**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**---------------o0o---------------**



**BÁO CÁO BÀI TẬP LỚN**

**TÌM ĐƯỜNG ĐI TRONG LƯỢC ĐỒ BẰNG CÁC GIẢI THUẬT TÌM KIẾM**

**HỌC PHẦN: IT3160 – NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

**Giảng viên hướng dẫn: PSG.TS TRẦN ĐÌNH KHANG**

**Mã lớp học: 157487**

**Sinh viên thực hiện : Nhóm 10**

|  |  |  |
| --- | --- | --- |
| **Đào Trọng Nguyên** | **:** | **20235390** |
| **Trần Ngọc Huyền** | **:** | **20235347** |
| **Nguyễn Đào Nam Hải** | **:** | **20235321** |
| **Lưu Quốc Dũng** | **:** | **20235302** |
| **Đặng Phương Uyên** | **:** | **20235456** |

**HÀ NỘI, THÁNG 5 NĂM 2025**

**Phân công công việc**

|  |  |  |  |
| --- | --- | --- | --- |
| **Họ và tên** | **MSSV** | **Công việc thực hiện** | **Đánh giá** |
| Đào Trọng Nguyên | 20235390 | Thiết kế và thực thi giao diện, thiết kế và thực thi lớp đồ thị và thuật toán | 10 |
| Trần Ngọc Huyền | 20235347 | Thiết kế và thực thi giao diện, thiết kế và thực thi lớp vật cản và vùng cấm | 10 |
| Nguyễn Đào Nam Hải | 20235321 | Nghiên cứu triển khai thuật toán UCS, Dijkstra, Bidirectional A\* và Bellman Ford | 10 |
| Lưu Quốc Dũng | 20235302 | Nghiên cứu triển khai thuật toán DFS, BFS, Greedy và A\* | 10 |
| Đặng Phương Uyên | 20235456 | Thu thập dữ liệu bản đồ, xủ lí dữ liệu bản đồ, tối ưu hóa mã nguồn, kiểm thử | 10 |

**Mục lục**

[I. Giới thiệu đề tài bài toán, các chế độ thực hiện và cộng cụ cài đặt 4](#_Toc198340250)

[II. Biểu diễn không gian bài toán 6](#_Toc198340251)

[III. Các phương pháp tìm kiếm lời giải 7](#_Toc198340252)

[1. Thuật toán DFS và BFS 7](#_Toc198340253)

[DFS: 7](#_Toc198340254)

[BFS 8](#_Toc198340255)

[2. Thuật toán Greedy Best-First-Search và A\* 9](#_Toc198340256)

[GREEDY: 9](#_Toc198340257)

[A\* 10](#_Toc198340258)

[3. Thuật toán UCS và Dijkstra 12](#_Toc198340259)

[UCS 12](#_Toc198340260)

[DIJKSTRA 13](#_Toc198340261)

[4. Thuật toán Bidirectional A\* và Bellman Ford 15](#_Toc198340262)

[BIDIRECTIONAL A\*: 15](#_Toc198340263)

[BELLMAN FORD: 17](#_Toc198340264)

[5. So sánh các giải thuật tìm kiếm và đánh giá 19](#_Toc198340265)

[IV. Cài đặt chương trình 20](#_Toc198340266)

[1. Xử lí dữ liệu và tạo bản đồ 20](#_Toc198340267)

[2. Cài đặt các thuật toán 23](#_Toc198340268)

[3. Quản lí vật cản và vùng cấm 23](#_Toc198340269)

