Lab Report - 01

Vector

Problem Link : https://codeforces.com/problemset/problem/2113/C

Problem : Smilo is mining gold on a `n × m` grid where each cell is either empty (`.`), stone (`#`), or gold (`g`). He can place **one dynamite** in any empty cell. The explosion affects a square of size `(2k+1) × (2k+1)` centered at that cell. Gold on the **boundary** of this square is collected. Gold **strictly inside** the square is destroyed. The explosion area may go beyond the grid but must be centered in a valid empty cell.

Through Process : My first observation is that **only one dynamite explosion is enough** to collect all the gold in the mine. Therefore, the goal becomes choosing a valid explosion position such that the **loss of gold is minimized**. To efficiently calculate the number of gold ores lost for any explosion position, I use a **2D prefix sum**. This allows me to quickly find the total amount of gold that would be destroyed within the explosion square for each valid empty cell. Finally, I choose the position where the **number of destroyed (lost) golds is minimum**, ensuring the **maximum collection**.

Source Code :

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

    // Typedef
typedef long long ll;
    // Macros
#define PB push_back
#define all(x) x.begin(), x.end()
#define trav(i, a) for (auto &i : a)
#define F first
#define S second
#define endl '\n'
#define FOR(i, a, b) for (int i = a; i < b; i++)
#define REP(i, a, b) for (int i = a; i <= b; i++)
void fast() {
    ios::sync_with_stdio(false);
    cin.tie(0);
}
void solve(ll tc){
    int n, m, k;
    cin >> n >> m >> k;
    vector<vector<char>> v(n + 1 ,vector<char>(m + 1, ' '));
    REP(i, 1, n){
        REP(j, 1, m) cin >> v[i][j];
    }
    vector<vector<int>> pre(n + 1 ,vector<int>(m + 1, 0));
    REP(i, 1, n){
        REP(j, 1, m){
            if (v[i][j] == 'g') pre[i][j] = 1;
        }
    }
    REP(i, 1, n){
        REP(j, 1, m) pre[i][j] += pre[i - 1][j] + pre[i][j - 1] - pre[i - 1][j - 1];
    }
    int mn_loss = INT_MAX;
    REP(i, 1, n){
        REP(j, 1, m){
            if (v[i][j] != '.') continue;
            int x1 = max(1, i - k + 1);
            int y1 = max(1, j - k + 1);
            int x2 = min(n, i + k - 1);
            int y2 = min(m, j + k - 1);
            int diff = pre[x2][y2] - pre[x1 - 1][y2] - pre[x2][y1 - 1] + pre[x1 - 1][y1 -
1];
            mn_loss = min(mn_loss, diff);
        }
    }
    int ans = pre[n][m] - mn_loss;
    cout << ans << '\n';
}
int main(void) {
    fast();
    ll t = 1;
    int i = 1;
    cin >> t;
    for (ll i = 1; i <= t; i++)
        solve(i);
}
```

Input :

```
3
2 3 1
#.#
g.g
2 3 2
#.#
g.g
3 4 2
.gg.
g..#
g##.
```

Output :

```
2
0
4
```

Vector & Map

Problem Link : https://codeforces.com/contest/2123/problem/E

Problem : Given an array `a` of size `n` with nonnegative integers, define **MEX** as the smallest nonnegative integer **not present** in the array. For each `k` from `0` to `n`, compute how many **distinct MEX values** are possible by removing exactly `k` elements from the array.

Thought Process: The key observation is that the last `mex` elements in a valid configuration must be the values from `mex` down to `1` in some order — this part is fixed for any input, where `mex` is the smallest non-negative integer not present in the array. To begin, we sort the array and compute the initial `mex`. For `k = 0`, the only valid sequence is the one ending with the elements `1` to `mex`, so the minimum number of segments needed is always `1`. We then analyze all elements in the array that are less than or equal to the `mex`, counting their frequencies. These frequencies are then grouped by how many times each number occurs. Using this, for each `k > 0`, we can compute how many new segments are needed by incrementally adding the number of values that appear `k` times. The result for each `k` is stored in a vector `ans`, and any undefined positions (initially marked as `-1`) are filled by extending the logic using the frequency map. Finally, the `ans` vector is printed, giving the minimum number of operations or segments for each possible `k`.

Source Code :

