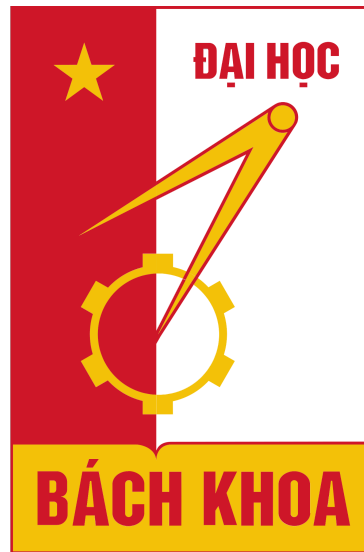


Trường Công nghệ Thông tin và Truyền thông  
Đại học Bách khoa Hà Nội



## Bài tập lớn nhập môn trí tuệ nhân tạo

Tìm đường đi trên bản đồ phòng Phan Chu Trinh

**Giảng viên hướng dẫn:**  
PGS.TS. Trần Đình Khang

**Nhóm 7:**  
Nguyễn Anh Đức - 20200167  
Phạm Trung Đức - 20205069  
Trần Văn Hiếu - 20200231  
Lê Đức Quý - 20205121  
Đào Huy Trường - 20200651

## Lời mở đầu

Với sự phát triển của khoa học công nghệ, các thành tựu của công nghệ đã đi sâu vào cuộc sống của chúng ta. Đối với việc tìm đường đi, thay vì phải nhìn đường đi qua bản đồ giấy thì với sự phát triển của các công cụ AI, đặc biệt phải kể đến Google Maps thì việc tìm kiếm đường đi giữa các điểm đã trở nên dễ dàng hơn.

Bài toán tìm kiếm đường đi là một trong những bài toán cơ bản và quan trọng trong lĩnh vực Trí tuệ nhân tạo. Được lấy cảm hứng từ cách con người tìm kiếm lời giải cho các vấn đề phức tạp, mục tiêu của bài toán này là tìm ra một chuỗi các hành động tối ưu để đi từ một trạng thái ban đầu đến một trạng thái đích.

Trong lĩnh vực Trí tuệ nhân tạo, có nhiều phương pháp và thuật toán khác nhau để giải quyết bài toán tìm kiếm đường đi. Một số phương pháp phổ biến bao gồm thuật toán DFS (Depth-First Search), thuật toán BFS (Breadth-First Search), thuật toán A\* (A-star), thuật toán Dijkstra và thuật toán quy hoạch động.

Các thuật toán tìm kiếm đường đi trong Trí tuệ nhân tạo không chỉ có ứng dụng trong các bài toán về robot tự hành, trò chơi điện tử hay hệ thống định tuyến mạng, mà còn được sử dụng trong nhiều lĩnh vực khác nhau như lập kế hoạch, lập lịch, và tối ưu hóa.

Bài toán tìm kiếm đường đi đóng góp không nhỏ vào việc giải quyết các vấn đề thực tế và là một trong những nền tảng quan trọng trong lĩnh vực Trí tuệ nhân tạo. Để đạt được mục tiêu tối ưu, các nhà nghiên cứu liên tục phát triển và cải tiến các thuật toán tìm kiếm đường đi, tạo nên sự phát triển không ngừng cho lĩnh vực này.

Bài tập lớn môn Nhập môn trí tuệ nhân tạo của nhóm chúng em nhằm khám phá và hiểu sâu hơn về các phương pháp biểu diễn đường đi trên đồ thị, hướng đến việc sử dụng các công cụ cần thiết, giúp người dùng có thể tìm kiếm, nhìn thấy đường đi giữa điểm bắt đầu và điểm kết thúc trên giao diện tương tác gần như tương tự sản phẩm Google Maps.

Bài tập lớn này sẽ đòi hỏi khả năng tư duy logic, khả năng triển khai thuật toán và khả năng phân tích kết quả. Qua đó, chúng ta sẽ nắm vững các kiến thức cơ bản về đường đi trên đồ thị và có thể ứng dụng chúng vào các bài toán thực tế như tìm đường đi ngắn nhất trong mạng lưới giao thông, lập lịch công việc hoặc tối ưu hóa tuyến đường vận chuyển.

