair<int, int> hs({0, 0});
        hs.first = pref[j].first;
        if (i) hs.first = (hs.first - pref[i - 1].
        first + mod1) % mod1;
        hs.first = 1LL * hs.first * ipw[i].first %
        mod1;
        hs.second = pref[j].second;
        if (i) hs.second = (hs.second - pref[i - 1].
        second + mod2) % mod2;
        hs.second = 1LL * hs.second * ipw[i].second %
        mod2;
        return hs;
    }
} A, B;

int n;
string a, b;
string res;

bool ok(int k){ // is there a k length substring
that occurs in both a and b
    set<pair<int, int>> substring_hashes_in_a;
    for (int i = 0; i + k - 1 < n; i++)
        substring_hashes_in_a.insert(A.get_hash(i, i +
        k - 1));
    for (int i = 0; i + k - 1 < n; i++){
        auto substring_hash_in_b = B.get_hash(i, i + k
        - 1);
        if (substring_hashes_in_a.find(
        substring_hash_in_b) !=
        substring_hashes_in_a.end()){
            res = b.substr(i, k);
            return true;
        }
    }
    return false;
}

int32_t main(){
    prec();
    cin >> n;

```

```

    cin >> a >> b;
    A.build(a);
    B.build(b);
    int l = 1, r = n, ans = 0;
    while (l <= r){
        int mid = (l + r) / 2;
        if (ok(mid)){ ans = mid; l = mid + 1;}
        else { r = mid - 1; }
    }
    ok(ans);
    cout << res << '\n'; // O(n log^2 n)
    return 0;
} // Find the Longest Common Substring of Two Strings

```

## 6.7. Pattern Matching

```

const int N = 1e6 + 9;
const int p1 = 137, mod1 = 127657753, p2 = 277, mod2
= 987654319;

int power(long long n, long long k, int mod){
    int ans = 1 % mod;
    n %= mod;
    if (n < 0) n += mod;
    while (k){
        if (k & 1) ans = (long long)ans * n % mod;
        n = (long long)n * n % mod;
        k >>= 1;
    }
    return ans;
}

int ip1, ip2;
pair<int, int> pw[N], ipw[N];

void prec(){
    pw[0] = {1, 1};
    for (int i = 1; i < N; i++){
        pw[i].first = 1LL * pw[i - 1].first * p1 %
        mod1;
        pw[i].second = 1LL * pw[i - 1].second * p2 %
        mod2;
    }
    ip1 = power(p1, mod1 - 2, mod1);
    ip2 = power(p2, mod2 - 2, mod2);
    ipw[0] = {1, 1};
    for (int i = 1; i < N; i++){
        ipw[i].first = 1LL * ipw[i - 1].first * ip1 %
        mod1;
        ipw[i].second = 1LL * ipw[i - 1].second * ip2
        % mod2;
    }
}

pair<int, int> string_hash(string s){
    int n = s.size();
    pair<int, int> hs({0, 0});
    for (int i = 0; i < n; i++){

```

```

        hs.first += 1LL * s[i] * pw[i].first % mod1;
        hs.first %= mod1;
        hs.second += 1LL * s[i] * pw[i].second % mod2;
        hs.second %= mod2;
    }
    return hs;
}

pair<int, int> pref[N];

void build(string s){
    int n = s.size();
    for (int i = 0; i < n; i++){
        pref[i].first = 1LL * s[i] * pw[i].first %
        mod1;
        if (i) pref[i].first = (pref[i].first + pref[i
        - 1].first) % mod1;
        pref[i].second = 1LL * s[i] * pw[i].second %
        mod2;
        if (i) pref[i].second = (pref[i].second + pref
        [i - 1].second) % mod2;
    }
}

pair<int, int> get_hash(int i, int j){
    assert(i <= j);
    pair<int, int> hs({0, 0});
    hs.first = pref[j].first;
    if (i) hs.first = (hs.first - pref[i - 1].first +
    mod1) % mod1;
    hs.first = 1LL * hs.first * ipw[i].first % mod1;
    hs.second = pref[j].second;
    if (i) hs.second = (hs.second - pref[i - 1].
    second + mod2) % mod2;
    hs.second = 1LL * hs.second * ipw[i].second %
    mod2;
    return hs;
}