```
#include <bits/stdc++.h>
using namespace std;
    // Typedef
typedef long long ll;
typedef vector<int> vi;
typedef map<int, int> mii;
    // Macros
#define PB push_back
#define all(x) x.begin(), x.end()
#define trav(i, a) for (auto &i : a)
#define F first
#define S second
#define endl '\n'
#define DEBUG(i) cout << "DEBUG " << i << "\n";
#define FOR(i, a, b) for (int i = a; i < b; i++)
#define GT() greater<>()
#define print(x) for(auto &i : x) cout << i << " " ; cout '\n'
    // Functions
template <typename T>
void pv(vector<T> &a){
    trav(u, a) cout << u << ' ';
    cout << '\n';
}
void fast() {
    ios::sync_with_stdio(false);
    cin.tie(0);
}
void solve(ll tc){
    int n;
    cin >> n;
    vi v(n);
    FOR(i, 0, n) cin >> v[i];
    sort(all(v));
    int mex = 0;
    FOR(i, 0, n) if (v[i] == mex) mex++;
    mii m;
    FOR(i, 0, n){if (v[i] > mex) break; m[v[i]]++;}
    mii mfreq;
    trav(u, m) mfreq[u.S]++;
    // trav(u, mfreq) cout << u.F << " = " << u.S << '\n';
    vi ans(n + 1, -1);
    ans[0] = 1;
    int cnt = 0, ind = 1;;
    while (cnt < mex){ans[n - cnt] = cnt + 1; cnt++;}
    while (ans[ind] == -1){
        ans[ind] = ans[ind - 1] + mfreq[ind];
        ind++;
    }
    pv(ans);
}
int main(void){
    fast();
    ll t = 1;
    int i = 1;
    cin >> t;
    for (ll i = 1; i <= t; i++)
        solve(i);
}
```

Input :

```
5
5
1 0 0 1 2
6
3 2 0 4 5 1
6
1 2 0 1 3 2
4
0 3 4 1
5
0 0 0 0 0
```

Output :

```
1 2 4 3 2 1
1 6 5 4 3 2 1
1 3 5 4 3 2 1
1 3 3 2 1
1 1 1 1 1 1
```

Queue

Problem Link : https://cses.fi/problemset/task/1667

Problem : Syrjälä's network has `n` computers and `m` connections. You need to determine if **Uolevi** can send a message to **Maija** (from computer `1` to computer `n`). If possible, output the **minimum number of computers** on the path from `1` to `n` (including both). Otherwise, print that it is **not possible**.

Thought Process : This problem is essentially about finding the shortest path in an unweighted graph. To solve it efficiently, I used Breadth-First Search (BFS) since BFS guarantees the shortest path from the source to all reachable nodes in such graphs. While performing BFS, I also kept track of the parent of each node. This allows me to reconstruct the shortest path by backtracking from the destination node to the source. Finally, by reversing the traced path, I get the correct direction from start to end.

Source Code :

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

```cpp
    // Typedef
typedef long long ll;
typedef vector<int> vi;
typedef vector<vi> vvi;

#define PB push_back
#define all(x) x.begin(), x.end()
#define trav(i, a) for (auto &i : a)
#define endl '\n'
#define FOR(i, a, b) for (int i = a; i < b; i++)
#define REP(i, a, b) for (int i = a; i <= b; i++)
template <typename T>
void pv(vector<T> &a){
    trav(u, a) cout << u << ' ';
    cout << '\n';
}
void fast() {
    ios::sync_with_stdio(false);
    cin.tie(0);
}
const int N = 1e5 + 10;
vvi adj(N);
vi dis(N, -1);
vi parent(N, -1);
void BFS(){
    queue<int> pos;
    pos.push(1);
    dis[1] = 0;
    while(!pos.empty()){
        int node = pos.front();
        pos.pop();
        trav(u, adj[node]){
            if (dis[u] != -1) continue;
            dis[u] = dis[node] + 1;
            parent[u] = node;
            pos.push(u);
        }
    }
}
void solve(ll tc){
    int n, m;
    cin >> n >> m;
    FOR(i, 0, m){
        int u, v;
        cin >> u >> v;
        adj[u].PB(v);
        adj[v].PB(u);
    }
    BFS();
    if (dis[n] == -1){cout << "IMPOSSIBLE\n";return;}
    cout << dis[n] + 1 << '\n';
    int ind = n;
    vi ans;
    ans.PB(ind);
    while(parent[ind] != -1){
        ans.PB(parent[ind]);
        ind = parent[ind];
    }
    reverse(all(ans));
    pv(ans);
}
int main(void) {
    fast();
    ll t = 1;
    int i = 1;
    solve(i);
}
```

Input :

```
5 5
1 2
1 3
1 4
2 3
5 4
```

Output :

```
3
1 4 5
```

Set

Problem Link : https://cses.fi/problemset/task/1621/

Problem : You are given a list of n integers, and your task is to calculate the number of *distinct* values in the list.

Through Process : This is a straightforward **set-based problem**. The goal is simply to determine the number of **distinct elements**, which can be done by inserting all elements into a set and then **printing the size** of the set. Since sets automatically handle uniqueness, this provides a clean and efficient solution.

Source Code :

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

    // Typedef
typedef long long ll;
    // Macros
#define IN insert
#define endl '\n'
#define FOR(i, a, b) for (int i = a; i < b; i++)
#define SZ(x) (ll)x.size()
void fast() {
    ios::sync_with_stdio(false);
    cin.tie(0);
}
void solve(ll tc){
    int n;
    cin >> n;
    set<int> s;
    FOR(i, 0, n){
        int x;
        cin >> x;
        s.IN(x);
    }
    cout << SZ(s) << '\n';
}
int main(void) {
    fast();
    ll t = 1;
    int i = 1;
    solve(i);
}
```

Input :

```
5
2 3 2 2 3
```

Output :

```
2
```

Pair

Problem Link : https://atcoder.jp/contests/abc413/tasks/abc413_c

Through Process : At first, I realized that a brute-force approach would be too slow to handle the constraints efficiently. To optimize, I used a **pair of deque** to store both the values and their frequencies. When adding a number, I push it to the **back** of the deque while maintaining its frequency. If a number already exists at the back, I simply increase its frequency instead of adding a duplicate. For outputting the result (e.g., sum or count), I remove elements from the **front** of the deque as needed. This structure allows me to efficiently maintain and access the required information in **amortized constant time** per operation.

