Đại học Quốc gia TP.HCM

Trường Đại học Công nghệ Thông tin

Báo cáo về việc sử dụng các thuật toán tìm đường đi cho trò chơi Sokoban

Nguyễn Duy Hoàng
MSSV: 22520467
Lớp CS106.O21
Giảng viên môn học: Lương Ngọc Hoàng

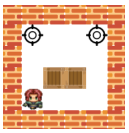
Mục lục

1 Sokoban được mô hình hóa ra sao?

Sokoban là một trò chơi giải đố nơi người chơi điều khiển một nhân vật với tên gọi **Sokoban** (thường là một người lao động hoặc một con robot) để đẩy một hoặc nhiều hộp (hoặc vật thể khác) đến các ô đích trên một bản đồ. Mục tiêu của trò chơi là đẩy tất cả các hộp đến các ô đích mà không bị kẹt hoặc không thể hoàn thành.

Trạng thái khởi đầu của trò chơi (*start state - initial state*) là một bản đồ hoặc một bố cục của các ô được đại diện bằng ký hiệu, trong đó **"#"** là vật cản (như là tường), **"B"** là hộp, **"."** là ô đích, " " là khoảng trống có thể di chuyển và "&" là vị trí khởi tạo của nhân vật **Sokoban** tại mỗi level trò chơi. Ví dụ trong màn chơi thứ nhất, trạng thái khởi đầu là: ((4, 1), ((3, 2), (3, 3))); trong đó (4,1) là tọa độ của nhân vật *Sokoban* và ((3, 2), (3, 3)) là tọa độ của 2 hộp khi mới bắt đầu màn chơi. Trạng thái khởi đầu có thể được minh họa thông qua các hình dưới đây:

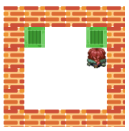
```
# # # # # #
# . .   . #
#         #
#   B B   #
# &       #
# # # # # #
```



Hình 1: Trạng thái khởi đầu của màn chơi đầu tiên (Level 1) (trái) và
Ảnh minh họa cho trạng thái khởi đầu (phải)

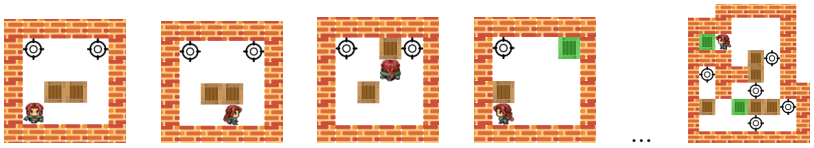
Trạng thái kết thúc chỉ có thể xảy ra khi tất cả các hộp đã được đẩy đến các ô đích. Ví dụ trong màn chơi 1, trạng thái kết thúc là: ((2, 4), ((1, 1), (1, 4))); trong đó (2,4) là tọa độ của nhân vật *Sokoban* và ((1, 1), (1, 4)) là tọa độ của 2 hộp khi màn chơi hoàn thành và cũng tương đương với tọa độ của 2 ví trí đích (vị trí mục tiêu). Trạng thái kết thúc có thể được minh họa thông qua các hình dưới đây:

```
# # # # # #
# B     B #
#       & #
#         #
#         #
# # # # # #
```



Hình 2: Trạng thái kết thúc của màn chơi đầu tiên (Level 1) (trái) và
Ảnh minh họa cho trạng thái kết thúc (phải)

Không gian trạng thái của trò chơi bao gồm tất cả các trạng thái có thể có của trò chơi, trong đó mỗi trạng thái được xác định bởi vị trí của người chơi và các hộp trên bản đồ. Không gian trạng thái có thể được minh họa thông qua các hình dưới đây:



Hình 3: Không gian trạng thái của trò chơi

Có tất cả 8 hành động hợp lệ (*legal actions*) trong trò chơi: người chơi có thể di chuyển lên *'u'*, xuống *'d'*, sang trái *'l'*, sang phải *'r'* từ một ô trống đến một ô trống khác liền kề và các tương tác với hộp (*'U','D','L','R'*) .

Hàm tiến triển *(successor function)* trong trò chơi nhằm xác định các trạng thái kế tiếp có thể đạt được từ trạng thái hiện tại bằng cách thực hiện một hành động cụ thể (di chuyển hộp, di chuyển nhân vật Sokoban). Hàm tiến triển trong trường hợp này là hàm `updateState()`.

