ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

ASSIGNMENT 1

HỌC PHẦN: TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI: XÂY DỰNG THUẬT TOÁN DFS/UFS/UCS CHO TRÒ CHƠI SOKOBAN

Giảng viên hướng dẫn: Lương Ngọc Hoàng

Lớp: Trí tuệ nhân tạo - CS106.O22

Sinh viên thực hiện: Nguyễn Vũ Khai Tâm – 22521293

TP. Hồ Chí Minh, tháng 3, năm 2023

TỔNG QUAN ĐỀ TÀI

I. Giới thiệu trò chơi Sokoban

- Trò chơi Sokoban là một trò chơi giải đố cổ điển của Nhật Bản. Tên "Sokoban" trong tiếng Nhật nghĩa là "người dắt xe hàng". Trò chơi này đòi hỏi sự suy nghĩ chiến lược và tiếp cận có hệ thống để giải quyết. Trong Sokoban, người chơi điều khiển một nhân vật trong một kho chứa các thùng với mục tiêu của trò chơi là đẩy các thùng qua một kho lưu trữ đầy đủ các ngóc ngách và chướng ngại vật để đặt chúng lên các điểm đích đánh dấu trước đó trên sàn.
- Đặc Điểm:

  + Tư Duy Chiến Lược và Giải Quyết Vấn Đề: Sokoban đòi hỏi người chơi phải có khả năng tư duy logic và chiến lược tiên tiến để giải quyết các bài toán. Mỗi cấp độ có thể coi là một bài toán tối ưu hóa, nơi mục tiêu là tìm ra dãy hành động ngắn nhất để hoàn thành mục tiêu.

  + Không Gian Trạng Thái Lớn: Mặc dù quy tắc chơi khá đơn giản, nhưng không gian trạng thái (tất cả các vị trí có thể của người chơi và hộp) có thể trở nên rất lớn ngay cả với số lượng hộp và vị trí đích tương đối ít. Điều này tạo ra thách thức trong việc tìm kiếm lời giải.

II. Phương pháp giải quyết trò chơi Sokoban

- Tiến hành sử dụng 3 thuật toán tìm kiếm mù Uninformed Search Algorithms là Depth First Search, Breadth First Search, Uniform-Cost Search để giải quyết trò chơi Sokoban

III. Mục tiêu của bài báo cáo

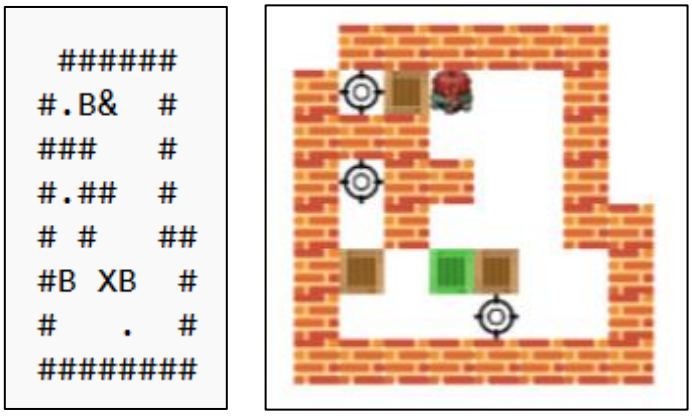
- Cách trò chơi Sokoban được mô hình hóa: Trạng thái khởi đầu, trạng thái kết thúc, không gian trạng thái, các hành động hợp lệ, hàm tiến triển (successor function)
- Thống kê về độ dài đường đi (số bước đi) tìm được bởi 3 thuật toán DFS, BFS, UCS tại tất cả các bản đồ có sẵn. Rút ra kết luận về lời giải (đường đi) tìm ra bởi mỗi thuật toán. Tìm ra thuật toán nào là tốt hơn cả, giải thích.
- Xác định bản đồ nào khó giải nhất, lý giải nguyên nhân.

