Tên: Huỳnh Công Tính

MSSV: 3124411315

Github: congtinh06

Lớp: DCT124C6

1)

#include <iostream>

#include <vector>

#include <fstream>

#include <stack>

using namespace std;

int n;

vector<vector<int> > adj;

vector<int> eulerPath;

void dfs(int u, vector<vector<int> > &graph) {

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

if (graph[u][v] == 1) {

graph[u][v] = graph[v][u] = 0;

dfs(v, graph);

}

}

eulerPath.push\_back(u + 1);

}

bool isConnected(vector<vector<int> > &graph) {

vector<int> visited(n, 0);

stack<int> st;

int start = -1;

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

int deg = 0;

for (int j = 0; j < n; j++) deg += graph[i][j];

if (deg > 0) {

start = i;

break;

}

}

if (start == -1) return true;

st.push(start);

visited[start] = 1;

while (!st.empty()) {

int u = st.top(); st.pop();

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

if (graph[u][v] && !visited[v]) {

visited[v] = 1;

st.push(v);

}

}

}

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

int deg = 0;

for (int j = 0; j < n; j++) deg += graph[i][j];

if (deg > 0 && !visited[i]) return false;

}

return true;

}

int main() {

ifstream fin("DoThiEuler.inp");

ofstream fout("DoThiEuler.out");

fin >> n;

adj.assign(n, vector<int>(n, 0));

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

for (int j = 0; j < n; j++)

fin >> adj[i][j];

bool ok = true;

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

int deg = 0;

for (int j = 0; j < n; j++) deg += adj[i][j];

if (deg % 2 != 0) {

ok = false;

break;

}

}

if (!isConnected(adj)) ok = false;

if (!ok) {

fout << 0;

return 0;

}

fout << 1 << endl;

vector<vector<int> > temp = adj;

dfs(0, temp);

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

fout << eulerPath[i];

if (i > 0) fout << " ";

}

fin.close();

fout.close();

return 0;

}

Ý tưởng thuật toán:

1️⃣ Đọc đồ thị từ file DoThiEuler.inp dưới dạng ma trận kề.

2️⃣ Kiểm tra điều kiện cần của đồ thị Euler:

* Đồ thị phải liên thông (mọi đỉnh có bậc > 0 đều nằm trong cùng một thành phần liên thông).
* Mọi đỉnh phải có bậc chẵn (tổng cạnh nối đến nó là số chẵn).

3️⃣ Nếu không thỏa điều kiện, ghi 0 ra file (đồ thị không có chu trình Euler).

4️⃣ Nếu thỏa điều kiện, áp dụng thuật toán Hierholzer (dạng đệ quy):

* Bắt đầu từ một đỉnh bất kỳ (thường là đỉnh 0).
* Duyệt theo cạnh còn lại (giá trị 1 trong ma trận kề).
* Khi đi qua cạnh (u, v), ta xóa cạnh đó khỏi đồ thị (đặt graph[u][v] = graph[v][u] = 0).
* Gọi đệ quy sang v.
* Sau khi không còn cạnh để đi, thêm u vào danh sách kết quả (EulerPath).
* Kết quả thu được là đường đi Euler ngược chiều, nên cần in ngược lại.

5️⃣ Ghi kết quả ra file:

* Ghi 1 (đồ thị có chu trình Euler).
* Sau đó ghi ra thứ tự các đỉnh trong chu trình Euler.

Độ phức tạp:

* Duyệt qua tất cả cạnh đúng một lần → O(E)
* Kiểm tra điều kiện và liên thông: O(V^2)  
  → Tổng thể: O(V²) với ma trận kề.

