

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**  
**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

=====\*\*\*=====



**BÁO CÁO BÀI TẬP LỚN THUỘC HỌC PHẦN:**  
**TRÍ TUỆ NHÂN TẠO**

**ÁP DỤNG THUẬT TOÁN BFS TRONG BÀI TOÁN TÌM ĐƯỜNG ĐI**  
**GIỮA 2 TỈNH CỦA VIỆT NAM**

**GVHD: Mai Thanh Hồng**

**Lớp: 20251IT6094002**

**Nhóm: 03**

**Thành viên: Nguyễn Trọng Cường - 2023602667**

**Đoàn Huy Hoàng - 2023603112**

**Phạm Ngọc Thạch - 2023601930**

**Lý Ngọc Quyền - 2023603244**

Hà Nội, Năm 2025

## LỜI CẢM ƠN

Trong suốt quá trình nghiên cứu và thực hiện đề tài “Áp dụng thuật toán BFS trong bài toán tìm đường đi giữa 2 tỉnh của Việt Nam”, chúng em đã nhận được sự quan tâm, chỉ bảo và hỗ trợ quý báu từ các thầy cô và nhà trường.

Trước hết, chúng em xin bày tỏ lòng biết ơn sâu sắc đến cô Mai Thanh Hồng, người trực tiếp hướng dẫn, đã tận tình chỉ bảo, định hướng phương pháp nghiên cứu và luôn theo sát, động viên chúng em trong từng giai đoạn thực hiện báo cáo. Nhờ sự hỗ trợ và kiến thức chuyên môn của cô, chúng em đã có thể định hình rõ ràng vấn đề nghiên cứu, lựa chọn cách tiếp cận phù hợp và hoàn thành đề tài này.

Chúng em cũng xin gửi lời cảm ơn chân thành đến tập thể các thầy cô Trường Công Nghệ Thông tin và Truyền Thông – Trường Đại Học Công Nghiệp Hà Nội, những người đã giảng dạy và truyền đạt cho chúng em nền tảng kiến thức vững chắc từ các môn học cơ sở đến chuyên ngành. Đây chính là hành trang quan trọng giúp chúng em có đủ cơ sở để áp dụng vào quá trình nghiên cứu và triển khai đề tài.

Bên cạnh đó, chúng em cũng xin cảm ơn các bạn bè và gia đình, những người đã luôn ủng hộ, động viên, chia sẻ và tạo điều kiện tốt nhất để chúng em yên tâm học tập và hoàn thành nhiệm vụ được giao.

Mặc dù đã rất nỗ lực, song do giới hạn về thời gian cũng như kinh nghiệm nghiên cứu còn hạn chế, báo cáo chắc chắn không tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự đóng góp ý kiến, nhận xét và chỉ dẫn quý báu từ các thầy cô để chúng em có thể hoàn thiện hơn trong các nghiên cứu sau này, cũng như tích lũy thêm kinh nghiệm phục vụ cho công việc và học tập trong tương lai.

Một lần nữa, chúng em xin chân thành cảm ơn!

## MỤC LỤC

<b>LỜI CẢM ƠN.....</b>	<b>1</b>
<b>MỤC LỤC.....</b>	<b>3</b>
<b>DANH MỤC HÌNH ẢNH.....</b>	<b>5</b>
<b>DANH MỤC BẢNG BIỂU .....</b>	<b>6</b>
<b>Chương I. TỔNG QUAN BÀI TOÁN.....</b>	<b>7</b>
1.1. Mục tiêu và phạm vi nghiên cứu .....	7
1.1.1. Mục tiêu .....	7
1.1.2. Phạm vi nghiên cứu .....	8
1.2. Mô hình hoá bài toán .....	9
1.2.1. Dữ liệu đầu vào.....	9
1.2.2. Dữ liệu đầu ra .....	9
1.2.3. Thuật toán giải quyết .....	9
1.2.4. Ràng buộc .....	9
<b>Chương II. CƠ SỞ LÝ THUYẾT VÀ THUẬT TOÁN BFS .....</b>	<b>11</b>
2.1. Cơ sở lý thuyết.....	11
2.1.1. Đồ thị .....	11
2.1.2. Thuật toán BFS .....	12
2.1.3. Ứng dụng BFS trong tìm đường đi.....	13
2.2. Thuật toán BFS .....	14
2.2.1. Ý tưởng thuật toán .....	14
2.2.2. Lưu đồ của thuật toán .....	16
2.2.3. Mã giả của thuật toán.....	17
2.3. Ứng dụng BFS vào bài toán tìm đường đi.....	19

2.3.1. Mô hình hóa bài toán .....	19
2.3.2. Cách hoạt động của thuật toán BFS trong bài toán tìm đường đi .....	19
2.3.3. Ưu điểm và nhược điểm .....	20
<b>Chương III. THIẾT KẾ VÀ CÀI ĐẶT .....</b>	<b>22</b>
3.1. Cài đặt môi trường, ngôn ngữ lập trình .....	22
3.2. Thiết kế cấu trúc dữ liệu và biểu diễn đồ thị .....	23
3.2.1. Mô hình dữ liệu đồ thị .....	23
3.2.2. Thu thập và chuẩn hóa dữ liệu đồ thị .....	23
3.2.3. Các cấu trúc dữ liệu .....	24
3.3. Cài đặt thuật toán .....	25
3.3.1. Mục tiêu triển khai.....	25
3.3.2. Biểu diễn đồ thị bằng danh sách kề .....	25
3.3.3. Cài đặt thuật toán BFS tìm đường đi.....	30
3.3.4. Chương trình chính.....	32
3.3.5. Đánh giá phương pháp cài đặt.....	33
<b>Chương IV. KIỂM THỬ .....</b>	<b>35</b>
4.1. Test case.....	35
4.2. Kết quả chạy thử .....	36
4.3. Đánh giá .....	36
<b>Chương V: HƯỚNG PHÁT TRIỂN .....</b>	<b>38</b>
<b>KẾT LUẬN.....</b>	<b>40</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>41</b>

## DANH MỤC HÌNH ẢNH

<i>Hình 1: Cách thăm các đỉnh của thuật toán BFS.</i> .....	14
<i>Hình 2: Lưu đồ thuật toán BFS.</i> .....	16
<i>Hình 3: Mẫu request khi test chương trình</i> .....	30

## **DANH MỤC BẢNG BIỂU**

Bảng 1: Các kịch bản kiểm thử chương trình. ....	35
Bảng 2: Kết quả kiểm thử các kịch bản. ....	36

# Chương I. TỔNG QUAN BÀI TOÁN

## 1.1. Mục tiêu và phạm vi nghiên cứu

### 1.1.1. Mục tiêu

Mục tiêu của đề tài là nghiên cứu và áp dụng thuật toán BFS (Breadth-First Search) để giải quyết bài toán tìm đường đi giữa hai tỉnh bất kỳ trên bản đồ Việt Nam trong điều kiện mô hình hóa đơn giản. Thông qua việc tìm hiểu nguyên lý hoạt động và cách triển khai BFS, đề tài hướng tới việc khẳng định tính hiệu quả của thuật toán này trong đồ thị vô hướng, không trọng số, nơi mà mỗi tuyến đường liên tỉnh được coi là một cạnh có trọng số đồng nhất.

Bên cạnh đó, đề tài cũng nhằm cung cấp một cách tiếp cận trực quan và dễ hiểu trong việc biểu diễn các tỉnh thành và tuyến đường liên tỉnh dưới dạng đồ thị, giúp người đọc dễ dàng hình dung mối quan hệ kết nối giữa các khu vực. Trên cơ sở đó, một chương trình mô phỏng đơn giản được xây dựng, cho phép người dùng nhập vào điểm xuất phát và điểm đích, sau đó trả về đường đi được tìm thấy bởi thuật toán BFS. Chương trình không chỉ giúp minh họa ứng dụng thực tiễn của BFS trong việc tìm kiếm đường đi mà còn có ý nghĩa như một công cụ trực quan hỗ trợ giảng dạy và học tập thuật toán đồ thị.

