Họ và tên: Tạ Nguyễn Long

MSSV: 20224442

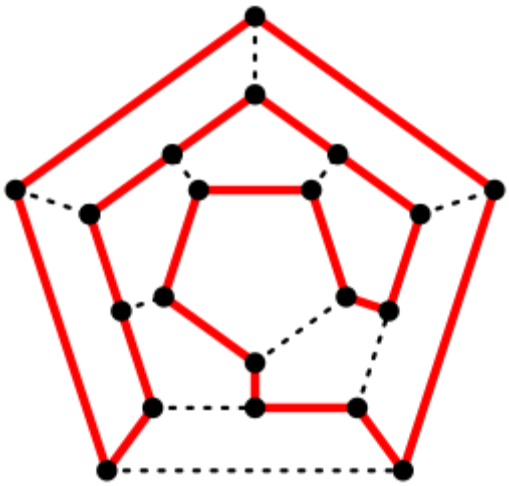
BÁO CÁO BÀI TẬP LỚN

PRINT ALL HAMILTONIAN PATHS PRESENT IN A GRAPH

1. Giới thiệu

1.1. Giới thiệu thuật toán

Hamiltonian Path là một đường đi trong đồ thị vô hướng hoặc có hướng, thăm mỗi đỉnh đúng một lần. Bài toán Hamiltonian Path là một trong những bài toán nổi tiếng trong lý thuyết đồ thị, liên quan đến việc tìm kiếm tất cả các đường đi Hamiltonian của một đồ thị. Bài toán Hamiltonian Path có thể được mở rộng thành bài toán **Hamiltonian Cycle**, trong đó yêu cầu không chỉ đi qua tất cả các đỉnh mà còn quay trở lại điểm xuất phát (tạo thành một chu trình khép kín).



Hình 1: Ví dụ về Hamiltonian Path (đường màu đỏ)

1.2. Đặc điểm chính

- Mỗi đỉnh trong đồ thị được thăm đúng một lần.
- Một Hamiltonian Path không bắt buộc phải quay về đỉnh ban đầu (khác với Hamiltonian Cycle).
- Đồ thị cần phải kết nối để đảm bảo tồn tại ít nhất một đường đi giữa các đỉnh.
- Đối với đồ thị có hướng, cần kiểm tra hướng của cạnh khi tìm đường đi.

2. Động lực (Motivation)

2.1. Phát biểu Bài toán

Cho đồ thị vô hướng G = (V, E), trong đó:

- **V** là tập hợp các đỉnh.
- **E** là tập hợp các cạnh nối giữa các đỉnh.

Bài toán yêu cầu tìm một dãy các đỉnh: **v1, v2, ..., vn**, sao cho:

1. Mỗi đỉnh trong V xuất hiện đúng một lần trong dãy.
2. Mỗi cặp đỉnh liên tiếp (vi, vi+1) (với 1 ≤ i < n) đều có một cạnh nối giữa chúng trong đồ thị G.

2.2. Ứng dụng Thực tế

- **Vận tải và Logistics**: Tìm các tuyến đường đi hiệu quả để vận chuyển hàng hóa.
- **Tối ưu hóa Mạng**: Tối ưu hóa mạng lưới máy tính hoặc mạng lưới điện.
- **Sinh học Tính toán**: Phân tích chuỗi DNA hoặc RNA.

3. Bối cảnh & Công việc Liên quan

Bài toán Hamiltonian Path thuộc lớp bài toán NP-complete, liên quan mật thiết đến bài toán Hamiltonian Cycle, Traveling Salesman Problem (TSP). Các kỹ thuật như **backtracking**, **lập trình động** thường được áp dụng để giải quyết.

4. Giao diện (Giả định và Dữ liệu Đầu vào)

- **Giả thiết**:
    - Đồ thị có thể là vô hướng hoặc có hướng.
    - Đồ thị được biểu diễn dưới dạng danh sách kề hoặc ma trận kề.
    - Đảm bảo đồ thị kết nối (nếu không, không tồn tại đường đi Hamiltonian).
- **Đầu vào**:
    - Một đồ thị với n đỉnh (đánh số từ 0 đến n-1).
    - Danh sách các cạnh nối giữa các đỉnh.
- **Đầu ra**:
    - Tất cả các đường đi Hamiltonian khả dĩ, hoặc thông báo không tồn tại nếu không tìm được đường đi.

5. Phân tích giải thuật

5.1. Ý tưởng

Ý tưởng là sử dụng backtracking. Ta kiểm tra xem mọi cạnh bắt đầu từ một đỉnh chưa được thăm có dẫn đến một giải pháp hay không. Vì một đường đi Hamiltonian thăm mỗi đỉnh đúng một lần, ta sử dụng sự trợ giúp của mảng visited[] trong giải pháp được đề xuất để chỉ xử lý các đỉnh chưa được thăm. Ngoài ra, ta sử dụng mảng path[] để lưu trữ các đỉnh được bao phủ trong đường dẫn hiện tại. Nếu tất cả các đỉnh đã được thăm, thì một đường đi Hamiltonian tồn tại trong đồ thị và in toàn bộ đường đi được lưu trữ trong mảng path[].