2)

#include <iostream>

#include <vector>

#include <fstream>

#include <stack>

using namespace std;

int n;

vector<vector<int> > adj;

vector<pair<int, int> > domino;

vector<pair<int, int> > euler;

void dfs(int u, vector<vector<int> > &graph) {

for (int v = 0; v < 7; v++) {

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

graph[u][v]--;

graph[v][u]--;

dfs(v, graph);

euler.push\_back(make\_pair(u, v));

}

}

}

bool isConnected(vector<vector<int> > &graph) {

vector<int> visited(7, 0);

stack<int> st;

int start = -1;

for (int i = 0; i < 7; i++) {

int deg = 0;

for (int j = 0; j < 7; j++) deg += graph[i][j];

if (deg > 0) {

start = i;

break;

}

}

if (start == -1) return true;

st.push(start);

visited[start] = 1;

while (!st.empty()) {

int u = st.top(); st.pop();

for (int v = 0; v < 7; v++) {

if (graph[u][v] && !visited[v]) {

visited[v] = 1;

st.push(v);

}

}

}

for (int i = 0; i < 7; i++) {

int deg = 0;

for (int j = 0; j < 7; j++) deg += graph[i][j];

if (deg > 0 && !visited[i]) return false;

}

return true;

}

int main() {

ifstream fin("Domino.inp");

ofstream fout("Domino.out");

fin >> n;

adj.assign(7, vector<int>(7, 0));

domino.resize(n);

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

int x, y;

fin >> x >> y;

domino[i] = make\_pair(x, y);

adj[x][y]++;

adj[y][x]++;

}

bool ok = true;

for (int i = 0; i < 7; i++) {

int deg = 0;

for (int j = 0; j < 7; j++) deg += adj[i][j];

if (deg % 2 != 0) {

ok = false;

break;

}

}

if (!isConnected(adj)) ok = false;

if (!ok) {

fout << 0;

return 0;

}

fout << 1 << endl;

vector<vector<int> > temp = adj;

dfs(0, temp);

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

fout << euler[i].first << " " << euler[i].second << endl;

}

fin.close();

fout.close();

return 0;

}

Ý tưởng thuật toán:

1️⃣ Mô hình hóa bài toán

* Ta coi mỗi mặt số (0–6) là một đỉnh của đồ thị.
* Mỗi quân domino (x, y) tương ứng với một cạnh nối giữa hai đỉnh x và y.
* Vì domino không có hướng (x–y như y–x), nên đồ thị là đồ thị vô hướng.

➡ Bài toán trở thành:

“Có thể đi qua mỗi cạnh đúng một lần trong đồ thị hay không?”

Tức là tìm chu trình Euler hoặc đường đi Euler.

2️⃣ Điều kiện tồn tại chu trình/đường đi Euler

Đối với đồ thị vô hướng:

* Chu trình Euler tồn tại nếu:
  + Đồ thị liên thông (không tính các đỉnh bậc 0).
  + Tất cả các đỉnh đều có bậc chẵn.
* Đường đi Euler tồn tại nếu:
  + Đồ thị liên thông.
  + Có chính xác 2 đỉnh bậc lẻ.

→ Trong bài này, code kiểm tra tất cả đỉnh đều có bậc chẵn, nghĩa là tìm chu trình Euler.

3️⃣ Kiểm tra liên thông

Hàm isConnected() dùng DFS (stack) để kiểm tra:

* Tìm đỉnh bắt đầu đầu tiên có cạnh.
* Duyệt DFS toàn bộ đồ thị.
* Sau khi duyệt xong, nếu còn đỉnh nào có cạnh mà chưa được thăm → đồ thị không liên thông → không thể có chu trình Euler.

4️⃣ Thuật toán tìm chu trình Euler

Dùng thuật toán Hierholzer:

* Bắt đầu từ một đỉnh u (thường là 0).
* Duyệt các cạnh (u, v):
  + Nếu tồn tại cạnh, xóa cạnh đó (graph[u][v]-- và graph[v][u]--).
  + Gọi đệ quy dfs(v).
  + Khi quay lui, ghi lại cạnh (u, v) vào danh sách euler.

