Tatyana Graesser and Helen Xu
g1625 and hjx201

CS 4613 Coding Project 1

Compilation instructions:
- put input files and code file in same folder (it doesn't matter but otherwise you'll have to specify the whole path when prompted)
- Run code (just needs IDLE or any Python compiler)
- enter name of input+output file when prompted (code will create new file if it doesn't already exist)
- Code can be terminated by entering 'EXIT' when prompted

Results of Input1.txt

```
8 0 6 4
3 10 7 9
11 5 2 1

8 10 6 4
0 3 2 9
11 7 5 1

6
21
D R D L U L
5 5 6 6 6 6 6
```

Results of Input2.txt

```
8 0 6 4
3 10 7 9
11 5 2 1

8 7 2 4
10 6 9 1
3 11 5 0

12
97
R D D L L U R U R D R D
9 10 11 12 12 12 12 12 12 12 12 12 12
```

Results of Input3.txt

```
8 7 2 4
10 6 9 1
0 11 5 3

10 6 8 4
9 0 7 2
11 5 3 1

16
770
R U R U L L D R D R R U L U L D
10 10 11 12 13 14 15 15 15 16 16 16 16 16 16 16 16
```

```
"""
CS 4613 - AI Project 1
Authors: Tatyana Graesser (tg1625), Helen Xu (hjx201)

"""

import heapq
from copy import deepcopy

width = 4
height = 3

class Node:
    def __init__(self, state, goal, depth, solnpath, flist):
        self.state = state                                  #2-d matrix of puzzle position
        self.hn = sumManhattans(self.state, goal)           #stored to check if this is goal node
        self.depth = depth                                  #keeps track of depth in tree
        self.fn = self.hn + self.depth                      #keeps track of total f(n)
        self.solnpath = solnpath                            #keeps track of current solution path
        self.flist = flist + str(self.fn) + " "             #keeps track of list of fvalues of nodes in solution path

    def __str__(self):
        return "(" + (str)(self.state) + ", f(n) = " + (str)(self.fn) + ", solution path = " + self.solnpath + ", f-list = " + self.flist + ")" #print fxn for debugging purposes

    def __repr__(self):
        return (str)(self)

    def __lt__(self, other): #allows for comparison of two nodes to use in the min heap
        return self.fn < other.fn

def sumManhattans(state1, state2): #get manhattan distance for two states
    dists = 0
    for y in range(height):
        for x in range(width):
            if state1[y][x] != state2[y][x] and state1[y][x] != 0:
                dists += 1
    return dists

def findzero(state): #find coordinates of the blank tile
    for y in range(height):
        for x in range(width):
            if(state[y][x] == 0):
                return y, x

def expand(nodeo, goal, expanded, expandable): #create 4 children of the node and push them into the expandable list
    if(nodeo.hn == 0): # check if goal has been reached
        return True, nodeo

    zeroy, zerox = findzero(nodeo.state) #find location of zero

    #Check moving left
    if(zerox > 0):
        lstate = deepcopy(nodeo.state)
        lstate[zeroy][zerox], lstate[zeroy][zerox-1] = lstate[zeroy][zerox-1], lstate[zeroy][zerox]
        #if not already in list with lower f(n) value
        if lstate not in expanded: #create node and push into heap
                lnode = Node(lstate, goal, nodeo.depth + 1, nodeo.solnpath + "L ", nodeo.flist)
                heapq.heappush(expandable,lnode)
    #Same procedures for the other actions (right, up, and down)
```

```
60      #check moving right
61      if (zerox < width - 1):
62          rstate = deepcopy(nodeo.state)
63          rstate[zeroy][zerox], rstate[zeroy][zerox+1] = rstate[zeroy][zerox+1], rstate[
            zeroy][zerox]
64          if rstate not in expanded:
65                  rnode = Node(rstate, goal, nodeo.depth + 1, nodeo.solnpath + "R ", nodeo
                    .flist)
66                  heapq.heappush(expandable,rnode)
67      #check moving up
68      if (zeroy > 0):
69          ustate = deepcopy(nodeo.state)
70          ustate[zeroy][zerox], ustate[zeroy - 1][zerox] = ustate[zeroy - 1][zerox],
            ustate[zeroy][zerox]
71          if ustate not in expanded:
72                  unode = Node(ustate, goal, nodeo.depth + 1, nodeo.solnpath + "U ", nodeo
                    .flist)
73                  heapq.heappush(expandable,unode)
74      #check moving down
75      if (zeroy < height - 1):
76          dstate = deepcopy(nodeo.state)
77          dstate[zeroy][zerox], dstate[zeroy + 1][zerox] = dstate[zeroy + 1][zerox],
            dstate[zeroy][zerox]
78          if dstate not in expanded:
79                  dnode = Node(dstate, goal, nodeo.depth + 1, nodeo.solnpath + "D ", nodeo
                    .flist)
