ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA: KHOA HỌC MÁY TÍNH

MÔN: TRÍ TUỆ NHÂN TẠO - CS106.O21.KHTN

BT1 - DFS/BFS/UCS for Sokoban

Sinh viên thực hiện:
Tô Anh Phát - 21520085

Mục lục

1 Giới thiệu tổng quan

1.1 Yêu cầu

- Cài đặt 2 hàm breadthFirstSearch(gameState) và uniformCostSearch(gameState) trong file solver.py.
- Để thay đổi thuật toán chạy thì cần comment/uncomment các dòng lệnh tương ứng trong hàm auto_move() trong file game.py.

1.2 Source code

Link Gtihub: https://github.com/pattan99/CS106.O21.KHTN/tree/main/sokoban

2 Mô hình hóa Sokoban

Sokoban là một trò chơi dạng câu đố, trong đó người chơi phải đẩy các khối vuông qua chướng ngại vật để đến đích. Trò chơi này đã được thiết kế vào năm 1981 bởi Hiroyuki Imabayashi và ra mắt lần đầu vào tháng 12 năm 1982. Trò chơi Sokoban được tham số hóa như sau:

- Vị trí của người chơi: **PosOfPlayer** $= (x, y)$
- Vị trí các khối vuông: **PosOfBoxes** $= set\,(x, y)$
- Vị trí các đích: **PosOfGoals** $= set\,(x, y)$
- Vị trí các bức tường: **PosOfWalls** $= set\,(x, y)$

với (x, y) là tọa độ trên bản đồ.
Sokoban được mô hình hóa như sau:

- **Trạng thái khởi đầu**: Vị trí của người chơi và các khối vuông: [(posOfPlayer, posOfBoxes)]
- **Trạng thái kết thúc**: Tất cả các hộp đã được đẩy vào các ô đích trên bản đồ: posOfBoxes = PosOfGoals
- **Không gian trạng thái (S)** : $\forall$ [(PosOfPlayer, PosOfBoxes)]
- **Các hành động hợp lệ**: Thực hiện các hành động *Up, Down, Left, Right* để di chuyển người chơi trong bản đồ. Người chơi chỉ có thể di chuyển nếu ô tiếp theo không phải là tường và nếu có thể đẩy một hộp mà không gây kẹt.
- **Hàm tiến triển (successor function)**: Các trạng thái tiếp theo đạt được từ một trạng thái hiện tại bằng cách thực hiện một hành động hợp lệ: di chuyển người chơi hoặc đẩy 1 hộp.

3 Thống kê độ dài đường đi của các thuật toán

Level	DFS	BFS	UCS
1	79	12	12
2	24	9	9
3	403	15	15
4	27	7	7
5	**Chạy không ra**	20	20
6	55	19	19
7	707	21	21
8	323	97	97
9	74	8	8
10	37	33	33
11	36	34	34
12	109	23	23
13	185	31	31
14	865	23	23
15	291	105	105
16	**Chạy không ra**	34	34
17	**Không có lời giải**	**Không có lời giải**	**Không có lời giải**

Bảng 1: Bảng thống kê về độ dài đường đi

4 Nhận xét

Ta có nhận xét về các thuật toán ở bảng 1 như sau:

- Thuật toán **DFS** đảm bảo tìm được lời giải nếu có tồn tại lời giải (do có hữu hạn các trạng thái và sử dụng tập đóng *exploredSet*). Tuy nhiên, các lời giải đó không tối ưu (đường đi dài dòng). Level 5 và 16 là trường hợp DFS đi sâu vào các nhánh không có lời giải hoặc lời giải ở quá xa, từ đó dẫn đến tràn bộ nhớ và tốn thời gian khi đi vào các nhánh đó.

- Thuật toán **BFS** luôn tìm được lời giải tối ưu (do độ dài các đường đi đều bằng 1). Tuy nhiên, thuật toán BFS sẽ tốn nhiều bộ nhớ hơn DFS đối với các không gian trạng thái lớn bởi vì phải mở tộng tất cả các nút cùng cấp.

- Thuật toán **UCS** luôn tìm được lời giải tối ưu trong thời gian ngắn hơn BFS trong đa số trường hợp (do thiết kế hàm cost tốt từ đó ưu tiên đi vào các đường đi tối ưu).

Tóm lại, trong cả ba thuật toán, DFS thì tệ nhất về thời gian và độ dài đường dài ở đa số trường hợp. Ngoài ra, cả BFS và UCS đều cho ra lời giải tối ưu. Tuy nhiên, đa số các màn chơi thì UCS chạy nhanh hơn BFS và phần nhỏ các màn chơi thì BFS chạy nhanh hơn.

Màn 5 là màn khó nhất trong các màn chơi vì màn này có không gian trạng thái lớn:

$$\begin{aligned}\text{totalStates} &= \text{numPlayerStates} \times \prod_{i=1}^{3} \text{numBoxStates}\,[i] \\ &= 81 \times 80 \times 79 \times 78 = 39,929,760(\text{ states})\end{aligned}$$

Ta có màn chơi 16 có không gian trạng thái lớn hơn là 235,989,936,000, nhưng có diện tích nhỏ là 30 và có nhiều box là 6, do đó sẽ tồn tại nhiều nhánh không chứa lời giải và thuật toán sẽ cắt tỉa những nhánh này. Ngược lại, màn 5 có diện tích lớn là 81 và có ít box là 3, từ đó sẽ cắt tỉa được ít nhánh không chứa lời giải và sẽ chứa nhiều trạng thái hợp lệ hơn màn 16.