ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

fit@hcmus
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM
KHOA CÔNG NGHỆ THÔNG TIN

Nguyễn Tấn Duy Anh - Bùi Hồng Phúc
Nguyễn Lê Anh Phúc - Hồ Minh Quang

BÁO CÁO ĐỒ ÁN

CƠ SỞ TRÍ TUỆ NHÂN TẠO

ĐỒ ÁN 1: BÀI TOÁN TÌM KIẾM

Lớp: 22_21

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 11 NĂM 2024

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN
CƠ SỞ TRÍ TUỆ NHÂN TẠO
ĐỒ ÁN 1: BÀI TOÁN TÌM KIẾM

Môn học: Cơ sở trí tuệ nhân tạo
Mã môn học: CSC14003
Lớp: 22_21
Giảng viên hướng dẫn: GS. TS. Lê Hoài Bắc; CN. Nguyễn Thanh Tình
Các sinh viên tham gia:

Họ và tên	Mã số sinh viên
Nguyễn Tấn Duy Anh	22120015
Bùi Hồng Phúc	22120270
Nguyễn Lê Anh Phúc	22120276
Hồ Minh Quang	22120295

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 11 NĂM 2024

Lời cảm ơn

Em xin chân thành càm ơn thầy Trần Hoàng Quân đã hướng dẫn và giúp em hoàn thành bài tập này. Em rất mong được nhận sự góp ý từ thầy nếu có sai sót trong bài tập này.

Thành phố Hồ Chí Minh, tháng 11 năm 2024

Mục lục

Danh sách hình

1 Giới thiệu

1.1 Bảng phân công và đánh giá tiến độ công việc

Dưới đây là bảng phân công và đánh giá tiến độ công việc của nhóm:

Họ và tên	Các công việc phụ trách	Tiến độ công việc
Nguyễn Tấn Duy Anh	Cài đặt thuật toán BFS	100%
	Viết giới thiệu trò chơi và thuật toán BFS	100%
Bùi Hồng Phúc	Cài đặt thuật toán DFS	100%
	Thiết kế giao diện chương trình	100%
	Viết thuật toán DFS	100%
Nguyễn Lê Anh Phúc	Cài đặt thuật toán A*	100%
	Kiểm thử chương trình	100%
	Viết thuật toán A*	100%
Hồ Minh Quang	Cài đặt thuật toán UCS	100%
	Viết toàn bộ khung báo cáo LaTeX	100%
	Viết thuật toán UCS	100%

1.2 Bài toán tìm kiếm

Bài toán tìm kiếm là một trong những bài toán phổ biến trên nhiều lĩnh vực khác nhau với mục tiêu là tìm một cấu trúc "Y" (gọi là *lời giải*) trong một thực thể "X" nào đó. Một vài ví dụ về bài toán tìm kiếm có thể kể đến như: truy vấn thông tin trên cơ sở dữ liệu, giải đố mê cung, tìm đường đi, ... [1] [2]

Một bài toán tìm kiếm có thể được biểu diễn như sau: [3]

- **Không gian trạng thái:** Đây là thành phần chứa toàn bộ các trạng thái mà bài toán có thể phát sinh.
- **Trạng thái ban đầu:** Đây là điểm xuất phát của bài toán tìm kiếm.
- **Hàm trạng thái con:** Đây là hàm phát sinh trạng thái mới từ một trạng thái cụ thể x. Hàm này cùng với trạng thái ban đầu tạo nên không gian trạng thái cho bài toán.
- **Tập hợp trạng thái đích:** Đây là tập hợp con của không gian trạng thái, chứa các trạng thái để kết thúc bài toán. Bài toán kết thúc khi đạt được một hoặc nhiều trạng thái đích khác nhau trong tập hợp này, tùy theo ngữ cảnh của bài toán.
- **Hàm tính chi phí:** Đây là hàm tính toàn bộ chi phí đường đi cho lời giải của bài toán.