int32_t main(){
    prec();
    string a, b;
    cin >> a >> b;
    build(a);
    int ans = 0, n = a.size(), m = b.size();
    auto hash_b = string_hash(b);
    for (int i = 0; i + m - 1 < n; i++)
        ans += get_hash(i, i + m - 1) == hash_b;
    cout << ans << '\n';
    return 0;
}
/*
Given a string a and pattern b, count the number of
positions where the pattern occurs in the string
.

Input:
saippuakauppias
pp
Output:2*/

```



## 6.8. Two Strings Are Equal Or Not

```
const int p = 137, mod = 1e9 + 7;
const int N = 1e5 + 9;
int pw[N];

void prec() {
    pw[0] = 1;
    for (int i = 1; i < N; i++)
        pw[i] = 1LL * pw[i - 1] * p % mod;
}

int get_hash(string s) {
    int n = s.size();
    int hs = 0;
    for (int i = 0; i < n; i++) {
        hs += 1LL * s[i] * pw[i] % mod;
        hs %= mod;
    }
    return hs;
}

int32_t main() {
    prec();
    string a, b;
    cin >> a >> b;
    cout << (get_hash(a) == get_hash(b)) << '\n';
    return 0; }
```

## 6.9. Lines

```
#include "basics.cpp"
// functions tested at: https://codeforces.com/group/3qadGzUdR4/contest/101706/problem/B
// WARNING: all distance functions are not realizing sqrt operation
// Suggestion: for line intersections check line_line_intersection and then use compute_line_intersection

point project_point_line(point c, point a, point b)
{
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    return a + (b - a) * dot(c - a, b - a) / dot(b - a, b - a);
}

point project_point_ray(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (le(r, 0)) return a;
    return a + (b - a) * r;
}

point project_point_segment(point c, point a, point b) {
    ld r = dot(b - a, b - a);
```

```

    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (le(r, 0)) return a;
    if (ge(r, 1)) return b;
    return a + (b - a) * r;
}

ld distance_point_line(point c, point a, point b) {
    return c.dist2(project_point_line(c, a, b));
}

ld distance_point_ray(point c, point a, point b) {
    return c.dist2(project_point_ray(c, a, b));
}

ld distance_point_segment(point c, point a, point b)
{
    return c.dist2(project_point_segment(c, a, b));
}

// not tested
ld distance_point_plane(ld x, ld y, ld z, ld a, ld b
, ld c, ld d) {
    return fabs(a * x + b * y + c * z - d) / sqrt(a *
        a + b * b + c * c);
}

bool lines_parallel(point a, point b, point c, point
    d) {
    return fabs(cross(b - a, d - c)) < EPS;
}

bool lines_collinear(point a, point b, point c,
    point d) {
    return lines_parallel(a, b, c, d) && fabs(cross(a
        - b, a - c)) < EPS && fabs(cross(c - d, c -
        a)) < EPS;
}

point lines_intersect(point p, point q, point a,
    point b) {
    point r = q - p, s = b - a, c(p % q, a % b);
    if (eq(r % s, 0)) return point(LINF, LINF);
    return point(point(r.x, s.x) % c, point(r.y, s.y)
        % c) / (r % s);
}

// be careful: test line_line_intersection before
// using this function
point compute_line_intersection(point a, point b,
    point c, point d) {
    b = b - a;
    d = c - d;
    c = c - a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b * cross(c, d) / cross(b, d);
}

bool line_line_intersect(point a, point b, point c,
    point d) {
    if (!lines_parallel(a, b, c, d)) return true;

```

