ฉันท์สินี เมืองหนู 63070501221

Homework 1

Search on the sliding puzzle

ฉันท์สินี เมืองหนู 63070501221

Breadth first search for the 3x3 sliding puzzle

1. Explanation of the BFS source code in each part and answer 4 questions below.

1.1 What datatype is the variable ‘Node’?

1.2 What does function ‘swap’ do?

1.3 What does variable ‘nodelist’ do?

1.4 what does variable ‘costlist do?

BFS source code description:

Source code นี้เป็นตัวอย่างของการใช้การค้นหาแบบ BFS เพื่อแก้ปัญหาเกม Sliding Puzzle ซึ่งเป็นเกมที่ผู้เล่นต้องจัดเรียงตัวเลขในกระดานให้เรียงลำดับตามเป้าหมาย ซึ่งมีตัวเลข 0 ซึ่งเป็นตัวว่างที่สามารถเคลื่อนที่ได้ในทิศทางข้างเคียง เพื่อให้ได้ลำดับเรียงที่ถูกต้องตาม Goal ในที่นี้ต้องการให้ตัวเลขในกระดานเรียงกันตั้งแต่ 1-8

Source code Part 1: Node definitions

```
12  class Node:
13      def __init__(self, state, action, parent, cost):
14          self.s = state
15          self.a = action
16          self.p = parent
17          self.c = cost
18          self.expand = 0
19
20      def printstate(self):
21          print(self.s)
22
23      def printaction(self):
24          print(self.a)
```

Source code ในส่วนนี้ เริ่มต้นด้วยการสร้างคลาส ‘Node’ โดยใช้ฟังก์ชัน ‘__init__’ ซึ่งคลาส ‘Node’ นี้จะใช้เก็บข้อมูลเกี่ยวกับ Node ในการค้นหา Sliding Puzzle ประกอบไปด้วย สถานะปัจจุบัน (State), การกระทำที่ทำให้เกิด Node ปัจจุบัน (Action), Node แม่ (Parent), ระยะในการเคลื่อนที่มายัง Node ปัจจุบัน (Cost), และค่าสถานะการขยาย Node (Expand) ในที่นี้ Expand = 0 หมายความว่า ไม่มี Node ลูก

นอกจากนี้ในคลาส ‘Node’ ยังมีการใช้ฟังก์ชัน ‘printstate’ และ ‘printaction’ ที่ใช้ในการแสดงผลสถานะและการกระทำของ Node อีกด้วย

Source Code Part 2: Action Functions

```
27  def swap(array,p1,p2,p3,p4):
28      temp=array[p1,p2]
29      array[p1,p2]=array[p3,p4]
30      array[p3,p4]=temp
31      return array
```

ฟังก์ชัน ‘swap’ ใช้สำหรับสลับค่าของ 2 ตำแหน่งใน Array 2 มิติ ซึ่งในที่นี้ช่วยในการสลับและย้ายตัวเลขในตาราง Sliding Puzzle ให้สามารถแก้ปัญหาของเกมนี้ได้

```
33  def goup(value):
34      value = np.array(value)
35      a=(np.where(value==0))
36      if a[0][0]-1<0:
37          return value
38      else:
39          ans=swap(value,a[0][0],a[1][0],a[0][0]-1,a[1][0])
40          return ans
41
42  def godown(value):
43      value = np.array(value)
44      a=(np.where(value==0))
45      if a[0][0]+1==value.shape[1]:
46          return value
47      else:
48          ans=swap(value,a[0][0],a[1][0],a[0][0]+1,a[1][0])
49          return ans
50
51  def goleft(value):
52      value = np.array(value)
53      a=(np.where(value==0))
54      if a[1][0]-1<0:
55          return value
56      else:
57          ans=swap(value,a[0][0],a[1][0],a[0][0],a[1][0]-1)
58          return ans
59
60  def goright(value):
61      value = np.array(value)
62      a=(np.where(value==0))
63      if a[1][0]+1==value.shape[1]:
64          return value
65      else:
66          ans=swap(value,a[0][0],a[1][0],a[0][0],a[1][0]+1)
67          return ans
```