Ngoài ra, mục tiêu xa hơn của nghiên cứu là đặt nền móng cho các hướng phát triển tiếp theo như mở rộng phạm vi dữ liệu, bổ sung các yếu tố thực tế về khoảng cách địa lý hay thời gian di chuyển, từ đó tiến tới xây dựng một hệ thống tìm đường hoàn chỉnh và tối ưu hơn.

### 1.1.2. Phạm vi nghiên cứu

Đề tài được triển khai trong một phạm vi giới hạn nhằm đảm bảo tính khả thi và phù hợp với thời gian nghiên cứu. Trước hết, nghiên cứu chỉ tập trung vào khía cạnh lý thuyết và mô phỏng cơ bản, chưa xét đến các yếu tố thực tế như khoảng cách địa lý chính xác, tốc độ di chuyển, chi phí, hay tình trạng giao thông. Mỗi tuyến đường liên tỉnh trong mô hình đều được giả định có trọng số bằng nhau, từ đó biến bài toán thành một đồ thị vô hướng, không trọng số, phù hợp với đặc điểm hoạt động của thuật toán BFS.

Dữ liệu sử dụng trong đề tài là dữ liệu mô phỏng, bao quát 34 tỉnh, thành phố của Việt Nam, trong đó mỗi tỉnh được biểu diễn bằng một đỉnh và mỗi tuyến đường liên tỉnh trực tiếp được biểu diễn bằng một cạnh của đồ thị.

Thuật toán duy nhất được áp dụng trong nghiên cứu là BFS, bởi nó đặc biệt phù hợp với yêu cầu tìm đường đi trong đồ thị vô hướng, không trọng số.

Về mặt chương trình mô phỏng, nghiên cứu chỉ xây dựng một hệ thống đơn giản, chủ yếu thể hiện việc nhập điểm xuất phát, điểm đích và xuất ra đường đi dưới dạng danh sách các tỉnh đi qua. Giao diện của chương trình có thể ở mức cơ bản (ví dụ như nhập xuất dữ liệu bằng console), không đặt nặng yêu cầu trực quan hóa bản đồ hay tối ưu trải nghiệm người dùng. Tuy nhiên, từ nền tảng cơ bản này, đề tài cũng mở ra khả năng mở rộng trong tương lai, bao gồm việc tích hợp thêm dữ liệu bản đồ chi tiết, bổ sung trọng số dựa trên khoảng cách hoặc thời gian, và áp dụng các thuật toán tìm đường nâng cao để giải quyết những yêu cầu thực tế phức tạp hơn.

## **1.2. Mô hình hoá bài toán**

### **1.2.1. Dữ liệu đầu vào**

- Đồ thị mô phỏng bản đồ 34 tỉnh thành Việt Nam, trong đó:
  - + Mỗi tỉnh/thành phố được biểu diễn bằng một đỉnh.
  - + Nếu hai tỉnh có chung đường biên giới thì giữa hai đỉnh tương ứng tồn tại một cạnh.
- Tỉnh thành xuất phát, tỉnh thành đích.

### **1.2.2. Dữ liệu đầu ra**

- Đường đi từ tỉnh xuất phát đến tỉnh đích dưới dạng danh sách liệt kê các tỉnh thành từ điểm xuất phát đến điểm đích.
- Ngoài ra, có thể kèm theo số bước di chuyển (số cạnh) để tới đích.

### **1.2.3. Thuật toán giải quyết**

- Sử dụng thuật toán BFS (Breadth-First Search), với các đặc điểm:
  - + Duyệt đồ thị theo từng lớp (tầng) từ đỉnh nguồn.
  - + Bảo đảm tìm được đường đi trong đồ thị vô hướng, không trọng số.
  - + Dữ liệu hỗ trợ:
    - Một hàng đợi để quản lý các đỉnh cần duyệt theo thứ tự
    - Một mảng visited để đánh dấu các đỉnh đã được thăm.
    - Một mảng predecessor để lưu vết đường đi, hỗ trợ việc truy xuất lại đường đi.

### **1.2.4. Ràng buộc**

- Bài toán giả định đồ thị là vô hướng và không trọng số (mỗi tuyến đường giữa hai tỉnh được coi như có chi phí bằng nhau).

- Không xét đến các yếu tố thực tế như khoảng cách địa lý, thời gian di chuyển, điều kiện giao thông, địa hình.
- Mô hình chỉ mang tính mô phỏng để minh họa cách áp dụng thuật toán BFS trong tìm đường đi.

## Chương II. CƠ SỞ LÝ THUYẾT VÀ THUẬT TOÁN BFS

### 2.1. Cơ sở lý thuyết

#### 2.1.1. Đồ thị

**Định nghĩa:** Đồ thị là một cấu trúc toán học cơ bản được sử dụng để biểu diễn các mối quan hệ giữa các đối tượng. Cụ thể, một đồ thị bao gồm một tập hợp các phần tử được gọi là đỉnh (hoặc nút) và một tập hợp các cặp đỉnh được kết nối với nhau bởi các cạnh, thể hiện mối liên hệ hoặc sự tương tác giữa các đỉnh đó. Đồ thị có thể được áp dụng trong nhiều lĩnh vực như mạng lưới giao thông, mạng xã hội, hoặc hệ thống viễn thông.

**Biểu diễn:** Một đồ thị thường được biểu diễn bằng ký hiệu toán học  $(G = (V, E))$ , trong đó:

- $(V)$  là tập hợp tất cả các đỉnh (vertices), đại diện cho các thực thể riêng lẻ.
- $(E)$  là tập hợp các cạnh (edges), thể hiện các mối quan hệ trực tiếp giữa các đỉnh.

**Đồ thị trong bài toán:** Trong ngữ cảnh bài toán tìm đường đi giữa các tỉnh của Việt Nam:

- Các đỉnh: Mỗi đỉnh tượng trưng cho một tỉnh hoặc thành phố cụ thể trong nước, ví dụ như Hà Nội, Đà Nẵng, hoặc Cần Thơ.
- Các cạnh: Mỗi cạnh đại diện cho một tuyến đường trực tiếp nối hai tỉnh hoặc thành phố với nhau, chẳng hạn như quốc lộ hoặc đường cao tốc. Trong bài toán này, đồ thị được giả định là vô hướng (undirected), nghĩa là đường đi từ tỉnh A đến tỉnh B có thể đi ngược lại từ B đến A, và không trọng số

(unweighted), tức là không xét đến độ dài hoặc thời gian di chuyển trên từng cung đường, chỉ quan tâm đến số lượng đỉnh trung gian.

### **Một số bài toán liên quan đến đồ thị:**

- Bài toán tìm đường đi: Xác định tuyến đường tối ưu từ một đỉnh nguồn đến các đỉnh khác với chi phí thấp nhất (theo số cạnh hoặc trọng số).
- Bài toán Euler: Tìm một chu trình đi qua tất cả các cạnh của đồ thị đúng một lần, thường áp dụng cho các mạng lưới giao thông hoặc thiết kế mạch điện.
- Bài toán Hamilton: Tìm một chu trình hoặc đường đi đi qua tất cả các đỉnh của đồ thị đúng một lần, hữu ích trong lập kế hoạch hành trình.
- Bài toán tô màu đồ thị: Phân bổ màu sắc cho các đỉnh sao cho các đỉnh kề nhau không cùng màu, thường dùng trong bài toán phân bổ tài nguyên.
- Bài toán tìm cây khung nhỏ nhất: Tìm một tập hợp các cạnh tạo thành cây bao phủ tất cả các đỉnh với tổng trọng số nhỏ nhất, áp dụng trong tối ưu hóa mạng lưới.

