FACULTY OF COMPUTING & INFORMATION TECHNOLOGY
KING ABDULAZIZ UNIVERSITY

FCIT
KAU

كلية الحاسبات وتقنية المعلومات
جامعة الملك عبدالعزيز

CPCS 331 Artificial intelligence

Search Assignment

Implement an agent that solve 8 puzzle

Razan Muhammed Aljuhani

Contents

Problem description

The 8 puzzle is a simple game invented and popularized by Noyes Palmer Chapman in the 1870s. it consisting of a 3 x 3 grid containing 9 square labeled 1 to 8 and a blank square . The object is to rearrange the blocks horizontally or vertically into the blank square. to reach "goal state" .

Example:

Initial state: Goal state:

1	**2**	**3**
4	5	6
7	8	

	1	**3**
4	2	5
7	8	6

Problem formulation

States: The state specifies the location of each of the 9 tiles, one of tiles is blank, and the other 8 tiles have a number from 1 to 8.

1- Initial state: The location of each of the 8 tiles in one of the nine squares.
2- Actions: Move blank either left, right, up, down. As constraints no diagonally move.
3- Transition model: The excepted effect of the actions.
4- Goal test: The goal state is given, so check if state matches the goal configuration.
5- Path cost: 1 per move , so the total path cost = number of moves .

Search technique and the implementation process

A* is type of informed search algorithms , it use heuristics to guide the search .it aims to find a path to the given goal node having the smallest cost by avoid expanding paths that are already expensive . A* selects the paths by evaluation function f(n)=g(n)+h(n) where :

g(n) - (cost from initial node to n node)

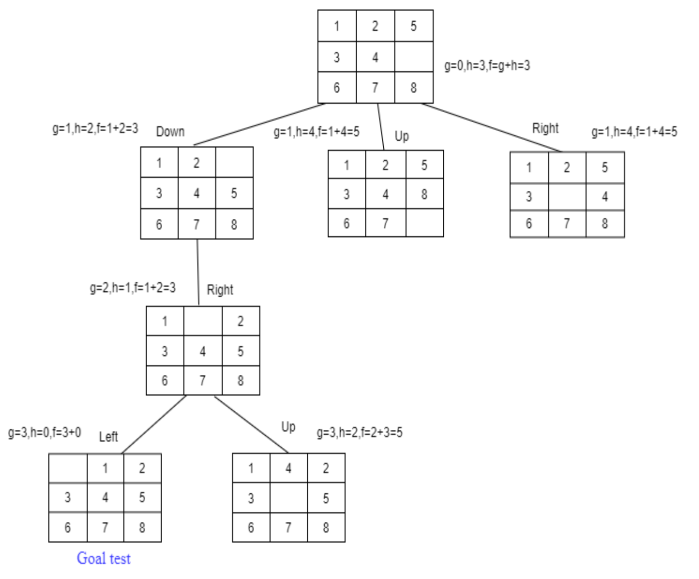
h(n) - (estimated cost from n node to goal node)

f(n) - (estimated total cost of path through n to goal)

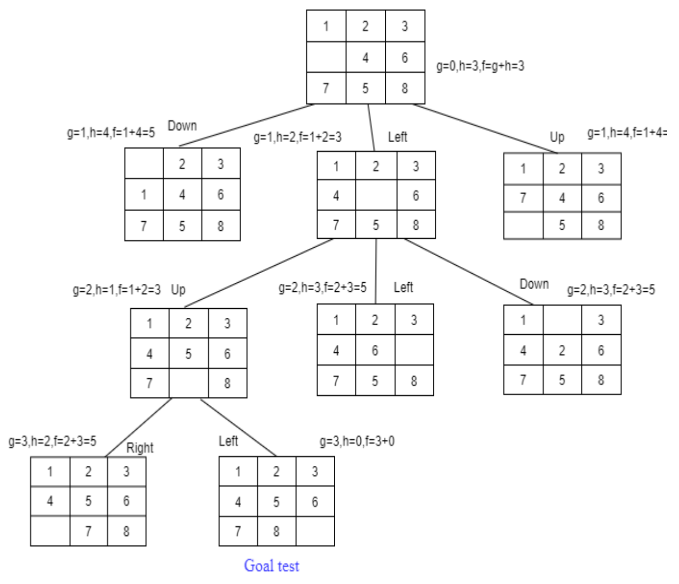
It works by maintaining a tree of paths originating at the start node and expanding those paths one edge at a time until access the goal. At each iteration of its main loop , A* needs to determine which of its paths to expand(the minimum path based on cost) to way to the goal. A* terminates when the path that selected to expand is a path from state to goal or if there are no paths is shorter than this path .it guaranteed to return a least-cost path from start to goal.