ฟังก์ชัน ‘goup’, ‘godown’, ‘goleft’, และ ‘goright’ ใช้สำหรับเลื่อนตัวเลขว่างในเกม Sliding Puzzle ไปในทิศทางที่กำหนด คือ ย้ายช่องว่างไปด้านบน ด้านล่าง ด้านซ้าย และด้านขวา ตามลำดับ

Source Code Part 3: Main

```
71  maxdepth = 9999
72  #start = np.array([[1,2,3],[4,5,6],[0,7,8]])  # change your starting here
73  start = np.array([[4,1,0],[7,2,3],[5,8,6]])   # change your starting here
74  goal = np.array([[1,2,3],[4,5,6],[7,8,0]])
75
76  root = Node(start,0,0,0)
77  nodelist = [root]
78  costlist = np.array([0])
79  nodecount = 1
```

เริ่มต้นจากการตั้งค่า ‘maxdepth’ ซึ่งเป็นค่าสูงสุดของความลึกในการค้นหา ในที่นี้ตั้งค่าเป็น 9999 เพื่อให้มีความลึกในการค้นหามากที่สุด โดยการแก้ปัญหา Sliding Puzzle จะต้องมีการกำหนดสถานะเริ่มต้นและสถานะปลายทาง ในที่นี้ใช้ NumPy Array สำหรับแทนสถานะ start และ goal

จากบรรทัดที่ 76 เป็นการสร้าง Node แรก (Root Node) โดยใช้สถานะเริ่มต้น ‘Start’ และตั้งค่า ‘action’, ‘parent’ และ ‘cost’ เป็น 0

มีการกำหนด nodecount =1 ในบรรทัดที่ 79 เพื่อนับจำนวน Node ที่ถูกสร้างในการค้นหา ในที่นี้นับ Root Node เป็น Node แรก

```
81  found = None
82  while found==None:
83      # Search for a node to expand
84      breadth = np.argmin(costlist)
85      costlist[breadth] = maxdepth        # Eliminate found node from the list
86      parent = nodelist[breadth]
87
88      # Expand
89      parent.expand = 1   # Mark expanded
90      depth = parent.c + 1
91      up = Node(goup(parent.s), 'up', parent, depth)
92      down = Node(godown(parent.s), 'down', parent, depth)
93      left = Node(goleft(parent.s), 'left', parent, depth)
94      right = Node(goright(parent.s), 'right', parent, depth)
95      nodelist.extend([up,down,left,right])
96      costlist = np.append(costlist,[depth,depth,depth,depth])
97
98      # Check if a solution is found
99      if sum(sum(up.s != goal)) == 0:
100         found = up
101     if sum(sum(down.s != goal)) == 0:
102         found = down
103     if sum(sum(left.s != goal)) == 0:
104         found = left
105     if sum(sum(right.s != goal)) == 0:
106         found = right
107
108     nodecount = nodecount + 4
```

Source code ข้างต้นมีหน้าที่ขยาย Node และตรวจสอบว่าพบ Goal หรือไม่ โดย Source code ดังกล่าวมีการใช้คำสั่งหลัก ๆ และมีขั้นตอนการทำงานต่าง ๆ ดังนี้