### **2.1.2. Thuật toán BFS**

**Khái niệm:** BFS (Breadth-First Search) là một thuật toán duyệt đồ thị theo chiều rộng, thường được sử dụng để khám phá tất cả các đỉnh của đồ thị theo thứ tự từ gần đến xa, bắt đầu từ một đỉnh nguồn đã cho. Thuật toán này hoạt động dựa trên nguyên tắc thăm các đỉnh theo từng "tầng" hoặc mức (level), từ gần nhất đến xa nhất so với đỉnh khởi đầu.

### **Ý tưởng:**

- Thuật toán bắt đầu từ một đỉnh nguồn (source node) được chọn trước.

- Tất cả các đỉnh kề trực tiếp với đỉnh nguồn sẽ được thăm trước, sau đó là các đỉnh kề của các đỉnh đó, và tiếp tục theo từng tầng một cách có hệ thống.
- Quá trình này sử dụng một cấu trúc dữ liệu hàng đợi (queue) để lưu trữ các đỉnh cần thăm, đảm bảo thứ tự duyệt theo chiều rộng.

### **Đặc tính quan trọng:**

- Trong một đồ thị vô hướng và không trọng số, BFS có khả năng đảm bảo tìm ra đường đi từ đỉnh nguồn đến bất kỳ đỉnh nào khác, với độ đo được tính dựa trên số lượng cạnh (số bước) cần đi qua.
- Điều này rất hữu ích trong các bài toán như tìm tuyến đường giữa các địa điểm, nơi chỉ cần quan tâm đến số lượng điểm trung gian mà không xét đến khoảng cách thực tế.

### **2.1.3. Ứng dụng BFS trong tìm đường đi**

**Mục đích:** BFS được áp dụng hiệu quả trong việc tìm tuyến đường giữa hai tỉnh hoặc thành phố, chẳng hạn từ tỉnh A đến tỉnh B trong hệ thống giao thông Việt Nam.

### **Cơ chế triển khai:**

- Thuật toán sử dụng một hàng đợi (queue) để quản lý các đỉnh cần thăm, bắt đầu từ đỉnh nguồn (tỉnh A).
- Mỗi khi thăm một đỉnh, tất cả các đỉnh kề chưa được thăm sẽ được thêm vào hàng đợi, cùng với việc ghi lại đường đi từ đỉnh nguồn.
- Quá trình tiếp tục cho đến khi đến được đỉnh đích (tỉnh B) hoặc duyệt hết tất cả các đỉnh khả thi.

### **Kết quả:**

- Đường đi: BFS có thể tái tạo toàn bộ lộ trình từ tỉnh A đến tỉnh B, ví dụ:  $A \rightarrow C \rightarrow D \rightarrow B$ .
- Số bước: Thuật toán trả về số cạnh trên đường đi, tương ứng với số tỉnh trung gian cộng thêm 1 (từ A đến B).
- Ưu điểm của phương pháp này là tính đơn giản và hiệu quả trong các đồ thị không trọng số, đồng thời dễ dàng mở rộng để xử lý các yêu cầu phức tạp hơn như ghi lại toàn bộ đường đi.

## 2.2. Thuật toán BFS

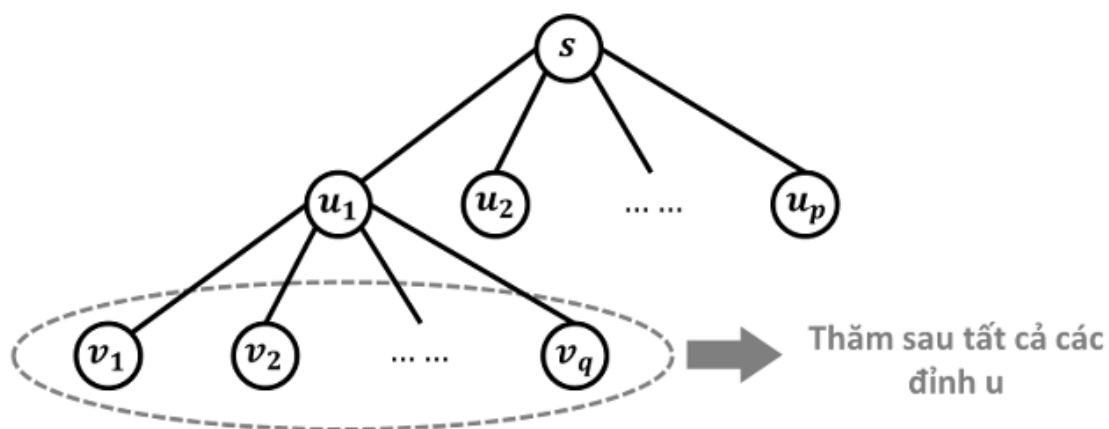
### 2.2.1. Ý tưởng thuật toán

Với đồ thị không trọng số và đỉnh nguồn. Đồ thị này có thể là đồ thị có hướng hoặc vô hướng, điều đó không quan trọng đối với thuật toán. Có thể hiểu thuật toán như một ngọn lửa lan rộng trên đồ thị:

- Ở bước thứ nhất, chỉ có đỉnh nguồn đang cháy.
- Ở mỗi bước tiếp theo, ngọn lửa đang cháy ở mỗi đỉnh lại lan sang tất cả các đỉnh kề với nó.

Trong mỗi lần lặp của thuật toán, "vòng lửa" lại lan rộng ra theo chiều rộng. Những đỉnh nào gần hơn sẽ bùng cháy trước. Chính xác hơn, thuật toán có thể được mô tả như sau:

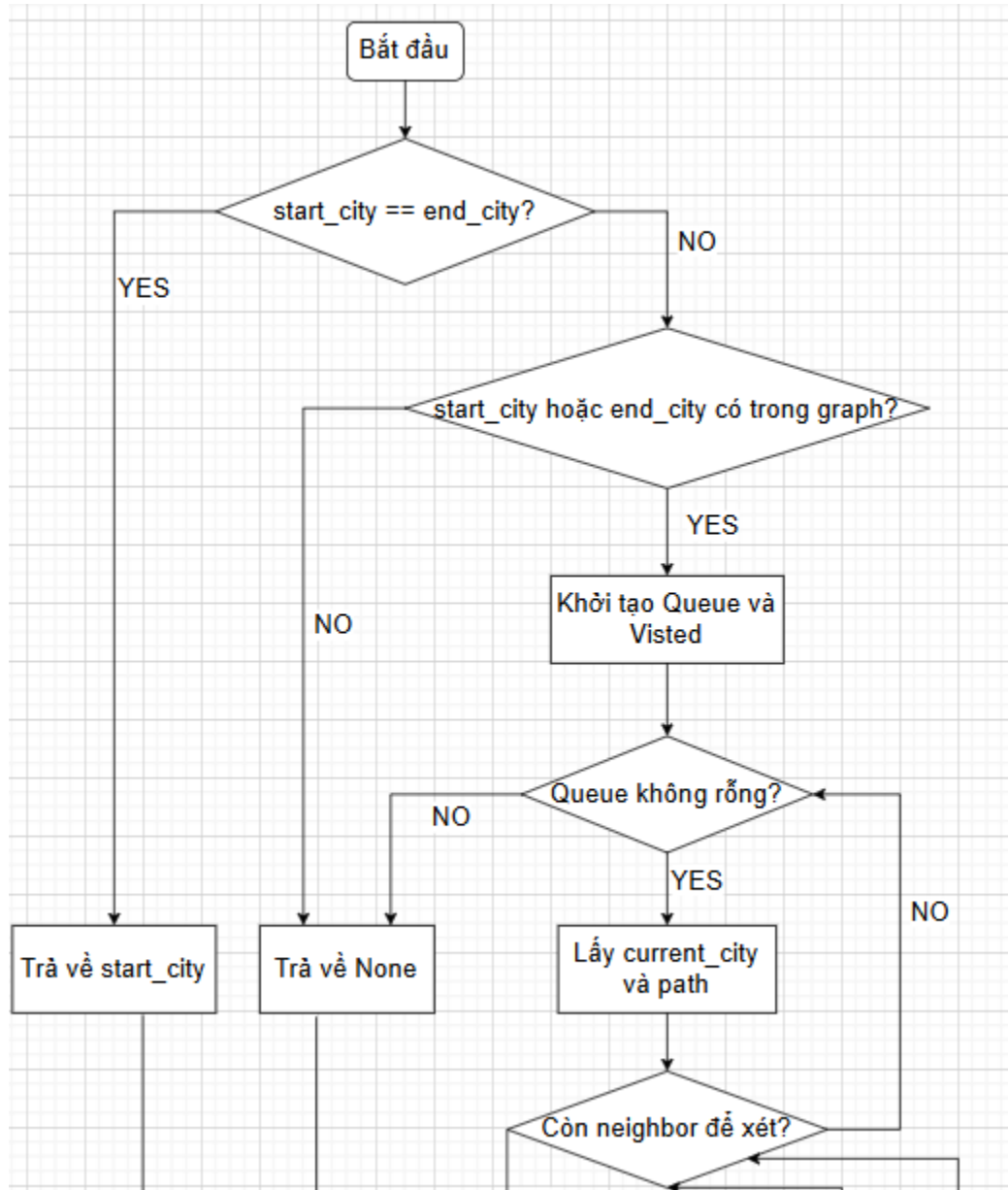
- Đầu tiên ta thăm đỉnh nguồn.
- Việc thăm đỉnh sẽ phát sinh thứ tự thăm các đỉnh kề với (những đỉnh gần nhất). Tiếp theo, ta thăm đỉnh, khi thăm đỉnh sẽ lại phát sinh yêu cầu thăm những đỉnh kề với. Nhưng rõ ràng những đỉnh này “xa” hơn những đỉnh nên chúng chỉ được thăm khi tất cả những đỉnh đều đã được thăm.

