TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG ĐẠI HỌC CNTT & TRUYỀN THÔNG

======*======**

BÁO CÁO THỰC NGHIỆM

HỌC PHẦN: TRÍ TUỆ NHÂN TẠO

CHỦ ĐỀ: TÌM HIỂU THUẬT TOÁN TÌM KIẾM HEURISTIC VÀ ỨNG DỤNG VÀO BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT

Giảng viên hướng dẫn:	Ths. Trần Thanh Huân
Nhóm - Lớp:	10-20242IT6094002
Nhóm sinh viên thực hiện:	Phan Vân Anh – 2023607666
	Lê Trung Khánh – 2023603578
	Trần Thị Trinh – 2023603566
	Đào Anh Tú – 2023605293

Hà Nội, Năm 2025

LỜI MỞ ĐẦU

Ngày nay, khi công nghệ thông tin phát triển vượt bậc kèm theo đó là sự ra đời và phát triển của trí tuệ nhân tạo (Artificial Intelligence) đã mang đến cho nhân loại nhiều lợi ích to lớn, phát triển toàn xã hội. Nhờ đó mà chúng em được tiếp cận với kiến thức nền móng của AI qua học phần trí tuệ nhân tạo và nhiệm vụ của chúng em khi kết thúc học phần là tìm hiểu về các thuật toán tìm kiếm Heuristic và ứng dụng vào bài toán tìm kiếm đường đi ngắn nhất.

Sau khoảng thời gian lựa chọn đề tài, nhóm 10 chúng em đã bắt tay vào nghiên cứu, các thành viên trong nhóm đã rất có trách nhiệm tìm hiểu, giúp đỡ lẫn nhau hoàn thành đề tài nghiên cứu.

Để hoàn thành bản báo cáo này, chúng em đã nhận được nhiều sự hướng dẫn và giúp đỡ từ phía thầy cô bộ môn, đặc biệt là thầy Trần Thanh Huân đã dành nhiều thời gian hướng dẫn nhiệt tình chúng em hiểu ra nhiều vấn đề và hoàn thành bản báo cáo này một cách tốt nhất.

Chúng em xin chân thành cảm ơn!

DANH MỤC HÌNH ẢNH

MỤC LỤC

CHƯƠNG 1. TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO

1.1 Khái niệm về Trí tuệ nhân tạo

Trong lĩnh vực Công nghệ thông tin, Trí tuệ nhân tạo (TTNT) cũng có thể hiểu là "thông minh nhân tạo", tức là sự thông minh của máy móc do con người tạo ra, đặc biệt tạo ra cho máy tính, robot, hay các máy móc có các thành phần tính toán điện tử. TTNT là một ngành mới, nhưng phát triển rất mạnh mẽ và đem lại nhiều kết quả to lớn. Mùa hè 1956, tại hội thảo ở Dartmouth John McCarthy đã đưa ra thuật ngữ trí tuệ nhân tạo (Artificial Intelligence - AI). Mốc thời gian này được xem là thời điểm ra đời thực sự của lĩnh vực nghiên cứu TTNT.

TTNT là một lĩnh vực nghiên cứu của khoa học máy tính và khoa học tính toán nói chung. Có nhiều quan điểm khác nhau về TTNT. Do đó có nhiều định nghĩa khác nhau về lĩnh vực này. Sau đây là một số định nghĩa:

"Sự nghiên cứu các năng lực trí tuệ thông qua việc sử dụng các mô hình tính toán" (Charniak và McDormott, 1985).

"Nghệ thuật tạo ra các máy thực hiện các chức năng đòi hỏi sự thông minh khi được thực hiện bởi con người" (Kurweil, 1990).

"Lĩnh vực nghiên cứu tìm cách giải thích và mô phỏng các hành vi thông minh trong thuật ngữ các quá trình tính toán" (Schalkoff, 1990).

"Sự nghiên cứu các tính toán để có thể nhận thức, lập luận và hành động" (Winston, 1992).

"Một nhánh của khoa học máy tính liên quan đến sự tự động hóa các hành vi thông minh" (Luger and Stubblefield, 1993).

1.2 Một số kỹ thuật cơ bản trong Trí tuệ nhân tạo

Có nhiều kỹ thuật nghiên cứu, phát triển ngành khoa học TTNT. Tuy vậy, các kỹ thuật TTNT thường khá phức tạp khi cài đặt cụ thể, lý do là các kỹ thuật này thiên về xử lý các ký hiệu tượng trưng và đòi hỏi phải sử dụng những tri thức chuyên môn thuộc nhiều lĩnh vực khác nhau. Do vậy, các kỹ thuật TTNT hướng tới khai thác những tri thức về lĩnh vực đang quan tâm được mã hoá trong máy sao cho đạt được mức độ tổng quát, dễ hiểu, dễ diễn đạt thông qua ngôn ngữ chuyên môn gần gũi với ngôn ngữ tự nhiên, dễ khai thác nhằm thu hẹp các khả năng cần xét để đi tới lời giải cuối cùng.

Các kỹ thuật Trí tuệ nhân tạo cơ bản bao gồm:

- Lý thuyết giải bài toán và suy diễn thông minh: Lý thuyết giải bài toán cho phép viết các chương trình giải câu đố, các trò chơi thông qua các suy luận mang tính người
- Lý thuyết tìm kiếm may rủi: Lý thuyết này bao gồm các phương pháp và kỹ thuật tìm kiếm với sự hỗ trợ của thông tin phụ để giải bài toán một cách có hiệu quả.
- Các ngôn ngữ về TTNT: Để xử lý các tri thức người ta không chỉ sử dụng các ngôn ngữ lập trình dùng cho các xử lý dữ liệu số, mà cần có ngôn ngữ khác. Các ngôn ngữ chuyên dụng này cho phép lưu trữ và xử lý thông tin ký hiệu. Một số ngôn ngữ được nhiều người biết đến là LISP, PROLOG, ...
- Lý thuyết thể hiện tri thức và hệ chuyên gia: Trí tuệ nhân tạo là khoa học về thể hiện và sử dụng tri thức. Mạng ngữ nghĩa, logic vị từ, Frame, … là các phương pháp biểu diễn tri thức thông dụng. Việc gắn liền cách thể hiện và sử dụng tri thức là cơ sở hình thành hệ chuyên gia.
- Lý thuyết nhận dạng và xử lý tiếng nói: Giai đoạn phát triển đầu của TTNT gắn với lý thuyết nhận dạng. Ứng dụng của phương pháp này trong việc nhận dạng chữ viết, âm thanh, …

- Người máy: Cuối những năm 70, người máy trong công nghiệp đã đạt được nhiều tiến bộ. Người máy có bộ phận cảm nhận và các cơ chế hoạt động được nối ghép theo sự điều khiển thông minh. Khoa học về cơ học và TTNT được tích hợp trong khoa học người máy.
- Tâm lý học xử lý thông tin: Các kết quả nghiên cứu của tâm lý học giúp Trí tuệ nhân tạo xây dựng các cơ chế trả lời theo hành vi, có ý thức; nó giúp cho việc thực hiện các suy diễn mang tính người.
- Ngoài ra, xử lý danh sách, kỹ thuật đệ quy, kỹ thuật quay lui và xử lý cú pháp hình thức là những kỹ thuật cơ bản của tin học truyền thống có liên quan trực tiếp đến TTNT.

1.3 Ưu và nhược điểm của Trí tuệ nhân tạo

Ưu điểm:

- Mạng lưới thần kinh nhân tạo và công nghệ trí tuệ nhân tạo có khả năng học tập giỏi.
- AI xử lý được số lượng lớn dữ liệu nhanh hơn nhiều và đưa ra các dự đoán chính xác hơn khả năng của con người.
- Một khối lượng dữ liệu khổng lồ được tạo ra hàng ngày sẽ gây khó khăn cho các nhà nghiên cứu, AI dùng học máy để lấy những dữ liệu này và biến nó trở thành thông tin có thể thực hiện được.

Nhược điểm:

- Việc sử dụng AI sẽ gây tốn kém nhiều vì phải xử lý một lượng lớn dữ liệu mà lập trình AI yêu cầu.
- Khả năng giải thích cũng là một trở ngại trong việc sử dụng AI trong các lĩnh vực khác nhau theo các yêu cầu phải tuân thủ quy định nghiêm ngặt.

Ví dụ: Các tổ chức tài chính, nếu từ chối cấp tín dụng được đưa ra bởi A thì khó có thể đưa ra giải thích rõ ràng, các lý do không cấp tín dụng cho khách hàng.

1.4 Ứng dụng của Trí tuệ nhân tạo

- AI trong lĩnh vực sức khỏe:

AI đóng vai trò lớn trong việc cải thiện sức khỏe của người bệnh, cũng như giảm chi phí tối đa. Một trong những công nghệ chăm sóc sức khỏe tốt nhất chính là IBM Watson, có thể hiểu được các ngôn ngữ tự nhiên và phản hồi các câu hỏi được yêu cầu. Hệ thống này khai thác dữ liệu của bệnh nhân và các nguồn dữ liệu sẵn có khác để tạo ra giả thuyết.

Tiếp theo nó sẽ trình bày một lược đồ điểm tin cậy. Ứng dụng khác của AI như: chatbot, chương trình máy tính trực tuyến để trả lời câu hỏi và hỗ trợ khách hàng, sắp xếp các cuộc hẹn hoặc trợ giúp bệnh nhân thông qua việc thanh toán và các trợ lý y tế ảo cung cấp phản hồi y tế cơ bản. - Sản sinh ngôn ngữ tự nhiên (Nature Language Generation, NLG) AI có thể tạo ra các văn bản từ những dữ liệu máy tính tự tổng hợp. Cũng tương tự như con người, Ai có thể tự soạn cho mình những tài liệu phù hợp với công việc được giao, các kế hoạch cũng như thực hiện các phương án tối ưu nhất với vốn từ được học hỏi từ chính con người.

- Nhận diện giọng nói:

Nó còn có thể chuyển lời nói của con người sang dạng mà các ứng dụng máy tính có thể đọc hiểu được. Điều này cũng tương tự như việc bạn có thể nói chuyện giao tiếp với máy móc giống như một con người thật sự.

- Quản trị viên ảo:

Từ chatbot đơn giản cho đến những hệ thống tiên tiến có thể kết nối được với con người, công nghệ này ngày càng được sử dụng trong dịch vụ khách hàng, hỗ trợ người dùng và quản lý nhà thông minh. Do vậy ngày càng có nhiều ứng dụng công nghệ trí tuệ nhân tạo AI.

- Nền tảng máy học (Machine Learning):

Machine Learning phát triển và huấn luyện, dữ liệu cũng như các công nghệ điện toán để thiết kế, huấn luyện và triển khai các mô hình máy vào các ứng dụng, tiến trình và máy móc từ đó sử dụng vào công việc đem lại kết quả tốt.

- Phần cứng tối ưu hóa AI:

Bao gồm các bộ xử lý GPU và các thiết bị được thiết kế để thực hiện các công việc của AI một cách hiệu quả nhất. Để xử lý các AI tốt thì bạn cần phải trang bị một bộ máy tính chuyên dụng cho ứng dụng của AI - Trí Tuệ Nhân Tạo (Deep Learning), đây là những bộ PC được xây dựng rất đặc biệt, có thể chạy song song nhiều card màn hình.

- Quản lý việc ra quyết định:

Đây chính là công nghệ đưa các quy tắc và logic vào trong hệ thống AI để dùng cho việc thiết lập hoặc huấn luyện ban đầu nhằm giúp chúng có khả năng duy trì và điều chỉnh liên tục. Nhờ vào AI mà việc ra quyết định cũng trở nên đơn giản và chính xác hơn rất nhiều.

- Nền tảng Deep Learning:

Cũng là một lĩnh vực đặc biệt trong máy học (machine learning), deep learning được biết đến là chương trình chạy trên một mạng thần kinh nhân tạo, có khả năng huấn luyện máy tính học một lượng lớn dữ liệu. Áp dụng nền tảng này sẽ nâng cao hiệu quả trong quá trình làm việc của mình.

- Sinh trắc học:

Công nghệ này còn cho phép tương tác tự nhiên hơn giữa con người và máy móc, bao gồm cả việc nhận diện hình ảnh, dấu vân tay, giọng nói và cử chỉ của con người. Một ứng dụng hữu ích và cần thiết được nhiều người quan tâm và tìm hiểu.

- Quy trình tự động hóa robot (Robotic Process Automation):

Sử dụng mã hóa và các phương pháp khác để tự động hóa hoạt động của con người bằng robot giúp hỗ trợ công việc hiệu quả hơn. Hiện tại trí tuệ nhân tạo và các cỗ máy Ai Machine đã và đang được các tập đoàn lớn ứng dụng, tại

Việt Nam có Viettel đang sử dụng để tạo ra con bot giọng cũng như nhiều ứng dụng tương lai khác. Sức mạnh của Trí tuệ nhân tạo - Ai là vô cùng quan trọng cho công nghiệp sau này và trong tương lai chúng sẽ trở thành công cụ chính phát triển vượt bậc.

- Big Data Analytics:

AI được sử dụng để xử lý số lượng dữ liệu cực lớn nhằm tìm ra thông tin hữu ích. Nó còn có lợi cho việc phân tích doanh thu tiềm năng của lợi nhuận hoặc tìm ra cách để tối ưu hóa các quy trình hiện có. Big Data Analytics không chỉ được tìm thấy thông tin chi tiết mà còn có thể dự đoán các yếu tố, ví dụ như: lợi nhuận, tổn thất, yêu cầu về kho bãi, kết nối chặng cuối... Lĩnh vực AI hiện đang được sử dụng trong toàn bộ khu vực doanh nghiệp do những lợi ích mà nó mang lại cho nhiều loại công ty. Đồng thời có một hệ sinh thái mạnh mẽ của các công ty cung cấp giải pháp một cửa cho Big Data Analytics.

1.5 Những thách thức trong Trí tuệ nhân tạo

Thách thức số 1: Công suất tính toán

- Một trong những thách thức đầu tiên trong hành trình phát triển trí tuệ nhân tạo chính là công suất tính toán. Các thuật toán đòi hỏi sự mạnh mẽ của các bộ nhớ và bộ xử lý đồng thời để hoạt động hiệu quả. Máy học và học sâu là những bước tiếp theo của trí tuệ nhân tạo và chúng yêu cầu một số lượng ngày càng tăng các lõi và GPU để hoạt động một cách hiệu quả.