```

    if (lines_collinear(a, b, c, d)) return true;
    return false;
}

// ray in direction a -> b, c -> d
bool ray_ray_intersect(point a, point b, point c,
    point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS || b.
        dist2(c) < EPS || b.dist2(d) < EPS) return
        true;
    if (lines_collinear(a, b, c, d)) {
        if (ge(dot(b - a, d - c), 0)) return true;
        if (ge(dot(a - c, d - c), 0)) return true;
        return false;
    }
    if (!line_line_intersect(a, b, c, d)) return
        false;
    point inters = lines_intersect(a, b, c, d);
    if (ge(dot(inters - c, d - c), 0) && ge(dot(
        inters - a, b - a), 0)) return true;
    return false;
}

bool segment_segment_intersect(point a, point b,
    point c, point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS || b.
        dist2(c) < EPS || b.dist2(d) < EPS) return
        true;
    int d1, d2, d3, d4;
    d1 = direction(a, b, c);
    d2 = direction(a, b, d);
    d3 = direction(c, d, a);
    d4 = direction(c, d, b);
    if (d1 * d2 < 0 and d3 * d4 < 0) return 1;
    return a.on_seg(c, d) or b.on_seg(c, d) or c.
        on_seg(a, b) or d.on_seg(a, b);
}

bool segment_line_intersect(point a, point b, point
    c, point d) {
    if (!line_line_intersect(a, b, c, d)) return
        false;
    point inters = lines_intersect(a, b, c, d);
    if (inters.on_seg(a, b)) return true;
    return false;
}

// ray in direction c -> d
bool segment_ray_intersect(point a, point b, point c
    , point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS || b.
        dist2(c) < EPS || b.dist2(d) < EPS) return
        true;
    if (lines_collinear(a, b, c, d)) {
        if (c.on_seg(a, b)) return true;
        if (ge(dot(d - c, a - c), 0)) return true;
        return false;
    }
    if (!line_line_intersect(a, b, c, d)) return
        false;

```



```

point inters = lines_intersect(a, b, c, d);
if (!inters.on_seg(a, b)) return false;
if (ge(dot(inters - c, d - c), 0)) return true;
return false;
}

// ray in direction a -> b
bool ray_line_intersect(point a, point b, point c,
point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS || b.
        dist2(c) < EPS || b.dist2(d) < EPS) return
        true;
    if (!line_line_intersect(a, b, c, d)) return
        false;
    point inters = lines_intersect(a, b, c, d);
    if (!line_line_intersect(a, b, c, d)) return
        false;
    if (ge(dot(inters - a, b - a), 0)) return true;
    return false;
}

ld distance_segment_line(point a, point b, point c,
point d) {
    if (segment_line_intersect(a, b, c, d)) return 0;
    return min(distance_point_line(a, c, d),
        distance_point_line(b, c, d));
}

ld distance_segment_ray(point a, point b, point c,
point d) {
    if (segment_ray_intersect(a, b, c, d)) return 0;
    ld min1 = distance_point_segment(c, a, b);
    ld min2 = min(distance_point_ray(a, c, d),
        distance_point_ray(b, c, d));
    return min(min1, min2);
}

ld distance_segment_segment(point a, point b, point
c, point d) {
    if (segment_segment_intersect(a, b, c, d)) return
        0;
    ld min1 = min(distance_point_segment(c, a, b),
        distance_point_segment(d, a, b));
    ld min2 = min(distance_point_segment(a, c, d),
        distance_point_segment(b, c, d));
    return min(min1, min2);
}

ld distance_ray_line(point a, point b, point c,
point d) {
    if (ray_line_intersect(a, b, c, d)) return 0;
    ld min1 = distance_point_line(a, c, d);
    return min1;
}

ld distance_ray_ray(point a, point b, point c, point
d) {
    if (ray_ray_intersect(a, b, c, d)) return 0;
    ld min1 = min(distance_point_ray(c, a, b),
        distance_point_ray(a, c, d));
    return min1;
}

```

```

}

ld distance_line_line(point a, point b, point c,
point d) {
    if (line_line_intersect(a, b, c, d)) return 0;
    return distance_point_line(a, c, d);
}

6.10. Polygons

#include "basics.cpp"
#include "lines.cpp"