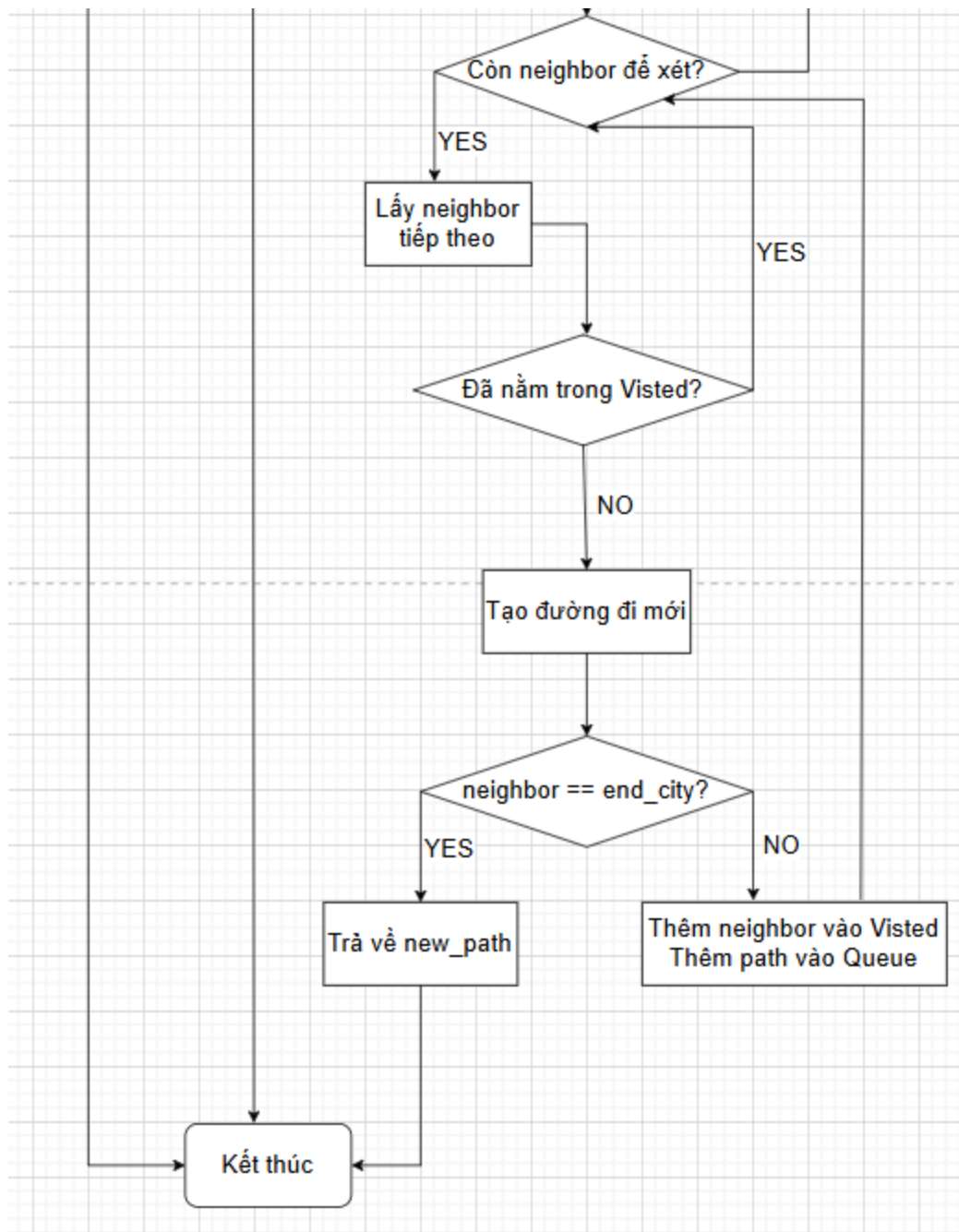


Hình 1: Cách thăm các đỉnh của thuật toán BFS.

Thuật toán tìm kiếm theo chiều rộng sử dụng một danh sách để chứa những đỉnh đang “chờ” thăm. Tại mỗi bước, ta thăm một đỉnh đầu danh sách, loại nó ra khỏi danh sách và cho những đỉnh kề với nó chưa được thăm xếp hàng vào cuối danh sách. Thuật toán sẽ kết thúc khi danh sách rỗng.

### 2.2.2. Lưu đồ của thuật toán





Hình 2: Lưu đồ thuật toán BFS.

### 2.2.3. Mã giả của thuật toán

HÀM BFS (start\_city, end\_city, graph):

# 1. KIỂM TRA ĐIỀU KIỆN DỪNG SỚM

Nếu start\_city bằng end\_city:

Trả về danh sách [start\_city]

Nếu start\_city không có trong graph hoặc end\_city không có trong

graph:

Trả về None

## # 2. KHỞI TẠO

Tạo hàng đợi (Queue) chứa một phần tử là danh sách: [[start\_city]]

Tạo tập hợp (Set) visited chứa: {start\_city}

## # 3. VÒNG LẶP CHÍNH

Lặp khi Queue không rỗng:

# Lấy đường đi hiện tại ở đầu hàng đợi

path = Lấy phần tử đầu tiên ra khỏi Queue

current\_city = Phần tử cuối cùng của path

# Duyệt các thành phố lân cận

Lặp qua mỗi neighbor trong danh sách kề của graph[current\_city]:

Nếu neighbor chưa có trong visited:

# Tạo đường đi mới kế thừa từ đường cũ

new\_path = Sao chép path

Thêm neighbor vào new\_path

# Kiểm tra xem đã đến đích chưa

Nếu neighbor bằng end\_city:

Trả về new\_path

# Đánh dấu và thêm vào hàng đợi để xét tiếp

Thêm neighbor vào visited

Thêm new\_path vào cuối Queue

## # 4. KẾT THÚC

Trả về None

## 2.3. Ứng dụng BFS vào bài toán tìm đường đi

### 2.3.1. Mô hình hóa bài toán

Một bài toán tìm đường đi giữa hai vị trí có thể được biểu diễn dưới dạng đồ thị  $G = (V, E)$ , trong đó:

- $V$  là tập các đỉnh, mỗi đỉnh tương ứng với một địa điểm (tỉnh/thành phố, nút giao thông...).
- $E$  là tập các cạnh, mỗi cạnh biểu diễn mối liên kết trực tiếp hoặc khả năng di chuyển giữa hai đỉnh.

