**CHƯƠNG 6: ĐỒ THỊ (GRAPH)**

**1. KHÁI NIỆM ĐỒ THỊ**

* **Đồ thị (Graph)** là một tập gồm các đỉnh (**vertex**) và các cạnh (**edge**).
* G = (V, E) với:
  + V: tập hợp các đỉnh.
  + E: tập hợp các cạnh (nối giữa các đỉnh).

**Phân loại đồ thị:**

* **Đồ thị vô hướng** và **đồ thị có hướng**.
* **Đồ thị có trọng số** (mỗi cạnh có một giá trị).
* **Đồ thị liên thông** / **không liên thông**.

**2. BIỂU DIỄN ĐỒ THỊ**

**2.1 Ma trận kề (Adjacency Matrix)**

const int MAX = 100;

int adjMatrix[MAX][MAX]; // adjMatrix[i][j] = 1 nếu có cạnh i -> j

**2.2 Danh sách kề (Adjacency List)**

#include <vector>

using namespace std;

vector<int> adjList[MAX]; // adjList[i] chứa các đỉnh kề với đỉnh i

**3. DUYỆT ĐỒ THỊ**

**3.1 Duyệt theo chiều sâu (DFS)**

#include <iostream>

#include <vector>

using namespace std;

const int MAX = 100;

vector<int> adj[MAX];

bool visited[MAX];

void DFS(int u) {

visited[u] = true;

cout << u << " ";

for (int v : adj[u]) {

if (!visited[v])

DFS(v);

}

}

**3.2 Duyệt theo chiều rộng (BFS)**

#include <queue>

void BFS(int start) {

queue<int> q;

bool visited[MAX] = {false};

visited[start] = true;

q.push(start);

while (!q.empty()) {

int u = q.front(); q.pop();

cout << u << " ";

for (int v : adj[u]) {

if (!visited[v]) {

visited[v] = true;

q.push(v);

}

}

}

}

**4. ĐỒ THỊ CÓ TRỌNG SỐ**

**4.1 Dijkstra – Tìm đường đi ngắn nhất từ 1 đỉnh**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

const int INF = 1e9;

const int MAX = 100;

vector<pair<int, int>> adj[MAX]; // {đỉnh kề, trọng số}

void Dijkstra(int start, int n) {

vector<int> dist(n + 1, INF);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;

dist[start] = 0;

pq.push({0, start});

while (!pq.empty()) {

int d = pq.top().first, u = pq.top().second; 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});

}

}

}

for (int i = 1; i <= n; ++i)

cout << "Khoảng cách từ " << start << " đến " << i << ": " << dist[i] << endl;

}

**4.2 Floyd – Tìm đường đi ngắn nhất giữa mọi cặp đỉnh**

void Floyd(int n, int dist[MAX][MAX]) {

for (int k = 1; k <= n; ++k)

for (int i = 1; i <= n; ++i)

for (int j = 1; j <= n; ++j)

if (dist[i][j] > dist[i][k] + dist[k][j])

dist[i][j] = dist[i][k] + dist[k][j];

}

**4.3 Bellman-Ford – Chấp nhận cạnh có trọng số âm**

void BellmanFord(int start, int n, vector<tuple<int, int, int>> &edges) {

vector<int> dist(n + 1, INF);

dist[start] = 0;

for (int i = 1; i < n; ++i) {

for (auto [u, v, w] : edges) {

if (dist[u] != INF && dist[v] > dist[u] + w)

dist[v] = dist[u] + w;

}

}

// Kiểm tra chu trình âm

for (auto [u, v, w] : edges) {

if (dist[u] != INF && dist[v] > dist[u] + w) {

cout << "Đồ thị có chu trình âm!\n";

return;

}

}

for (int i = 1; i <= n; ++i)

cout << "Từ " << start << " đến " << i << ": " << dist[i] << endl;

}