VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY

CS106 KHTN

Báo cáo bài tập Sokoban

Sinh Viên :
Đồng Quốc Thắng

Giảng viên :
Lương Ngọc Hoàng

Ngày 21 tháng 3 năm 2025

Mục lục

Chương 1

Mô hình hóa Sokoban

1.1 Trạng thái khởi đầu

Ở bài tập Sokoban này, ta chọn cách biểu diễn một trạng thái bằng 2 phần:

- Vị trí của người chơi trên bản đồ
- Vị trí của các hộp trên bản đồ

Khi đó nên trạng thái ban đầu được lấy bằng cách lấy ra một tuple gồm có vị trí của người chơi ban đầu và vị trí của các hộp ban đầu

```
1 beginBox = PosOfBoxes(gameState)
2 beginPlayer = PosOfPlayer(gameState)
3 startState = (beginPlayer, beginBox)
```

1.2 Trạng thái kết thúc

Trạng thái kết thúc là khi mà vị trí của tất cả các hộp trùng với tất cả các vị trí đích các hộp đều là như nhau, không phân biệt các hộp.

Ngoài ra khi kiểm tra trạng thái kết thúc, chỉ cần quan tâm đến vị trí của các hộp mà không cần quan tâm vị trí của người chơi.

Ta kiểm tra trạng thái kết thúc mỗi khi ta xem xét một trạng thái mới.

1.3 Không gian trạng thái

Vì một trạng thái gồm có vị trí của người chơi và vị trí của các hộp nên giả sử trong một map có x ô(một ô có thể có một hộp hoặc một người chơi) và y hộp, bởi vì các hộp đều giống nhau nên không gian trạng thái có thể xấp xỉ là:

$\approx \binom{x}{y} \times (x - y)$

Ta nhân với $(x - y)$ vì người chơi có thể ở gần như bất kì ô nào trong các ô còn lại.

Công thức này chỉ tính xấp xỉ vì có thể có những trạng thái mà người chơi ở trong góc tường và chỉ có một hướng để đi ra nhưng lại có một hộp ngay trên đường đi. Trường hợp này sẽ không thể xảy ra trong thực tế(trừ khi được đặt ở trạng thái bắt đầu) nhưng công thức trên cũng sẽ tính luôn trường hợp này.

1.4 Các hành động hợp lệ

Dựa vào phần code có sẵn:

```
def isLegalAction(action, posPlayer, posBox):
    """Check if the given action is legal"""
    xPlayer, yPlayer = posPlayer
    if action[-1].isupper(): # the move was a push
        x1, y1 = xPlayer + 2 * action[0], yPlayer + 2 * action[1]
    else:
        x1, y1 = xPlayer + action[0], yPlayer + action[1]
    return (x1, y1) not in posBox + posWalls
```

Ta có thể biết được các hành động là hợp lệ nếu:

- Người chơi không di chuyển vào tường
- Người chơi không di chuyển vào một thùng trừ khi đó là một lần đẩy hợp lệ
- Một lần đẩy chỉ hợp lệ nếu thùng có thể được đẩy vào một ô trống

1.5 Hàm tiến triển

Hàm tiến triển là hàm updateState như sau:

```
def updateState(posPlayer, posBox, action):
    """Return updated game state after an action is taken"""
    xPlayer, yPlayer = posPlayer # the previous position of player
    newPosPlayer = [xPlayer + action[0], yPlayer + action[1]] # the
    ↪  current position of player
    posBox = [list(x) for x in posBox]
    if action[-1].isupper(): # if pushing, update the position of box
        posBox.remove(newPosPlayer)
        posBox.append([xPlayer + 2 * action[0], yPlayer + 2 * action[1]])
    posBox = tuple(tuple(x) for x in posBox)
    newPosPlayer = tuple(newPosPlayer)
    return newPosPlayer, posBox
```

Hàm tiến triển này sẽ update trạng thái của game qua một trạng thái mới, vì một trạng thái sẽ gồm có vị trí của người chơi và vị trí của các hộp. Hàm update trạng thái sẽ trả về vị trí của người chơi và các hộp sau khi thực hiện một hành động