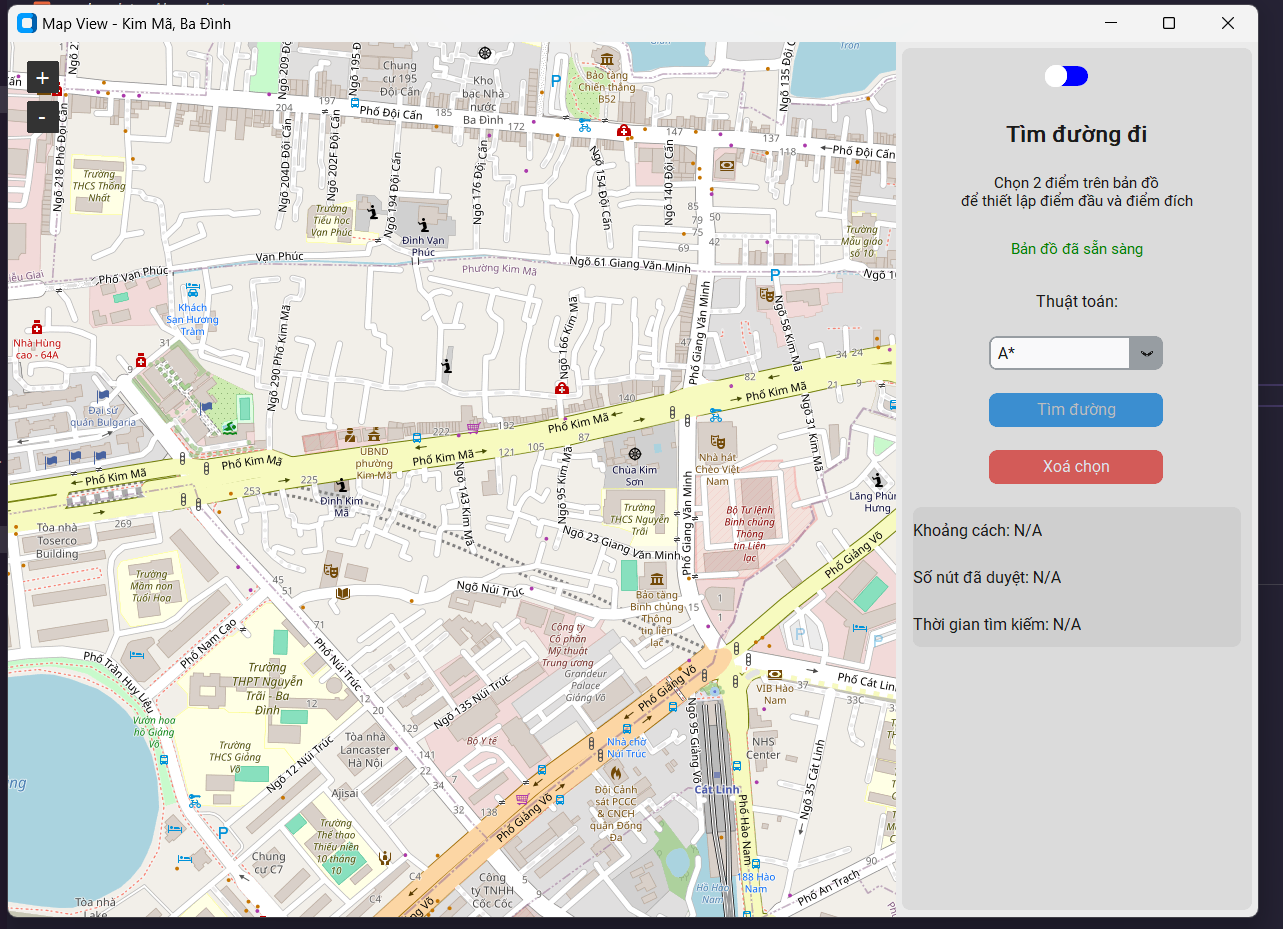
[4. Xử lí giao diện người dùng 28](#_Toc198340270)

[V. Kết quả đánh giá và cải thiện 29](#_Toc198340271)

# Giới thiệu đề tài bài toán, các chế độ thực hiện và cộng cụ cài đặt

Đề tài tập chung xây dựng một ứng dụng trực quan cho bài toán tìm đường đi ngắn nhất trên bản đồ thực tế, cụ thể là khu Kim Mã – Ba Đình, sử dụng dữ liệu từ OpenStreetMap. Cho phép người dùng tương tác trực tiếp với bản đồ để chọn điểm xuất phát và điểm đích, sau đó sử dụng các thuật toán để tìm đường đi tối ưu.

Ứng dụng hỗ trợ 2 chế độ sử dụng chính:



Chế độ User:

A screenshot of a phone

AI-generated content may be incorrect.

Cho phép người dùng chọn điểm đầu và điểm cuối trên bản đồ.

Lựa chọn thuật toán tìm đường trong danh sách: A\*, Dijkstra, UCS, DFS, BFS, Greedy, Bidirectional A\* và Bellman Ford.

Hiển thị kết quả đường đi, khoảng cách, thời gian thực thi và số nút đã duyệt.

---------------------------------------------------------------------------------------------------

Chế độ Admin:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, biểu tượng

Nội dung do AI tạo ra có thể không chính xác.

Cho phép thêm và xóa vật cản tại vị trí cụ thể.

Hỗ trợ vẽ vùng cấm trên bản đồ (khu vực không thể đi qua).

Hệ thống tự cập nhật lại đường đi sau thay đổi để phản ánh các ràng buộc mới.

Công cụ cài đặt:

Ngôn ngữ lập trình chính: Python

Thư viện: Tkintermapview để xây dựng giao diện

Công cụ thiết kế giao diện: customtkinter

Cài đặt:

Python: Đảm bảo đã cài đặt Python phiên bản 3.10 trở lên

Pip : Xác nhận rằng trình quản lí gói ‘pip’ đã được cài đặt

Các bước cài đặt:

Sau khi đã có đủ điều kiện ở cài đặt hãy thực hiện các bước sau:

Clone mã nguồn dự án : [*nguyen10t2/in**tro\_ai*](https://github.com/nguyen10t2/Intro_AI)

Chạy file : *run.bat*

# Biểu diễn không gian bài toán

Không gian trạng thái của bài toán:

Không gian bài toán là tập hợp các vị trí (node) trong bản đồ thực tế (Kim Mã, Ba Đình) được biểu diễn dưới dạng đồ thị có hướng có trọng số. Mỗi node là một vị trí địa lý có tọa độ (lat, lon), còn cạnh (edge) biểu diễn đường đi giữa các vị trí này với chi phí thực tế.

