

Name - Vasu Kalariya

Roll - PE29

Sub - AI

Lab Assignment - 1

Dim: Solve 8 puzzle problem using A^* algorithm

Objective: To study and implement A^* algorithm for 8 puzzle problem

Theory:

1 → Best-first search methods.

Best first search is a traversal technique that decides which node is to be visited next by checking which node is the most promising one then check it. For this it uses an evaluation function to decide the traversal. Best first technique of tree traversal comes under the category of heuristic search or informed search technique. The cost of nodes is stored in a priority queue. This makes implementation of best-first search is same as that of breadth first search. We will use the priority queue just like we use a queue for BFS.

→ OR graphs

The AND-OR graph is useful for representing the solution of problem that can solved by decomposing them into a set of smaller problems, all of which must ~~be~~ then be solved. This decomposition, ~~then~~ or reduction, generates arcs that we call AND-^{OR}~~OR~~ arcs. One AND arcs may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution. Just as in an OR graph, several arcs may emerge

from a single node, indicating a variety of ways in which the original problem might be solved. This is why the structure is called not simply an AND-graph but rather an AND-OR graph.

2 8-Puzzle Problem

The 8-puzzle problem is a puzzle invented and popularized by naves palmer chapman in the 1820's. It is played on a 3-by-3 grid with 8 square block labeled 1 to 8 and a blank square. Your goal is to rearrange the block so that they are in order. you are permitted to slide blocks horizontally or vertically into the blank space.

→ Data structure and other details about A^* algorithm

A^* search is most commonly known form of best first search. It uses heuristic function $h(n)$ and cost $g(n)$. A^* search algorithm finds the shortest path through the search space using heuristic function this search algorithm expands less search tree and provides optimal result faster. A^* algorithm is similar to UCS except that uses $g(n) + h(n)$ instead of $g(n)$.

In A^* search algorithm we use search heuristic as well as the cost to reach the node. Hence we can combine both cost as following and this sum is called as fitness number.

$$f(n) = g(n) + h(n)$$

Input : initial state

Output : solution state with optimal path

Algorithm : A^*

programming language : C, C++, Python, etc

FADS

1. What is heuristic function? What is the advantage of using heuristic function?

Heuristic function: Heuristic is a function which is used in informed search and it finds the most promising path. It takes the current state of the agent and its input and output produces the estimation of how close an agent is from the goal.

The heuristic method might not always give the best solutⁿ. but it guaranteed to find a good solutⁿ in reasonable time. Heuristic functⁿ estimates ~~heuristic~~^{how} close a state is to the goal.

Admissibility of the heuristic function is given as.

$$h(n) \leq h^*(n)$$

Advantages using heuristic function

- It can provide some quick and relatively inexpensive feedback to designers.
- you can obtain feedback early in the design process.
- Assigning the correct heuristic can help suggest the best connective measure to design.

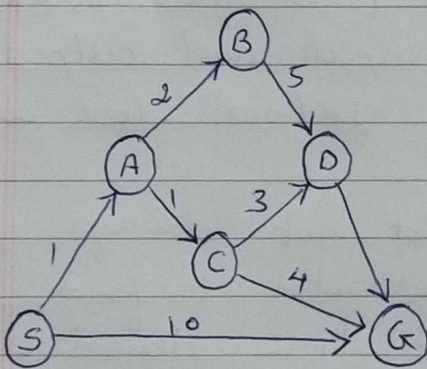
Q 2 Explain A^* algorithm with example

A^* algorithm is a searching algorithm that searches for the shortest path between the initial and the final state. It is used in various applications such as maps.

Example: In this example, we will traverse the given graph using the A^* algorithm. The heuristic value of all states is given in the below table so we will calculate the $f(n)$ of each state using the formula

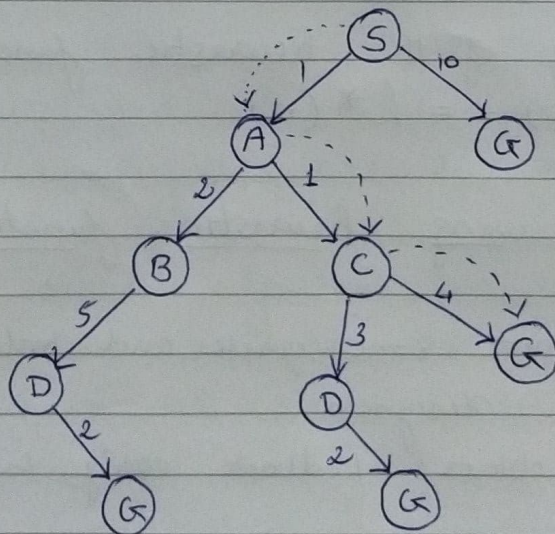
$f(n) = g(n) + h(n)$, where $g(n)$ is the cost to reach any node from start state.

Here we will use OPEN and CLOSE list.