// Monotone chain O(nlog(n))
#define REMOVE_REDUNDANT
#ifdef REMOVE_REDUNDANT
bool between(const point &a, const point &b, const
point &c) {
    return (fabs(area_2(a, b, c)) < EPS &&
        (a.x - b.x) * (c.x - b.x) <= 0 &&
        (a.y - b.y) * (c.y - b.y) <= 0);
}
#endif

// new change: <= 0 / >= 0 became < 0 / > 0 (yet to
be tested)
void convex_hull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end
        ());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area_2(up[up.size() -
            2], up.back(), pts[i]) > 0) up.pop_back();
        while (dn.size() > 1 && area_2(dn[dn.size() -
            2], dn.back(), pts[i]) < 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int)up.size() - 2; i >= 1; i--) pts
        .push_back(up[i]);
#ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size() - 2], dn[dn.size() -
            1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0],
        dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
}

```

```

#endif
}

// avoid using long double for comparisons, change
type and remove division by 2
type compute_signed_area(const vector<point> &p) {
    type area = 0;
    for (int i = 0; i < p.size(); i++) {
        int j = (i + 1) % p.size();
        area += p[i].x * p[j].y - p[j].x * p[i].y;
    }
    return area;
}

ld compute_area(const vector<point> &p) {
    return fabs(compute_signed_area(p) / 2.0);
}

ld compute_perimeter(vector<point> &p) {
    ld per = 0;
    for (int i = 0; i < p.size(); i++) {
        int j = (i + 1) % p.size();
        per += p[i].dist(p[j]);
    }
    return per;
}

// not tested
// TODO: test this code. This code has not been
// tested, please do it before proper use.
// http://codeforces.com/problemset/problem/975/E is
// a good problem for testing.
point compute_centroid(vector<point> &p) {
    point c(0, 0);
    ld scale = 6.0 * compute_signed_area(p);
    for (int i = 0; i < p.size(); i++) {
        int j = (i + 1) % p.size();
        c = c + (p[i] + p[j]) * (p[i].x * p[j].y - p[j]
            .x * p[i].y);
    }
    return c / scale;
}

// TODO: test this code. This code has not been
// tested, please do it before proper use.
// http://codeforces.com/problemset/problem/975/E is
// a good problem for testing.
point centroid(vector<point> &v) {
    int n = v.size();
    type da = 0;
    point m, c;
    for (point p : v) m = m + p;
    m = m / n;
    for (int i = 0; i < n; ++i) {
        point p = v[i] - m, q = v[(i + 1) % n] - m;
        type x = p % q;
        c = c + (p + q) * x;
        da += x;
    }
    return c / (3 * da);
}

```

```
// O(n^2)
bool is_simple(const vector<point> &p) {
    for (int i = 0; i < p.size(); i++) {
        for (int k = i + 1; k < p.size(); k++) {
            int j = (i + 1) % p.size();
            int l = (k + 1) % p.size();
            if (i == 1 || j == k) continue;
            if (segment_segment_intersect(p[i], p[j], p[k], p[l])) return false;
        }
    }
    return true;
}

bool point_in_triangle(point a, point b, point c,
    point cur) {
    ll s1 = abs(cross(b - a, c - a));
    ll s2 = abs(cross(a - cur, b - cur)) + abs(cross(
        b - cur, c - cur)) + abs(cross(c - cur, a -
        cur));
    return s1 == s2;
}

void sort_lex_hull(vector<point> &hull) {
    if (compute_signed_area(hull) < 0) reverse(hull.
        begin(), hull.end());
    int n = hull.size(); // Sort hull by x
    int pos = 0;
    for (int i = 1; i < n; i++) if (hull[i] < hull[
        pos]) pos = i;
    rotate(hull.begin(), hull.begin() + pos, hull.end
        ());
}

