Thực hành 4

Bài 1: Đồ thị Euler

Ý tưởng

-Tính **in-degree** và **out-degree** mỗi đỉnh.

-Nếu mọi đỉnh có in = out và đồ thị liên thông → có chu trình Euler.

-Dùng **Hierholzer**:

* Đi DFS, mỗi lần đi qua cạnh thì xóa cạnh đó.
* Khi hết cạnh thì thêm đỉnh vào kết quả.
* Đảo ngược kết quả sẽ ra chu trình Euler.

#include <iostream>

using namespace std;

const int MAX = 1005;

int n;

int adj[MAX][MAX]; // ma trận kề

int inDeg[MAX], outDeg[MAX];

int path[1000000]; // lưu chu trình Euler

int idx = 0;

void euler(int u) {

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

while (adj[u][v] > 0) {

adj[u][v]--;

euler(v);

}

}

path[idx++] = u;

}

int main() {

cin >> n;

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

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

cin >> adj[i][j];

if (adj[i][j]) {

outDeg[i] += adj[i][j];

inDeg[j] += adj[i][j];

}

}

}

// Kiểm tra điều kiện Euler có hướng

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

if (inDeg[i] != outDeg[i]) {

cout << 0;

return 0;

}

}

// Có Euler cycle

cout << 1 << "\n";

// Tìm chu trình Euler bắt đầu từ đỉnh 1

euler(1);

// In kết quả (đường đi đảo ngược do quay lui)

for (int i = idx - 1; i >= 0; i--) {

cout << path[i] << " ";

}

return 0;

}

Bài 2: Domino

Ý tưởng

* Mỗi số chấm (0–6) là một **đỉnh** của đồ thị.
* Mỗi quân domino (x, y) là một **cạnh** nối giữa hai đỉnh x và y.
* Đặt domino thành **một vòng tròn hợp lệ** ⇔ tìm được **chu trình Euler** trong đồ thị (dùng hết tất cả cạnh đúng 1 lần và quay về điểm xuất phát).

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

ios::sync\_with\_stdio(false);

cin.tie(NULL);

int n;

if (!(cin >> n)) return 0;

const int V = 7; // số đỉnh 0..6

int adj[V][V] = { 0 };

int deg[V] = { 0 };

// đọc n quân domino

for (int i = 0; i < n; ++i) {

int x, y;

cin >> x >> y;

// giả sử input hợp lệ: 0 <= x,y <= 6

adj[x][y] += 1;

adj[y][x] += 1;

deg[x]++; deg[y]++;

}

// kiểm tra mọi đỉnh có bậc chẵn

for (int i = 0; i < V; ++i) {

if (deg[i] % 2 != 0) {

cout << 0 << '\n';

return 0;

}

}

// kiểm tra liên thông với các đỉnh có bậc > 0

int start = -1;

for (int i = 0; i < V; ++i) if (deg[i] > 0) { start = i; break; }

if (start == -1) {

// không có cạnh nào — coi như có chu trình rỗng?

// theo đề nếu n=0 có thể in 1 và không in dòng tiếp

cout << 1 << '\n';

return 0;

}

vector<int> stk;

vector<bool> vis(V, false);

// DFS để check liên thông (vô hướng)

vector<int> s;

s.push\_back(start);

vis[start] = true;

while (!s.empty()) {

int u = s.back(); s.pop\_back();

for (int v = 0; v < V; ++v) {

if (adj[u][v] > 0 && !vis[v]) {

vis[v] = true;

s.push\_back(v);

}

}

}

for (int i = 0; i < V; ++i) {

if (deg[i] > 0 && !vis[i]) {

cout << 0 << '\n';

return 0;

}

}

// Hierholzer (iterative) để tìm chu trình Euler

vector<int> circuit; // sẽ chứa dãy đỉnh (edges+1)

stk.push\_back(start);

while (!stk.empty()) {

int v = stk.back();

int u\_found = -1;

for (int u = 0; u < V; ++u) {

if (adj[v][u] > 0) {

u\_found = u;

break;

}

}

if (u\_found != -1) {

// dùng cạnh v-u\_found

adj[v][u\_found]--;

adj[u\_found][v]--;

stk.push\_back(u\_found);

}

else {

// không còn cạnh đi tiếp từ v

circuit.push\_back(v);

stk.pop\_back();

}

}

// circuit bây giờ là dãy đỉnh theo thứ tự ngược (từ cuối về đầu),

// đảo ngược để được chu trình đúng: v0, v1, ..., vk (k = n)

reverse(circuit.begin(), circuit.end());

// kiểm tra số cạnh có đúng n không (circuit.size() phải = n+1)

if ((int)circuit.size() != n + 1) {

// không dùng hết tất cả cạnh → không hợp lệ

cout << 0 << '\n';

return 0;

}

// In ra theo đề: 1 rồi n dòng domino (cặp đỉnh consecutive)

cout << 1 << '\n';

for (size\_t i = 0; i + 1 < circuit.size(); ++i) {

cout << circuit[i] << " " << circuit[i + 1] << '\n';

}

return 0;

}

Bài 4: Đồ thị Euler và Hamilton

Ý tưởng

 Xử lý đồ thị **vô hướng đa cạnh** (có thể có nhiều cạnh giữa hai đỉnh).