Thách thức số 2: Bảo mật và riêng tư dữ liệu

- Một thách thức quan trọng khác mà trí tuệ nhân tạo đối mặt là vấn đề về bảo mật và riêng tư dữ liệu. Dữ liệu chính là yếu tố mà tất cả các mô hình học sâu và máy học dựa trên. Tuy nhiên, với sự gia tăng vượt bậc của lượng dữ liệu và các thuật toán phức tạp, việc bảo mật và bảo vệ dữ liệu trở nên khó khăn hơn. Trường hợp dữ liệu bị rò rỉ có thể gây hậu quả nghiêm trọng, đặc biệt trong lĩnh vực y tế hoặc tài chính

Thách thức số 3: Vấn đề giải thích

- Các thuật toán học sâu, một phần của trí tuệ nhân tạo, thường rất khó hiểu trong quá trình ra quyết định của chúng. Điều này làm cho việc giải thích trở nên khó khăn đối với con người. Vấn đề này gây khó khăn đặc biệt trong các lĩnh vực như chăm sóc sức khỏe và tài chính, nơi các quyết định của thuật toán có thể có hậu quả nghiêm trọng. Người nghiên cứu đang phát triển các kỹ thuật để làm cho các thuật toán trí tuệ nhân tạo trở nên minh bạch và dễ hiểu hơn

Thách thức số 4: Thiên vị dữ liệu

- Trí tuệ nhân tạo nhận diện và học từ các tập dữ liệu lớn. Tuy nhiên, các tập dữ liệu này thường không công bằng và có thiên vị. Nếu một thuật toán được đào tạo trên dữ liệu phản ánh sự thiên vị về giới tính hoặc chủng tộc, nó sẽ duy trì các thiên vị đó trong quá trình ra quyết định. Điều này có thể gây ra những hậu quả nghiêm trọng trong các lĩnh vực như nhân sự, tài chính và công lý. Việc xử lý thiên vị dữ liệu là một trong những thách thức quan trọng mà cộng đồng trí tuệ nhân tạo đang nỗ lực giải quyết.

Thách thức số 5: Thay thế công việc

- Một trong những thách thức đáng chú ý nhất của trí tuệ nhân tạo là khả năng thay thế công việc của con người. Với sự phát triển của các hệ thống trí tuệ nhân tạo, có nguy cơ rằng chúng sẽ thay thế người lao động trong một số công việc và ngành cụ thể, dẫn đến việc thất nghiệp và không ổn định xã hội. Tuy nhiên, đã có những nghiên cứu cho thấy rằng trí tuệ nhân tạo cũng có thể tạo ra cơ hội việc làm mới trong các lĩnh vực như phân tích dữ liệu và kỹ thuật phần mềm. Câu hỏi liên quan đến việc thay thế công việc đòi hỏi sự khảo sát và đưa ra giải pháp nhằm đảm bảo lợi ích của trí tuệ nhân tạo được chia sẻ một cách công bằng trong xã hội.

Thách thức số 6: Nguy cơ đối với nhân loại

- Trí tuệ nhân tạo đối mặt với một thách thức phổ biến là có thể trở thành một nguy hiểm đối với nhân loại nếu nó trở nên vượt ngoài sự kiểm soát của chúng

ta. Các tác phẩm khoa học viễn tưởng và sách mô tả AI như là một thực thể tự trị xấu xa muốn hủy diệt nhân loại, tuy nhiên những hình ảnh này không dựa trên các sự thật khoa học và không phản ánh đúng tình trạng hiện tại của công nghệ AI. Sự thật là các hệ thống trí tuệ nhân tạo chỉ nguy hiểm nếu chúng được thiết kế hoặc vận hành sai, không có khả năng tự nhận thức hoặc ra quyết định tự động. Chúng chỉ đơn giản là các ông cụ thực hiện nhiệm vụ dựa trên dữ liệu và thuật toán được cung cấp. Do đó, nếu một hệ thống trí tuệ nhân tạo gây hậu, đó là do nó đã được thiết kế hoặc được lập trình để làm như vậy, điều quan trọng là chúng ta đảm bảo rằng các hệ thống trí tuệ nhân tạo được phát triển và sử dụng một cách có trách nhiệm và đạo đức, đồng thời xem xét các hậu quả và rủi ro tiềm năng.

1.6 Các bước cơ bản trong xây dựng mô hình Trí tuệ nhân tạo

1.6.1 Xác định vấn đề

- Mục tiêu: Đầu tiên, cần xác định rõ mục tiêu của mô hình AI. Điều này bao gồm việc hiểu rõ vấn đề cần giải quyết và các yêu cầu cụ thể của nó.
- Phạm vi: Xác định phạm vi của dự án, bao gồm loại dữ liệu cần thiết và các tiêu chí đánh giá hiệu suất của mô hình.

1.6.2 Thu thập dữ liệu

- Nguồn dữ liệu:
    - Nguồn nội bộ: Dữ liệu từ các hệ thống nội bộ như cơ sở dữ liệu doanh nghiệp, nhật ký hoạt động, v.v.
    - Nguồn bên ngoài: Dữ liệu từ các API, trang web, nguồn mở, hoặc mua từ bên thứ ba.
- Chất lượng dữ liệu: Đảm bảo rằng dữ liệu thu thập được có chất lượng tốt, đầy đủ, và phù hợp với vấn đề cần giải quyết.

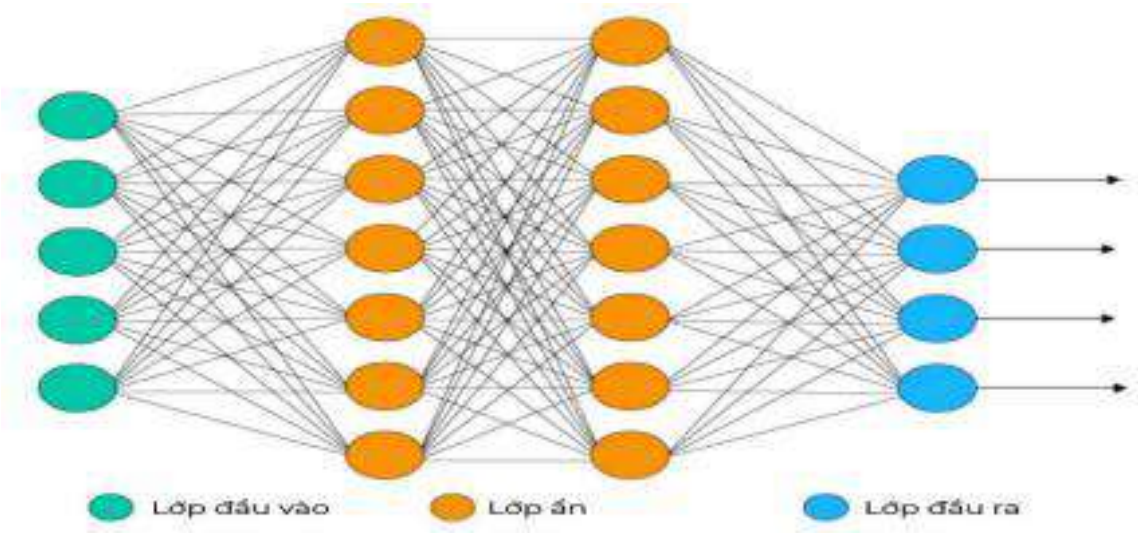
- Dữ liệu được chia thành 2 loại gồm dữ liệu có cấu trúc và dữ liệu không có cấu trúc: Dữ liệu có cấu trúc là loại dữ liệu tuân theo một định dạng cứng nhắc để đảm bảo tính nhất quán trong quá trình xử lý và cũng dễ dàng phân tích.
Ví dụ như hồ sơ khách hàng với họ, tên, ngày sinh, địa chỉ, v.v. o Dữ liệu phi cấu trúc là dữ liệu được duy trì theo mẫu không đồng nhất. Nó có thể bao gồm âm thanh, hình ảnh, hình ảnh, từ ngữ và đồ họa thông tin.
Ví dụ như email, cuộc trò chuyện qua điện thoại, tin nhắn WhatsApp, WeChat.

1.6.3 Tiền xử lý dữ liệu

- Dọn dẹp dữ liệu: Loại bỏ các giá trị thiếu, xử lý dữ liệu sai lệch hoặc lỗi thời.
- Chuyển đổi dữ liệu: Chuẩn hóa và mã hóa dữ liệu để phù hợp với mô hình. Điều này có thể bao gồm việc chuyển đổi các biến categorical thành biến số, chuẩn hóa dữ liệu, hoặc tạo ra các đặc trưng mới.

1.6.4 Chọn mô hình

- Loại mô hình: Chọn loại mô hình phù hợp với vấn đề cần giải quyết. Có thể là mô hình hồi quy, phân loại, clustering, mạng nơ-ron, hoặc các mô hình học sâu khác.



Hình 1: Mô hình mạng nơ-ron

- Thư viện và công cụ: Sử dụng các thư viện và công cụ AI/ML như TensorFlow, PyTorch, Scikit-learn, v.v.

Hình 2: Một số công cụ AI/ML

1.6.5 Chọn thuật toán

- Học có giám sát: Máy học từ dữ liệu có nhãn để dự đoán kết quả trên tập dữ liệu mới. Các thuật toán phổ biến gồm SVM, Hồi quy logistic, Rừng ngẫu nhiên, và Phân loại Bayes ngây thơ.
- Học không giám sát: Máy tự tìm cấu trúc từ dữ liệu không có nhãn. Bao gồm các phương pháp phân cụm (nhóm đối tượng), hiệp hội (tìm liên kết giữa đối tượng), và giảm kích thước (giảm số biến để loại bỏ nhiễu).

1.6.6 Huấn luyện mô hình

- Tập huấn luyện và kiểm tra: Chia dữ liệu thành tập huấn luyện và tập kiểm tra để đánh giá hiệu suất mô hình.
- Huấn luyện mô hình: Sử dụng tập huấn luyện để dạy mô hình bằng cách điều chỉnh các tham số của nó.

1.6.7 Đánh giá mô hình

- Kiểm tra hiệu suất: Sử dụng tập kiểm tra để đánh giá hiệu suất của mô hình thông qua các chỉ số như độ chính xác, độ nhạy, độ đặc hiệu, F1- score, v.v.

- Điều chỉnh mô hình: Tối ưu hóa và điều chỉnh các tham số của mô hình dựa trên kết quả đánh giá để cải thiện hiệu suất.

1.6.8 Triển khai mô hình

- Triển khai: Đưa mô hình vào hoạt động trong môi trường thực tế, có thể là một ứng dụng web, ứng dụng di động, hoặc một hệ thống khác.
- Theo dõi và bảo trì: Theo dõi hiệu suất của mô hình sau khi triển khai và thực hiện bảo trì định kỳ để đảm bảo mô hình vẫn hoạt động hiệu quả.

1.6.9 Cải thiện liên tục

- Thu thập phản hồi: Thu thập phản hồi từ người dùng và từ các hệ thống giám sát để hiểu rõ hơn về hiệu suất thực tế của mô hình.
- Cập nhật mô hình: Dựa trên phản hồi và dữ liệu mới, cập nhật và cải thiện mô hình để duy trì hoặc nâng cao hiệu suất.

Quá trình này có thể lặp lại nhiều lần khi dữ liệu mới xuất hiện hoặc khi có nhu cầu cải tiến mô hình.

CHƯƠNG 2. KHÔNG GIAN TRẠNG THÁI VÀ CÁC THUẬT TOÁN TÌM KIẾM

2.1 Không gian trạng thái

2.1.1 Mô tả trạng thái

Giải bài toán trong không gian trạng thái, trước hết phải xác định dạng mô tả trạng thái bài toán sao cho bài toán trở nên đơn giản hơn, phù hợp bản chất vật lý của bài toán (Có thể sử dụng các cấu trúc dữ liệu: véctơ, mảng hai chiều, cây, danh sách,...).

Mỗi trạng thái chính là mỗi hình trạng của bài toán, các tình trạng ban đầu và tình trạng cuối của bài toán gọi là trạng thái đầu và trạng thái cuối.

Ví dụ: Bài toán đong nước: Cho 2 bình có dung tích lần lượt là m và n (lit). Với nguồn nước không hạn chế, dùng 2 bình trên để đong k lít nước. Không mất tính tổng quát có thể giả thiết k <= min(m,n).

- Tại mỗi thời điểm xác định, lượng nước hiện có trong mỗi bình phản ánh bản chất hình trạng của bài toán ở thời điểm đó.
- Gọi x là lượng nước hiện có trong bình dung tích m và y là lượng nước hiện có trong bình dung tích n.
- Như vậy bộ có thứ tự (x,y) có thể xem là trạng thái của bài toán. Với cách mô tả như vậy, các trạng thái đặc biệt của bài toán sẽ là:

+ Trạng thái đầu: (0,0)

+ Trạng thái cuối: (x,k) hoặc (k,y)

2.1.2 Toán tử chuyển trạng thái

- Toán tử chuyển trạng thái thực chất là các phép biến đổi đưa từ trạng thái này sang trạng thái khác. Có hai cách dùng để biểu diễn các toán tử:

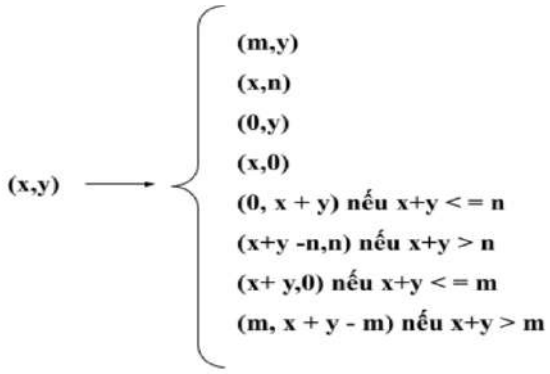
- Biểu diễn như một hàm xác định trên tập các trạng thái và nhận giá trị cũng trong tập này.
- Biểu diễn dưới dạng các quy tắc sản xuất S? A có nghĩa là nếu có trạng thái S thì có thể đưa đến trạng thái A.

Ví dụ. Bài toán đong nước

Các thao tác sử dụng để chuyển trạng thái này sang trạng thái khác gồm:

- Đổ đầy một bình
- Đổ hết nước trong một bình ra ngoài
- Đổ nước từ bình này sang bình khác.

Như vậy, nếu trạng thái đang xét là (x,y) thì các trạng thái kế tiếp có thể chuyển đến sẽ là:



2.1.3 Không gian trạng thái của bài toán

Không gian trạng thái là tập hợp tất cả các trạng thái của bài toán ứng với một cấu trúc biểu diễn nào đó

Không gian trạng thái là một bộ bốn, Ký hiệu: K= (T, S, G, F). Trong đó:

+ T: tập tất cả các trạng thái có thể có của bài toán.
+ S: trạng thái đầu.
+ G: tập các trạng thái đích.
+ F: tập các toán tử

Ví dụ 1. Không gian trạng thái của bài toán đong nước là bộ bốn T, S, G, F xác định như sau:

T = { (x,y) / 0 <= x <= m; 0 <= y <= n }

S = (0,0)

G = { (x,k) hoặc (k,y) / 0 <= x <= m; 0 <= y <= n}

F = Tập các thao tác đong đầy, đổ ra hoặc đổ sang bình khác thực hiện trên một bình

2.2 Các thuật toán tìm kiếm mù

2.2.1 Thuật toán tìm kiếm theo chiều sâu DFS (Depth First Search)

a. Tư tưởng của chiến lược tìm kiếm theo chiều sâu

- Từ đỉnh xuất phát duyệt một đỉnh kề.
- Các đỉnh của đồ thị được duyệt theo các nhánh đến nút lá.
- Nếu chưa tìm thấy đỉnh TG thì quay lui tới một đỉnh nào đó để sang nhánh khác.
- Việc tìm kiếm kết thúc khi tìm thấy đỉnh TG hoặc đã hết các đỉnh.

b. Thuật toán tìm kiếm theo chiều sâu

Lưu trữ: Sử dụng hai danh sách DONG và MO trong đó:

DONG: Chứa các đỉnh đã xét, hoạt động theo kiểu FIFO (hàng đợi).