2 So sánh 3 thuật toán tìm đường: DFS, BFS và UCS

Trò chơi giải đố **Sokoban** có thể tồn tại nhiều lời giải khả thi cho mỗi màn chơi (*level*), với độ dài của lời giải thay đổi tùy thuộc vào kích thước và độ phức tạp của mỗi cấp độ. Việc xác định thuật toán hiệu quả nhất đòi hỏi xem xét nhiều khía cạnh, như số bước đi của nhân vật **Sokoban** hoặc thời gian cần để tìm ra một lời giải cho mỗi cấp độ. Dưới đây là hai bảng theo dõi số bước và thời gian cần để tìm ra lời giải cho nhân vật Sokoban qua từng cấp độ, sử dụng các thuật toán *DFS, BFS, UCS*:

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

Bảng 1: **Thống kê số bước di chuyển của nhân vật Sokoban để di chuyển hộp tới đích của từng thuật toán**

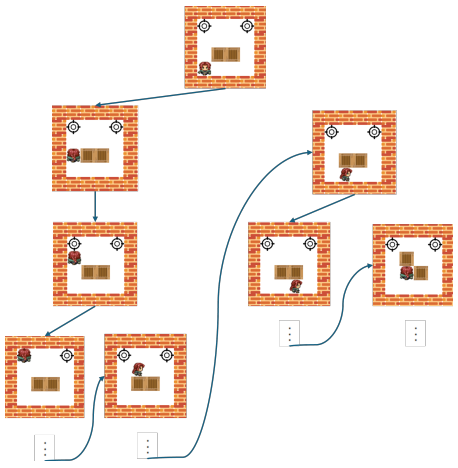
Level	Algorithm		
	DFS	BFS	UCS
1	0.05	0.07	0.06
2	0.00	0.00	0.01
3	0.19	0.16	0.1
4	0.00	0.01	0.00
5	DNF	179.72	140.03
6	0.01	0.01	0.01
7	0.46	0.77	0.59
8	0.07	0.18	0.21
9	0.22	0.01	0.01
10	0.01	0.01	0.02
11	0.01	0.01	0.02
12	0.11	0.08	0.09
13	0.16	0.13	0.17
14	3.41	2.54	3.01
15	0.14	0.25	0.28
16	DNF	21.74	20.42

Bảng 2: **Thống kê thời gian tìm ra đường đi của nhân vật Sokoban để di chuyển hộp tới đích của từng thuật toán**

Ghi chú: Màn chơi số 5 và 16 được thực hiện bằng thuật toán DFS, tuy nhiên sau 1 giờ chờ vẫn chưa thực hiện xong (DNF: Did Not Finish).

2.1 Depth first search (*DFS*)

Depth first search (DFS) có nghĩa là tìm kiếm theo chiều sâu, đây là một thuật toán dùng để duyệt qua hoặc tìm kiếm cấu trúc dữ liệu trong bài toán đồ thị hoặc bài toán dạng cây. Thuật toán bắt đầu tại nút gốc (trong trường hợp này là vị trí ban đầu tại mỗi màn chơi của nhân vật **Sokoban**) rồi bắt đầu kiểm tra từng nhánh xa nhất có thể trước khi quay lui (*backtracking*).



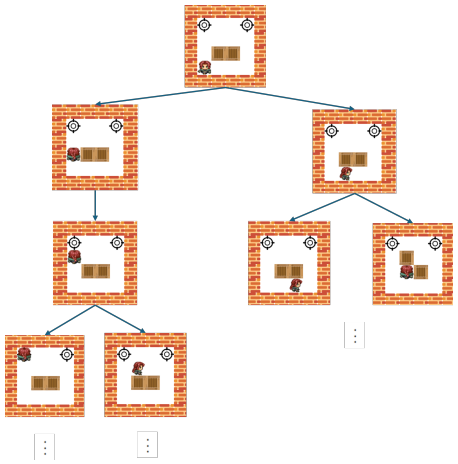
Hình 4: Ví dụ minh họa về thuật toán DFS

Nếu quan sát *Bảng 1* và *Bảng 2*, ta có thể nhận thấy rằng thuật toán DFS thường hoàn thành màn

chơi với số bước đi lớn hơn đáng kể so với hai thuật toán còn lại, tuy nhiên thời gian thực hiện không chênh lệch nhiều, thậm chí có thể nhỉnh hơn hai thuật toán còn lại. Nguyên nhân cho hiện tượng này là DFS tập trung chủ yếu vào việc tìm ra lời giải mà không quan tâm đến số lượng bước đi, chỉ cần khi duyệt qua các nhánh một cách sâu nhất có thể và tìm thấy lời giải, thuật toán sẽ dừng lại và xuất ra kết quả.