TRÒ CHƠI SOKOBAN

I. Mô tả trò chơi sokoban dựa trên cài đặt

- **Trạng thái khởi đầu:** gồm vị trí của người chơi, các hộp, và môi trường bao quanh như tường và điểm đích. Trạng thái khởi đầu được tạo ra bởi hàm transferToGameState và transferToGameState2 bởi các layout được cung cấp. Trạng thái này bao gồm một ma trận biểu diễn vị trí của tất cả các đối tượng trên bản đồ.
- **Trạng thái kết thúc:** Trạng thái mà tại đó tất cả các hộp đều được đặt trên các điểm đích. Hàm isEndState kiểm tra điều này bằng cách so sánh vị trí của tất cả các hộp với vị trí của các điểm đích.
- **Không gian trạng thái:** Tất cả các trạng thái có thể có của trò chơi, từ trạng thái ban đầu đến trạng thái kết thúc, bao gồm tất cả các vị trí có thể của người chơi và các hộp.
- **Các hành động hợp lệ:** Được xác định bởi hàm legalActions, đây là các bước di chuyển mà người chơi có thể thực hiện mà không vi phạm các quy tắc của trò chơi như di chuyển qua tường hoặc đẩy hộp vào nhau. Các hành động bao gồm di chuyển lên, xuống, trái, và phải, cũng như đẩy hộp nếu có thể.
- **Hàm tiến triển (Successor Function):** Với chế độ chơi thông thường, hàm updateState được sử dụng để cập nhật trạng thái của trò chơi dựa trên quyết định của người chơi sau mỗi hành động hợp lệ. Hàm này xác định vị trí mới cho người chơi và hộp, và trả về trạng thái mới của trò chơi. Với chế độ auto, hàm updateState được sử dụng bởi các thuật toán như DFS, BFS, và UCS để tự động tạo ra quyết định và tiến triển trò chơi.

II. Mô hình hóa trò chơi



Hình 1: Ví dụ mô phỏng bản đồ trò chơi level 3

+ Vị trí của người chơi ký hiệu “&”.

+ Vị trí của các hộp ký hiệu “B”.

+ Vị trí của các điểm mục tiêu ký hiệu “.”

+ Tường ký hiệu “#” bao quanh và trong bản đồ, nơi không thể di chuyển vào.

+ Không gian trống ký hiệu “ ”

- **Trạng thái khởi đầu:**

- Vị trí của của 4 thùng:

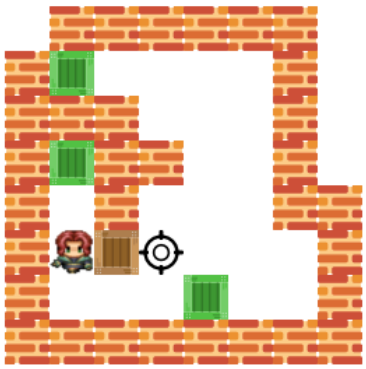
```
print(PosOfBoxes(gameState))
((1, 2), (5, 1), (5, 3), (5, 4))
```

- Vị trí của nhân vật:

```
print(PosOfPlayer(gameState))
(1, 3)
```

Hình 2: Trạng thái khởi đầu của level 3

- **Trạng thái kết thúc:** Khi tất cả các hộp B đều được di chuyển vào vị trí của các điểm mục tiêu “.”



- Trạng thái kết thúc:

```
Level:  3
((1, 1), (6, 4), (3, 1), (5, 3))
```

Hình 3: Trạng thái kết thúc của level 3

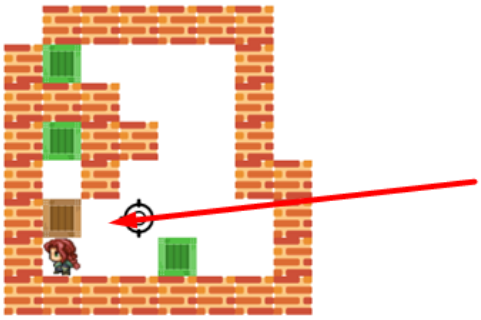
- **Không gian trạng thái:**

+ Phần tử của không gian trạng thái: Gồm vị trí của tất cả các tường, người chơi, hộp, và điểm đích. Trong Sokoban, tường và điểm đích cố định, vị trí của người chơi và hộp có thể thay đổi.

