TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO BÀI TẬP LỚN
HỌC PHẦN TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI: TÌM HIỂU THUẬT TOÁN BREADTH-FIRST SEARCH VÀ ỨNG DỤNG VÀO GIẢI QUYẾT TRÒ CHƠI SOKOBAN

GVHD:	Ths. Mai Thanh Hồng
Nhóm – Lớp:	5 – 2024IT6094003
Họ tên sinh viên thực hiện:	
Triệu Kim Dung:	2023601181
Vũ Ngọc Hân:	2023600411
Nguyễn Thị Tuyết Nhung:	2023603641
Vũ Thị Hồng Oanh:	2023601723

Hà Nội, 2025

PHIẾU HỌC TẬP NHÓM 5

I. Thông tin chung

1. Tên lớp:**2024IT6094003** Khóa: ĐH K18

2. Tên nhóm: **Nhóm 5**

Họ và tên thành viên trong nhóm:

(1) Nguyễn Thị Tuyết Nhung - 2023603641

(2) Triệu Kim Dung - 2023601181

(3) Vũ Ngọc Hân - 2023600411

(4) Vũ Thị Hồng Oanh - 2023601723

II. Nội dung học tập

1. Tên chủ đề: Tìm hiểu thuật toán Breadth-First Search và ứng dụng vào giải quyết trò chơi Sokoban

2. Hoạt động của sinh viên:

- Hoạt đông/Nội dung 1: Tổ chức thảo luận và phân chia nhiệm vụ cho các thành viên nhóm. Mục tiêu/chuẩn đầu ra: L4

- Hoạt đông/Nội dung 2: Viết biên bản họp nhóm, xây dựng kế hoạch hoạt động nhóm. Mục tiêu/chuẩn đầu ra: L4

- Hoạt đông/Nội dung 3: Tìm hiểu thuật toán BFS và trò chơi Sokoban. Mục tiêu/chuẩn đầu ra: L3

- Hoạt động/Nội dung 4: Thực hiện cài đặt thuật toán BFS để giải bài toán trò chơi sokoban với ngôn ngữ lập trình Python. Mục tiêu chuẩn đầu ra: L3

- Hoạt động/Nội dung 5: Viết báo cáo bài tập lớn bản Word. Mục tiêu chuẩn đầu ra: L3

3. Sản phẩm nghiên cứu: Báo cáo thực nghiệm tìm hiểu thuật toán Breadth-First Search và ứng dụng vào giải quyết trò chơi Sokoban

III. Nhiệm vụ học tập

1. Hoàn thành Tiểu luận, Bài tập lớn, Đồ án/Dự án theo đúng thời gian quy định (từ ngày 07/04/2025 đến ngày 11/06/2025)

2. Báo cáo sản phẩm nghiên cứu theo chủ đề được giao trước giảng viên và những sinh viên khác

IV. Học liệu thực hiện Tiểu luận, Bài tập lớn, Đồ án/Dự án

1. Tài liệu học tập: Giáo trình trí tuệ nhân tạo trường Đại học Công Nghiệp Hà Nội, các tài liệu tìm kiếm trên mạng

2. Phương tiện, nguyên liệu thực hiện Tiểu luận, Bài tập lớn, Đồ án/Dự án (nếu có)

KẾ HOẠCH THỰC HIỆN TIỂU LUẬN, BÀI TẬP LỚN, ĐỒ ÁN/DỰ ÁN

Tên lớp: **2024IT6094003** Khóa ĐH: K18

Tên nhóm: Nhóm 5

Họ và tên thành viên trong nhóm :

Nguyễn Thị Tuyết Nhung - 2023603641

Triệu Kim Dung - 2023601181

Vũ Ngọc Hân - 2023600411

Vũ Thị Hồng Oanh - 2023601723

Tên chủ đề : Tìm hiểu thuật toán Breadth-First Search và ứng dụng vào giải quyết trò chơi Sokoban

Tuần	Người thực hiện	Nội dung công việc	Phương pháp thực hiện
1	Cả nhóm	- Thống nhất chủ đề, tìm hiểu về nội dung chủ đề - Bầu ra nhóm trưởng, phân chia nhiệm vụ cho các thành viên	-Sưu tầm dữ liệu, nghiên cứu các tài liệu -Thảo luận nhóm, phân tích yêu cầu đề tài
2	Cả nhóm	Tìm hiểu các thuật toán tìm kiếm và trò chơi Sokoban	- Tìm dữ liệu theo từ khóa và dựa trên giáo trình - Viết báo cáo dựa trên những tài liệu đã tìm hiểu được -Đọc tài liệu, chắt lọc nội dung lý thuyết và trình bày rõ ràng, mạch lạc.
	Nguyễn Thị Tuyết Nhung	Tạo bìa báo cáo, trình bày tổng quan về thuật toán tìm kiếm	
	Vũ Thị Hồng Oanh	Trình bày khái niệm, chức năng của bfs	
	Triệu Kim Dung	Trình bày cấu trúc dữ liệu, ưu nhược điểm của bfs	
	Vũ Ngọc Hân	So sánh bfs với các thuật toán khác	
3	Nguyễn Thị Tuyết Nhung Vũ Thị Hồng Oanh	Giới thiệu trò chơi Sokoban, các luật chơi, điều kiện trạng thái đầu, tổng hợp file word	- Tìm dữ liệu theo từ khóa và dựa trên giáo trình

	Triệu Kim Dung Vũ Ngọc Hân	Trình bày không gian trạng thái của bài toán và các hành động hợp lệ , kiểm tra trạng thái đích	- Viết báo cáo dựa trên những tài liệu đã tìm hiểu được -Đọc tài liệu, chắt lọc nội dung lý thuyết và trình bày rõ ràng, mạch lạc.
4	Nguyễn Thị Tuyết Nhung	Cài đặt chương trình	-Tham khảo code trên Google và các tài liệu hiện có -Nghiên cứu, tìm hiểu và chỉnh sửa code -Chạy thử chương trình, kiểm tra tính chính xác và ghi nhận kết quả kiểm thử.
	Vũ Thị Hồng Oanh		
	Vũ Ngọc Hân		
	Triệu Kim Dung	Thực nghiệm trên bản đồ mẫu	
5	Nguyễn Thị Tuyết Nhung	Tổng hợp chỉnh sửa định dạng file word	-Tổng hợp, đánh giá - Kiểm tra, chỉnh sửa nội dung, định dạng trình bày, hoàn thiện báo cáo.
	Vũ Thị Hồng Oanh	Làm chương 3 hướng phát triển	
	Triệu Kim Dung	Đánh giá và bổ sung các nội dung còn sai hoặc thiếu	
	Vũ Ngọc Hân	Làm chương 3 kết luận	
6	Cả nhóm	Kiểm tra chéo các bài làm của nhau toàn bộ nội dung các chương và đưa ra nhận xét	Đọc lại bài làm của mình và của các thành viên khác, xem xét lại 1 cách nghiêm túc và chi tiết
7	Cả nhóm	Tóm tắt nội dung để chuẩn bị trình bày báo cáo Chỉnh sửa định dạng báo cáo hoàn thiện	Nhận xét và bổ sung nếu có

Ngày….tháng…..năm……

XÁC NHẬN CỦA GIẢNG VIÊN

(Ký, ghi rõ họ tên)

BÁO CÁO HỌC TẬP CÁ NHÂN/NHÓM

Tên lớp: **2024IT6094003** Khóa: ĐH K18

Tên nhóm: Nhóm 5

Tên chủ đề: Tìm hiểu thuật toán Breadth-First Search và ứng dụng vào giải quyết trò chơi Sokoban

Tuần	**Người thực hiện**	**Nội dung công việc**	**Kết quả đạt được**	**Kiến nghị với giảng viên hướng dẫn**
1	Nguyễn Thị Tuyết Nhung	-Phân công công việc cho các thành viên -Thống nhất chủ đề BTL	- Đã phân chia nhiệm vụ rõ ràng - Chọn được chủ đề phù hợp	Không có
	Triệu Kim Dung Vũ Thị Hồng Oanh Vũ Ngọc Hân	-Thống nhất chủ đề BTL -Tìm hiểu nội dung của chủ đề	-Hiểu được yêu cầu đề tài và hướng tiếp cận	Không có
2	Cả nhóm	Tìm hiểu thuật toán Breadth-First Search và trò chơi Sokoban	Đã nắm được nguyên lý thuật toán BFS và luật chơi sokoban	Không có
	Nguyễn Thị Tuyết Nhung	Tạo bìa báo cáo, trình bày tổng quan về thuật toán tìm kiếm	Hoàn thành báo cáo chương 1	Không có
	Vũ Thị Hồng Oanh	Trình bày khái niệm, chức năng của bfs	Đã hoàn thành đầy đủ	Không có
	Triệu Kim Dung	Trình bày cấu trúc dữ liệu, ưu nhược điểm của bfs	Đã trình bày được	Không có
	Vũ Ngọc Hân	So sánh bfs với các thuật toán khác	So sánh được chi tiết được BFS với các thuật toán khác	Không có