➡ Sau khi duyệt xong, danh sách euler chứa các cạnh theo thứ tự ngược của chu trình.

5️⃣ Xuất kết quả

* Nếu không thỏa điều kiện Euler → in 0.
* Nếu có → in 1 và sau đó in danh sách các quân domino theo thứ tự chu trình Euler

3)

#include <iostream>

#include <vector>

#include <stack>

#include <fstream>

using namespace std;

int n;

vector<vector<int> > adj;

vector<int> visited;

void dfs(int u) {

visited[u] = 1;

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

if (adj[u][v] && !visited[v])

dfs(v);

}

}

bool isConnected() {

visited.assign(n, 0);

int start = -1;

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

int deg = 0;

for (int j = 0; j < n; j++)

deg += adj[i][j];

if (deg > 0) {

start = i;

break;

}

}

if (start == -1) return false;

dfs(start);

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

int deg = 0;

for (int j = 0; j < n; j++)

deg += adj[i][j];

if (deg > 0 && !visited[i])

return false;

}

return true;

}

vector<int> findEulerPath(int start) {

vector<int> path;

stack<int> st;

st.push(start);

vector<vector<int> > temp = adj;

while (!st.empty()) {

int u = st.top();

int v;

for (v = 0; v < n; v++)

if (temp[u][v]) break;

if (v == n) {

path.push\_back(u + 1);

st.pop();

} else {

st.push(v);

temp[u][v] = temp[v][u] = 0;

}

}

return path;

}

int main() {

ifstream fin("DoThiNuaEuler.inp");

ofstream fout("DoThiNuaEuler.out");

fin >> n;

adj.assign(n, vector<int>(n, 0));

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

for (int j = 0; j < n; j++)

fin >> adj[i][j];

if (!isConnected()) {

fout << 0;

return 0;

}

int odd = 0, start = 0;

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

int deg = 0;

for (int j = 0; j < n; j++)

deg += adj[i][j];

if (deg % 2 == 1) {

odd++;

start = i;

}

}

if (odd != 2) {

fout << 0;

return 0;

}

fout << 1 << endl;

vector<int> path = findEulerPath(start);

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

fout << path[i] << " ";

fin.close();

fout.close();

return 0;

}

Ý tưởng thuật toán:

Khái niệm cơ bản

* Chu trình Euler (Euler Circuit): đi qua mọi cạnh đúng 1 lần, bắt đầu và kết thúc cùng đỉnh.
* Đường đi Euler (Euler Path): đi qua mọi cạnh đúng 1 lần, bắt đầu và kết thúc ở 2 đỉnh khác nhau.

➡ Bài này tìm đường đi Euler (nửa Euler).

2️⃣ Điều kiện tồn tại đường đi Euler (vô hướng)

Một đồ thị vô hướng có đường đi Euler khi và chỉ khi:

1. Đồ thị liên thông (không tính đỉnh bậc 0).
2. Có chính xác 2 đỉnh có bậc lẻ (bắt đầu và kết thúc đường đi).

Nếu có:

* Bắt đầu từ 1 trong 2 đỉnh bậc lẻ.
* Duyệt tất cả các cạnh sao cho mỗi cạnh được đi đúng 1 lần.

3️⃣ Kiểm tra liên thông

Hàm isConnected() làm nhiệm vụ:

* Tìm đỉnh đầu tiên có cạnh (deg > 0) → chọn làm start.
* Duyệt DFS từ start.
* Sau đó kiểm tra xem tất cả các đỉnh có cạnh đều được thăm chưa.
* Nếu còn đỉnh chưa được thăm → đồ thị không liên thông → không có đường đi Euler.

4️⃣ Xác định đỉnh bậc lẻ

Sau khi đảm bảo đồ thị liên thông:

* Duyệt qua tất cả các đỉnh.
* Đếm số đỉnh có bậc lẻ (deg % 2 == 1).
* Nếu không đúng 2 đỉnh → không tồn tại đường đi Euler.

