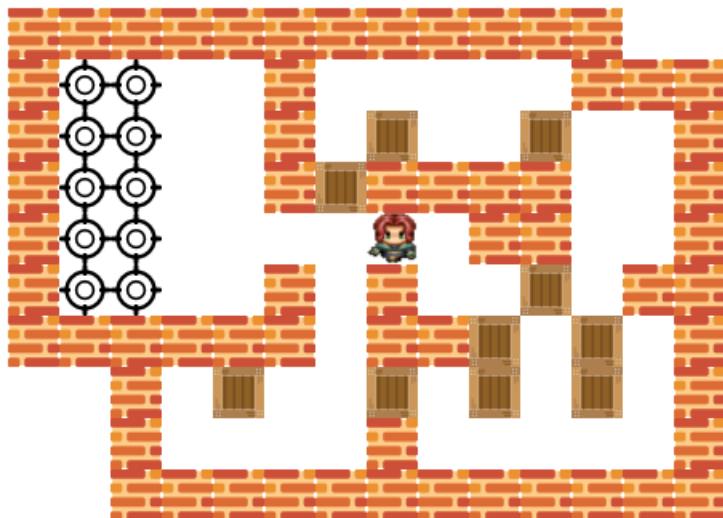


BÁO CÁO MÔN CS106

BT2 – A* algorithm for Sokoban



Giảng viên hướng dẫn:

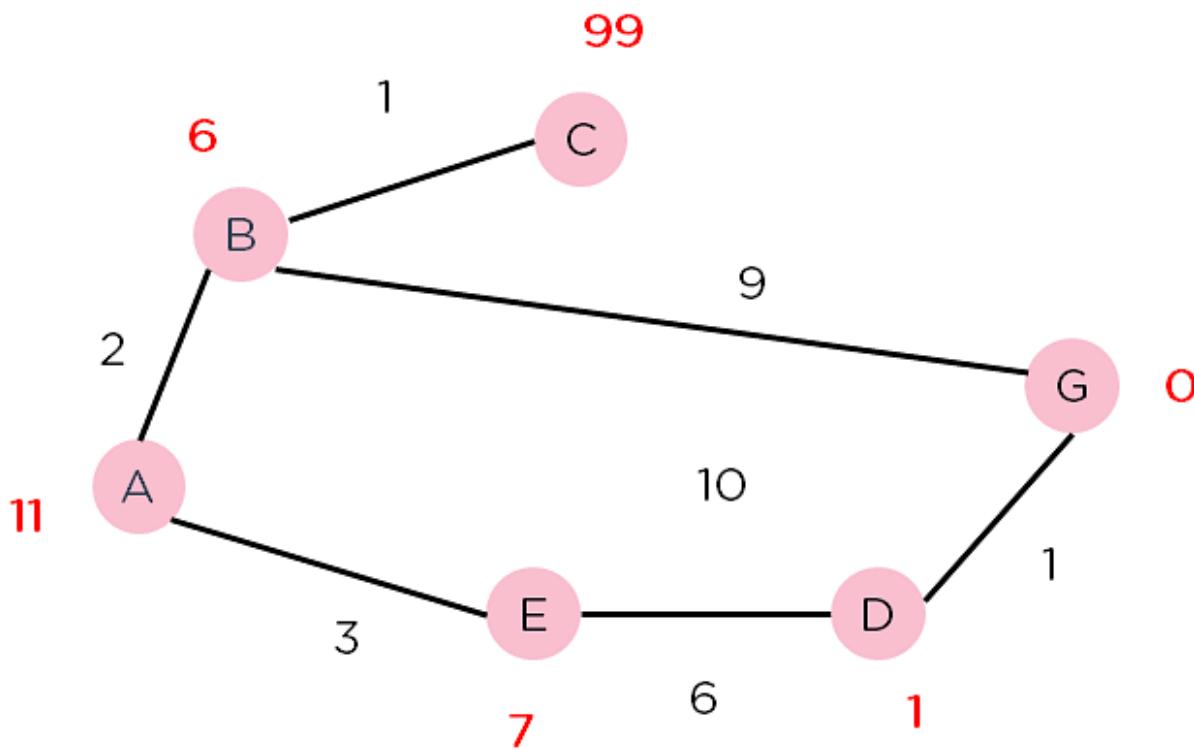
Lương Ngọc Hoàng

Sinh viên thực hiện:

Nguyễn Gia Bảo - 22520109

I. Giới thiệu về thuật toán A*

Theo Wikipedia, trong khoa học máy tính, A* (đọc là A sao) là thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm một đường đi từ một nút khởi đầu tới một nút đích cho trước (hoặc tới một nút thỏa mãn một điều kiện đích). Thuật toán này sử dụng một "đánh giá heuristic" để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của đánh giá heuristic này. Do đó, thuật toán A* là một ví dụ của tìm kiếm theo lựa chọn tốt nhất (best-first search).



