Name - Vasu Kalariya
Roll - PE29
Sub - AI

Lab Assignment - 1

Aim: 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 ~~do~~ 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-~~arcs~~ 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 noyes 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 side blocks horizontally or vertically into the blank space

→ Data structure and other detailes 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

FAQs

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 agent is from the goal.
The heuristic method might not always give the bst solutn. but it guarrented to find a good solutn in reasonable time. Heuristic functn estimates how close a state is to the goal.

Admissibility of the heuristic function is given as.

$$h(n) <= h*(n)$$

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.

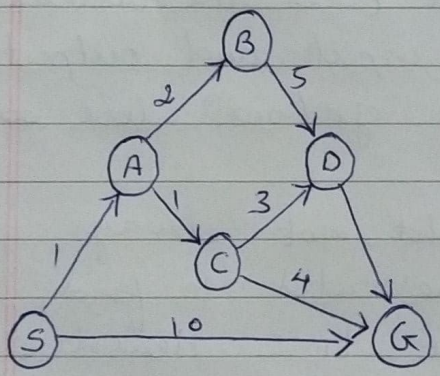
Q. 2 Explain A^* algorithm with example

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

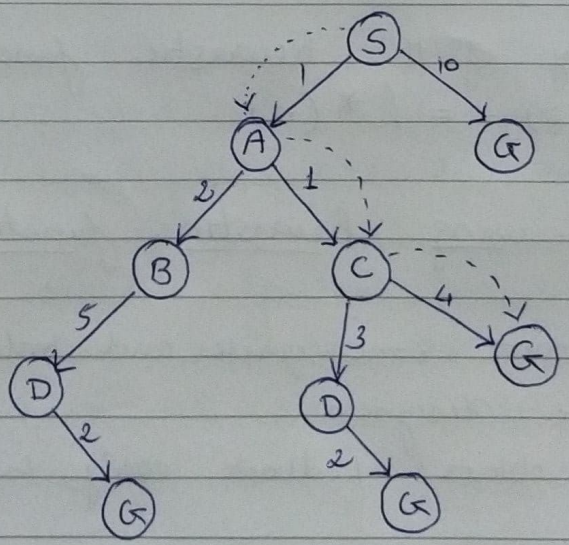
$f(n) = g(n) + h(n)$, where $g(n)$ is the cost to ~~each~~ reach any node from start state.

Here we will us 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 → A, 4), (S → G, 10)}

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

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

iteration 4 : will give the final result as
S → A → C → 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 given be given as

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

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

```python
def print_puzzle(arr):
    print(20*'-')
    for i in range(len(arr)):
        print(arr[i],end ="  ")
        if(i==2 or i==5 or i==8):
            print('')

def misplaced_elements(curr,goal):
    count = 0
    for i in range(len(goal)):
        if(goal[i]!=curr[i]):
            count +=1
    return count

def puzzle_sol(current,goal,function_value,Node_number):
    print_puzzle(current)
    if((current==goal)!=True):
        Node_number += 1
        index_of_empty = current.index('_')
    #---------------for Index 0---------------------
        if(index_of_empty == 0):
            arr1 = current.copy()
            arr1[1] = current[0]
            arr1[0] = current[1]
            c1 = Node_number + misplaced_elements(arr1,goal)
            if(c1<=function_value):
                function_value = c1
                current = arr1.copy()
            arr3 = current.copy()
            arr3[3] = current[0]
            arr3[0] = current[3]
            c3 = Node_number + misplaced_elements(arr3,goal)
            if(c3<=function_value):
                function_value = c3
                current = arr3.copy()

            puzzle_sol(current,goal,function_value,Node_number)
    #---------------for Index 1---------------------
        elif(index_of_empty == 1):
            arr2 = current.copy()
            arr2[2] = current[1]
            arr2[1] = current[2]
            c2 = Node_number + misplaced_elements(arr2,goal)
            if(c2<=function_value):
                function_value = c2
                current = arr2.copy()
            arr4 = current.copy()
            arr4[4] = current[1]
            arr4[1] = current[4]
            c4 = Node_number + misplaced_elements(arr4,goal)
            if(c4<=function_value):
                function_value = c4
                current = arr4.copy()
            arr0 = current.copy()
            arr0[0] = current[1]
            arr0[1] = current[0]
            c0 = Node_number + misplaced_elements(arr0,goal)
            if(c0<=function_value):
                function_value = c0
                current = arr0.copy()
```

