PBA-2

Subsequence:

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

void sub(int ind, int n, vector<int> &ds, int arr[]) {
    if(ind == n) {
        if(ds.size() == 0) {
            cout << "{}";
        } else {
            for(auto it : ds) {
                cout << it;
            }
        }
        cout << endl;
        return;
    }

    ds.push_back(arr[ind]);
    sub(ind + 1, n, ds, arr);
    ds.pop_back();

    sub(ind + 1, n, ds, arr);
}

int main() {
    int n;
    cin >> n;

    int arr[n];
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    vector<int> ds;
    sub(0, n, ds, arr);
}
```

Longest Common Subsequence(only number):

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

int f(int i, int j, string &s, string &t, vector<vector<int>> &dp) {
    if(i < 0 || j < 0) return 0;

    if(dp[i][j] != -1) return dp[i][j];
    if(s[i] == t[j]) return dp[i][j] = 1 + f(i-1, j-1, s, t, dp);

    return dp[i][j] = max(f(i-1, j, s, t, dp), f(i, j-1, s, t, dp));
}

int main() {
    string s, t;
    cin >> s >> t;

    int n = s.size();
    int m = t.size();

    vector<vector<int>> dp(n, vector<int>(m, -1));

    cout << f(n-1, m-1, s, t, dp);
    return 0;
}
```

Longest Common Subsequence(number and string):

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

int f(int i, int j, string &s, string &t, vector<vector<int>> &dp) {
    if (i < 0 || j < 0) return 0;

    if (dp[i][j] != -1) return dp[i][j];

    if (s[i] == t[j])
```

```
        return dp[i][j] = 1 + f(i-1, j-1, s, t, dp);
    else
        return dp[i][j] = max(f(i-1, j, s, t, dp), f(i, j-1, s, t, dp));
}

// function to recover LCS string
string getLCS(int i, int j, string &s, string &t, vector<vector<int>> &dp) {
    if (i < 0 || j < 0) return "";

    if (s[i] == t[j])
        return getLCS(i-1, j-1, s, t, dp) + s[i];

    if (i > 0 && dp[i-1][j] >= dp[i][j-1])
        return getLCS(i-1, j, s, t, dp);
    else
        return getLCS(i, j-1, s, t, dp);
}

int main() {
    string s, t;
    cin >> s >> t;

    int n = s.size();
    int m = t.size();

    vector<vector<int>> dp(n, vector<int>(m, -1));

    int len = f(n-1, m-1, s, t, dp);
    string lcs = getLCS(n-1, m-1, s, t, dp);

    cout << len << "\n";
    cout << lcs << "\n";
}
```

LCS(Full Program):

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

int solveLCS(int i, int j, string &s1, string &s2, vector<vector<int>> &dp) {
    if (i < 0 || j < 0)
        return 0;
    if (dp[i][j] != -1)
        return dp[i][j];

    if (s1[i] == s2[j])
        return dp[i][j] = 1 + solveLCS(i - 1, j - 1, s1, s2, dp);

    return dp[i][j] = max(solveLCS(i - 1, j, s1, s2, dp), solveLCS(i, j - 1, s1, s2,
dp));
}

string buildLCS(int i, int j, string &s1, string &s2, vector<vector<int>> &dp) {
    if (i < 0 || j < 0)
        return "";

    if (s1[i] == s2[j])
        return buildLCS(i - 1, j - 1, s1, s2, dp) + s1[i];

    if (i > 0 && dp[i - 1][j] >= dp[i][j - 1])
        return buildLCS(i - 1, j, s1, s2, dp);

    return buildLCS(i, j - 1, s1, s2, dp);
}

long long countLCS(string &s, string &lcs) {
    int n = s.size();
    int m = lcs.size();

    vector<vector<long long>> dp(n + 1, vector<long long>(m + 1, 0));

    for (int i = 0; i <= n; i++)
        dp[i][0] = 1;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s[i - 1] == lcs[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
            else
                dp[i][j] = dp[i - 1][j];
        }
    }

    return dp[n][m];
}
```

```
int main() {
    string s1, s2;

    getline(cin, s1);
    getline(cin, s2);

    int n = s1.size();
    int m = s2.size();

    vector<vector<int>> dp(n, vector<int>(m, -1));

    int lcsLen = solveLCS(n - 1, m - 1, s1, s2, dp);
    string lcsStr = buildLCS(n - 1, m - 1, s1, s2, dp);
    long long count = countLCS(s1, lcsStr);

    cout << lcsLen << endl;
    cout << count << endl;
    cout << lcsStr << endl;

    return 0;
}
```