Minh họa 1 đồ thị dùng thuật toán A* sẽ có thêm các giá trị heuristic (màu đỏ)

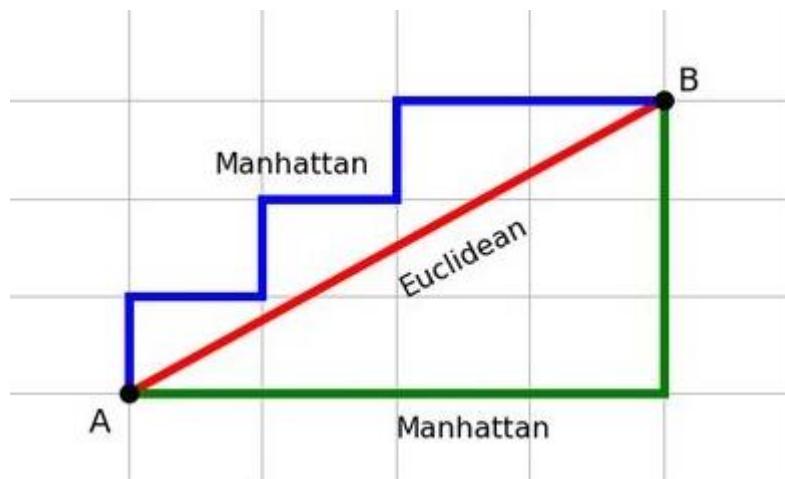
Sự khác biệt của thuật toán A* so với các thuật toán đã thực hiện ở bài tập trước là “hàm đánh giá heuristic”. Đây là một hàm tính toán, để ước lượng khoảng cách (chi phí ước lượng) từ node hiện tại đến node đích. Dựa vào giá trị này, chúng ta có thể cài đặt cho máy tính chọn những node “có khả năng” là ngắn nhất để đến được đích.

Tương tự như UCS, ta sẽ dùng một cấu trúc dữ liệu ưu tiên để lưu danh sách các node, nhưng thay vì giá trị ưu tiên là cost (chi phí thực hiện bước đi), ta sẽ dùng tổng: cost + heuristic.

➔ Việc xây dựng hàm heuristic rất quan trọng, vì nó có thể ảnh hưởng đến quyết định của máy tính tại từng node, từ đó có thể dẫn đến kết quả tốt/xấu nhất.

II. Cách xây dựng hàm heuristic

Xét 2 điểm A và B, ta sẽ có thể tính khoảng cách giữa chúng bằng 2 cách phổ biến: Manhattan distance và Euclidean distance



Minh họa 2 cách tính khoảng cách

Euclidean distance được tính là khoảng cách thẳng từ điểm A đến B (hay còn gọi là quãng đường chim bay), trong khi Manhattan distance được tính bằng tổng khoảng cách giữa tung độ và hoành độ giữa hai điểm.

Trong bài lần này chúng ta sẽ sử dụng Manhattan distance, bởi vì phương pháp này sẽ cho phép nhân vật của chúng ta di chuyển theo 4 hướng khác nhau.Thêm vào đó, trong các màn chơi, giữa 2 điểm bất kì trong bản đồ, đều sẽ xuất hiện vật cản (wall), nên chúng ta càng không thể đi thẳng đến đích như Euclidean distance.

```


def heuristic(posPlayer, posBox):
    # print(posPlayer, posBox)
    """A heuristic function to calculate the overall distance between the else boxes and the else goals"""
    distance = 0    # khai tao bien distance = 0
    completes = set(posGoals) & set(posBox) # complete = nhung box da duoc dat vao dung vi tri
    sortposBox = list(set(posBox).difference(completes)) # tao 1 list gom nhung box chua duoc xep vao goal
    sortposGoals = list(set(posGoals).difference(completes)) # tao 1 list gom nhung goal dang chua co box
    for i in range(len(sortposBox)):    #duyet qua tung box chua duoc dat vao goal
        # tinh do dai manhattan tu box toi goal = |x_box - x_goal| + |y_box - y_goal|
        distance += (abs(sortposBox[i][0] - sortposGoals[i][0])) + (abs(sortposBox[i][1] - sortposGoals[i][1]))
    return distance


