CCCC Recruitment Contest 2025 Cheatsheet

Prefix Sums & C++ Basics

Prepared for Contest Participants

0. Sample Template

On Codeforces or similar platforms, problems usually require reading multiple test cases. Here is a template you can always start with:

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t; // number of test cases
    cin >> t;
    while (t--) {
        // Solve each test case here
    }
    return 0;
}
```

1. C++ Containers Documentation

C++ provides powerful Standard Template Library (STL) containers. Below are the most useful ones for contest programming.

1.1 std::vector

A dynamic array that grows/shrinks in size.

```cpp
// Declaration
vector<int> v;                  // empty vector
vector<int> v2(5, 0);           // size 5, all elements = 0
int n = 5;
vector<int> v3(n);              // size n, all elements default-initialized

// Adding and removing
v.push_back(10);                // add element at end
v.pop_back();                   // remove last element

// Accessing (0-based indexing!)
v[0] = 5;                       // set first element
cout << v[0];                   // get first element
cout << v.size();               // number of elements

// Iterating: Method 1 (index based)
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << " ";
}
```

```cpp

// Iterating: Method 2 (range-based for loop)
for (auto x : v) {
    cout << x << " ";
}

// Iterating: Method 3 (iterators)
for (auto it = v.begin(); it != v.end(); it++) {
    cout << *it << " ";
}
```

1.2 std::map

A sorted key-value store (like a dictionary). Keys are unique.

```cpp
// Declaration
map<string, int> mp;

// Insert or update
mp["apple"] = 5;
mp["banana"] = 2;

// Access
cout << mp["apple"];          // prints 5

// Iteration
for (auto [key, value] : mp) {
    cout << key << " -> " << value << "\n";
}

// Check existence
if (mp.count("apple")) {
    cout << "apple exists";
}
```

1.3 std::pair

Stores two values together.

```cpp
pair<int, string> p = {1, "hello"};
cout << p.first << " " << p.second;

// Vector of pairs
vector<pair<int,int>> vp;
vp.push_back({1, 2});
vp.push_back({3, 4});

// Iterating
for (auto [x, y] : vp) {
    cout << x << "," << y << "\n";
}
```

1.4 std::string

Stores text and characters.

```cpp
string s = "hello";

// Access characters
cout << s[0];                 // 'h'
```

```cpp

// Length
cout << s.size();              // 5

// Add characters
s.push_back('!');
cout << s;                     // "hello!"

// Iterating
for (char c : s) {
    cout << c << " ";
}
```

1.5 sort() function

```cpp
vector<int> arr = {4, 2, 7, 1};
sort(arr.begin(), arr.end());        // ascending
sort(arr.rbegin(), arr.rend());      // descending

// Custom comparator
sort(arr.begin(), arr.end(), [](int a, int b) {
    return a % 10 < b % 10;
});
```

2. Prefix Sums

Prefix sums are a way to quickly compute sums of subarrays or submatrices.

2.1 1D Prefix Sum

Suppose we have an array `arr` of size n. We define a prefix array `pref` of size $n+1$, where:

$$pref[0] = 0$$

$$pref[i] = arr[0] + arr[1] + \ldots + arr[i-1], \quad (1 \le i \le n)$$

That means: - `pref[i]` stores the sum of the first i elements of `arr`. - To get the sum of subarray from L to R (0-based indexing):

$$sum(L, R) = pref[R+1] - pref[L]$$

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

// Build prefix sum
vector<int> pref(n+1, 0);
for (int i = 1; i <= n; i++) {
    pref[i] = pref[i-1] + arr[i-1];
}

// Query sum from L to R
int L = 2, R = 4; // example
int rangeSum = pref[R+1] - pref[L];
```

2.2 2D Prefix Sum

For a 2D matrix `mat` of size $n \times m$, define prefix matrix `pref` of size $(n+1) \times (m+1)$, all initialized to 0.

Formula:

$$pref[i][j] = mat[i-1][j-1] + pref[i-1][j] + pref[i][j-1] - pref[i-1][j-1]$$

To query sum of rectangle $(x1, y1)$ to $(x2, y2)$ (0-based, inclusive):

$$sum = pref[x2+1][y2+1] - pref[x1][y2+1] - pref[x2+1][y1] + pref[x1][y1]$$

```cpp
int n, m;
cin >> n >> m;
vector<vector<int>> mat(n, vector<int>(m));
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        cin >> mat[i][j];

vector<vector<int>> pref(n+1, vector<int>(m+1, 0));

// Build 2D prefix sum
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        pref[i][j] = mat[i-1][j-1]
                   + pref[i-1][j]
                   + pref[i][j-1]
                   - pref[i-1][j-1];
    }
}

// Query sum of rectangle [x1..x2][y1..y2]
int x1, y1, x2, y2;
cin >> x1 >> y1 >> x2 >> y2;
int rectSum = pref[x2+1][y2+1] - pref[x1][y2+1]
            - pref[x2+1][y1] + pref[x1][y1];
```

3. Sample Problem 1: Range Sum Queries

Problem: Given an array, answer queries asking the sum of elements between indices L and R.

Input:

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

Output:

```
6
9
12
```

Solution:

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t; cin >> t;
    while (t--) {
        int n, q;
        cin >> n >> q;
        vector<int> arr(n);
        for (int i = 0; i < n; i++) cin >> arr[i];

        vector<int> pref(n+1, 0);
        for (int i = 1; i <= n; i++) {
            pref[i] = pref[i-1] + arr[i-1];
        }

        while (q--) {
            int L, R;
            cin >> L >> R;
            cout << pref[R+1] - pref[L] << "\n";
        }
    }
}
```

4. Sample Problem 2: Most Frequent Element

Problem: Given an array of integers ($1 \leq a_i \leq 100$), print the most frequently occurring element. If multiple elements occur the same maximum number of times, print the smallest one.

Input:

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

Output:

```
1
2
```

Solution:

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t; cin >> t;
    while (t--) {
        int n; cin >> n;
        vector<int> arr(n);
        for (int i = 0; i < n; i++) cin >> arr[i];
```

```
13
14          // Frequency vector (values are between 1 and 100)
15          vector<int> freq(101, 0);
16          for (int x : arr) {
17              freq[x]++;
18          }
19
20          int bestVal = -1, bestCount = -1;
21          for (int val = 1; val <= 100; val++) {
22              if (freq[val] > bestCount) {
23                  bestCount = freq[val];
24                  bestVal = val;
25              }
26          }
27          cout << bestVal << "\n";
28      }
29  }
```