1. สร้างตัวแปร ‘found’ และกำหนดให้เป็น ‘None’ เพื่อรอการค้นหาและค้นหา Node ที่มีคำตอบไว้ในตัวแปรนี้ และเริ่มการวนลูปแบบไม่มีที่สิ้นสุดโดยรอการค้นหาจนกว่า ‘found’ จะไม่เป็น ‘None’
2. ค้นหา Node ที่จะขยายต่อไป โดยการเลือก Node ที่มีค่า cost น้อยที่สุดใน ‘costlist’ โดยคำสั่ง ‘np.argmin’ เก็บตำแหน่งนี้ไว้ในตัวแปร ‘breadth และนำ Node ที่ถูกเลือกมาใช้เป็น Parent Node สำหรับการขยาย Node หลังจากนั้นก็ตั้งค่า ‘expand’ ของ Parent Node เป็น 1 เพื่อระบุว่า Node นี้ถูกขยายแล้ว
3. บรรทัดที่ 90 มีการใช้คำสั่ง ‘depth = parent.c +1’ เพื่อคำนวณความลึกของ Node ลูก โดยเพิ่มความลึกของ Parent Node ไปอีก 1
4. สร้าง Node ลูก เป็น ‘up’, ‘down’, ‘left’, และ ‘right’ โดยใช้ฟังก์ชัน ‘goup’, ‘godown’, ‘goleft’, และ ‘goright’ เพื่อทำการย้ายเลขต่าง ๆ ใน Sliding Puzzle และทำการเพิ่ม Node ลูก เข้าไปใน ‘nodelist’ เพื่อใช้ในการขยาย Node หลังจากนั้นเพิ่มค่า cost ของ Node ลูก ที่ถูกสร้างใหม่นี้ลงใน ‘costlist’ โดยใช้ค่าความลึก ‘depth’
5. ตรวจสอบว่าได้คำตอบหรือยัง โดยการตรวจสอบค่าความต่างระหว่างสถานะของ Node ลูก และสถานะปลายทาง (Goal) ถ้าหากค่าความต่างเป็น 0 แสดงว่าได้คำตอบแล้ว และกำหนด ‘found’ ให้เป็น Node ลูกที่พบคำตอบ และจะออกจากลูป ‘while’ หลังจากนั้นนับจำนวน Node ที่ถูกสร้างขึ้นในรอบการค้นหา

Source Code Part 4: Print Solution

```
110  # Print solution
111  print('Solution found in ' + str(found.c) + ' moves')
112  print('Generated ' + str(nodecount) + ' nodes')
113  solution = []
114  while found.c > 0 :
115      solution.append(found)
116      found = found.p
117
118  print(start)
119  for i in range(len(solution)-1,-1,-1):
120      solution[i].printaction()
121      solution[i].printstate()
```

Source Code ส่วนนี้เป็น Code แสดงผลลัพธ์ของการค้นหา รวมถึงแสดงคำตอบโดยเริ่มจากการแสดงความลึกของการเคลื่อนที่ของตัวเลขในตาราง Sliding Puzzle เพื่อหาคำตอบ ‘found.c’, แสดงจำนวน Node ที่ถูกสร้าง ‘nodecount’, และแสดงลำดับของการเคลื่อนที่ที่ทำให้เกิดสถานะปลายทาง ‘Goal’

Questions:

1.1 What datatype is the variable ‘Node’?

ตัวแปร ‘Node’ เป็นชนิดข้อมูลของ Class ที่ถูกสร้างขึ้นใน Source Code ไม่ใช่ตัวแปรที่เก็บค่าข้อมูล แต่สร้างขึ้นเพื่อสร้าง Object ของ Class ‘Node’ ซึ่งจะมีข้อมูลเกี่ยวกับสถานะและคุณสมบัติของ Node ในการค้นหา Sliding Puzzle

1.2 What does function ‘swap’ do?

ฟังก์ชัน ‘swap’ ใช้สำหรับสลับค่าของ 2 ตำแหน่งใน Array 2 มิติ ในที่นี้ทำหน้าที่ย้ายตัวเลขว่างใน Sliding Puzzle ให้เคลื่อนที่ไปในตำแหน่งต่าง ๆ โดยทำการรับพารามิเตอร์เป็น array และตำแหน่งของ 2 คู่ แถวและคอลัมน์ที่ต้องการสลับ (p1, p2, p3, p4) จากนั้นสลับค่าของคู่แถวและคอลัมน์โดยทำให้ค่าตำแหน่งที่ 1 เท่ากับตำแหน่งที่ 2 และส่งค่า Array ที่ถูกแก้ไขกลับ

1.3 What does variable ‘nodelist’ do?

ตัวแปร ‘nodelist’ เป็น list ที่มีหน้าที่เก็บ Node ที่เกิดขึ้นทุกขั้นตอนในการค้นหา Sliding Puzzle โดยเริ่มต้นด้วย Node เริ่มต้น (root) และเพิ่ม Node ลูก ที่ถูกสร้างขึ้นในแต่ละรอบของการค้นหาเข้าไปใน list นี้ ทำให้เราสามารถเข้าถึง Node ทั้งหมดที่เกิดขึ้นในการค้นหาได้

1.4 what does variable ‘costlist do?