3	Nguyễn Thị Tuyết Nhung Vũ Thị Hồng Oanh	Giới thiệu trò chơi Sokoban, các luật chơi, điều kiện trạng thái ầu, tổng hợp file word	Đã giới thiệu được căn bản về Sokoban và tổng hợp đầy đủ các bản word của thành viên trong nhóm	Không có
	Triệu Kim Dung Vũ Ngọc Hân	Trình bày không gian trạng thái của bài toán và các hành động hợp lệ, kiểm tra trạng thái đích	Đã hoàn thành nhiệm vụ được giao	Không có
4	Nguyễn Thị Tuyết Nhung Vũ Thị Hồng Oanh Vũ Ngọc Hân	Cài đặt chương trình	Đã cài đặt được thuật toán	Không có
	Triệu Kim Dung	Thử nghiệm trên bản đồ mẫu	Chạy thử thành công trên bản đồ mẫu	Không có
5	Vũ Ngọc Hân	Làm chương 3 kết luận	Đã đưa ra được kết luận về báo cáo	Không có
	Vũ Thị Hồng Oanh	Làm chương 3 hướng phát triển	Đưa ra được hướng phát triển sau khi hoàn thành báo cáo	Không có
	Nguyễn Thị Tuyết Nhung	Tổng hợp chỉnh sửa định dạng file word	Đã định dạng được và trình bày báo cáo theo khuôn mẫu	Không có
	Triệu Kim Dung	Đánh giá và bổ sung	Xem xét lại nội dung và đưa ra được đánh giá	Không có
6	Nguyễn Thị Tuyết Nhung Vũ Thị Hồng Oanh Vũ Ngọc Hân Triệu Kim Dung	Kiểm tra chéo các bài làm của nhau toàn bộ nội dung các chương và đưa ra nhận xét	Đã hoàn thành được các nội dung còn thiếu sót và hỗ trợ lẫn nhau	Không có

7	Triệu Kim Dung Vũ Ngọc Hân	Chỉnh sửa định dạng báo cáo hoàn thiện	Hoàn thành	Không có
	Nguyễn Thị Tuyết Nhung Vũ Thị Hồng Oanh	Tóm tắt nội dung để chuẩn bị trình bày báo cáo	Hoàn thành	

Ngày….tháng…..năm……

XÁC NHẬN CỦA GIẢNG VIÊN

(Ký, ghi rõ họ tên)

LỜI CẢM ƠN

Trước hết, nhóm 5 xin gửi lời cảm ơn chân thành và sâu sắc đến giảng viên bộ môn – Cô Mai Thanh Hồng đã nhiệt tình hướng dẫn, giúp đỡ và giảng dạy trong suốt quá trình thực hiện bài tập lớn này, những kiến thức quý báu và sự động viên của cô đã giúp chúng em hoàn thiện hơn, phát triển hơn trong quá trình nghiên cứu bài tập lớn và đạt được hiệu quả tốt nhất trong suốt quá trình vừa qua.

Đồng thời nhóm 5 chúng em cũng xin gửi lời cảm ơn chân thành tới khoa CNTT trường Đại Học Công Nghiệp Hà Nội và toàn thể các thầy cô giáo trong khoa đã cung cấp những kiến thức và tài liệu, giáo trình giúp chúng em dễ dàng hoàn thiện bài báo cáo đúng thời gian và cùng nhau làm việc hiệu quả nhất.

Với điều kiện thời gian có hạn cũng như kinh nghiệm còn hạn chế nên bài báo cáo sẽ không tránh khỏi những thiếu sót. Nhóm 5 rất mong nhận được sự chỉ bảo, đóng góp ý kiến của cô để nhóm em có điều kiện bổ sung, nâng cao kiến thức của mình, phục vụ tốt hơn công tác thực tế sau này.

Chúng em xin trân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI MỞ ĐẦU

Trong thời đại công nghệ số phát triển mạnh mẽ hiện nay, trí tuệ nhân tạo (AI) ngày càng đóng vai trò quan trọng trong nhiều lĩnh vực, đặc biệt là trong việc giải quyết các bài toán tự động hóa, ra quyết định và mô phỏng tư duy con người. Một trong những hướng tiếp cận nền tảng trong lĩnh vực này là việc áp dụng các thuật toán tìm kiếm để giải quyết các bài toán có không gian trạng thái lớn, nơi mà việc lựa chọn hành động tối ưu là một thách thức không nhỏ.

Thuật toán Breadth-First Search (BFS) – tìm kiếm theo chiều rộng – là một trong những thuật toán cơ bản và hiệu quả trong việc duyệt hoặc tìm kiếm trong đồ thị và cây. Với đặc tính duyệt theo từng lớp, BFS đảm bảo tìm ra lời giải ngắn nhất trong các bài toán không có trọng số. Do đó, thuật toán này thường được ứng dụng trong các trò chơi, bài toán đường đi, bài toán mê cung, và đặc biệt là trò chơi Sokoban – một trò chơi logic kinh điển đòi hỏi khả năng tư duy chiến lược để di chuyển các khối hộp về đúng vị trí mục tiêu trong không gian hẹp, là một ví dụ điển hình cho bài toán tìm kiếm trong không gian trạng thái phức tạp, có chứa ràng buộc chuyển động và đòi hỏi chiến lược đẩy hộp hợp lý – điều này khiến nó trở thành một môi trường lý tưởng để kiểm thử và đánh giá các thuật toán tìm kiếm.

Báo cáo này được thực hiện với mục tiêu tìm hiểu chi tiết thuật toán Breadth-First Search và ứng dụng thuật toán này vào việc giải bài toán trong trò chơi Sokoban. Qua đó, nhóm nghiên cứu mong muốn củng cố kiến thức lý thuyết, đồng thời nâng cao khả năng vận dụng các thuật toán AI vào các bài toán thực tiễn, góp phần phát triển tư duy thuật toán và kỹ năng lập trình.

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1 Tổng quan về thuật toán tìm kiếm trong trí tuệ nhân tạo.

Tìm kiếm là một kỹ thuật cốt lõi trong lĩnh vực trí tuệ nhân tạo, xử lý dữ liệu, và khoa học máy tính. Nhiều bài toán trong thực tế có thể được mô hình hóa dưới dạng không gian trạng thái, nơi mỗi trạng thái tương ứng với một tình huống cụ thể, và bài toán là tìm ra dãy hành động để chuyển từ trạng thái ban đầu đến trạng thái mục tiêu.

Dựa vào việc có hay không sử dụng thông tin về mục tiêu trong quá trình tìm kiếm, các thuật toán được phân thành hai nhóm chính: tìm kiếm không có thông tin (tìm kiếm mù) và tìm kiếm có thông tin (tìm kiếm theo chiến lược có hướng dẫn).

1.1.1 Tìm kiếm không có thông tin (Uninformed Search)

Thuật toán trong nhóm này không có kiến thức bổ sung nào về vị trí hay đặc điểm của trạng thái đích, ngoài mô hình không gian trạng thái và tập hành động có thể thực hiện. Do đó, việc duyệt trạng thái hoàn toàn dựa trên cấu trúc duyệt cố định, không ưu tiên.

Một số thuật toán tiêu biểu bao gồm:

- **Breadth-First Search (BFS)**:
  - Duyệt các trạng thái theo từng lớp (level).
  - Bắt đầu từ trạng thái gốc, sau đó mở rộng tất cả các trạng thái con ở mức tiếp theo trước khi đi sâu hơn.
  - Đảm bảo tìm được lời giải tối ưu (ngắn nhất) nếu tồn tại, trong đồ thị không trọng số.
  - Thường sử dụng hàng đợi (Queue) và tập visited để tránh lặp lại.
- **Depth-First Search (DFS)**:

- Mở rộng theo chiều sâu, đi sâu vào nhánh con đầu tiên cho đến khi gặp trạng thái đích hoặc ngõ cụt.
- Tiết kiệm bộ nhớ hơn BFS, nhưng không đảm bảo tìm được lời giải ngắn nhất.
- Có thể dùng đệ quy hoặc ngăn xếp (Stack).

- **Uniform-Cost Search (UCS)**:
  - Mở rộng nút có tổng chi phí đường đi từ gốc nhỏ nhất.
  - Là biến thể của BFS cho đồ thị có trọng số, sử dụng hàng đợi ưu tiên (priority queue)
  - Tìm lời giải tối ưu về mặt chi phí thay vì độ dài đường đi.

Nhóm thuật toán này thường được ưu tiên khi thông tin về không gian trạng thái rất hạn chế hoặc không thể ước lượng hướng đi tốt hơn.

1.1.2 Tìm kiếm có thông tin (Informed Search / Heuristic Search)

Tìm kiếm có thông tin sử dụng các hàm **heuristic (ước lượng)** để đoán trước mức độ "gần" trạng thái hiện tại với trạng thái đích, nhằm hướng dẫn quá trình tìm kiếm thông minh hơn, tránh duyệt không cần thiết.

