Total time: 0.014s
File: /Users/rishabhjain/Documents/Masters/SEM 2/Aritificial Intelligence/Program/Program 1/program_1.py
Function: expand at line 48

Line #	Hits	Time	Per Hit	% Time	Line Contents
48					@cpu
49					def expand(board):
50	412	0.1ms	.	0.8%	for i in range(len(board.data)):
51	1013	0.3ms	.	2.0%	for j in range(len(board.data[i])):
52	807	0.2ms	.	1.6%	if board.data[i][j] == '*':
53	103	.	.	0.2%	location = [i,j];
54	103	.	.	0.2%	break
55					
56	103	.	.	0.1%	actions = []
57	381	1.4ms	.	10.4%	for move in possible_actions(constants.board, location):
58	278	11.5ms	.	84.6%	actions.append([result(location, move, board.data) , move])
59					
60	103	.	.	0.1%	return actions

Total time: 0.001s
File: /Users/rishabhjain/Documents/Masters/SEM 2/Aritificial Intelligence/Program/Program 1/program_1.py
Function: possible_actions at line 62

Line #	Hits	Time	Per Hit	% Time	Line Contents
62					@cpu
63					def possible_actions(board, location):
64	103	.	.	4.5%	actions = ["RIGHT","LEFT","UP","DOWN"]
65	103	.	.	3.7%	actionstopeform = []
66					
67	515	0.1ms	.	19.1%	for x in actions:
68					# for moving right
69	412	0.1ms	.	12.2%	if x == "RIGHT":
70	103	.	.	5.7%	if location[1]+1 < len(board):
71	80	.	.	4.8%	actionstopeform.append([x,location[0],location[1]+1])
72					# for moving left
73	309	0.1ms	.	11.2%	elif x == "LEFT":
74	103	.	.	4.2%	if location[1]-1 >= 0:
75	63	.	.	4.0%	actionstopeform.append([x,location[0],location[1]-1])
76					# for moving up
77	206	0.1ms	.	8.2%	elif x == "UP":
78	103	.	.	3.9%	if location[0]-1 >= 0:
79	65	.	.	3.4%	actionstopeform.append([x,location[0]-1,location[1]])
80					# for moving down
81	103	.	.	1.9%	elif x == "DOWN":
82	103	.	.	5.7%	if location[0]+1 < len(board):
83	70	.	.	4.2%	actionstopeform.append([x,location[0]+1,location[1]])
84					
85	103	.	.	3.3%	return actionstopeform

Total time: 0.011s
File: /Users/rishabhjain/Documents/Masters/SEM 2/Aritificial Intelligence/Program/Program 1/program_1.py
Function: result at line 87

Line #	Hits	Time	Per Hit	% Time	Line Contents
87					@cpu
88					def result(location,action,board):
89					# copy of a board so that we can modify it
90	278	9.0ms	.	82.2%	newBoard = copy.deepcopy(board)
91	278	0.6ms	.	5.9%	temp = copy.deepcopy(newBoard[action[1]][action[2]])
92	278	0.6ms	.	5.9%	newBoard[action[1]][action[2]] = copy.deepcopy('*')
93	278	0.6ms	.	5.5%	newBoard[location[0]][location[1]] = copy.deepcopy(temp)
94					# return new board after moving * - NIL to the new location
95	278	0.1ms	.	0.5%	return newBoard

Total time: 0.004s
File: /Users/rishabhjain/Documents/Masters/SEM 2/Aritificial Intelligence/Program/Program 1/program_1.py
Function: manhattan at line 205