ตัวแปร ‘costlist’ เป็น Array ที่ใช้เก็บค่า cost สำหรับแต่ละ Node ที่ถูกสร้างขึ้นในการค้นหาแบบ BFS ในเกม Sliding Puzzle นี้ โดย ‘costlist’ ช่วยในการเรียงลำดับการขยายตัวเองของ Node ลูก ในแต่ละการค้นหาโดยการเลือก Node ที่จะขยายตัวเองจาก ‘nodelist’ แต่จะขึ้นอยู่กับค่า cost ที่น้อยที่สุด ซึ่งจะถูกคัดเลือกโดยใช้คำสั่ง ‘np.argmin(costlist)’ ดังนั้นค่า cost ที่เก็บไว้ในตัวแปร ‘costlist’ นี้มีหน้าที่ในการควบคุมและกำหนดลำดับของการค้นหาเพื่อให้การค้นหาครอบคลุมในทุก ๆ Node

2. Modify the source code that implement A* search and explanation

(A* search reduce the number of nodes a lot. Use heuristic h2 (Manhattan distance)

ผลลัพธ์ที่ได้จาก BFS search เปรียบเทียบกับผลลัพธ์ที่ได้จาก A* search:

```
Solution found in 10 moves
Generated 718381 nodes
[[4 1 0]
 [7 2 3]
 [5 8 6]]
down
[[4 1 3]
 [7 2 0]
 [5 8 6]]
down
[[4 1 3]
 [7 2 6]
 [5 8 0]]
left
[[4 1 3]
 [7 2 6]
 [5 0 8]]
left
[[4 1 3]
 [7 2 6]
 [0 5 8]]
up
[[4 1 3]
 [0 2 6]
 [7 5 8]]
up
[[0 1 3]
 [4 2 6]
 [7 5 8]]
right
[[1 0 3]
 [4 2 6]
 [7 5 8]]
down
[[1 2 3]
 [4 0 6]
 [7 5 8]]
down
[[1 2 3]
 [4 5 6]
 [7 0 8]]
right
[[1 2 3]
 [4 5 6]
 [7 8 0]]
```

BFS search

```
Solution found in 10 moves
Generated 249 nodes
[[4 1 0]
 [7 2 3]
 [5 8 6]]
down
[[4 1 3]
 [7 2 0]
 [5 8 6]]
down
[[4 1 3]
 [7 2 6]
 [5 8 0]]
left
[[4 1 3]
 [7 2 6]
 [5 0 8]]
left
[[4 1 3]
 [7 2 6]
 [0 5 8]]
up
[[4 1 3]
 [0 2 6]
 [7 5 8]]
up
[[0 1 3]
 [4 2 6]
 [7 5 8]]
right
[[1 0 3]
 [4 2 6]
 [7 5 8]]
down
[[1 2 3]
 [4 0 6]
 [7 5 8]]
down
[[1 2 3]
 [4 5 6]
 [7 0 8]]
right
[[1 2 3]
 [4 5 6]
 [7 8 0]]
```

A* search

Modify Source Code (A* search):

Source code Part 1: ในส่วนของ Node definitions เราได้เพิ่มค่า heuristic เข้ามา ส่วนที่เหลือคงไว้แบบเดิม

```
3   ##################### Node definitions ############################
4   class Node:
5       def __init__(self, state, action, parent, cost, heuristic):
6           self.s = state
7           self.a = action
8           self.p = parent
9           self.c = cost
10          self.h = heuristic
11          self.expand = 0
12
13      def printstate(self):
14          print(self.s)
15
16      def printaction(self):
17          print(self.a)
```