State	$h(n)$
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



initialization : $\{(S, 5)\}$

iteration 1 : $\{(S \rightarrow A, 4), (S \rightarrow G, 10)\}$

iteration 2 : $\{(S \rightarrow A \rightarrow C, 4), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

iteration 3 : $\{(S \rightarrow A \rightarrow C \rightarrow G, 6), (S \rightarrow A \rightarrow C \rightarrow D, 11),$
 $(S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

iteration 4 : will give the final result as
 $S \rightarrow A \rightarrow C \rightarrow G$ It provides

optimal path with cost 4.

3 Explain different heuristic function that can be used for the eight puzzle problem.

The function for 8 puzzle problem can be given as $f(n) = g(n) + h(n)$

where $h(n)$ is the heuristic function which can be given as

→ $h(n)$ = The number of misplaced tiles

→ $h(n)$ = The sum of the distance of the tiles from their goal positions. (Manhattan distance)

```

1 def print_puzzle(arr):
2     print(20*'-')
3     for i in range(len(arr)):
4         print(arr[i],end = " ")
5         if(i==2 or i==5 or i==8):
6             print('\n')
7
8 def misplaced_elements(curr,goal):
9     count = 0
10    for i in range(len(goal)):
11        if(goal[i]!=curr[i]):
12            count +=1
13    return count
14
15 def puzzle_sol(current,goal,function_value,Node_number):
16     print_puzzle(current)
17     if((current==goal)!=True):
18         Node_number += 1
19         index_of_empty = current.index('_')
20         #-----for Index 0-----
21         if(index_of_empty == 0):
22             arr1 = current.copy()
23             arr1[1] = current[0]
24             arr1[0] = current[1]
25             c1 = Node_number + misplaced_elements(arr1,goal)
26             if(c1<=function_value):
27                 function_value = c1
28                 current = arr1.copy()
29             arr3 = current.copy()
30             arr3[3] = current[0]
31             arr3[0] = current[3]
32             c3 = Node_number + misplaced_elements(arr3,goal)
33             if(c3<=function_value):
34                 function_value = c3
35                 current = arr3.copy()
36
37         puzzle_sol(current,goal,function_value,Node_number)
38         #-----for Index 1-----
39         elif(index_of_empty == 1):
40             arr2 = current.copy()
41             arr2[2] = current[1]
42             arr2[1] = current[2]
43             c2 = Node_number + misplaced_elements(arr2,goal)
44             if(c2<=function_value):
45                 function_value = c2
46                 current = arr2.copy()
47             arr4 = current.copy()
48             arr4[4] = current[1]
49             arr4[1] = current[4]
50             c4 = Node_number + misplaced_elements(arr4,goal)
51             if(c4<=function_value):
52                 function_value = c4
53                 current = arr4.copy()
54             arr0 = current.copy()
55             arr0[0] = current[1]
56             arr0[1] = current[0]
57             c0 = Node_number + misplaced_elements(arr0,goal)
58             if(c0<=function_value):
59                 function_value = c0
60                 current = arr0.copy()