+ Định nghĩa: Tất cả các cách sắp xếp có thể có của người chơi và các hộp trong bản đồ, không bao gồm các vị trí của tường "#".

+ Chuyển đổi giữa các Trạng thái: Mỗi lần người chơi di chuyển hoặc đẩy một hộp, một trạng thái mới được tạo ra. Tất cả các trạng thái này liên kết với nhau thông qua chuỗi hành động hợp lệ có thể xảy ra. Ví dụ các hộp không thể chồng lên nhau hoặc bị đẩy vào tường, và người chơi không thể di chuyển qua tường.

- **Các hành động hợp lệ:** Di chuyển người chơi lên, xuống, trái, phải nếu không có tường. Đẩy hộp vào không gian trống hoặc trên mục tiêu nếu có không gian phía sau hộp đó.
- **Trạng thái dẫn đến thất bại:** Đây là tình huống mà trong đó có một hoặc nhiều thùng không được đặt đúng vị trí mục tiêu và không còn khả năng di chuyển thêm, bất kể thực hiện bao nhiêu lần hành động đi chăng nữa.



Đây là một trạng thái không hợp lệ

- **Hàm tiến triển:** Xác định trạng thái mới dựa trên hành động được thực hiện từ trạng thái hiện tại, cập nhật vị trí người chơi và hộp sau mỗi hành động hợp lệ.

III. Các thuật toán tìm kiếm

- Cả 3 hàm tiến triển trong bài báo cáo này đề cập đến đều thuộc loại thuật toán tìm kiếm mù là DFS, BFS và UCS, các thuật toán được triển khai theo Graph Search.
- Tiến hành khám phá không gian trạng thái của một bài toán bằng cách mở rộng từng node một, mỗi node đại diện cho một trạng thái trong trò chơi. Để quản lý quá trình này, các thuật toán cần sử dụng frontier lưu trữ các node cần được xét, actions lưu trữ chuỗi hành động dẫn đến từng node, và exploredSet lưu trữ các trạng thái đã được khám phá để tránh xét lại chúng. Trong mỗi lần lặp, một node được lấy ra từ frontier để xét; nếu trạng thái của node là trạng thái kết thúc, quá trình tìm kiếm kết thúc và lời giải được trả về. Nếu không, các hành động hợp lệ từ trạng thái hiện tại được áp dụng để tạo ra các node mới,

và chỉ những node không dẫn đến thất bại mới được thêm vào frontier, với sự trợ giúp của hàm isFailed() để loại trừ các trạng thái không mong muốn. Quá trình này lặp lại cho đến khi tìm thấy lời giải hoặc khi frontier trống rỗng, tức là không có lời giải nào khả dĩ từ trạng thái bắt đầu.

1) Depth First Search (Tìm kiếm theo chiều sâu):

- DFS khám phá không gian trạng thái của Sokoban bằng cách ưu tiên mở rộng những đường đi sâu nhất trước, sử dụng một ngăn xếp (stack) LIFO (Last In, First Out) để quản lý các trạng thái. Mỗi khi một trạng thái được khám phá, nó sẽ thêm tất cả các hành động hợp lệ tiếp theo vào ngăn xếp để xử lý (cấu trúc dữ liệu của frontier và actions là stack).
- DFS tiếp tục mở rộng trạng thái cho đến khi tìm thấy lời giải hoặc khi không còn trạng thái nào trong ngăn xếp. Thuật toán có thể quay lui khi một nhánh không đưa đến lời giải, thử các nhánh khác cho đến khi tìm ra lời giải.
- Điểm yếu của thuật toán này là không đảm bảo tìm được lời giải tối, điều này dễ nhận thấy khi ta chạy Sokoban bằng phương pháp DFS khi nó chạy rất nhiều bước đi thừa. Và ở một số test ta thấy thuật toán này có thể bị mắc kẹt trong vòng lặp vô hạn nếu không gian trạng thái có chu trình và không kiểm soát được độ sâu.
- Ưu điểm của thuật toán này là DFS chỉ sử dụng bộ nhớ là $O(bd)$, ít hơn so với BFS, điều này giúp cho không quá nhiều dữ liệu chờ được duyệt trong ngăn xếp.