Source code Part 2: ในส่วนของ Action Functions ยังคงไว้เหมือนเดิม แต่มีการเพิ่ม Code ดังรูปด้านล่าง

```
62  # Define the Manhattan Distance Heuristic (h2)
63  def manhattan_distance(state, goal):
64      distance = 0
65      for i in range(3):
66          for j in range(3):
67              if state[i, j] != 0:
68                  goal_position = np.where(goal == state[i, j])
69                  distance += abs(i - goal_position[0][0]) + abs(j - goal_position[1][0])
70      return distance
```

ในที่นี้เราได้ใช้ A* search โดยใช้การวิเคราะห์ heuristic h2 (Manhattan distance) จึงมีการสร้างฟังก์ชัน ‘manhattan_distance’ ซึ่งใช้สำหรับคำนวณค่า Manhattan Distance Heuristic ระหว่างสถานะปัจจุบันของ Sliding Puzzle และสถานะเป้าหมาย (goal state) ของ Sliding Puzzle

จาก Code เริ่มต้นด้วยการกำหนดค่า Manhattan Distance เท่ากับ 0 เนื่องจากยังไม่มีค่าระยะทางใด ๆ ที่คำนวณได้ หลังจากนั้นก็วนลูปเพื่อเข้าถึงแต่ละตำแหน่งใน Sliding Puzzle โดยใช้ตัวแปร ‘i’ และ ‘j’ แทนแถวและคอลัมน์ตามลำดับ แล้วตรวจสอบว่าในตำแหน่ง ‘(i, j)’ ไม่ใช่ตำแหน่งว่าง (0) และเข้าสู่บล็อกคำสั่งใน if เพื่อคำนวณค่า Manhattan Distance โดยใช้ค่าตำแหน่งปัจจุบัน ‘(i, j)’ และค่าตำแหน่งในสถานะเป้าหมาย ‘goal_position’ โดยคำนวณตามสูตร Manhattan Distance นั่นคือ ค่าสัมบูรณ์ของความต่างของแถวและคอลัมน์ระหว่างทั้งสถานะปัจจุบันและสถานะเป้าหมาย และเพิ่มค่านี้เข้าไปใน ‘distance’

Source code Part 3: ในส่วนของ Main มีการ Modify Code บางส่วน ดังรูปด้านล่าง

```
while found is None:
    # Search for a node to expand
    min_cost_index = np.argmin(costlist)
    costlist[min_cost_index] = maxdepth  # Eliminate found node from the list
    parent = nodelist[min_cost_index]

    # Expand
    parent.expand = 1  # Mark expanded
    depth = parent.c + 1
    up = Node(goup(parent.s), 'up', parent, depth, manhattan_distance(goup(parent.s), goal))
    down = Node(godown(parent.s), 'down', parent, depth, manhattan_distance(godown(parent.s), goal))
    left = Node(goleft(parent.s), 'left', parent, depth, manhattan_distance(goleft(parent.s), goal))
    right = Node(goright(parent.s), 'right', parent, depth, manhattan_distance(goright(parent.s), goal))
    nodelist.extend([up, down, left, right])
    costlist = np.append(costlist, [up.c + up.h, down.c + down.h, left.c + left.h, right.c + right.h])

    # Check if a solution is found
    if np.array_equal(up.s, goal):
        found = up
    elif np.array_equal(down.s, goal):
        found = down
    elif np.array_equal(left.s, goal):
        found = left
    elif np.array_equal(right.s, goal):
        found = right

    nodecount += 4
```