Source Code :

```
#include <bits/stdc++.h>
using namespace std;
    // Typedef
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
    // Macros
#define PB push_back
#define all(x) x.begin(), x.end()
#define trav(i, a) for (auto &i : a)
#define F first
#define S second
#define endl '\n'
#define sz(x) (ll)x.size()
void fast() {
    ios::sync_with_stdio(false);
    cin.tie(0);
}
void solve(ll tc){
    ll q;
    cin >> q;
    deque<pair<ll, pair<ll, ll>>> v;
    ll ind = 1;
    while (q--){
        ll type;
        cin >> type;
        if (type == 1){
            ll c, x;
            cin >> c >> x;
            v.PB({x, {ind, ind + c - 1}});
            ind += c;
        }
        else{
            ll k; cin >> k;
            ll sum = 0;
            while (k){
                auto u = v.front();
                v.pop_front();
                ll cnt = u.S.S - u.S.F + 1;
                if (cnt <= k){sum += u.F * cnt * 1ll; k -= cnt;}
                else{
                    sum += u.F * k;
                    u.S.F += k;
                    v.push_front(u);
                    k = 0;
                }
            }
            cout << sum << '\n';
        }
    }
}
int main(void) {
    fast();
    ll t = 1;
    int i = 1;
    solve(i);
}
```

Input :

```
5
1 2 3
1 4 5
2 3
1 6 2
2 5
```

Output :

```
11
19
```

Stack

Problem Link : https://leetcode.com/problems/valid-parentheses/

Problem : Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

Source Code :

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        int n = s.length();
        for(int i = 0; i < n; i++){
            if (s[i] == '(' || s[i] == '{' || s[i] == '[') st.push(s[i]);
            else{
                if (st.empty()) return false;
                char c = st.top();
                if ((s[i] == ')' && c != '(') || (s[i] == '}' && c != '{') || (s[i] == ']'
&& c != '[')) return 0;
                st.pop();
            }
        }
        return st.empty();
    }
};
```

Sieve

Problem Link : https://toph.co/p/n-th-prime

Problem : In this problem, you will have to print the nn-th prime number. The first few prime numbers are given below:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...
2 is the 1st prime, 3 is the 2nd prime, 5 is the 3rd prime, ...

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

    // Typedef
typedef long long ll;
typedef vector<int> vi;
typedef vector<ll> vll;
typedef vector<bool> vb;
    // Macros
#define PB push_back
#define endl '\n'
#define FOR(i, a, b) for (int i = a; i < b; i++)
void fast() {
    ios::sync_with_stdio(false);
    cin.tie(0);
}
const int N = 1e7;
vi v;
vb vis(N, 1);
void sieve(){
    vis[0] = 0;
    vis[1] = 0;
    for (int i = 2; i * i < N; i++){
        if (vis[i]){
                for (int j = i * i; j < N; j += i) vis[j] = 0;
            }
        }
        for (int i = 2; i < N; i++){
            if (vis[i]) v.PB(i);
        }
}
void solve(ll tc) {
    int n;
    cin >> n;
    sieve();
    cout << v[n - 1] << '\n';
}
int main(void) {
    fast();
    ll t = 1;
    int i = 1;
    solve(i);
}
```

Input :

```
2
```

Output :

```
3
```

GCD & LCM

Problem : Find the gcd and lcm of x, y.

Thought Process : This uses the **Euclidean algorithm**, which repeatedly replaces `(a, b)` with `(b, a % b)` until `b` becomes `0`. At that point, `a` is the GCD. This uses the identity:

$$\mathrm{LCM(a, b)} = \frac{\mathrm{(a \times b)}}{\mathrm{GCD(a, b)}}$$

It calculates the smallest number divisible by both `a` and `b`.

Source Code :

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

typedef long long ll;
int GCD(int a, int b) {
    while (b != 0) {
        int r = a % b;
        a = b;
        b = r;
    }
    return a;
}
ll LCM(ll a, ll b){
   return (a * b) / GCD(a, b);
}
```

```
int main(){
    int x, y;
    cin >> x >> y;
    int gcd = GCD(x, y);
    int lcm = LCM(x, y);
    cout << "GCD = " << gcd << '\n' << "LCM = " << lcm << '\n';
}
```

Input :

```
3 4
```

Output :

```
GCD = 1
LCM = 12
```