## Mục lục

1	Giới thiệu bài toán	3
2	Biểu diễn bài toán	3
3	Phương pháp tìm kiếm lời giải	3
4	Cài đặt chương trình	4
5	Hướng dẫn	11
6	Kết quả	11

## 1 Giới thiệu bài toán

Xây dựng một chương trình tìm đường đi trên ảnh bản đồ của phường Phan Chu Trinh. Chương trình cho phép người dùng chọn điểm đầu và điểm cuối để thực hiện tìm kiếm đường đi. Chương trình có các chức năng sau:

- Hiện thị bản đồ khu vực phường Phan Chu Trinh.
- Vẽ các đường giới hạn xung quanh khu vực phường Phan Chu Trinh.
- Cho phép người dùng chọn 2 điểm A, B
- Hiện thị đường đi từ điểm A đến điểm B.

## 2 Biểu diễn bài toán

Đầu tiên, ta tạo dựng một đồ thị có hướng mô tả đường đi trong bản đồ. Bao gồm việc tạo các điểm (point) là các nút giao giữa các đường, điểm cuối các đường và khúc cua. Các cạnh (edges) là đường đi có thể đi được thẳng trực tiếp từ điểm này đến điểm kia. Đồ thị có hướng ( $G$ ) này được biểu diễn bằng ma trận kề  $M(G)$  với:

$$M_{ij} = \begin{cases} 1 & \text{nếu có cạnh từ đỉnh } i \text{ đến } j \\ 0 & \text{nếu không có cạnh từ } i \text{ đến } j \end{cases}$$

Trạng thái hiện tại là vị trí hiện tại. Vị trí người dùng chọn có thể là ngẫu nhiên trên bản đồ. Vậy nên ta thực hiện chọn trạng thái ban đầu là điểm thuộc đồ thị gần nhất với điểm xuất phát người dùng chọn ( $V_{xp}$ ). Tương tự với vị trí đích, ta chọn được điểm đích thuộc đồ thị ( $V_{dich}$ ).

$N = V_i$  với  $V_i$  là điểm thuộc đồ thị

$$N_0 = V_{xp} \mid \text{Đích} = V_{dich}$$

Sơ đồ chuyển đổi trạng thái :

$$A = \{V_i - > V_j \mid \text{nếu } M_{ij} = 1\}$$

## 3 Phương pháp tìm kiếm lời giải

Trong bài tập lớn này nhóm chúng em sau khi đã mô hình hóa, biểu diễn được bài toán trên không gian bản đồ qua các điểm giao nhau, điểm giới hạn khu vực của phường thì với yêu cầu phải tìm kiếm đường đi giữa 2 điểm trên bản đồ chúng em đã chọn phương pháp tìm kiếm đường đi bằng BFS

So với các cách tìm kiếm đường đi khác tương tự như DFS thì chúng em đã chọn phương pháp BFS bởi vì BFS có những ưu thế hơn ở điểm BFS sẽ khám phá tất cả các đỉnh cùng mức trước khi đi sâu hơn vào các đỉnh cấp dưới. Điều này đảm bảo rằng BFS sẽ tìm thấy đường đi nếu có, ngay cả khi đồ thị không liên thông. Trong khi đó, DFS có thể "mắc kẹt" trong một nhánh khiến cho việc tìm kiếm không hoàn thành nếu đỉnh đích không được tìm thấy trong nhánh đó. Hơn nữa BFS duyệt qua các đỉnh cùng mức trước khi di chuyển xuống các đỉnh cấp dưới. Điều này đảm bảo rằng BFS tìm được đường đi ngắn nhất từ đỉnh xuất phát đến đỉnh đích (nếu tồn tại). Trong khi đó, DFS có xu hướng di chuyển sâu vào cây tìm kiếm trước khi quay lại các nhánh khác, do đó không đảm bảo tìm được đường đi ngắn nhất và BFS sử dụng ít không gian bộ nhớ hơn.