MO: chứa các đỉnh đang xét , hoạt động theo kiểu LIFO (ngăn xếp).

MO = ∅; MO = MO ∪ $\{T_0\}$

while (MO != ∅){

n = get(MO) // lấy đỉnh đầu trong danh sach MO

if (n == T_G) // nếu n là trạng thái kết thúc

return TRUE // tìm kiếm thành công, dừng

DONG = DONG ∪ {n} //đánh dấu n đã được xét

for các đỉnh kề v của n

if (v chưa đc xét) //v chưa ở trong DONG

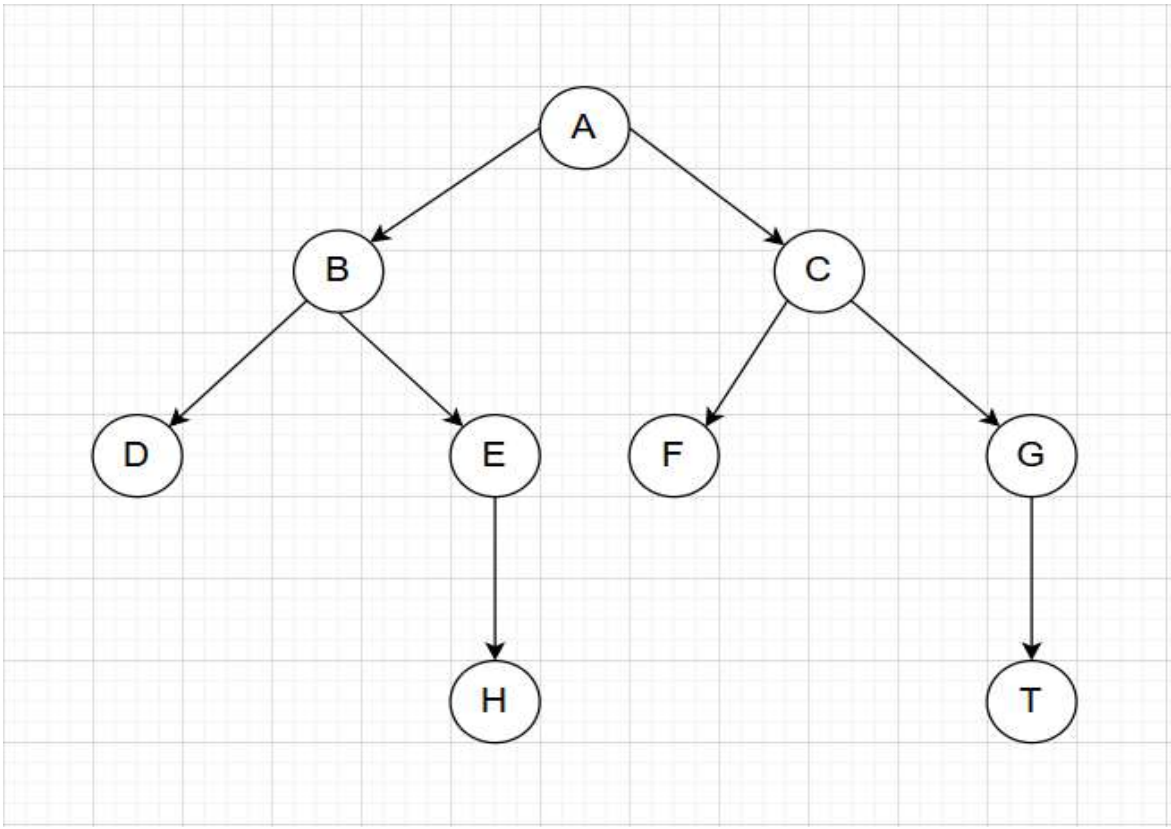
MO = MO ∪ {v} //đưa v vào đầu DS MO

father(v)=n// lưu lại vết đường đi từ n đến v

}

c. Ví dụ thuật toán tìm kiếm theo chiều sâu.

Cho đồ thị hình vẽ sau:



Yêu cầu bài toán: Tìm đường đi p từ A đến một đỉnh T_G Goal bằng thuật toán tìm kiếm theo chiều sâu biết đỉnh đầu T_0=A, Goal = {H,F}

n	B(n)	MO	DONG
		A	
A	B,C	B,C	A
B	D,E	D,E,C	A,B
D	∅	E,C	A,B,D
E	H	H,C	A,B,D,E
D	Đích		

Cha	Con
A	B
A	C
B	D
B	E
E	H

Đường đi từ A đến Goal là : A -> B -> E -> H

Nhận xét:

+ Nếu trong đồ thị G tồn tại đường đi từ T0 đến 1 đỉnh TG Goal thì hàm DFS sẽ dừng lại và cho đường đi p có độ dài có thể không ngắn nhất

+ Với DFS các đỉnh được duyệt theo từng nhánh (theo chiều sâu) .

+ Thuật toán DFS có độ phức tạp O(bd) với b là bậc của cây và d là chiều sâu của cây. Tuy nhiên trong trường hợp xấu nhất cũng là O(b^d)

2.2.2 Thuật toán tìm kiếm theo chiều sâu BFS (Breadth First Search)

a) Tư tưởng của chiến lược tìm kiếm theo chiều rộng

- Từ đỉnh xuất phát duyệt tất cả các đỉnh kề.

- Làm tương tự với các đỉnh vừa được duyệt.

- Quá trình duyệt kết thúc khi tìm thấy đỉnh TG hoặc đã hết các đỉnh để duyệt.

b) Thuật toán tìm kiếm theo chiều rộng

Lưu trữ: Sử dụng hai danh sách DONG và MO hoạt động theo kiểu FIFO (hàng đợi).

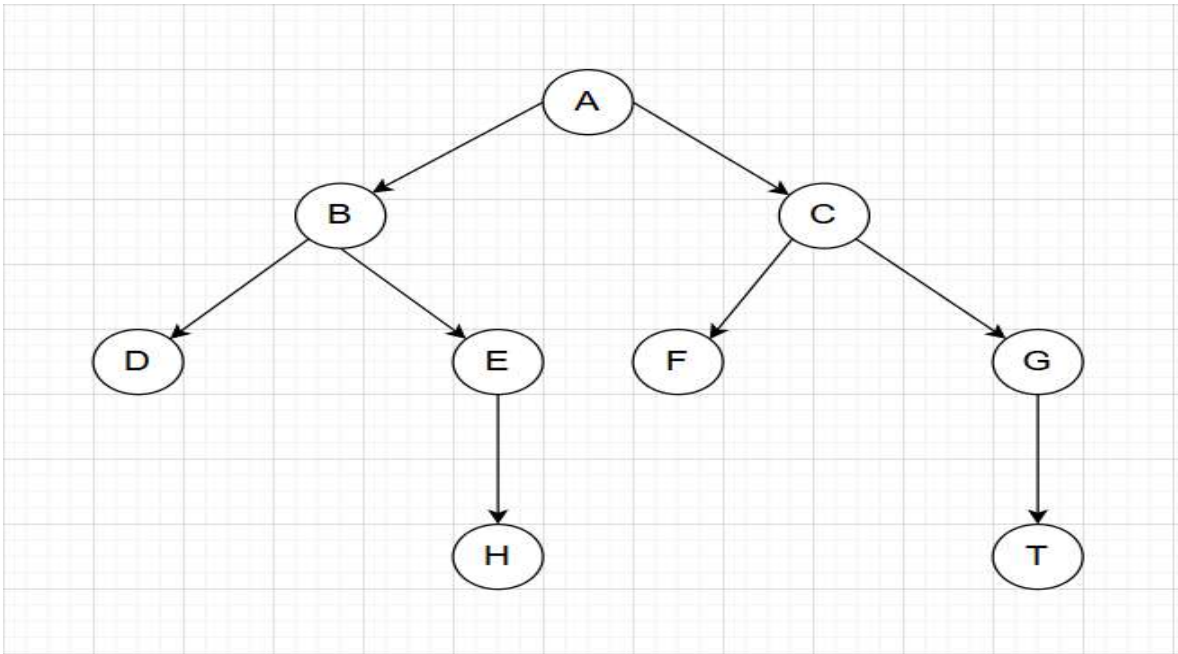
DONG: Chứa các đỉnh đã xét

MO: chứa các đỉnh đang xét

```
MO = ∅; MO = MO ∪ {T0}
    while (MO != ∅)
    {   n = get(MO) // lấy đỉnh đầu trong danh sach MO
        if (n == TG) // nếu n là trạng thái kết thúc
            return TRUE // tìm kiếm thành công, dừng
        DONG = DONG ∪ {n} //đánh dấu n đã được xét
        for các đỉnh kề v của n
            if (v chưa đc xét) //v chưa ở trong DONG
            MO = MO ∪{v} //đưa v vào cuối DS MO
                father(v)=n// lưu lại vết đường đi từ n đến v
    }
```

c) Ví dụ thuật toán tìm kiếm theo chiều rộng.

Cho đồ thị như hình vẽ sau:



Yêu cầu bài toán: Tìm đường đi p từ A đến một đỉnh T_G Goal bằng thuật toán tìm kiếm theo chiều Rộng biết đỉnh đầu T_0=A, Goal = {H,F}

n	B(n)	MO	DONG
		A	
A	B,C	B,C	A
B	D,E	C,D,E	A,B
C	F,G	D,E,F,G	A,B,C
D	∅	E,F,G	A,B,C,D
E	H	F,G,H	A,B,C,D,E
F	Đích		

Cha	Con
A	B
A	C
B	D
B	E
C	F
C	G
E	H

Đường đi từ A đến Goal là : A -> C -> F

Nhận xét:

+ Nếu trong đồ thị tồn tại đường đi từ T_0 đến 1 đỉnh T_G Goal thì hàm BFS sẽ dừng lại và cho đường đi p có độ dài ngắn nhất.

+ Với BFS các đỉnh được duyệt theo từng mức (theo chiều rộng).

+ Thuật toán BFS có độ phức tạp O(b^d) với b là bậc của cây và d là chiều sâu của cây

2.2.3 So sánh 2 thuật toán DFS và BFS

	BFS	DFS
Thứ tự các đỉnh khi duyệt đồ thị	Các đỉnh được duyệt theo từng mức	Các đỉnh được duyệt theo từng nhánh
Độ dài đường đi p từ T_0 đến T_G	Ngắn nhất	Có thể không ngắn nhất
Tính hiệu quả	- Chiến lược có hiệu quả khi lời giải nằm ở mức thấp (gần gốc cây) -Thuận lợi khi tìm kiếm nhiều lời giải	- Chiến lược có hiệu quả khi lời giải nằm gần hướng đi được chọn theo phương án - Thuận lợi khi tìm kiếm 1 lời giải
Sử dụng bộ nhớ	Lưu trữ toàn bộ Không Gian TT của bài toán	Lưu trữ các TT đang xét
Trường hợp tốt nhất	Vét cạn toàn bộ	Phương án chọn đường đi chính xác có lời giải trực tiếp
Trường hợp xấu nhất	Vét cạn	Vét cạn

Kết luận:

- Thuật toán tìm kiếm theo chiều rộng (BFS): lý tưởng khi cần đường đi ngắn nhất và đích gần gốc, nhưng yêu cầu bộ nhớ lớn. Trong các bài toán thực tế, BFS thường được sử dụng trong các ứng dụng như điều hướng GPS (tìm đường đi ngắn nhất qua số giao lộ), phân tích mạng xã hội (tìm

bạn bè ở mức gần nhất), hoặc tìm đường trong game (như tìm lối ra gần nhất).

- Thuật toán tìm kiếm theo chiều sâu (DFS): thích hợp khi bộ nhớ hạn chế và đích có thể nằm sâu. Tuy nhiên, nó không đảm bảo tối ưu và có thể chậm nếu nhánh sai. Để cải tiến, có thể kết hợp DFS với kiểm tra chu trình hoặc sử dụng biến thể như Iterative Deepening DFS (IDDFS) để kết hợp ưu điểm của DFS và BFS.

⟹ Cả hai thuật toán đều là nền tảng cho các phương pháp tìm kiếm nâng cao như Heuristic, giúp hiểu rõ cách khám phá không gian trạng thái.

2.3 Các thuật toán tìm kiếm Heuristic

2.3.1 Tổng quan về thuật toán tìm kiếm Heuristic

a. Khái niệm

Thuật toán tìm kiếm Heuristic là một trong những phương pháp phổ biến trong trí tuệ nhân tạo (AI) và tối ưu hóa. Thuật toán tìm kiếm Heuristic là các phương pháp tìm kiếm không hoàn toàn dựa vào các phương pháp chính xác mà thay vào đó sử dụng các chiến lược hoặc "quy tắc kinh nghiệm" để dẫn dắn quá trình tìm kiếm.

Thuật toán Heuristic hướng tới tìm ra một giải pháp hợp lý hoặc gần tối ưu trong thời gian ngắn bằng việc sử dụng hàm đánh giá (Hàm Heuristic) thay vì thử tất cả các khả năng.

Hàm Heuristic (Hàm đánh giá): là một hàm giúp ước lượng chi phí hoặc giá trị của một trạng thái, từ đó quyết định bước đi tiếp theo trong quá trình tìm kiếm. Nó giúp quyết định hướng đi nào khả thi nhất dựa trên thông tin hiện có.

b. Các thành phần cơ bản trong thuật toán tìm kiếm Heuristic

- Không gian tìm kiếm: Là tập hợp tất cả các trạng thái có thể xảy ra từ trạng thái ban đầu.

- Hàm Heuristic (h(n)): Là một hàm được sử dụng để đánh giá mức độ "tốt" của một trạng thái trong không gian tìm kiếm.

- Chi phí: Mỗi hành động hoặc chuyển động từ trạng thái này sang trạng thái khác có thể có một chi phí nhất định.

- Kết quả (GOAL): Là trạng thái mà thuật toán muốn đạt được, nó có thể là một trạng thái cụ thể hoặc một điều kiện đạt được mục tiêu.

c. Ưu điểm và nhược điểm của thuật toán tìm kiếm Heuristic

Ưu điểm:

Tiết kiệm thời gian và tài nguyên: Heuristic giúp giảm thiểu thời gian tính toán và bộ nhớ nhờ việc không cần thử tất cả các khả năng để tìm kiếm.

Giải pháp nhanh chóng: Heuristic có thể tìm ra giải gần tối ưu nhanh chóng trong thời gian ngắn nhờ việc sử dụng hàm đánh giá.

Ứng dụng rộng rãi: Heuristic được sử dụng trong nhiều lĩnh vực như AI, tối ưu hóa và học máy.

Nhược điểm:

Giải thuật Heuristic có thể tìm ra giải pháp tối ưu nhưng có thể chứ là tối ưu nhất.

