Report – Thực Hành Tuần 04

Cài Đặt Bài Toán 4 Góc Mê Cung Và Heuristic Tương Ứng

Họ tên: Nguyễn Sanh Tuấn.

MSSV: 1760227.

Lớp: 17CK2_ca 3 thực hành chiều t7.

**** Tìm hiểu về Heuristic cho bài toán bốn góc mê cung****

Hàm này nhận hai tham số đầu vào là trạng thái state và
bài toán problem, và trả về chi phí ước lượng từ state đến đích. Lưu ý là, để A* tìm được đường đi tối ưu trong Graph Search, heuristic của bạn phải thỏa tính nhất quán (consistency). Thông thường, cách thiết kế một heuristic như sau. Đầu tiên, bạn làm dễ bài toán để có được một heuristic thỏa tính chấp nhận được (admissibility); chẳng hạn, Pacman giống như superman, có thể nhảy đến một ô bất kỳ chỉ trong một bước (để ý là, nếu bạn càng làm dễ bài toán so với bài toán gốc ban đầu thì heuristic sẽ càng dễ tính, nhưng tác dụng định hướng sẽ càng giảm đi; ta nên thiết kế một heuristic trung dung, vừa không quá chính xác để có thể tính được nhanh và vừa không quá không chính xác để có thể giúp định hướng tìm kiếm). Và heuristic chấp nhận được này thường sẽ nhất quán (lưu ý là, tính nhất quán sẽ kéo theo tính chấp nhận được, nhưng chiều ngược lại thì không chắc; ở đây, khi bạn thiết kế heuristic bằng cách làm dễ bài toán, chiều ngược lại thường sẽ xảy ra).
Ngoài ra, để ý là heuristic cần trả về giá trị không âm, và trả về giá trị 0 với trạng thái đích

Yêu cầu cài đặt hoàn chỉnh class CornersProblem(search.SearchProblem): *trong file searchAgents.py*

1. **Ý tưởng viết các hàm trong class.**

- def __init__(self, startingGameState):

Đây là hàm lưu trữ các bức tường, vị trí bắt đầu và góc của pacman.

⇨ YOUR CODE HERE: Ta sẽ khởi tạo trạng thái bắt đầu bao gồm trạng thái bắt đầu và các góc không được truy cập.

- def getStartState(self):

Đây là hàm trả về trạng thái bắt đầu(trong không gian trạng thái của _Init_ ta khởi tạo ở trên, không phải trại thái pacman đẩy dủ không gian).

⇨ YOUR CODE HERE: Trả về trạng thái mà ta đã khởi tạo ở hàm def __init__(self, startingGameState):

- def isGoalState(self, state):

Hàm này kiểm tra xem trạng thái khởi tạo ban đầu có phải là trạng trạng thái mục tiêu ta cần tìm kiếm.

⇨ YOUR CODE HERE: trạng thái [1] chứa các góc không truy cập. Nếu nó trống, chiều dài của nó sẽ là 0, có nghĩa là tất cả các góc đã được truy cập.

- def getSuccessors(self, state):

Hàm này Trả về trạng thái kế, các hành động họ yêu cầu và chi phí là 1.

Như đã lưu ý trong search.py:

Đối với một trạng thái nhất định, điều này sẽ trả về một danh sách ba lần, (vị trí kế tiếp, đường đi, chi phí bước đi), trong đó 'vị trí kế vị' là vị trí kế thừa hiện tại, trạng thái, 'bước đi ' là hành động cần thiết để đạt được điều đó và 'chi phí bước đi - stepcost' là chi phí gia tăng của việc mở rộng cho người kế nhiệm đó.

⇨ YOUR CODE HERE: sử dụng vòng lặp for duyệt các hành động của các hướng đi NORTH, SOUTH, EAST, WEST.

- Khởi tạo trạng thái kế tiếp:
  successors = []
- Vòng lặp:
  for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:
- Khởi tạo trạng thái [0] chứa tọa độ vị trí hiện tại.
- Tính tọa độ nút kế tiếp
- Kiểm tra nếu một bức tường đang bị tấn công.
- Nếu không thì:
  - kiểm tra xem vị trí mới có phải là góc không truy cập không
  - nếu không, hãy thêm nó vào danh sách không mong muốn mới
  - thêm tọa độ mới và danh sách không mong muốn vào người kế nhiệm
- Cuối cùng trả về vị trí đích đến.

