Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 17-1-2021	Date of Submission: 31-1-2022

To implement A* Algorithm

Theory

This Algorithm is the advanced form of the BFS algorithm (Breadth-first search), which searches for the shorter path first than, the longer paths. It is a complete as well as an optimal solution for solving path and grid problems.

$$f(n) = g(n) + h(n)$$

Where
g (n): The actual cost path from the start node to the current node.
h (n): The actual cost path from the current node to goal node.
f (n): The actual cost path from the start node to the goal node.

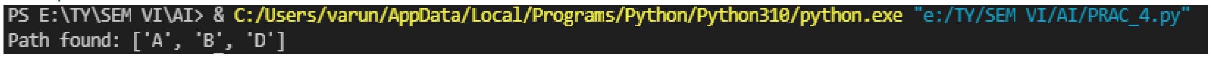
Algorithm

1. Firstly, Place the starting node into OPEN and find its f (n) value.
2. Then remove the node from OPEN, having the smallest f (n) value. If it is a goal node, then stop and return to success.
3. Else remove the node from OPEN and find all its successors.
4. Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.
5. Goto Step-2.
6. Exit.

Code

```python
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')
```

Output



```
PS E:\TY\SEM VI\AI> & C:/Users/varun/AppData/Local/Programs/Python/Python310/python.exe "e:/TY/SEM VI/AI/PRAC_4.py"
Path found: ['A', 'B', 'D']
```