Các thuật toán nổi bật gồm:

- **Greedy Best-First Search**:
  - Chọn trạng thái tiếp theo dựa trên giá trị của hàm ước lượng h(n) – khoảng cách dự đoán từ trạng thái hiện tại đến mục tiêu.
  - Nhanh nhưng không đảm bảo tìm được lời giải tối ưu.
- **A* (A-star)**:
  - Kết hợp chi phí đã đi g(n) và ước lượng chi phí còn lại h(n) thành hàm tổng f(n) = g(n) + h(n).
  - Khi h(n) là hàm ước lượng chấp nhận được (không vượt quá chi phí thực), A* đảm bảo tìm được lời giải tối ưu.

- Rất hiệu quả trong các bài toán tìm đường, lập kế hoạch, chơi cờ…

Nhóm này thường được sử dụng trong các ứng dụng thực tế có thể định nghĩa tốt hàm heuristic, giúp tăng tốc đáng kể quá trình tìm kiếm.

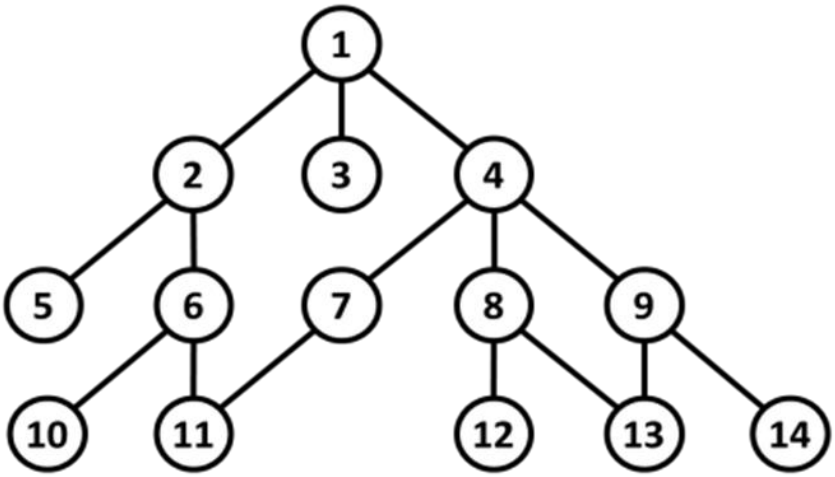
1.2 Thuật toán BFS

1.2.1 Khái niệm

Thuật toán duyệt đồ thị ưu tiên chiều rộng (Breadth-first search - BFS) là một trong những thuật toán tìm kiếm cơ bản và thiết yếu trên đồ thị. Mà trong đó, những đỉnh nào gần đỉnh xuất phát hơn sẽ được duyệt trước.

Breadth-first search là một trong những thuật toán của tìm kiếm mù (tìm kiếm không có thông tin). Tìm kiếm không có thông tin, còn gọi là tìm kiếm mù (blind, uninformed search) là phương pháp duyệt không gian trạng thái chỉ sử dụng các thông tin theo phát biểu của bài toán tìm kiếm tổng quát trong quá trình tìm kiếm, ngoài ra không sử dụng thêm thông tin nào khác. Tìm kiếm không có thông tin bao gồm một số thuật toán khác nhau. Điểm khác nhau căn bản của các thuật toán là ở thứ tự mở rộng các nút biên.

BFS là một dạng vét cạn, nguyên tắc của tìm kiếm theo chiều rộng là trong số những nút biên lựa chọn nút nông nhất (gần nút gốc nhất) để mở rộng. Thuật toán sẽ mở rộng từ những nút có độ sâu nông nhất trước(nút 0), sau đó đến các nút có độ sâu 1, rồi cứ tiếp tục như vậy.



Hình 1.1: Thứ tự duyệt các đỉnh của BFS

1.2.2 Chức năng của thuật toán BFS

Ứng dụng của nó có thể giúp ta giải quyết tốt một số bài toán trong thời gian và không gian tối thiểu. Đặc biệt là bài toán tìm kiếm đường đi ngắn nhất từ một đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số hoặc tất cả trọng số bằng nhau, thuật toán sẽ luôn trả ra đường đi ngắn nhất có thể. Ngoài ra, thuật toán này còn được dùng để tìm các thành phần liên thông của đồ thị, hoặc kiểm tra đồ thị hai phía, ...

Một số chức năng của thuật toán BFS:

- Tìm đường đi ngắn nhất trong đồ thị vô hướng và đồ thị có trọng số đều: Vì BFS duyệt các nút theo từng mức độ, đường đi đầu tiên tìm thấy đến một nút bất kỳ sẽ là đường đi có số cạnh ít nhất (ngắn nhất) từ nút nguồn. Điều này đặc biệt hữu ích trong các ứng dụng như định tuyến mạng, tìm đường đi trong trò chơi, v.v.

- Tìm kiếm trên đồ thị: BFS có thể được sử dụng để kiểm tra xem một nút cụ thể có tồn tại trong đồ thị hay không và nếu có, nó có thể cung cấp đường đi từ nút nguồn đến nút đó.

- Duyệt và khám phá tất cả các nút trong một thành phần liên thông: Bắt đầu từ một nút, BFS có thể duyệt qua tất cả các nút có thể truy cập được từ nút đó, tức là tất cả các nút trong cùng một thành phần liên thông. Điều này hữu ích trong việc phân tích cấu trúc của đồ thị.

- Kiểm tra tính liên thông của đồ thị: Bằng cách chạy BFS từ một nút bất kỳ, nếu thuật toán có thể truy cập đến tất cả các nút khác trong đồ thị, thì đồ thị đó là liên thông.

- Tìm chu trình trong đồ thị vô hướng: Mặc dù không phải là cách hiệu quả nhất, BFS vẫn có thể được điều chỉnh để phát hiện chu trình trong đồ thị vô hướng bằng cách theo dõi các nút đã được truy cập và phát hiện các cạnh quay lại (back edge) đến một nút đã được thăm.

- Tìm các thành phần liên thông của đồ thị: Bằng cách lặp lại BFS từ các nút chưa được thăm, chúng ta có thể xác định tất cả các thành phần liên thông riêng biệt trong một đồ thị.

- Ứng dụng trong các bài toán liên quan đến khoảng cách: Do tính chất tìm đường đi ngắn nhất (theo số cạnh), BFS được sử dụng trong các bài toán mà khoảng cách giữa các đối tượng được đo bằng số bước hoặc số kết nối trung gian.

- Ứng dụng trong trí tuệ nhân tạo

- Tìm kiếm lời giải trong không gian trạng thái: Trong các bài toán tìm kiếm như giải mê cung, giải các bài toán logic, BFS có thể được sử dụng để khám phá không gian trạng thái theo chiều rộng, đảm bảo tìm được lời giải có số bước ít nhất (nếu tồn tại).

- Thu thập dữ liệu web (Web Crawling): BFS có thể được sử dụng để duyệt các trang web theo chiều rộng, thu thập thông tin từ các trang được liên kết.

1.2.3 Nguyên lý của thuật toán

1.2.3.1 Tư tưởng của thuật toán

- 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 được đỉnh TG hoặc đã hết các đỉnh để duyệt.

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

+ DONG: chứa các đỉnh đã xét

+MO: chứa các đỉnh đang xét

1.2.3.2 Mô tả thuật toán

Thuật toán tìm kiếm theo chiều rộng hoạt động theo nguyên tắc hàng đợi FIFO(first in first out), ta chỉ cần lựa chọn những nút được thêm vào sớm hơn trong danh sách nút biên MO để mở rộng. Khi mở rộng một nút ta cần sử dụng con trỏ ngược để ghi lại nút cha của nút vừa được mở ra. Con trỏ này được sử dụng để tìm ngược lại đường đi về trạng thái xuất phát khi tìm được trạng thái đích. Khi cài đặt thuật toán, mỗi nút được biểu diễn bằng một cấu trúc dữ liệu có chứa một con trỏ ngược trỏ tới nút cha.

1.2.4 Cấu trúc dữ liệu và cài đặt của thuật toán

Thuật toán được thể hiện như sau: BFS (Q, T_o, T_G, P) // Q: tập trạng thái, T_o: trạng thái bắt đầu, T_G: đích, P: hành động

Đầu vào: bài toán tìm kiếm

Đầu ra: trạng thái đích

Khởi tạo: MO <- T_o//MO là tập hàng đợi FIFO

While(MO không rỗng) do:

1.Chọn nút đầu tiên n từ O và xóa khỏi O.

2.If n T_G , return (đường đi tới n)

3.Thêm P(n) vào cuối MO.

Return : không có lời giải.

Cài đặt:

1.MO = ∅ ; MO = MO ∪ {To};

2. while(MO!= ∅)

```
{       n=get(MO);
        if(n==TG)
                return true;
        DONG=DONG ∪ {n}
        for các đỉnh kề v của n
                if (v chưa được xét)
```

MO = MO ∪ {v}

father(v)=n

}

Giải thích thuật toán: Trước tiên cần khởi tạo tập biên MO, thêm T_o vào tập biên MO(To là điểm bắt đầu). Tập biên MO hoạt động tương tự như một hàng đợi.