Biểu diện đồ thị:

Mỗi node: là một vị trí vật lý, có *ID, latitude, longitude (ID, lat, lon)*

Mỗi cạnh: kết nối giữa hai node có chi phí *(node1, node2, cost)*

Dữ liệu được lấy từ file *‘.osm’* (OpenStreetMap) được xử lý và đưa vào Graph



------------------------------------------------------------------------------------------------------------

Trạng thái trong bài toán tìm đường:

Một trạng thái bao gồm:

-Vị trí hiện tại *<curr node > (ID, lat\_x, lon\_x)*

-Trạng thái điểm đích *<goal node> (ID, lat\_n, lon\_n)*

-Danh sách các node đã duyệt: Closed Set

-Danh sách các node kế cần duyệt: Open Set

-Tổng chi phí từ đầu đến hiện tại: G-Score

-Với 1 số thuật toán yêu cầu thêm ước lượng khoảng cách tới đích: *F-Score*

Tập biến đổi trạng thái là toàn bộ những node hàng xóm (neighbors) không nằm trong tập bị chặn (obstacles) cụ thể:



# Các phương pháp tìm kiếm lời giải

## Thuật toán DFS và BFS

### DFS:

Tập thuộc tính:

A computer screen shot of a program

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Biến / Thành phần** | **Giải thích ngắn gọn** |
| open\_set = [] | Stack (danh sách LIFO) dùng để lưu các node sẽ duyệt tiếp |
| current = open\_set.pop() | Lấy node hiện tại ra (theo LIFO – duyệt sâu trước) |
| closed = set() | Lưu các node đã duyệt để tránh lặp lại |
| came\_from | Dictionary để truy vết đường đi từ start đến goal |

Nguyên lý hoạt động:

**DFS** là thuật toán *tìm kiếm theo chiều sâu*. Nó bắt đầu từ điểm xuất phát, đi theo 1 nhánh sâu nhất có thể, sau đó *quay lui* (backtrack) khi gặp ngõ cụt.

**DFS** không đảm bảo tìm được đường đi ngắn nhất, nhưng *rất nhanh* với đồ thị không có trọng số và không cần tối ưu hóa chi phí.

A computer screen shot of code

AI-generated content may be incorrect.

Độ phức tạp:

|  |  |
| --- | --- |
| **Loại độ phức tạp** | **Đánh giá** |
| Thời gian (Time) | O(V+E) với V: số đỉnh, E: số cạnh |
| Không gian (Space) | *O(V)* do sử dụng closed, came\_from, open\_set |

------------------------------------------------------------------------------------------------------------------

### BFS

Tập thuộc tính:

A computer screen shot of a program

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Biến / Thành phần** | **Giải thích ngắn gọn** |
| open\_set = Queue() | Hàng đợi (FIFO) – duyệt node theo thứ tự vào |
| current = open\_set.get() | Lấy node đầu tiên trong hàng đợi để duyệt |
| closed = set() | Tập hợp các node đã duyệt |
| came\_from | Dictionary lưu vết đường đi từ start đến goal |

Nguyên lý hoạt động:

**BFS** là thuật toán tìm kiếm *theo chiều rộng*. Nó duyệt từng lớp (level) của đồ thị, lần lượt mở rộng các node kề trước khi đi sâu hơn.

Vì **BFS** duyệt theo *số bước ít nhất*, nên luôn tìm ra *đường đi ngắn nhất* về *số lượng bước* (với đồ thị không có trọng số).

A computer screen shot of a code

AI-generated content may be incorrect.

Độ phức tạp

|  |  |
| --- | --- |
| **Loại độ phức tạp** | **Đánh giá** |
| Thời gian (Time) | *O(V+E)* |
| Không gian (Space) | *O(V)* do lưu open\_set, closed, came\_from |

------------------------------------------------------------------------------------------------------------------

## Thuật toán Greedy Best-First-Search và A\*

### GREEDY:

Tập thuộc tính:

A screen shot of a computer program

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Biến / Thành phần** | **Giải thích** |
| heuristic(n, goal) | Hàm ước lượng chi phí từ node n đến đích goal |
| open\_set = PriorityQueue() | Hàng đợi ưu tiên theo heuristic – node nào có ước lượng gần đích hơn sẽ được duyệt trước |
| open\_set.put(heuristic(neighbor, goal), neighbor) | Thêm node vào hàng đợi với ưu tiên là giá trị heuristic |
| closed = set() | Lưu các node đã duyệt để tránh lặp lại |
| came\_from | Dictionary để truy vết đường đi |

Nguyên lý hoạt động:

**Greedy** là thuật toán tìm đường đi ngắn nhất dựa trên *ước lượng chi phí* giữa node hiện tại và node đích.

Nó luôn chọn node có giá trị *heuristic nhỏ nhất* để mở rộng, giống như *"tham lam"* lao về đích càng nhanh càng tốt.

Tuy nhiên, **Greedy** không đảm bảo tìm ra đường đi ngắn nhất, vì nó không xét đến chi phí đã đi qua.

A computer screen shot of text

AI-generated content may be incorrect.

