Team Notebook

August 14, 2023

Contents

1 Graph

1.1 0-1 BFS

```
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
            if (w == 1)
                q.push_back(u);
            else
                q.push_front(u);
        }
    }
}
```

1.2 Bellman-Ford

```
#define ar array
#define ll long long

const int MAX_N = 2.5e3 + 1;
const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LINF = 1e15;

int n, m, par[MAX_N];
vector<ar<ll,2>> adj[MAX_N];
vector<ll> dist;

void bellman_ford(int s) {
    dist.assign(n + 1, LINF);
    dist[s] = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int u = 1; u <= n; u++) {
            for (auto [v, w] : adj[u]) {
                if (dist[u] + w < dist[v]) {
                    par[v] = u;
                    dist[v] = dist[u] + w;
                }
            }
        }
    }
}

void cycle_detect() {
    int cycle = 0;
    for (int u = 1; u <= n; u++) {
        for (auto [v, w] : adj[u]) {
            if (dist[u] + w < dist[v]) {
                cycle = v;
                break;
            }
        }
    }
    if (!cycle) cout << "NO\n";
    else {
        cout << "YES\n";
        // backtrack to print the cycle
        for (int i = 0; i < n; i++) cycle = par[cycle];
        vector<int> ans; ans.push_back(cycle);
        for (int i = par[cycle]; i != cycle; i = par[i]) ans.
            push_back(i);
        ans.push_back(cycle);
        reverse(ans.begin(), ans.end());
        for (int x : ans) cout << x << " ";
        cout << "\n";
    }
}

void solve() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v, w; cin >> u >> v >> w;
        adj[u].push_back({v, w});
    }
    bellman_ford(1);
    cycle_detect();
}
```

1.3 DSU

```
struct dsu {
    vt<int> par, sz;
    explicit dsu(int n)
    {
        par.assign(n+1, 0);
        iota(all(par), 0);
        sz.assign(n+1, 1);
    }
    int get_par(int x) {
        if (par[x] == x) return x;
        return par[x] = get_par(par[x]);
    }
    bool join(int a, int b) {
        a = get_par(a), b = get_par(b);
        if (a == b) return false;
        if (sz[a] > sz[b]) swap(a, b);
        par[a] = par[b];
        sz[b] += sz[a];
        return true;
    }
};
```

1.4 Dijkstra

```
vector<int> dist; // answer -> dist[destination]
vector<vector<pair<int,int>>> g; // {cost, to}

void djikstra(int src = 1) {
    dist.resize(g.size(), INFINITY);
    priority_queue<pair<int,int>, vector<pair<int,int>>,
        greater<pair<int,int>>> pq;
    pq.push({dist[src] = 0, src});
    while(!pq.empty()) {
        auto curr = pq.top(); pq.pop();
        for(auto to : g[curr.second])
            if(dist[to.second] > curr.first + to.first)
                pq.push({dist[to.second] = curr.first + to.
                    first, to.second});
    }
}
```

1.5 Floyd

```
// Find all pair shortest paths
// Time complexity: O(n^3)
// Problem link: https://cses.fi/problemset/task/1672

#include <bits/stdc++.h>

using namespace std;

#define ar array
#define ll long long
```

```cpp
const int MAX_N = 500 + 1;
const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LINF = 1e15;

int n, m, q;
ll dist[MAX_N][MAX_N];

void floyd_warshall() { // 4 lines
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist
                     [k][j]);
}

void solve() {
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            dist[i][j] = (i == j) ? 0 : LINF;
    for (int i = 0; i < m; i++) {
        int u, v, w; cin >> u >> v >> w;
        dist[u][v] = dist[v][u] = min(dist[u][v], (ll)w);
    }
    floyd_warshall();
    while (q--) {
        int u, v; cin >> u >> v;
        cout << (dist[u][v] < LINF ? dist[u][v] : -1) << "\n"
             ;
    }
}
```

2 Number theory

2.1 Fast Power

```cpp
ll power(ll b, ll p, ll mod) {
    ll res = 1;
    b = b % mod;
    if (b == 0) return 0;
    while (p)
    {
        if (p & 1)
            res = (res * b) % mod;
        p >>= 1;
        b = (b * b) % mod;
    }
    return res;
}
```

2.2 nCr with Mod Inverse

```cpp
ll modInverse(ll n, ll mod)
{
    return power(n, mod - 2, mod);
}

ll nCr(ll n, ll r, ll mod)
{
    if (n < r)
        return 0;
    if (r == 0)
        return 1;
    ll fac[n + 1];
    fac[0] = 1;
    for (int i = 1; i <= n; i++)
        fac[i] = (fac[i - 1] * i) % mod;
    return (fac[n] * modInverse(fac[r], mod) % mod
            * modInverse(fac[n - r], mod) % mod)
        % mod;
}
```