Để sử dụng BFS thì chúng ta cần có deque cũng như một số hàm đặc trưng hàm `dis(point1, point2)` để tính khoảng cách bình phương Euclide giữa hai điểm `point1` và `point2`. Hàm tính khoảng cách này được sử dụng để xác định điểm gần nhất với một điểm mục tiêu. Hàm `nearest_point(points, target)` để tìm điểm gần nhất với một điểm mục tiêu (`target`) trong danh sách các điểm (`points`). Hàm này sử dụng hàm `dis` để tính khoảng cách giữa các điểm và trả về điểm có khoảng cách nhỏ nhất với điểm mục tiêu.

Hàm `path(start_point, end_point)` để tìm đường đi từ `start_point` đến `end_point`. Đầu tiên, nó tìm điểm gần nhất với `start_point` và `end_point` bằng cách sử dụng hàm `nearest_point`. Sau đó, nó khởi tạo một hàng đợi (`queue`) và thêm `start_point` vào hàng đợi với một danh sách đường đi ban đầu chứa chỉ một điểm `start_point`.

Trong vòng lặp `while`, chương trình lấy phần tử đầu tiên từ hàng đợi và kiểm tra xem nó có phải là điểm đích (`end_point`) hay không. Nếu là điểm đích, nghĩa là đã tìm thấy đường đi từ `start_point` đến `end_point`, chương trình trả về đường đi đó. Nếu không phải, chương trình tìm các điểm kề với điểm hiện tại (`current`) mà chưa được thăm, và thêm chúng vào hàng đợi với đường đi mới là `path + [point]`. Quá trình này tiếp tục cho đến khi hàng đợi trống.

Cuối cùng, nếu không tìm thấy đường đi từ `start_point` đến `end_point`, hàm trả về `None` để chỉ ra không có đường đi.

## 4 Cài đặt chương trình

```
[26] !pip install ipyleaflet
Requirement already satisfied: traitlets in /usr/local/lib/python3.10/site-packages (6.4.0)
Requirement already satisfied: xyzservices in /usr/local/lib/python3.10/site-packages (2022.9.1)
```

Hình 1: Thư viện ipyleaflet

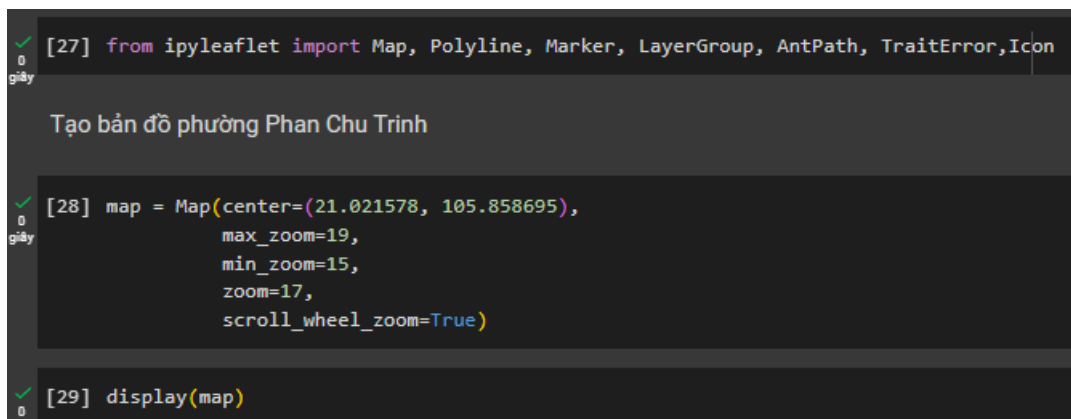
Trong bài tập này của chúng em thì chúng em viết chương trình trên công cụ Jupyter notebook và để lấy dữ liệu, tương tác,... với các thành phần của bản đồ thì cần sử dụng thư viện ipyleaflet.