Độ phức tạp:

|  |  |
| --- | --- |
| **Loại độ phức tạp** | **Đánh giá** |
| Thời gian (Time) | *O(b^m)* – b là số node con trung bình của 1 node, m là độ sâu tối đa không gian tìm kiếm. |
| Không gian (Space) | *O(b^m)* – do lưu closed, came\_from và open\_set |

------------------------------------------------------------------------------------------------------------------

### A\*

Tập thuộc tính:

A computer screen shot of a program code

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Biến / Thành phần** | **Giải thích ngắn gọn** |
| open\_queue = PriorityQueue() | Hàng đợi ưu tiên theo f\_score – node có tổng chi phí thấp nhất sẽ được duyệt trước |
| open\_set.put(heuristic(neighbor, goal) + g\_score, neighbor) | Thêm node vào hàng đợi với ưu tiên là giá trị g\_score + heuristic |
| f\_score | Tổng chi phí ước lượng: g + h |
| g\_score | Chi phí thực từ start đến node |
| open\_set | Set lưu các node đang nằm trong hàng đợi giúp kiểm tra nhanh |

Nguyên lý hoạt động

**A\*** tìm đường đi có *tổng chi phí thấp nhất*, và bảo đảm tìm được *đường đi ngắn nhất nếu heuristic là admissible*.

-g(n): Chi phí thực từ điểm bắt đầu đến node hiện tại

-h(n): Hàm heuristic ước lượng chi phí từ node hiện tại đến đích

-Tổng lại thành: **f\_score(n) = g\_score(n) + heurictic(n)**

A computer screen shot of text

AI-generated content may be incorrect.

Độ phức tạp:

|  |  |
| --- | --- |
| **Loại độ phức tạp** | **Đánh giá** |
| Thời gian (Time) | *O(b^d)* – b là số node con trung bình của 1 node, d là độ sâu của lời giải ngắn nhất |
| Không gian (Space) | *O(b^d)* – lưu came\_from, g\_score, f\_score, open\_set |

## Thuật toán UCS và Dijkstra

### UCS

Tập thuộc tính:

A screenshot of a computer code

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Biến / Thành phần** | **Giải thích** |
| heuristic(n, goal) | Hàm ước lượng chi phí từ node n đến đích goal |
| open\_set = PriorityQueue() | Hàng đợi ưu tiên theo heuristic – node nào có ước lượng gần đích hơn sẽ được duyệt trước |
| open\_set.put(g\_score, neighbor) | Thêm node vào hàng đợi với ưu tiên là giá trị heuristic |
| g\_score | Chi phí thực tế từ start đến node |
| closed = set() | Lưu các node đã duyệt để tránh lặp lại |

Nguyên lý hoạt động:

**UCS** luôn ưu tiên đường đi sao cho *tổng chi phí từ start đến node kế tiếp là nhỏ nhất* khác với Greedy là ước lượng chi phí tương lai hay A\* là tổng cả hai.

**UCS** đảm bảo tìm ra đường đi có tổng chi phí nhỏ nhất nếu tồn tại đường đi và không bị chướng ngại vật cản.

A computer screen shot of text

AI-generated content may be incorrect.

Độ phức tạp:

|  |  |
| --- | --- |
| **Loại độ phức tạp** | **Đánh giá** |
| Thời gian (Time) | *O(b^(C/e))* trong đó b là số node con trung bình, C là tổng chi phí từ start đến goal, e là min(cost(edge)) |
| Không gian (Space) | *O(b^(C/e))* – do lưu came\_from, g\_score, open\_set |

------------------------------------------------------------------------------------------------------------------

### DIJKSTRA

Tập thuộc tính:

A computer code on a dark background

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Biến / Thành phần** | **Giải thích** |
| open\_queue = PriorityQueue() | Hàng đợi ưu tiên lưu các node cần xét, ưu tiên theo chi phí g\_score nhỏ nhất |
| open\_queue.put((0, start)) | Khởi tạo hàng đợi với start và chi phí ban đầu là 0 |
| open\_set | Tập các node hiện đang nằm trong hàng đợi, dùng để tránh thêm trùng lặp |
| g\_score | Dictionary lưu chi phí nhỏ nhất từ start đến từng node; khởi tạo là vô cực với tất cả node |

Nguyên lý hoạt động:

**Dijkstra** là thuật toán tìm đường đi ngắn nhất dựa trên chi phí thực tế đã đi từ điểm bắt đầu đến node hiện tại.

Nhìn ban đầu bạn sẽ thấy **Dijkstra** trông có vẻ chả khác gì **UCS** mấy khi đều tính toán trên g\_score nhưng trong khi **UCS** chỉ quan tâm là đường đi tối ưu đến goal mà không quan tâm trung gian thì **Dijkstra** sẽ tiếp tục duyệt đồ thị và đảm bảo rằng toàn bộ trung gian là đường đi tối ưu.

**Dijkstra** đảm bảo tìm ra đường đi có tổng chi phí nhỏ nhất nếu tồn tại đường đi và không bị vật cản cản trở.

A computer screen shot of text

AI-generated content may be incorrect.