Range Query Algorithm(To find max b/w the range and update the value):

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

int main() {
    int n, m;
    cin >> n >> m;

    vector<int> arr(n);
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    for(int i = 0; i < m; i++) {
        char c;
        int fi, li;
        cin >> c >> fi >> li;
        if(c == 'Q') {
            int maxval = arr[fi];
            for(int j = fi; j <= li; j++) {
                if(arr[j] > maxval) {
                    maxval = arr[j];
                }
            }
            cout << maxval << endl;
        }

        if(c == 'U') {
            arr[fi] = li;
        }
    }

    return 0;
}
```

Range Query Algorithm(To find min b/w the range and update the value):

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

int main() {
    int n, m;
    cin >> n >> m;

    vector<int> arr(n);
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    for(int i = 0; i < m; i++) {
        char c;
        int fi, li;
        cin >> c >> fi >> li;

        if(c == 'Q') {
            int minval = arr[fi];
            for(int j = fi; j <= li; j++) {
                if(arr[j] < minval) {
```

```
                minval = arr[j];
            }
        }
        cout << minval << endl;
    }

    if(c == 'U') {
        arr[fi] = li;
    }
}

return 0;
}
```

Range Query Algorithm(To find sum b/w the range and update the value):

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

int main() {
    int n, m;
    cin >> n >> m;

    vector<int> arr(n);
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    for(int i = 0; i < m; i++) {
        char c;
        int x, y;
        cin >> c >> x >> y;

        if(c == 'S') {
            int sum = 0;
            for(int j = x; j <= y; j++) {
                sum += arr[j];
            }
            cout << sum << endl;
        }
        else if(c == 'U') {
            arr[x] = y;
        }
    }

    return 0;
}
```

Range Query Algorithm(To find Mode b/w the range and update the value):

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

int main() {
    int n, m;
    cin >> n >> m;

    vector<int> arr(n);
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    for(int i = 0; i < m; i++) {
        char c;
        int x, y;
        cin >> c >> x >> y;

        if(c == 'M') {
            int freq[101] = {0};

            for(int j = x; j <= y; j++) {
                freq[arr[j]]++;
            }

            int mode = 1;
            int maxf = freq[1];

            for(int k = 2; k <= 100; k++) {
                if(freq[k] > maxf) {
                    maxf = freq[k];
                    mode = k;
                }
            }

            cout << mode << endl;
        }
        else if(c == 'U') {
            arr[x] = y;
        }
    }

    return 0;
}
```

Range Frequency Query & Update Algorithm (Count users appearing ≥ k times in range):

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

int main() {
    int n;
    cin >> n;

    vector<int> arr(n);
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    int Q;
    cin >> Q;

    while(Q--) {
        int l, r, k;
        cin >> l >> r >> k;

        int freq[101] = {0};

        // Count frequency in the given range (convert to 0-based index)
        for(int i = l - 1; i <= r - 1; i++) {
            freq[arr[i]]++;
        }

        int count = 0;

        // Count how many values appear at least k times
        for(int i = 1; i <= 100; i++) {
            if(freq[i] >= k) {
                count++;
            }
        }