Chúng ta sẽ lặp với điều kiện tập MO không rỗng. Lấy nút đầu tiên của MO ra, nếu đó là đích thì dừng vòng lặp trả về đường đi, nếu nó không phải đích thì ta sẽ tìm xem có những điểm nào mà điểm MO có thể mở rộng tới, tập P(n) sẽ được thêm vào cuối hàng đợi MO.

Lặp lại như vậy nếu khi hết hàng đợi mà không tìm được lời giải tức là bài toán không có lời giải.

Để tránh việc xem xét lại các nút đã mở rộng, không thêm nút vào hàng đợi nếu nút đã được duyệt hoặc đang nằm trong hàng đợi (tập các nút chờ được duyệt).

1.2.5 Ưu, Nhược điểm của thuật toán

- **Ưu điểm của thuật toán BFS:**
    - **Tính đầy đủ:** Nếu lời giải tồn tại, BFS đảm bảo tìm thấy lời giải đó (trong đồ thị có số lượng nút và cạnh hữu hạn).
    - **Tính tối ưu (đường đi ngắn nhất)**: BFS luôn tìm ra đường đi ngắn nhất (theo số cạnh) từ nút bắt đầu đến nút đích trong đồ thị vô hướng hoặc đồ thị có trọng số đều. Điều này rất hữu ích trong các ứng dụng như tìm đường đi trong mê cung, định tuyến mạng với chi phí mỗi bước nhảy là như nhau.
    - **Dễ cài đặt:** Thuật toán BFS tương đối đơn giản để hiểu và cài đặt, đặc biệt khi sử dụng hàng đợi (Queue).

- **Khám phá theo mức:** BFS khám phá các nút theo từng mức độ, điều này hữu ích trong việc tìm kiếm các giải pháp gần với trạng thái ban đầu nhất.
- **Ứng dụng rộng rãi:** BFS có nhiều ứng dụng trong các lĩnh vực khác nhau như tìm kiếm trên web (web crawling), mạng máy tính, trí tuệ nhân tạo (tìm kiếm trong không gian trạng thái), xử lý ảnh (tô màu vùng liên thông), v.v.

- **Nhược điểm của thuật toán BFS:**
  - **Độ phức tạp bộ nhớ cao:** BFS cần lưu trữ tất cả các nút ở mức độ hiện tại trong hàng đợi để khám phá các nút ở mức độ tiếp theo. Trong trường hợp xấu nhất, nếu không gian tìm kiếm rộng và độ sâu lời giải lớn, yêu cầu bộ nhớ có thể tăng theo hàm mũ ($O(b^d)$, với b là hệ số rẽ nhánh và d là độ sâu của lời giải). Điều này có thể dẫn đến tình trạng "hết bộ nhớ" trước khi tìm thấy lời giải.
  - **Độ phức tạp thời gian cao:** Tương tự như độ phức tạp bộ nhớ, trong trường hợp xấu nhất, thời gian chạy của BFS cũng có thể tăng theo hàm mũ ($O(b^d)$). Điều này làm cho BFS không phù hợp cho các bài toán có không gian tìm kiếm quá lớn.
  - **Không tối ưu cho đồ thị có trọng số khác nhau:** Nếu chi phí (trọng số) của các cạnh trong đồ thị khác nhau, BFS không đảm bảo tìm ra đường đi có tổng chi phí thấp nhất. Trong trường hợp này, các thuật toán như Dijkstra hoặc A* sẽ phù hợp hơn.
  - **Có thể chậm hơn DFS trong một số trường hợp:** Nếu lời giải nằm rất sâu trong không gian tìm kiếm và không gian tìm kiếm rất rộng, BFS có thể phải khám phá một lượng lớn các nút ở các mức độ nông hơn trước khi đến được lời giải, trong khi DFS có thể đi sâu vào một nhánh và tìm thấy lời giải nhanh hơn (mặc dù không đảm bảo là đường đi ngắn nhất).

1.3 So sánh BFS với các thuật toán khác

1. So sánh BFS và DFS (Depth-First Search)

Thuật toán tìm kiếm theo chiều rộng (BFS – *Breadth-First Search*) và tìm kiếm theo chiều sâu (DFS – *Depth-First Search*) là hai phương pháp cơ bản và phổ biến trong nhóm các thuật toán tìm kiếm không có thông tin. Cả hai đều được sử dụng rộng rãi trong việc duyệt đồ thị, cây, và giải các bài toán trí tuệ nhân tạo cơ bản. Tuy nhiên, hai thuật toán này có cách thức hoạt động, mức độ hiệu quả và phạm vi ứng dụng khác nhau đáng kể. Việc phân tích và đối chiếu chi tiết giữa BFS và DFS giúp đánh giá rõ hơn đặc điểm của từng thuật toán, từ đó lựa chọn phương án phù hợp với từng loại bài toán cụ thể.

Tiêu chí	BFS (Breadth-First Search)	DFS (Depth-First Search)
Cấu trúc dữ liệu hỗ trợ	Hàng đợi – theo nguyên tắc FIFO	Ngăn xếp – theo nguyên tắc LIFO hoặc dùng đệ quy
Cách mở rộng trạng thái	Mở rộng các trạng thái theo từng lớp, lần lượt từ gốc đến tất cả các nút con gần nhất	Mở rộng sâu nhất có thể theo một nhánh trước, sau đó quay lại nhánh khác
Trật tự duyệt	Theo bề rộng: từ trái sang phải	Theo chiều sâu: đi sâu vào từng nhánh một

Bảng 1.1: So sánh cơ chế hoạt động giữa BFS và DFS

Tiêu chí	BFS (Breadth-First Search)	DFS (Depth-First Search)
Tính đầy đủ	Đảm bảo tìm được lời giải nếu tồn tại và không gian hữu hạn	Không đảm bảo tìm được lời giải, có thể rơi vào vòng lặp vô hạn
Tính tối ưu	Đảm bảo lời giải tối ưu nếu chi phí các bước bằng nhau	Không đảm bảo tìm được lời giải tối ưu
Độ phức tạp thời gian	O(b^d) – với b là bậc phân nhánh, d là độ sâu lời giải	O(b^m) – với m là độ sâu tối đa của cây trạng thái
Độ phức tạp bộ nhớ	O(b^d) – tốn nhiều bộ nhớ	O(bm) – tiết kiệm bộ nhớ hơn đáng kể

Khả năng áp dụng	Phù hợp bài toán yêu cầu lời giải ngắn nhất, không gian trạng thái nhỏ	Phù hợp bài toán có không gian lớn nhưng lời giải không quá sâu
Ưu điểm nổi bật	- Tìm lời giải ngắn nhất - Đầy đủ	- Ít tốn bộ nhớ - Cài đặt đơn giản - Nhanh nếu lời giải ở nhánh đầu
Hạn chế chính	- Tốn bộ nhớ lớn - Có thể chậm khi không gian rộng	- Dễ rơi vào vòng lặp - Không tìm được lời giải tối ưu

Bảng 1.2: So sánh chi tiết giữa BFS và DFS

BFS và DFS đều là những thuật toán nền tảng, quan trọng trong các hệ thống tìm kiếm không thông tin. Tuy cùng xuất phát từ nguyên lý tìm kiếm tuần tự nhưng chúng thể hiện sự khác biệt rõ rệt cả về cách tiếp cận và hiệu quả xử lý. Tùy vào yêu cầu cụ thể của bài toán – chẳng hạn về tính tối ưu, giới hạn bộ nhớ, hoặc không gian trạng thái – mà người thiết kế hệ thống sẽ lựa chọn thuật toán phù hợp để đạt hiệu quả tối ưu nhất.

2. So sánh thuật toán BFS với các thuật toán Heuristic

- So sánh BFS với thuật toán A*

Khái quát thuật toán A*

A* là thuật toán tìm kiếm sử dụng hàm đánh giá f(n) = g(n) + h(n) trong đó:

g(n) là chi phí thực tế từ điểm xuất phát đến nút n.

h(n) là ước lượng chi phí còn lại từ n đến đích (heuristic).

A* hướng đến việc mở rộng các nút có giá trị f(n) thấp nhất, từ đó tối ưu hoá hành trình tìm kiếm, vừa đảm bảo hiệu quả, vừa giữ được tính tối ưu.