Độ phức tạp:

|  |  |
| --- | --- |
| **Loại độ phức tạp** | **Đánh giá** |
| Thời gian (Time) | *O((V+E)logV)* E là số lượng cạnh và V là số lượng đỉnh |
| Không gian (Space) | *O(V)* – do lưu came\_from, g\_score, open\_queue, open\_set |

------------------------------------------------------------------------------------------------------------------

## Thuật toán Bidirectional A\* và Bellman Ford

### BIDIRECTIONAL A\*:

Tập thuộc tính:

A computer screen shot of a program

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Biến / Thành phần** | **Giải thích** |
| heuristic | Hàm đánh giá chi phí ước lượng từ node hiện tại đến đích (heuristic function) |
| open\_start = PriorityQueue() | Hàng đợi ưu tiên lưu các node chờ xét từ phía start, ưu tiên theo chi phí f\_score nhỏ nhất |
| open\_goal = PriorityQueue() | |  | | --- | |  |  |  | | --- | | Hàng đợi ưu tiên lưu các node chờ xét từ phía goal, ưu tiên theo chi phí f\_score nhỏ nhất | |
| open\_set\_start | Tập các node đang chờ xét ở phía start |
| open\_set\_goal | Tập các node đang chờ xét ở phía goal |
| closed\_start | Tập các node đã xét ở phía start |
| closed\_goal | Tập các node đã xét ở phía goal |
| g\_start | |  | | --- | |  |  |  | | --- | | Dictionary lưu chi phí nhỏ nhất từ start đến từng node phía start | |
| g\_goal | Dictionary lưu chi phí nhỏ nhất từ goal đến từng node phía goal |
| f\_start | Dictionary lưu chi phí ước lượng tổng (g + h) từ start đến node |
| f\_goal | Dictionary lưu chi phí ước lượng tổng (g + h) từ goal đến node |
| best\_node | Node giao nhau giữa hai phía tìm kiếm có chi phí tổng nhỏ nhất |
| best\_path\_cost | Chi phí đường đi nhỏ nhất tìm được đến thời điểm hiện tại |
| mu | Giới hạn chi phí để cắt tỉa quá trình tìm kiếm |

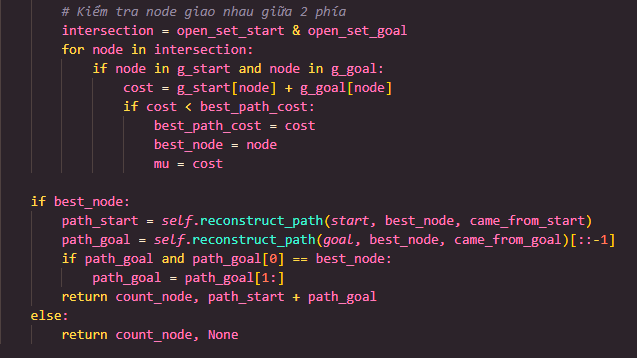
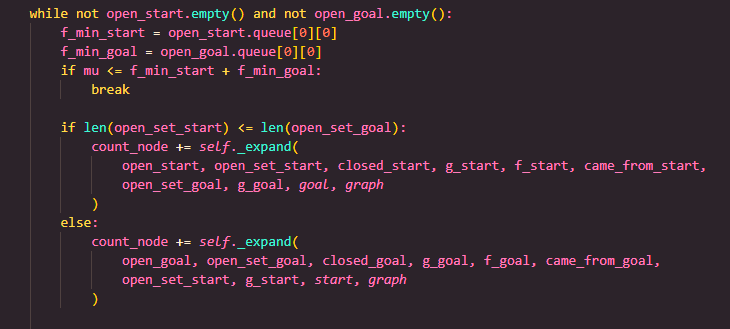
Nguyên lý hoạt động:

Thuật toán **Bidirectional** **A\*** là biến thể của **A\*** giúp tăng hiệu quả bằng cách tìm kiếm đồng thời từ hai phía: từ node bắt đầu (start) và node đích (goal).

Ở mỗi bước, thuật toán chọn mở rộng từ phía có số *node chờ xét ít hơn* nhằm cân bằng việc mở rộng không gian tìm kiếm.

Khi phát hiện một node xuất hiện trong cả hai hàng đợi (tức là điểm gặp giữa hai hướng) thì tổng chi phí: g\_start[node] + g\_goal[node] và cập nhật best\_path\_cost nếu nhỏ hơn giá trị tốt nhất hiện tại

Điều kiện dừng khi hai hàng đợi đều rỗng, hoặc best\_path\_cost ≤ f\_min\_start + f\_min\_goal, tức là không còn đường đi nào tốt hơn có thể được tìm thấy



Độ phức tạp:

|  |  |
| --- | --- |
| **Loại độ phức tạp** | **Đánh giá** |
| Thời gian (Time) | *O(b^(m/2))* – với b là số node con trung bình, m là độ sâu tối đa (do tìm kiếm hai chiều cùng lúc) |
| Không gian (Space) | *O(b^(m/2))* – lưu trữ các tập came\_from, g\_score, f\_score, open\_set và closed cho hai phía tìm kiếm |

------------------------------------------------------------------------------------------------------------------

### BELLMAN FORD:

Tập thuộc tính:

A computer screen shot of colorful text

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Biến / Thành phần** | **Giải thích** |
| dist | Dictionary lưu chi phí nhỏ nhất từ start đến từng node, khởi tạo vô cực với tất cả node |
| prev | Dictionary lưu node trước đó để truy vết đường đi |

Nguyên lý hoạt động:

Thực hiện tối đa |V|-1 vòng lặp, với mỗi cạnh (u, v, w), nếu node u hoặc v là chướng ngại vật thì bỏ qua. Nếu khoảng cách qua u nhỏ hơn khoảng cách hiện tại tới v, cập nhật khoảng cách và node trước đó.