```

```
61         puzzle_sol(current,goal,function_value,Node_number)
62     #-----for Index 2-----
63     elif(index_of_empty == 2):
64         arr5 = current.copy()
65         arr5[5] = current[2]
66         arr5[2] = current[5]
67         c5 = Node_number + misplaced_elements(arr5,goal)
68         if(c5<=function_value):
69             function_value = c5
70             current = arr5.copy()
71         arr1 = current.copy()
72         arr1[1] = current[2]
73         arr1[2] = current[1]
74         c1 = Node_number + misplaced_elements(arr1,goal)
75         if(c1<=function_value):
76             function_value = c1
77             current = arr1.copy()
78
79         puzzle_sol(current,goal,function_value,Node_number)
80     #-----for Index 3-----
81     elif(index_of_empty == 3):
82         arr4 = current.copy()
83         arr4[3] = current[4]
84         arr4[4] = current[3]
85         c4 = Node_number + misplaced_elements(arr4,goal)
86         if(c4<=function_value):
87             function_value = c4
88             current = arr4.copy()
89         arr6 = current.copy()
90         arr6[3] = current[6]
91         arr6[6] = current[3]
92         c6 = Node_number + misplaced_elements(arr6,goal)
93         if(c6<=function_value):
94             function_value = c6
95             current = arr6.copy()
96         arr0 = current.copy()
97         arr0[3] = current[0]
98         arr0[0] = current[3]
99         c0 = Node_number + misplaced_elements(arr0,goal)
100        if(c0<=function_value):
101            function_value = c0
102            current = arr0.copy()
103        puzzle_sol(current,goal,function_value,Node_number)
104    #-----for Index 4-----
105    elif(index_of_empty == 4):
106        arr5 = current.copy()
107        arr5[4] = current[5]
108        arr5[5] = current[4]
109        c5 = Node_number + misplaced_elements(arr5,goal)
110        if(c5<=function_value):
111            function_value = c5
112            current = arr5.copy()
113        arr7 = current.copy()
114        arr7[4] = current[7]
115        arr7[7] = current[4]
116        c7 = Node_number + misplaced_elements(arr7,goal)
117        if(c7<=function_value):
118            function_value = c7
119            current = arr7.copy()
120        arr3 = current.copy()
```

```

121     arr3[3] = current[4]
122     arr3[4] = current[3]
123     c3 = Node_number + misplaced_elements(arr3,goal)
124     if(c3<=function_value):
125         function_value = c3
126         current = arr3.copy()
127     arr1 = current.copy()
128     arr1[1] = current[4]
129     arr1[4] = current[1]
130     c1 = Node_number + misplaced_elements(arr1,goal)
131     if(c1<=function_value):
132         function_value = c1
133         current = arr1.copy()
134     puzzle_sol(current,goal,function_value,Node_number)
135     #-----for Index 5-----
136     elif(index_of_empty == 5):
137         arr8 = current.copy()
138         arr8[5] = current[8]
139         arr8[8] = current[5]
140         c8 = Node_number + misplaced_elements(arr8,goal)
141         if(c8<=function_value):
142             function_value = c8
143             current = arr8.copy()
144         arr4 = current.copy()
145         arr4[4] = current[5]
146         arr4[5] = current[4]
147         c4 = Node_number + misplaced_elements(arr4,goal)
148         if(c4<=function_value):
149             function_value = c4
150             current = arr4.copy()
151         arr2 = current.copy()
152         arr2[5] = current[2]
153         arr2[2] = current[5]
154         c2 = Node_number + misplaced_elements(arr2,goal)
155         if(c2<=function_value):
156             function_value = c2
157             current = arr2.copy()
158         puzzle_sol(current,goal,function_value,Node_number)
159     #-----for Index 6-----
160     elif(index_of_empty == 6):
161         arr7 = current.copy()
162         arr7[7] = current[6]
163         arr7[6] = current[7]
164         c7 = Node_number + misplaced_elements(arr7,goal)
165         if(c7<=function_value):
166             function_value = c7
167             current = arr7.copy()
168         arr3 = current.copy()
169         arr3[3] = current[6]
170         arr3[6] = current[3]
171         c3 = Node_number + misplaced_elements(arr3,goal)
172         if(c3<=function_value):
173             function_value = c3
174             current = arr3.copy()
175         puzzle_sol(current,goal,function_value,Node_number)
176     #-----for Index 7-----
177     elif(index_of_empty == 7):
178         arr8 = current.copy()
179         arr8[7] = current[8]
180         arr8[8] = current[7]

```



```
181         c8 = Node_number + misplaced_elements(arr8,goal)
182         if(c8<=function_value):
183             function_value = c8
184             current = arr8.copy()
185         arr4 = current.copy()
186         arr4[4] = current[7]
187         arr4[7] = current[4]
188         c4 = Node_number + misplaced_elements(arr4,goal)
189         if(c4<=function_value):
190             function_value = c4
191             current = arr4.copy()
192         arr6 = current.copy()
193         arr6[6] = current[7]
194         arr6[7] = current[6]
195         c6 = Node_number + misplaced_elements(arr6,goal)
196         if(c6<=function_value):
197             function_value = c6
198             current = arr6.copy()
199         puzzle_sol(current,goal,function_value,Node_number)
200     #-----for Index 8-----
201     elif(index_of_empty == 8):
202         arr5 = current.copy()
203         arr5[5] = current[8]
204         arr5[8] = current[5]
205         c5 = Node_number + misplaced_elements(arr5,goal)
206         if(c5<=function_value):
207             function_value = c5
208             current = arr5.copy()
209         arr7 = current.copy()
210         arr7[8] = current[7]
211         arr7[7] = current[8]
212         c7 = Node_number + misplaced_elements(arr7,goal)
213         if(c7<=function_value):
214             function_value = c7
215             current = arr7.copy()
216         puzzle_sol(current,goal,function_value,Node_number)
217
218
219
220 Node_number = 0
221 function_value = 0
222 goal = [1,2,3,4,5,6,7,8,'_']
223 current = [1,2,3,'_',4,6,7,5,8]
224 final_hx = misplaced_elements(current,goal)
225 function_value = Node_number + final_hx
226 puzzle_sol(current,goal,function_value,Node_number)
```

PROBLEMSOUTPUTDEBUG CONSOLETERMINAL

1: Python

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\kalar> & python "f:/T9/AI/Lab 1/PE29_VasuKalariya_Ass11.py"

1 2 3
_ 4 6
7 5 8

1 2 3
4 _ 6
7 5 8

1 2 3
4 5 6
7 _ 8

1 2 3
4 5 6
7 8 _
PS C:\Users\kalar>

Python 3.8.5 64-bit (conda) 0 0 0Ln 12, Col 22Spaces: 4UTF-8CRLFPython