80                  heapq.heappush(expandable, dnode)
81
82      return False, None
83
84
85  def createStates(filepath): #create 2D matrices for initial state and goal state from
    file
86      initialS = [[]] #initial state matrix
87      goalS = [[]] #goal state matrix
88      with open(filepath, 'r') as fp:
89          char = '' #keeping track of number
90          row = 0 #keeping track of matrix row
91          addingM = initialS #first adding to initial state matrix
92          for c in fp.read():
93              if c.isnumeric():
94                  char += c
95              elif c == ' ':
96                  addingM[row].append(int(char))
97                  char = ''
98              elif c == '\n':
99                  if (char != ''): #add any found numbers
100                     addingM[row].append(int(char))
101                     char = ''
102                 if row == height: #switch to adding to goal state matrix
103                     addingM = goalS
104                     row = 0
105                 else:
106                     row += 1 #move to a new row
107                     addingM.append([ ])
108         if char != '':
109             addingM[row].append(int(char)) #getting last character we may have missed
110     #cleaning up any extra rows, probably a better way of making sure we just don't
        have to do this
111     initialS = initialS[:height]
112     goalS = goalS[:height]
113     print("Reading states from", filepath)
114     return initialS, goalS
115
116
117 def printOutput(initialS, goalS, goalNode, numNodes): #print final output to file
118         printed = False
```

```
119         while not printed:
120             filepath = input("Enter output file name: ")
121             try:
122                 with open(filepath, "w") as fp:
123                     #printing out initial state
124                     for row in initialS:
125                         for char in row:
126                             fp.write(str(char) + " ")
127                         fp.write("\n")
128                     fp.write("\n")
129                     #printing out goal state
130                     for row in goalS:
131                         for char in row:
132                             fp.write(str(char) +  " ")
133                         fp.write("\n")
134                     fp.write("\n" + str(goalNode.depth) + "\n")      #depth
135                     fp.write(str(numNodes) + "\n")                   #N
136                     fp.write(goalNode.solnpath + "\n")               #Solution Path
137                     fp.write(goalNode.flist)                         #list of f(n) values
138             except:
139                 print("An error occurred, please try again")
140             else:
141                 print("Output printed to", filepath)
142                 printed = True
143 
144 
145 def solve(initialS, goalS): #actually running A*
146     goalReached = False #are we at the goal
147     goalNode = None #will hold the goal node if we find one
148 
149     expandable = [] #min-heap used to keep track of expandable nodes
150     expanded = [] #list of nodes that have already been expanded, used for graph-search
151 
152     root = Node(initialS, goalS, 0, "", "") #initialize root node
153 
154     heapq.heappush(expandable,root) #add root to the min-heap
155 
156     while expandable and not goalReached: #while there are expandable nodes (heap isn't
        empty) and we haven't gotten a goal:
157         currnode = heapq.heappop(expandable) #expand node with least cost (uses min-heap)
158         goalReached, goalNode = expand(currnode, goalS, expanded, expandable) #expand
            the current node. Will return Boolean for if goal was found, as well as the
            goal node
159         expanded.append(currnode.state) #add current node to list of expanded nodes
160 
161     printOutput(initialS, goalS, goalNode, len(expanded) + len(expandable)) #print
        output to file
162 
163 def main():
164     filepath = ""
165     while True:
166         filepath = input("Enter filepath for input file. Enter EXIT to end code: ")
167         if filepath == "EXIT":
168             print("Goodbye :)")
169             break
170         try:
171             initialS, goalS = createStates(filepath) #read in initial and goal states
                from starting node
172         except:
173             print("Incorrect filepath")
174         else:
175             solve(initialS, goalS) #run fun algorithm
176 
177 
178 if __name__ == "__main__":
179     main()
180 
```