Hiệu quả của thuật toán phụ thuộc vào việc xây dựng hàm Heuristic có hợp lý và chính xác hay không.

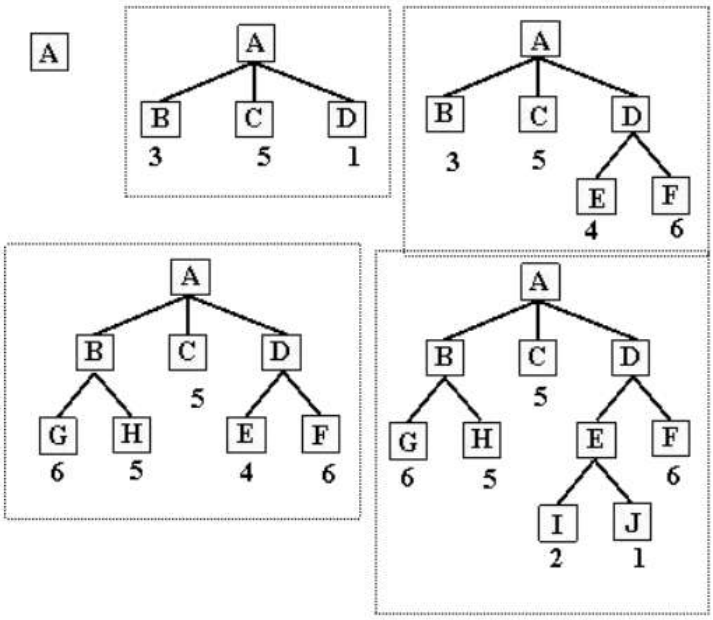
2.3.2 Tìm kiếm tối ưu (Best-First-Search)

Tìm kiếm tối ưu (Best-First Search-BFS): là sự kết hợp của của hai thuật toán tìm kiếm chiều sâu và tìm kiếm chiều rộng cho phép ta đi theo một con đường duy nhất tại một thời điểm,nhưng đồng thời vẫn xét được những hướng khác. Nếu con đường đang đi không triển vọng bằng những con đường đang quan sát, ta sẽ chuyển sang đi theo một trong số các con đường này.

Một cách cụ thể, tại mỗi bước của tìm kiếm BFS, ta chọn đi theo trạng thái có khả năng cao nhất trong số các trạng thái đã được xét cho đến thời điểm

đó. BeFS khác với tìm kiếm leo đồi là chỉ chọn trạng thái có khả năng cao nhất trong số các trạng thái kế tiếp có thể đến được t trạng thái hiện tại. Như vậy, với tiếp cận này, ta sẽ ưu tiên đi vào những nhánh tìm kiếm có khả năng nhất (giống tìm kiếm leo đồi), nhưng ta sẽ không bị lẩn quẩn trong các nhánh này vì nếu càng đi sâu vào một hướng mà ta phát hiện ra rằng hướng này càng đi thì càng xấu, đến mức nó xấu hơn cả những hướng mà ta chưa đi, thì ta sẽ không đi tiếp hướng hiện tại nữa mà chọn đi theo một hướng tốt nhất trong số những hướng chưa đi. Đó là tư tưởng chủ đạo của tìm kiếm tối ưu.

Ví dụ minh họa:



Khởi đầu, chỉ có một nút (trạng thái) A nên nó sẽ được mở rộng tạo ra 3 nút mới B,C và D. Các con số dưới nút là giá trị cho biết độ tốt của nút. Con số càng nhỏ, nút càng tốt. Do D là nút có khả năng nhất nên nó sẽ được mở rộng tiếp sau nút A và sinh ra 2 nút kế tiếp là E và F. Đến đây, ta lại thấy nút B có vẻ có khả năng nhất (trong các nút B,C,E,F) nên ta sẽ chọn mở rộng nút B và tạo ra 2 nút G và H. Nhưng lại một lần nữa, hai nút G, H này được đánh giá ít khả năng hơn E, vì thế sự chú ý lại trở về E. E được mở rộng và các nút được sinh ra từ E là I và J. Ở bước kế tiếp, J sẽ được mở rộng vì nó có khả năng nhất.

Quá trình này tiếp tục cho đến khi tìm thấy một lời giải.

Để cài đặt các thuật giải theo kiểu tìm kiếm BFS, thường cần dùng 2 tập hợp:

- OPEN : tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến (vì ta đã chọn một trạng thái khác). Thực ra, OPEN là một loại hàng đợi ưu tiên (priority queue) mà trong đó, phần tử có độ ưu tiên cao nhất là phần tử tốt nhất. Người ta thường cài đặt hàng đợi ưu tiên bằng Heap.

- CLOSE : tập chứa các trạng thái đã được xét đến. Chúng ta cần lưu trữ những trạng thái này trong bộ nhớ để đề phòng trường hợp khi một trạng thái mới được tạo ra lại trùng với một trạng thái mà ta đã xét đến trước đó. Trong trường hợp không gian tìm kiếm có dạng cây thì không cần dùng tập này.

Thuật giải

- Đặt OPEN chứa trạng thái khởi đầu T_0.

- Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

+ Chọn trạng thái tốt nhất (T_{max}) trong OPEN (và xóa T_{max} khỏi OPEN)

+ Nếu T_{max} là trạng thái kết thúc thì thoát.

+ Ngược lại, tạo ra các trạng thái kế tiếp T_k có thể có từ trạng thái T_{max}. Đối với mỗi trạng thái kế tiếp T_k thực hiện:

Tính $f(T_k)$;

Thêm T_k vào OPEN

BFS khá đơn giản. Tuy vậy, trên thực tế, cũng như tìm kiếm chiều sâu và chiều rộng, hiếm khi ta dùng BFS một cách trực tiếp. Thông thường, người ta thường dùng các phiên bản của BFS là A^T, A^{KT}, A^*.

Thông tin về quá khứ và tương lai:

Thông thường, trong các phương án tìm kiếm theo kiểu BFS, chi phí f của một trạng thái được tính dựa theo hai giá trị mà ta gọi là là g và h. Trong đó

- h: như đã biết, đó là một ước lượng về chi phí từ trạng thái hiện hành cho đến trạng thái đích (thông tin tương lai).

- g là chiều dài quãng đường đã đi từ trạng thái ban đầu cho đến trạng thái hiện tại (thông tin quá khứ).

- Khi đó hàm ước lượng tổng chi phí f(n) được tính theo công thức:

f(n) = g(n) + h(n)

2.3.3 Thuật giải A^T

a, Tư tưởng và chiến lược tìm kiếm A^T

Thuật giải A^T là một phương pháp tìm kiếm theo kiểu BeFS với chi phí của đỉnh là giá trị hàm g(n) (tổng chiều dài thực sự của đường đi từ đỉnh bắt đầu đến đỉnh hiện tại)

Input: Đồ thị G = (V, E) với V: Tập đỉnh, E: Tập cung. Với mỗi một cung người ta gắn thêm một đại lượng gọi là giá của cung

Đỉnh đầu T_0 và Goal là tập chứa các đỉnh đích

Output: Đường đi từ T_0 -> T_G sao cho:

$C(p) = g(n_k) = \min\{g(n)/n \quad Goal\}$ trong đó:

- C(p) là giá của đường đi

- g(n) là tổng chiều dài thực sự của đường đi

b, Giải thuật tìm kiếm

Lưu trữ: Sử dụng hai danh sách DONG và MO

DONG: Chứa các đỉnh đã xét

MO: Chứa các đỉnh đang xét

MO = {T_0}, g(T_0} = 0, DONG =

While MO # do

{

n □ getNew(MO)

If (n = TG) then return True

Else

{

For each m A(n) do

If(m□MO) and (m□DONG) then

{

g(m) = g(n) + cost(m,n)

MO=MO∗{m}

}

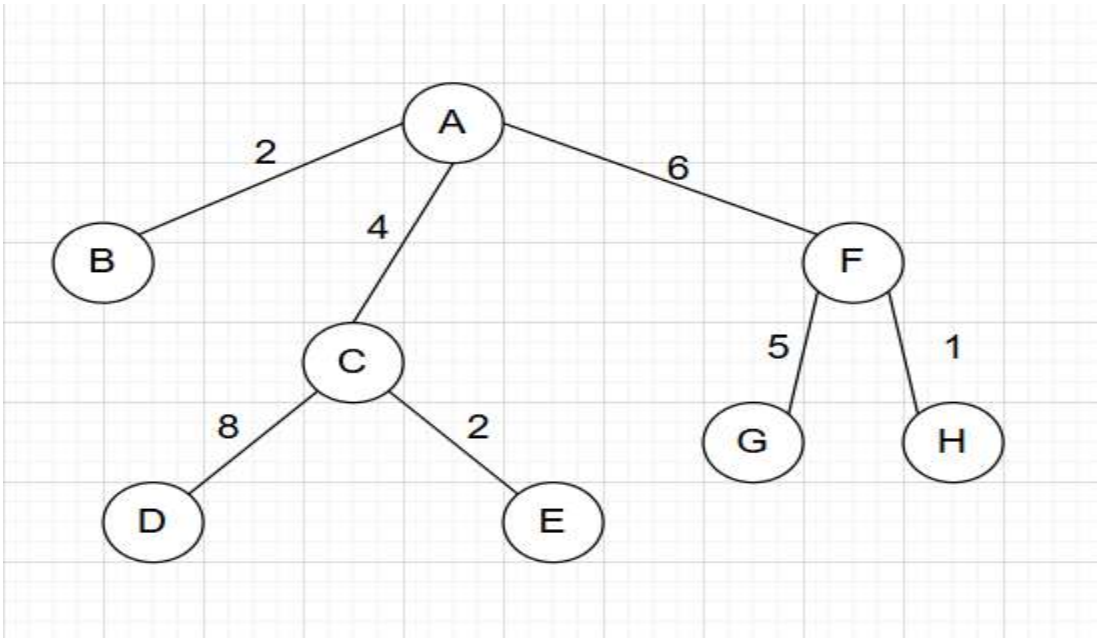
Else g(m) = min{g(m), g_{new}(m)}

DONG = DONG∗{n}

}

Return False

c, Ví dụ: Cho đồ thị sau. Đỉnh xuất phát A và Goal = {D, H}



Lặp	n	B(n)	MO	DONG
Khởi tạo			A(0)	Ø
1	A(0)	B, C, F	B(2), C(4), F(6)	A

2	B(2)	Ø	C(4), F(6)	A, B
3	C(4)	D, E	D(12), E(6), F(6)	A, B, C
4	E(6)	Ø	D(12), F(6)	A, B, C, E
5	F(6)	G, H	G(11), H(7), D(12)	A, B, C, E, F
6	H(7)			

Thuật giải kết thúc khi gặp H Goal và tìm được đường đi p = A -> F -> H là đường đi ngắn nhất với chi phí C(p) = 7

2.3.4 Thuật giải A^{KT}

a, Tư tưởng và chiến lược tìm kiếm A^{KT}

Input: Đồ thị G(V, E)

f: V->R^+ (f(n) là hàm chi phí)

Đỉnh đầu T_0 và đỉnh T_G Goal

Output: Đường đi p: T_0 -> T_G Goal

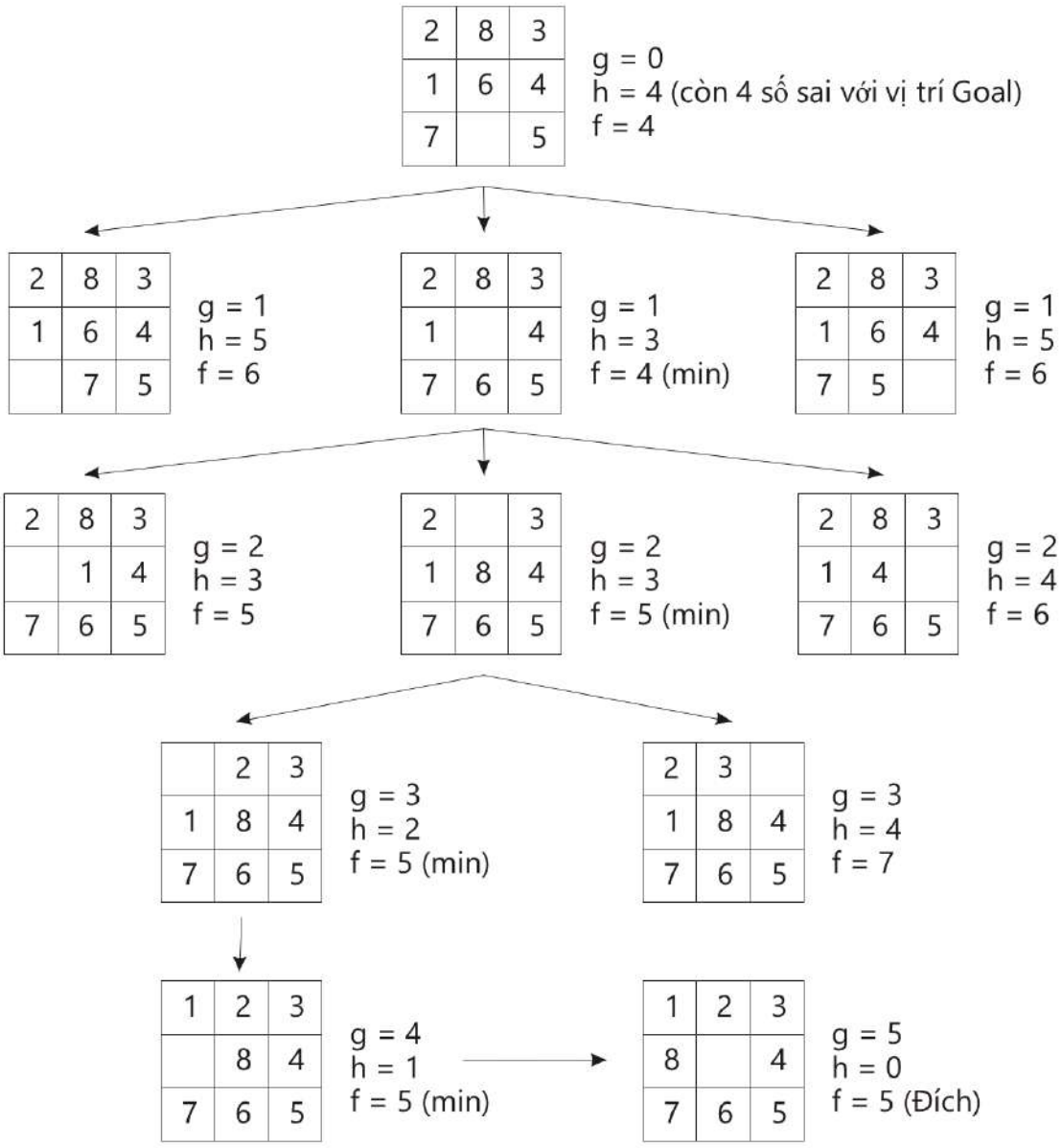
b, Giải thuật

```
MO = {T0}, g(T0) = 0
Tính h(T0), f(T0) = g(T0)+h(T0)
 while MO≠ Ø do
        { n ← getnew(MO)//lấy đỉnh n sao cho f(n) min
        if( n=TG) then return True
        else
                { for each m∈ B(n) do
                        { g(m)=g(n) + cost(m,n)
                          Tính h(m), f(m)= g(n) + h(m)
                           MO=MO ∪ {m}    Parent(m)=n
                        }
                }
        return False
        }
```

c, Ví dụ: Bài toán trò chơi xếp ô số

Chọn f(n) = g(n) + h(n)

Trong đó g(n) là giá của đường đi từ T_0 đến đỉnh n, h(n) là số các ô nằm không đúng vị trí so với trạng thái đích



2.3.5 Thuật giải A*

a, Tư tưởng và chiến lược tìm kiếm A*

A* là một phiên bản đặc biệt của A^{KT} áp dụng cho trường hợp đồ thị. Thuật giải A* có sử dụng tập hợp DONG để lưu trữ những trường hợp đã được xét đến. A* mở rộng A^{KT} bằng cách bổ sung cách giải quyết trường hợp khi mở một nút mà nút này đã có sẵn trong MO hoặc DONG. Khi xét đến một trạng thái T_i, bên cạnh việc lưu trữ 3 giá trị cơ bản g, h, f để phản ánh chi pho của trạng thái đó, A* còn lưu trữ thêm hai thông số sau:

- Trạng thái cha của trạng thái T_i (ký hiệu là Cha(T_i) : cho biết trạng thái dẫn đến

trạng thái Ti. Trong trường hợp có nhiều trạng thái dẫn đến Ti thì chọn Cha(T_i)

sao cho chi pho đi t trạng thái khởi đầu đến Ti là thấp nhất, ngha là :

$g(T_i) = g(T_{cha}) + cost(T_{cha}, T_i)$ là thấp nhất.

