ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

TRÍ TUỆ NHÂN TẠO

DFS/BFS/UCS for Sokoban

BT1_23520127

Môn học: CS106.P21.KHTN

Sinh viên thực hiện:

Nguyễn Thiên Bảo - 23520127

Giáo viên hướng dẫn:

TS.Lương Ngọc Hoàng

Ngày 16 tháng 3 năm 2025

Mục lục

1 Tổng quan về bài tập Sokoban

Ở bài tập này ta sẽ cài đặt và so sánh 3 thuật toán DFS (Depth First Search) , BFS (Breadth First Search) , UCS (Uniform Cost Search) để tìm đường đi cho game Sokoban.

2 Mô hình hóa

Bài toán Sokoban có thể được mô hình hóa dưới dạng một bài toán tìm kiếm đường đi, trong đó người chơi di chuyển trong không gian giới hạn để đẩy các thùng vào vị trí mục tiêu. Dưới đây là cách biểu diễn bài toán theo các thành phần chính:

2.1 Tham số hóa các đối tượng

- Vị trí người chơi: `posPlayer (x, y)`
- Vị trí hộp: `posBox [(x1, y1), (x2, y2), ...]`
- Vị trí mục tiêu: `posGoals [(x1, y1), (x2, y2), ...]`
- Tường/biên: `posWalls [(x1, y1), (x2, y2), ...]`

2.2 Không gian trạng thái (State Space)

Không gian trạng thái gồm tất cả các cấu hình hợp lệ của vị trí player và box trên bản đồ Sokoban. Mỗi trạng thái là một tuple gồm `(posPlayer, posBox)`.
Kích thước không gian trạng thái = (Số ô trống S để đặt player) $\times$ (Số cách xếp N thùng vào số ô trống còn lại (trừ ô của player))

$$= S \times \binom{S-1}{N}$$

$$= \big(\text{Số cách để đặt } N \text{ thùng vào } S \text{ ô trống}\big) \times \big(\text{số ô trống còn lại để đặt player}\big)$$

$$= \binom{S}{N} \times (S - N)$$

Trong đó:

- S: Số ô trống
- N: số thùng
- $\binom{S}{K}$: số cách chọn k vị trí từ S ô trống

2.3 Trạng thái khởi đầu

Vị trí ban đầu của player và box:

```
startState = (beginPlayer, beginBox)
```

`beginPlayer` và `beginBox` được xác định từ `gameState` chuyển hóa từ layout của level.

2.4 Trạng thái kết thúc (Goal State)

Là trạng thái mà tất cả các box đều nằm đúng trên các vị trí mục tiêu (goals).

```
def isEndState(posBox):
    return sorted(posBox) == sorted(posGoals)
```

Điều kiện kết thúc: `posBox` và `posGoals` giống nhau về phần tử (vị trí các hộp khớp với mục tiêu).

2.5 Tập các hành động hợp lệ (Legal Actions)

- Một hành động (action) là **di chuyển người chơi** theo 4 hướng (trái, phải, lên, xuống).
  - Ký tự thường ('u', 'd', 'l', 'r'): chỉ di chuyển người chơi đến ô trống.
  - Ký tự in hoa ('U', 'D', 'L', 'R'): đẩy thùng đến ô trống kế tiếp
- Hành động được mô tả bằng `list` di chuyển theo trục x, y:

```
[dx, dy, 'lowercase', 'uppercase']
```

- Hàm `legalActions(posPlayer, posBox)`:

- Trả về danh sách các hành động **hợp lệ** mà người chơi có thể thực hiện từ trạng thái hiện tại.
- Điều kiện hợp lệ: Vị trí đích của người chơi hoặc thùng không phải là tường hoặc thùng khác, chỉ được đẩy 1 hộp tại 1 thời điểm.

2.6 Hàm tiến triển (Successor Function)

- Hàm `updateState(posPlayer, posBox, action)` mô tả **hàm kế thừa (successor function)**.
- **Input**: `posPlayer`, `posBox`, và một `action`.
- **Output**: trạng thái mới `(newPosPlayer, newPosBox)` sau khi thực hiện hành động đó.

2.7 Kiểm tra trạng thái thất bại (Prune/Deadlock Detection)

- Hàm `isFailed(posBox)`: Sử dụng các pattern xoay, lật ma trận để xem trạng thái thùng có rơi vào **deadlock** không. Mục đích là cắt tỉa (prune) các nhánh tìm kiếm không dẫn đến trạng thái đích, tăng hiệu quả tìm kiếm
- Nếu thùng bị đẩy vào vị trí không thể thoát được (góc chết, kẹp bởi tường và thùng khác, v.v.), trạng thái đó bị loại bỏ khỏi không gian tìm kiếm.