```

Xây dựng hàm heuristic trong Python

III. Thực hiện thuật toán

Lần lượt chạy các màn chơi sử dụng A*, so sánh với thuật toán UCS, ta sẽ có bảng sau:

Level	A*			UCS		
	Time (second)	Number of explored nodes	Cost	Time (second)	Number of explored nodes	Cost
1	0.01	91	7	0.05	720	6
2	0.00	39	5	0.00	64	5
3	0.01	58	9	0.07	509	9
4	0.00	29	4	0.00	55	4
5	0.08	493	12	69.83	357203	10
6	0.01	214	14	0.01	250	14
7	0.06	791	10	0.49	6046	10
8	0.2	2353	65	0.2	2383	65
9	0.00	38	5	0.01	74	5
10	0.01	199	25	0.01	218	25
11	0.02	285	27	0.02	296	27
12	0.04	573	16	0.09	1225	16
13	0.12	1692	24	0.16	2342	24
14	0.86	8079	16	2.69	26352	16

15	0.24	2187	62	0.26	2505	62
16	0.27	1294	28	13.75	57275	22
17	24.48	0	0	23.47	0	0
18	-	-	-	-	-	-

Chú thích: “-” là thuật toán không trả về kết quả do giới hạn của phần cứng. Các màn chơi có cost 0 nghĩa là màn chơi không có lời giải.

Number of explored nodes là số node mà thuật toán phải duyệt qua trước khi đến được node đích (end state).

- Ta có thể thấy, tổng thời gian chạy và số node phải duyệt của A* qua các màn chơi là **vượt trội** hoàn toàn so với UCS, đặc biệt là:
- Ở level 5, A* nhanh hơn UCS khoảng 872,8 lần và số node phải duyệt qua ít hơn 724,5 lần.
 - Ở level 16, A* nhanh hơn UCS khoảng 51 lần và số node phải duyệt qua ít hơn khoảng 44,2 lần.

Tuy nhiên, khi xét đến kết quả trả về thì ở những **level 1, 5, 16** UCS cho ra những phương án tối ưu hơn về cost.

Nhưng level 18 vẫn là “bất khả thi” với 2 thuật toán trên.

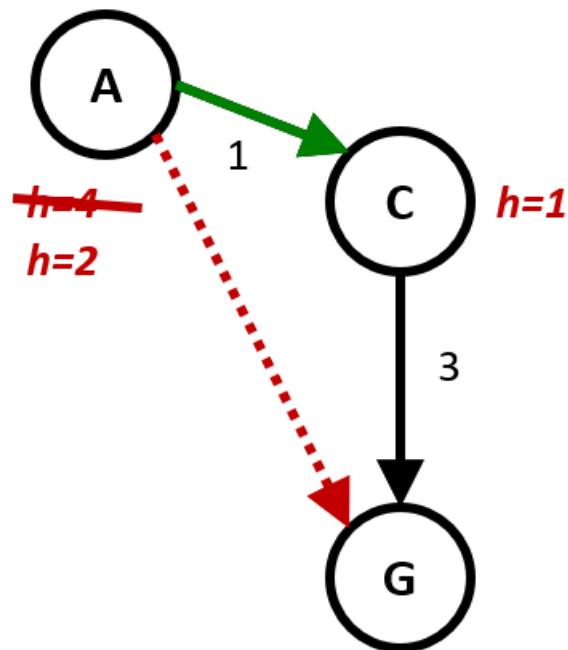
Để giải thích cho sự khác biệt này, ta sẽ có thể dễ dàng thấy được tốc độ của A* nằm ở “sự chỉ dẫn” của hàm heuristic. Với giá trị heuristic, A* có thể nhanh chóng chọn ra những bước đi khiến người chơi đến gần với end state hơn (do không cần mở ra những node không dẫn đến end state), so với việc tìm kiếm trong tất cả các bước đi có thể như UCS. Tuy nhiên, cũng do hàm heuristic chưa được tối ưu hoàn toàn, nên trong những level nhất định, nó sẽ chỉ ra những con đường không tối ưu bằng UCS.

IV. Cải tiến heuristic

Ta nhận thấy, sự khác biệt giữa A* và UCS nằm ở hàm heuristic, độ hiệu quả của A* sẽ phụ thuộc vào giá trị trả về của hàm heuristic.

➔ UCS là một trường hợp đặc biệt của A* khi tất cả giá trị heuristic đều bằng 0.