Đồ thị được xem là đồ thị vô hướng nếu việc di chuyển giữa hai đỉnh  $A$  và  $B$  là hai chiều; điều này phù hợp với hầu hết các hệ thống đường bộ thông thường. Ngoài ra, bài toán trong phạm vi này không xét đến trọng số, độ dài hay thời gian di chuyển, nên cạnh được xem là không trọng số.

Khi đó, bài toán tìm đường đi giữa hai vị trí trở thành bài toán tìm một dãy các đỉnh liên tiếp nối từ đỉnh xuất phát  $s$  đến đỉnh đích  $g$  sao cho:

- Tồn tại cạnh giữa hai đỉnh liên tiếp.
- Số lượng cạnh trong đường đi là nhỏ nhất.

### 2.3.2. Cách hoạt động của thuật toán BFS trong bài toán tìm đường đi

Khi áp dụng vào bài toán tìm đường, BFS tiến hành duyệt đồ thị theo từng tầng. Cách hoạt động của BFS có thể được mô tả như sau:

- Khởi tạo tập visited để đánh dấu các đỉnh đã được duyệt và một hàng đợi chứa đỉnh bắt đầu s.
- Duyệt theo từng lớp: BFS lấy đỉnh hiện tại ra khỏi hàng đợi, sau đó duyệt toàn bộ các đỉnh kề chưa được thăm.
- Mở rộng không gian tìm kiếm theo mức độ: Tất cả các đỉnh cách s đúng một cạnh được duyệt trước, sau đó đến các đỉnh cách s hai cạnh, ... và tiếp tục cho đến khi gặp được đỉnh đích.
- Khi gặp đỉnh g, thuật toán dừng lại và trả về đường đi đã tìm được.

Điểm quan trọng nhất là BFS không đi sâu vào đồ thị ngay lập tức, mà khám phá toàn bộ các khả năng gần nhất trước. Cách tiếp cận này đảm bảo rằng đường đi đầu tiên đến g luôn là đường đi có số cạnh nhỏ nhất.

### 2.3.3. Ưu điểm và nhược điểm

Việc áp dụng BFS vào loại bài toán này mang lại nhiều lợi ích:

- Dễ triển khai, nhờ cấu trúc dữ liệu đơn giản (queue + visited).
- Đảm bảo kết quả đúng và ngắn nhất theo số bước.
- Không cần thông tin bổ sung như trọng số, heuristic hoặc chi phí đường đi.
- Hoạt động ổn định trên dữ liệu đồ thị mô tả mạng lưới giao thông hoặc bản đồ hành chính.

Mặc dù BFS là lựa chọn tốt cho bài toán cơ bản, nhưng phương pháp này cũng có một số giới hạn lý thuyết:

- Không xét chính xác độ dài thực tế của đường đi (km).
- Không phù hợp khi bản đồ có quá nhiều đỉnh (vì queue sẽ phình to).

- Không tính toán được đường đi dựa trên yếu tố như thời gian, chi phí, tốc độ di chuyển....

## **Chương III. THIẾT KẾ VÀ CÀI ĐẶT**

### **3.1. Cài đặt môi trường, ngôn ngữ lập trình**

Quá trình thiết lập môi trường phát triển Python là bước quan trọng đầu tiên trước khi xây dựng bất kỳ dự án lập trình Python nào. Môi trường này bao gồm hai thành phần cốt lõi: Python Interpreter và Môi trường Phát triển Tích hợp (Integrated Development Environment - IDE).

Đầu tiên, cần cài đặt Python Interpreter, bộ công cụ chính cung cấp trình thông dịch, thư viện chuẩn và các tiện ích phục vụ cho việc chạy và quản lý mã nguồn Python. Phiên bản Python 3.12 có thể được tải từ trang chủ Python.org. Sau khi cài đặt, người dùng nên kiểm tra lại biến môi trường để đảm bảo Python được thêm vào Path, giúp hệ thống có thể gọi lệnh python và pip từ bất kỳ thư mục nào.

Tiếp theo, để nâng cao hiệu suất làm việc và quản lý dự án Python, cần cài đặt một IDE. Visual Studio Code (VS Code) là lựa chọn phổ biến nhờ tính nhẹ, linh hoạt và khả năng mở rộng mạnh mẽ thông qua hệ sinh thái Extension. Sau khi cài đặt VS Code, người dùng nên cài thêm tiện ích Python Extension do Microsoft phát triển, giúp bổ sung các tính năng quan trọng như tự động hoàn thành mã, tô sáng cú pháp, chạy và gỡ lỗi trực tiếp trong IDE, cùng khả năng quản lý môi trường ảo. VS Code sẽ tự động phát hiện phiên bản Python trên máy khi mở dự án, từ đó hoàn tất quá trình chuẩn bị môi trường phát triển, sẵn sàng cho việc lập trình Python một cách hiệu quả và chuyên nghiệp.

## 3.2. Thiết kế cấu trúc dữ liệu và biểu diễn đồ thị

### 3.2.1. Mô hình dữ liệu đồ thị

Đồ thị trong bài toán có các đặc trưng:

- Số lượng đỉnh tương đối nhỏ.
- Số lượng cạnh không quá lớn nhưng phân bố không đều.
- Không phải tỉnh nào cũng có quan hệ kết nối trực tiếp với tất cả tỉnh còn lại.

Với những đặc điểm trên, danh sách kề (Adjacency List) được lựa chọn làm cấu trúc biểu diễn chính cho đồ thị.

Lý do lựa chọn:

- Tiết kiệm bộ nhớ so với ma trận kề, vốn cần  $O(n^2)$  không gian.
- Thao tác duyệt BFS hiệu quả vì có thể truy xuất trực tiếp danh sách các đỉnh kề của một node.
- Dễ dàng cập nhật khi thêm dữ liệu từ Google Directions API.
- Tính linh hoạt cao, phù hợp cho đồ thị thưa và dữ liệu địa lý thực tế.

Đồ thị được cài đặt trong Python bằng kiểu dữ liệu dict, trong đó:

- Key: tên tỉnh/thành phố (string).
- Value: danh sách các tỉnh có thể đi trực tiếp đến (list of strings).

### 3.2.2. Thu thập và chuẩn hóa dữ liệu đồ thị

Để Ứng dụng sử dụng dữ liệu kề đã được chuẩn bị sẵn từ một nguồn chính xác và đáng tin cậy trên Github. Quy trình thu thập/chuẩn hóa:

- Biên soạn danh sách mã tỉnh/thành và quan hệ kề dựa trên nguồn bản đồ/địa giới hành chính tin cậy (các tuyến đường bộ thực tế).
- Rà soát thủ công các cặp tỉnh (A, B) để xác nhận có kết nối giao thông hợp lệ; loại bỏ các cặp không có tuyến đường.
- Lưu danh sách tỉnh (provinces.json) và danh sách kề (adjacency.json) ở định dạng JSON.
- Khi khởi chạy, chương trình nạp hai tệp JSON, kiểm tra mã tỉnh hợp lệ, đảm bảo mọi mã trong adjacency đều tồn tại trong danh sách tỉnh, và loại bỏ các kết nối không hợp lệ.

Cách làm này:

- Tránh phụ thuộc API bên ngoài và hạn chế chi phí/gọi mạng.
- Đảm bảo dữ liệu kề bám sát thực tế và đã được kiểm tra thủ công.
- Tăng độ tin cậy cho bài toán tìm đường vì dữ liệu đầu vào được kiểm chứng.