- Danh sách các trạng thái kế tiếp của Ti: danh sách này lưu trữ các trạng thái kế

tiếp Tk của Ti sao cho chi pho đến Tk thông qua Ti t trạng thái ban đầu là thấp

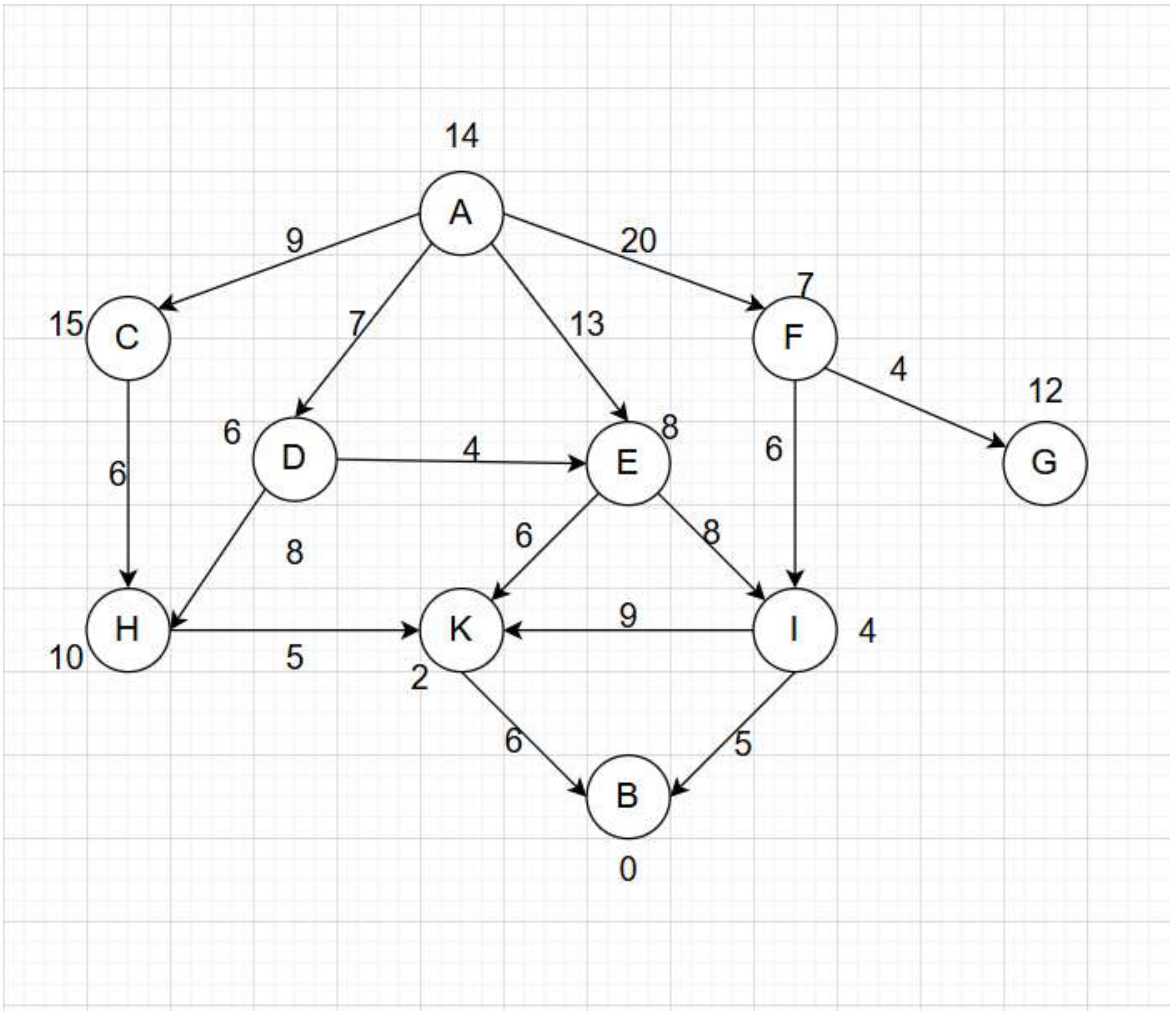
nhất

b, Giải thuật tìm kiếm

```
MO = {T0}, DONG=, g(T0)=0, Tính h(T0), f(T0)=g(T0)+h(T0);
while (MO!=){
        n=getnew(MO)//lấy đỉnh n sao cho f(n) đạt min.
        if (n==TG) return TRUE;
        else{
                for (mB(n)) {
                        if (m(MODONG)){
                                Tính h(m), g(m), f(m)=g(m)+h(m);
                                Cha(m)=n; MO=MO{m};
                        }else{
                                g(m) = min{gold(m),gnew(m)};
                                Cập nhật lại MO;

                        }
                }
        }
        DONG = DONG{n};
}
return FALSE;
```

c, Ví dụ thuật toán A*

Cho đồ thị sau, đỉnh xuất phát A, Goal = {B}



Lặp	n	B(n)	MO	DONG
khởi tạo			A^{0}_{14}	Ø
1	A^{0}_{14}	C^{9}_{24} D^{7}_{13} $E^{13}_{21}F^{20}_{27}$	D^{7}_{13} $E^{13}_{21}C^{9}_{24}$ F^{20}_{27}	A
2	D^{7}_{13}	H^{15}_{25} E^{11}_{19}	E^{11}_{19} C^{9}_{24} H^{15}_{25} F^{20}_{27}	AD
3	E^{11}_{1}	K^{15}_{17} I^{14}_{18}	K^{15}_{17} I^{14}_{18} C^{9}_{24} H^{15}_{25} F^{20}_{27}	ADE
4	K^{15}_{17}	B^{21}_{21}	I^{14}_{18} B^{21}_{21} C^{9}_{24} H^{15}_{25} F^{20}_{27}	ADEK
5	I^{14}_{18}	B^{19}_{19}	B^{19}_{19} C^{9}_{24} H^{15}_{25} F^{20}_{27}	ADEKI
6	B^{19}_{19}			

m	Parent
A	
B	K/I
C	A
D	A
E	A/D
F	A
G	
H	D
I	E
K	E

Tìm ngược trên quan hệ cha con:

Đường đi: A->D->E->I-> có tổng chi phí C(p) = 19

2.3.6 So sánh A*, A^T và A^(KT)

Tiêu chí	A^(T)	A^(KT)	A*
Độ phức tạp thời gian	O(b^d)	O(b^d) nhưng cải thiện nếu h(n) chính xác	O(b^d) nhưng thường tốt hơn nhờ f(n)
Độ phức tạp không gian	O(b^d)	O(b^d)	O(b^d)
Ưu điểm	- Đơn giản, dễ triển khai. - Dễ hiểu cho các bài toán nhỏ.	- Cải thiện hiệu suất nếu h(n) chính xác. - Có thể nhanh chóng tìm ra giải pháp.	- Tối ưu trong tìm kiếm đường đi ngắn nhất. - Sử dụng hàm f(n) hiệu quả.
Nhược điểm	- Không đảm bảo tìm được đường đi ngắn nhất. - Có thể duyệt qua nhiều đỉnh không cần thiết.	- Phụ thuộc vào độ chính xác của h(n). - Nếu h(n) không chính xác, hiệu suất có thể giảm.	- Đòi hỏi nhiều bộ nhớ hơn. - Cần lưu trữ thông tin về các đỉnh đã xét.
Lựa chọn	- Phù hợp cho bài toán đơn giản.	- Sử dụng khi có hàm ước lượng chính xác.	- Tối ưu nhất cho nhiều tình huống khác nhau.

Kết luận:

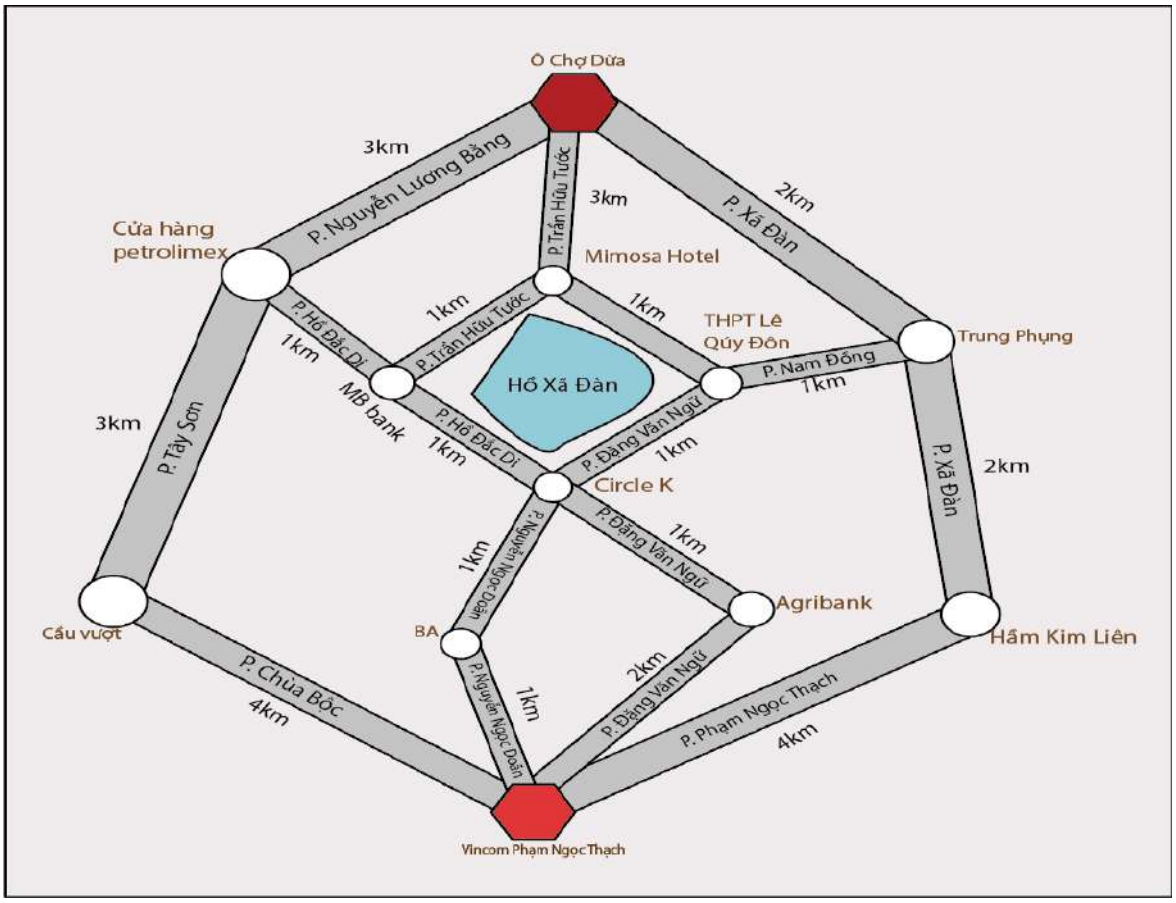
Dựa trên bảng so sánh, A* nổi bật như một thuật toán tối ưu nhất trong việc tìm kiếm đường đi ngắn nhất, nhờ vào cách tiếp cận hiệu quả và khả năng đảm bảo tìm được giải pháp tối ưu.