2) Breadth First Search (Tìm kiếm theo chiều rộng):

- BFS khám phá không gian trạng thái bằng cách mở rộng đường đi theo chiều rộng trước, đảm bảo rằng tất cả các nút ở một độ sâu nhất định được khám phá trước khi chuyển sang độ sâu tiếp theo. Điều này được thực hiện bằng cách sử dụng collections.deque như một hàng đợi FIFO (First In, First Out), nơi các nút mới được thêm vào cuối và được lấy ra từ đầu của hàng đợi (cấu trúc dữ liệu của frontier và actions là hàng đợi).
- Khi một nút được khám phá, tất cả các nút con của nó được thêm vào hàng đợi, đảm bảo rằng BFS sẽ xét đến mọi nút ở cùng một độ sâu trước khi di chuyển đến độ sâu tiếp theo. BFS tiếp tục khám phá không gian trạng thái cho đến khi tìm thấy mục tiêu. Vì khám phá theo chiều rộng, BFS đảm bảo rằng lời giải tìm được (nếu có) là lời giải ngắn nhất đến mục tiêu khi mỗi bước di chuyển có cùng chi phí.
- Giả sử số lượng nhánh cố định ở mỗi nút là b và ở độ sâu d, thì độ lớn của search frontier là $O(b^d)$. Điều này dẫn đến việc sử dụng bộ nhớ tăng mạnh theo hàm mũ khi d tăng lên.

3) Uniform Cost Search (Tìm kiếm với chi phí đồng nhất):

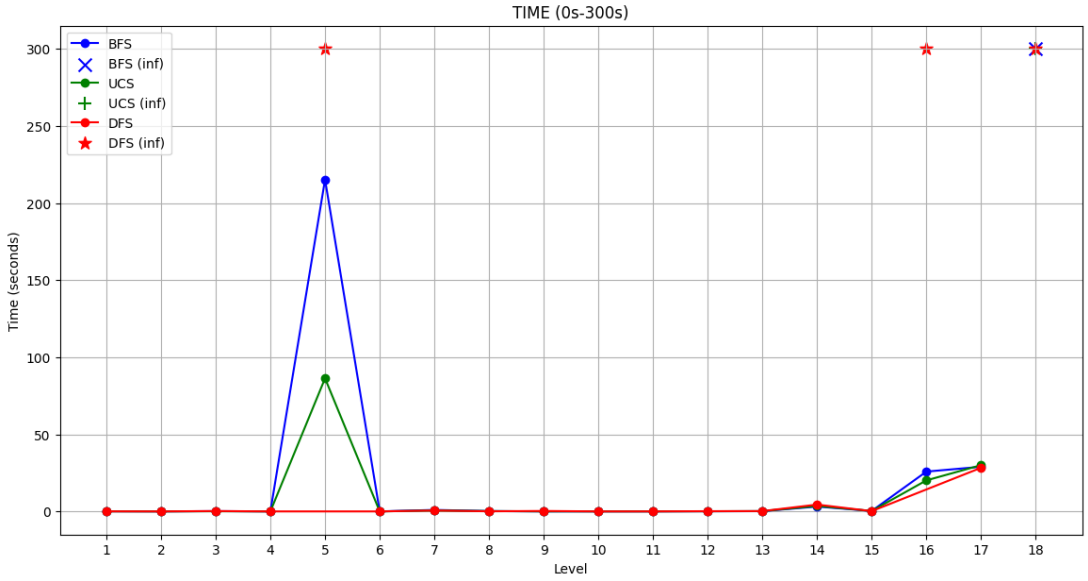
- UCS là phiên bản cải tiến của BFS, mở rộng trạng thái dựa vào tổng chi phí từ trạng thái bắt đầu đến trạng thái hiện tại, không chỉ dựa vào độ sâu, UCS sử dụng hàng đợi ưu tiên (PriorityQueue) để đảm bảo trạng thái có tổng chi phí thấp nhất được mở rộng trước.
- Hàm chi phí được cài đặt như sau: nó đếm số lượng hành động được biểu diễn bằng các ký tự chữ thường trong danh sách actions và sử dụng số lượng này như là chi phí của các hành động đó. Một hành động di chuyển (chữ cái thường) sẽ có chi phí là 1, và hành động đẩy thùng (chữ cái in hoa) sẽ không được tính chi phí.
- Dù tương đồng với BFS là vậy, UCS vẫn có ưu thế về mặt chi phí và thời gian do có cải tiến lượt bỏ chi phí hành động đẩy thùng.