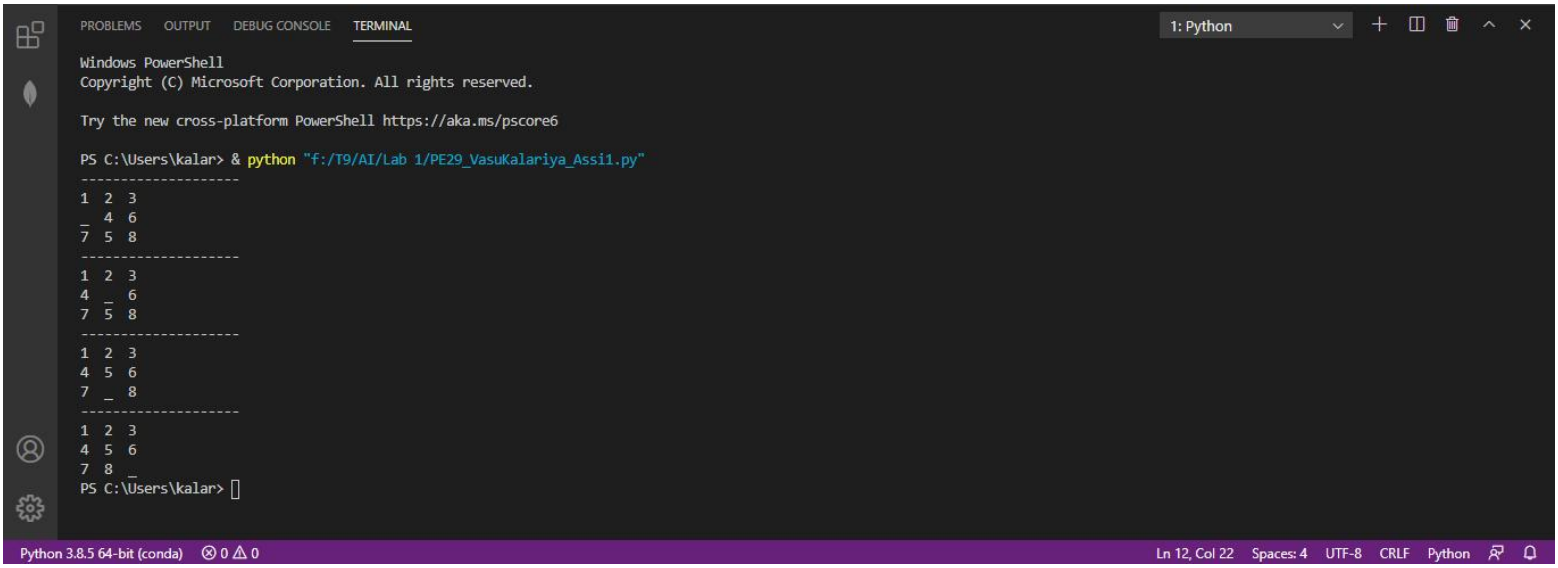
```
            puzzle_sol(current,goal,function_value,Node_number)
    #---------------for Index 2---------------------
        elif(index_of_empty == 2):
            arr5 = current.copy()
            arr5[5] = current[2]
            arr5[2] = current[5]
            c5 = Node_number + misplaced_elements(arr5,goal)
            if(c5<=function_value):
                function_value = c5
                current = arr5.copy()
            arr1 = current.copy()
            arr1[1] = current[2]
            arr1[2] = current[1]
            c1 = Node_number + misplaced_elements(arr1,goal)
            if(c1<=function_value):
                function_value = c1
                current = arr1.copy()

            puzzle_sol(current,goal,function_value,Node_number)
    #---------------for Index 3---------------------
        elif(index_of_empty == 3):
            arr4 = current.copy()
            arr4[3] = current[4]
            arr4[4] = current[3]
            c4 = Node_number + misplaced_elements(arr4,goal)
            if(c4<=function_value):
                function_value = c4
                current = arr4.copy()
            arr6 = current.copy()
            arr6[3] = current[6]
            arr6[6] = current[3]
            c6 = Node_number + misplaced_elements(arr6,goal)
            if(c6<=function_value):
                function_value = c6
                current = arr6.copy()
            arr0 = current.copy()
            arr0[3] = current[0]
            arr0[0] = current[3]
            c0 = Node_number + misplaced_elements(arr0,goal)
            if(c0<=function_value):
                function_value = c0
                current = arr0.copy()
            puzzle_sol(current,goal,function_value,Node_number)
    #---------------for Index 4---------------------
        elif(index_of_empty == 4):
            arr5 = current.copy()
            arr5[4] = current[5]
            arr5[5] = current[4]
            c5 = Node_number + misplaced_elements(arr5,goal)
            if(c5<=function_value):
                function_value = c5
                current = arr5.copy()
            arr7 = current.copy()
            arr7[4] = current[7]
            arr7[7] = current[4]
            c7 = Node_number + misplaced_elements(arr7,goal)
            if(c7<=function_value):
                function_value = c7
                current = arr7.copy()
            arr3 = current.copy()
```

```
            arr3[3] = current[4]
            arr3[4] = current[3]
            c3 = Node_number + misplaced_elements(arr3,goal)
            if(c3<=function_value):
                function_value = c3
                current = arr3.copy()
            arr1 = current.copy()
            arr1[1] = current[4]
            arr1[4] = current[1]
            c1 = Node_number + misplaced_elements(arr1,goal)
            if(c1<=function_value):
                function_value = c1
                current = arr1.copy()
            puzzle_sol(current,goal,function_value,Node_number)
    #---------------for Index 5---------------------
        elif(index_of_empty == 5):
            arr8 = current.copy()
            arr8[5] = current[8]
            arr8[8] = current[5]
            c8 = Node_number + misplaced_elements(arr8,goal)
            if(c8<=function_value):
                function_value = c8
                current = arr8.copy()
            arr4 = current.copy()