Ipyleaflet là một thư viện Python mã nguồn mở được sử dụng để tạo và hiển thị bản đồ tương tác trong ứng dụng Jupyter Notebook. Nó cung cấp các công cụ và lớp để tạo ra các bản đồ tương tác, cho phép bạn hiển thị dữ liệu địa lý và thực hiện các tương tác như di chuyển, thu phóng và thêm các đối tượng lên bản đồ.

Ipyleaflet sử dụng thư viện Leaflet JavaScript để tạo ra các bản đồ tương tác. Thư viện này rất phổ biến trong cộng đồng phát triển web và cung cấp nhiều tính năng mạnh mẽ để làm việc với dữ liệu địa lý trên các trình duyệt web.

Với ipyleaflet, bạn có thể tạo các đối tượng bản đồ như marker (đánh dấu), polyline (đường đa giác), polygon (đa giác), circle (hình tròn), heatmap (biểu đồ nhiệt) và nhiều hơn nữa. Bạn cũng có thể tùy chỉnh giao diện người dùng, thêm lớp dữ liệu từ các nguồn bản đồ khác nhau và thực hiện các tương tác phức tạp trên bản đồ.

Ipyleaflet là một công cụ hữu ích cho việc trực quan hóa dữ liệu địa lý và thực hiện phân tích địa lý trong môi trường Jupyter Notebook.



```
[27] from ipyleaflet import Map, Polyline, Marker, LayerGroup, AntPath, TraitError, Icon

Tạo bản đồ phường Phan Chu Trinh

[28] map = Map(center=(21.021578, 105.858695),
              max_zoom=19,
              min_zoom=15,
              zoom=17,
              scroll_wheel_zoom=True)

[29] display(map)
```

Hình 2: Khởi tạo bản đồ phường Phan Chu Trinh

Đoạn mã trên tạo một đối tượng bản đồ (map) bằng cách sử dụng thư viện ipyleaflet trong Python. Dưới đây là các tham số và ý nghĩa của chúng:

- **center=(21.021578, 105.858695):** Đây là tọa độ trung tâm của bản đồ. Trong trường hợp này, bản đồ sẽ được căn chỉnh để trung tâm ở vị trí có tọa độ **latitude** là 21.021578 và **longitude** là 105.858695. Đây là điểm mà bản đồ sẽ được hiển thị ban đầu.
- **max\_zoom=19:** Đây là mức độ phóng to tối đa cho phép trên bản đồ. Trong trường hợp này, người dùng có thể phóng to tới mức độ 19.
- **min\_zoom=15:** Đây là mức độ thu nhỏ tối thiểu cho phép trên bản đồ. Trong trường hợp này, người dùng có thể thu nhỏ tới mức độ 15.

- **zoom=17:** Đây là mức độ phóng to ban đầu của bản đồ. Trong trường hợp này, bản đồ sẽ được hiển thị ban đầu ở mức độ phóng to 17.
- **scroll\_wheel\_zoom=True:** Tham số này cho phép người dùng sử dụng bánh xe cuộn của chuột để phóng to và thu nhỏ trên bản đồ.