// determine if point is inside or on the boundary
// of a polygon (O(logn))
bool point_in_convex_polygon(vector<point> &hull,
    point cur) {
    int n = hull.size();
    // Corner cases: point outside most left and most
    // right wedges
    if (cur.dir(hull[0], hull[1]) != 0 && cur.dir(
        hull[0], hull[1]) != hull[n - 1].dir(hull[0],
        hull[1])) return false;
    if (cur.dir(hull[0], hull[n - 1]) != 0 && cur.dir(
        hull[0], hull[n - 1]) != hull[1].dir(hull
        [0], hull[n - 1])) return false;
    // Binary search to find which wedges it is
    // between
    int l = 1, r = n - 1;
    while (r - l > 1) {
        int mid = (l + r) / 2;
        if (cur.dir(hull[0], hull[mid]) <= 0) l = mid;
        else r = mid;
    }
    return point_in_triangle(hull[l], hull[l + 1],
        hull[0], cur);
}
```

```
// determine if point is on the boundary of a
// polygon (O(N))
bool point_on_polygon(vector<point> &p, point q) {
    for (int i = 0; i < p.size(); i++)
        if (q.dist2(project_point_segment(p[i], p[(i +
            1) % p.size()], q)) < EPS) return true;
    return false;
}
```

## 7. Mathematical Formulas

### 7.1. Math Formula

summation a to b = (ba+1)(a+b)/2

Orientation of 3 point  
 // a(x,y), b(x,y), c(x,y), d(x,y)  
 // slop ab = (yb-ya)/(xb-xa);  
 // slop bc = (yc-yb)/(xc-xb);  
 #include <bits/stdc++.h>  
 using namespace std;  
 int main() {  
 // Orientation of 3 points(a,b,c)->  
 // Counterclockwise (abc<0): ab slop uoper c point  
 // Clockwise (abc>0): ab slop nicha c point  
 // Collinear(abc=0): ab slop soja c point  
  
 //how to calculate  
 ABCslop = slopAB - slopBC  
 = (By-Ay)/(Bx-Ax) - (By-Cy)/(Bx-Cx);  
 = ((By-Ay)\*(Cx-Bx) - (Cy-By)\*(Bx-Ax));  
 }

### 7.2. Matrices and Determinants

1. The number of rows or columns (Order) in a square matrix is called its order.
2. A matrix whose all elements are zero is called a null matrix.
3. Matrix multiplication  $AB$  is possible only when the number of columns in matrix  $A$  is equal to the number of rows in matrix  $B$ .
4. If  $A$  is a non-singular matrix, then the inverse is given by:

$$A^{-1} = \frac{1}{|A|} \text{Adj}(A)$$

5. Adjoint of  $A$ :  $\text{Adj}(A) = (\text{Cofactor Matrix})^T$ .

6. Determinant minors and cofactors (for 3x3 D):

$$D = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

Minor of  $a_{11}$ :  $m_{11} = a_{22}a_{33} - a_{23}a_{32}$  Cofactor of  $a_{ij}$ :  
 $A_{ij} = (-1)^{i+j}m_{ij}$  Example:  $A_{11} = a_{22}a_{33} - a_{23}a_{32}$ ,  
 $A_{12} = -m_{12}$ .

7. Important Properties of Determinants:

- i. If any two columns (or rows) are identical, its value is zero.

$$\begin{vmatrix} a & a & c \\ b & b & f \\ d & d & g \end{vmatrix} = 0$$

- ii. If all elements in any one row (or column) of a determinant are multiplied by a scalar  $k$ , the value is  $k$  times the value of the original determinant.

$$\begin{vmatrix} ka & kb & kc \\ d & e & f \\ g & h & i \end{vmatrix} = k \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

- iii. The value of the determinant remains unchanged if a multiple of one row (or column) is added to another.