Cụ thể như sau:

***Khởi tạo**:*

- Chọn một đỉnh bắt đầu (ví dụ đỉnh đầu tiên).
- Đánh dấu đỉnh đó là đã thăm trong mảng visited[] và thêm vào mảng path[].

***Kiểm tra các đỉnh kề**:*

- Từ đỉnh hiện tại, duyệt qua các đỉnh kề (theo danh sách kề của đồ thị).

- Đối với mỗi đỉnh kề chưa được thăm, ta kiểm tra xem có thể tiếp tục thêm nó vào đường đi hay không (tức là nếu nó chưa được thăm trong mảng visited[]).

***Thêm đỉnh vào đường đi**:*

- Nếu một đỉnh kề chưa được thăm, đánh dấu nó là đã thăm trong mảng visited[] và thêm vào mảng path[].
- Tiến hành đệ quy kiểm tra tiếp các đỉnh kề tiếp theo từ đỉnh mới.

***Kiểm tra điều kiện dừng**:*

- Nếu tất cả các đỉnh đã được thăm (tức là kích thước của path[] bằng số lượng đỉnh trong đồ thị), thì in ra đường đi Hamiltonian vì ta đã đi qua tất cả các đỉnh mà không bị lặp lại.

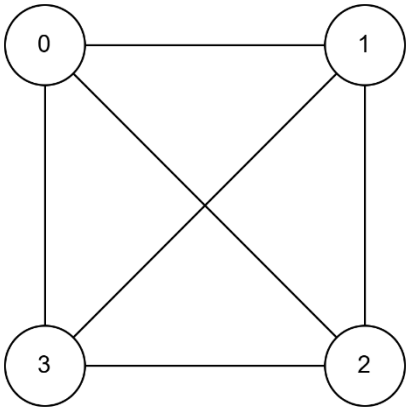
***Quay lại (Backtrack)**:*

- Nếu không thể tiếp tục với một đỉnh kề, ta quay lại (backtrack) để thử các lựa chọn khác.
- Đánh dấu lại đỉnh hiện tại là chưa thăm trong mảng visited[] và loại bỏ đỉnh đó khỏi path[].

5.2. Ví dụ minh hoạ

Giả sử ta có đồ thị với 4 đỉnh (0, 1, 2, 3) và các cạnh sau:

(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)



Hình 2: Đồ thị ví dụ

Mô tả quá trình thực hiện

Vị trí trong path[]	Đỉnh hiện tại	Đỉnh kề thử	Mảng visited[]	Mảng path[]	Quyết định
0	0	-	[True, False, False, False]	[0]	Bắt đầu tại đỉnh 0.
1	0	1	[True, True, False, False]	[0, 1]	Thử đỉnh 1 từ 0.
2	1	2	[True, True, True, False]	[0, 1, 2]	Thử đỉnh 2 từ 1.

3	2	3	[True, True, True, True]	[0, 1, 2, 3]	Đã tìm được đường đi Hamiltonian: [0, 1, 2, 3].
3 (quay lại)	2	3	[True, True, True, False]	[0, 1, 2]	Không có giải pháp mới, quay lại đỉnh 2.
2 (quay lại)	1	3	[True, True, False, False]	[0, 1]	Quay lại đỉnh 1 và thử đỉnh 3.
3	1	3	[True, True, False, True]	[0, 1, 3]	Thử đỉnh 3 từ 1.
2	3	2	[True, True, True, True]	[0, 1, 3, 2]	Đã tìm được đường đi Hamiltonian: [0, 1, 3, 2].

Lần lượt như vậy, ta sẽ tìm được tất cả các đường đi Hamiltonian

0 1 2 3
0 1 3 2
0 2 1 3
0 2 3 1
0 3 1 2
0 3 2 1
1 0 2 3
1 0 3 2
1 2 0 3
1 2 3 0
1 3 0 2
1 3 2 0
2 0 1 3
2 0 3 1
2 1 0 3
2 1 3 0
2 3 0 1
2 3 1 0
3 0 1 2
3 0 2 1
3 1 0 2
3 1 2 0
3 2 0 1
3 2 1 0

6. Minh hoạ thuật toán bằng C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

// Data structure to store a graph edge
struct Edge {
    int src, dest;
};

// A class to represent a graph object
class Graph
{
public:
    // a vector of vectors to represent an adjacency list
    vector<vector<int>> adjList;

    // Constructor
    Graph(vector<Edge> const &edges, int n)
    {
        // resize the vector to hold `n` elements of type `vector<int>`
        adjList.resize(n);

