DATA: 25/01/2022 EXP No: 05	Title of the Lab 5a) Implementation of Best First Search 5b) Implementation of A* Algorithm	Name:Avinash reddy Vasipalli Registration Number: RA1911027010007 Section: N1 Lab Batch: 1 Day Order: 2

AIM:

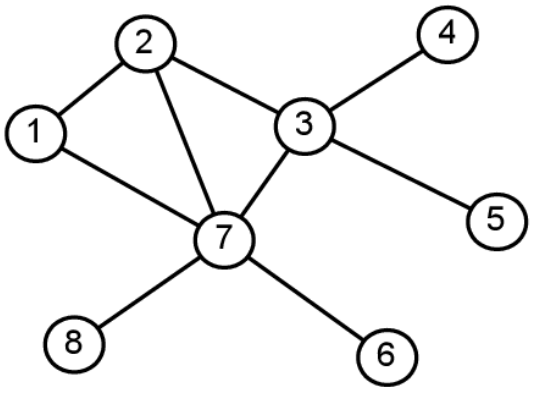
1) To implement Best First Search Algorithm to find the shortest path.
2) To implement A* algorithm to find best solution for 8 puzzle problem

Description of concept or problem given

1) Implementation of best first Search specifies that we have to find shortest path to a target node in provided graph
2) 8 puzzle problem consist of N tiles where N can be 8,16,24….and so on. In this puzzle the grid is divided into 3X3 matrix and contains 8 spaces filled with integers from 1 to 8 and one empty space. The start and goal state are also providing so that we know the end. Puzzle can be solved y moving integers tiles one by one in the empty spaces and confirming the goal.

Manual Solution:

1)



In this graph let the source node as 4 and target node to be 8. BFS uses the priority queue which stores the cost of the node and will start to check all the neighbour of 4 and selects the node with least cost. It keeps on adding the neighbour of visited nodes and remove the visited node in the queue. It returns to path if the target node is in visited queue or it keeps going until it reaches the target.

2) A* uses combination of heuristic value(H-score) and G- score. H-score says how far the goal node is? And G- score gives no of nodes transvers from the start node to current node. In this problem we define the H-score as no of misplaced tiles by comparing the current start and goal. G-score is no of node transvers from start node to get to current node so a new function is defined called F- score which is f(n) = h(n)+g(n).

This algorithm moves empty spaces in all directions in start state and calculates the F-Score for each state. After expanding the current state, it is pushed to clos list and newly generation state is pushed to open list. A state with least F- Score is expanded again. This continuous until goal is reached

Source Code

Breath First Search

```
from queue import PriorityQueue
import matplotlib.pyplot as plt
import networkx as nx


def best_first_search(source, target, n):
    visited = [0] * n
    visited[source] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()


def addedge(x, y, cost):
    graph[x].append((y, cost))
```

```
    graph[y].append((x, cost))

G = nx.Graph()
v = int(input("Enter the number of nodes: "))
graph = [[] for i in range(v)]
e = int(input("Enter the number of edges: "))
print("Enter the edges along with their weights:")
for i in range(e):
    x, y, z = list(map(int, input().split()))
    addedge(x, y, z)
    G.add_edge(x, y, weight = z)

source = int(input("Enter the Source Node: "))
target = int(input("Enter the Target/Destination Node: "))
print("\nPath: ", end = "")
best_first_search(source, target, v)
```

Output:

```
Enter the number of nodes: 9
Enter the number of edges: 8
Enter the edges along with their weights:
0 1 4
0 2 7
1 3 5
1 4 9
1 6 7
3 6 8
4 7 9
4 8 5
Enter the Source Node: 0
Enter the Target/Destination Node: 8

Path: 0 1 3 2 6 4 8
```

A*Star Algorithm

```
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\n")
    while True:
        cur = self.open[0]
        print("")
        print("  | ")
        print("  | ")
        print(" \\\'/ \n")
        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]
```

```
        """ sort the opne list based on f value """
        self.open.sort(key = lambda x:x.fval,reverse=False)


puz = Puzzle (3)
puz.process()
```

Result:

Successfully implemented both Breath first search to find the shortest path and A* algorithm to find best solution for 8 puzzles.