CHƯƠNG 3. XÂY DỰNG CHƯƠNG TRÌNH ỨNG DỤNG BÀI TOÁN TÌM KIẾM ĐƯỜNG ĐI NGẮN NHẤT

3.1 Không gian trạng thái của bài toán tìm đường đi ngắn nhất

3.1.1 Mô tả bài toán

Thắng là một nhân viên giao hàng của Shopee Food, do Thắng đăng ký chạy ở khu vực Đống Đa, nơi mà Thắng không thuộc đường nhiều do đó Thắng đã nhận sự giúp đỡ từ Google Map (Ứng dụng tìm đường đi phổ biến nhất hiện nay). Đơn hàng đầu tiên của Thắng cần phải giao là một xuất gà rán KFC. Được biết đơn hàng được đặt ở Vincom Phạm Ngọc Thạch và địa chỉ của người mua là ở Ô chợ Dừa. Google Map đã sử dụng thuật toán tìm đường đi ngắn nhất để giúp Thắng giao hàng nhanh nhất có thể, hãy mô tả lại thuật toán đó trong tình huống trên, biết đường đi như sau:



Hình 3: Mô tả bài toán

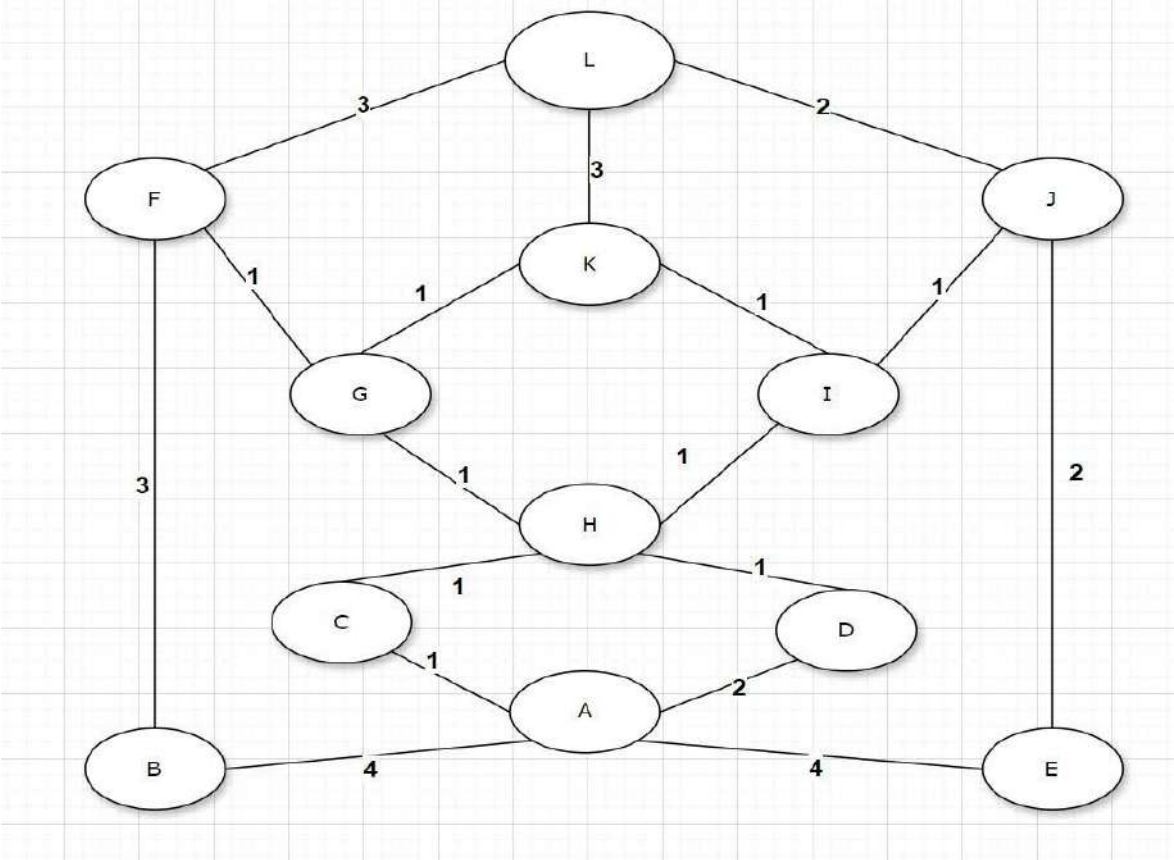
Để tiện cho việc giải bài toán, ta đặt tên các địa chỉ của map thành các chữ cái như sau:

- A = Vincom Phạm Ngọc Thạch
- B = Cầu vượt
- C = BA (Banking Academy)
- D = Agribank
- E = Hầm Kim Liên
- F = Cửa hàng Petrolimex
- G = MB bank
- H = Circle K
- I = THPT Lê Quý Đôn
- J = Trung Phụng
- K = Mimosa Hotel

- L = Ô Chợ Dừa

Nhiệm vụ sẽ là tìm đường đi ngắn nhất từ Vincom Phạm Ngọc Thạch đến Ô chợ Dừa hay từ điểm A đến điểm L.

3.1.2 Không gian trạng thái



Ta có đồ thị G(V, E), trong đó:

V = {A,B,C,D,E,F,G,H,I,J,K,L}

E = {AB,AC,AD,AE,CH,DH,HG,HI,BF,FL,GF,IJ,EJ,JL,IK,GK,KL}

Cần tìm đường đi ngắn nhất từ A đến L, Goal = {L}

Trạng thái đầu: T0 = {A}

Trạng thái cuối: TG = {L}

C(p) = min{g(L)}

3.1.3 Lời giải

Lặp	N	B_n	MO	DONG	Cha	Con
Khởi tạo			A(0)		A	B
1	A(0)	B, C, D, E	B(4), C(1), D(2), E(4)	A	A	C
2	C(1)	H	H(2), B(4), D(2), E(4)	A, C	A	D
3	H(2)	G, I, D	G(3), I(3), B(4), D(2), E(4)	A, C, H	A	E
4	D(2)	∅	G(3), I(3), B(4), E(4)	A, C, H, D	C	H
5	G(3)	F, K	F(4), K(4), I(3), B(4), E(4)	A, C, H, D, G	H	G
6	I(3)	K, J	K(4), J(4), F(4), B(4), E(4)	A, C, H, D, G, I	H	I
7	K(4)	L	L(7), J(4), F(4), B(4), E(4)	A, C, H, D, G, I, K	G	F
8	J(4)	E, L	E(4), L(6), F(4), B(4)	A, C, H, D, G, I, K, J	G	K
9	E(4)	∅	L(6), F(4), B(4)	A, C, H, D, G, I, K, L, E	I	J
10	F(4)	B, L	B(4), L(6)	A, C, H, D, G, I, K, J, E, F	~~K~~	~~L~~
11	B(4)		L(6)	A, C, H, D, G, I, K, J, E, F, B	J	L
12	L(6)	Đích	Dừng			

=> Đường đi A->C->H->I->J->L C(p)=6

3.2 Phân tích các bước cài đặt thuật toán

3.2.1 Thuật toán

a. Lớp Node

Lớp Node là một cấu trúc cơ bản để sử dụng trong các thuật toán tìm đường, chứa các thông tin về tên nút, chi phí ước lượng, chi phí thực tế, tổng chi phí dự kiến, và tham chiếu đến nút cha. Phương thức so sánh được sử dụng để hỗ trợ sắp xếp các nút theo chi phí thấp nhất trong quá trình tìm kiếm.

```
# lop Node
class Node:
    def __init__(self, name, h_value):
        self.name = name
        self.h = h_value
        self.g = float('inf')
        self.f = float('inf')
        self.parent = None

    def __lt__(self, other):
        return self.f < other.f
```

Hình 4: Đoạn code demo lớp Node

1. ‘__init__(self, name, h_value)’
    - Mô tả: Phương thức khởi tạo (constructor) của lớp Node.
    - Tham số đầu vào:

    + ‘name’: Tên hoặc ký hiệu của nút, thường là một chuỗi (string).

    + ‘h_value’: Giá trị heuristic (h), là một số đại diện cho ước lượng chi phí còn lại từ nút hiện tại đến đích trong các thuật toán tìm đường.

    - Các thuộc tính khởi tạo:

    + ‘self.name’: Gán giá trị name cho thuộc tính name của đối tượng.

    + ‘self.h’: Gán giá trị ‘h_value’ cho thuộc tính h (giá trị heuristic).

+ ‘self.g’: Khởi tạo chi phí từ điểm bắt đầu đến nút hiện tại (g) là vô cùng (float(‘inf’)), vì chưa có đường đi nào được tính toán.

+ ‘self.f’: Khởi tạo tổng chi phí dự kiến ‘f’ bằng ‘g’ + ‘h’ (chi phí thực tế và chi phí ước lượng), ban đầu là vô cùng (float(‘inf’)).

+ ‘self.paren’t: Khởi tạo thuộc tính parent để lưu nút cha của nút hiện tại trong đường đi, ban đầu là None.

2. ‘__lt__(self,other)’

- Mô tả: Phương thức đặc biệt này được sử dụng để so sánh hai đối tượng Node dựa trên giá trị ‘f’. Phương thức này giúp sắp xếp các đối tượng Node trong một hàng đợi ưu tiên (priority queue).
- Tham số đầu vào: ‘other’ là một đối tượng Node khác.
- Kết quả: Trả về True nếu ‘self.f’ nhỏ hơn ‘other.f’, ngược lại trả về False.

b. Lớp Graph

Lớp Graph đại diện cho một đồ thị (graph) mà trong đó các nút được kết nối với nhau bởi các cạnh (edges) có trọng số (weight). Đồ thị này có thể được sử dụng trong các thuật toán tìm kiếm đường đi như A*, Dijkstra, hoặc BFS/DFS.

```
# Lop do thi
class Graph:
    def __init__(self):
        self.edges = {}

    def add_edge(self, from_node, to_node, distance):
        if from_node not in self.edges:
            self.edges[from_node] = {}
        self.edges[from_node][to_node] = distance
```

Hình 5: Đoạn code demo lớp Graph

1. ‘__init__(self)’
    - Mô tả: Phương thức khởi tạo (constructor) của lớp Graph.
    - Tham số đầu vào: Không có tham số đầu vào.
    - Các thuộc tính khởi tạo: ‘self.edges’ là một từ điển (dictionary) rỗng ban đầu để lưu trữ tất cả các cạnh của đồ thị. Từ điển này ánh xạ từ một nút (from_node) đến một từ điển con chứa các nút đích (to_node) và khoảng cách (trọng số) giữa các nút.
2. ‘add_edge’(self, from_node, to_node, distance)
    - Mô tả: Phương thức này thêm một cạnh (edge) có trọng số vào đồ thị.
    - Tham số đầu vào:

        + ‘from_node’: Tên hoặc ký hiệu của nút bắt đầu của cạnh o’to_node’: Tên hoặc ký hiệu của nút kết thúc của cạnh.

        + ‘distance’: Trọng số của cạnh, thương là một số đại diện cho khoảng cách hoặc chi phí giữa hai nút.

    - Hoạt động:

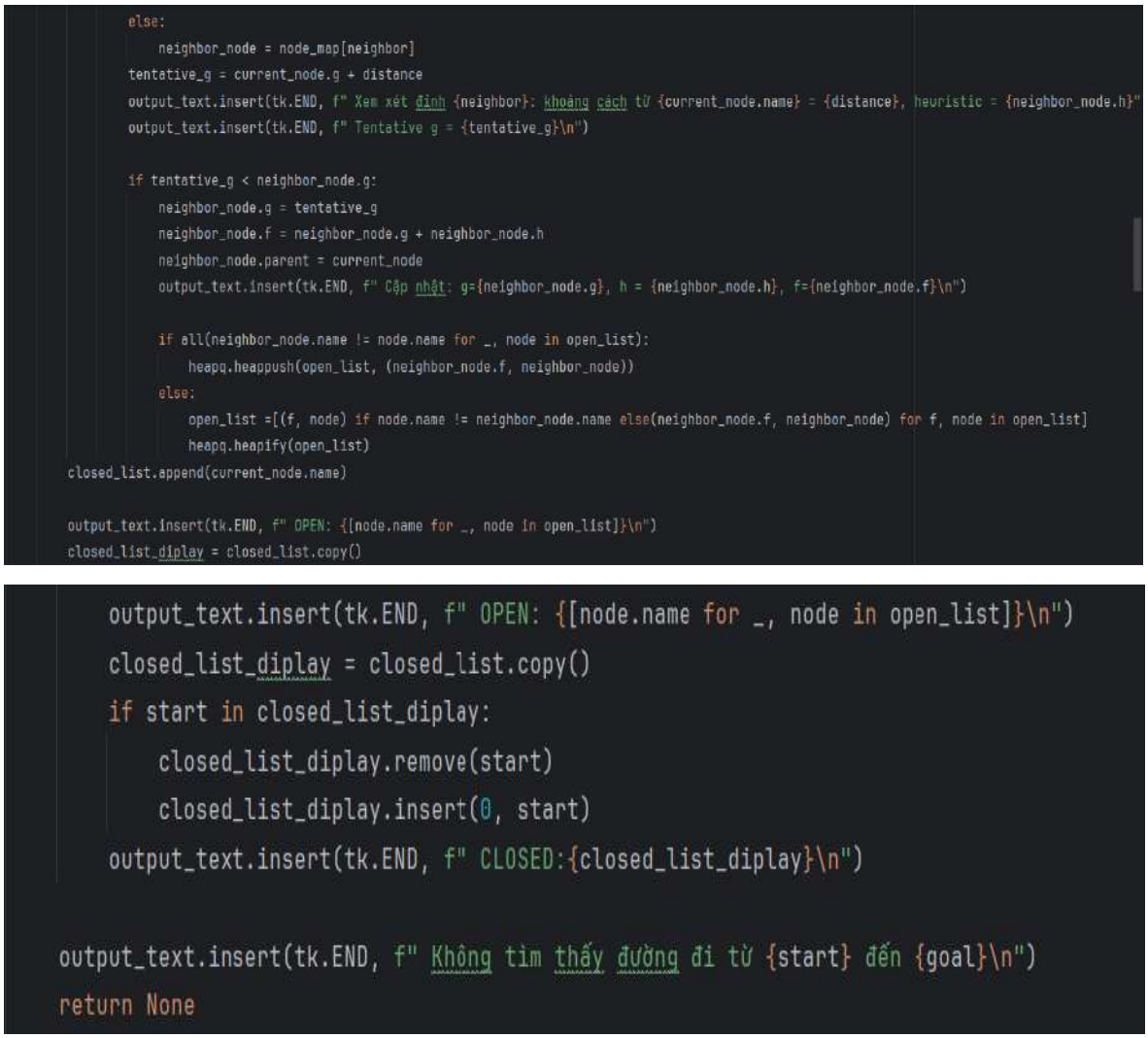
        + Kiểm tra nếu ‘from_node’ chưa có trong ‘self.edges’, thì thêm ‘from_node’ vào từ điển edges và gán cho nó một từ điển rỗng để chứa cạnh xuất phát từ nó.