• def getCostOfActions(self, actions):

Trả về chi phí của một chuỗi hành động cụ thể, tức hàm này dùng để tính chi phí. Code đã có sẵn nên không phải làm.

****** Hàm Heuristic cho 4 góc mê cung******

• def cornersHeuristic(state, problem):

Hàm này cần xác định trạng thái: trạng thái tìm kiếm hiện tại., sẽ luôn trả về một số bị giới hạn dưới con đường ngắn nhất từ trạng thái bắt đầu đến mục tiêu của vấn đề.

⇨ Ý tưởng:

- Khởi tạo tọa độ các góc, và các bức tường mê cung.
  corners = problem.corners
  walls = problem.walls

Nội dung cần làm trong hàm này:

Khởi tạo Hàm này tính toán:

a. khoảng cách Manhattan giữa vị trí hiện tại và thức ăn gần nhất

b. khoảng cách Manhattan giữa thức ăn gần nhất và thực phẩm xa nhất với nó

2 giá trị này được thêm vào và trả về là heuristic.

⇨ Ý tưởng thuật toán:

- Chứa tọa các điểm chưa được truy cập “nonvisited”
  nonvisited = state[1]
- Biến chứa vị trí hiện tịa cảu pacman
  currentpos = state[0]
- Biến chứa khoảng cách để lưu trữ khoảng cách Manhattan giữa vị trí hiện tại và từng vị trí cho pacman ăn.
  distances = []
  heuristic = 0
- nếu không còn thức ăn, hàm trả về 0.
- vòng lặp để tính khoảng cách Manhattan giữa vị trí hiện tại và từng vị trí thức ăn.
  for elem in nonvisited:

```
                    dist = abs(elem[0] - currentpos[0]) + abs(elem[1] - currentpos[1])
                    distances.append((dist,elem))
```

- khoảng cách được sắp xếp theo thứ tự tăng dần

```
distances = sorted(distances)
```

- khoảng cách Manhattan giữa vị trí hiện tại và vị trí thức ăn gần nhất được thêm vào heuristic

```
heuristic = heuristic + distances[0][0]
```

- Vị trí hiện tại được thay đổi thành vị trí của thức ăn gần nhất

```
closest_food = distances[0][1]
```

- Loại bỏ thực phẩm gần nhất khỏi danh sách không mong muốn

```
nonvisited = list(nonvisited)
```

- nếu không còn thức ăn, hãy quay lại heuristic.
- vòng lặp để tính khoảng cách Manhattan giữa thức ăn gần nhất và từng địa điểm thức ăn còn lại

```
distances = []
for elem in nonvisited:
    dist = abs(elem[0] - closest_food[0]) + abs(elem[1] - closest_food[1])
```

- khoảng cách được sắp xếp theo thứ tự giảm dần

```
distances = sorted(distances, reverse=True)
```

- khoảng cách Manhattan giữa thực phẩm gần nhất và thực phẩm xa nhất với nó được thêm vào heuristic

```
heuristic = heuristic + distances[0][0]
```

