Mô phỏng các thuật toán Uninformed Search trong trò chơi Sokoban

Bùi Minh khoa

Tháng 3/2024

1 Mô hình hoá trò chơi Sokoban như thế nào?

Sokoban là trò chơi giải đố logic trong đó người chơi phải điều khiển một nhân vật đi khắp mê cung để đưa các thùng hàng về vị trí yêu cầu.

Cụ thể hơn trong báo cáo này, mục tiêu là tạo ra một chương trình máy tính có thể tự động giải 18 màn chơi Sokoban từ dễ đến khó.

Mỗi màn chơi sẽ được cấu trúc thông qua các kí tự để đưa vào chương trình máy tinh. Cụ thể, **"#"** là bức tường, **"B"** là các thùng hàng, **"."** là đích đến của các thùng hàng, " " là khoảng trống và "&" là vị trí hiện tại của nhân vật ta cần điều khiển. "state space" chính là thứ ta input vào các thuật toán và nó chính là vị trí của nhân vật và các thùng hàng. Các hạnh động trong Sokoban bao gồm up, down, left và right, đồng thời cũng sẽ có hàm kiểm tra xem hành động có hợp lệ hay có dẫn đến việc đẩy các thùng hàng vào góc kẹt khiến vấn đề không thể giải được. Hành động của Sokoban khi di chuyển trong khoảng trống sẽ được biểu diễn bằng các chữ cái viết thường: 'l', 'r', 'u', 'd' và sẽ được viết hoa khi di chuyển đồng thời di chuyển các thùng hàng: 'L', 'R', 'U', 'D'.

Vị trí của tất cả vật thể có trong trò chơi sẽ được biểu diễn bằng một tuple (y, x), theo với trục x hướng từ dưới lên trên, trục y hướng từ trái sang phải. Khi đưa các file .txt vào, sẽ có các hàm chuyên dụng để chuyển các kí tự đã nêu trên thành các số, đồng thời có các hàm trả ra toạ độ của tất cả vật thể theo loại.

Cụ thể, trạng thái khởi đầu là vị trí ban đầu của tất cả vật thể khi chưa di chuyển, ở đây chính file .txt của mỗi màn chơi là trạng thái bắt đầu cho mỗi màn chơi đó. Trạng thái bắt đầu này sẽ được đưa vào các hàm thành phần với tên gọi gameState. Kế tiếp, ta định nghĩa trạng thái kết thúc của trò chơi chính là khi tất cả các thùng đều nằm vào đúng đích. Không gian trạng thái ở đây chính là tất cả trạng thái có thể có của mỗi màn chơi, hay có thể nói là tập hợp tất cả sắp xếp có thể có được của các vật thể ta đã định nghĩa dựa trên gameState của mỗi màn chơi. Ta sẽ dựa trên các không gian trạng thái này để đi từ trạng thái bắt đầu, tìm ra trạng thái kết thúc.

Để có thể tìm được lời giải cho trò chơi, từ một trạng thái có sẵn, ta phải xác định được các hành động hợp lệ tiếp theo mà nhân vật có thể thực hiện, xem như ràng buộc cho nhân vật để có thể hoàn thiện việc xây dựng bài toán của trò chơi. Cụ thể, ta xây dựng hàm legalActions với input đầu vào là vị trí của người chơi và vị trí các thùng hàng và hàm isLegalAction với input là action, vị trí nhân vật và vị trí thùng hàng. Dựa vào việc đã định nghĩa toạ độ của nhân vật là một tuple (y, x) với x là trục có hướng từ trái sang phải, y là trục có hướng từ dưới lên trên, ta sẽ liệt kê ra 4 hướng có thể di chuyển là up, down, left và right. Hàm isLegalAction sẽ kiểm tra hành động hợp lệ bằng cách thực hiện hành động đó và kiểm tra xem vị trí mới có trùng với thùng hàng hay tường hay không.

2 Phương pháp để giải quyết bài toán đặt ra trong Sokoban

2.1 Xem bài toán cần giải quyết là bài toán tìm kiếm

Bài toán ta cần giải để phá đảo trò chơi Sokoban chính là bài toán tìm đường đi dành cho nhân vật để có thể đẩy các thùng hàng vào đúng vị trí đích mà không bị kẹt. Với bản chất là một bài toán tìm kiếm, ta sẽ làn lượt áp dụng 3 thuật toán tìm kiếm không có thông tin, áp dụng cụ thể vào việc tìm đường đi. Node gốc của thuật toán tìm kiếm trong trường hợp này sẽ là trạng thái bắt đầu, còn các node con sẽ

lần lượt là các trạng thái có thể thực hiện được từ trạng thái gốc.

Chiến lược lựa chọn các node tiếp theo trong hàng đợi từ 1 node gốc cũng gây ảnh hưởng đến độ hiệu quả của việc tìm kiếm, hàm để thực hiện chiến lược lựa chọn này là hàm tiến triển - Successor function. Cụ thể ta sẽ quan sát 3 thuật toán Depth First Search, Breadth First Search và Uniform Cost Search với 3 hàm Successor function khác nhau, sẽ được nói cụ thể hơn bên dưới.