5️⃣ Thuật toán tìm đường đi Euler

Sử dụng thuật toán Hierholzer (phiên bản không đệ quy):

Ý tưởng chính:

* Dùng một stack để lưu các đỉnh đang đi qua.
* Duyệt từ đỉnh start.
* Nếu còn cạnh (u, v) chưa đi → đi tiếp, xóa cạnh đó, push v vào stack.
* Nếu không còn cạnh → pop khỏi stack, thêm vào path.
* Sau cùng, path sẽ chứa đường đi Euler (theo thứ tự ngược).

6️⃣ Xuất kết quả

* Nếu đồ thị không liên thông hoặc không có đúng 2 đỉnh bậc lẻ → in 0.
* Nếu có đường đi Euler → in 1, sau đó in ra toàn bộ các đỉnh trong đường đi (theo thứ tự).

4)

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

int n;

vector<vector<int> > adj;

vector<int> path;

vector<int> visited;

bool found = false;

void Try(int k) {

if (found) return;

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

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

path[k] = v;

visited[v] = 1;

if (k == n) {

if (adj[v][path[1]]) found = true;

} else Try(k + 1);

visited[v] = 0;

if (found) return;

}

}

}

int main() {

ifstream fin("Hamilton.inp");

ofstream fout("Hamilton.out");

fin >> n;

adj.assign(n + 1, vector<int>(n + 1, 0));

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

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

fin >> adj[i][j];

path.assign(n + 1, 0);

visited.assign(n + 1, 0);

path[1] = 1;

visited[1] = 1;

Try(2);

if (!found) fout << 0;

else {

fout << 1 << endl;

for (int i = 1; i <= n; i++) fout << path[i] << " ";

fout << path[1];

}

fin.close();

fout.close();

return 0;

}

Ý tưởng thuật toán:

1️⃣ Khái niệm cơ bản

* Chu trình Hamilton (Hamiltonian Cycle) là một chu trình đi qua mỗi đỉnh đúng 1 lần, và trở về đỉnh ban đầu.
* Khác với chu trình Euler (đi qua mọi *cạnh* đúng 1 lần), bài này yêu cầu đi qua mọi *đỉnh* đúng 1 lần.

2️⃣ Phương pháp giải: Duyệt quay lui (Backtracking)

Ý tưởng chính:

* Duyệt tất cả các hoán vị có thể của các đỉnh để tạo chu trình.
* Với mỗi bước, chọn một đỉnh chưa thăm mà có cạnh nối với đỉnh hiện tại.
* Nếu đến bước thứ n mà đỉnh cuối nối được lại với đỉnh đầu → tìm thấy chu trình Hamilton.

3️⃣ Biến và cấu trúc sử dụng

| Biến | Ý nghĩa |
| --- | --- |
| adj | Ma trận kề lưu cạnh (adj[i][j] = 1 nếu có cạnh). |
| path | Mảng lưu thứ tự các đỉnh trong chu trình. |
| visited | Mảng đánh dấu đỉnh đã đi qua. |
| found | Biến cờ báo hiệu đã tìm được chu trình. |

4️⃣ Hàm đệ quy Try(k)

Hàm Try(k) cố gắng chọn đỉnh thứ k của chu trình.

Các bước:

1. Duyệt tất cả các đỉnh v từ 1 đến n.
2. Nếu v chưa thăm và có cạnh từ path[k-1] → v:
   * Gán path[k] = v, đánh dấu visited[v] = 1.
3. Nếu k == n:
   * Kiểm tra xem đỉnh cuối có nối được với path[1] không (chu trình khép kín).
   * Nếu có → found = true.
4. Nếu chưa đủ n đỉnh → gọi đệ quy Try(k+1).
5. Quay lui: bỏ đánh dấu visited[v] = 0.