IV. Thống kê

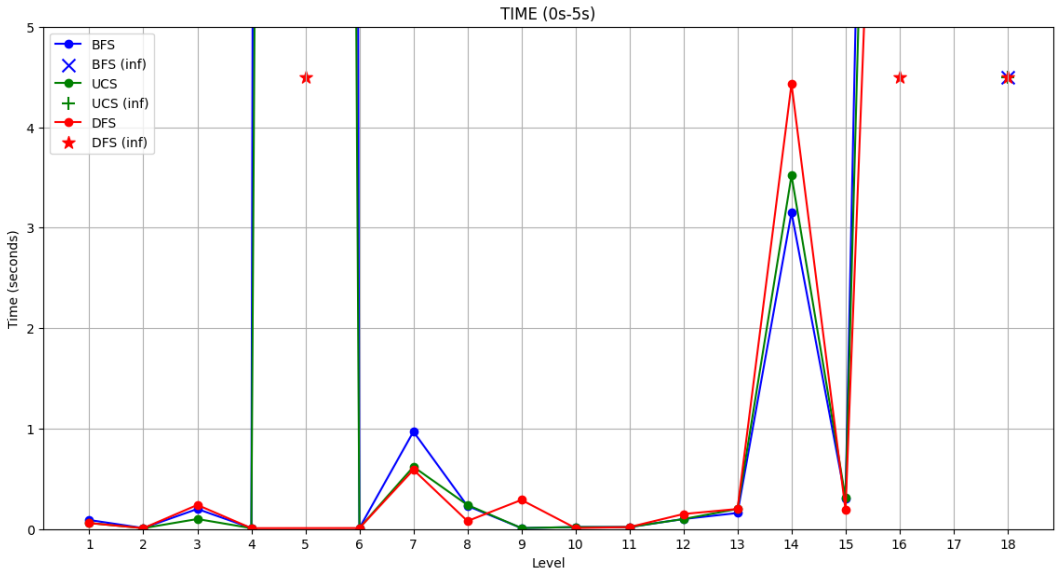
Bảng thống kê về thời gian và số bước của các thuật toán trong game Sokoban