$$D = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \implies D = \begin{vmatrix} a_1 + mb_1 & b_1 & c_1 \\ a_2 + mb_2 & b_2 & c_2 \\ a_3 + mb_3 & b_3 & c_3 \end{vmatrix}$$

- iv. If two adjacent rows (or columns) are interchanged, the sign of the determinant's value changes.

8. Cramer's Rule for solving simultaneous linear equations: Given:  $a_1x + b_1y + c_1z = d_1$ ,  $a_2x + b_2y + c_2z = d_2$ ,  $a_3x + b_3y + c_3z = d_3$ .

$$D = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \neq 0$$

$$D_x = \begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix} \quad D_y = \begin{vmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{vmatrix}$$

$$D_z = \begin{vmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{vmatrix}$$

Solution:  $x = \frac{D_x}{D}$ ,  $y = \frac{D_y}{D}$ , and  $z = \frac{D_z}{D}$ .

### 7.3. Vectors

1. If  $\vec{a} = a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$ , the magnitude is:

$$|\vec{a}| = \sqrt{a_1^2 + a_2^2 + a_3^2}$$

2. Position Vector: The position vector of point  $(x, y, z)$  is:

$$\vec{r} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

3. Vector Subtraction: If  $\vec{OP} = x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k}$  and  $\vec{OQ} = x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k}$ , then:

$$\vec{PQ} = \vec{OQ} - \vec{OP} = (x_2 - x_1)\mathbf{i} + (y_2 - y_1)\mathbf{j} + (z_2 - z_1)\mathbf{k}$$

4. Section Formula:

- i. Internal Division: If  $R$  divides  $AB$  internally in the ratio  $m : n$ , and  $\vec{a}, \vec{b}$  are position vectors of  $A$  and  $B$ , then:

$$\vec{r} = \frac{m\vec{b} + n\vec{a}}{m + n}$$

- ii. Midpoint: If  $C$  is the midpoint of  $AB$ , the position vector of  $C$  is:

$$\vec{c} = \frac{\vec{a} + \vec{b}}{2}$$

5. Unit vector parallel to  $\vec{a}$ :  $\pm \frac{\vec{a}}{|\vec{a}|}$ .

6. Unit vector perpendicular to  $\vec{a}$  and  $\vec{b}$ :  $\pm \frac{\vec{a} \times \vec{b}}{|\vec{a} \times \vec{b}|}$ .

7. Unit vector perpendicular to the plane containing  $\vec{a}$  and  $\vec{b}$ :  $\pm \frac{\vec{a} \times \vec{b}}{|\vec{a} \times \vec{b}|}$ .

8. Vector addition: If  $\vec{a} = a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$  and  $\vec{b} = b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}$ , then:

$$\vec{a} + \vec{b} = (a_1 + b_1)\mathbf{i} + (a_2 + b_2)\mathbf{j} + (a_3 + b_3)\mathbf{k}$$

9. Scalar Product (Dot Product):

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos \theta = a_1b_1 + a_2b_2 + a_3b_3$$

10. Unit Vector Dot Products:

$$\mathbf{i} \cdot \mathbf{i} = \mathbf{j} \cdot \mathbf{j} = \mathbf{k} \cdot \mathbf{k} = 1, \quad \mathbf{i} \cdot \mathbf{j} = \mathbf{j} \cdot \mathbf{k} = \mathbf{k} \cdot \mathbf{i} = 0$$

11. Vector Product (Cross Product):

- i. Magnitude:  $|\vec{a} \times \vec{b}| = |\vec{a}||\vec{b}| \sin \theta$ .

- ii. Cartesian form:

$$\vec{a} \times \vec{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

- iii. Property:  $\vec{a} \times \vec{b} = -(\vec{b} \times \vec{a})$ .

12. Unit Vector Cross Products:  $\mathbf{i} \times \mathbf{i} = 0$ . Cyclic relations:

$$\mathbf{i} \times \mathbf{j} = \mathbf{k}, \quad \mathbf{j} \times \mathbf{k} = \mathbf{i}, \quad \mathbf{k} \times \mathbf{i} = \mathbf{j}$$