Chương 2

Thống kê và nhận xét

2.1 Thời gian chạy và độ dài đường đi của DFS, BFS, UCS

2.1.1 BFS

Thới gian để tìm ra kết quả và số bước đi khi sử dụng thuật toán BFS:

Level	Số bước đi	Thời gian chạy
1	12	0.05
2	9	0.00
3	15	0.11
4	7	0.00
5	NaN	NaN
6	19	0.01
7	21	0.53
8	97	0.12
9	8	0.00
10	33	0.01
11	34	0.01
12	23	0.05
13	21	0.09
14	23	1.71
15	105	0.17
16	34	14.66
17	NaN	NaN
18	NaN	NaN

Phần chạy BFS có 3 map không ra được kết quả

- Map 5 treo khá lâu và không ra được đáp án
- Map 17 chạy khoảng 17 giây nhưng không có solve(ra list rỗng)
- Map 18 treo lâu và không ra được đáp án

2.1.2 UCS

Thời gian để tìm ra kết quả và số bước đi khi sử dụng thuật toán UCS:

Level	Số bước đi	Thời gian chạy
1	12	0.03
2	9	0.00
3	15	0.05
4	7	0.00
5	20	53.99
6	19	0.01
7	21	0.33
8	97	0.13
9	8	0.01
10	33	0.01
11	34	0.01
12	23	0.05
13	31	0.11
14	23	1.95
15	105	0.18
16	34	11.23
17	NaN	NaN
18	NaN	NaN

Phần chạy UCS có 2 map không ra được kết quả

- Map 17 chạy khoảng 15 giây nhưng không có solve(ra list rỗng)
- Map 18 treo lâu và không ra được đáp án

2.1.3 DFS

Thời gian để tìm ra kết quả và số bước đi khi sử dụng thuật toán DFS:
Phần chạy DFS có 4 map không ra được kết quả

- Map 5 treo khá lâu và không ra được đáp án
- Map 16 treo khá lâu và không ra được đáp án
- Map 17 chạy khoảng 17 giây nhưng không có solve(ra list rỗng)
- Map 18 treo lâu và không ra được đáp án

Level	Số bước đi	Thời gian chạy
1	79	0.03
2	24	0.00
3	403	0.14
4	27	0.00
5	NaN	NaN
6	55	0.01
7	707	0.32
8	323	0.5
9	74	0.17
10	37	0.01
11	36	0.01
12	109	0.08
13	185	0.12
14	865	2.45
15	291	0.11
16	NaN	NaN
17	NaN	NaN
18	NaN	NaN

2.2 Nhận xét

Sau bài tập này, em rút ra một số nhận xét như sau

- DFS cho đáp án có số bước đi rất dài. DFS cho nhiều bước đi vì DFS cứ bung các nút có thể và chọn một nút bất kì, nên sẽ có nhiều bước đường vòng để tới một state nhất định
- BFS và UCS cho kết quả optimal nên số bước đi của BFS và UCS giống nhau
- UCS có thời gian chạy tốt hơn BFS, điều này là do BFS chạy hết từng độ sâu, còn UCS thì đi tìm dựa trên cost, nên UCS sẽ tìm được cách giải ở độ sâu cao tốt hơn BFS.
- Map thứ 5 chạy rất lâu mà không có kết quả trong DFS và BFS, điều này là vì diện tích map khá lớn và có thể di chuyển thuận tiện(map hình vuông, không có nhiều góc nhỏ) nên DFS và BFS sẽ thử rất nhiều trường hợp gây ra chạy rất lâu, không ra nổi kết quả
- UCS không chạy theo khuynh hướng như BFS và DFS(thử tất cả các trường hợp mà không quan tâm tới cost hay độ hiệu quả) nên UCS có thể tìm ra được đáp án nhanh hơn, tuy nhiên chạy cũng khá lâu.
- Map thứ 18 khá lớn và phức tạp nên không thuật nào chạy nổi
- DFS không chạy nổi map thứ 16, có thể vì DFS thử tất cả các trường hợp ở độ sâu cao nhất và map cũng khá rộng nên thử rất lâu, treo máy rất lâu nhưng mà DFS vẫn không chạy ra kết quả.