2. Phần code của chương trình.

```
class CornersProblem(search.SearchProblem):
    """
    This search problem finds paths through all four corners of a layout.

    You must select a suitable state space and successor function
    """

    def __init__(self, startingGameState):
        """
        Stores the walls, pacman's starting position and corners.
        """
        self.walls = startingGameState.getWalls()
        self.startingPosition = startingGameState.getPacmanPosition()
        top, right = self.walls.height-2, self.walls.width-2
        self.corners = ((1,1), (1,top), (right, 1), (right, top))
        for corner in self.corners:
            if not startingGameState.hasFood(*corner):
                print('Warning: no food in corner ' + str(corner))
        self._expanded = 0 # DO NOT CHANGE; Number of search nodes expanded
        # Please add any code here which you would like to use
        # in initializing the problem
        "*** YOUR CODE HERE ***"
        self.right = right
```

```
26.         self.top = top
27.
28.     def getStartState(self):
29.         """
30.         Returns the start state (in your state space, not the full Pacm
    an state
31.         space)
32.         """
33.         "*** YOUR CODE HERE ***"
34.         allCorners = (False, False, False, False)
35.         start = (self.startingPosition, allCorners)
36.         return start
37.         util.raiseNotDefined()
38.
39.     def isGoalState(self, state):
40.         """
41.         Returns whether this search state is a goal state of the proble
    m.
42.         """
43.         "*** YOUR CODE HERE ***"
44.         corners = state[1]
45.         boolean = corners[0] and corners[1] and corners[2] and corners[
    3]
46.         return boolean
47.         util.raiseNotDefined()
48.
49.     def getSuccessors(self, state):
50.         """
51.         Returns successor states, the actions they require, and a cost
    of 1.
52.
53.          As noted in search.py:
54.             For a given state, this should return a list of triples, (s
    uccessor,
55.             action, stepCost), where 'successor' is a successor to the
    current
56.             state, 'action' is the action required to get there, and 's
    tepCost'
57.             is the incremental cost of expanding to that successor
58.         """
59.
60.         successors = []
61.         for action in [Directions.NORTH, Directions.SOUTH, Directions.E
    AST, Directions.WEST]:
62.             # Add a successor state to the successor list if the action
     is legal
63.             # Here's a code snippet for figuring out whether a new posi
    tion hits a wall:
```

```
64.             #   x,y = currentPosition
65.             #   dx, dy = Actions.directionToVector(action)
66.             #   nextx, nexty = int(x + dx), int(y + dy)
67.             #   hitsWall = self.walls[nextx][nexty]
68.
69.             "*** YOUR CODE HERE ***"
70.             x,y = state[0]
71.             holdCorners = state[1]
72.             dx, dy = Actions.directionToVector(action)
73.             nextx, nexty = int(x + dx), int(y + dy)
74.             hitsWall = self.walls[nextx][nexty]
75.             newCorners = ()
76.             nextState = (nextx, nexty)
77.             if not hitsWall:
78.                 if nextState in self.corners:
79.                     if nextState == (self.right, 1):
80.                         newCorners = [True, holdCorners[1], holdCorners
    [2], holdCorners[3]]
81.                     elif nextState == (self.right, self.top):
82.                         newCorners = [holdCorners[0], True, holdCorners
    [2], holdCorners[3]]
83.                     elif nextState == (1, self.top):
84.                         newCorners = [holdCorners[0], holdCorners[1], T
    rue, holdCorners[3]]
85.                     elif nextState == (1,1):
86.                         newCorners = [holdCorners[0], holdCorners[1], h
    oldCorners[2], True]
87.                     successor = ((nextState, newCorners), action,  1)
88.                 else:
89.                     successor = ((nextState, holdCorners), action, 1)
90.                 successors.append(successor)
91.
92.         self._expanded += 1 # DO NOT CHANGE
93.         return successors
94.
95.     def getCostOfActions(self, actions):
96.         """
97.         Returns the cost of a particular sequence of actions.  If those
     actions
98.         include an illegal move, return 999999.  This is implemented fo
    r you.
99.         """
100.             if actions == None: return 999999
101.             x,y= self.startingPosition
102.             for action in actions:
103.                 dx, dy = Actions.directionToVector(action)
104.                 x, y = int(x + dx), int(y + dy)
105.                 if self.walls[x][y]: return 999999
```

```
106.             return len(actions)
107.
108.     def cornersHeuristic(state, problem):
109.         """
110.         A heuristic for the CornersProblem that you defined.
111.
112.           state:   The current search state
113.                    (a data structure you chose in your search problem
    )
114.
115.           problem: The CornersProblem instance for this layout.
116.
117.         This function should always return a number that is a lower b
    ound on the
118.         shortest path from the state to a goal of the problem; i.e.
    it should be
119.         admissible (as well as consistent).
120.         """
121.         corners = problem.corners # These are the corner coordinates
122.         walls = problem.walls # These are the walls of the maze, as a
     Grid (game.py)
123.
124.         "*** YOUR CODE HERE ***"
125.         position = state[0]
126.         stateCorners = state[1]
127.         corners = problem.corners
128.         top, right = problem.walls.height-2, problem.walls.width-2
129.         cornerNot = []
130.         for c in corners:
131.             if c == (1,1):
132.                 if not stateCorners[3]:
133.                     cornerNot.append(c)
134.             if c == (1, top):
135.                 if not stateCorners[2]:
136.                     cornerNot.append(c)
137.             if c == (right, top):
138.                 if not stateCorners[1]:
139.                     cornerNot.append(c)
140.             if c == (right, 1):
141.                 if not stateCorners[0]:
142.                     cornerNot.append(c)
143.
144.         cost = 0
145.         currPosition = position
146.         while len(cornerNot) > 0:
147.             distArr= []
148.             x = 0
149.             for c in range(0, len(cornerNot)):
```

```
150.                 corner = cornerNot[c]
151.                 x = x + 1
152.                 dist = util.manhattanDistance(currPosition, corner)
153.                 distArr.append(dist)
154.             minDist = min(distArr)
155.             cost += minDist
156.             minDistI= distArr.index(minDist)
157.             currPosition = cornerNot[minDistI]
158.             del cornerNot[minDistI]
159.
160.         return cost
```