So sánh chi tiết

Tiêu chí	BFS (Breadth-First Search)	A* (A-Star)
Hướng tìm kiếm	Không định hướng	Có định hướng dựa trên hàm đánh giá
Tính tối ưu	Có, nếu chi phí mỗi bước bằng nhau	Có, nếu h(n) là admissible (không đánh giá vượt thực tế)
Tính đầy đủ	Có nếu không gian hữu hạn	Có nếu h(n) tốt và không gian hữu hạn
Độ phức tạp thời gian	O(b^d)	Phụ thuộc vào h(n) – tốt có thể nhanh hơn nhiều
Bộ nhớ sử dụng	Rất lớn, cần lưu tất cả các đỉnh ở mức sâu nhất	Cũng rất lớn, cần lưu toàn bộ frontier đang mở rộng
Khả năng mở rộng	Kém khi không gian trạng thái lớn	Tốt hơn, có thể điều chỉnh thông qua heuristic
Ưu điểm	Đảm bảo tìm lời giải ngắn nhất, dễ cài đặt	Kết hợp giữa hiệu quả và độ chính xác
Nhược điểm	Không linh hoạt, dễ bùng nổ bộ nhớ	Rất phụ thuộc vào chất lượng hàm heuristic

Bảng 1.3: So sánh giữa thuật toán BFS và A (A-star)*

Đánh giá:

A* là lựa chọn phù hợp cho các bài toán có không gian trạng thái lớn, trong khi BFS chỉ thích hợp với bài toán nhỏ hoặc cần tìm lời giải ngắn nhất mà không quan tâm đến chi phí tính toán. Khi có thể thiết kế được một hàm heuristic tốt, A* có thể vượt trội so với BFS cả về tốc độ lẫn tính thực tiễn.

- So sánh BFS với thuật toán Greedy (tìm kiếm tham lam)

Khái quát Greedy Search

Greedy Search là thuật toán tìm kiếm có thông tin, sử dụng hàm đánh giá f(n) = h(n) – chỉ xét đến khoảng cách còn lại đến đích, bỏ qua chi phí đã đi. Thuật toán

này ưu tiên mở rộng các trạng thái "có vẻ" gần đích nhất, giống như một người đi đường chỉ nhìn về phía trước mà không quan tâm đến đoạn đường đã đi qua.

So sánh chi tiết:

Tiêu chí	BFS (Breadth-First Search)	Greedy Search (Tìm kiếm tham lam)
Hướng tìm kiếm	Không có định hướng	Có định hướng, nhưng chỉ dựa vào h(n)
Tính tối ưu	Có (nếu chi phí đều)	Không đảm bảo
Tính đầy đủ	Có nếu không gian hữu hạn	Không đảm bảo – dễ rơi vào ngõ cụt
Tốc độ tìm kiếm	Chậm nếu không gian lớn	Nhanh nếu h(n) tốt
Bộ nhớ sử dụng	Tốn nhiều (lưu toàn bộ mức)	Thường ít hơn BFS
Ứng dụng thực tế	Bị giới hạn, khó mở rộng	Dùng trong các bài toán cần kết quả nhanh, không cần tối ưu
Ưu điểm	Đơn giản, dễ lập trình, luôn tìm ra lời giải ngắn nhất nếu tồn tại	Tốc độ nhanh, định hướng trực tiếp đến mục tiêu
Nhược điểm	Không định hướng, dễ xét thừa trạng thái	Dễ chọn sai đường đi, bỏ lỡ lời giải tốt hơn

Bảng 1.4: So sánh giữa thuật toán BFS và thuật toán tìm kiếm tham lam (Greedy Search)

1.4 Ứng dụng thực tiễn của BFS trong tin học

Thuật toán Breadth-First Search (BFS) không chỉ là một thuật toán cơ bản trong lĩnh vực trí tuệ nhân tạo và khoa học máy tính, mà còn có nhiều ứng dụng thực tiễn trong các lĩnh vực khác nhau. Nhờ khả năng duyệt theo chiều rộng và đảm bảo tìm được đường đi ngắn nhất trong đồ thị phi trọng số, BFS được áp

dụng rộng rãi trong các bài toán xử lý dữ liệu, trò chơi, mạng máy tính và hơn thế nữa.

1.4.1 Ứng dụng trong tìm đường (Pathfinding)

Một trong những ứng dụng nổi bật nhất của BFS là trong các hệ thống tìm đường. Thuật toán được dùng để tìm đường đi ngắn nhất trong mê cung, bản đồ lưới (grid), hoặc trong môi trường thực tế như hệ thống dẫn đường GPS (trong trường hợp đồ thị không có trọng số).

Ví dụ điển hình:

- Robot di chuyển trong không gian có chướng ngại vật.
- Tìm đường trong trò chơi (game) 2D như Pac-Man, Sokoban, hoặc các trò chơi dạng mê cung.
- Tìm đường trong bản đồ thành phố khi không xét trọng số (tức là khoảng cách không quan trọng bằng số bước đi).

1.4.2 Ứng dụng trong AI trò chơi

Trong lĩnh vực trí tuệ nhân tạo cho trò chơi (Game AI), BFS thường được dùng để giải quyết các trò chơi logic có không gian trạng thái rời rạc và hữu hạn. Khi biểu diễn trò chơi dưới dạng cây trạng thái, BFS giúp tìm lời giải ngắn nhất (tính theo số bước di chuyển) hoặc kiểm tra tính khả thi của một trạng thái.

Ví dụ trong các trò chơi:

- Sokoban: tìm chuỗi hành động để đẩy tất cả hộp về đúng vị trí đích.

Hình 1.2: Ví dụ về trò chơi Sokoban

- Chess puzzles: tìm dãy nước đi để chiếu bí đối phương trong số bước tối thiểu.
- Trò chơi rubik (phiên bản đơn giản): tìm chuỗi xoay ngắn nhất về trạng thái ban đầu.

1.4.3 Một số ví dụ khác

Ngoài tìm đường và AI trò chơi, BFS còn được ứng dụng trong các bài toán khác như:

- **Kiểm tra tính liên thông của đồ thị**: BFS giúp xác định xem tất cả các đỉnh trong đồ thị có được kết nối với nhau không.

- **Web crawling**: Khi xây dựng công cụ thu thập dữ liệu từ internet, BFS được dùng để lần lượt duyệt qua các trang web theo cấp độ liên kết.
- **Phân tích mạng xã hội**: BFS có thể dùng để tìm khoảng cách (số bước kết nối) giữa hai người trong mạng xã hội (gọi là "độ ngắn nhất của đường đi xã hội").
- **Giải bài toán chuỗi ký tự**: Ví dụ, biến đổi một chuỗi thành chuỗi khác thông qua một số phép biến đổi hợp lệ với số bước ít nhất.

CHƯƠNG 2: ỨNG DỤNG THUẬT TOÁN BFS VÀO GIẢI QUYẾT BÀI TOÁN SOKOBAN

2.1 Giới thiệu trò chơi Sokoban

2.1.1 Luật chơi cơ bản

Trong Sokoban, người chơi điều khiển một nhân vật trong một kho hàng, nơi họ phải đẩy các thùng hàng đến vị trí chỉ định. Điểm đặc biệt của trò chơi là người chơi chỉ có thể đẩy các thùng, không thể kéo chúng, và chỉ có thể đẩy một thùng tại một thời điểm. Điều này yêu cầu người chơi phải lên kế hoạch và tính toán cẩn thận từng động thái của mình để không làm kẹt thùng hàng trong góc hoặc tạo ra tình huống không thể giải quyết. Sokoban được biết đến với độ khó tăng dần qua mỗi cấp độ. Các bản đồ trở nên phức tạp hơn, yêu cầu người chơi phải sử dụng tư duy logic và kỹ năng giải quyết vấn đề để hoàn thành. Trò chơi không chỉ là một hình thức giải trí mà còn là một công cụ giáo dục, giúp cải thiện khả năng tư duy không gian và lập kế hoạch chiến lược. Sokoban có ảnh hưởng sâu rộng đến thế giới game giải đố. Nó đã được chuyển thể thành nhiều phiên bản khác nhau, từ các ứng dụng di động đến các phiên bản trực tuyến có thể chơi trên trình duyệt. Ngoài ra, các nguyên tắc của Sokoban cũng đã được áp dụng trong lĩnh vực trí tuệ nhân tạo, nơi các thuật toán được sử dụng để tự động giải các cấp độ của trò chơi, đóng góp vào nghiên cứu về tìm kiếm không gian trạng thái và lập kế hoạch.

Hình 2.1: Hình ảnh minh họa trò chơi Sokoban

2.1.2 Các yếu tố trong bản đồ Sokoban

Trong trò chơi Sokoban, bản đồ đóng vai trò là không gian trạng thái nơi người chơi tương tác và thực hiện các hành động đẩy hộp về vị trí đích. Bản đồ thường được biểu diễn dưới dạng ma trận hai chiều, trong đó mỗi ô (cell) tương ứng với một thành phần cụ thể của trò chơi. Việc hiểu và mô hình hóa chính xác các yếu tố trong bản đồ là điều kiện cần thiết để áp dụng các thuật toán tìm kiếm như BFS một cách hiệu quả.

Dưới đây là các yếu tố cơ bản trong bản đồ Sokoban:

- Tường (Wall)
  - Là các ô không thể di chuyển vào.
  - Cả người chơi lẫn hộp đều không thể đi xuyên qua hoặc đứng trên các ô tường.
  - Tường được dùng để giới hạn không gian bản đồ.
- Sàn trống (Floor)

- Là các ô có thể di chuyển vào.
- Người chơi có thể đi qua hoặc đứng trên ô sàn.
- Nếu không có ký hiệu đặc biệt, đây là các ô bình thường không có mục tiêu hay đối tượng nào.