Examples for solve puzzle problem by A*:

Example 1



Example 2



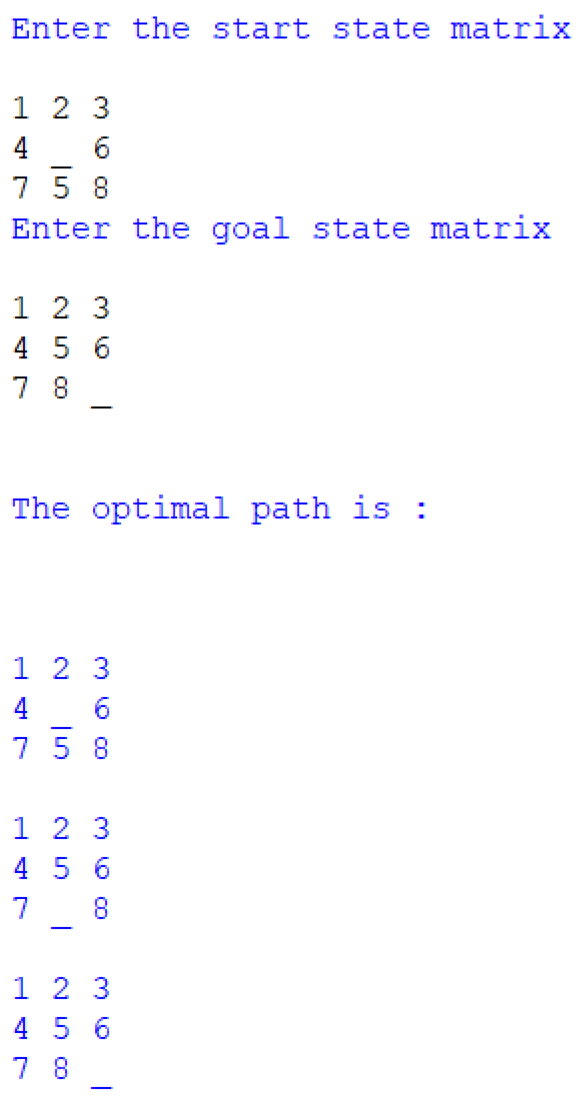
The implementation process of the problem, we will using python language to solve the problem and run it in by python IDEL app on Windows 10, The A* depends on repeated steps (after every expand we will compare cost to select the minimum), so we will use a lots of loop in the code and display the optimal solution (optimal path) and when find it there is no need for other paths (other paths help me for solve to right solution, but it is not the best solution).

Sample outputs

Example 1

```
Enter the start state matrix

1 2 5
3 4 _
6 7 8
Enter the goal state matrix

_ 1 2
3 4 5
6 7 8


The optimal path is :



1 2 5
3 4 _
6 7 8

1 2 _
3 4 5
6 7 8

1 _ 2
3 4 5
6 7 8

_ 1 2
3 4 5
6 7 8
```

Example 2



```
Enter the start state matrix

1 2 3
4 _ 6
7 5 8
Enter the goal state matrix

1 2 3
4 5 6
7 8 _


The optimal path is :



1 2 3
4 _ 6
7 5 8

1 2 3
4 5 6
7 _ 8

1 2 3
4 5 6
7 8 _
```

Method evaluation

As represented in this report, we used A* search technique for solve 8 puzzle problem , and we implement an agent for this problem by python language .

This search technique is **complete** because it always find a solution if exist unless there are infinitely many nodes with f < f(G) , Also it find the **optimal** solution if h(n) satisfies certain conditions (admissibility and consistency).

As efficiency, **Time complexity** of A* depends on h and can be exponential. And **space complexity** of A* that stores all nodes is O(b^m) where m is maximum depth .

Appendix

This section is contain the python's code used :

```
class Node:
    def __init__(self,data,level,fval):
        self.data = data
        self.level = level
        self.fval = fval
    def generate_child(self):
        x,y = self.find(self.data,'_')
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children
    def shuffle(self,puz,x1,y1,x2,y2):
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None
```

```
 def copy(self,root):
        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            temp.append(t)
        return temp
    def find(self,puz,x):
        for i in range(0,len(self.data)):
            for j in range(0,len(self.data)):
                if puz[i][j] == x:
                    return i,j
class Puzzle:
    def __init__(self,size):
        self.n = size
        self.open = []
        self.closed = []
    def accept(self):
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz
    def f(self,start,goal):
        return self.h(start.data,goal)+start.level
    def h(self,start,goal):
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp
```

```
def process(self):
        print("Enter the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()
        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        self.open.append(start)
        print("\n")
        print("The optimal path is :")
        print("\n")
        while True:
            cur = self.open[0]
            print("")
            for i in cur.data:
                for j in i:
                     print(j,end=" ")
                print("")
            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]
            self.open.sort(key = lambda x:x.fval,reverse=False)
puz = Puzzle(3)
puz.process()
```