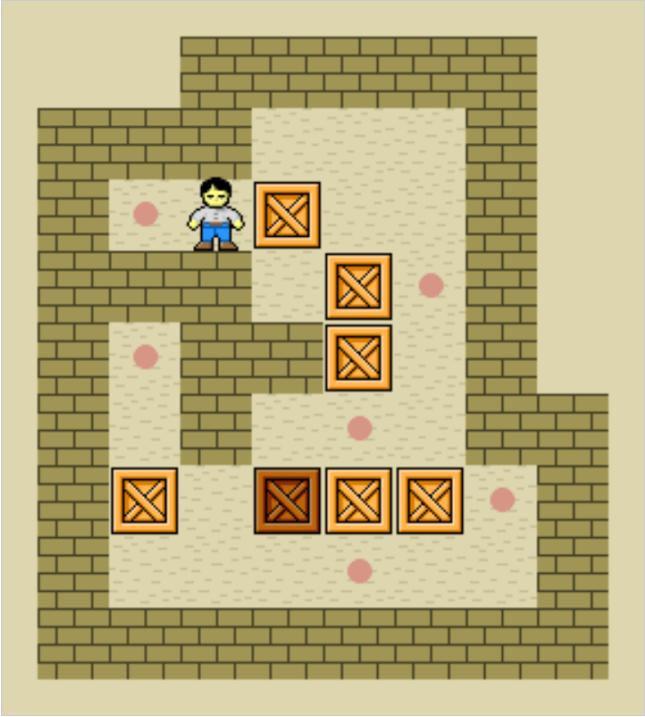
Các thuật toán tìm kiếm có thể được phân thành hai loại thuật toán: *Tìm kiếm không có thông tin* (Tìm kiếm "mù") và *Tìm kiếm có thông tin* (Tìm kiếm "tối ưu"). Dưới đây là một số đặc trưng và so sánh giữa hai loại thuật toán:

Loại thuật toán	**Tìm kiếm "mù"**	**Tìm kiếm "tối ưu"**
Hiệu suất	Chậm hơn do không có thông tin	Nhanh hơn
Tính xác định	Luôn có kết quả	Có thể có kết quả hoặc không
Thời gian	Thời gian chạy chậm hơn	Thời gian chạy nhanh hơn do có hàm tối ưu
Bộ nhớ	Sử dụng nhiều bộ nhớ hơn	Sử dụng bộ nhớ ít hơn do sử dụng hàm tối ưu
Phạm vi tìm kiếm	Phạm vi tìm kiếm lớn do không loại bỏ trạng thái xấu	Phạm vi tìm kiếm nhỏ nhờ loại bỏ trạng thái xấu thông qua hàm tối ưu
Một số thuật toán	Tìm kiếm theo chiều sâu (Depth-first search/DFS) Tìm kiếm theo chiều rộng (Breadth-first search/BFS) Tìm kiếm đồng nhất chi phí (Uniform-cost search/UCS)	Tìm kiếm tham lam (Greedy search) Tìm kiếm leo đồi (Hill-climbing search) Tìm kiếm A*

Bài toán tìm kiếm được đề cập trong báo cáo này là giải đố trò chơi Sokoban.

1.3 Trò chơi Sokoban

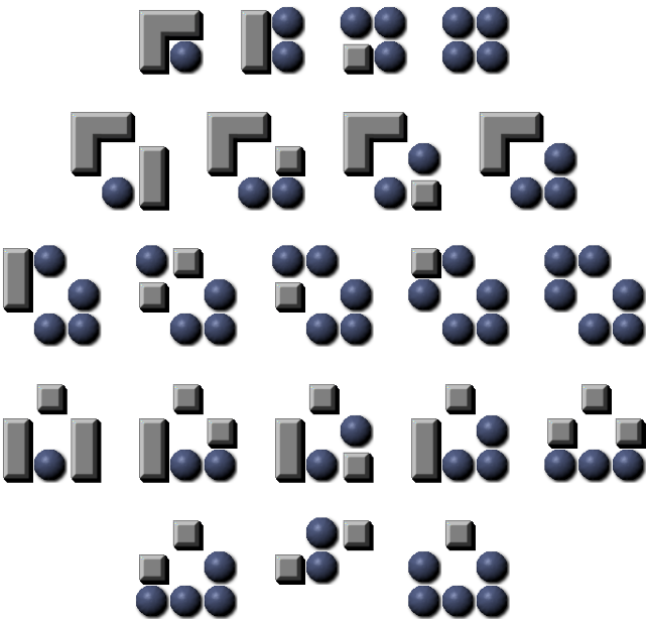
Sokoban là một trò chơi giải đố với mục tiêu là phải đẩy các thùng hàng trong kho vào các vị trí lưu trữ. Trò chơi này được thiết kế vào năm 1981 bởi Hiroyuki Imabayashi và ra mắt công chúng lần đầu vào tháng 12 năm 1982 bởi công ty Thinking Rabbit tại Takarazuka, Nhật Bản. [4]