2.2 Breadth first search (*BFS*)

Breadth first search (BFS) là một thuật toán tìm kiếm theo chiều rộng, được sử dụng để duyệt qua hoặc tìm kiếm cấu trúc dữ liệu trong bài toán đồ thị hoặc bài toán dạng cây. Thuật toán bắt đầu tại nút gốc (trong trường hợp này là vị trí ban đầu tại mỗi màn chơi của nhân vật **Sokoban**) và kiểm tra tất cả các nút cùng cấp trước khi di chuyển sang các nút cấp tiếp theo.



Hình 5: Ví dụ minh họa về thuật toán BFS

Khi triển khai `def bfs` trong tệp `solver.py`, em đã dựa trên `def dfs` và thay đổi một số điểm cần thiết để phù hợp với phương pháp tìm kiếm theo chiều rộng. Cụ thể, phương pháp lựa chọn nút từ hàng đợi sử dụng phương thức `popleft()` thay vì `pop()` để lấy ra nút ở đầu hàng đợi để mở rộng, và sau đó sử dụng `append()` để thêm các nút con mới được mở rộng vào cuối hàng đợi. Điều này tương ứng với việc duyệt qua các nút ở cùng một cấp trước khi di chuyển xuống các nút ở cấp tiếp theo, như mô tả của thuật toán BFS.

Trong Python, để triển khai hàng đợi, em sử dụng `deque` từ thư viện `collections`, cung cấp các phương thức như `append()` để thêm một phần tử vào cuối hàng đợi và `popleft()` để lấy ra phần tử đầu tiên của hàng đợi.

```
from collections import deque

queue = deque([1, 2, 3, 4, 5])
last_element = queue.pop()
first_element = queue.popleft()
print(last_element)
print(first_element)
```

```
Output:
5
1
```

Listing 1: Ví dụ về sử dụng `popleft()` và `pop()` trong `deque`

Nếu quan sát *Bảng 1* và *Bảng 2*, ta có thể nhận thấy rằng thuật toán BFS luôn hoàn thành màn chơi với số bước đi ít hơn đáng kể so với thuật toán DFS. Điều này là do BFS duyệt tất cả các nút cùng cấp trước khi di chuyển sang cấp tiếp theo, đảm bảo rằng lời giải tìm được là lời giải ngắn nhất giữa hai nút nếu có.

2.3 Uniform cost search (*UCS*)

Uniform Cost Search (UCS) là một thuật toán tìm kiếm thông tin trong đồ thị hoặc cây, có mục tiêu là tìm đường đi có chi phí thấp nhất. Ý tưởng đằng sau UCS là tính toán các đường đi theo thứ tự tăng dần của chi phí đã trải qua. Điều này đảm bảo rằng thuật toán sẽ tìm ra đường đi với chi phí nhỏ nhất trước khi chuyển sang các đường đi có chi phí cao hơn.

Hàm `def ucs` sử dụng một hàng đợi ưu tiên (Priority Queue) thay vì deque như trong BFS. Hàng đợi ưu tiên được sắp xếp theo chi phí đã trải qua, với các đường đi có chi phí thấp nhất được xử lý trước. Điều này giúp thuật toán lựa chọn đường đi ngắn nhất một cách hiệu quả.

3 Kết luận

3.1 Nhận xét về thuật toán

Trong các bài toán tìm đường đi, việc sử dụng thuật toán UCS sẽ thường tối ưu hơn so với hai thuật toán còn lại là BFS và DFS, đặc biệt là khi chi phí mỗi bước đi là khác nhau. Tuy nhiên, trong trường hợp của bài toán Sokoban, UCS và BFS đều là lựa chọn tốt hơn so với DFS do chi phí mỗi bước đi là như nhau. Dựa vào các thử nghiệm và so sánh giữa các thuật toán trên *Bảng 1* và *Bảng 2*, ta có thể kết luận rằng hiệu suất của hai thuật toán này trong bài toán Sokoban là gần như tương đương nhau.

3.2 Nhận xét về màn chơi khó nhất

Trong số các màn chơi của Sokoban, màn chơi có độ khó cao nhất là màn chơi level 15. Điều này đến từ sự phức tạp của bản đồ và lượng lớn các hộp và mục tiêu trên bản đồ. Cấu trúc phức tạp của bản đồ cũng làm tăng độ khó bằng cách chia bản đồ thành nhiều phần nhỏ, tạo ra các ngõ ngách và điều này có thể dẫn đến tình huống một số hộp bị mắc kẹt.