            arr4[4] = current[5]
            arr4[5] = current[4]
            c4 = Node_number + misplaced_elements(arr4,goal)
            if(c4<=function_value):
                function_value = c4
                current = arr4.copy()
            arr2 = current.copy()
            arr2[5] = current[2]
            arr2[2] = current[5]
            c2 = Node_number + misplaced_elements(arr2,goal)
            if(c2<=function_value):
                function_value = c2
                current = arr2.copy()
            puzzle_sol(current,goal,function_value,Node_number)
    #---------------for Index 6---------------------
        elif(index_of_empty == 6):
            arr7 = current.copy()
            arr7[7] = current[6]
            arr7[6] = current[7]
            c7 = Node_number + misplaced_elements(arr7,goal)
            if(c7<=function_value):
                function_value = c7
                current = arr7.copy()
            arr3 = current.copy()
            arr3[3] = current[6]
            arr3[6] = current[3]
            c3 = Node_number + misplaced_elements(arr3,goal)
            if(c3<=function_value):
                function_value = c3
                current = arr3.copy()
            puzzle_sol(current,goal,function_value,Node_number)
    #---------------for Index 7---------------------
        elif(index_of_empty == 7):
            arr8 = current.copy()
            arr8[7] = current[8]
            arr8[8] = current[7]
```

```
            c8 = Node_number + misplaced_elements(arr8,goal)
            if(c8<=function_value):
                function_value = c8
                current = arr8.copy()
            arr4 = current.copy()
            arr4[4] = current[7]
            arr4[7] = current[4]
            c4 = Node_number + misplaced_elements(arr4,goal)
            if(c4<=function_value):
                function_value = c4
                current = arr4.copy()
            arr6 = current.copy()
            arr6[6] = current[7]
            arr6[7] = current[6]
            c6 = Node_number + misplaced_elements(arr6,goal)
            if(c6<=function_value):
                function_value = c6
                current = arr6.copy()
            puzzle_sol(current,goal,function_value,Node_number)
    #---------------for Index 8---------------------
        elif(index_of_empty == 8):
            arr5 = current.copy()
            arr5[5] = current[8]
            arr5[8] = current[5]
            c5 = Node_number + misplaced_elements(arr5,goal)
            if(c5<=function_value):
                function_value = c5
                current = arr5.copy()
            arr7 = current.copy()
            arr7[8] = current[7]
            arr7[7] = current[8]
            c7 = Node_number + misplaced_elements(arr7,goal)
            if(c7<=function_value):
                function_value = c7
                current = arr7.copy()
            puzzle_sol(current,goal,function_value,Node_number)


Node_number = 0
function_value = 0
goal = [1,2,3,4,5,6,7,8,'_']
current = [1,2,3,'_',4,6,7,5,8]
final_hx = misplaced_elements(current,goal)
function_value = Node_number + final_hx
puzzle_sol(current,goal,function_value,Node_number)
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
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_Assi1.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)
Ln 12, Col 22
Spaces: 4
UTF-8
CRLF
Python