# Algorithmic Notes For ICPC 2021

Oscar Skean

March 2022

## 1 Template

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

#define ll long long
#define SPEED ios::sync with stdio(false); cin.tie(0); cout.tie(0)
#define pb push back
#define rsz resize
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define FOR(i,a,b) for(int i=a;i<b;++i)
#define REP(i,n) FOR(i,0,n)


int main() {
    SPEED;
}
```

## 2 Data Structures

### 2.1 Segment Tree

```cpp
// load data directly into first row of seg tree
// make sure range is [start, end+1)
const int MAXN = 2e5 + 1;
ll seg[MAXN*4];
ll n;

void construct() {
    for (ll i = n-1; i > 0; i--) {
        seg[i] = seg[i<<1]  + seg[i<<1|1];
    }
}

void update(ll pos, ll val) {
    for (seg[pos += n] = val; pos > 1; pos >>= 1) {
        seg[pos>>1] = seg[pos] + seg[pos^1];
    }
}

void query(ll l, ll r) {
    ll sum = 0;
    for (l += n, r+= n; l < r; l >>= 1, r >>= 1) {
        if (l&1) sum += seg[l++];
        if (r&1) sum += seg[--r];
    }
    cout << sum << endl;
}
```

### 2.2 Minimum Sparse

```cpp
//read directly into first row of sparse table
#define MAXLOG 20
#define MAXN 200000
int sparse[MAXN][MAXLOG];
int logs[MAXN+1];

void construct(int n) {
    //build log table
    logs[1] = 0;
    for (int i = 2; i <= MAXN; i++) {
        logs[i] = logs[i/2] + 1;
    }

    //build sparse table
    for (int row = 1; row < MAXLOG; row++) {
        for (int i = 0; i + (1 << row) <= n; i++) {
            sparse[i][row] = min(sparse[i][row-1],
                                 sparse[i + (1 << (row-1))][row-1]);
        }
    }
}

int query(int l, int r) {
    int row = logs[r - l + 1];
    return min(sparse[l][row], sparse[r - (1 << row) + 1][row]);
}
```

### 2.3 Binary Jumping

```cpp
//read directly into first row of sparse table
#define MAXLOG 20
#define MAXN 200000
int sparse[MAXN][MAXLOG];
int logs[MAXN+1];

void construct(int n) {
    //build log table
    logs[1] = 0;
    for (int i = 2; i <= MAXN; i++) {
        logs[i] = logs[i/2] + 1;
    }

    //build sparse table
    for (int row = 1; row < MAXLOG; row++) {
        for (int i = 0; i + (1 << row) <= n; i++) {
            sparse[i][row] = min(sparse[i][row-1],
                                 sparse[i + (1 << (row-1))][row-1]);
        }
    }
}

int query(int l, int r) {
    int row = logs[r - l + 1];
    return min(sparse[l][row], sparse[r - (1 << row) + 1][row]);
}
```

# 3 Graph Algorithms

## 3.1 DFS with Cycle Detection

```
void dfs(int s) {
    if (finished[s]) {
        return;
    }
    else if (seen[s]) {
        cout << "IMPOSSIBLE" << endl;
        exit(0);
    }


    seen[s] = true;
    for (int i : adj[s]) {
        dfs(i);
    }

    seen[s] = false;
    finished[s] = true;
```

## 3.2 BFS

```
const int MAXN = 1e5+1;
vector<int> adj[MAXN];

bool seen[MAXN];
int distances[MAXN];
int parents[MAXN];

void bfs(int start, int end) {
    queue<int> q;
    q.push(start);
    seen[start] = true;

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

        //early break
        if (a == end) return;

        for (int b : adj[a]) {
            if (!seen[b]) {
                seen[b] = true;
                parents[b] = a;
                distances[b] = distances[a] + 1;
                q.push(b);
            }
        }
    }
}
```

## 3.3 BFS Route Reconstruction

```
//reconstruct the bfs route from parents array
void shortest route(int start, int end) {
    //run bfs
    bfs(start, end);

    if (distances[end] == 0) {
        cout << "IMPOSSIBLE" << endl;
        return;
    }

    // build route
    int length = distances[end];
    vector<int> route(length+1);

    int loc = end;
    for (int i = length; i >= 0; i--) {
        route[i] = loc;
        loc = parents[loc];
    }

    // print route
    cout << distances[end]+1 << endl;
    for(auto a : route) {
        cout << a << " ";
    }
    cout << endl;
}
```

## 3.4 Djikstra

```
//use with edges of form <node, weight>
const int MAXN = 1e5+1;
vector<pair<int, ll>> adj[MAXN];
ll distances[MAXN];
bool seen[MAXN];

void djikstra(int start, int n) {
    FOR(i, 2, n+1) {
        distances[i] = LONG MAX;
    }

    priority queue<pair<ll, int>> q;
    q.push({0, start});

    while (!q.empty()) {
        int a = q.top().second; q.pop();

        if (!seen[a]) {
            seen[a] = true;
            for (auto e : adj[a]) {
                int b; ll w;
                tie(b, w) = e;

                if (distances[a]+w < distances[b]) {
                    distances[b] = distances[a]+w;
                    q.push({-distances[b], b});
                }
            }
        }
    }
}
```

### 3.5 Bellman-Ford

### 3.6 Floyd-Warshall

```cpp
const int MAXN = 505;
const ll bfn = 1e14;

ll distances[MAXN][MAXN];

void floyd(int n) {
    FOR(k, 1, n+1) {
        FOR(i, 1, n+1) {
            FOR(j, 1, n+1) {
                distances[i][j] = min(distances[i][j],
                        distances[i][k] + distances[k][j]);
            }
        }
    }
}

