Terna Engineering College
Computer Engineering Department

Program: Sem VII

Course: Artificial Intelligence & Soft Computing (AI&SC)

Experiment No. 04

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Blackboard access available)

Roll No. 50	Name: AMEY THAKUR
Class: BE-COMPS-50	Batch: B3
Date of Experiment: 20-08-2021	Date of Submission: 20-08-2021
Grade :	

Aim: To Implement informed A* search methods using C or Java.

B.1 Document created by the student:

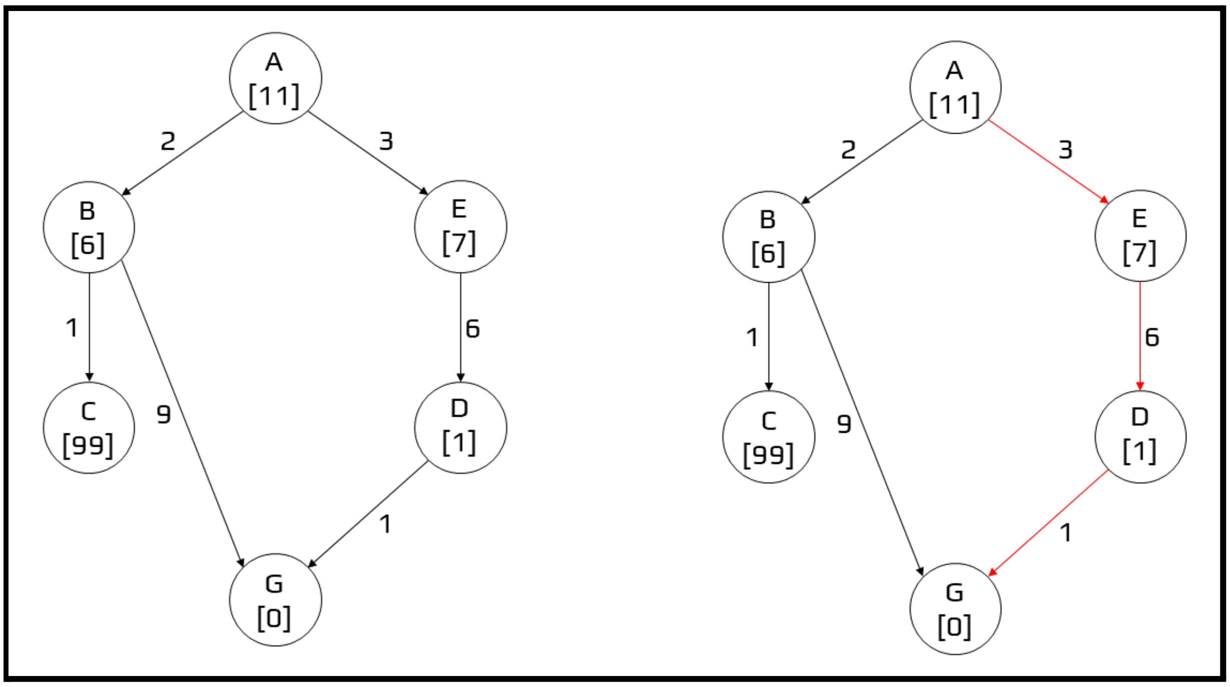
(Write the answers to the questions given in section 4 during the 2 hours of practice in the lab here)

ASTAR.PY

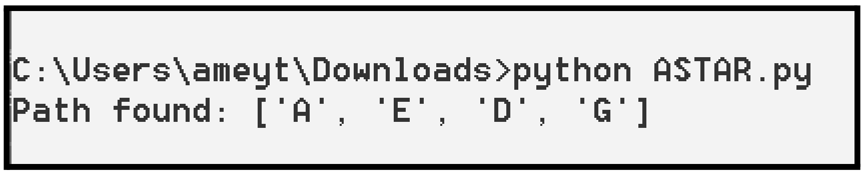
```
def aStarAlgo(start_node, stop_node):

    open_set = set(start_node)
    closed_set = set()
    g = {} #store distance from starting node
    parents = {}# parents contains an adjacency map of all nodes

    #ditance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node


    while len(open_set) > 0:
      n = None
```

```
#node with lowest f() is found
for v in open_set:
    if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
        n = v


if n == stop_node or Graph_nodes[n] == None:
    pass
else:
    for (m, weight) in get_neighbors(n):
        #nodes 'm' not in first and last set are added to first
        #n is set its parent
        if m not in open_set and m not in closed_set:
            open_set.add(m)
            parents[m] = n
            g[m] = g[n] + weight


        #for each node m,compare its distance from start i.e g(m) to the
        #from start through n node
        else:
            if g[m] > g[n] + weight:
                #update g(m)
                g[m] = g[n] + weight
                #change parent of m to n
                parents[m] = n

                #if m in closed set,remove and add to open
                if m in closed_set:
                    closed_set.remove(m)
                    open_set.add(m)

if n == None:
    print('Path does not exist!')
    return None

# if the current node is the stop_node
# then we begin reconstructin the path from it to the start_node
if n == stop_node:
    path = []

    while parents[n] != n:
        path.append(n)
```

```
                n = parents[n]

            path.append(start_node)

            path.reverse()

            print('Path found: {}'.format(path))
            return path


        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)

    print('Path does not exist!')
    return None

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
  if v in Graph_nodes:
    return Graph_nodes[v]
  else:
    return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
      'A': 11,
      'B': 6,
      'C': 99,
      'D': 1,
      'E': 7,
      'G': 0,

    }

    return H_dist[n]

#Describe your graph here
Graph_nodes = {
  'A': [('B', 2), ('E', 3)],
  'B': [('C', 1),('G', 9)],
  'C': None,
  'E': [('D', 6)],
  'D': [('G', 1)],

}
aStarAlgo('A', 'G')
```



Output:



```
C:\Users\ameyt\Downloads>python ASTAR.py
Path found: ['A', 'E', 'D', 'G']
```

B.2 Observations and learning:

(Students are expected to understand the selected topic Prepare a flow of the steps defined in the paper)

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.

In maps, the A* algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state).

Node (also called State) — All potential positions or stops with a unique identification Transition — The act of moving between states or nodes.

Starting Node — Where to start searching Goal Node — The target to stop searching.

Search Space — A collection of nodes, like all board positions of a board game

Cost — Numerical value (say distance, time, or financial expense) for the path from a node to another node.

g(n) — this represents the exact cost of the path from the starting node to any node n h(n) — this represents the heuristic estimated cost from node n to the goal node.

f(n) — lowest cost in the neighbouring node n

Each time A* enters a node, it calculates the cost, f(n), to travel to all of the neighbouring nodes, and then enters the node with the lowest value of f(n).

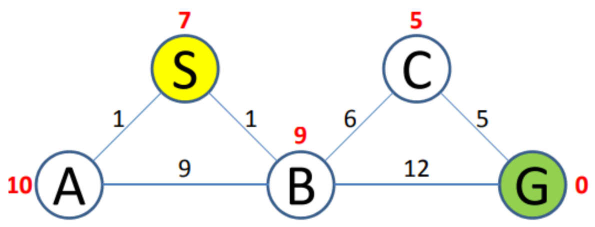
These values we calculate using the following formula: f(n) = g(n) + h(n)

B.3 Conclusion:

Hence we've successfully implemented a program for the A* search algorithm.

B.4 Question of Curiosity:

Q1) Apply A* algorithm in the following example and find the cost



Ans: S → B → C → G [COST = 12]

Q2) What is the other name of informed search strategy?

a. Simple search

b. heuristic search

c. online search

d. none of the above

Ans: b. heuristic search