A. Đồ thị vô hướng

1.

- Đồ thị 1: có Euler

- Đồ thị 2:Không Euler

- Đồ thị 3: có Euler

- Đồ thị 4: Không Euler

-> Không đồ thị nào có chu trình Halminton  
  
2.

Số đồ thị vô hướng khác nhau có V đỉnh và E cạnh là 2^n

n = VC3

3.

Duyệt qua tất cả các cạnh và kiểm tra xem có cạnh nào có điểm chung không. Nếu có, đó là cạnh song song.

4.

- Giả sử có một đồ thị G chứa chu trình độ dài lẻ C và là đồ thị hai màu. Ta CM giả thuyết sai.

- Chia đỉnh của C thành hai tập A và B (mọi cạnh trong C đều nối một đỉnh từ A với một đỉnh từ B).

- C đỉnh lẻ -> có C cạnh lẻ

- Bắt đầu từ một đỉnh v bất kỳ trong C. Đỉnh v thuộc một trong hai tập màu, giả sử vthuộc tập A. Theo quy tắc, tất cả các đỉnh lân cận của v trong C phải thuộc tập màu B => số cạnh nối đỉnh thuộc tập A với tập B là lẻ (do C có số cạnh lẻ).

- Nếu chúng ta tiếp tục đi dọc theo chu trình C, mỗi lần chuyển đỉnh, chúng ta chuyển từ một tập màu sang tập màu khác. Vì số cạnh của C là lẻ, nếu chúng ta chuyển qua một đỉnh mới, chúng ta phải chuyển từ một tập màu sang tập màu khác. Khi đi qua tất cả các đỉnh của C, chúng ta sẽ quay lại đỉnh ban đầu và cần phải chuyển sang tập màu khác -> Mẫu thuẫn

=> Giả thuyết sai => ĐPCM

5.

- TH cơ bản:

+ Nếu đồ thị chỉ có hai đỉnh V = 2 thì nó là đồ thị biconnected vì mọi cặp đỉnh đều được nối với nhau bằng một đường đi.

* TH tổng quát:

+ Giả sử đồ thị có nhiều hơn hai đỉnh V > 2

+Chọn một đỉnh bất kỳ strong đồ thị và một đỉnh t không kề với s

+ Tính P đường đi nối giữa s và t

* Bởi vì không có điểm articulation, việc xóa một đỉnh bất kỳ không làm mất tính liên thông của đồ thị.

+ Điều này có nghĩa là nếu ta xóa một đỉnh bất kỳ trên đường đi P, thì svà t vẫn kết nối bằng một đường đi khác.

+ Do đó, ta có thể xây dựng hai đường đi khác nhau từ s đến tbằng cách lần lượt xóa các đỉnh trên đường đi P

+ Điều này chứng tỏ rằng mọi cặp đỉnhs và t đều có hai đường đi không giao nhau nối chúng.

* Do mọi cặp đỉnh s và t đều có hai đường đi không giao nhau, ta có thể kết luận rằng đồ thị là biconnected.

6.

#include <iostream>

#include <list>

#include <vector>

using namespace std;

class Graph {

int V;

list<int> \*adj;

public:

Graph(int V);

void addEdge(int v, int w);

bool isEdgeConnected();

bool isEdgeConnectedUtil(int v, vector<bool>& visited, vector<int>& disc, vector<int>& low, vector<int>& parent);

};

Graph::Graph(int V) {

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w) {

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

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

}

bool Graph::isEdgeConnectedUtil(int v, vector<bool>& visited, vector<int>& disc, vector<int>& low, vector<int>& parent) {

static int time = 0;

visited[v] = true;

disc[v] = low[v] = ++time;

for (const auto& neighbor : adj[v]) {

if (!visited[neighbor]) {

parent[neighbor] = v;

if (!isEdgeConnectedUtil(neighbor, visited, disc, low, parent))

return false;

low[v] = min(low[v], low[neighbor]);

if (low[neighbor] > disc[v]) {

cout << "Edge " << v << "-" << neighbor << " is a bridge.\n";

return false;

}

} else if (neighbor != parent[v]) {

low[v] = min(low[v], disc[neighbor]);

}

}

return true;

}

bool Graph::isEdgeConnected() {

vector<bool> visited(V, false);

vector<int> disc(V, -1);

vector<int> low(V, -1);

vector<int> parent(V, -1);

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

if (!visited[i]) {

if (!isEdgeConnectedUtil(i, visited, disc, low, parent))

return false;

}

}

return true;

}

7.

#include <iostream>

#include <vector>

using namespace std;

#define ROWS 8

#define COLS 8

bool isValid(int x, int y) {

return (x >= 0 && x < ROWS && y >= 0 && y < COLS);

}

void floodFillDFS(vector<vector<int>>& image, int x, int y, int newColor, int oldColor) {

if (!isValid(x, y) || image[x][y] != oldColor) {

return;

}

image[x][y] = newColor;

floodFillDFS(image, x + 1, y, newColor, oldColor);

floodFillDFS(image, x - 1, y, newColor, oldColor);

floodFillDFS(image, x, y + 1, newColor, oldColor);

floodFillDFS(image, x, y - 1, newColor, oldColor);

}

void floodFill(vector<vector<int>>& image, int x, int y, int newColor) {

int oldColor = image[x][y];

if (oldColor != newColor) {

floodFillDFS(image, x, y, newColor, oldColor);

}

}

void printImage(const vector<vector<int>>& image) {

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

for (int j = 0; j < COLS; ++j) {

cout << image[i][j] << " ";

}

cout << endl;

}

}

Bài 8:

1. Đồ thị có trọng số: Nếu đồ thị có trọng số, thứ tự tô pô không được đảm bảo bởi BFS. BFS chỉ quan tâm đến khoảng cách, không phải trọng số của cạnh.