Tổng quan, đoạn mã trên tạo một đối tượng bản đồ với tọa độ trung tâm được định nghĩa, giới hạn phóng to và thu nhỏ, mức độ phóng to ban đầu và cho phép sử dụng bánh xe cuộn để phóng to và thu nhỏ.

```
Tạo các điểm giao, điểm lân cận và vẽ biên cho phường Phan Chu Trinh
Phần đầu của mảng points là các điểm ven của phường Phần sau của mảng points là các điểm giao của các đường trong phường
edges là tập hợp các đường đi từ điểm X đến các điểm khác
hàm map.add_layer là hiển thị vùng ven

[30] points = [[21.022749, 105.857527], [21.023684, 105.857239], [21.024110, 105.857130], [21.024452, 105.857254], [21.024500,
[21.024726, 105.858295], [21.024861, 105.858954], [21.025003, 105.859952], [21.023054, 105.860787],
[21.018512, 105.861815], [21.018391, 105.860996], [21.019042, 105.858728], [21.019099, 105.858469], [21.018701, 105.8
[21.018356, 105.855020], [21.017868, 105.854060], [21.018075, 105.852852], [21.019156, 105.853176], [21.020551, 105.
[21.022226, 105.854270], [21.022959, 105.854525], [21.023663, 105.854754], [21.023493, 105.855462], [21.023126, 105.

[21.022763, 105.857536], [21.023617, 105.857787], [21.024261, 105.858286], [21.023268, 105.859426],
[21.022344, 105.859743], [21.022269, 105.859494], [21.022176, 105.859114], [21.022037, 105.859566],
[21.021904, 105.859004], [21.022165, 105.859809], [21.022327, 105.860444], [21.021968, 105.859846],
[21.021915, 105.859621], [21.021733, 105.859917], [21.021869, 105.860435], [21.021555, 105.859948], [21.021379, 105.
[21.020948, 105.860155], [21.021186, 105.861045], [21.020638, 105.860239], [21.021145, 105.858044], [21.021946, 105.
[21.021250, 105.857481], [21.019593, 105.856934], [21.019026, 105.856753], [21.019459, 105.855394], [21.019061, 105.
[21.018323, 105.853975], [21.018977, 105.853674], [21.019312, 105.853846], [21.019994, 105.855584], [21.021644, 105.
[21.022754, 105.855205], [21.022425, 105.856391]]

def hash(point):
    return point[0]**2 + point[1]**2
```

Hình 3: Khởi tạo các tọa độ điểm

Để tối ưu tốc độ xử lý, bộ nhớ cần thiết để mô tả đồ thị cũng như phù hợp với cấu trúc dữ liệu sử dụng cho thuật toán BFS, nhóm sử dụng cấu trúc danh sách kề thay cho ma trận kề.

Đoạn code khai báo mảng `points` gồm các thành phần biên của vùng bản đồ phường Phan Chu Trinh và các thành phần điểm giao nhau giữa 2 đường bất kỳ trong bản đồ

```

edges = {
  hash(points[0]):[points[1],points[24]],
  hash(points[1]):[points[0],points[25],points[2]] ,
  hash(points[2]):[points[26],points[1],points[3]] ,
  hash(points[3]):[points[2],points[4]] ,
  hash(points[4]):[points[26],points[3],points[5]] ,
  hash(points[5]):[points[4],points[6],points[26]] ,
  hash(points[6]):[points[5],points[7],points[27]] ,
  hash(points[7]):[points[8],points[6]] ,
  hash(points[8]):[points[9]] ,
  hash(points[9]):[points[10]] ,
  hash(points[10]):[points[9],points[11]] ,
  hash(points[11]):[points[44],points[47],points[10],points[12]] ,
  hash(points[12]):[points[47],points[11],points[13]] ,
  hash(points[13]):[points[48],points[12],points[14]] ,
  hash(points[14]):[points[50],points[52],points[13],points[15]] ,
  hash(points[15]):[points[16],points[14]] ,
  hash(points[16]):[points[15]] ,
  hash(points[17]):[points[54],points[16]] ,
  hash(points[18]):[points[56],points[17]] ,
  hash(points[19]):[points[58],points[18]] ,
  hash(points[20]):[points[59],points[19]] ,
  hash(points[21]):[points[20]] ,
  hash(points[22]):[points[59]] ,
  hash(points[23]):[points[60]] ,
  hash(points[24]):[points[0],points[27],points[45]] ,
  hash(points[25]):[points[1]] ,
  hash(points[26]):[points[2],points[4],points[5]] ,
  hash(points[27]):[points[6],points[28],points[24]] ,
  hash(points[28]):[points[27],points[29],points[33]] ,
  hash(points[29]):[points[28],points[30],points[31]] ,
  hash(points[30]):[points[29]] ,
  hash(points[31]):[points[29],points[32]] ,
  hash(points[32]):[points[31]] ,
  hash(points[33]):[points[28],points[34],points[35]] ,
  hash(points[34]):[points[33]] ,
  hash(points[35]):[points[33],points[36],points[37]] ,
  hash(points[36]):[points[35]] ,
  hash(points[37]):[points[35],points[38],points[39]] ,
  hash(points[38]):[points[37]] ,
  hash(points[39]):[points[37],points[40],points[41]] ,
  hash(points[40]):[points[39]] ,
  hash(points[41]):[points[39],points[42],points[43]] ,
  hash(points[42]):[points[41]] ,
  hash(points[43]):[points[41]] ,
  hash(points[44]):[points[11],points[45],points[46]] ,
  hash(points[45]):[points[44],points[46],points[24]] ,
  hash(points[46]):[points[44],points[45],points[47],points[57]] ,
  hash(points[47]):[points[11],points[46],points[48],points[12]] ,
  hash(points[48]):[points[13],points[47],points[49]] ,
  hash(points[49]):[points[48],points[50],points[56]] ,
  hash(points[50]):[points[49],points[14],points[51]] ,
  hash(points[51]):[points[50]] ,
  hash(points[52]):[points[14],points[53],points[54]] ,
  hash(points[53]):[points[52]] ,
  hash(points[54]):[points[52],points[55],points[17]] ,
  hash(points[55]):[points[54]] ,
  hash(points[56]):[points[49],points[18],points[57]] ,
  hash(points[57]):[points[56],points[58],points[46],points[60]] ,
  hash(points[58]):[points[57],points[19],points[59]] ,
  hash(points[59]):[points[58],points[60],points[20],points[22]] ,
  hash(points[60]):[points[59],points[23],points[57]]
}