Sau |V|-1 vòng lặp, thuật toán thực hiện một vòng lặp kiểm tra bổ sung để phát hiện chu trình âm. Nếu vẫn có thể giảm khoảng cách, thuật toán trả về None báo hiệu không tồn tại đường đi hợp lệ (do chu trình âm).

Nếu không có chu trình âm, thuật toán trả về số node đã xét cùng đường đi truy vết từ start đến goal.

A computer screen shot of text

AI-generated content may be incorrect.

Độ phức tạp:

|  |  |
| --- | --- |
| **Loại độ phức tạp** | **Đánh giá** |
| Thời gian (Time) | *O(V \* E)* – với V là số node, E là số cạnh trong đồ thị |
| Không gian (Space) | *O(V)* – lưu dist và prev cho tất cả các node |

## 5. So sánh các giải thuật tìm kiếm và đánh giá

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thuật toán | Tối ưu | Heuristic | Tốc độ | Số node duyệt |
| DFS | ❌ Không | ❌ Không | ⚡Rất nhanh | 🚫 Nhiều |
| BFS | ✅ Có | ❌ Không | ⚡Nhanh | 🚫 Rất nhiều |
| UCS | ✅ Có | ❌ Không | 🐢 Chậm | 🚫 Rất nhiều |
| Dijkstra | ✅ Có | ❌ Không | 🐢 Chậm | 🚫 Rất nhiều |
| Greedy | ❌ Không | ✅ Có | ⚡Nhanh | 🔄 Trung bình |
| A\* | ✅ Có | ✅ Có | ⚡Tốt | 🔄 Trung bình |
| Bidirectional A\* | ✅ Có | ✅ Có | ⚡⚡Rất nhanh | 🔄 Rất ít |
| Bellman-Ford | **✅** Có | ❌ Không | 🐢🐢 Rất chậm | 🚫 Rất lớn |

Các giải thuật không dùng tri thức bổ sung ở đây là ước lượng chi phí thưởng sẽ cho tốc độ trung bình và tập node phải duyệt cực kì lớn.

Còn lại thì thường cho tốc độ khá nhanh đường đi về chi phí cũng khá nhỏ, tập node phải duyệt cũng rất nhỏ.

Về A\* và Bidirectional A\* - BinA\* là 1 bản nâng cấp nhẹ của A\* khi sử dụng tìm kiếm từ 2 chiều thay vì 1 chiều. Nó giúp tối ưu hóa về mặt thời gian, không gian tìm kiếm nhưng đường đi chưa chắc đã tối ưu tốt nhất như A\* nhưng cũng rất là tốt. BinA\* hiệu quả hơn với đường đi xa, bản đồ lớn nhưng A\* vẫn làm rất tốt.

# Cài đặt chương trình

## Xử lí dữ liệu và tạo bản đồ

Xử lí dữ liệu:

Dữ liệu là một file xml có đuôi ‘.osm’ chứa đầy đủ thông tin liên quan đến các node, các cạnh, dạng node,... được lấy từ OpenStreetMap.

Một file xml có 55,000 dòng trong đó có khoảng xấp xỉ 15,000 node và 35,000 cạnh.

A screenshot of a computer screen

AI-generated content may be incorrect.

Đẻ phục vụ các giải thuật tìm kiếm có độ chính xác cao hơn nên đồ thj sẽ giữ lại gần như toàn bộ các node, các edge.

Sau đó chuyển hóa về class Graph cho trực quan hóa và dễ dàng thao tác với các thuật toán

A computer screen shot of a program code

AI-generated content may be incorrect.

Tạo bản đồ:

Đồ thị xậy dựng dưới dạng danh sách kề adj\_list và danh sách cạnh edges, có thêm tính năng hộc trợ thêm/ xóa vật cản và tính chi phí các cạnh

Mỗi nút được gán một node\_id và tọa độ (lat, lon) trong self.nodes.

Tập hợp các node\_id là vật cản được lưu trữ trong self.obstacles

Sử dụng các hàm heuristic (geopy.distance.geodesic), KDtree để tối ưu hóa việc tìm nút gần nhất trong không gian.

Class ‘**Graph**’:

A computer code with many colorful text

AI-generated content may be incorrect.

Phương thức a**dd\_node**: thêm đỉnh mới với tọa độ.

Phương thức **add\_edge**: thêm cạnh có hướng từ nút u đến nút v với chi phí.

A computer screen shot of text

AI-generated content may be incorrect.

Phương thức **neighbors**: hàm trả về danh sách hàng xóm không phải vật cản.

Phương thức **cost**: hoàn trả về chi phí giữa hai điểm.

A computer code with text

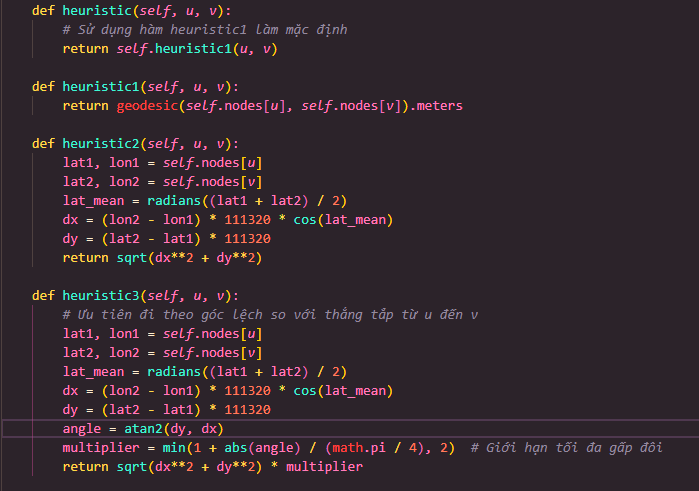
AI-generated content may be incorrect.