        cout << count << endl;
    }

    return 0;
}
```

Floyd Warshall Algorithm (Shortest Path Matrix for Directed Graph):

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

int main() {
    int n;
    cin >> n;

    vector<vector<int>> graph(n, vector<int>(n));

    // Input adjacency matrix
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cin >> graph[i][j];
        }
    }

    // Floyd-Warshall Algorithm
    for(int k = 0; k < n; k++) {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                if(graph[i][k] != 999 && graph[k][j] != 999) {
                    graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j]);
                }
            }
        }
    }

    // Output shortest path matrix
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cout << graph[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Floyd-Warshall Algorithm to Find All-Pairs Shortest Path:

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

const int INF = 1e9;

int main() {
    int v, e;
    cin >> v >> e;

    vector<vector<int>> dist(v, vector<int>(v, INF));

    // Distance from a node to itself is 0
    for(int i = 0; i < v; i++) {
        dist[i][i] = 0;
    }

    // Read edges (u -> v with weight w)
    for(int i = 0; i < e; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        dist[u][v] = w;
        dist[v][u] = w; // Undirected graph
    }

    // Print the original matrix
    cout << "Original matrix" << endl;
    for(int i = 0; i < v; i++) {
        for(int j = 0; j < v; j++) {
            if(dist[i][j] == INF) cout << "INF ";
            else cout << dist[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;

    // Floyd-Warshall Algorithm
    for(int k = 0; k < v; k++) {
        for(int i = 0; i < v; i++) {
            for(int j = 0; j < v; j++) {
                if(dist[i][k] != INF && dist[k][j] != INF) {
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }

    // Print the shortest path matrix
    cout << "Shortest path matrix" << endl;
    for(int i = 0; i < v; i++) {
        for(int j = 0; j < v; j++) {
            if(dist[i][j] == INF) cout << "INF ";
            else cout << dist[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Shortest Delivery Route Using Dijkstra's Algorithm:

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

// Dijkstra's Algorithm to find shortest path
vector<int> dijkstra(int n, vector<vector<pair<int,int>>> &adj, int s,
vector<int> &parent) {
    vector<int> dist(n, INT_MAX);
    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>>
pq;

    dist[s] = 0;
    pq.push({0, s});

    while (!pq.empty()) {
        int u = pq.top().second;
        int d = pq.top().first;
        pq.pop();

        if (d > dist[u]) continue;

        for (auto [v, w] : adj[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                parent[v] = u;
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}

int main() {
    int n, m;
    cin >> n >> m;

    vector<vector<pair<int,int>>> adj(n);

    // Graph Input
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w}); // Undirected graph
    }

    int source, destination;
    cin >> source >> destination;

    vector<int> parent(n, -1);
    vector<int> dist = dijkstra(n, adj, source, parent);

    // No path exists
    if (dist[destination] == INT_MAX) {
        cout << "No path found";
        return 0;
    }

    // Reconstruct path
    vector<int> path;
    for (int v = destination; v != -1; v = parent[v]) {
        path.push_back(v);
    }

    reverse(path.begin(), path.end());

    // Output
    cout << "Shortest path: ";
    for (int i = 0; i < path.size(); i++) {
        cout << path[i];
        if (i != path.size() - 1) cout << " -> ";
    }

    cout << "\nShortest distance: " << dist[destination];

    return 0;
}
```

Dijkstra's Algorithm Using Adjacency Matrix:

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

// Dijkstra's Algorithm using Adjacency Matrix
vector<int> dijkstra(int n, vector<vector<int>> &graph, int s) {
    vector<int> dis(n, INT_MAX);
    priority_queue<vector<int>, vector<vector<int>>, greater<vector<int>>>
pq;

    dis[s] = 0;
    pq.push({0, s});

    while (!pq.empty()) {
        int u = pq.top()[1];
        pq.pop();

        for (int v = 0; v < n; v++) {
            if (graph[u][v] != 0 && dis[v] > dis[u] + graph[u][v]) {
                dis[v] = dis[u] + graph[u][v];
                pq.push({dis[v], v});
            }
        }
    }
    return dis;
}
```

```
int main() {
    int n;
    cin >> n;

    vector<vector<int>> graph(n, vector<int>(n));

    // Input adjacency matrix
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
        }
    }

    int s;
    cin >> s;

    vector<int> ans = dijkstra(n, graph, s);

    cout << "Vertex\tDistance from Source\n";
    for (int i = 0; i < ans.size(); i++) {
        cout << i << "\t" << ans[i] << endl;
    }

    return 0;
}
```

Range Minimum Query using Brute Force (AQI Analysis):

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

int main() {
    int N;
    cin >> N;

    vector<int> arr(N);
    for(int i = 0; i < N; i++) {
        cin >> arr[i];
    }

    int Q;
    cin >> Q;

    for(int i = 0; i < Q; i++) {
        int x, y;
        cin >> x >> y;

        int mn = arr[x];
        for(int j = x; j <= y; j++) {
            if(arr[j] < mn) {
                mn = arr[j];
            }
        }

        cout << mn << endl;
    }

    return 0;
}
```

Sliding Window Min of Window Maximums:

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

int main() {
    int n;
    cin >> n;

    int k;
    cin >> k;

    vector<int> arr(n);
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    int q;
    cin >> q;

    while(q--) {
        int u, w;
        cin >> u >> w;
        vector<int> max_arr;

        for(int i = u; i <= w - k + 1; i++) {
            int mx = arr[i];
            for(int j = i; j < i + k; j++) {
                mx = max(mx, arr[j]);
            }
            max_arr.push_back(mx);
        }

        int mn = *min_element(max_arr.begin(), max_arr.end());
        cout << mn << endl;
    }
}
```