```

Hình 4: Khởi tạo đường đi

Đoạn code mô hình hóa các đường đi trên bản đồ bằng việc biến nó thành các cạnh với 2 đầu là 2 điểm đầu cuối của các đường

Đoạn mã trên định nghĩa một hàm `hash` nhận một tham số `point`, tính toán giá trị bình phương của hai phần tử trong `point`, và trả về tổng của hai giá trị bình phương đó.

```

map.add_layer(Polyline(
  locations=[points[0:23] + [points[0]]],
  dash_array="4",
  color="red",
  weight=2,
  fill=False,
))

```

Hình 5: Tạo giới hạn khu vực



Đoạn mã trên thêm một lớp Polyline (đường đa giác) vào đối tượng bản đồ map trong thư viện ipyleaflet. Dưới đây là ý nghĩa của các tham số và đối số trong mã:

- **map.add\_layer:** Phương thức `add_layer` được gọi trên đối tượng bản đồ map để thêm một lớp mới vào bản đồ.
- **Polyline:** Đối tượng Polyline được tạo ra và được sử dụng làm lớp để hiển thị đường đa giác trên bản đồ.
- **locations=[points[0:23] + [points[0]]]:** Đây là một đối số của Polyline và định nghĩa các điểm tạo thành đường đa giác. Trong trường hợp này, danh sách `points[0:23]` được cắt từ danh sách `points`, sau đó được nối với phần tử đầu tiên `points[0]` để tạo thành một vòng đa giác. Điểm cuối cùng của danh sách cũng là điểm đầu tiên của vòng đa giác, tạo thành một vòng tròn đóng.
- **dash\_array="4":** Đây là một đối số của Polyline và xác định kiểu đường vẽ. Trong trường hợp này, `dash_array="4"` chỉ định một đường nét đứt với các đoạn có độ dài 4 pixels.
- **color="red":** Đây là một đối số của Polyline và xác định màu sắc của đường đa giác. Trong trường hợp này, `color="red"` đặt màu sắc của đường đa giác là màu đỏ.
- **weight=2:** Đây là một đối số của Polyline và xác định độ dày của đường đa giác. Trong trường hợp này, `weight=2` đặt độ dày của đường đa giác là 2 pixels.
- **fill=False:** Đây là một đối số của Polyline và xác định xem đường đa giác có được tô màu hay không. Trong trường hợp này, `fill=False` đặt đường đa giác không được tô màu.