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

    // prepare distances matrix
    REP(i, n+1) {
        REP(j, n+1) {
            distances[i][j] = bfn;
        }
    }
    REP(i, m) {
        ll a, b, c;
        cin >> a >> b >> c;

        distances[a][a] = 0; distances[b][b] = 0;
        distances[a][b] = distances[b][a] = min(distances[a][b], c);
    }

    floyd(n);

    REP(i, q) {
        int a, b;
        cin >> a >> b;
        ll res = distances[a][b];
        cout << (res < bfn ? res : -1) << endl;
    }
}
```

### 3.7 Topological Sort

```cpp
const int MAXN=1e5+1;
vector<int> adj[MAXN];
bool seen[MAXN];
bool finished[MAXN];

// remember to reverse sort this when printing it
vector<int> topo;

void dfs(int s) {
    if (finished[s]) {
        return;
    }
    else if (seen[s]) {
        cout << "IMPOSSIBLE" << endl;
        exit(0);
    }


    seen[s] = true;
    for (int i : adj[s]) {
        dfs(i);
    }

    seen[s] = false;
    finished[s] = true;
    topo.push back(s);
}

void solve(int n) {
    FOR(i, 1, n+1) {
        if (!finished[i]) dfs(i);
    }
}
```

### 3.8 Kruskal with DSU

```
// put weight first in edge tuple for sorting
vector<int> parent, ranks;
vector<tuple<ll, int, int>> edges;

void make set(int v) {
    parent[v] = v;
    ranks[v] = 0;
}

int find set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find set(parent[v]);
}

void union sets(int a, int b) {
    a = find set(a);
    b = find set(b);
    if (a != b) {
        if (ranks[a] < ranks[b])
            swap(a, b);
        parent[b] = a;
        if (ranks[a] == ranks[b])
            ranks[a]++;
    }
}

// can be modified to return cost or minimal edge set
ll kruskal(int n) {
    sort(edges.begin(), edges.end());

    parent.resize(n+1);
    ranks.resize(n+1);

    FOR(i, 1, n+1) {
        make set(i);
    }

    ll cost = 0;
    vector<pair<int,int>> result;

    for (auto e : edges) {
        ll w; int u, v;
        tie(w, u, v) = e;

        if (find set(u) != find set(v)) {
            cost += w;
            result.push back({u, v});
            union sets(u, v);
        }
    }

    // check for impossibility
    if (result.size() < n-1) {
        return -1;
    }

    return cost;
}
```

### 3.9 Connected Components

For counting, use DFS and increment whenever the recursive call is completely finished. For listing, keep a vector that gets appended to during DFS. Print the vector, then reset it for the next component.

## 4 Dynamic Programming

### 4.1 Longest Increasing Subsequence

```
// performs DP algorithm
// initialize endings array to INT MAX
// endings array position i stores minimum ending to i-length increasing
void solve(int n) {
    int ans = 0;

    REP(i, n) {
        int bestLengthToAppendTo = binsearch(arr[i], 0, ans);

        if (arr[i] < endings[bestLengthToAppendTo]) {
            if (bestLengthToAppendTo == ans) {
                endings[ans] = arr[i];
                ans = max(1, ans+1);
            }
            else {
                endings[bestLengthToAppendTo] = arr[i];
            }

        }
    }

    cout << ans << endl;
}
```

### 4.2 Edit Distance

### 4.3 Coins Problem

### 4.4 Knapsack

### 4.5 Rod Cutting

## 5 Miscellaneous

### 5.1 Binary Search

```
// find  what location key should go in array
int binsearch(int key, int l, int r) {
    while (l <= r) {
        int mid = (l + r) / 2;
        if (key < arr[mid]) r = mid - 1;
        else if (key > arr[mid]) l = mid + 1;
        else return mid;
    }

    return l;
}
```

## 5.2 Binary Exponentiation

```
const ll MOD = (ll) 1e9 + 7;

void exponentiation(ll a, ll b) {
    ll val = 1;
    while (b > 0) {
        if (b & 1) {
            val *= a;
        }
        a *= a;

        a %= MOD;
        val %= MOD;
        b >>= 1;
    }

    cout << val << endl;
}
```

## 5.3 Gray Code

```
vector<string> construct(int n) {
    vector<string> vec;

    //base case
    if (n == 1) {
        vec.pb("1");
        vec.pb("0");
        return vec;
    }

    // recusive reflection algorithm
    //
    vector<string> prev = construct(n-1);
    for (auto it = prev.begin(); it != prev.end(); it++) {
        vec.pb("0" + *it);
    }

    for (auto it = prev.rbegin(); it != prev.rend(); it++) {
        vec.pb("1" + *it);
    }

    return vec;

}
```

## 5.4 Towers of Hanoi

```
// call like hanoi(n, 1, 2, 3)
vector<pair<int,int>> moves;

void hanoi(int d, int l, int m, int r) {
    if (d == 1) {
        moves.pb(make pair(l, r));
        return;
    }

    else {
        hanoi(d-1, l, r, m);
        moves.pb(make pair(l, r));
        hanoi(d-1, m, l, r);
    }
}
```

## 5.5 Josephus Queries

O(n)

```
int josephus(int n, int k) {
    int res = 0;
    for (int i = 1; i <= n; ++i)
      res = (res + k) % i;
    return res + 1;
}
```

O(klogn)

```
int josephus(int n, int k) {
    if (n == 1)
        return 0;
    if (k == 1)
        return n-1;
    if (k > n)
        return (josephus(n-1, k) + k) % n;
    int cnt = n / k;
    int res = josephus(n - cnt, k);
    res -= n % k;
    if (res < 0)
        res += n;
    else
        res += res / (k - 1);
    return res;
}
```