**Artificial and Computational Intelligence Assignment - Group 108**

**Assignment 1 - Question 8**

**Team Members:**

|  |  |
| --- | --- |
| **Name** | **BITSID** |
| Surakaram Adarsh Goud | 2019HC04682 |
|  |  |
| Dhandapani M | 2019HC04716 |
|  |  |
|  |  |

**Explanation of the heuristic and algorithm chosen:**

In the given problem Arun wants to travel from city S to city G by exploring least number of cities in the way. When Arun arrives at an intermediate city, he gets to know about the distance from that city to G. This approximation in distance from any particular city to G is the heuristic given.

In A\* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs. Using this heuristics of the neighboring cities and also the actual distance to reach current city, A\* algorithm can be used to find the next best path to be taken, by computing the potential distance that it might take to the goal node from those neighboring cities and choosing the lowest value among those neighbors. So for every neighbor we calculate ‘f’ value and choose the lowest ‘f’ value (fitness number) and only explore that neighboring city to move forward which perfectly fits our problem statement of exploring least number of cities.



**Representation of the environment, fringe and the data structures used:**

To represent the graph G’, we have used the **Networkx** python library.

Nodes in the graph hold the city names and also the heuristic.

And the edges connect the nodes also hold the distance between any two nodes.

Using these nodes and edges we have connected the graph G.

We have used python library **queue** to make use of **PriorityQueue** to store the list of **opened** nodes based on the computed ‘f’ value as priority.

A simple python list to hold the list of **closed** cities.

A **parent\_tracker** list to track the immediate parent city from which the city is explored.

And **path** list to store the path used to travel to goal city G

We have also used the following helper functions:

**get\_parent(current):**

Which accepts any node traversed so far and gives its immediate parent city from which this city is explored. This function iterates through the **parent\_tracker** list.

**get\_g(current, start\_node):**

This function accepts the current node we are exploring and also the start node and computes and returns the cost/distance incurred to reach the current city.

Finally **astar\_search(graph, start\_node, goal\_node):**

This is the function where the A\* Search Algorithm is implemented. This function accepts a Graph, a start node and goal node and returns the path that it discovered to reach the goal node. And also the cost incurred to reach the goal node.

**Implementation of the algorithm in PYTHON**:

**A\* Algorithm:**

Please refer to a\_start.py for the PYTHON code. And please find below the pseudo algorithm.

**Step1**: Place the starting node in the OPEN list.

**Step 2**: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

**Step 3**: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

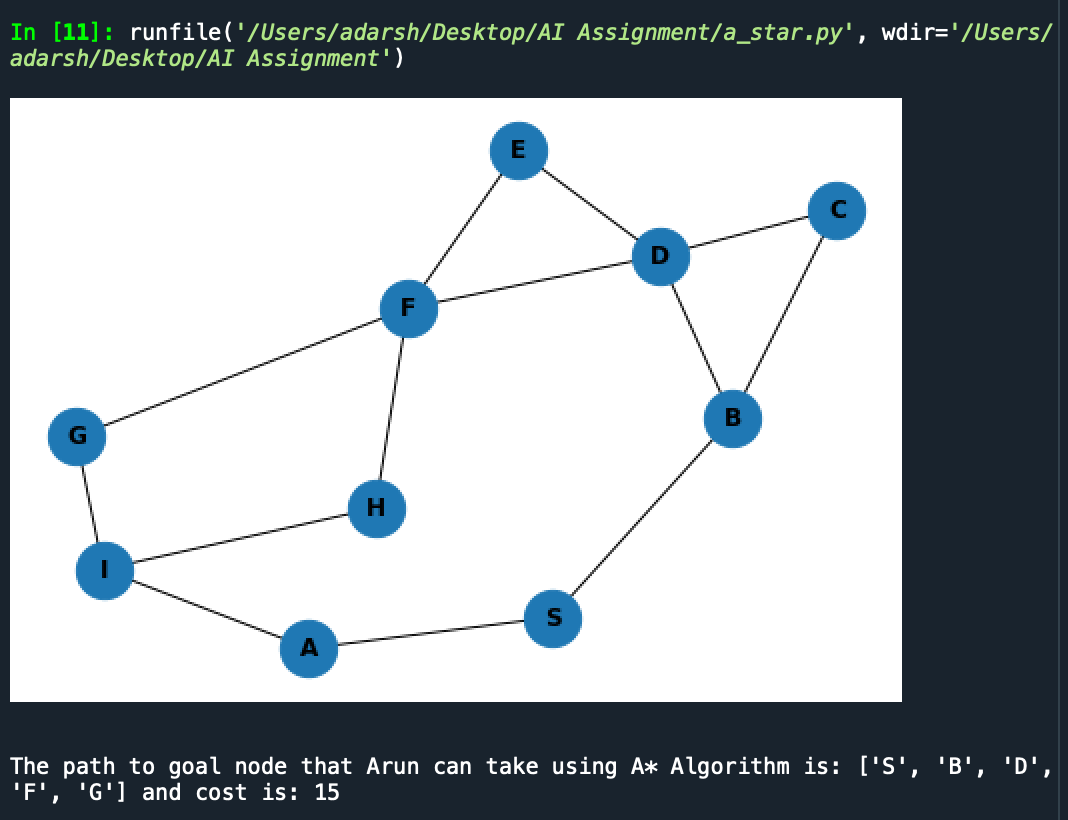
**Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

**Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

**Step 6**: Return to Step 2.

**The path taken by arun to reach the destination:**

**Please find below a screenshot of the output with the path and the cost after executing the PYTHON code:**

****