Hình 1.1: Hình ảnh một màn chơi của Sokoban (Nguồn:)

Mô tả trò chơi: Nhà kho chứa các thùng hàng được biểu diễn bằng các ô trên bản đồ, mỗi ô biểu diễn một phần sàn hoặc tường của nhà kho. Một số ô sàn chứa các thùng hàng, một số khác là các vị trí lưu trữ. Người chơi chỉ có thể đẩy một thùng hàng khi đứng sát bên thùng hàng đó và không có tường hoặc thùng hàng khác chặn đường. Màn chơi hoàn thành khi tất cả các thùng hàng đều được đẩy vào vị trí lưu trữ.

Những khó khăn có thể gặp: Trong quá trình diễn ra trò chơi, nếu đẩy các thùng hàng không cẩn thận thì người chơi có thể rơi vào trạng thái bế tắc (có ít nhất một thùng hàng không thể di chuyển được) khi gặp phải một trong các hình như dưới đây [5]:



Hình 1.2: Các hình có thể gặp trong trạng thái bế tắc

Vì thế, việc đẩy các thùng hàng cần phải được tính toán hết sức cẩn thận, trừ phi trạng thái bắt đầu chỉ có thể dẫn đến một trong các trạng thái bế tắc.

1.4 Nội dung đồ án và chương trình của nhóm

Nội dung đồ án

Nội dung của đồ án lần này là viết một chương trình để giải đố các màn chơi của Sokoban biến thể. Chủ đề của trò chơi là mê cung kho báu với các tảng đá được dùng để kích hoạt các công tắc trong mê cung để mở cánh cổng dẫn tới kho báu. Người chơi đóng vai một nhà thám hiểm trẻ với tham vọng tìm toàn bộ kho báu được ẩn giấu phía trong. Nhiệm vụ của người chơi là tìm ra một cách giải (nếu có) cho mê cung kho báu.

Điểm khác biệt rõ rệt của biến thể này so với trò chơi truyền thống là mỗi tảng đá đều có khối lượng. Điều này gợi thêm yêu cầu về tính tối ưu cho lời giải của bài toán.

Chương trình giải của nhóm

Nhóm đã viết một chương trình bằng ngôn ngữ Python để tìm lời giải cho mê cung kho báu bằng các thuật toán tìm kiếm trên đồ thị sau: Tìm kiếm theo chiều sâu (DFS); Tìm kiếm theo chiều rộng (BFS); Tìm kiếm đồng nhất chi phí (UCS); Tìm kiếm A*. Các thành phần của chương trình này bao gồm:

1) Lớp không gian trạng thái: Toàn bộ không gian trạng thái được thiết lập bên trong tập tin `maze.py`, gồm hai lớp đối tượng `Node` và `SearchSpace`.

- Lớp `Node` được xây dựng để chứa một trạng thái (bộ hai biến `(agent_pos; stones_state_id)` đươc dùng để miêu tả một trạng thái, với `agent_pos` là vị trí của người chơi và `stone_state_id` là mã trạng thái trong danh sách các trạng thái của các tảng đá). Ngoài hai biến trên thì các biến `prev_state, steps, weight` và `move_label` lần lượt được dùng để lưu mã trạng thái đường đi trước đó, số bước di chuyển, khối lượng đá đã đẩy và ký tự đặc trưng cho hành động của người chơi.
- Lớp `SearchSpace` được dùng để khởi tạo trò chơi cũng như lưu các trạng thái và tác vụ của người chơi. Biến `stone_weights` được dùng để lưu khối lượng của từng tảng đá, `stones_state_list` là một mảng lưu các trạng thái của các tảng đá trong màn chơi, với `stones_state_list[0]` là trạng thái đầu tiên của chúng. Biến `start` lưu vị trí ban đầu của người chơi, còn `switches` là một mảng được dùng để lưu vị trí các công tắc trong mê cung.
- Ngoài ra, để phục vụ cho quá trình tìm kiếm, lớp `SearchSpace` còn có biến `row` để lưu số ô trong một dòng của màn chơi, biến `column` để lưu số ô trong một cột, `open_set` là một mảng lưu các `Node` đang mở và `closed_set` là mảng lưu các `Node` đã đóng.