- Vị trí đích (Target / Goal)
  - Là các vị trí mà người chơi cần đẩy hộp đến để hoàn thành màn chơi.
  - Khi hộp nằm đúng trên vị trí đích, nó có thể được hiển thị bằng ký hiệu khác (tuỳ phiên bản trò chơi).
- Hộp (Box)
  - Là đối tượng người chơi cần di chuyển.
  - Người chơi chỉ có thể đẩy hộp, không được kéo.
  - Một lần đẩy chỉ đẩy được một hộp và chỉ khi phía sau hộp là một ô trống hợp lệ.
- Người chơi (Player)
  - Là nhân vật điều khiển bởi người chơi hoặc thuật toán.
  - Có thể di chuyển lên, xuống, trái, phải.
  - Có thể đứng trên sàn trống hoặc ô đích, nhưng không được đi qua tường hoặc qua hộp trừ khi đẩy hợp lệ.

2.2 Mô hình hóa Sokoban thành bài toán tìm kiếm

2.2.1 Điều kiện của trạng thái đầu

- Số lượng hộp và đích

  Số lượng hộp phải bằng đúng số lượng vị trí đích.

  Nếu số lượng không khớp, sẽ không thể hoàn thành trò chơi.

- Vị trí ban đầu hợp lệ

Không có hộp nào được đặt tại vị trí bị kẹt vĩnh viễn ngay từ đầu (ví dụ: bị đẩy vào góc kín bởi hai bức tường, không thể thoát).

Người chơi và các hộp không nằm trên tường hoặc ngoài ranh giới bản đồ.

- Tính khả thi của việc di chuyển

  Từ vị trí ban đầu, phải tồn tại một chuỗi hành động hợp lệ (đẩy hộp từng bước) có thể dẫn đến trạng thái đích.

  Một hộp bị đẩy vào góc không có đường ra sẽ khiến trò chơi không thể tiếp tục, và đó là trạng thái không hợp lệ.

- Cấu hình bản đồ

  Bản đồ phải đảm bảo có đủ không gian di chuyển để người chơi tiếp cận các hộp và đẩy chúng đến mục tiêu.

  Cần có đường dẫn khả thi giữa người chơi và các hộp.

2.2.2 Không gian trạng thái của bài toán

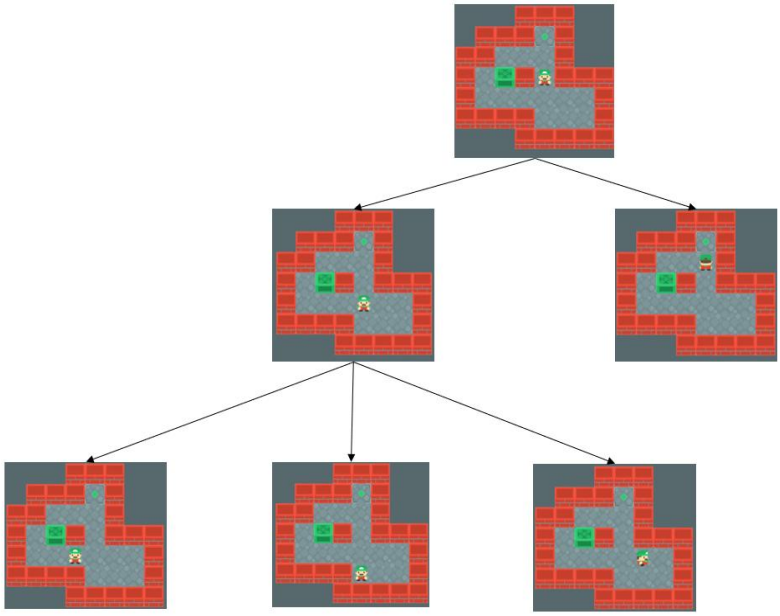
Không gian trạng thái của bài toán Sokoban là tập hợp tất cả các trạng thái hợp lệ mà trò chơi có thể đạt được từ trạng thái ban đầu thông qua các hành động hợp lệ như di chuyển người chơi hoặc đẩy hộp. Một trạng thái được xác định bởi vị trí của người chơi và vị trí của tất cả các hộp trên bản đồ. Không gian trạng thái này là cơ sở để áp dụng các thuật toán tìm kiếm trong giải bài toán Sokoban như BFS, DFS hay A*.

Một trạng thái trong Sokoban bao gồm:

- Vị trí của người chơi (player)
- Vị trí của tất cả các hộp (boxes)

- (Cấu trúc bản đồ – bao gồm tường, ô trống, và ô đích – được xem là bất biến)

→ Như vậy, một trạng thái là một cấu hình cụ thể của người chơi và các hộp trên bản đồ.



Hình 2.2: Hình mô tả không gian trạng thái của trò chơi

2.2.3 Các hành động hợp lệ

Trong bài toán Sokoban, không gian trạng thái được mở rộng thông qua các hành động mà người chơi có thể thực hiện. Hai loại hành động chính là: di chuyển đơn thuần và đẩy hộp. Các hành động chỉ hợp lệ nếu chúng không vi phạm cấu trúc bản đồ, không vượt ranh giới và không dẫn đến trạng thái không thể giải. Việc xác định rõ các hành động hợp lệ là bước rất quan trọng để xây dựng không gian tìm kiếm cho thuật toán BFS.

- Di chuyển

Người chơi có thể di chuyển một ô theo bốn hướng cơ bản: lên, xuống, trái, phải. Hành động này được sử dụng khi người chơi không tương tác trực tiếp với

hộp, mà chỉ di chuyển qua các ô trống để tiếp cận vị trí thuận lợi nhằm đẩy hộp sau đó.

- Ô mà người chơi muốn di chuyển tới phải là một ô trống có thể đứng được, bao gồm: Sàn (floor), vị trí đích (goal)
- Không có vật cản tại ô đó: không được chứa hộp, không phải là tường, không nằm ngoài ranh giới bản đồ

Vai trò của di chuyển:

- Giúp người chơi tiếp cận hộp từ hướng phù hợp để thực hiện hành động đẩy.
- Cho phép người chơi quay trở lại các khu vực khác trên bản đồ nếu cần thiết, tạo điều kiện cho việc tìm đường linh hoạt.

- Đẩy hộp

Điều kiện để một hành động đẩy là hợp lệ:

- Ô trước mặt người chơi (ô kế bên theo hướng di chuyển) phải chứa một hộp.
- Ô tiếp theo sau hộp (nơi hộp sẽ được đẩy đến) phải là một ô trống hợp lệ, tức là: không phải tường, không chứa 1 hộp khác, không nằm ngoài bản đồ
- Không có vật cản giữa người chơi và hộp (người chơi phải đứng ngay bên cạnh hộp theo hướng muốn đẩy).

Các ràng buộc:

- Chỉ được đẩy một hộp tại một thời điểm.
- Nếu đẩy hộp vào góc tường, hốc kín, hoặc vị trí không thể tiếp cận lại, hộp có thể bị "mắc kẹt", dẫn đến trạng thái không thể giải.

Hình 2.3: Mô tả các hướng đi của người chơi

2.2.4 Kiểm tra trạng thái đích

Trạng thái đích trong Sokoban là mục tiêu cuối cùng mà người chơi cần đạt được: tất cả các hộp phải được đặt chính xác lên các vị trí đích đã được xác định trước. Khi trạng thái này xảy ra, trò chơi được xem là hoàn thành thành công và thuật toán tìm kiếm có thể dừng lại.

Điều kiện xác định trạng thái đích hợp lệ:

- Số lượng hộp đúng bằng số lượng vị trí đích.
  Nếu thiếu hoặc thừa hộp/vị trí đích, bài toán ngay từ đầu đã không hợp lệ.
- Tất cả các vị trí đích đều có đúng một hộp được đặt lên.
  Vị trí của hộp phải khớp với vị trí đích về tọa độ.
- Không còn bất kỳ hộp nào nằm ngoài các vị trí đích.
  Hộp trên sàn bình thường không được chấp nhận trong trạng thái đích.

- Vị trí của người chơi không ảnh hưởng đến điều kiện thắng.

  Người chơi có thể đứng ở bất cứ đâu miễn là tất cả các hộp đã đúng chỗ.

2.3 Cài đặt giải thuật BFS cho Sokoban

2.3.1 Cài đặt bằng Python

1. Khởi tạo bản đồ và hằng số

```
from collections import deque

# Bản đồ
grid = [
    "#######",
    "#.....#",
    "#.###.#",
    "#.B.GP#",
    "#.....#",
    "#######"
]

ROWS, COLS = len(grid), len(grid[0])
```

- grid: ma trận bản đồ gồm tường “#” , đường trống “.”, hộp “B”, đích “G”, người chơi P.
- ROWS, COLS: số hàng và cột của bản đồ.

