Academy of Cryptography Techniques ACM-ICPC Notebook 2025

Contents

1 Initial Setup

1.1 Template

```cpp
#include <bits/stdc++.h>
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; i++)
#define FORD(i, b, a) for (int i = (b), _a = (a); i >= _a; i--)
#define FORE(i, v) for (__typeof((v).begin()) i = (v).begin(); i != (v).end(); i++)
#define ALL(v) (v).begin(), (v).end()
#define ff first
#define ss second
#define MASK(i) (1LL << (i))
#define BIT(x, i) (((x) >> (i)) & 1)
#define __builtin_popcount __builtin_popcountll
using namespace std;
template <class X, class Y>
bool minimize(X &x, const Y &y)
{
    if (x > y)
    {
        x = y;
        return true;
    }
    else
        return false;
}
template <class X, class Y>
bool maximize(X &x, const Y &y)
{
    if (x < y)
    {
        x = y;
        return true;
    }
    else
        return false;
}
template <class T>
T Abs(const T &x)
{
    return (x < 0 ? -x : x);
}
/// template by buiduckhanh
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    // freopen("input.txt", "r", stdin);
    // freopen("output.txt", "w", stdout);
}
```

2 Data Structures

2.1 Disjoint Set Union (DSU)

```cpp
struct DSU {
    vector<int> p, sz;
    DSU(int n): p(n), sz(n,1) {
        iota(p.begin(), p.end(), 0);
    }
    int find(int x) {
        return (p[x]==x ? x : p[x]=find(p[x]));
    }
    bool unite(int a,int b){
        a=find(a); b=find(b);
        if(a==b) return false;
        if(sz[a]<sz[b]) swap(a,b);
        p[b]=a; sz[a]+=sz[b];
        return true;
    }
};
```

2.2 Segment Tree

```cpp
struct SegTree {
    vector<int> tree;
    int n;

    SegTree(int n) : n(n) {
        tree.resize(4*n);
    }

    void update(int node, int start, int end,
                int idx, int val) {
        if (start == end) {
            tree[node] = val;
        } else {
            int mid = (start + end) / 2;
            if (idx <= mid)
                update(2*node, start, mid, idx, val);
            else
                update(2*node+1, mid+1, end, idx, val);
            tree[node] = tree[2*node] + tree[2*node+1];
        }
    }

    int 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);
    }
};
```

3 Graph Algorithms

3.1 Dijkstra

```cpp
vector<int> dijkstra(vector<vector<pair<int,int>>>& adj, int start) {
    int n = adj.size();
    vector<int> dist(n, INF);
    dist[start] = 0;
    priority_queue<pair<int,int>,
                   vector<pair<int,int>>,
                   greater<pair<int,int>>> pq;
    pq.push({0, start});

    while (!pq.empty()) {
        auto [d, u] = pq.top(); pq.pop();
        if (d > dist[u]) continue;

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

```
            }
        }
    }
    return dist;
}
```

4 Math

4.1 Number Theory

```
// GCD and LCM
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

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

// Modular inverse
int mod_inv(int a, int m) {
    int g = gcd(a, m);
    if (g != 1) return -1;
    return mod_pow(a, m - 2, m);
}

// Fast power
int mod_pow(int base, int exp, int mod) {
    int result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp & 1) result = (1LL * result * base) % mod;
        base = (1LL * base * base) % mod;
        exp >>= 1;
    }
    return result;
}
```

5 Geometry

5.1 Convex Hull

```
// Graham scan algorithm
struct Point {
    int x, y;
    bool operator<(Point p) const {
        return x < p.x || (x == p.x && y < p.y);
    }
};

// Cross product
int cross(const Point& O, const Point& A, const Point& B) {
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}

vector<Point> convex_hull(vector<Point>& points) {
    int n = points.size(), k = 0;
    if (n <= 3) return points;
    vector<Point> hull(2*n);

    sort(points.begin(), points.end());

    // Lower hull
    for (int i = 0; i < n; i++) {
        while (k >= 2 && cross(hull[k-2], hull[k-1], points[i]) <= 0) k--;
        hull[k++] = points[i];
    }

    // Upper hull
    int t = k + 1;
    for (int i = n - 2; i >= 0; i--) {
        while (k >= t && cross(hull[k-2], hull[k-1], points[i]) <= 0) k--;
        hull[k++] = points[i];
    }

    hull.resize(k-1); // Last point is same as first
    return hull;
}
```