3. Kiểm tra kết quả cài đặt.

Chạy các câu lệnh sau: (kiểm tra nội dung cài đặt trong ***class CornersProblem(search.SearchProblem):***)

python pacman.py -l tinyCorners -p SearchAgent -a "fn=bfs,prob=CornersProblem"

python pacman.py -l mediumCorners -p SearchAgent -a "fn=bfs,prob=CornersProblem"

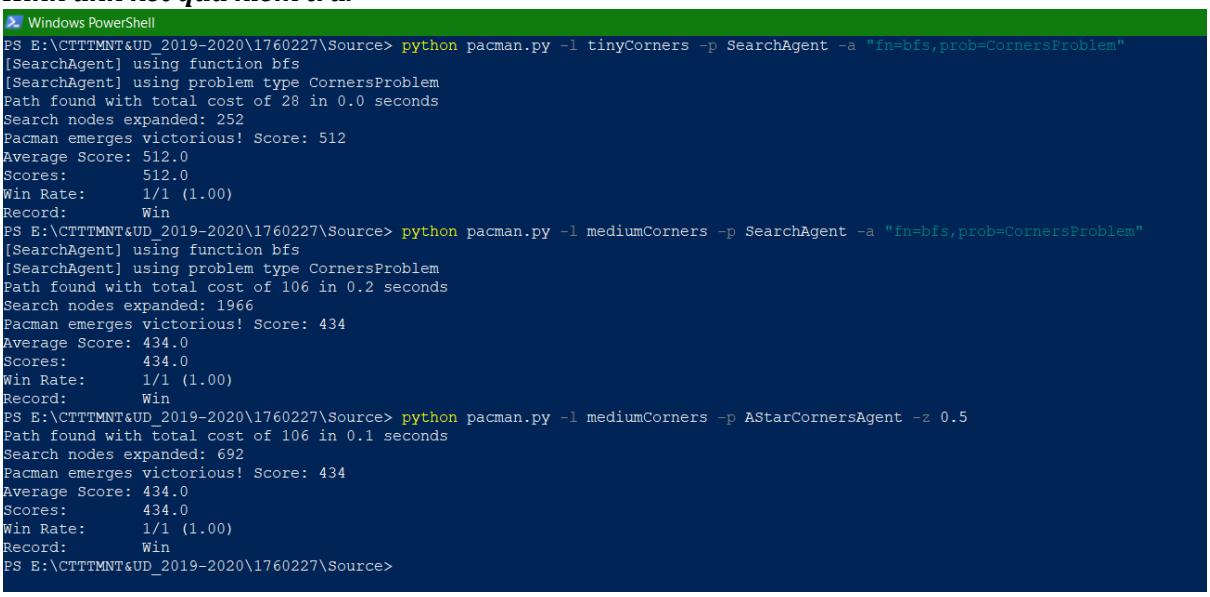
Thuật toán Heuristic

python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5

với StarCornersAgent là "shortcut" của:

-p SearchAgent -a "fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic"

Hình ảnh kết quả kiểm tra.



----------THE END----------