Chương trình tìm đường đi trên bản đồ, sử dụng thuật toán tìm đường BFS

```
from collections import deque

def dis(point1, point2):
    return (point1[0] - point2[0])**2 + (point1[1] - point2[1])**2

def nearest_point(points, target):
    return min(points, key=lambda point: dis(point, target))

def path(start_point, end_point):
    # Tìm điểm gần nhất với điểm xuất phát
    start = nearest_point(points, start_point)
    # Tìm điểm gần nhất với điểm đích
    end = nearest_point(points, end_point)

    # Khởi tạo hàng đợi
    queue = deque()
    # Thêm điểm xuất phát vào hàng đợi
    queue.append((start, [start]))

    while queue:
        current, path = queue.popleft()
        if current == end:
            # Trả về đường đi nếu tìm thấy
            return [start_point] + path + [end_point]

        # Tìm điểm có cạnh với điểm hiện tại mà chưa được thăm
        edges_not_visited = [
            adjacent for adjacent in edges[hash(current)] if adjacent not in path]

        for point in edges_not_visited:
            new_path = path + [point]
            queue.append((point, new_path))

    # Không tìm thấy đường đi
    return None
```

Hình 6: Tìm kiếm đường đi

Đoạn code tìm kiếm đường đi nhóm đã triển khai dựa trên thuật toán BFS

```
Khởi tạo các biến của chương trình
start_maker là điểm xuất phát end_maker là điểm kết thúc

[ ] path_wrapper = LayerGroup()
map.add_layer(path_wrapper)

marker_wrapper = LayerGroup()
map.add_layer(marker_wrapper)

red_icon = Icon(icon_url='https://cdn.rawgit.com/pointhi/leaflet-color-markers/master/img/marker-icon-2x-red.png',
               icon_size=[25, 41], icon_anchor=[12, 41])
green_icon = Icon(icon_url='https://cdn.rawgit.com/pointhi/leaflet-color-markers/master/img/marker-icon-2x-green.png',
                  icon_size=[25, 41], icon_anchor=[12, 41])
start_marker = Marker(location=points[0],
                      draggable=True, title="A", icon = red_icon)
end_marker = Marker(location=points[13],
                    draggable=True, title="B", icon = green_icon)
marker_wrapper.add_layer(start_marker)
marker_wrapper.add_layer(end_marker)
```

Hình 7: Khởi tạo biến của chương trình

Khởi tạo các thành phần của bản đồ như `path_wrapper`, `map.add_layer`, `start_marker`, `end_marker`

```
[ ] def draw_path(start_location, end_location):
    # Xóa đường đi đang hiện
    path_wrapper.clear()
    try:
        # Tạo đường dẫn mới dựa trên vị trí bắt đầu và kết thúc
        path_wrapper.add_layer(AntPath(locations=paths(
            start_location, end_location), fill=False))
    except TraitError:
        print("No route found!")

def handle_move_start_marker(**kwargs):
    # Vẽ lại lớp đường dẫn bất cứ khi nào chúng ta di chuyển điểm đánh dấu bắt đầu
    draw_path(kwargs["location"], end_marker.location)

def handle_move_end_marker(**kwargs):
    # Vẽ lại lớp đường dẫn bất cứ khi nào chúng ta di chuyển điểm đánh dấu kết thúc
    draw_path(start_marker.location, kwargs["location"])

def handle_onclick_marker(**kwargs):
    for layer in marker_wrapper.layers:
        if layer.location == kwargs["coordinates"]:
            if layer.title == "A":
                marker_wrapper.clear()
            else:
                marker_wrapper.remove(layer)
    path_wrapper.clear()

def handle_map_interaction(**kwargs):
    if kwargs.get('type') == 'click':
        location = kwargs.get('coordinates')
        if len(marker_wrapper.layers) == 0:
            global start_marker
            start_marker = Marker(location=location,
                                   draggable=True, title="A")
            start_marker.on_move(handle_move_start_marker)
            start_marker.on_click(handle_onclick_marker)
            marker_wrapper.add_layer(start_marker)
        elif len(marker_wrapper.layers) == 1:
            end_marker = Marker(location=location,
                                draggable=True, title="B")
            end_marker.on_move(handle_move_end_marker)
            end_marker.on_click(handle_onclick_marker)
            marker_wrapper.add_layer(end_marker)
            draw_path(start_marker.location, end_marker.location)

[ ] start_marker.on_move(handle_move_start_marker)
start_marker.on_click(handle_onclick_marker)
end_marker.on_move(handle_move_end_marker)
end_marker.on_click(handle_onclick_marker)
map.on_interaction(handle_map_interaction)

# Vẽ đường dẫn ban đầu giữa các điểm đánh dấu bắt đầu và kết thúc
draw_path(start_marker.location, end_marker.location)

# Hiển thị bản đồ
display(map)
```