Như chúng ta đã biết, A* sẽ thực sự tối ưu khi heuristic là nhất quán (consitent) đối với graph search. Nghĩa là heuristic của 1 cung bất kì sẽ nhỏ hơn hoặc bằng chi phí thực sự giữa cung đó.



$$h(A) - h(C) \leq \text{cost}(A \text{ to } C)$$

➔ Càng giảm giá trị heuristic thì kết quả của A* sẽ càng giống với UCS.

Sau một loạt những thử nghiệm, em kết luận được nên dùng giá trị heuristic/2.5 thì sẽ được kết quả tối ưu của A* trên các màn chơi (trừ level 17 và level 18).

```

203
204 def heuristic(posPlayer, posBox):
205     # print(posPlayer, posBox)
206     """A heuristic function to calculate the overall distance between the else boxes and the else goals"""
207     distance = 0 # khai tao bien distance = 0
208     completes = set(posGoals) & set(posBox) # complete = nhung box da duoc dat vao dung vi tri
209     sortposBox = list(set(posBox).difference(completes)) # tao 1 list gom nhung box chua duoc xep vao goal
210     sortposGoals = list(set(posGoals).difference(completes)) # tao 1 list gom nhung goal dang chua co box
211     for i in range(len(sortposBox)): #duyet qua tung box chua duoc dat vao goal
212         # tinh do dai manhattan tu box toi goal = |x_box - x_goal| + |y_box - y_goal|
213         distance += (abs(sortposBox[i][0] - sortposGoals[i][0])) + (abs(sortposBox[i][1] - sortposGoals[i][1]))
214
215 return distance/2.5 # giam gia tri cua heuristic di 2.5 lan

```

Hàm heuristic trả về giá trị heuristic giảm đi 2.5 lần

Lúc này, ta sẽ có được một bảng so sánh mới:

Level	A* cũ			A* cải tiến			UCS		
	Time (second)	Number of explored nodes	Cost	Time (second)	Number of explored nodes	Cost	Time (second)	Number of explored nodes	Cost
1	0.01	91	7	0.02	268	6	0.05	720	6
2	0.00	39	5	0.00	52	5	0.00	64	5
3	0.01	58	9	0.01	75	9	0.07	509	9
4	0.00	29	4	0.00	44	4	0.00	55	4
5	0.08	493	12	0.08	5648	10	69.83	357203	10
6	0.01	214	14	0.01	237	14	0.01	250	14
7	0.06	791	10	0.27	2836	10	0.49	6046	10
8	0.2	2353	65	0.2	2385	65	0.2	2383	65
9	0.00	38	5	0.00	36	5	0.01	74	5
10	0.01	199	25	0.02	215	25	0.01	218	25
11	0.02	285	27	0.02	290	27	0.02	296	27
12	0.04	573	16	0.08	988	16	0.09	1225	16
13	0.12	1692	24	0.18	2171	24	0.16	2342	24
14	0.86	8079	16	2.47	20232	16	2.69	26352	16
15	0.24	2187	62	0.3	2430	62	0.26	2505	62
16	0.27	1294	28	1.47	5781	22	13.75	57275	22
17	24.48	0	0	24.48	0	0	23.47	0	0
18	-	-	-	-	-	-	-	-	-

Từ bảng trên , chúng ta có thể thấy khi giảm giá trị heuristic đi, sẽ khiến thời gian chạy cũng như số node phải duyệt của A* tăng lên, nhưng lượng tăng về thời gian là không đáng kể, A* mới này vẫn có thời gian thực thi tốt hơn rất nhiều so với UCS ở các level phức tạp như level 5 và level 16. Thêm nữa, kết quả A* mới cũng đã đưa ra được những phương pháp tối ưu như UCS, ta có thể thấy ở các **level 1, 5, 16** giá trị cost đã tối ưu hơn so với trước khi giảm giá trị heuristic. Có sự đánh đổi về thời gian để đổi lấy kết quả tối ưu khi thực hiện A*, nhưng nhìn chung, sự đánh đổi về thời gian này là “chấp nhận được” khi chơi những level có độ phức tạp lớn, thời gian tăng lên ít nhưng kết quả luôn là đường đi tối ưu nhất.

Tuy nhiên, A* mới này vẫn chưa thể giải được level 18.

➔**Tổng kết:** A* với hàm heuristic bị giảm đi 2.5 lần là lời giải tối ưu cho các level từ 1 đến 16 khi ta muốn tìm được đường đi tốt nhất, còn khi muốn ưu tiên tốc độ thì chúng ta nên sử dụng A* với hàm heuristic mặc định

—HẾT—