Phương **thức heuristic**: Hàm mặc định gọi là heuristic1 dùng trong App

Phương thức **heuristic1**: Sử dụng API geodesic để tìm khoảng cách đường chim bay.

Phương thức **heuristic2**: Coi trái đất là một mặt phẳng, tính toán khoảng cách xấp xỉ trên mặt phẳng.

Phương thức **heuristic3**: Thêm yếu tố góc lệch để ưu tiên đường di chuyển thẳng hơn.



Phương thức **find\_nearest\_nodes** và **find\_nearest\_node\_within\_radius**: Tìm điểm gần nhất theo tọa độ thực tế.

Phương thức **\_build\_kdtree**: Xây dựng KDtree từ danh sách tọa độ các nút, sử dụng thư viện API KDTree để hộ trợ tìm kiếm không gian hiệu quả.

A computer screen shot of code

AI-generated content may be incorrect.

## Cài đặt các thuật toán

Cài đặt:

A computer screen shot of a program

AI-generated content may be incorrect.

Các thuật toán được cài đặt trong module Algorithm.py, đều kế thừa từ một class Algorithm. Điều này giúp cho các thuật toán có tính thống nhất, dễ kiểm soát lỗi, dễ dàng mở rộng thêm các thành phần và các thuật toán mới.

## Quản lí vật cản và vùng cấm

Ý tưởng:

Các vật cản là các node bị loại bỏ trên graph, các node bị loại bỏ sẽ được thêm vào node\_list để đảm bảo không bị mất mát dữ liệu.

Khi bôi đen một vùng bất kỳ thì các node trong vùng đấy sẽ trở thành “vật cản”, từ đó ta thu được vùng cấm.

Class **‘Graph’**:

Phương thức **add\_obstacle**: Đánh dấu node là vật cản và loại bỏ cạnh liên quan.

A screenshot of a computer program

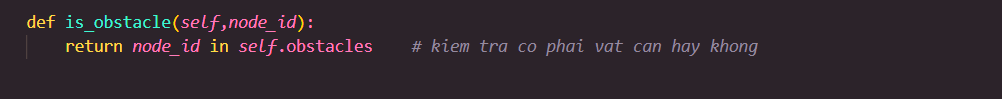
AI-generated content may be incorrect.

Phương thức remove**\_obstacle**: Xóa các vật cản và khôi phục lại các cạnh.

A computer screen shot of a program code

AI-generated content may be incorrect.

Phương thức **is\_obstacle**: kiểm tra node có phải vật cản hay không.



Class **‘App’**:

Phương thức **add\_**obstacle**:** Thêm một vật cản vào bản đồ với **coords-**tọa độ của vật cản. Kiểm tra xem người dùng đang ở User hay Admin. Chuyển đổi tọa độ canvas sang tọa độ địa lý. Nếu nút chưa có trong danh sách vật cản, thêm nó vào danh sách **obstacles**, cập nhật đồ thị và thêm biểu tượng vật cản trên bản đồ.

Ảnh có chứa văn bản, ảnh chụp màn hình

Nội dung do AI tạo ra có thể không chính xác.

Phương thức **remove\_last\_obstacle:** Kiểm tra xem có vật nào trong obstacle\_stack hay không. Nếu có xóa vật cản cuối cùng khỏi đồ thị

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

**On\_region\_draw\_start; on\_region\_draw\_motio; on\_region\_draw\_end**: là các phương thức vẽ vùng cấm trên bản đồ.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

Phương thức remove**\_last\_region:** Kiểm tra xem có vùng cấm nào trong region\_stack hay không. Nếu có xóa vùng cấm cuối cùng và các nút ra khỏi đồ thị.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

Class **‘ObstacleManager’**:

Phương thức **\_init\_**: khởi tạo cho Class. Nhận đối tượng ‘**app**’ để có thể truy nhập vào các **thành** phần khác của ứng dụng. Khởi tạo **region\_stacks** để lưu trữ các vùng cấm đã được thêm vào.

Ảnh có chứa Phông chữ, văn bản, ảnh chụp màn hình, Đồ họa

Nội dung do AI tạo ra có thể không chính xác.

Phương thức **\_add\_area\_marker:** Thêm biểu tượng vùng cấm tại tọa độ trung tâm, tải ảnh biểu tượng bằng PhotoImage, đặt marker lên bản đồ. Sau khi tạo vùng cấm, xóa mọi đường đi hiện có và nếu đã chọn điểm bắt đầu và kết thúc thì tự động chạy lại thuật toán tìm đường.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hoa tím

Nội dung do AI tạo ra có thể không chính xác.