        + Thêm ‘to_node’ vào từ điển con của ‘from_node’ với trọng số là distance.

c. Thuật toán A*

```python
# thuat toan A*
def astar(graph, start, goal, heuristics, output_text):
    open_list = []
    closed_list = []
    node_map = {}

    start_node = Node(start, heuristics[start])
    goal_node = Node(goal, heuristics[goal])
    start_node.g = 0
    start_node.f = start_node.h

    heapq.heappush(open_list,(start_node.f, start_node))
    node_map[start] = start_node

    output_text.insert(tk.END, f"Khởi tạo:\n")
    output_text.insert(tk.END, f"Thêm đỉnh{start_node.name} vào OPEN với:\n")
    output_text.insert(tk.END, f" g({start_node.name}) = {start_node.g}\n")
    output_text.insert(tk.END, f" h({start_node.name}) = {start_node.g}\n")
    output_text.insert(tk.END, f" f({start_node.name}) = {start_node.g}\n")
    output_text.insert(tk.END, f" OPEN:{[start_node.name]}\n")
    output_text.insert(tk.END, f" CLOSED:{closed_list}\n")
```

```python
    while open_list:
        current_f,current_node = heapq.heappop(open_list)
        output_text.insert(tk.END, f"\nXử lý đỉnh: {current_node.name}\n")

        if current_node.name == goal:
            path=[]
            while current_node:
                path.append(current_node.name)
                current_node = current_node.parent
            output_text.insert(tk.END, f"Đường đi từ {start} đến {goal}:{path[::-1]}\n")
            output_text.insert(tk.END, f"Chi phí từ {start} đến {goal}:{node_map[goal].g}\n")
            output_text.insert(tk.END," Mối quan hệ cha con:\n")
            for node_name, node in node_map.items():
                parent_name = node.parent.name if node.parent else "None"
                output_text.insert(tk.END, f" Đỉnh {node_name}: Cha = {parent_name}\n")
            return path[::-1]
        if current_node.name in graph.edges:
            output_text.insert(tk.END, "Các đỉnh kế:\n")
            for neighbor, distance in graph.edges[current_node.name].items():
                if neighbor not in node_map:
                    neighbor_node = Node(neighbor, heuristics[neighbor])
                    node_map[neighbor] = neighbor_node
```



```
            else:
                neighbor_node = node_map[neighbor]
            tentative_g = current_node.g + distance
            output_text.insert(tk.END, f" Xem xét đỉnh {neighbor}: khoảng cách từ {current_node.name} = {distance}, heuristic = {neighbor_node.h}"
            output_text.insert(tk.END, f" Tentative g = {tentative_g}\n")

            if tentative_g < neighbor_node.g:
                neighbor_node.g = tentative_g
                neighbor_node.f = neighbor_node.g + neighbor_node.h
                neighbor_node.parent = current_node
                output_text.insert(tk.END, f" Cập nhật: g={neighbor_node.g}, h = {neighbor_node.h}, f={neighbor_node.f}\n")

                if all(neighbor_node.name != node.name for _, node in open_list):
                    heapq.heappush(open_list, (neighbor_node.f, neighbor_node))
                else:
                    open_list =[(f, node) if node.name != neighbor_node.name else(neighbor_node.f, neighbor_node) for f, node in open_list]
                    heapq.heapify(open_list)
        closed_list.append(current_node.name)

        output_text.insert(tk.END, f" OPEN: {[node.name for _, node in open_list]}\n")
        closed_list_diplay = closed_list.copy()
```

```
    output_text.insert(tk.END, f" OPEN: {[node.name for _, node in open_list]}\n")
    closed_list_diplay = closed_list.copy()
    if start in closed_list_diplay:
        closed_list_diplay.remove(start)
        closed_list_diplay.insert(0, start)
    output_text.insert(tk.END, f" CLOSED:{closed_list_diplay}\n")

output_text.insert(tk.END, f" Không tìm thấy đường đi từ {start} đến {goal}\n")
return None
```

*Hình 6: Đoạn code demo thuật toán A**

Đầu tiên khởi tạo các cấu trúc dữ liệu:

- ‘open_list’: Hàng đợi ưu tiên dùng để lưu trữ các đỉnh đang được xem xét, sắp xếp theo giá trị ‘f’ (tổng chi phí từ điểm bắt đầu tới điểm đó cộng với ước lượng chi phí từ điểm đó tới đích).
- ‘closed_list’: Danh sách chứa các đỉnh đã được mở rộng, tức là các đỉnh mà thuật toán đã xem xét và sẽ không được xem xét lại.
- ‘node_map’: Một bản đồ để lưu trữ các đỉnh đã được tạo, giúp truy cập và cập nhật nhanh chóng các giá trị ‘g’, ‘h’, ‘f’ và ‘parent’ của các đỉnh.

Khởi tạo đỉnh bắt đầu và đỉnh đích:

- ‘start_node’: Tạo một đối tượng ‘Node’ đại diện cho đỉnh bắt đầu (‘start’) với giá trị ‘h’ là ước lượng chi phí từ đỉnh bắt đầu tới đỉnh đích.
- goal_node’: Tạo một đối tượng ‘Node’ đại diện cho đỉnh đích (‘goal’) với giá trị ‘h’ tương tự.
- Giá trị ‘g’ và ‘f’ của ‘start_node’ được khởi tạo:
  + ‘g = 0’: Chi phí từ đỉnh bắt đầu đến chính nó là 0.
  + ‘f = h’: Giá trị ‘f’ của đỉnh bắt đầu bằng giá trị ‘h’ (ước lượng).

Vòng lặp chính của thuật toán:

Bước 1: Lấy đỉnh có ‘f’ nhỏ nhất từ ‘open_list’

- Đỉnh này là đỉnh có khả năng dẫn tới đích với chi phí thấp nhất tính tới thời điểm hiện tại.
- Nếu đỉnh này là đỉnh đích (‘goal’), thuật toán dừng lại và đường đi được xác định bằng cách truy ngược lại từ đỉnh đích qua các liên kết cha (‘parent’).

Bước 2: Mở rộng đỉnh hiện tại.

- Xem xét tất cả các đỉnh kề (neighbor) của đỉnh hiện tại.
- Tính toán giá trị ‘tentative_g’ (giá trị ‘g’ tạm thời) cho các đỉnh kề.
- Nếu ‘tentative_g’ nhỏ hơn giá trị ‘g’ hiện tại của đỉnh kề, cập nhật lại các giá trị ‘g’, ‘h’, ‘f’ và thiết lập ‘parent’ của đỉnh kề là đỉnh hiện tại

Bước 3: Cập nhật danh sách ‘open_list’ và ‘closed_list’

- Đỉnh hiện tại được thêm vào ‘closed_list’.
- Các đỉnh kề mới hoặc được cập nhật sẽ được thêm hoặc cập nhật vào ‘open_list’.

Kết thúc thuật toán:

- Nếu ‘open_list’ rỗng và không có đường đi tới đỉnh đích, thuật toán kết thúc với thông báo không tìm thấy đường đi.

- Nếu tìm được đường đi, thuật toán sẽ trả về đường đi từ ‘start’ đến ‘goal’ và chi phí tương ứng.

Các phần thông báo (output_text:

- Các thông báo trong đoạn mã là để ghi lại quá tình thực hiện của thuật toán, được in qua ‘output_text’ để giúp dễ dàng theo dõi tiến trình.

d. Hàm `draw_graph(graph, path=None)`

```
# ham ve do thi
def draw_graph(graph,path = None):
    G = nx.Graph()

    for from_node, neighbors in graph.edges.items():
        for to_node, distance in neighbors.items():
            G.add_edge(from_node, to_node, weight=distance)

    pos = nx.spring_layout(G, seed = 42)

    labels = {node : f'{node}\n h={heuristics.get(node,"N/A")}' for node in G.nodes()}

    nx.draw(G, pos, with_labels = False, node_color = 'lightblue', node_size = 500, font_size=10)
    nx.draw_networkx_labels(G, pos,labels, font_size = 10, font_weight = 'bold')
    edge_labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

    if path:
        edge_list = [(path[i],path[i+1]) for i in range(len(path)-1)]
        nx.draw_networkx_edges(G, pos, edgelist = edge_list, edge_color = 'r', width = 2)

    return plt.gcf(), pos
```

Hình 7: Đoạn code demo hàm vẽ đồ thị

- Hàm `draw_graph` được sử dụng để vẽ đồ thị dựa trên dữ liệu từ lớp `Graph`. Đồ thị được biểu diễn dưới dạng đồ thị vô hướng (undirected graph) và có thể hiển thị đường đi (path) nếu đường đi này được cung cấp.
- Hàm `draw_graph` được sử dụng để vẽ đồ thị dựa trên dữ liệu từ lớp `Graph`. Đồ thị được biểu diễn dưới dạng đồ thị vô hướng (undirected graph) và có thể hiển thị đường đi (path) nếu đường đi này được cung cấp.

- Hàm `draw_graph` được sử dụng để vẽ đồ thị dựa trên dữ liệu từ lớp `Graph`. Đồ thị được biểu diễn dưới dạng đồ thị vô hướng (undirected graph) và có thể hiển thị đường đi (path) nếu đường đi này được cung cấp.
- Tham số đầu vào:

  + `graph`: Một đối tượng thuộc lớp `Graph`, chứa các đỉnh và các cạnh của đồ thị.

  + `graph`: Một đối tượng thuộc lớp `Graph`, chứa các đỉnh và các cạnh của đồ thị.

  + `path` (tuỳ chọn): Một danh sách các đỉnh thể hiện đường đi từ đỉnh bắt đầu đến đỉnh đích, được tô màu đỏ trên đồ thị. Nếu không có, đường đi sẽ không được vẽ.
- Hoạt động:

1. Khởi tạo đồ thị

   - Tạo một đối tượng 'Graph' từ thư viện NetworkX.

   - Các cạnh của đồ thị (cùng với trọng số) được thêm vào đối tượng 'G'.

2. Sắp xếp vị trí các đỉnh ('pos')

   - Sử dụng phương pháp 'spring_layout' của NetworkX để tính toán vị trí các đỉnh. `seed=42` đảm bảo rằng các vị trí này luôn được xác định giống nhau mỗi khi hàm được gọi, nhằm tạo sự ổn định trong việc hiển thị đồ thị.

3. Hiển thị đồ thị

   - Các đỉnh của đồ thị được vẽ bằng màu xanh nhạt (`lightblue`) và các cạnh được vẽ bằng màu đen mặc định.

   - Sử dụng `nx.draw_networkx_labels` để hiển thị tên đỉnh và giá trị heuristic của chúng.

4. Hiển thị đường đi (nếu có)

   - Nếu tham số ‘path’ được cung cấp, các cạnh của đường đi sẽ được vẽ bằng màu đỏ với độ rộng là 2 (`width=2`).

5. Kết quả

   - Hàm trả về đối tượng ‘Figure’ (đồ thị được vẽ) và ‘pos’ (vị trí các đỉnh).

e. Hàm `plot_graph_in_gui(graph, canvas, path=None)`

```
# ve do thi trong giao dien
def plot_graph_in_gui(graph, canvas, path = None):
    fig, pos = draw_graph(graph, path)

    fig.set_size_inches(6,6)

    canvas.figure = fig
    canvas.draw()
```

Hình 8: Đoạn code demo hàm vẽ đồ thị trong giao diện

- Hàm `plot_graph_in_gui` vẽ đồ thị trên giao diện đồ họa Tkinter.
- Tham số đầu vào:

  + `graph`: Một đối tượng thuộc lớp `Graph`, chứa các đỉnh và các cạnh của đồ thị.

  + `canvas`: Đối tượng canvas (khung vẽ) của Tkinter, nơi đồ thị sẽ được hiển thị.

  + `path` (tuỳ chọn): Một danh sách các đỉnh thể hiện đường đi từ đỉnh bắt đầu đến đỉnh đích, được tô màu đỏ trên đồ thị.

- Hoạt động:

  + Gọi hàm ‘draw_graph’: Tạo đồ thị và nhận về đối tượng ‘Figure’ và vị trí các đỉnh (‘pos’).

+ Kích thước đồ thị: Thiết lập kích thước của đồ thị (‘6x6’ inch).

+ Vẽ đồ thị trên canvas:

- Gán đồ thị vừa tạo vào ‘canvas.figure’.
- Gọi phương thức ‘ canvas.draw()’ để hiển thị đồ thị trên giao diện người dùng.

f. Hàm ‘run_astar()’:

```
# chay thuat toan A*
def run_astar():
    start = start_entry.get().strip()
    goal = goal_entry.get().strip()

    if start not in heuristics or goal not in heuristics:
        messagebox.showerror("Error","Điểm bắt đầu hoặc điểm đích không hợp lệ!")
        return

    output_text.delete(1.0, tk.END)
    path = astar(graph, start, goal, heuristics, output_text)

    plot_graph_in_gui(graph, canvas,path)
```

*Hình 9: Hàm chạy thuật toán A**

- Mô tả: Hàm ‘run_astar’ là nơi khởi động thuật toán A* khi người dùng nhấn nút “RUN” trên giao diện.
- Hoạt động:

+ Lấy dữ liệu đầu vào từ giao diện:

  - Lấy đỉnh bắt đầu và đỉnh đích từ các ô nhập liệu (‘start_entry’, ‘goal_entry’).

+ Kiểm tra hợp lệ:

  - Kiểm tra xem đỉnh bắt đầu và đỉnh đích có nằm trong tập heuristic đã khai báo hay không. Nếu không, hiển thị thông báo lỗi và kết thúc hàm.

+ Xóa nội dung cũ:

- Xóa nội dung cũ trong ‘output_text’ để chuẩn bị ghi lại quá trình mới.

+ Chạy thuật toán A:

- Gọi hàm ‘astar’ với tham số cần thiết.
- Hiển thị đường đi (nếu có) bằng cách gọi ‘plot_graph_in_gui’.

g. Hàm `create_gui()`

```
#tao giao dien
def create_gui():
    global start_entry, goal_entry, output_text, canvas

    root = tk.Tk()
    root.title("Thuật Toán A*")
    root.geometry("1400x800")

    title_label = tk.Label(root, text = "Thuật Toán A* - Bài toán tìm đường đi", font=("Helvetica",20,"bold"))
    title_label.pack(pady=8)

    input_frame = tk.Frame(root)
    input_frame.pack(pady=5)

    tk.Label(input_frame, text="Điểm đầu: ", font=("Helvetica",12)).grid(row = 0, column = 0, padx = 5, pady = 5)
    start_entry = ttk.Entry(input_frame, font=("Helvetica", 12), width = 10)
    start_entry.grid(row=0, column = 1,padx = 5, pady =5)

    tk.Label(input_frame, text = "Điểm đích: ", font=("Helvetica", 12)).grid(row=1, column = 0,padx = 5,pady = 5)
    goal_entry = ttk.Entry(input_frame, font=("Helvetica",12),width=10)
    goal_entry.grid(row=1,column = 1,padx = 5, pady = 5)

    run_button = ttk.Button(input_frame, text ="Run", command =run_astar)
    run_button.grid(row = 2, column = 1, columnspan = 3, pady = 15)
```

```
    # them khung ve do thi
    canvas_frame = tk.Frame(root)
    canvas_frame.pack(side = tk.RIGHT, padx = 20, pady=20)

    fig = plt.figure(figsize=(6,6))
    canvas = FigureCanvasTkAgg(fig, master = canvas_frame)
    canvas.get_tk_widget().pack(side=tk.TOP, fill = tk.BOTH, expand=True)

    # ve do thi lan dau khi giao dien khoi tao
    plot_graph_in_gui(graph, canvas, path=None)
    # hop van ban hien thi qua trinh va ket qua
    output_text = tk.Text(root, height=31, width =110, font=("Consolas", 13),wrap="word", borderwidth=1,relief="groove")
    output_text.pack(side = tk.LEFT, pady=12, padx=12)
    root.mainloop()
```

Hình 10: Tạo giao diện

- Hàm `create_gui` khởi tạo và cấu hình giao diện người dùng (GUI) bằng Tkinter, cho phép người dùng nhập điểm bắt đầu và điểm đích, chạy thuật toán A*, và xem kết quả trên giao diện.
- Hoạt động:
    + Tạo cửa sổ chính (`root`):
        - Thiết lập tiêu đề cửa sổ và kích thước.
    + Tiêu đề và khu vực nhập liệu:
        - Tạo nhãn tiêu đề, khung nhập liệu với các ô nhập điểm bắt đầu và điểm đích.
        - Thêm nút "RUN" để kích hoạt thuật toán A*.
    + Khung vẽ đồ thị:
        - Tạo khung vẽ (`canvas_frame`) để chứa đồ thị, và kết nối Matplotlib với Tkinter thông qua `FigureCanvasTkAgg`.
        - Vẽ đồ thị ban đầu khi giao diện khởi tạo.
    + Hộp văn bản hiển thị kết quả:
        - Tạo hộp văn bản (`output_text`) để hiển thị chi tiết quá trình và kết quả của thuật toán A*.
    + Chạy giao diện:
        - ‘root.mainloop()’ để bắt đầu vòng lặp sự kiện cảu Tkinter, giữ cho giao diện hiển thị và chờ đợi tương tác người dùng.

h. **Khai báo heuristic và đồ thị**

```
#khai bao heuristic cho cac dinh
heuristics = {
    'A' :14,
    'C' : 15,
    'D' : 6,
    'E' : 8,
    'F' : 7,
    'G' : 12,
    'H' : 10,
    'K' : 2,
    'I' : 4,
    'B' : 5,
    'L' : 8,
    'J' : 7,
}
```

```
#khai bao do thi va cac khoang cach giua cac dinh

graph = Graph()
graph.add_edge( from_node: 'A', to_node: 'C', distance: 1)
graph.add_edge( from_node: 'A', to_node: 'D', distance: 2)
graph.add_edge( from_node: 'A', to_node: 'B', distance: 4)
graph.add_edge( from_node: 'A', to_node: 'E', distance: 4)
graph.add_edge( from_node: 'C', to_node: 'H', distance: 1)
graph.add_edge( from_node: 'D', to_node: 'H', distance: 1)
graph.add_edge( from_node: 'H', to_node: 'G', distance: 1)
graph.add_edge( from_node: 'H', to_node: 'I', distance: 1)
graph.add_edge( from_node: 'G', to_node: 'F', distance: 1)
graph.add_edge( from_node: 'G', to_node: 'K', distance: 1)
graph.add_edge( from_node: 'F', to_node: 'L', distance: 3)
graph.add_edge( from_node: 'K', to_node: 'L', distance: 3)
graph.add_edge( from_node: 'J', to_node: 'L', distance: 2)
graph.add_edge( from_node: 'I', to_node: 'J', distance: 1)
graph.add_edge( from_node: 'B', to_node: 'F', distance: 3)
graph.add_edge( from_node: 'E', to_node: 'J', distance: 2)
graph.add_edge( from_node: 'I', to_node: 'K', distance: 1)
```

Hình 11: Khai báo heuristic và đồ thị

- ‘heuritic’: Khai báo giá trị heuristic chop từng đỉnh trong đồ thị. Đây là ước lượng chi phí từ mỗi đỉnh đến đỉnh đích.
- graph’: Tạo đối tượng đồ thị và thêm các cạnh với trọng số vào đồ thị.