2. Phân tích bản đồ để lấy vị trí người, hộp và đích

```
# Tìm vị trí người chơi, các hộp, các đích
player_start = None
boxes_start = set()
goals = set()

for r in range(ROWS):
    for c in range(COLS):
        ch = grid[r][c]
        if ch == 'P':
            player_start = (r, c)
        elif ch == 'B':
            boxes_start.add((r, c))
        elif ch == 'G':
            goals.add((r, c))
```

- player_start: một tuple (r, c) chứa vị trí người chơi.
- boxes_start: một set gồm tất cả vị trí của hộp.
- goals: một set gồm tất cả vị trí của các đích.
- for r in range(ROWS):
  Lặp qua từng hàng trong bản đồ (grid), biến r là chỉ số dòng
- for c in range(COLS):
  Với mỗi dòng r, tiếp tục lặp qua từng cột, c là chỉ số cột
- Truy xuất ký tự tại vị trí (r, c) trong bản đồ grid
- Nếu là 'P': lưu tọa độ này làm **vị trí người chơi**
- Nếu là 'B': thêm vào **tập hợp vị trí hộp**
- Nếu là 'G': thêm vào **tập hợp đích**

3. Định nghĩa hướng đi

```
# 4 hướng: (delta_r, delta_c, tên hướng)
directions = [
    (-1, 0, 'Up'),
    (1, 0, 'Down'),
    (0, -1, 'Left'),
    (0, 1, 'Right')
]
```

- Danh sách các hướng người chơi có thể di chuyển: lên, xuống, trái, phải.
- Mỗi phần tử gồm (dx, dy, tên_hướng).

4. Hàm kiểm tra điều kiện thành công

```
def is_solved(boxes, goals):
    return boxes == goals
```

Hàm kiểm tra xem hộp đã được đẩy đến **tất cả vị trí đích** chưa.

5. Khởi tạo cấu trúc cho BFS

```
# BFS setup
visited = set()
start_state = (player_start, frozenset(boxes_start))
visited.add(start_state)
queue = deque()
queue.append((player_start, frozenset(boxes_start), 0))

# Lưu cha và hướng đi từ cha đến con
parent = {}
move_dir = {}  # map state → direction string
parent[start_state] = None
move_dir[start_state] = None

end_state = None
```

❖ visited = set()
- Dùng để lưu những trạng thái đã xét (đã đi qua) để tránh lặp lại, giúp BFS không bị rơi vào vòng lặp vô hạn.

❖ start_state = (player_start, frozenset(boxes_start))
- Mã hóa trạng thái đầu tiên:
  - Gồm player_start: tọa độ người chơi ban đầu.
  - Và boxes_start: vị trí các hộp → dùng frozenset để có thể hash, thêm vào set.

❖ visited.add(start_state)
- Đánh dấu trạng thái ban đầu là đã xét.

- ❖ queue = deque()
  - Tạo hàng đợi hiệu quả để lưu và xử lý các trạng thái trong thuật toán BFS.
- ❖ queue.append((player_start, frozenset(boxes_start), 0))
  - Tạo hàng đợi BFS.
  - Thêm trạng thái đầu tiên vào hàng đợi.
  - Mỗi phần tử trong queue có dạng: (người chơi, hộp, số bước đã đi).
- ❖ parent = {}
  - Lưu lại cha của mỗi trạng thái.
  - Dùng để truy ngược lại đường đi sau khi tìm ra trạng thái đích.
- ❖ move_dir = {}
  - Với mỗi trạng thái, lưu lại hướng di chuyển từ cha sang con.
  - Giúp khi truy đường đi có thể in ra
- ❖ parent[start_state] = None, move_dir[start_state] = None
  - Gán giá trị gốc:
    - Trạng thái ban đầu không có cha, cũng không có hướng.
- ❖ end_state = None
  - Khởi tạo biến để lưu trạng thái kết thúc khi tìm được lời giải, ban đầu chưa có.

6. Thuật toán BFS chính

```
while queue:
    player, boxes, steps = queue.popleft()

    if is_solved(boxes, goals):
        end_state = (player, boxes)
        print("Giải thành công sau", steps, "bước.")
        break
    for dr, dc, dir_name in directions:
        pr, pc = player[0] + dr, player[1] + dc
        if not (0 <= pr < ROWS and 0 <= pc < COLS):
            continue
        if grid[pr][pc] == '#':
            continue
        if (pr, pc) in boxes:
            # Người muốn đẩy hộp
            br, bc = pr + dr, pc + dc
            if not (0 <= br < ROWS and 0 <= bc < COLS):
                continue
            if grid[br][bc] == '#' or (br, bc) in boxes:
                continue
            new_boxes = set(boxes)
            new_boxes.remove((pr, pc))
            new_boxes.add((br, bc))
            new_state = ((pr, pc), frozenset(new_boxes))
            if new_state not in visited:
                visited.add(new_state)
                queue.append(((pr, pc), frozenset(new_boxes), steps + 1))
                parent[new_state] = (player, boxes)
                move_dir[new_state] = dir_name
        else:
            # Người đi bình thường
            new_state = ((pr, pc), boxes)
            if new_state not in visited:
                visited.add(new_state)
                queue.append(((pr, pc), boxes, steps + 1))
                parent[new_state] = (player, boxes)
                move_dir[new_state] = dir_name
else:
    print("Không tìm thấy đường đẩy hộp đến đích.")
```

- while queue:
  Vòng lặp chạy đến khi queue rỗng (tức là hết trạng thái cần duyệt).

- player, boxes, steps = queue.popleft()
  Lấy trạng thái ở đầu queue, gồm: vị trí người chơi, vị trí các hộp, số bước đã đi.

- if is_solved(boxes, goals):

  Kiểm tra xem tất cả hộp đã nằm đúng vị trí đích chưa. Nếu có:

  - Gán end_state để ghi nhớ trạng thái thành công.

    - In ra số bước và thoát vòng lặp.
- Duyệt 4 hướng đi:
  Với mỗi hướng, tính tọa độ mới của người chơi pr, pc.

- Kiểm tra điều kiện di chuyển:
    - Vị trí mới phải nằm trong bản đồ và không phải tường ('#').
    - Nếu vị trí mới là hộp, cần kiểm tra xem có thể đẩy hộp tiếp theo không (ô tiếp theo không phải tường, không có hộp khác).
- Cập nhật trạng thái mới:
    - Nếu đẩy hộp được, tạo trạng thái mới với vị trí mới của người và hộp.
    - Nếu đi bình thường, tạo trạng thái mới với vị trí mới của người và hộp giữ nguyên.
- Thêm trạng thái mới vào queue nếu chưa được xét:
    - Đồng thời lưu parent và move_dir để truy vết đường đi sau này.

7. Truy ngược đường đi và in kết quả

```
if end_state:
    directions_path = []
    state = end_state
    while state and move_dir[state] is not None:
        directions_path.append(move_dir[state])
        state = parent[state]
    directions_path.reverse()

    print("Đường đi:")
    print(" → ".join(directions_path))
```

- if end_state:

  Nếu đã tìm thấy lời giải (tức là hộp đã đến đúng đích).

- directions_path = []

  Tạo danh sách để lưu lại đường đi từ đầu đến cuối dưới dạng các chuỗi

- state = end_state

Bắt đầu truy ngược từ trạng thái cuối cùng.

- while state and move_dir[state] is not None:

Dừng khi đã về tới trạng thái bắt đầu (không còn cha hoặc không có hướng đi nữa).

- directions_path.append(move_dir[state])

Lấy hướng đi để đến trạng thái hiện tại, thêm vào danh sách.

- state = parent[state]

Di chuyển ngược về trạng thái cha.

- directions_path.reverse()

Vì ta đang truy ngược từ cuối về đầu, cần đảo ngược lại để đúng thứ tự di chuyển từ ban đầu đến lời giải.

- print("Đường đi:")

In tiêu đề đường đi.

- print(" → ".join(directions_path))

Nối các bước di chuyển lại bằng dấu "→" và in ra

2.3.2 Kết quả thử nghiệm trên bản đồ mẫu

Bản đồ mẫu:

```
# Bản đồ Sokoban
grid = [
    "#######",
    "#.....#",
    "#.###.#",
    "#.B.GP#",
    "#.....#",
    "#######"
]
```

Phân tích tọa độ:

- Người chơi P: (3,5)
- Hộp B: (3,2)
- Đích G: (3,4)

Kết quả:

```
Giải thành công sau 8 bước.
Đường đi:
Down → Left → Left → Left → Left → Up → Right → Right
```

Chi tiết các bước đi:

Giải thành công sau 8 bước.

Đường đi và loại hành động:

Bước 1: Down (Đi bình thường)

Bước 2: Left (Đi bình thường)

Bước 3: Left (Đi bình thường)

Bước 4: Left (Đi bình thường)

Bước 5: Left (Đi bình thường)

Bước 6: Up (Đi bình thường)

Bước 7: Right (Đẩy hộp)

Bước 8: Right (Đẩy hộp)

Phân tích lộ trình:

- Ban đầu người chơi ở (3,5) đi xuống (4,5) rồi đi sang trái, vòng quanh phía dưới hộp để tránh bị chặn bởi các bức tường ở hàng 2.
- Sau khi đến vị trí (3,1) người chơi di chuyển sang phải tới vị trí có hộp (3,2), lúc này người chơi bắt đầu đẩy hộp sang phải.
- Đẩy hộp 2 lần liên tiếp giúp đưa hộp vào vị trí đích (3,4), kết thúc bài toán.

