# ICPC 2025 Code Template

## Team: Team Awesome

**Team Members:**

1. Alice Johnson
2. Bob Smith
3. Charlie Davis

Generated: December 15, 2025

# Contents

# Data Structures

## Segment Tree

A segment tree is a tree data structure for storing intervals or segments. It allows querying which of the stored segments contain a given point efficiently.

**Time Complexity:**

- Build: $O(n)$
- Query: $O(\log n)$
- Update: $O(\log n)$

***Code in C++:***

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

template<typename T>
class SegmentTree {
private:
    vector<T> tree;
    int n;

    void build(vector<T>& arr, int node, int start, int
    ↪   end) {
        if (start == end) {
            tree[node] = arr[start];
        } else {
            int mid = (start + end) / 2;
            build(arr, 2*node, start, mid);
            build(arr, 2*node+1, mid+1, end);
            tree[node] = tree[2*node] + tree[2*node+1];
        }
    }

    T query(int node, int start, int end, int l, int r) {
        if (r < start || end < l) return 0;
        if (l <= start && end <= r) return tree[node];
        int mid = (start + end) / 2;
        return query(2*node, start, mid, l, r) +
               query(2*node+1, mid+1, end, l, r);
    }

public:
    SegmentTree(vector<T>& arr) {
        n = arr.size();
        tree.resize(4 * n);
        build(arr, 1, 0, n-1);
    }

    T query(int l, int r) {
        return query(1, 0, n-1, l, r);
    }
};
```

# Math

## Combinatorics

**Binomial Coefficient:**

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

**Pascal's Identity:**

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

***Code in Python:***

```python
MOD = 10**9 + 7

def factorial(n, mod=MOD):
    """Calculate factorial modulo mod"""
    fact = [1] * (n + 1)
    for i in range(1, n + 1):
        fact[i] = (fact[i-1] * i) % mod
    return fact

def modinv(a, mod=MOD):
    """Calculate modular inverse using Fermat's Little
    ↪ Theorem"""
    return pow(a, mod - 2, mod)

def nCr(n, r, mod=MOD):
    """Calculate nCr modulo mod"""
    if r > n or r < 0:
        return 0

    fact = factorial(n, mod)
    return (fact[n] * modinv(fact[r], mod) % mod *
            modinv(fact[n-r], mod)) % mod
```

## Number Theory

***Code in C++:***

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

// Extended Euclidean Algorithm
long long extgcd(long long a, long long b, long long &x,
↪   long long &y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    long long x1, y1;
    long long gcd = extgcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return gcd;
}

// Modular inverse
long long modinv(long long a, long long m) {
    long long x, y;
    long long gcd = extgcd(a, m, x, y);
    if (gcd != 1) return -1;  // No inverse exists
    return (x % m + m) % m;
}

// Fast exponentiation
long long power(long long a, long long b, long long mod) {
    long long res = 1;
    a %= mod;
    while (b > 0) {
        if (b & 1) res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
```

# Graphs

## BFS

***Code in C++:***

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

```cpp
vector<int> adj[100005];
int dist[100005];
bool visited[100005];

void bfs(int start) {
    queue<int> q;
    q.push(start);
    visited[start] = true;
    dist[start] = 0;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int v : adj[u]) {
            if (!visited[v]) {
                visited[v] = true;
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

**Applications:**

- Finding shortest path in unweighted graphs
- Level order traversal of trees
- Finding connected components
- Bipartite graph checking

## DFS

Depth-First Search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root and explores as far as possible along each branch before backtracking.

***Code in C++:***

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

vector<int> adj[100005];
bool visited[100005];

void dfs(int u) {
    visited[u] = true;
    // Process node u

    for (int v : adj[u]) {
        if (!visited[v]) {
            dfs(v);
        }
    }
}

// Iterative DFS
void dfs_iterative(int start) {
    stack<int> st;
    st.push(start);

    while (!st.empty()) {
        int u = st.top();
        st.pop();

        if (visited[u]) continue;
        visited[u] = true;
        // Process node u

        for (int v : adj[u]) {
            if (!visited[v]) {
                st.push(v);
            }
        }
    }
}
```