 Kiểm tra: mọi đỉnh có bậc chẵn và các đỉnh có cạnh nằm trong cùng một thành phần liên thông.

 Dùng **Hierholzer** để tìm chu trình Euler và in ra danh sách các cạnh theo thứ tự chu trình.

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

int main() {

ios::sync\_with\_stdio(false);

cin.tie(NULL);

int m;

if (!(cin >> m)) return 0;

// We'll allow vertices numbered up to, say, 10000 (tăng nếu cần)

const int MAXV = 10005;

// We'll discover max vertex value from input to shrink loops

vector<pair<int, int>> edges; edges.reserve(m + 1);

int u, v;

int maxv = 0;

for (int i = 0; i < m; ++i) {

cin >> u >> v;

edges.push\_back({ u, v });

if (u > maxv) maxv = u;

if (v > maxv) maxv = v;

}

int V = maxv + 1;

vector<vector<pair<int, int>>> adj(V); // (neighbor, edge\_id)

vector<int> deg(V, 0);

for (int i = 0; i < m; ++i) {

u = edges[i].first;

v = edges[i].second;

adj[u].push\_back({ v, i });

adj[v].push\_back({ u, i });

deg[u]++; deg[v]++;

}

// Check all vertices with degree>0 are in one connected component

int start = -1;

for (int i = 0; i < V; ++i) if (deg[i] > 0) { start = i; break; }

if (start == -1) {

// no edges: depending đề, treat as Euler (empty). We'll print 1 and nothing else.

cout << 1 << '\n';

return 0;

}

// connectivity DFS (on underlying undirected graph)

vector<char> seen(V, 0);

stack<int> stc;

stc.push(start);

seen[start] = 1;

while (!stc.empty()) {

int x = stc.top(); stc.pop();

for (auto& p : adj[x]) {

int nb = p.first;

if (!seen[nb]) { seen[nb] = 1; stc.push(nb); }

}

}

for (int i = 0; i < V; ++i) {

if (deg[i] > 0 && !seen[i]) {

cout << 0 << '\n';

return 0;

}

}

// Check all degrees even

for (int i = 0; i < V; ++i) if (deg[i] % 2 != 0) {

cout << 0 << '\n';

return 0;

}

// Hierholzer: iterative using stack, track used edges

vector<char> usedEdge(m, 0);

vector<int> edgePos(V, 0); // next edge index to try for each vertex

vector<int> vertexStack;

vector<int> circuitVertices; // will hold vertices in order (length = m+1)

vertexStack.push\_back(start);

while (!vertexStack.empty()) {

int vcur = vertexStack.back();

// find next unused edge from vcur

while (edgePos[vcur] < (int)adj[vcur].size() && usedEdge[adj[vcur][edgePos[vcur]].second]) {

edgePos[vcur]++;

}

if (edgePos[vcur] == (int)adj[vcur].size()) {

// no more edges -> add to circuit

circuitVertices.push\_back(vcur);

vertexStack.pop\_back();

}

else {

auto pr = adj[vcur][edgePos[vcur]];

int nxt = pr.first;

int eid = pr.second;

usedEdge[eid] = 1;

vertexStack.push\_back(nxt);

// advance edgePos[vcur] (next time skip this edge)

edgePos[vcur]++;

}

}

// circuitVertices should have length m+1

if ((int)circuitVertices.size() != m + 1) {

cout << 0 << '\n';

return 0;

}

// Print result: 1 then each consecutive pair as an edge in the traversal

cout << 1 << '\n';

for (int i = 0; i + 1 < (int)circuitVertices.size(); ++i) {

cout << circuitVertices[i] << " " << circuitVertices[i + 1] << '\n';

}

return 0;

}

Bài 3: Đồ thi nua Euler

Ý tưởng

 **Đọc ma trận kề** và tính **bậc của từng đỉnh**.

 **Kiểm tra điều kiện Euler**:

* Nếu có **hơn 2 đỉnh bậc lẻ** → không phải đồ thị nửa Euler → in 0.
* Nếu có **0 hoặc 2 đỉnh bậc lẻ** → là đồ thị nửa Euler → tiếp tục.

 **Xác định điểm bắt đầu**:

* Nếu có 2 đỉnh bậc lẻ → bắt đầu từ một trong hai đỉnh đó.
* Nếu tất cả bậc chẵn → bắt đầu từ đỉnh bất kỳ có cạnh.

 **Dùng thuật toán Hierholzer**:

* Dùng ngăn xếp để đi theo cạnh chưa dùng.
* Khi không đi được nữa, đưa đỉnh vào kết quả.
* Đảo ngược kết quả cuối cùng thành đường đi Euler.