LEVELS	*Time (seconds)*			*Steps*		
	BFS	**UCS**	**DFS**	**BFS**	**UCS**	**DFS**
1	0.09	0.06	0.06	12	12	79
2	0.01	0.009	0.009	9	9	24
3	0.20	0.10	0.24	15	15	403
4	0.01	0.009	0.009	7	7	27
5	215.26	86.32	**infinity**	20	20	**-**
6	0.01	0.01	0.01	19	19	55
7	0.97	0.62	0.59	21	21	707
8	0.23	0.24	0.08	97	97	323
9	0.01	0.01	0.29	8	8	74
10	0.02	0.02	0.01	33	33	37
11	0.02	0.02	0.02	34	34	36
12	0.10	0.10	0.15	23	23	109
13	0.16	0.20	0.20	31	31	185
14	3.15	3.52	4.43	23	23	865
15	0.3	0.31	0.19	105	105	291
16	25.83	20.14	**infinity**	34	34	**-**
17	28.95	30.00	28.30	**0**	**0**	**0**
18	**infinity**	**infinity**	**infinity**	**-**	**-**	**-**

- **Thời gian**



Hình 4: Thời gian tìm ra lời giải của các thuật toán qua các màn



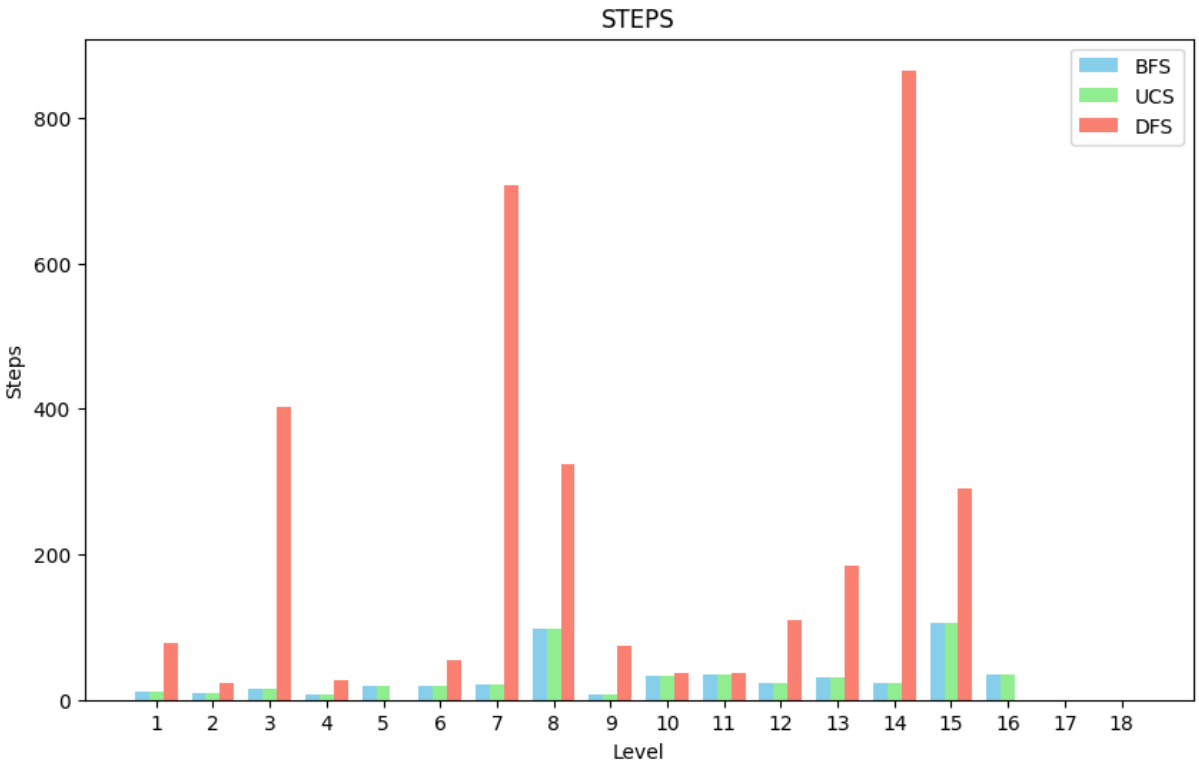
Hình 5: Thời gian tìm ra lời giải của các thuật toán qua các màn với trục thời gian nhỏ hơn

Dựa vào bảng kết quả và đồ thị mô hình hóa kết quả, ta có những nhận xét cơ bản sau :

- Mặc dù cả BFS và UCS đều tìm được giải pháp với số bước tương tự nhau nhưng nhờ việc tính toán cả chi phí cho việc đẩy thùng để tìm ra lời giải tối ưu, UCS trở nên vượt trội hơn BFS về mặt thời gian.

