ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

ASSIGNMENT 1

DFS/BFS/UCS FOR SOKOBAN

GIẢNG VIÊN HƯỚNG DẪN:
TS. LƯƠNG NGỌC HOÀNG

SINH VIÊN THỰC HIỆN:
NGUYỄN VIỆT NHẬT - 21520378

3, 2023

Mục lục

Chương 1

GIỚI THIỆU TỔNG QUAN

1.1 Yêu cầu

Cài đặt thuật toán BFS và UCS để chơi game Sokoban.

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

Một lời giải tối ưu được quy định là lời giải có số bước đi ngắn nhất.

1.2 Source code (Github)

- Project: nv259/CS106.N21.KHCL/Sokoban
- Solver.py: nv259/CS106.N21.KHCL/Sokoban/solver.py

Chương 2

ÁP DỤNG DFS/BFS/UCS CHO SOKOBAN

2.1 Mô hình hóa

Sokoban là một trò chơi mà mục tiêu của người chơi là đẩy các hộp vào các vị trí đích trên bản đồ. Trò chơi bao gồm một bản đồ, các hộp và một nhân vật, và các hành động có thể thực hiện là di chuyển nhân vật để đẩy hộp. Trò chơi kết thúc khi tất cả các hộp đều được đặt vào các vị trí đích. Với x, y là giá trị thể hiện tọa độ trên bản đồ, trò chơi được tham số hóa như sau:

- Người chơi: posPlayer(x, y).
- Hộp: posBoxes = set(posBox(x, y)).
- Vị trí đích: posGoals = set(posGoal(x, y)).
- Biên: posWalls = set(posWall(x, y)).

Mô hình hóa trò chơi trong bài toán tìm đường:

- Trạng thái khởi đầu (start state): Vị trí của nhân vật và các hộp ban đầu, hay $[(posPlayer, posBoxes)]$.
- Trạng thái kết thúc (goal state): $\forall posBox \in posGoals$.

- Không gian trạng thái: $\forall[(posPlayer, posBoxes)]$.
- Hành động: $Up, Down, Left, Right$.
- Hành động hợp lệ (legal action): Hành động thỏa các ràng buộc:
  + Nhân vật không vượt qua biên của bản đồ: $posPlayer \notin posWalls$.
  + Nhân vật không đẩy các hộp vượt qua biên: $\forall posBox \notin posWalls$.
  + Nhân vật không được đẩy nhiều hộp cùng lúc: $newPosBox \notin posBoxes$.
- Hàm tiến triển (successor function): Cập nhật $posPlayer$ và $posBoxes$ với mỗi hành động hợp lệ.

2.2 Thống kê độ dài đường đi của DFS, BFS và UCS trên các bản đồ

Level	DFS	BFS	UCS
1	79	12	12
2	24	9	9
3	403	15	15
4	27	7	7
5	•	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	•	34	34
17	•	•	•

Bảng 2.1: Bảng thống kê độ dài đường đi

**DFS gặp MemoryError và StackOverFlowError tại các bản đồ 5 và 16.*
**Bản đồ 17 không có lời giải.*

2.3 Nhận xét

Từ bảng 2.1, ta có thể rút ra được các nhận xét về lời giải tìm ra bởi các thuật toán trong trò chơi Sokoban như sau:

- **Thuật toán DFS** (Depth-First Search) có thể giải được bài toán nhưng không đảm bảo đưa ra được lời giải tối ưu. Đối với các bản đồ có không gian trạng thái lớn (như map 5, 16), DFS đi quá sâu vào từng nhánh đường đi và tốn nhiều tài nguyên cho các nhánh không chạm được trạng thái đích, dẫn đến việc hao phí thời gian và bộ nhớ.
- **Thuật toán BFS** (Breath-First Search) đưa ra lời giải chính xác và tối ưu. Tuy nhiên, độ phức tạp thời gian của BFS là lớn đối với các bản đồ có không gian trạng thái lớn, xảy ra do BFS tìm kiếm trên tất cả các trạng thái cùng cấp (trong đó tồn tại các trạng thái không thể dẫn đến lời giải).
- **Thuật toán UCS** (Uniform-Cost Search) về cơ bản cho ra kết quả tối ưu, nhưng có sự khác biệt về thời gian tìm ra lời giải so với BFS. Sự khác biệt này ảnh hưởng chủ yếu bởi hàm cost của UCS và một phần bởi cấu trúc dữ liệu heap cần nhiều thời gian xử lý hơn so với deque thông thường (dùng trong DFS và BFS).

Tóm lại, trong bài toán tìm đường Sokoban, nếu không quan tâm đến độ phức tạp thời gian và bộ nhớ, ta có thể sử dụng DFS, BFS và UCS để tìm ra đường đi đến trạng thái đích của trò chơi.

Tuy nhiên, trong yêu cầu tìm ra lời giải tối ưu, thuật toán DFS là tệ hơn cả. Nhưng không có thuật toán nào là tốt hơn cả trong mọi bản đồ giữa BFS và UCS. Tạm bỏ qua ảnh hưởng từ cấu trúc dữ liệu heap, lý do cho việc này xuất phát từ hàm cost của UCS là một hàm heuristics. Thông qua hàm cost, UCS ưu tiên tìm kiếm trên các đường đi có chi phí thấp hơn (ưu tiên đường đi đẩy hộp hơn các đường đi không đẩy hộp). Chiến thuật này tốt hơn trong các bản đồ có lời giải tối ưu cần nhiều bước đẩy hộp để đi đến trạng thái đích, dẫn đến việc UCS nhiều khả năng đưa ra lời giải nhanh hơn BFS. Tuy nhiên, rõ ràng rằng không phải tất cả các bản đồ đều như vậy và BFS đưa ra kết quả nhanh hơn trong các trường hợp còn lại. Trong các bản đồ có sẵn, UCS đưa ra kết quả nhanh hơn BFS tại 12/16 bản đồ.

Từ thực nghiệm, cả DFS, BFS và UCS đều gặp khó khăn tại bản đồ 5. Không như các bản đồ khác, bản đồ 5 có không gian trạng thái lớn:

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

Mặc dù bản đồ 16 có không gian trạng thái lớn hơn (235,989,936,000). Tuy nhiên, vì diện tích của bản đồ 16 nhỏ (30), nên tồn tại rất nhiều trạng thái có khả năng thất bại tiềm ẩn (như đẩy hộp vào góc tường) và thuật toán sẽ tỉa những nhánh chứa các trạng thái này, làm thu hẹp không gian tìm kiếm. Không như bản đồ 16, bản đồ 5 có diện tích lớn hơn(81), nên số lượng nhánh tỉa được ít hơn đáng kể và tồn tại không gian tìm kiếm lớn hơn so với mọi bản đồ còn lại.