2) Các bước đi của người chơi: Lớp `SearchSpace` có thiết lập các hàm tạo trạng thái mới từ các bước đi của người chơi, bao gồm:

- Kiểm tra nước đi cần thiết: Hàm `isRedundant` được dùng để kiểm tra nếu trạng thái đang xét là không cần thiết. Có năm tính chất được dùng trong hàm này để kiểm tra, bao gồm: `isLooped` được dùng để kiểm tra người chơi có đang đi trong một vòng lặp hay không, `isAlternativeMove` được dùng để kiểm tra nếu trạng thái đã tồn tại trước đó, `stoneInLoop` được dùng để kiểm tra nếu có một tảng đá bị đẩy ngược về một trạng thái trước đó, `isDeadlocked` được dùng để kiểm tra nếu trạng thái đang xét là một trong bốn trạng thái bế tắc đầu tiên ở hình 1.2 và kiểm tra nếu người chơi di chuyển xa ra khỏi các tảng đá mà không gặp bất kỳ chướng ngại vật nào.
- Thao tác di chuyển của người chơi: Các hàm `move_up, move_down, move_left, move_right` được dùng để khởi tạo và kiểm tra hợp lệ của các trạng thái di chuyển lên, xuống, trái, phải của người chơi. `move_label` tương ứng với các thao tác di chuyển trên lần lượt là `u, d, l ,r`.
- Thao tác di chuyển một tảng đá của người chơi: Các hàm `move_up, move_down, move_left, move_right` được dùng để khởi tạo và kiểm tra hợp lệ của các trạng thái di chuyển một tảng đá lên, xuống, trái, phải của người chơi. `move_label` tương ứng với các thao tác di chuyển trên lần lượt là `U, D, L ,R`.

3) Các thuật toán tìm kiếm: Toàn bộ thuật toán tìm kiếm kể trên được miêu tả trong các hàm `DFS, BFS, UCS` và `AStar`. Chi tiết thuật toán sẽ được miêu tả ở các phần 2, 3, 4 và 5. Đối với việc mở các trạng thái mới, lớp `SearchSpace` có hỗ trợ hàm `nodeExpansion` để mở các nước đi cần thiết và hợp lệ mới.

4) Giao diện: Chương trình có giao diện đi kèm, giúp cho người dùng sử dụng chương trình dễ dàng cũng như biểu diễn đường đi của lời giải một cách trực quan hơn. Giao diện sẽ được biểu diễn ở phần 6.

Về mã giả và mã nguồn của các thuật toán

Các thuật toán đều có cấu trúc mã nguồn khá tương tự nhau, do đó mã giả của chúng cũng sẽ tương tự nhau. Dưới đây là quy trình của một thuật toán tìm kiếm trong chương trình của nhóm:

1) Khởi tạo tập các trạng thái mở và thêm vị trí xuất phát: Trong mã nguồn của nhóm, mảng `open_set` trong lớp `SearchSpace` đã khởi tạo một mảng với `Node(start)` là phần tử đầu tiên. `Node(start)` là đối tượng lưu trữ trạng thái đầu tiên của màn chơi.

2) Chọn một trạng thái đang mở và đóng nó lại: Trong mã nguồn của nhóm, mảng `closed_set` trong lớp `SearchSpace` được dùng để lưu các trạng thái đã được đóng. Sau khi chọn được một trạng thái, trạng thái này sẽ được loại bỏ ra khỏi tập các trạng thái mở (`open_set`) và thêm vào tập các trạng thái đóng (`closed_set`). Tiêu chí chọn trạng thái tùy thuộc vào tính chất của thuật toán và sẽ được trình bày ở phần phía dưới.

3) Kiểm tra nếu trạng thái đích đã đóng: Việc kiểm tra này sẽ cho biết nếu đã có một lời giải thỏa mãn. Dòng mã `if goalReached(game.closed_set[-1])` sẽ đảm nhiệm việc này. Nếu chưa có trạng thái đích, thuật toán sẽ tiếp tục bước **4**. Ngược lại, thuật toán trả về một lời giải.

4) Mở các trạng thái mới từ trạng thái vừa đóng: Việc mở các trạng thái mới sẽ tương ứng với các nước đi của người chơi trong trò chơi. Dòng mã `new_nodes = game.nodeExpansion(...)` sẽ đảm nhiệm việc đóng trạng thái và mở các nước đi mới hợp lệ và cần thiết. Việc thêm trạng thái vào tập các trạng thái mở sẽ tùy thuộc vào thuật toán.