13. Condition for Perpendicularity: If  $\vec{a} \perp \vec{b}$ , then  $\vec{a} \cdot \vec{b} = 0$ .

14. Scalar projection of  $\vec{a}$  onto  $\vec{b}$ :

$$a \cos \theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{b}|}$$

15. Vector projection of  $\vec{a}$  onto  $\vec{b}$  (along  $\vec{b}$ ):

$$\frac{(\vec{a} \cdot \vec{b})}{|\vec{b}|^2} \vec{b}$$

16. Scalar projection of  $\vec{b}$  onto  $\vec{a}$ :

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}|}$$

17. Volume of the parallelepiped formed by  $\vec{a}, \vec{b}, \vec{c}$ :

$$\left| \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} \right| \text{ cubic units}$$

18. Condition for Coplanarity: If vectors  $\vec{a}, \vec{b}, \vec{c}$  are coplanar, then  $\vec{a} \cdot (\vec{b} \times \vec{c}) = 0$ .

19. Area Formulas (Parallelogram and Triangle):

- i. Area of parallelogram formed by  $\vec{a}$  and  $\vec{b}$ :  $|\vec{a} \times \vec{b}|$  sq units.  
ii. Area of triangle formed by  $\vec{a}$  and  $\vec{b}$ :  $\frac{1}{2} |\vec{a} \times \vec{b}|$  sq units.

### 7.4. Straight Lines

1. Coordinates on the axes: A point on the  $x$ -axis is  $(x, 0)$ ; a point on the  $y$ -axis is  $(0, y)$ .  
2. Distance of point  $(x, y)$ : From the  $x$ -axis is  $|y|$ ; from the  $y$ -axis is  $|x|$ .  
3. Cartesian  $(x, y)$  and Polar  $(r, \theta)$  Coordinate relations:

$$x = r \cos \theta, \quad y = r \sin \theta$$

$$r = \sqrt{x^2 + y^2}, \quad \theta = \tan^{-1} \left( \frac{y}{x} \right)$$

(Conditions:  $0 \leq r < \infty$  and  $0 \leq \theta < 2\pi$ ).

4. Location of Polar Coordinates  $(r, \theta)$  from Cartesian  $(x, y)$ :

- i. For the **first quadrant** ( $x > 0, y > 0$ ):  $\theta = \tan^{-1} \left( \frac{y}{x} \right)$ ;  $0 < \theta < 90^\circ$ .

- ii. For the **second quadrant** ( $-x, y$ ):  $\theta = 180^\circ - \tan^{-1} \left( \frac{y}{|x|} \right)$ ;  $90^\circ < \theta < 180^\circ$ .

- iii. For the **third quadrant** ( $-x, -y$ ):  $\theta = 180^\circ + \tan^{-1} \left( \frac{|y|}{|x|} \right)$ ;  $180^\circ < \theta < 270^\circ$ .

- iv. For the **fourth quadrant** ( $x, -y$ ):  $\theta = 360^\circ - \tan^{-1} \left( \frac{|y|}{x} \right)$ ;  $270^\circ < \theta < 360^\circ$ .

- v. Alternatively, for Cartesian  $(x, y)$ , the angle  $\theta$  is:  $\theta = \tan^{-1} \left( \frac{y}{x} \right)$  (If  $x > 0$ ) or  $\pi + \tan^{-1} \left( \frac{y}{x} \right)$  (If  $x < 0$ )

5. **Distance Formula** between two points  $(x_1, y_1)$  and  $(x_2, y_2)$ :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

6. Distance between two points  $(x_1, b)$  and  $(x_2, b)$ :  $|x_2 - x_1|$  (since  $y_1 = y_2 = b$ ).

7. Distance between two points  $(a, y_1)$  and  $(a, y_2)$ :  $|y_2 - y_1|$  (since  $x_1 = x_2 = a$ ).

8. Distance between two **Polar Points**  $(r_1, \theta_1)$  and  $(r_2, \theta_2)$ :

$$d = \sqrt{r_1^2 + r_2^2 - 2r_1r_2 \cos(\theta_1 - \theta_2)}$$

9. **Division Formula (Internal)**: Coordinates of point dividing line segment  $(x_1, y_1)$  and  $(x_2, y_2)$  **internally** in ratio  $m_1 : m_2$ :

$$\left( \frac{m_1x_2 + m_2x_1}{m_1 + m_2}, \frac{m_1y_2 + m_2y_1}{m_1 + m_2} \right)$$