2. Đồ thị có chu trình âm: BFS không thích ứng tốt với đồ thị có chu trình âm. Trong khi BFS có thể hoạt động với đồ thị có chu trình âm, nó không đảm bảo rằng thứ tự tô pô sẽ được duyệt đúng cách.

3. Nhiều thứ tự tô pô hợp lệ: Đồ thị có thể có nhiều thứ tự tô pô hợp lệ. BFS có thể tạo ra một trong số đó, nhưng không đảm bảo rằng nó sẽ tạo ra thứ tự tô pô cụ thể nào.

Bài 9:

Bài 10:

Bài 12:

function hasUniqueTopologicalOrder(graph):

vertices = graph.getVertices()

inDegree = array of size vertices filled with 0

stack = empty stack

// Tính bậc vào của mỗi đỉnh

for each vertex in graph:

for each neighbor in vertex.neighbors:

inDegree[neighbor] += 1

// Thêm các đỉnh có bậc vào bằng 0 vào stack

for i from 0 to vertices - 1:

if inDegree[i] == 0:

stack.push(i)

count = 0

while stack is not empty:

u = stack.pop()

// Kiểm tra số lượng đỉnh đã được xử lý

count += 1

if count > 1:

return false // Nếu có nhiều hơn một thứ tự tô pô, trả về false

for each neighbor in u.neighbors:

inDegree[neighbor] -= 1

if inDegree[neighbor] == 0:

stack.push(neighbor)

return count == vertices // Kiểm tra xem tất cả đỉnh đã được xử lý hay không

Bài 13:

Đồ thị có hướng giữa V đỉnh có thể được biểu diễn bằng một ma trận kề V \* V, trong đó phần tử Aij bằng 1 nếu có cạnh từ i đến j, và 0 nếu không có cạnh. Với mỗi phần tử trong ma trận, chúng ta có 2 lựa chọn: có cạnh hoặc không có cạnh, do đó tổng số đồ thị có hướng là 2^V^2.

Bài 14:

*Bn*+1​=∑*k*=0*n*​(*kn*​)*Bk*​

Bài

Bài 19:

1. Xây dựng cây bao trùm min (Minimum Spanning Tree - MST) ban đầu:

- Sử dụng thuật toán Prim hoặc Kruskal để xây dựng cây bao trùm min T

- Lưu trữ trọng số của mỗi đỉnh trong cây.

2. Khởi tạo danh sách W chứa trọng số của mỗi đỉnh:

- Với mỗi đỉnh v, W[v] chứa trọng số của đỉnh đó trong cây bao trùm min.

3. Lặp qua từng cạnh trong T

- Tính trọng số của cạnh hiện tại e là trọng số lớn nhất trên đường từ e đến gốc của cây.

- Nếu e có trọng số bằng W[u], thì đánh dấu e là cạnh khó.

- Ngược lại, nếu e có trọng số lớn hơn W[u], thì cập nhật W[u] = e

4. Thuật toán trên sẽ tìm tất cả các cạnh khó

- Điều này dựa trên việc rằng mỗi cạnh chỉ thay đổi trọng số một số hữu hạn lần trong quá trình xây dựng cây bao trùm min.

Bài 21:

Input: Đồ thị có trọng số G, tập cạnh S không chứa chu trình.

Output: Cây bao trùm min của G chứa tất cả các cạnh trong S.

1. Khởi tạo một cây bao trùm min T rỗng.

2. Khởi tạo một hàng đợi ưu tiên (min heap) Q với các đỉnh của G, sắp xếp theo trọng số.

3. Đưa tất cả các cạnh trong tập S vào T.

4. Đối với mỗi đỉnh u thuộc tập S, đánh dấu u đã được thăm.

5. Lặp lại cho đến khi T trở thành cây bao trùm min:

a. Lấy cạnh (u, v) có trọng số nhỏ nhất từ Q, với u đã được thăm và v chưa được thăm.

b. Nếu không có cạnh nào thỏa mãn điều kiện trên, dừng thuật toán.

c. Thêm cạnh (u, v) vào T.

d. Đánh dấu v đã được thăm.

6. Trả về cây bao trùm min T.

Bài 26:

Sử dụng khái niệm về đồ thị phụ:

Xét một đồ thị có hướng G với các trọng số không âm tại các đỉnh. Chúng ta sẽ xây dựng một đồ thị phụ G’ sao cho mỗi cạnh (u, v) trong G có thể được biểu diễn bằng một chuỗi các cạnh trong G’ với trọng số cộng dồn.

Đồ thị phụ G’sẽ có cùng tập đỉnh với G nhưng với mỗi cạnh (u, v)trong G có trọng số w, thêm w cạnh từ u đến v trong G' với trọng số 1

Nếu có một cạnh(u, v) trong G với trọng số , thì trong G' , chúng ta sẽ thêm 3 cạnh từ u đến v với trọng số 1 . Như vậy, đường đi từ u đến v trong G với trọng số sẽ được biểu diễn bằng một đường đi có 3 cạnh từ u đến vtrong G' với trọng số 1

Do G' chỉ chứa cạnh có trọng số 1, nên tất cả các đường đi ngắn nhất trong G' đều có trọng số là số lượng cạnh trên đường đi đó. Điều này chính là tổng các trọng số của các đỉnh trên đường đi tương ứng trong G

* ĐPCM