5) Kiểm tra nếu còn trạng thái mở: Nếu không còn trạng thái nào được mở, kết thúc thuật toán. Ngược lại, thuật toán lặp lại từ bước **2**.

Mã giả cho quy trình trên được biểu diễn như sau:

```
thuật_toán_tìm_kiếm(màn_chơi):
    tập_mở ← [trạng_thái_đầu]
    tập_đóng ← []
    khi tập_mở còn trạng thái:
        trạng_thái_đóng_mới ← chọn_trạng_thái(tiêu_chí)
        thêm trạng_thái_đóng_mới vào tập_đóng
        nếu trạng_thái_đóng_mới là đích:
            lời_giải ← biểu_diễn_lời_giải(trạng_thái_đóng_mới)
            trả về lời_giải
        các_trạng_thái_mới ← mở_rộng_trạng_thái(trạng_thái_đóng_mới)
        nếu có trạng thái trong các_trạng_thái_mới:
            thêm các_trạng_thái_mới vào tập_mở
        nếu không có trạng thái trong tập_mở:
            trả về "Không có lời giải!"
```

2 Tìm kiếm theo chiều sâu (DFS)

2.1 Chi tiết về thuật toán

Thuật toán tìm kiếm theo chiều sâu cho phép tìm kiếm các trạng thái dựa theo một hướng nào đó, đến khi hướng đó không thể tìm kiếm được nữa thì thuật toán mới đổi hướng và thực hiện tiếp đến khi có lời giải hoặc không còn trạng thái nào được mở [3] [6]. Quy trình thực hiện sẽ tương tự như quy trình được trình bày ở phần 1.4, với tập các trạng thái mở là một ngăn xếp [3], và tiêu chí chọn trạng thái kế tiếp để mở là chọn trạng thái đầu tiên. Dưới đây là mã giả của thuật toán:

thuật_toán_DFS(màn_chơi):
  tập_mở ← [trạng_thái_đầu]
  tập_đóng ← []
  khi tập_mở **còn trạng thái**:
    trạng_thái_đóng_mới ← **chọn_trạng_thái**(đầu tiên)
    thêm trạng_thái_đóng_mới **vào** tập_đóng
    nếu trạng_thái_đóng_mới **là đích**:
      lời_giải ← **biểu_diễn_lời_giải**(trạng_thái_đóng_mới)
      trả về lời_giải
    các_trạng_thái_mới ← **mở_rộng_trạng_thái**(trạng_thái_đóng_mới)
    nếu có trạng thái trong các_trạng_thái_mới:
      thêm các_trạng_thái_mới **vào đầu** tập_mở
    nếu không có trạng thái trong tập_mở:
      trả về "Không có lời giải!"

2.2 Ưu, nhược điểm của thuật toán

3 Tìm kiếm theo chiều rộng (BFS)

4 Tìm kiếm đồng nhất chi phí (UCS)

5 Tìm kiếm A*

6 Kết quả kiểm thử chương trình

7 So sánh các thuật toán tìm kiếm

Tài liệu tham khảo

Sách

[3] Lê Hoài Bắc, Tô Hoài Việt. *Giáo trình Cơ sở trí tuệ nhân tạo*. Thành phố Hồ Chí Minh: Nhà xuất bản Khoa học và Kỹ thuật, 2014.

[6] Trần Ngọc Danh. *Toán rời rạc nâng cao*. Thành phố Hồ Chí Minh: Nhà xuất bản Đại học Quốc gia Thành phố Hồ Chí Minh, 2004.

Internet

[1] *Search problem - Wikipedia*. URL: https://en.wikipedia.org/wiki/Search_problem (visited on 10/31/2024).

[2] *Graph Search*. URL: https://www.cs.ubc.ca/~kevinlb/teaching/cs322%20-%202009-10/Lectures/Search2.pdf (visited on 11/01/2024).

[4] *Sokoban - Wikipedia*. URL: https://en.wikipedia.org/wiki/Sokoban (visited on 11/01/2024).

[5] Timo Virkkala. *Solving Sokoban - Master Thesis*. Apr. 12, 2011. URL: https://sokoban.dk/wp-content/uploads/2016/02/Timo-Virkkala-Solving-Sokoban-Masters-Thesis.pdf.