3 Thống kê thuật toán

3.1 Bảng thống kê độ dài đường đi

Map	steps - Runtime DFS	steps - Runtime BFS	steps - Runtime UCS
1	79 - 0.05s	12 - 0.07s	12 - 0.04s
2	24 - 0.00s	9 - 0.00s	9 - 0.00s
3	403 - 0.17s	15 - 0.13s	15 - 0.07s

4	27 - 0.00s	7 - 0.00s	7 - 0.00s
5	treo máy	20 - 146.25s	20 - 59.35s
6	55 - 0.02s	19 - 0.01s	19 - 0.01s
7	707 - 0.42s	21 - 0.66s	21 - 0.43s
8	323 - 0.06s	97 - 0.16s	97 - 0.18s
9	74 - 0.21s	8 - 0.01s	8 - 0.01s
10	37 - 0.01s	33 - 0.01s	33 - 0.01s
11	36 - 0.01s	34 - 0.01s	34 - 0.01s
12	109 - 0.11s	23 - 0.07s	23 - 0.07s
13	185 - 0.14s	31 - 0.11s	31 - 0.14s
14	865 - 3.07s	23 - 2.13s	23 - 2.38s
15	291 - 0.13s	105 - 0.21s	105 - 0.23s
16	86 - 2710.43s	34 - 18.05s	34 - 13.78s
17	treo máy	0 (no answers) - 19.19s	0 (no answers) - 20.19s
18	treo máy	treo máy	treo máy

Nhìn chung 3 thuật toán đều cho ra lời giải, đặc biệt lời giải được tìm ra bởi BFS và UCS giống nhau ở 17 bản đồ. Bản đồ 17 không có lời giải. Bản đồ 18 em chạy bị treo máy ạ. DFS giải khá lâu và "ngoằn ngoèo".

3.2 Nhận xét về đường đi tìm bởi mỗi thuật toán DFS, BFS, UCS

Depth-First Search (DFS):

- **Cách hoạt động**: DFS sử dụng ngăn xếp (stack), ưu tiên duyệt sâu, đi thật xa theo một nhánh trước khi quay lại nhánh khác.
- **Ưu điểm**:
  - Bộ nhớ thấp (chỉ lưu các nhánh hiện tại).

  - Thường **tìm ra lời giải nhanh trên bản đồ nhỏ và đơn giản**, đặc biệt khi lời giải ở nhánh "gần" đầu tiên được thăm.

- **Nhược điểm**:
  - Có thể **dài ngoằn ngoèo** và **không tối ưu**.
  - Hiệu suất kém với bài toán có không gian trạng thái rộng và lời giải ở mức sâu, khó kiểm soát tính toàn vẹn.
  - Dễ rơi vào trạng thái **lặp vô nghĩa** hoặc **deadlock** nếu không có kiểm tra tốt (ở đây có `isFailed` giúp tránh một số deadlock).

- **Hiệu suất**:
  - Tốt cho các bản đồ có nhiều nhánh giải pháp, nhưng kém hiệu quả cho bài toán tối ưu

Breadth-First Search (BFS):

- **Cách hoạt động**: BFS sử dụng hàng đợi để duyệt rộng theo tầng, mở rộng lần lượt theo mức độ sâu tăng dần.

- **Ưu điểm**:
  - Tìm ra được đường đi ngắn nhất về số bước di chuyển.

- **Nhược điểm**:
  - **Tốn bộ nhớ cực lớn**, đặc biệt trong Sokoban với nhiều thùng và không gian trạng thái rộng.
  - Với bài toán lớn hoặc bản đồ phức tạp, BFS dễ hết bộ nhớ hoặc tốn rất nhiều thời gian.

- **Hiệu suất**:
  - Hiệu quả cho các bản đồ nhỏ và trung bình

Uniform Cost Search (UCS):

- **Cách hoạt động**: UCS sử dụng hàng đợi ưu tiên, mở rộng node có **tổng chi phí thấp nhất**, ở đây chi phí được định nghĩa là **số lần di chuyển**
- **Ưu điểm**:
  - Đảm bảo tìm lời giải tối ưu về chi phí tổng thể.
  - Phù hợp với bản chất bài toán Sokoban.
- **Nhược điểm**:
  - Tốn tài nguyên (CPU + RAM), thời gian tính toán dài hơn BFS với bản đồ nhỏ.
  - Dễ **chậm** với bản đồ lớn nếu không có heuristic tốt.
- **Hiệu suất**:
  - Phù hợp khi bài toán cần tính chi phí ràng buộc tối ưu.

3.3 Thuật toán nào là tốt nhất?

⇒ **Uniform Cost Search (UCS)** là **tốt nhất** trong 3 thuật toán xét ở đây, vì:

- Nó đảm bảo lời giải tối ưu về chi phí tổng thể.
- UCS mở rộng các trạng thái theo thứ tự chi phí tăng dần, nên sẽ khám phá các giải pháp tiềm năng tốt trước.
- UCS cân nhắc giữa số bước và độ khó của mỗi bước. Nếu chỉ dùng BFS, ta sẽ dễ gặp đường đi tốn công đẩy hơn.
- DFS rất dễ **bỏ sót** giải pháp tốt hoặc mất kiểm soát trong không gian tìm kiếm rộng.

3.4 Bản đồ nào khó giải nhất? Tại sao?

3.4.1 Tiêu chí bản đồ khó:

- **Số thùng nhiều** ⇒ Số lượng trạng thái bùng nổ.
- **Không gian rộng** ⇒ kích thước không gian trạng thái lớn.
- **Không gian thoáng** ⇒ Ít chướng ngại vật thì thuật toán không cắt tỉa được các trạng thái không có tiềm năng nên phải xét hầu hết không gian trạng thái.
- **Không gian hẹp** ⇒ Bản đồ có nhiều tường và khu vực hẹp, hạn chế khả năng di chuyển và đẩy thùng, dễ deadlock.

3.4.2 Trong project này (theo sơ đồ bản đồ):

Bản đồ 5, 18 là hai bản đồ khó. Vì bản đồ 5 không gian vừa rộng vừa thoáng. Bản đồ 18 lại vừa nhiều thùng, lại vừa hẹp. Khiến cho cả 3 thuật toán đều không dễ tìm ra kết quả trong thời gian ngắn.