Hình 8: Thao tác với bản đồ

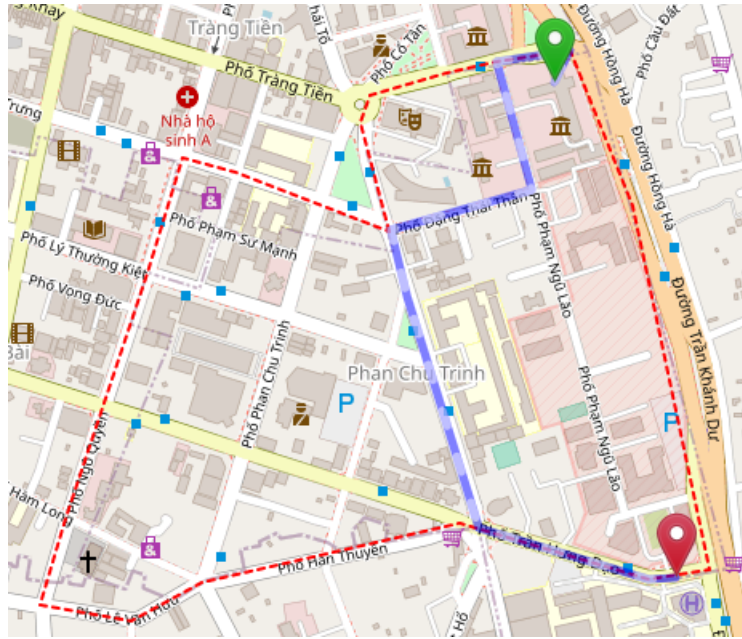
Điều khiển các thành phần tương tác trên bản đồ











Hình 14: Kết quả số 6

## Lời cảm ơn

Trước tiên với tình cảm sâu sắc và chân thành nhất, cho phép chúng em được bày tỏ lòng biết ơn đến tất cả các cá nhân và tổ chức đã tạo điều kiện hỗ trợ, giúp đỡ chúng em trong suốt quá trình học tập và hoàn thiện bài tập lớn này.

Với lòng biết ơn sâu sắc nhất, chúng em xin gửi đến thầy Trần Đình Khang đã truyền đạt vốn kiến thức quý báu cho chúng em trong suốt thời gian học tập tại trường. Nhờ có những lời hướng dẫn, dạy bảo của các thầy cô nên bài tập lớn của chúng em mới có thể hoàn thiện tốt đẹp.

Một lần nữa, chúng em xin chân thành cảm ơn thầy – người đã trực tiếp giúp đỡ, quan tâm, hướng dẫn chúng em hoàn thành tốt bài báo cáo này trong thời gian qua.

Bước đầu đi vào thực tế của chúng em còn hạn chế và còn nhiều bỡ ngỡ nên không tránh khỏi những thiếu sót, chúng em rất mong nhận được những ý kiến đóng góp quý báu của thầy để kiến thức của chúng em trong lĩnh vực này được hoàn thiện hơn đồng thời có điều kiện bổ sung, nâng cao ý thức của mình.