Phương thức **\_process\_area:** Xử lý thêm vật cản vào vùng đã chọn: Tính toán phạm vi tọa độ (vĩ độ, kinh độ), tìm các nút trong đồ thị thuộc vùng và chưa là vật cản, thêm vào **region\_stacks**, gọi **add\_obstacles** để cập nhật vật cản, sau cùng gọi **\_add\_area\_marker** để hiển thị biểu tượng vùng cấm trên bản đồ.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, phần mềm

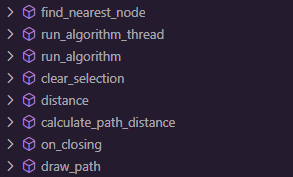
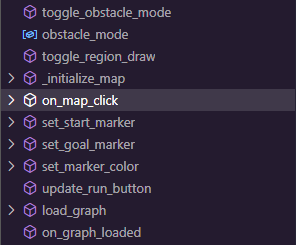
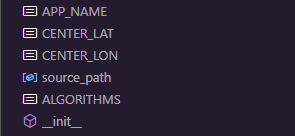
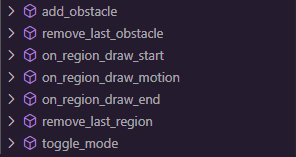
Nội dung do AI tạo ra có thể không chính xác.

Phương thức **add\_area\_obstacles\_async:** Thêm vật cản vào vùng xác định được bôi đen, (lat1, lon1) và (lat2, lon2) lần lượt là toạ độ góc trên bên trái và góc dưới bên phải. Tạo một luồng mới để gọi **process\_area** với các tọa độ đã cho.

Ảnh có chứa ảnh chụp màn hình, Phông chữ, hoa tím, màu tím

Nội dung do AI tạo ra có thể không chính xác.

## Xử lí giao diện người dùng



Một số thuộc tính và phương thức chính class ‘**App**’:

Tên ứng dụng: **APP\_NAME.**

Tọa độ trung tâm: **CENTER\_LAT, CENTER\_LON** định vị bản đồ khi khởi động

Đường dẫn tệp: **source\_path = r"res/KimMa.osm"**

Thuật Toán Tìm Đường: **ALGORITHMS** chứa các thuật toán: **A\*, Dijkstra, Greedy, BFS, DFS, Bidirectional A\*, UCS, và Bellman-Ford**

Phương thức **\_setup\_ui**: Tạo layout chính thiết lập các widget như nút chuyển chế độ, hướng dẫn, nút tìm đường, và các nút để thêm/xóa vật cản và vùng cấm.

Phương thức **\_initialize\_map**: Đặt vị trí và mức zoom, thêm các lệnh nhấp chuột phải để đặt điểm đầu và điểm đích, giữ sự kiện chuột để điều hướng bản đồ.

Phương thức **on\_map\_click**: Xử lý sự kiện nhấp chuột. Trong User, cho phép người dùng đặt điểm đầu và điểm đích. Trong Admin, cho phép thêm vật cản.

Phương **thức toggle\_mode**:  Chuyển đổi giữa User và Admin.

Phương thức **load\_graph, on\_graph\_loaded**: Chuyển đổi dữ liệu từ file ‘.osm’ về dạng Graph trong một thread riêng biệt với UI.

Phương thức **run\_algorithm, run\_algorithm\_thread**: Khởi động chạy giải thuật trong một thread riêng với UI.

Phương **thức find\_nearest\_node, distance**: Được thêm vào trong App dưới dạng method cache (nhằm tối ưu hóa cache).

Phương thức **calculate\_path\_distance, draw\_path**: Dùng để tính toán chi phí tổng và mô phỏng đường đi trên UI.

Và một số phương thức khác..v..v

# Kết quả đánh giá và cải thiện

Kết quả đánh giá:

Ứng dụng đã hoạt động hiệu quả trong việc tìm đường đi ngắn nhất trên bản đồ khu vực Kim Mã – Ba Đình – Hà Nội.

Các thuật toán được triển khai cho thấy khác biệt rõ rệt về hiệu suất:

**DFS, BFS:** Cho kết quả nhanh nhưng không tối ưu về mặt chi phi, hiệu quả. BFS thường tốt hơn DFS trong môi trường thực tế có nhiều vật cản.

**Dijkstra, UCS:** Luôn tìm được đường đi tối ưu nhưng mất nhiều thời gian do duyệt toàn bộ đồ thị.

**A\*, Greedy:** Hiệu quả cao đặc biệt là A\* cân bằng giữa tốc độ và độ tối ưu. Greedy nhanh hơn nhưng chưa chắc đã tìm được đường đi tối ưu.

**Bidirectional A\*:** Hiệu quả cao với đường đi dài, tốc độ cả thiện đáng thể.

**Bellman Ford:** Chính xác, xử lí tốt đồ thị có cạnh âm, nhưng chậm hơn rất nhiều so với các thuật toán khác.

Các cải tiến đã thực hiện:

Sử dụng *thread* để tách thuật toán ra khỏi UI, tránh treo giao diện.

Áp dụng *KDTree* để tăng tốc độ tìm node gần nhất

Hiển thị *vùng cấm* và *vật cản* bằng marker trực quan.

Hệ thống *tự cập nhật đường đi* sau mỗi lần thay đổi trên bản đồ.

Hướng phát triển tiếp theo:

Tối ưu đồ thị cho bài toán lớn hơn

Lưu lại lịch sử đường đi để phục vụ phân tích sau (như Google đã làm với GoogleMap)

Hỗ trợ lưu/tải lại trạng thái vật cản và vùng cấm