Lab_Experiment : 5
Subject : Artificial Intelligence

A.HARICHANDANA
RA1911026010009
CSE-K1

AIM : Developing Best first search and A* Algorithm for real world problems

PROBLEM STATEMENT OF BEST FIRST SEARCH :

Best First Search falls under the category of Heuristic Search or Informed Search.The main idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore.

ALGORITHM :

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until GOAL node is reached

- If OPEN list is empty, then EXIT the loop returning 'False'
- Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node
- If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path

● If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list.

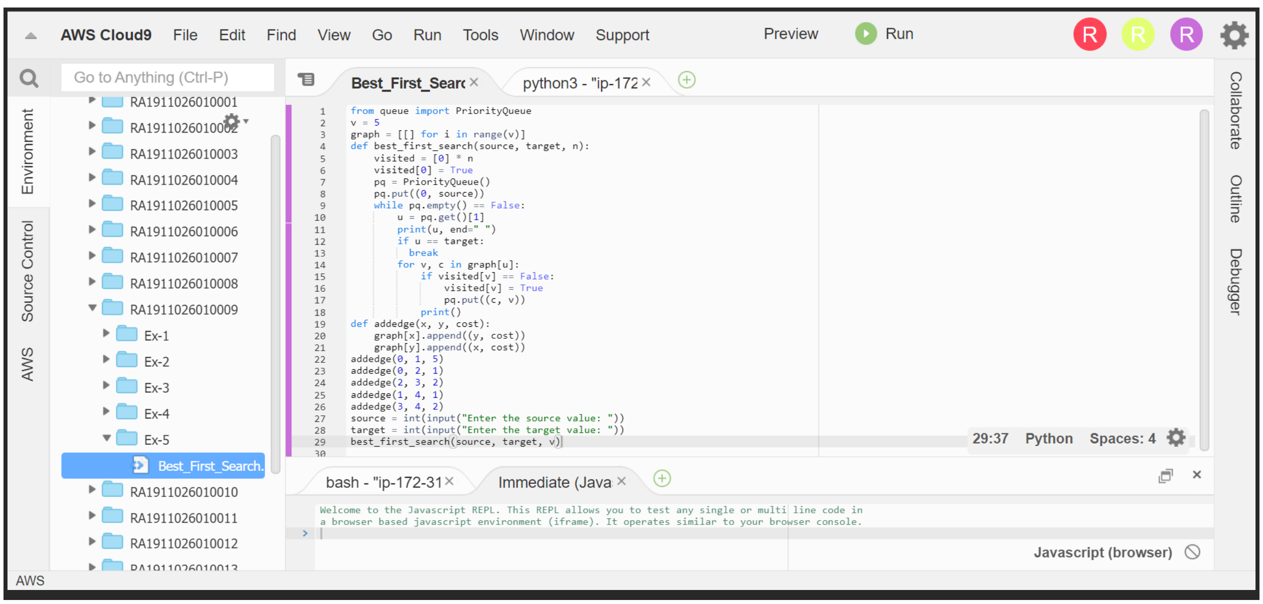
● Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

CODE:

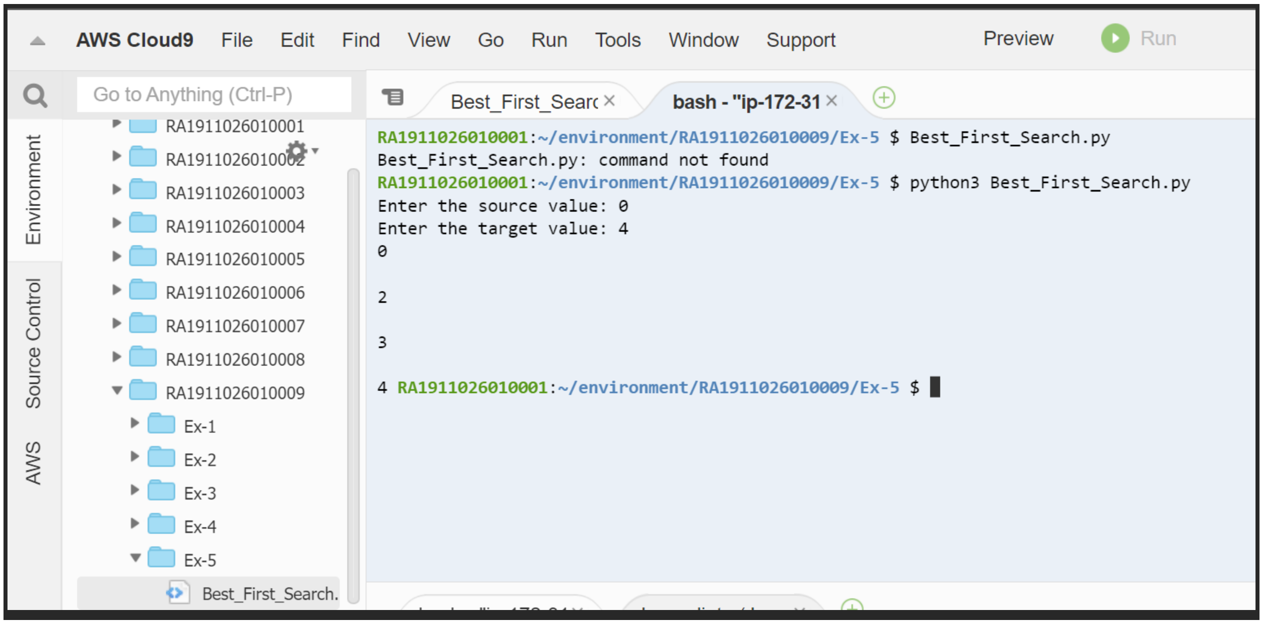
```
from queue import PriorityQueue
v = 5
graph = [[] for i in range(v)]
def best_first_search(source, target, n):
    visited = [0] * n
    visited[0] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == 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))
addedge(0, 1, 5)
```

```
addedge(0, 2, 1)
addedge(2, 3, 2)
addedge(1, 4, 1)
addedge(3, 4, 2)
source = int(input("Enter the source value: "))
target = int(input("Enter the target value: "))
best_first_search(source, target, v)
```