- Ở những màn khó, BFS và UCS vượt trội hơn hẳn về mặt thời gian so với DFS trong việc tìm ra lời giải. Ở những màn chơi mức độ dễ, thuật toán DFS cho ra kết quả nhanh chóng nhưng không quá nổi bật so với hai thuật toán trên.
- Màn chơi số 5 đặc biệt khó khăn cho cả ba thuật toán, và mất thời gian chạy lâu nhất để đưa ra kết quả. Điều này có thể do việc xếp chồng ba thùng và sự thiếu vắng các bức tường làm chướng ngại vật khiến cho số lượng hành động hợp lệ tăng lên và từ đó tạo ra nhiều lựa chọn đường đi hơn, dẫn đến việc tốn nhiều thời gian hơn để tìm ra lời giải.
- Khi bản đồ lớn, khả năng tìm ra lời giải của thuật toán DFS giảm đi đáng kể. Điều này là do DFS có xu hướng đi sâu vào từng nhánh một cách mù quáng, ví dụ như màn chơi 5 và 16, DFS gặp khó khăn do thời gian chạy lớn mà vẫn không tìm thấy đáp án.
- Với level 18, cả ba thuật toán đều không thể tìm ra lời giải trong vòng năm tiếng, điều này cho thấy rõ đây là một trong những màn chơi khó nhất.

- **Số bước đi**



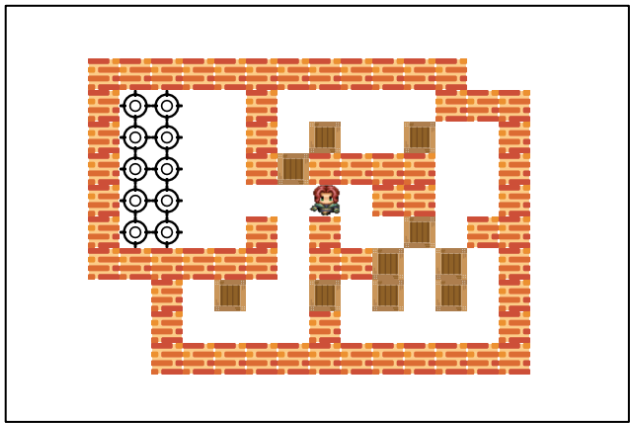
Hình 5: Số bước đi của lời giải các thuật toán qua các màn

Dựa vào bảng kết quả trực nghiệm và đồ thị mô hình hóa kết quả, ta có những nhận xét cơ bản sau :

- Số bước ở hai thuật toán BFS và UCS tương đồng ở mọi mức độ do mỗi bước đi có chi phí đồng nhất là 1.
- DFS có số bước để đi đến lời giải thường cao hơn nhiều so với BFS và UCS. Điều này có thể phản ánh việc thuật toán DFS khám phá sâu vào cây tìm kiếm mà không quay lại các nút đã đi qua trừ khi không còn lựa chọn nào khác, dẫn đến một lượng lớn các bước không cần thiết. Điều này được minh họa rõ nét ở cấp độ 14, DFS cần tới 865 bước để đạt được kết quả, trong khi BFS và UCS chỉ cần 23 bước.
- Level 17 thì cả 3 giải thuật đều trả về mảng rỗng do đây là bài toán không có lời giải.

KẾT LUẬN

- Level 17 là màn chơi không có lời giải.
- Màn chơi khó nhất chính là level 18, cả 3 giải thuật trên đều không thể giải được trong năm tiếng.



- Chung quy lại, thuật toán UCS là tốt nhất.