ใน Code ส่วนนี้มีการปรับแก้ส่วนต่าง ๆ ดังนี้

1. ‘min_cost_index’ ได้รับค่าจากการใช้ np.argmin(costlist) เพื่อเลือก Node ที่มีค่า F-score ต่ำที่สุดในรายการ costlist ในการนำมาขยายตัวเอง ซึ่งเป็นการเลือก Node ที่มีค่าที่ดีที่สุดในทุก ๆ รอบของ A* search.
2. กำหนดค่าของ Node ที่ถูกเลือกให้มีค่า F-score เท่ากับ ‘maxdepth’ ในการลบ Node นี้ออกจากรายการ ‘costlist’ เพื่อไม่ให้ถูกเลือกขยายตัวเองในรอบต่อ ๆ ไป และกำหนด Node ‘parent’ ให้เป็น Node ที่ถูกเลือกขยายตัวเอง ซึ่งจะถูกใช้ในการสร้าง Node ใหม่ที่เป็น Node ลูก
3. สร้าง Node ลูก 4 Node (up, down, left, right) โดยใช้ฟังก์ชัน goup, godown, goleft, และ goright เพื่อคำนวณสถานะของ Node ลูกแต่ละตัว และระบุ Node ที่เป็น Node ลูก, ระบุ parent, ความลึก depth ของ Node, และค่า H-score ซึ่งคำนวณโดยใช้ manhattan_distance ระหว่างสถานะปัจจุบันกับสถานะเป้าหมาย
4. เพิ่ม Node ลูกทั้ง 4 Node (up, down, left, right) เข้าในรายการ nodelist เพื่อให้สามารถขยายตัวเองในรอบถัดไป และเพิ่มค่า F-score ของ Node ลูกทั้ง 4 Node (up, down, left, right) เข้าในรายการ costlist เพื่อให้ A* search ใช้ค่า F-score ในการเลือก Node ที่จะขยายตัวเอง
5. ทำการตรวจสอบว่า Node ลูก ที่ขยายตัวเองได้ถูกตั้งค่าให้เป็น Node เป้าหมายหรือไม่ ถ้าเป็น Node เป้าหมายจะทำการตั้งค่า found เพื่อหยุดการค้นหา แล้วเพิ่มค่า ‘nodecount’ ขึ้นทีละ 4 เนื่องจากในแต่ละรอบได้ขยาย 4 Node ที่เป็นไปได้ คือ up, down, left, right

3. Test the performance of A* by comparing it with the BFS. Make initial positions that require solutions with different number of moves. Record the time used and the number of nodes generated for both BFS and A* like in this table.

Initial position	Depth	BFS	A*
1 2 3 4 5 6 7 0 8	1	0s (5 nodes)	0s (5 nodes)
1 2 3 4 0 6 7 5 8	2	0s (13 nodes)	0s (9 nodes)
1 2 3 4 6 0 7 5 8	3	0s (61 nodes)	0s (13 nodes)
1 2 3 4 6 8 7 5 0	4	0s (125 nodes)	0s (17 nodes)
1 2 3 4 6 8 7 0 5	5	0s (1,149 nodes)	0s (21 nodes)
1 2 3 4 6 8 0 7 5	6	0s (5,245 nodes)	0s (25 nodes)
1 2 3 0 6 8 4 7 5	7	0s (13,437 nodes)	0s (29 nodes)
0 2 3 1 6 8 4 7 5	8	1s (46,205 nodes)	0s (33 nodes)
2 0 3 1 6 8 4 7 5	9	15s (242,813 nodes)	0s (37 nodes)

ฉันท์สินี เมืองหนู 63070501221

Initial position	Depth	BFS	A*
1 2 3 4 0 6 7 5 8	10	54s (504,957 nodes)	0s (49 nodes)
2 6 3 1 8 0 4 7 5	11	More than 15 minutes	0s (185 nodes)
2 6 0 1 8 3 4 7 5	12	More than 15 minutes	0s (1785 nodes)
2 0 6 1 8 3 4 7 5	13	More than 15 minutes	2s (24,785 nodes)
2 8 6 1 0 3 4 7 5	14	More than 15 minutes	3s (29,689 nodes)
2 8 6 1 7 3 4 0 5	15	More than 15 minutes	4s (42,833 nodes)