SCREENSHOT OF THE CODE IN AWS :



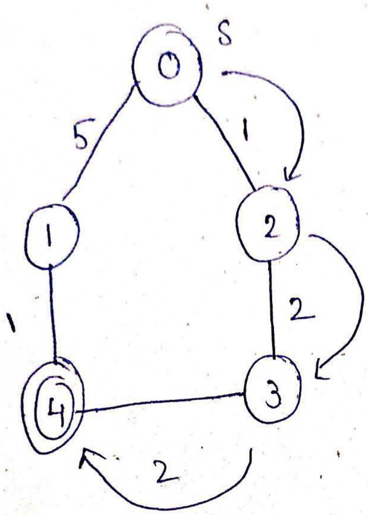
OUTPUT SCREENSHOT IN AWS:



MANUAL COMPUTATION:

Given

Graph :-



∴ The path taken will be 0 2 3 4.

with the cost 5.

OBSERVATION :

This algorithm will traverse the shortest path first in the queue. The time complexity of the algorithm is given by O(n*logn).

where n is a number of nodes. In the worst case, we may have to visit all nodes before we reach the goal. Note that priority queue is implemented using Min(or Max) Heap, and insert and remove operations take O(log n) time.

The performance of the algorithm depends on how well the cost or evaluation function is designed.

PROBLEM STATEMENT OF A* ALGORITHM:

A* Search algorithm is one of the best and one of the most popular techniques used in path-finding and graph traversals.

N-PUZZLE PROBLEM :

N Puzzle is a sliding blocks game that takes place on a k * k grid with ((k * k) - 1) tiles each numbered from 1 to N.

N-Puzzle is a classic 1 player game that teaches the basics of heuristics in arriving at solutions in Artificial Intelligence.

ALGORITHM OF A* ALGORITHM:

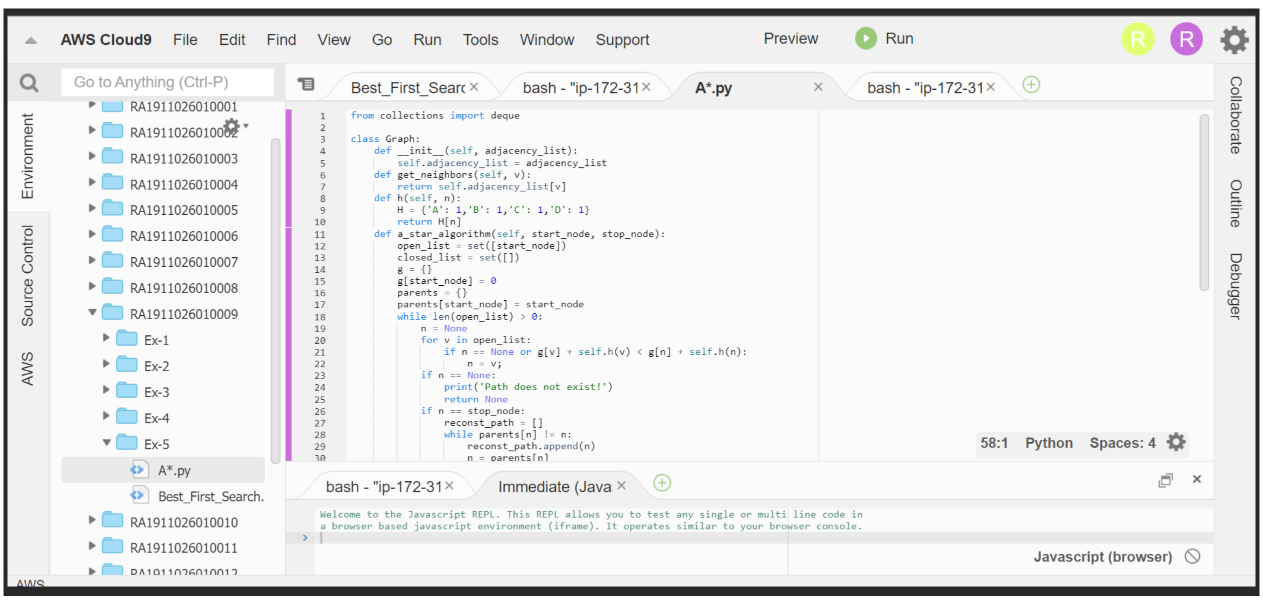
We create two lists – Open List and Closed List (just like Dijkstra Algorithm)