### 3.2.3. Các cấu trúc dữ liệu

Để triển khai BFS một cách tối ưu, chương trình sử dụng thêm các cấu trúc dữ liệu sau:

- Tập đánh dấu đỉnh đã duyệt:
  - + Được cài đặt bằng cấu trúc set của Python.
  - + Đảm bảo độ phức tạp trung bình  $O(1)$  cho phép kiểm.
  - + Tránh lặp lại đỉnh, ngăn hiện tượng duyệt vòng.
- Hàng đợi lưu trữ các đường đi:

- + Lý do chọn hàng đợi thay vì danh sách: Thao tác pop từ đầu hàng  $O(1)$  (list tốn  $O(n)$ ), thao tác push vào cuối  $O(1)$ , tối ưu cho mô hình hàng đợi FIFO trong BFS.
- + Mỗi phần tử trong queue không chỉ là một node, mà là cả một đường đi.
- + Cách cài đặt này giúp: Khi BFS tìm đến đỉnh đích, trả về đường đi đầy đủ ngay lập tức, không cần sử dụng mảng cha (parent array) để truy ngược đường đi.

### **3.3. Cài đặt thuật toán**

#### **3.3.1. Mục tiêu triển khai**

Phần này mô tả cách nhóm hiện thực thuật toán BFS trong chương trình Python, bao gồm:

- Biểu diễn đồ thị bằng danh sách kề.
- Cài đặt hàm BFS tìm đường đi giữa 2 tỉnh.
- Tổ chức mã nguồn chương trình.
- Kết hợp giao diện nhập/xuất đơn giản bằng swagger.

Nội dung nhằm thể hiện khả năng ứng dụng thuật toán vào chương trình thực tế, không chỉ lý thuyết

#### **3.3.2. Biểu diễn đồ thị bằng danh sách kề**

Đồ thị được lưu trong tệp JSON (adjacency.json) dưới dạng dictionary: mỗi tỉnh là một đỉnh (key), giá trị là danh sách mã tỉnh kề trực tiếp. Khi chạy, ứng dụng nạp JSON này và dùng làm danh sách kề cho BFS. Ví dụ rút gọn:

ADJACENCY\_DATA: Dict[str, List[str]] = {

# Miền Bắc

"01": ["24", "25", "33", "37", "19"], # Hà Nội giáp: Bắc Ninh, Phú Thọ, Hưng Yên, Ninh Bình, Thái Nguyên

"04": ["08", "19", "20"], # Cao Bằng giáp: Tuyên Quang, Thái Nguyên, Lạng Sơn

"08": ["04", "15", "19", "25"], # Tuyên Quang giáp: Cao Bằng, Lào Cai, Thái Nguyên, Phú Thọ

"11": ["12", "14"], # Điện Biên giáp: Lai Châu, Sơn La

"12": ["11", "14", "15"], # Lai Châu giáp: Điện Biên, Sơn La, Lào Cai

"14": ["11", "12", "15", "25", "38"], # Sơn La giáp: Điện Biên, Lai Châu, Lào Cai, Phú Thọ, Thanh Hóa

"15": ["08", "12", "14", "19", "25"], # Lào Cai giáp: Tuyên Quang, Lai Châu, Sơn La, Thái Nguyên, Phú Thọ

"19": ["01", "04", "08", "15", "20", "24", "25"], # Thái Nguyên giáp: Hà Nội, Cao Bằng, Tuyên Quang, Lào Cai, Lạng Sơn, Bắc Ninh, Phú Thọ

"20": ["04", "19", "22", "24"], # Lạng Sơn giáp: Cao Bằng, Thái Nguyên, Quảng Ninh, Bắc Ninh

"22": ["20", "24", "31"], # Quảng Ninh giáp: Lạng Sơn, Bắc Ninh, Hải Phòng

```
"24": ["01", "19", "20", "22", "31", "33"], # Bắc Ninh giáp: Hà Nội, Thái
Nguyên, Lạng Sơn, Quảng Ninh, Hải Phòng, Hưng Yên

"25": ["01", "08", "14", "15", "19", "37", "38"], # Phú Thọ giáp: Hà Nội,
Tuyên Quang, Sơn La, Lào Cai, Thái Nguyên, Ninh Bình, Thanh Hóa

"31": ["22", "24", "33"], # Hải Phòng giáp: Quảng Ninh, Bắc Ninh, Hưng Yên

"33": ["01", "24", "31", "37"], # Hưng Yên giáp: Hà Nội, Bắc Ninh, Hải
Phòng, Ninh Bình
```

Cách biểu diễn bằng danh sách kề giúp tiết kiệm bộ nhớ và tối ưu khi truy cập các đỉnh lân cận, phù hợp với kích thước đồ thị trung bình và đặc điểm bài toán.

Đây là dữ liệu JSON mô tả thông tin của các tỉnh/thành phố ở Việt Nam, mỗi tỉnh là một object bao gồm nhiều thuộc tính. Cụ thể hơn, đây chính là dataset hành chính Việt Nam chuẩn hóa, bao gồm mã tỉnh, tên tỉnh, tên tiếng Anh, tên đầy đủ, slug code, và đặc biệt tọa độ địa lý.

Do diện tích mỗi tỉnh/thành rất rộng, nhóm chọn tọa độ của trung tâm hành chính (UBND tỉnh/thành phố) làm đại diện, giúp việc tính toán khoảng cách và trực quan hóa trở nên nhất quán và khả thi.

```
[
  {
```

```
"id": 1,  
  
"code": "01",  
  
"name": "Hà Nội",  
  
"name_en": "Ha Noi",  
  
"full_name": "Thành phố Hà Nội",  
  
"full_name_en": "Ha Noi City",  
  
"code_name": "ha_noi",  
  
"coordinates": [21.028575813283485, 105.85412781421053]  
  
},  
  
{  
  
"id": 2,  
  
"code": "04",  
  
"name": "Cao Bằng",  
  
"name_en": "Cao Bang",  
  
"full_name": "Tỉnh Cao Bằng",  
  
"full_name_en": "Cao Bang Province",  
  
"code_name": "cao_bang",
```

```
"coordinates": [22.666324012172744, 106.25794629812626]

}

]
```

### 3.3.3. Cài đặt thuật toán BFS tìm đường đi

Thuật toán được triển khai theo đúng nguyên lý duyệt theo chiều rộng, sử dụng cấu trúc hàng đợi (queue) để lưu trữ tuần tự các đường đi đang xét. Điểm đặc biệt trong cài đặt là mỗi phần tử trong hàng đợi là một đường đi đầy đủ, cho phép trả về kết quả ngay khi tìm thấy đỉnh đích mà không phải truy ngược lại.

Hàm cài đặt như sau:

```
from collections import deque

def bfs_shortest_path(start_city, end_city, graph):

    if start_city == end_city:

        return [start_city]

    if start_city not in graph or end_city not in graph:

        return None

    queue = deque([[start_city]])
```

```
visited = set([start_city])

while queue:

    path = queue.popleft()

    current = path[-1]

    for neighbor in graph[current]:

        if neighbor not in visited:

            new_path = path + [neighbor]

            if neighbor == end_city:

                return new_path

            visited.add(neighbor)

            queue.append(new_path)

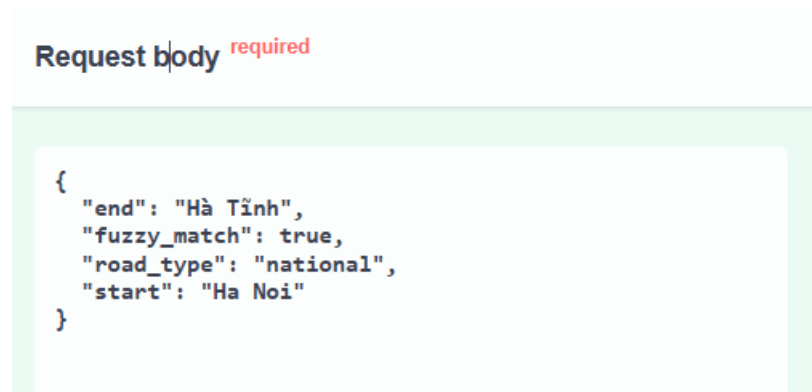
return None
```

Hàm trên đảm bảo:

- Tìm được đường đi ngắn nhất theo số cạnh.
- Không duyệt trùng lặp đỉnh đã thăm.
- Trả về kết quả dưới dạng danh sách các tỉnh theo thứ tự từ xuất phát đến đích.

