ICPC Reference Notebook

DuongNhuTruocKhiEmTonTai – University of Management and Technology HCM City

Contents

1 Template

1.1 Fast IO

```
ios::sync_with_stdio(false);
cin.tie(0); cout.tie(0);
```

1.2 Typedefs

```
typedef long long ll;
typedef pair<int, int> pii;
const int INF = 1e9;
```

2 Math

2.1 GCD / 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;
}
```

2.2 Modular Exponentiation

```
ll modpow(ll a, ll b, ll m) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
```

3 Graphs

3.1 Dijkstra's Algorithm

```
vector<pii> adj[N];
int dist[N];
void dijkstra(int src) {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    fill(dist, dist + N, INF);
    dist[src] = 0;
    pq.push({0, src});
    while (!pq.empty()) {
        int d, u;
        tie(d, u) = pq.top(); pq.pop();
        if (d > dist[u]) continue;
        for (auto [v, w] : adj[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
vector<pii> adj[N];
int dist[N];
void dijkstra(int src) {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    fill(dist, dist + N, INF);
    dist[src] = 0;
    pq.push({0, src});
    while (!pq.empty()) {
        int d, u;
        tie(d, u) = pq.top(); pq.pop();
        if (d > dist[u]) continue;
        for (auto [v, w] : adj[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
vector<pii> adj[N];
int dist[N];
void dijkstra(int src) {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    fill(dist, dist + N, INF);
    dist[src] = 0;
    pq.push({0, src});
    while (!pq.empty()) {
        int d, u;
        tie(d, u) = pq.top(); pq.pop();
        if (d > dist[u]) continue;
        for (auto [v, w] : adj[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
vector<pii> adj[N];
int dist[N];
void dijkstra(int src) {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    fill(dist, dist + N, INF);
    dist[src] = 0;
    pq.push({0, src});
    while (!pq.empty()) {
        int d, u;
        tie(d, u) = pq.top(); pq.pop();
        if (d > dist[u]) continue;
        for (auto [v, w] : adj[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}vector<pii> adj[N];
int dist[N];
void dijkstra(int src) {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    fill(dist, dist + N, INF);
    dist[src] = 0;
    pq.push({0, src});
    while (!pq.empty()) {
        int d, u;
        tie(d, u) = pq.top(); pq.pop();
        if (d > dist[u]) continue;
        for (auto [v, w] : adj[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
```