Line #	Hits	Time	Per Hit	% Time	Line Contents
205					@cpu
206					def manhattan(state):
207	172	.	.	1.1%	state = state.data
208	172	0.1ms	.	1.3%	goal_state = constants.goalBoard
209	172	.	.	1.0%	distance = 0
210					
211					# Create a dictionary that maps each value to its position in the goal state
212	172	.	.	1.0%	goal_dict = {}
213	688	0.2ms	.	5.2%	for i in range(len(goal_state)):
214	2064	0.6ms	.	14.5%	for j in range(len(goal_state[0])):
215	1548	0.4ms	.	11.3%	if goal_state[i][j] != '*':
216	1376	0.4ms	.	11.0%	goal_dict[goal_state[i][j]] = (i, j)
217					
218					# Calculate Manhattan distance
219	688	0.2ms	.	4.6%	for i in range(len(state)):
220	2064	0.6ms	.	14.9%	for j in range(len(state[0])):
221	1548	0.6ms	.	14.9%	if state[i][j] != '*' and state[i][j] != goal_state[i][j]:
222	764	0.2ms	.	5.6%	value = state[i][j]
223	764	0.2ms	.	5.1%	row, col = goal_dict[value]
224	764	0.3ms	.	7.7%	distance += abs(row - i) + abs(col - j)
225					

```
   226       172            .            .      0.9%        return distance
```

```
Total time: 0.023s
File: /Users/rishabhjain/Documents/Masters/SEM 2/Aritificial Intelligence/Program/Program 1/program_1.py
Function: a_star at line 235

Line #      Hits         Time      Per Hit    % Time    Line Contents
==============================================================
   235                                                  @cpu
   236                                                  def a_star(initialProblem, f):
   237         1            .            .         .        initialNode = Node(data = initialProblem)   # node←NODE(STATE=problem.INITIAL)
   238         1            .            .         .        frontier = PriorityQueue()
   239         1        0.1ms        0.1ms      0.2%        frontier.append((f(initialNode), initialNode))         # frontier←a priority queue
   240
   241         1            .            .         .        reached = {str(initialProblem): initialNode}           # reached←a lookup table, w
   242
   243       104        0.1ms            .      0.2%        while not frontier.empty():                            # while not IS-EMPTY(fronti
   244       104        0.1ms            .      0.3%            node = frontier.get()                              # node←POP(frontier)
   245
   246       104        0.1ms            .      0.2%            if constants.goalBoard == node[1].data:            # if problem.IS-GOAL(node.S
   247         1            .            .      0.1%                print('Max queue size:', frontier.getSize())
   248         1            .            .         .                return node[1]                                 # then return node
   249
   250       381       14.5ms            .     62.2%            for child in expand(node[1]):               # for each child in EXPAND(problem
   251                                                              # s←child.STATE
   252       278        0.4ms            .      1.5%                s =  Node( data = child[0], depth = node[1].depth + 1, move = child[1], pr
   253
   254                                                              # if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
   255       278        0.5ms            .      2.2%                if str(s.data) not in reached or s.depth < reached[str(s.data)].depth:
   256       171        0.2ms            .      1.0%                    reached[str(s.data)] = s                    # reached[s]←child
   257       171        7.5ms            .     32.0%                    frontier.append((f(s) ,s))                  # add child to frontier
   258
   259                                                  return constants.failure                              # return failure
```

```
Total time: 0.000s
File: /Users/rishabhjain/Documents/Masters/SEM 2/Aritificial Intelligence/Program/Program 1/program_1.py
Function: printStatistics at line 261

Line #      Hits         Time      Per Hit    % Time    Line Contents
==============================================================
   261                                                  @cpu
   262                                                  def printStatistics(solution):
   263         1            .            .         .        pathCost = 0
   264         1            .            .         .        stateSequence = []
   265         1            .            .         .        actionSequence = []
   266
   267        38            .            .      4.9%        while solution.prev != None:
   268        37            .            .      7.6%            stateSequence.insert(0, solution.data)
   269        37            .            .      6.0%            actionSequence.insert(0, solution.move)
   270        37            .            .      3.8%            solution = solution.prev
   271        37            .            .      6.0%            pathCost += 1
   272
   273         1            .            .      1.6%        print('Action sequence:')
   274         1        0.1ms        0.1ms     30.4%        print(*actionSequence, sep='\n')
   275
   276         1            .            .      1.1%        print('\nState sequence:')
   277         1        0.1ms        0.1ms     37.0%        print(*stateSequence, sep='\n')
   278
   279         1            .            .      1.6%        print('\nPath cost:', pathCost)
```