### 3.3.4. Chương trình chính

- Ứng dụng chạy FastAPI và tự sinh trang thử nghiệm Swagger UI tại [Finding Distance API - Swagger UI](http://localhost:8000/docs#/) (<http://localhost:8000/docs#/>)
- Người dùng nhập tỉnh bắt đầu/kết thúc, chọn fuzzy\_match (cho phép khớp tên gần đúng) và road\_type (national/expressway/local...) rồi gửi yêu cầu.
- Kết quả hiển thị: đường đi ngắn nhất (theo số cạnh), danh sách tỉnh trên lộ trình, số bước, thời gian xử lý.
- Các tình huống được xử lý: hai tỉnh trùng nhau, tỉnh không tồn tại, không có đường đi, vẫn đảm bảo trả về đường đi ngắn nhất theo số cạnh.
- Request đầu vào:



Hình 3: Mẫu request khi test chương trình

- Response dạng json nhận được khi thành công:

```
{
  "path": [
    "Hà Nội",
    "Phú Thọ",
    "Thanh Hóa",
    "Nghệ An",
    "Hà Tĩnh"
  ],
  "path_codes": [
    "01",
    "25",
```

```
"38",
"40",
"42"
],
"path_coordinates": [
  {
    "code": "01",
    "name": "Hà Nội",
    "latitude": 21.028575813283485,
    "longitude": 105.85412781421053
  },
  {
    "code": "25",
    "name": "Phú Thọ",
    "latitude": 21.32289871889187,
    "longitude": 105.40224696344426
  },
  {
    "code": "38",
    "name": "Thanh Hóa",
    "latitude": 19.807370696293415,
    "longitude": 105.77539935701147
  },
  {
    "code": "40",
    "name": "Nghệ An",
    "latitude": 18.673214284412587,
    "longitude": 105.69287136760134
  },
  {
    "code": "42",
    "name": "Hà Tĩnh",
    "latitude": 18.343380125903806,
    "longitude": 105.90565935224903
  }
],
"distance": 5,
"total_distance_km": 539.32,
"real_distance_km": 486.17,
"road_type": "national",
```

```

"start_province": {
  "code": "01",
  "name": "Hà Nội",
  "full_name": "Thành phố Hà Nội",
  "coordinates": {
    "latitude": 21.028575813283485,
    "longitude": 105.85412781421053
  }
},
"end_province": {
  "code": "42",
  "name": "Hà Tĩnh",
  "full_name": "Tỉnh Hà Tĩnh",
  "coordinates": {
    "latitude": 18.343380125903806,
    "longitude": 105.90565935224903
  }
},
"execution_time_ms": 0.2894999925047159,
"timestamp": "2025-12-13T18:46:22.900039"
}

```

- Kết quả trả về gồm cả mã tỉnh, tên tỉnh và toạ độ cho từng điểm trên lộ trình, thuận tiện để trực quan hóa (vẽ polyline trên bản đồ hoặc kiểm tra thủ công).
- Trường `total_distance_km` thể hiện tổng chiều dài ước lượng theo loại đường đã chọn thông qua duyệt bằng thuật toán BFS; `execution_time_ms` phản ánh thời gian xử lý BFS và dựng kết quả.

### 3.3.5. Đánh giá phương pháp cài đặt

#### Ưu điểm:

- Thuật toán đơn giản, dễ triển khai và kiểm chứng.
- Cấu trúc mã tách biệt, phù hợp chuẩn thiết kế phần mềm.
- Đảm bảo tìm được đường đi ngắn nhất trong đồ thị không trọng số.

- Đầu ra rõ ràng, dễ quan sát và kiểm thử.

### **Hạn chế:**

- Đồ thị sử dụng dữ liệu tĩnh, chưa cập nhật theo bản đồ thực tế.
- Chưa xét đến trọng số đường đi như khoảng cách hoặc thời gian.
- Giao diện tương tác còn đơn giản.

### **Khả năng mở rộng:**

- Tích hợp bản đồ thực qua Google Maps API hoặc OpenStreetMap.
- Bổ sung thuật toán Dijkstra hoặc A\* để xử lý đồ thị có trọng số.
- Phát triển giao diện web hoặc ứng dụng hiển thị đường đi trên bản đồ.

## Chương IV. KIỂM THỬ

### 4.1. Test case

Mục tiêu kiểm thử:

- Thuật toán BFS được cài đặt hoạt động đúng theo lý thuyết.
- Hệ thống có thể tìm đường đi giữa hai đỉnh (nếu tồn tại).
- Chương trình xử lý tốt các trường hợp biên.
- Thời gian xử lý và tài nguyên sử dụng đạt mức chấp nhận được cho dữ liệu bài toán.

Phạm vi kiểm thử:

- Kiểm thử chức năng nhập dữ liệu đồ thị.
- Kiểm thử hàm BFS tìm đường đi.
- Kiểm thử xử lý các trường hợp đặc biệt của đồ thị: đồ thị rỗng, đồ thị có chu trình, đồ thị có nhiều đường đi, đồ thị không liên thông...

Các kịch bản kiểm thử:

Bảng 1: Các kịch bản kiểm thử chương trình.

Tên test case	Mục đích	Input	Output kỳ vọng
TC01	Tìm đúng đường đi ngắn nhất.	Start = Nghe An, End = Quang Tri	Đường đi ngắn nhất trả về dạng mã/tên tỉnh, ví dụ: Nghệ An → Hà Tĩnh → Quảng

			Bình → Quảng Trị (4 đỉnh, 3 cạnh).
TC02	Trường hợp Start = End.	Start = Ha Noi, End = Ha Noi	Trả về danh sách chỉ chứa Hà Nội; số bước = 0.
TC03	Không tồn tại đường đi.	Start = Tokyo, End = Ha Noi	Báo lỗi/None: tỉnh không tồn tại trong hệ thống.
TC04	Kiểm thử API.	Start và / hoặc End rỗng	Thông báo lỗi: Vui lòng nhập tỉnh bắt đầu; Vui lòng nhập tỉnh kết thúc.

## 4.2. Kết quả chạy thử

Bảng 2: Kết quả kiểm thử các kịch bản.

Tên test case	Output	Đánh giá
TC01	Nghệ An → Hà Tĩnh → Quảng Bình → Quảng Trị; số bước 3; HTTP 200 nếu gọi qua API.	Pass
TC02	Hà Nội; số bước 0; HTTP 200.	Pass

TC03	Thông báo lỗi “tỉnh không tồn tại” (InvalidInputError/ValueError); HTTP 400.	Pass
TC04	status = 422 - ValidationError, thông báo lỗi: Vui lòng nhập tỉnh bắt đầu; Vui lòng nhập tỉnh kết thúc.	Pass

### 4.3. Đánh giá

- Đánh giá tính đúng đắn:
  - + BFS luôn trả về đường đi ít bước nhảy nhất (theo số cạnh).
  - + adjacency.json/ADJACENCY\_LIST đầy đủ cho bộ 34 tỉnh đang dùng.
  - + Đường đi thu được phù hợp logic BFS và tuyến kẻ đã chuẩn hóa.
- Đánh giá hiệu năng:
  - + BFS chạy trong thời gian  $O(V + E) \rightarrow$  rất nhanh với đồ thị 34 đỉnh.
  - + Phân nạp dữ liệu JSON và khởi tạo đồ thị nhẹ, không ảnh hưởng thời gian phản hồi.
- Đánh giá tính ổn định:
  - + Trường hợp không tìm thấy tỉnh/đường đi đã có thông báo lỗi rõ ràng.
  - + Không phụ thuộc dịch vụ ngoài khi chạy BFS; chỉ cần dữ liệu tĩnh sẵn có.
- Hạn chế:
  - + ADJACENCY\_LIST là dữ liệu tĩnh; nếu có thay đổi(sắp nhập, tách rời địa giới hành chính) phải chỉnh thủ công.