1. Initialize the open list
2. Initialize the closed list put the starting node on the open list (you can leave its f at zero)
3. while the open list is not empty a) find the node with the least f on the open list, call it "q" b) pop q off the open list c) generate q's 8 successors and set their parents to q d) for each successor

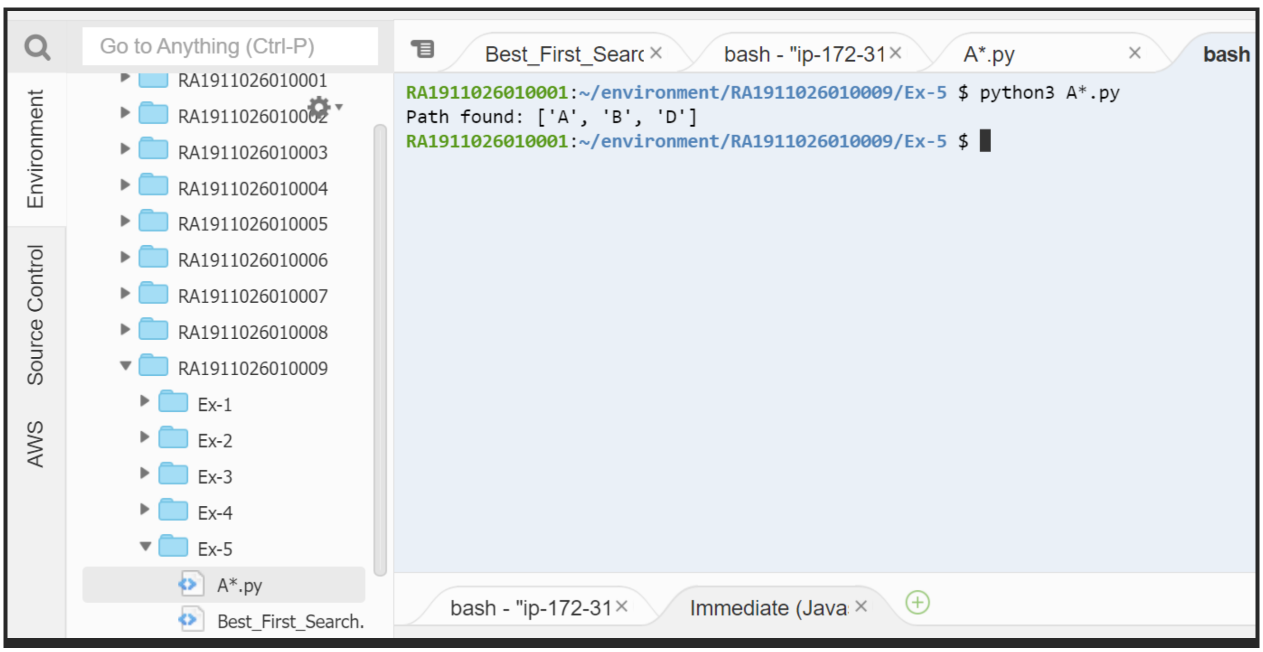
i) f successor is the goal, stop search

ii) else, compute both g and h for successor successor.g = q.g + distance between successor and q successor.h = distance from goal to successor successor.f = successor.g + successor.h
iii) if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor
iv) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list end (for loop) e) push q on the closed list end (while loop)

SCREEN SHOT OF CODE IN AWS :



OUTPUT SCREENSHOT:



CODE :

from collections import deque

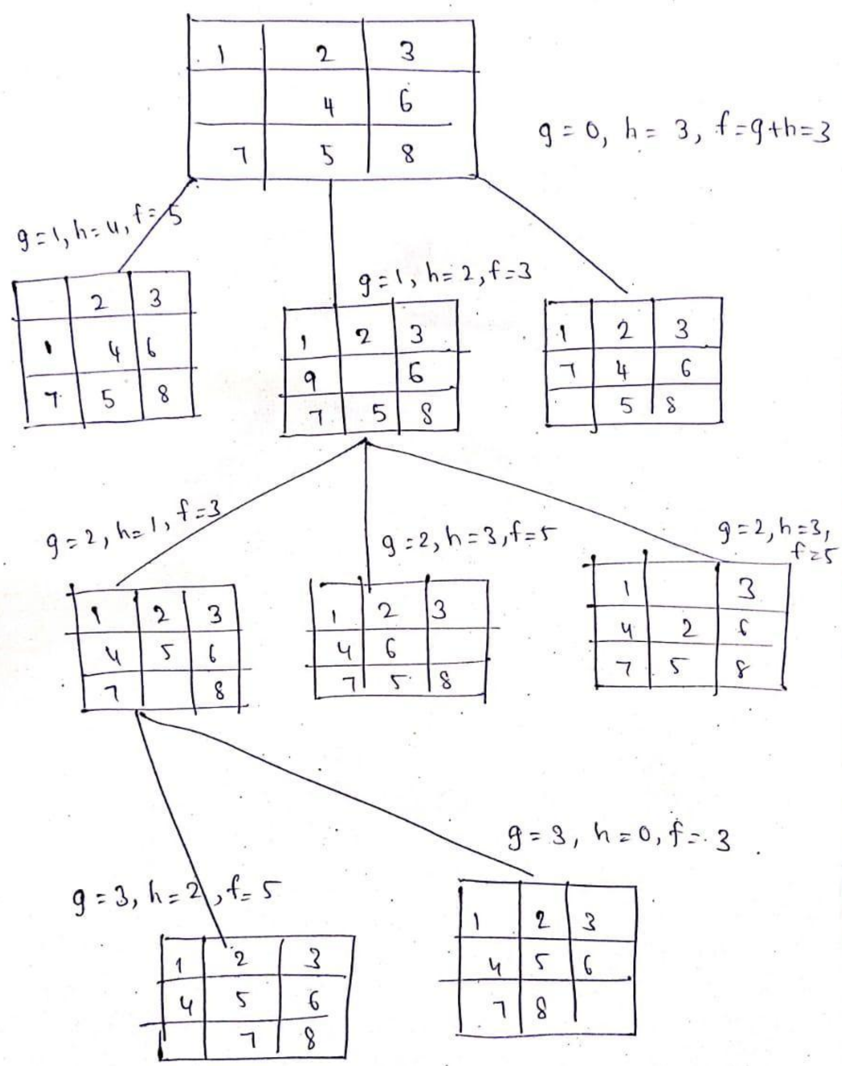
```
class Graph:

    def __init__(self, adjacency_list):

    self.adjacency_list = adjacency_list

    def get_neighbors(self, v):

    return self.adjacency_list[v]

    def h(self, n):

    H = {'A': 1,'B': 1,'C': 1,'D': 1}

    return H[n]

    def a_star_algorithm(self, start_node, stop_node):

    open_list = set([start_node])

    closed_list = set([])

    g = {}

    g[start_node] = 0

    parents = {}

    parents[start_node] = start_node

      while len(open_list) > 0:

    n = None

    for v in open_list:

        if n == None or g[v] + self.h(v) < g[n] + self.h(n):

        n = v;

    if n == None:
```

```
            print('Path does not exist!')
            return None
        if n == stop_node:
            reconst_path = []
            while parents[n] != n:
            reconst_path.append(n)
            n = parents[n]
            reconst_path.append(start_node)
            reconst_path.reverse()
            print('Path found: {}'.format(reconst_path))
            return reconst_path
        for (m, weight) in self.get_neighbors(n):
            if m not in open_list and m not in closed_list:
            open_list.add(m)
            parents[m] = n
            g[m] = g[n] + weight
            else:
            if g[m] > g[n] + weight:
            g[m] = g[n] + weight
            parents[m] = n
            if m in closed_list:
```

```
        closed_list.remove(m)
            open_list.add(m)
    open_list.remove(n)
    closed_list.add(n)
    print('Path does not exist!')
    return None
adjacency_list = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

MANUAL COMPUTATION :

Sample Computation of A* Search Algorithm :-



OBSERVATION :

We first move the empty space in all the possible directions in the start state and calculate the f-score for each state. This is called expanding the current state.
After expanding the current state, it is pushed into the closed list and the newly generated states are pushed into the open list.
A state with the least f-score is selected and expanded again.
This process continues until the goal state occurs as the current state.
Basically, here we are providing the algorithm with a measure to choose its actions. The algorithm chooses the best possible action and proceeds in that path. This solves the issue of generating redundant child states, as the algorithm will expand the node with the least f-score.

RESULT :

Hence, analysis and computation of Best First Search and A* algorithm is done.