        // add edges to the undirected graph
        for (Edge edge: edges)
        {
            int src = edge.src;
            int dest = edge.dest;

            adjList[src].push_back(dest);
            adjList[dest].push_back(src);
        }
    }
};

// Utility function to print a path
void printPath(vector<int> const &path)
{
    for (int i: path) {
        cout << i << ' ';
    }
    cout << endl;
}

```

```cpp
void hamiltonianPaths(Graph const &graph, int v, vector<bool> &visited,
                      vector<int> &path, int n)
{
    // if all the vertices are visited, then the Hamiltonian path exists
    if (path.size() == n)
    {
        // print the Hamiltonian path
        printPath(path);
        return;
    }

    // Check if every edge starting from vertex `v` leads to a solution or not
    for (int w: graph.adjList[v])
    {
        // process only unvisited vertices as the Hamiltonian
        // path visit each vertex exactly once
        if (!visited[w])
        {
            visited[w] = true;
            path.push_back(w);

            // check if adding vertex `w` to the path leads to the solution or not
            hamiltonianPaths(graph, w, visited, path, n);

            // backtrack
            visited[w] = false;
            path.pop_back();
        }
    }
}

void findHamiltonianPaths(Graph const &graph, int n)
{
    // start with every node
    for (int start = 0; start < n; start++)
    {
        // add starting node to the path
        vector<int> path;
        path.push_back(start);

        // mark the start node as visited
        vector<bool> visited(n);
        visited[start] = true;

        hamiltonianPaths(graph, start, visited, path, n);
    }
}

int main()
{
    // consider a complete graph having 4 vertices
    vector<Edge> edges = {
        {0, 1}, {0, 2}, {0, 3}, {1, 2}, {1, 3}, {2, 3}
    };

    // total number of nodes in the graph (labelled from 0 to 3)
    int n = 4;

    // build a graph from the given edges
    Graph graph(edges, n);

    findHamiltonianPaths(graph, n);

    return 0;
}
```

7. Phân tích

7.1. Độ phức tạp về thời gian

Giả sử đồ thị có n đỉnh và e cạnh. Độ phức tạp của thuật toán chủ yếu bị ảnh hưởng bởi số lượng các đường đi Hamiltonian có thể có trong đồ thị. Cụ thể:

Tối đa số đường đi Hamiltonian: Đối với một đồ thị đầy đủ (complete graph) với n đỉnh, số lượng đường đi Hamiltonian là n! (tất cả các hoán vị của các đỉnh). Mỗi hoán vị là một đường đi mà không thăm lại đỉnh nào.

Độ phức tạp đệ quy: Tại mỗi đỉnh, thuật toán sẽ thử tất cả các đỉnh kề chưa thăm. Do đó, tại mỗi bước đệ quy, có thể có n-1, n-2, ..., 1 lựa chọn, tùy thuộc vào số lượng đỉnh đã được thăm. Điều này dẫn đến một độ phức tạp thời gian là O(n!) trong trường hợp xấu nhất (khi đồ thị là đồ thị đầy đủ).

7.2. Độ phức tạp về không gian

- Mảng visited[]: Chứa thông tin về việc mỗi đỉnh đã được thăm hay chưa, có kích thước n.
- Mảng path[]: Lưu trữ đường đi hiện tại, có kích thước tối đa là n (vì mỗi đường đi Hamiltonian có thể chứa tất cả các đỉnh).

=> Độ phức tạp không gian: O(n) để lưu trữ mảng visited[] và path[].

7.3. Độ phức tạp trong trường hợp cụ thể

Trong trường hợp đồ thị có nhiều đỉnh và cạnh nhưng không phải là đồ thị đầy đủ (tức là có ít hơn n! đường đi Hamiltonian), số lượng phép toán có thể ít hơn rất nhiều, phụ thuộc vào cấu trúc của đồ thị. Tuy nhiên, trong trường hợp xấu nhất (đồ thị đầy đủ), độ phức tạp thời gian vẫn là O(n!).

7.4. Tổng kết

- Độ phức tạp thời gian: O(n!) trong trường hợp xấu nhất (đồ thị đầy đủ).
- Độ phức tạp không gian: O(n), vì cần lưu trữ thông tin về các đỉnh đã thăm và đường đi hiện tại.

8. Kết luận

Thuật toán Hamiltonian Path là một ví dụ minh họa cho sức mạnh và thách thức của các bài toán trong lý thuyết đồ thị. Mặc dù khó giải quyết đối với các trường hợp lớn, nhưng nó vẫn mang lại giá trị thực tiễn to lớn trong nhiều lĩnh vực. Với các cải tiến thuật toán và tài nguyên tính toán ngày càng tăng, việc giải quyết bài toán này sẽ trở nên khả thi hơn trong tương lai.