Tóm lại:

- Đường đi tìm được là hợp lệ và tối ưu theo số bước đẩy hộp.
- Việc phân biệt rõ hành động đi bình thường và đẩy hộp giúp dễ dàng theo dõi quá trình giải bài toán và chứng minh thuật toán hoạt động chính xác.
- Kết quả này phù hợp với đặc điểm của bài toán Sokoban, trong đó người chơi cần phải di chuyển linh hoạt để tiếp cận vị trí hộp, sau đó thực hiện các bước đẩy hợp lý để đưa hộp vào vị trí đích.

CHƯƠNG 3: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

3.1 Kết luận

Trong quá trình thực hiện đề tài **Tìm hiểu thuật toán breadth-first search và ứng dụng vào giải quyết trò chơi sokoban**, nhóm chúng em đã từng bước tiếp cận, phân tích và triển khai giải pháp một cách hệ thống. Từ những kiến thức lý thuyết cơ bản đến thực hành xây dựng chương trình, nhóm đã có cơ hội hiểu rõ hơn về bản chất của thuật toán BFS cũng như vai trò quan trọng của nó trong giải quyết các bài toán tìm kiếm trạng thái.

Đề tài đã được thực hiện theo lộ trình hợp lý: bắt đầu từ việc tìm hiểu lý thuyết nền tảng (Chương I), phân tích bài toán Sokoban và cách áp dụng BFS (Chương II), đến giai đoạn cài đặt, thử nghiệm và đánh giá hiệu quả thuật toán trên nhiều bản đồ thực tế. Trong quá trình phát triển, nhóm đã bước đầu cài đặt thành công thuật toán tìm kiếm theo chiều rộng (BFS) để giải quyết bài toán Sokoban trên một số bản đồ. Dù chưa triển khai được đầy đủ các tính năng nâng cao như chơi tự động, bản đồ tùy chỉnh hay hỗ trợ hiển thị đường đi, nhóm đã tập trung vào việc đảm bảo thuật toán hoạt động đúng trên các trường hợp cơ bản.

Thông qua đề tài, nhóm đã rút ra được nhiều bài học quý báu:

- Việc hiểu rõ không gian trạng thái và tối ưu hóa dữ liệu là yếu tố then chốt giúp giải quyết bài toán hiệu quả.
  Cấu trúc dữ liệu phù hợp sẽ ảnh hưởng trực tiếp đến hiệu năng của thuật toán.
- Cuối cùng, nhóm cũng nâng cao được khả năng tư duy thuật toán, lập trình và làm việc nhóm – những kỹ năng cần thiết trong học tập và công việc sau này.

3.2 Hướng phát triển

Sau quá trình tìm hiểu và áp dụng thuật toán Breadth-First Search (BFS) vào trò chơi Sokoban, nhóm nhận thấy còn nhiều hướng phát triển tiềm năng để mở rộng, nâng cao chất lượng và ứng dụng của đề tài. Dưới đây là một số hướng đi cụ thể có thể triển khai trong tương lai:

❖ Tối ưu hóa hiệu năng thuật toán BFS

Thuật toán BFS trong phiên bản hiện tại hoạt động dựa trên cơ chế duyệt theo chiều rộng, đảm bảo tìm được lời giải ngắn nhất nếu tồn tại. Tuy nhiên, nhược điểm của BFS là tiêu tốn nhiều bộ nhớ và thời gian xử lý khi không gian trạng thái quá lớn, như trong các bản đồ Sokoban phức tạp. Trong tương lai, nhóm có thể cải tiến thuật toán theo các hướng sau:

➢ **Giới hạn độ sâu tìm kiếm**: Áp dụng BFS có giới hạn độ sâu để tránh việc duyệt quá nhiều trạng thái không cần thiết.

➢ **BFS hai chiều (Bidirectional BFS)**: Thay vì tìm kiếm từ trạng thái ban đầu đến trạng thái đích, có thể đồng thời tìm từ hai phía, giúp giảm đáng kể số trạng thái cần duyệt.

➢ **Kết hợp với heuristic đơn giản**: Mặc dù BFS không dùng heuristic như A*, nhưng có thể thử áp dụng một số kỹ thuật định hướng đơn giản (ví dụ: ưu tiên các trạng thái gần đích) nhằm cải thiện hiệu suất.

❖ So sánh và kết hợp với các thuật toán tìm kiếm khác

Một hướng đi quan trọng là mở rộng nghiên cứu để so sánh BFS với các thuật toán khác nhằm xác định giải pháp phù hợp nhất với trò chơi Sokoban. Các thuật toán như:

➢ **DFS (Depth-First Search)**: ít tốn bộ nhớ hơn nhưng không đảm bảo tìm lời giải ngắn nhất.

➢ **A*** và **IDA***: sử dụng heuristic để hướng dẫn quá trình tìm kiếm, thường hiệu quả hơn trong các trò chơi như Sokoban.

Việc đánh giá, thử nghiệm và so sánh các thuật toán này trên nhiều bản đồ với độ phức tạp khác nhau sẽ cung cấp cái nhìn toàn diện hơn và có thể định hướng lựa chọn giải pháp tối ưu nhất.

❖ **Tối ưu kiểm tra trạng thái trùng lặp**

Trong bài toán Sokoban, việc tránh duyệt lại các trạng thái đã thăm là yếu tố then chốt để tăng hiệu quả. Hiện tại, các trạng thái thường được lưu trong danh sách hoặc tập hợp đơn giản, dẫn đến việc kiểm tra trùng lặp đôi khi chưa tối ưu. Hướng phát triển trong tương lai có thể là:

➢ **Sử dụng Hashing**: Tận dụng các kỹ thuật hashing mạnh như Zobrist Hashing để biểu diễn trạng thái một cách duy nhất và nhanh chóng kiểm tra trùng lặp.

➢ **Tối ưu hóa lưu trữ trạng thái**: Thay vì lưu toàn bộ thông tin bản đồ, chỉ lưu vị trí người chơi và các hộp, từ đó tiết kiệm bộ nhớ.

❖ **Xây dựng giao diện mô phỏng và trực quan hóa thuật toán**

Việc trực quan hóa quá trình giải thuật không chỉ hỗ trợ việc học tập mà còn giúp đánh giá hiệu quả thuật toán rõ ràng hơn. Một số hướng phát triển:

➢ **Xây dựng giao diện đồ họa người dùng (GUI)**: Cho phép người dùng tự tạo bản đồ Sokoban, quan sát quá trình giải từng bước bằng BFS, có thể tạm dừng, tua nhanh hoặc tua lại.

➢ **Thể hiện lộ trình lời giải**: Hiển thị đường đi tối ưu từ trạng thái ban đầu đến đích, giúp người chơi hiểu rõ thuật toán hoạt động như thế nào.

❖ **Ứng dụng vào các bài toán thực tế tương tự**

Thuật toán BFS và các biến thể không chỉ giới hạn ở trò chơi Sokoban mà còn có thể áp dụng vào nhiều bài toán thực tế có cấu trúc tương tự. Một số ví dụ:

➢ **Điều khiển robot trong môi trường chật hẹp**: Robot cần đẩy hoặc di chuyển vật cản để đến đích, tương tự như Sokoban.

➢ **Bài toán tìm đường trong mê cung có vật thể di động**: BFS có thể xử lý tốt trong các bài toán tìm đường có nhiều ràng buộc về chuyển động.

➢ **Ứng dụng trong lập kế hoạch di chuyển (Path Planning)**: BFS là nền tảng cho các thuật toán nâng cao trong lĩnh vực trí tuệ nhân tạo và robot tự hành.

❖ **Nâng cấp và phát triển thành hệ thống học tập tương tác**

Cuối cùng, một hướng phát triển mang tính giáo dục là biến sản phẩm thành một **công cụ hỗ trợ giảng dạy thuật toán tìm kiếm**. Cụ thể:

➢ Cho phép sinh viên thử nghiệm với các thuật toán khác nhau, đánh giá hiệu năng thực tế.

➢ Tích hợp tính năng giải thích từng bước hoạt động của thuật toán.

➢ Tạo ra nền tảng học tập tương tác giúp sinh viên dễ dàng tiếp cận các thuật toán tìm kiếm thông qua trò chơi

TÀI LIỆU THAM KHẢO

[1]: Nguyễn Phương Nga (ch.b), Trần Hùng Cường, *Giáo trình trí tuệ nhân tạo*, Nhà xuất bản Thống kê, Hà Nội, 2021.

[2]: Nguyễn Đức Nghĩa, *Trí tuệ nhân tạo – Cơ sở và ứng dụng*, NXB Khoa học và Kỹ thuật, Hà Nội, 2019.

[3]: Nguyễn Tấn Trần Minh Khang. *Trí tuệ nhân tạo: Các phương pháp và ứng dụng*. NXB Đại học Quốc gia TP.HCM, 2016.