5)

#include <iostream>

#include <vector>

#include <stack>

#include <fstream>

#include <numeric>

using namespace std;

int n;

vector<vector<int> > adj;

bool isConnected(vector<vector<int> > &graph) {

vector<int> visited(n, 0);

stack<int> st;

int start = -1;

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

int deg = accumulate(graph[i].begin(), graph[i].end(), 0);

if (deg > 0) {

start = i;

break;

}

}

if (start == -1) return true;

st.push(start);

visited[start] = 1;

while (!st.empty()) {

int u = st.top();

st.pop();

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

if (graph[u][v] && !visited[v]) {

visited[v] = 1;

st.push(v);

}

}

}

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

int deg = accumulate(graph[i].begin(), graph[i].end(), 0);

if (deg > 0 && !visited[i])

return false;

}

return true;

}

vector<int> eulerIterative(int start) {

vector<int> path;

stack<int> st;

vector<vector<int> > temp = adj;

st.push(start);

while (!st.empty()) {

int u = st.top();

int v;

for (v = 0; v < n; v++)

if (temp[u][v]) break;

if (v == n) {

path.push\_back(u + 1);

st.pop();

} else {

st.push(v);

temp[u][v] = temp[v][u] = 0;

}

}

return path;

}

int main() {

ifstream fin("EulerKhongDeQuy.inp");

ofstream fout("EulerKhongDeQuy.out");

fin >> n;

adj.assign(n, vector<int>(n, 0));

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

for (int j = 0; j < n; j++)

fin >> adj[i][j];

bool ok = true;

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

int deg = accumulate(adj[i].begin(), adj[i].end(), 0);

if (deg % 2 != 0) {

ok = false;

break;

}

}

if (!isConnected(adj)) ok = false;

if (!ok) {

fout << 0;

return 0;

}

fout << 1 << endl;

vector<int> res = eulerIterative(0);

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

fout << res[i];

if (i > 0) fout << " ";

}

fin.close();

fout.close();

return 0;

}

Ý tưởng thuật toán:

Ý tưởng bài toán

1. Chu trình Euler là đường đi qua mọi cạnh đúng 1 lần và quay lại đỉnh xuất phát.  
   Để tồn tại chu trình Euler trong đồ thị vô hướng, cần thỏa mãn:
   * Đồ thị liên thông (bỏ các đỉnh cô lập).
   * Mọi đỉnh đều có bậc chẵn.
2. Không dùng đệ quy → ta sử dụng ngăn xếp (stack) để mô phỏng thuật toán Fleury hoặc Hierholzer.

Bước giải thuật chi tiết

1️⃣ Kiểm tra điều kiện tồn tại chu trình Euler

* Duyệt toàn bộ ma trận kề.
* Tính bậc của mỗi đỉnh = tổng hàng trong ma trận.
* Nếu đỉnh nào có bậc lẻ → không tồn tại chu trình Euler.
* Sau đó, kiểm tra tính liên thông (bằng DFS hoặc stack):
  + Tìm 1 đỉnh có bậc > 0 làm điểm bắt đầu.
  + Duyệt toàn bộ đồ thị bằng stack.
  + Nếu còn đỉnh chưa thăm nhưng có bậc > 0 → không liên thông.

2️⃣ Tìm chu trình Euler

* Sử dụng stack để lưu các đỉnh đang đi.
* Ban đầu, push đỉnh xuất phát (thường là 0).
* Trong khi stack chưa rỗng:
  + Lấy đỉnh trên đỉnh stack (u).
  + Tìm đỉnh kề v còn cạnh với u.
  + Nếu tồn tại v:
    - Push v vào stack.
    - Xóa cạnh (u, v) khỏi đồ thị (đánh dấu là 0).
  + Nếu không còn đỉnh kề, pop u khỏi stack và đưa vào kết quả.
* Cuối cùng, chu trình Euler là dãy đỉnh trong stack được pop ra ngược thứ tự.