2.2 Depth First Search

Depth first search (DFS) là thuật toán tìm kiếm trên cấu trúc dữ liệu cây hoặc đồ thị. Thuật toán này bắt đầu từ một đỉnh (hoặc nút) khởi đầu và khám phá càng xa càng tốt theo mỗi nhánh trước khi quay lại.

Về Hàm tiến triển - Successor function của thuật toán DFS, chiến lược lựa chọn node để tìm đường đi kế tiếp sẽ theo nguyên tắc là luôn tìm các node sâu về một nhánh trước khi quay lại các node nông hơn. Ta sẽ cài đặt hàm này dựa trên ý tưởng của cấu trúc dữ liệu stack, cụ thể khi ta làm việc xong với một node, các node có nguồn gốc từ node vừa bung ra sẽ được cho vào ngay phía sau trong stack để tiếp tục bung các node vừa tìm được này.

2.3 Breadth first search

Breadth first search (BFS) là một thuật toán tìm kiếm trong đồ thị, hoạt động dựa trên nguyên tắc khám phá các đỉnh kề nhau theo từng lớp.

Về hàm tiến triển - Successor function của thuật toán BFS, chiến lược lựa chọn node để tìm đường đi kế tiếp sẽ theo nguyên tắc lần lượt bung hết tất cả các node ở cùng một tầng trước khi bung tầng kế tiếp. Thuật toán này được cài đặt bằng cấu trúc dữ liệu queue, cụ thể, ta sẽ lấy các node ở đầu hàng đợi để bung, và thêm các node vào sau hàng đợi. Nhờ vào ý tưởng này có thể hiện thực hoá được BFS.

2.4 Uniform cost search

Uniform cost search (UCS) là một thuật toán tìm kiếm trong đồ thị, hoạt động dựa trên nguyên tắc tìm kiếm đường đi có chi phí tổng thể thấp nhất từ đỉnh khởi đầu đến đỉnh đích.

Về hàm tiến triển - Successor function của thuật toán UCS, chiến lược chọn node tiếp theo để bung ra sẽ là node có cost nhỏ nhất. Trong ngữ cảnh bài toán Sokoban này, ta thiết kế cost là tổng số bước đi tới thùng. Về phần cài đặt, ta sử dụng cáu trúc dữ liệu heap để có thể luôn trả về node theo yêu cầu, ở đây là node có cost nhỏ nhất.

2.5 So sánh 3 thuật toán và nhận định về thuật toán phù hợp nhất cho trò chơi Sokoban

Bảng 1: So sánh hiệu suất của DFS, BFS, UCS

Level	DFS	BFS	UCS
1	0.08s, 79	0.11s, 12	0.08s, 12
2	0.01s, 24	0.01s, 9	0.01s, 9
3	0.25s, 403	0.20s, 15	0.12s, 15
4	0.00s, 27	0.01s, 7	0.01s, 7
5	error	172.79s, 20	53.45s, 20
6	0.02s, 55	0.02s, 19	0.03s, 19
7	0.55s, 707	0.89s, 21	0.60s, 21
8	0.10s, 323	0.23s, 97	0.25s, 99
9	0.31s, 74	0.02s, 8	0.02s, 8
10	0.03s, 37	0.03s, 34	0.02s, 33
11	0.03s, 36	0.03s, 34	0.03s, 34
12	0.16s, 109	0.12s, 23	0.10s, 23
13	0.21s, 185	0.18s, 31	0.20s, 31
14	3.91s, 865	2.79s, 23	2.78s, 23
15	0.20s, 291	0.30s, 105	0.30s, 105
16	error	23.28s, 34	13.87s, 34
17	24.51s, 0	25.21s, 0	error
18	error	error	error

Giá trị đầu tiên là thời gian cần để chạy màn chơi trên máy tính thực nghiệm, giá trị thứ hai là số bước cần để thuật toán tìm ra lời giải.

Qua bảng thống kê trên, ta thấy được hiệu suất của mỗi thuật toán. Với DFS, nhìn chung thời gian chạy cũng như số bước đi của DFS là nhiều hơn đáng kể, đặc biệt có những màn chơi DFS phải thực hiện số bước rất lớn. BFS và UCS cho ra kết quả tìm đường đi khá tương đồng nhau, song do tính chất của BFS tìm kiếm tất cả node của một tầng trước khi đi tới tầng tiếp theo nên có những màn chơi tốn nhiều thời gian hơn đáng kể so với UCS. Vậy, UCS chính là thuật toán tìm kiếm tối ưu nhất cả về thời gian lẫn bộ nhớ trong cả 3.

Màn chơi khó nhất là màn số 18, khi cả 3 thuật toán đều tốn rất nhiều thời gian nhưng không thể tìm ra được thời giải.