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

```cpp
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

```
void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    vector<int> p (n, -1);

    for (;;)
    {
        bool any = false;
        for (int j = 0; j < m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
                    d[e[j].b] = d[e[j].a] + e[j].cost;
                    p[e[j].b] = e[j].a;
                    any = true;
                }
        if (!any)  break;
    }

    if (d[t] == INF)
        cout << "No path from " << v << " to " << t << ".";
    else
    {
        vector<int> path;
        for (int cur = t; cur != -1; cur = p[cur])
            path.push back (cur);
        reverse (path.begin(), path.end());

        cout << "Path from " << v << " to " << t << ": ";
        for (size_t i=0; i<path.size(); ++i)
            cout << path[i] << ' ';
    }
}
```

3.6 Bellman-Ford with Negative Cycle Check

```
void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    vector<int> p (n - 1);
    int x;
    for (int i=0; i<n; ++i)
    {
        x = -1;
        for (int j=0; j<m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
                    d[e[j].b] = max (-INF, d[e[j].a] + e[j].cost);
                    p[e[j].b] = e[j].a;
                    x = e[j].b;
                }
    }

    if (x == -1)
        cout << "No negative cycle from " << v;
    else
    {
        int y = x;
        for (int i=0; i<n; ++i)
            y = p[y];

        vector<int> path;
        for (int cur=y; ; cur=p[cur])
        {
            path.push back (cur);
            if (cur == y && path.size() > 1)
                break;
        }
        reverse (path.begin(), path.end());

        cout << "Negative cycle: ";
        for (size_t i=0; i<path.size(); ++i)
            cout << path[i] << ' ';
    }
}
```

3.7 Floyd-Warshall

```
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;
    }
}
```

3.8 Find Bridges of Graph

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS BRIDGE(v, to);
        }
    }
}

void find bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

3.9 Topological Sort

```
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);
    }
}
```

3.10 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;
}
```

3.11 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.

3.12 Strongly 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.

3.13 Bipartite Check

3.14 Maximum Flow

3.15 Bipartite Matching

3.16 Number of Paths of Fixed Length

3.17 2SAT

3.18 TSP

3.19 Lowest Common Ancestor

3.20 Eulerian Path

3.21 Hamiltonian Path

4 Dynamic Programming

4.1 Longest Increasing Subseqeuence

```
// 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;
}
```

6.4 RabinKarp: String matching

```
vector<int> rabin karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p pow(max(S, T));
    p pow[0] = 1;
    for (int i = 1; i < (int)p pow.size(); i++)
        p pow[i] = (p pow[i-1] * p) % m;

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p pow[i]) % m;
    long long h s = 0;
    for (int i = 0; i < S; i++)
        h s = (h s + (s[i] - 'a' + 1) * p pow[i]) % m;

    vector<int> occurences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur h = (h[i+S] + m - h[i]) % m;
        if (cur h == h s * p pow[i] % m)
            occurences.push back(i);
    }
    return occurences;
}
```

6.5 Knuth-Morris-Pratt

```
vector<int> prefix function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

Uses: 1) Check if t is in s. Run KMP with the string s+#+t.

6.6 Manacher Palindroms

```
vector<int> manacherodd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 0, r = -1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}
vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}
```

7 Miscellaneous

7.1 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;
}
```

7.2 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;
}
```

7.3 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;

}
```

7.4 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);
    }
}
```

7.5 Expression Parsing

```
bool delim(char c) {
    return c == ' ';
}

bool is op(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

int priority (char op) {
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return -1;
}

void process op(stack<int>& st, char op) {
    int r = st.top(); st.pop();
    int l = st.top(); st.pop();
    switch (op) {
        case '+': st.push(l + r); break;
        case '-': st.push(l - r); break;
        case '*': st.push(l * r); break;
        case '/': st.push(l / r); break;
    }
}
```

```
int evaluate(string& s) {
    stack<int> st;
    stack<char> op;
    for (int i = 0; i < (int)s.size(); i++) {
        if (delim(s[i]))
            continue;

        if (s[i] == '(') {
            op.push('(');
        } else if (s[i] == ')') {
            while (op.top() != '(') {
                process op(st, op.top());
                op.pop();
            }
            op.pop();
        } else if (is op(s[i])) {
            char cur op = s[i];
            while (!op.empty() && priority(op.top()) >= priority(cur op)) {
                process op(st, op.top());
                op.pop();
            }
            op.push(cur op);
        } else {
            int number = 0;
            while (i < (int)s.size() && isalnum(s[i]))
                number = number * 10 + s[i++] - '0';
            --i;
            st.push(number);
        }
    }

    while (!op.empty()) {
        process op(st, op.top());
        op.pop();
    }
    return st.top();
}
```

7.6 Balanced Sequences

7.7 Korder Statistic

C++ standard library has this implemented already. The function is called *nth_element*.

```
int main()
{
    std::vector<int> v{5, 10, 6, 4, 3, 2, 6, 7, 9, 3};
    printVec(v);

    auto m = v.begin() + v.size()/2;
    std::nth_element(v.begin(), m, v.end());
    std::cout << "\nThe median is " << v[v.size()/2] << '\n';
    // The consequence of the inequality of elements before/after the Nth one:
    assert(std::accumulate(v.begin(), m, 0) < std::accumulate(m, v.end(), 0));
    printVec(v);

    // Note: comp function changed
    std::nth_element(v.begin(), v.begin()+1, v.end(), std::greater{});
    std::cout << "\nThe second largest element is " << v[1] << '\n';
    std::cout << "The largest element is " << v[0] << '\n';
```

7.8 Josephus Queries

```
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;
}
```

7.9 Convex Hull