10. **Division Formula (External)**: Coordinates of point dividing line segment  $(x_1, y_1)$  and  $(x_2, y_2)$  **externally** in ratio  $m_1 : m_2$ :

$$\left( \frac{m_1x_2 - m_2x_1}{m_1 - m_2}, \frac{m_1y_2 - m_2y_1}{m_1 - m_2} \right)$$

11. **Midpoint Formula**: Midpoint of line segment between  $(x_1, y_1)$  and  $(x_2, y_2)$ :

$$\left( \frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

12. **Centroid Formula**: Centroid of a triangle with vertices  $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ :

$$\left( \frac{x_1 + x_2 + x_3}{3}, \frac{y_1 + y_2 + y_3}{3} \right)$$

13. **Area of Triangle**  $\triangle ABC$  with vertices  $A(x_1, y_1), B(x_2, y_2), C(x_3, y_3)$ :

$$\text{Area} = \frac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \text{ sq units}$$

14. Area of Triangle with origin  $(0, 0)$  and vertices  $(x_1, y_1), (x_2, y_2)$ :

$$\text{Area} = \frac{1}{2} |x_1y_2 - x_2y_1| \text{ sq units}$$

15. **Collinearity Condition**: Three points  $A(x_1, y_1), B(x_2, y_2), C(x_3, y_3)$  are collinear if the area of  $\triangle ABC$  is zero:

$$\frac{1}{2} |x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)| = 0$$

$$\left( \text{or using the determinant form: } \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0 \right)$$

16. **Area of Quadrilateral**  $ABCD$  with vertices  $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ :

$$\text{Area} = \frac{1}{2} \left| \begin{vmatrix} x_1 & x_2 & x_3 & x_4 & x_1 \\ y_1 & y_2 & y_3 & y_4 & y_1 \end{vmatrix} \right|$$

17. **Slope (Gradient)** of a Straight Line:

- The **slope** is  $m = \tan \theta$ , where  $\theta$  is the angle with the positive  $x$ -axis ( $0^\circ \leq \theta < 180^\circ$ ).
- Slope of the **x-axis** (or parallel to  $x$ -axis) is  $m = 0$ .
- Slope of the **y-axis** (or parallel to  $y$ -axis) is **undefined**.
- Slope of the line passing through  $(x_1, y_1)$  and  $(x_2, y_2)$ :  $m = \frac{y_2 - y_1}{x_2 - x_1}$ .
- Slope of the line  **$ax + by + c = 0$** :  $m = -\frac{a}{b}$ .

18. Equation of the **x-axis**:  $y = 0$ .

19. Equation of the **y-axis**:  $x = 0$ .

20. Equation of a line **parallel to the x-axis** (or horizontal line):  $y = b$ .

21. Equation of a line **parallel to the y-axis** (or vertical line):  $x = a$ .

22. Equation of a line **passing through the origin**:  $y = mx$ .

23. **Different Forms of the Equation of a Straight Line**:

- i. **Point-Slope form** (passing through  $(x_1, y_1)$  with slope  $m$ ):  $y - y_1 = m(x - x_1)$ .

- ii. **Slope-Intercept form** (with slope  $m$  and  $y$ -intercept  $c$ ):  $y = mx + c$ .

- iii. **Two-Point form** (passing through  $(x_1, y_1)$  and  $(x_2, y_2)$ ):  $\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$ .

- iv. **Intercept form** (with  $x$ -intercept  $a$  and  $y$ -intercept  $b$ ):  $\frac{x}{a} + \frac{y}{b} = 1$ .

- v. **General form**:  $ax + by + c = 0$ .

24. **Area of the triangle** formed by the line  $\frac{x}{a} + \frac{y}{b} = 1$  and the coordinate axes:  $\frac{1}{2} |ab|$  sq units.

25. **Perpendicular Distance** from a point  $(x_1, y_1)$  to the line  $ax + by + c = 0$ :

$$d = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$