 **In kết quả**:

* Dòng 1: 1.
* Dòng 2: dãy đỉnh của đường đi Euler.

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

int main() {

ios::sync\_with\_stdio(false);

cin.tie(NULL);

int n;

cin >> n;

vector<vector<int>> adj(n + 1, vector<int>(n + 1));

vector<int> deg(n + 1, 0);

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

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

cin >> adj[i][j];

deg[i] += adj[i][j];

}

}

// Đếm số đỉnh bậc lẻ

vector<int> odd;

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

if (deg[i] % 2 == 1) odd.push\_back(i);

}

if (!(odd.size() == 0 || odd.size() == 2)) {

cout << 0;

return 0;

}

// Tìm điểm bắt đầu

int start = 1;

if (odd.size() == 2) start = odd[0];

else {

for (int i = 1; i <= n; i++) if (deg[i] > 0) { start = i; break; }

}

// Hierholzer để tìm đường đi Euler

vector<int> path;

stack<int> st;

st.push(start);

while (!st.empty()) {

int u = st.top();

bool found = false;

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

if (adj[u][v] > 0) {

found = true;

adj[u][v]--;

adj[v][u]--;

st.push(v);

break;

}

}

if (!found) {

path.push\_back(u);

st.pop();

}

}

cout << 1 << "\n";

for (int i = 0; i < (int)path.size(); i++) {

cout << path[i] << (i + 1 == (int)path.size() ? '\n' : ' ');

}

return 0;

}

Bài 4: Đồ thị Hamilton

 **Ý tưởng giải**:

* **Quay lui (Backtracking)**:
  + Bắt đầu từ một đỉnh (thường là đỉnh 1).
  + Duyệt thử từng đỉnh tiếp theo (theo ma trận kề), chỉ chọn đỉnh chưa được thăm.
  + Nếu đi đủ n đỉnh và cạnh cuối nối về đỉnh xuất phát → tìm thấy chu trình Hamilton.
* Nếu không tìm được đường đi thỏa mãn → in 0.
* Nếu có chu trình Hamilton → in 1 và đường đi.
* Nếu không có → in 0.

#include <iostream>

using namespace std;

int n;

int a[105][105]; // ma trận kề

int path[105]; // lưu chu trình

bool visited[105];

bool found = false;

void printPath() {

for (int i = 0; i <= n; i++) {

cout << path[i];

if (i < n) cout << " ";

}

cout << endl;

}

void Try(int k) {

if (found) return; // nếu đã tìm được rồi thì dừng

for (int v = 2; v <= n; v++) { // thử từ đỉnh 2 trở đi

if (!visited[v] && a[path[k - 1]][v]) {

path[k] = v;

visited[v] = true;

if (k == n) {

if (a[path[n]][path[0]]) { // có cạnh quay về đỉnh xuất phát

found = true;

cout << 1 << endl;

printPath();

}

}

else {

Try(k + 1);

}

visited[v] = false;

}

}

}

int main() {

cin >> n;

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

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

cin >> a[i][j];

}

}

path[0] = 1;

visited[1] = true;

Try(1);

if (!found) cout << 0 << endl;

return 0;

}

Bài 6 Euler trên danh sách kề

## Ý tưởng:

1. Dùng **stack** để mô phỏng quá trình duyệt Euler:
   * Bắt đầu từ đỉnh xuất phát.
   * Nếu còn cạnh đi từ đỉnh hiện tại → đi theo cạnh đó, xóa cạnh, đẩy đỉnh tiếp theo vào stack.
   * Nếu không còn cạnh → pop khỏi stack và thêm vào kết quả.
2. Vì ta xử lý bằng stack, đường đi cuối cùng sẽ có thứ tự ngược → cần đảo lại để được chu trình Euler.

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

int n, m; // số đỉnh, số cạnh

vector<vector<int>> adj; // danh sách kề

vector<int> res;

void euler\_iterative(int start) {

stack<int> st;

st.push(start);

while (!st.empty()) {

int u = st.top();

if (!adj[u].empty()) {

int v = adj[u].back();

adj[u].pop\_back();

// xóa cạnh ngược lại

for (auto it = adj[v].begin(); it != adj[v].end(); it++) {

if (\*it == u) {

adj[v].erase(it);

break;

}

}

st.push(v);

}

else {

res.push\_back(u);

st.pop();

}

}

}

int main() {

cin >> n >> m;

adj.assign(n + 1, {});

for (int i = 0; i < m; i++) {

int u, v;

cin >> u >> v;

adj[u].push\_back(v);

adj[v].push\_back(u); // đồ thị vô hướng

}

euler\_iterative(1); // bắt đầu từ đỉnh 1

for (int i = res.size() - 1; i >= 0; i--) {

cout << res[i] << " ";

}

cout << endl;

return 0;

}