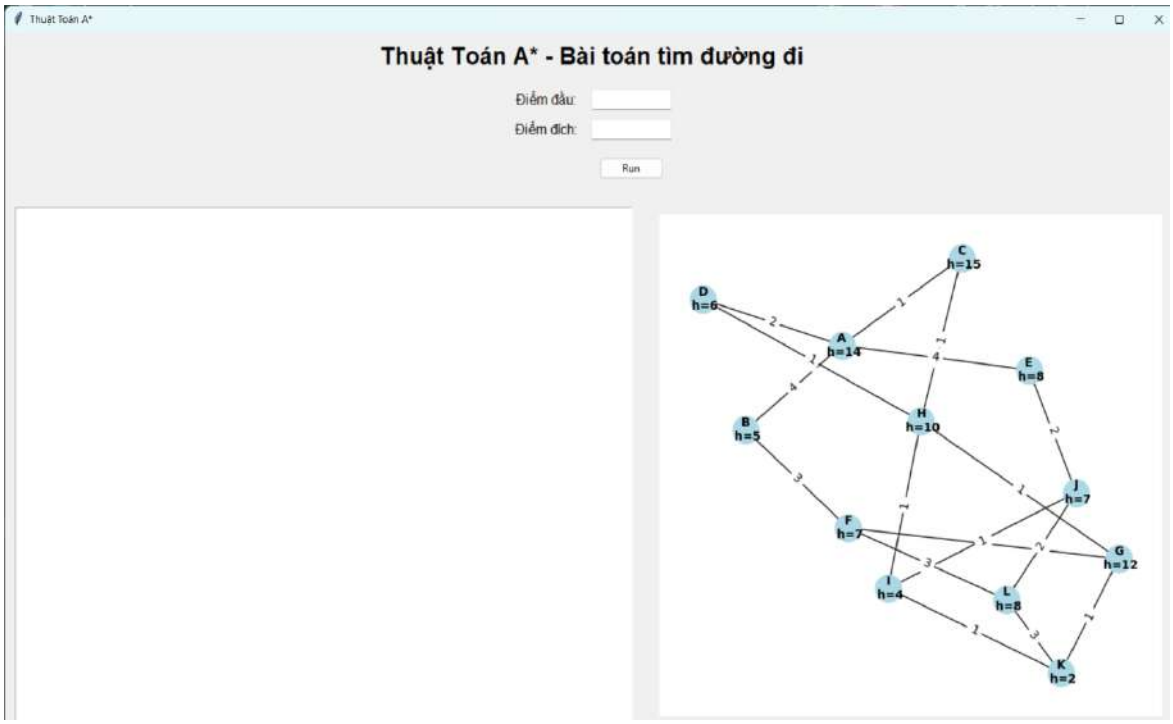
i. Chạy giao diện người dùng

```
#chay phan giao dien
create_gui()
```

Hình 12: Chạy giao diện người dùng

- ‘create_gui()’: Gọi hàm này để khởi tạo và hiển thị giao diện người dùng, từ đó người dùng có thể tương tác và ứng dụng.

3.2.2 Giao diện

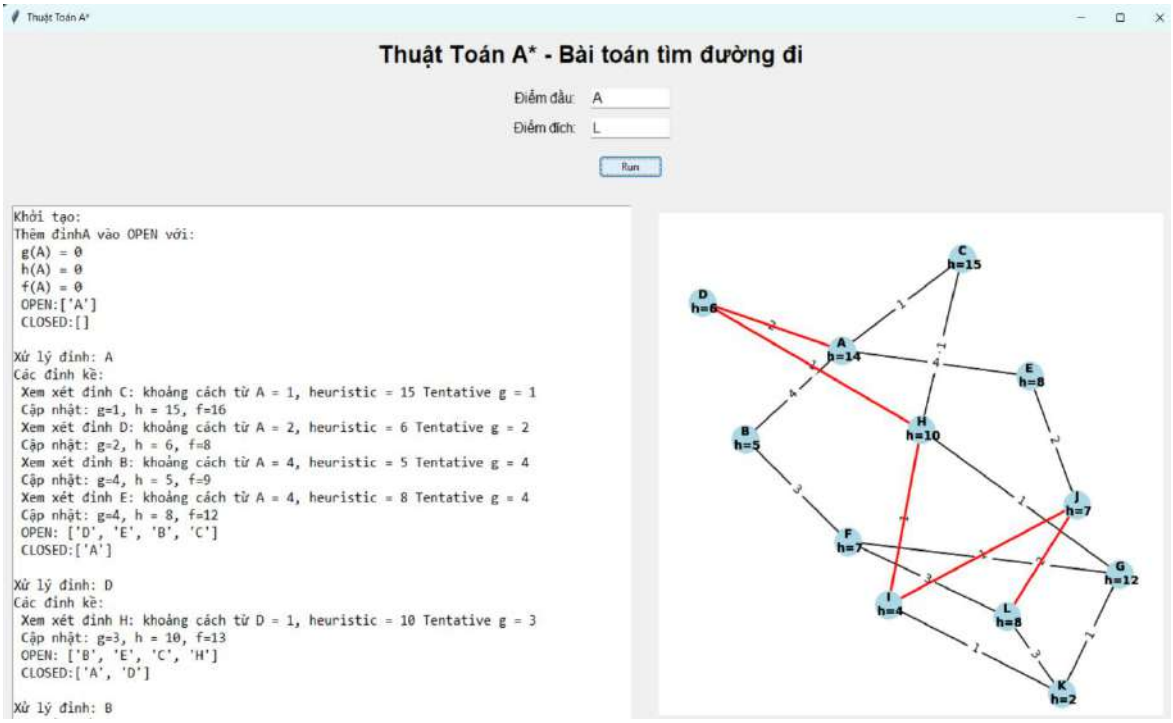


Hình 13: Tổng quan giao diện

Mô tả: Giao diện được chia làm hai phần:

+ Phần bên trên có hai phần input để nhập vào điểm bắt đầu và điểm đích để tìm ra được con đường có hàm chi phí nhỏ nhất. Tiếp theo là nút "Run' để chạy chương trình tìm kiếm đường đi.

+ Bên dưới được chia làm hai phần (bên phải và bên trái): Bên phải là hình ảnh đồ thị minh họa và hiện đường đi khi chúng ta chạy chương trình, còn bên trái hiển thị chương trình cụ thể của bài toán tìm đường đi.



```
Xử lý đỉnh: B
Các đỉnh kề:
 Xem xét đỉnh F: khoảng cách từ B = 3, heuristic = 7 Tentative g = 7
 Cập nhật: g=7, h = 7, f=14
 OPEN: ['E', 'H', 'C', 'F']
 CLOSED:['A', 'D', 'B']

Xử lý đỉnh: E
Các đỉnh kề:
 Xem xét đỉnh J: khoảng cách từ E = 2, heuristic = 7 Tentative g = 6
 Cập nhật: g=6, h = 7, f=13
 OPEN: ['H', 'J', 'C', 'F']
 CLOSED:['A', 'D', 'B', 'E']

Xử lý đỉnh: H
Các đỉnh kề:
 Xem xét đỉnh G: khoảng cách từ H = 1, heuristic = 12 Tentative g = 4
 Cập nhật: g=4, h = 12, f=16
 Xem xét đỉnh I: khoảng cách từ H = 1, heuristic = 4 Tentative g = 4
 Cập nhật: g=4, h = 4, f=8
 OPEN: ['I', 'J', 'C', 'G', 'F']
 CLOSED:['A', 'D', 'B', 'E', 'H']
```

```
Xử lý đỉnh: I
Các đỉnh kề:
 Xem xét đỉnh J: khoảng cách từ I = 1, heuristic = 7 Tentative g = 5
 Cập nhật: g=5, h = 7, f=12
 Xem xét đỉnh K: khoảng cách từ I = 1, heuristic = 2 Tentative g = 5
 Cập nhật: g=5, h = 2, f=7
 OPEN: ['K', 'J', 'C', 'G', 'F']
 CLOSED:['A', 'D', 'B', 'E', 'H', 'I']

Xử lý đỉnh: K
Các đỉnh kề:
 Xem xét đỉnh L: khoảng cách từ K = 3, heuristic = 8 Tentative g = 8
 Cập nhật: g=8, h = 8, f=16
 OPEN: ['J', 'F', 'C', 'G', 'L']
 CLOSED:['A', 'D', 'B', 'E', 'H', 'I', 'K']

Xử lý đỉnh: J
Các đỉnh kề:
 Xem xét đỉnh L: khoảng cách từ J = 2, heuristic = 8 Tentative g = 7
 Cập nhật: g=7, h = 8, f=15
 OPEN: ['F', 'L', 'C', 'G']
 CLOSED:['A', 'D', 'B', 'E', 'H', 'I', 'K', 'J']
```

```
Xử lý đỉnh: F
Các đỉnh kề:
 Xem xét đỉnh L: khoảng cách từ F = 3, heuristic = 8 Tentative g = 10
 OPEN: ['L', 'G', 'C']
 CLOSED:['A', 'D', 'B', 'E', 'H', 'I', 'K', 'J', 'F']

Xử lý đỉnh: L
Đường đi từ A đến L:['A', 'D', 'H', 'I', 'J', 'L']
Chi phi từ A đến L:7
 Mối quan hệ cha con:
 Đỉnh A: Cha = None
 Đỉnh C: Cha = A
 Đỉnh D: Cha = A
 Đỉnh B: Cha = A
 Đỉnh E: Cha = A
 Đỉnh H: Cha = D
 Đỉnh F: Cha = B
 Đỉnh J: Cha = I
 Đỉnh G: Cha = H
 Đỉnh I: Cha = H
 Đỉnh K: Cha = I
 Đỉnh L: Cha = J
```

*Hình 14: Kết quả của thuật toán A**

KẾT LUẬN

Qua quá trình nghiên cứu và triển khai đề tài, nhóm đã:

- Nắm vững kiến thức lý thuyết về các thuật toán tìm kiếm heuristic như BeFS, A^T, A*, và AKT.
- Hiểu rõ vai trò của hàm heuristic trong việc tối ưu hóa quá trình tìm kiếm.

Các thuật toán này đã chứng minh được hiệu quả khi giải bài toán tìm đường đi thông qua việc:

- Giảm chi phí tính toán.
- Định hướng tìm kiếm một cách hợp lý.

Ưu điểm của thuật toán heuristic (đặc biệt là A):*

- Giảm đáng kể số lượng trạng thái cần duyệt nhờ khả năng định hướng thông minh → tiết kiệm thời gian và tài nguyên.
- Kết hợp hiệu quả giữa chi phí đã đi g(n) và chi phí ước lượng còn lại h(n) → đảm bảo tìm được lời giải tối ưu nếu h(n) được thiết kế tốt.
- Linh hoạt trong việc thay đổi hàm heuristic → thích ứng với nhiều bài toán khác nhau.

Nhược điểm cần lưu ý:

- Phụ thuộc nhiều vào chất lượng của hàm heuristic:
    - Nếu h(n) không chính xác → có thể dẫn đến tìm kiếm chậm hoặc lời giải không tối ưu.
- Với các bài toán có không gian trạng thái lớn:
    - A* có thể tiêu tốn nhiều bộ nhớ vì cần lưu trữ nhiều trạng thái trong danh sách mở (OPEN).
- Việc tính toán h(n) phức tạp ở mỗi bước:
    - Có thể làm giảm hiệu suất tổng thể của thuật toán.

TÀI LIỆU THAM KHẢO

[1] Ths Trần Thanh Huân , *Bài giảng môn Trí Tuệ Nhân Tạo*, Trường Đại học CNTT & Truyền thông, 2025.

[2] Nguyễn Phương Nga, Trần Hùng Cường, Giáo trình Trí Tuệ Nhân Tạo, Thống kê 2021.

[3] Giáo trình Trí Tuệ Nhân Tạo – Đại học Bách Khoa Hà Nội.

[4] Sách "Trí tuệ nhân tạo - Cách tiếp cận hiện đại" – Stuart Russell, Peter Norvig (bản dịch tiếng Việt).

[5] Giáo trình "Trí tuệ nhân tạo" – TS. Nguyễn Ngọc Thuần (Đại học Công nghiệp Vinh, 2018).

[6] "Bài toán tìm đường đi ngắn nhất" (trích từ FIT – PTIT HCM, 2018)

[7] Giáo trình nhập môn Trí tuệ nhân tạo (Đại học Quốc gia TP.HCM).

[8] Wikipedia, "Heuristic (Computer Science)", https://en.wikipedia.org/wiki/Heuristic_(computer_science). Truy cập ngày 26/4/2025.

[9] Stuart Russell & Peter Norvig, *Artificial Intelligence: A Modern Approach*, 3rd Edition, Pearson, 2010.

[10] GeeksforGeeks, "A* Search Algorithm", https://www.geeksforgeeks.org/a-search-algorithm/. Truy cập ngày 20/5/2025.