- + Không có metadata giao thông (kẹt xe, tốc độ...), nên chỉ tối ưu theo số cạnh, không theo thời gian/chi phí.
- Khả năng mở rộng:
  - + Bổ sung metadata giao thông (loại đường, tốc độ gợi ý) để xếp hạng tuyến.
  - + Thêm cơ chế retry/validation nâng cao cho đầu vào và dữ liệu kê.

## Chương V: HƯỚNG PHÁT TRIỂN

Mặc dù hệ thống hiện tại đã mô phỏng thành công bài toán tìm đường đi giữa hai tỉnh của Việt Nam bằng thuật toán BFS, đề tài vẫn còn nhiều tiềm năng để mở rộng nhằm tiệm cận hơn với các yêu cầu thực tế của các bài toán định tuyến và xử lý dữ liệu không gian. Một số hướng phát triển tiêu biểu có thể thực hiện trong tương lai như sau:

- Bổ sung trọng số cho các cạnh: Đồ thị đang được mô hình hóa là đồ thị không trọng số, vì vậy BFS phù hợp nhưng chưa phản ánh được yếu tố thực tế như khoảng cách hoặc thời gian di chuyển. Trong bước phát triển tiếp theo, nhóm có thể gán trọng số cho mỗi cạnh dựa trên các tham số như: khoảng cách địa lý (km), thời gian di chuyển, hoặc loại đường (cao tốc, quốc lộ, tỉnh lộ). Khi đó, bài toán sẽ trở nên thực tế hơn và có thể áp dụng các thuật toán tìm đường tối ưu theo chi phí.
- Áp dụng các thuật toán nâng cao: BFS chỉ phù hợp trong đồ thị không trọng số. Khi đã bổ sung trọng số, nhiều thuật toán tìm kiếm tối ưu hơn có thể được triển khai như: Uniform Cost Search, Dijkstra, A\* hoặc Greedy Best First Search. Các thuật toán này cho phép tìm đường đi ngắn nhất theo quãng đường thực tế hoặc thời gian di chuyển, phù hợp hơn với các ứng dụng bản đồ số.
- Tích hợp dữ liệu bản đồ thực từ các API chuyên dụng:
- Trong tương lai, nhóm có thể tích hợp dữ liệu từ các dịch vụ bản đồ như Google Directions API, OpenStreetMap hoặc OpenRouteService để:
  - + Tự động xác định tỉnh/thành có tuyến đường di chuyển trực tiếp.
  - + Tính toán khoảng cách và thời gian di chuyển chính xác.
  - + Cập nhật dữ liệu theo thay đổi thực tế.
- Xây dựng giao diện bản đồ trực quan: Hệ thống hiện tại mới chỉ xuất dữ liệu dạng danh sách. Một hướng phát triển hữu ích là xây dựng giao diện đồ họa

trực quan bằng web (LeafletJS, Mapbox, ReactJS...) để hiển thị đường đi trên bản đồ Việt Nam, cho phép người dùng chọn điểm xuất phát và điểm đích trực tiếp bằng thao tác chuột. Điều này giúp quá trình mô phỏng trở nên trực quan và dễ sử dụng hơn.

- Phát triển API dịch vụ tìm đường hoàn thiện: Đề tài có thể được mở rộng thành một hệ thống cung cấp dịch vụ định tuyến thông qua REST API. API này cho phép các ứng dụng khác gọi để lấy đường đi, khoảng cách, hoặc số bước di chuyển. Đây là bước quan trọng để tích hợp hệ thống vào các ứng dụng lớn hơn như phần mềm quản lý giao thông, định vị đường đi hoặc ứng dụng học tập.
- Ứng dụng thêm các kỹ thuật trí tuệ nhân tạo: Trong các bước phát triển nâng cao, nhóm có thể xem xét áp dụng các phương pháp AI như: học máy để dự đoán thời gian di chuyển theo điều kiện giao thông, hoặc Graph Neural Network để phân tích và tối ưu hóa mạng lưới đường bộ. Đây là những hướng tiếp cận hiện đại, mang tính nghiên cứu sâu hơn và có khả năng ứng dụng thực tế mạnh mẽ.

## KẾT LUẬN

Trong khuôn khổ bài tập lớn môn Trí tuệ nhân tạo, nhóm đã nghiên cứu và áp dụng thuật toán BFS (Breadth-First Search) để giải quyết bài toán tìm đường đi giữa hai tỉnh tại Việt Nam thông qua mô hình đồ thị vô hướng, không trọng số. Với đặc tính duyệt theo từng mức và đảm bảo tìm được đường đi ngắn nhất theo số cạnh, BFS được chứng minh là thuật toán phù hợp cho các bài toán tìm đường trong đồ thị không trọng số.

Báo cáo đã trình bày đầy đủ cơ sở lý thuyết về đồ thị, nguyên lý hoạt động của BFS, cách xây dựng cấu trúc dữ liệu, thiết kế chương trình và đánh giá kết quả kiểm thử. Kết quả thực nghiệm cho thấy chương trình có khả năng tìm được đường đi hợp lý, đúng với logic địa lý mô phỏng, thời gian xử lý nhanh và thuật toán vận hành ổn định trong hầu hết các kịch bản.

Mặc dù đề tài đã đạt được các mục tiêu đặt ra, nhóm nhận thấy vẫn còn những hạn chế nhất định như dữ liệu đồ thị còn đơn giản, chưa xét đến yếu tố khoảng cách hay thời gian thực tế, và chương trình chưa bổ sung các thuật toán nâng cao hơn như UCS, Dijkstra hay A\*. Đây cũng chính là những hướng phát triển tiềm năng cho các nghiên cứu tiếp theo.

Tổng kết lại, đề tài đã giúp nhóm củng cố kiến thức về lý thuyết đồ thị, hiểu rõ hơn cơ chế hoạt động của BFS, đồng thời tăng khả năng áp dụng giải thuật vào các bài toán thực tiễn. Đây là nền tảng quan trọng để tiếp tục nghiên cứu các thuật toán tìm kiếm và tối ưu trong lĩnh vực trí tuệ nhân tạo.

## TÀI LIỆU THAM KHẢO

- [1] Bộ môn Trí tuệ Nhân tạo, *Giáo trình Trí tuệ nhân tạo*, Khoa Công nghệ Thông tin, Trường Đại học Công nghiệp Hà Nội, 2022.
- [2] Mai Thanh Hồng, *Slide bài giảng môn Trí tuệ Nhân tạo – Chương 3: Các thuật toán tìm kiếm trên đồ thị*, Trường Đại học Công nghiệp Hà Nội, 2024.
- [3] Stuart Russell & Peter Norvig, *Artificial Intelligence: A Modern Approach*, 3rd Edition, Pearson, 2010.
- [4] GeeksforGeeks, “Breadth-First Search (BFS) Algorithm”,  
<https://www.geeksforgeeks.org/breadth-first-search-bfs-algorithm/>
- [5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C., *Introduction to Algorithms*, MIT Press, 2009.
- [6] Python Software Foundation, *Python 3 Documentation*, <https://docs.python.org/>
- [7] OpenStreetMap & Google Directions API – Tài liệu tham khảo về dữ liệu bản đồ và kết nối địa lý.