Niyati_Savant_TE_C31_2103156

Experiment 4

AIM - Implement Greedy Best First Search / A* search algorithm in Python.

Code: Greedy Best First Search –

```
graph = {
    'S': [('A', 3), ('B', 2)],
    'A': [('C', 4), ('D', 1)],
    'B': [('E', 3), ('F', 1)],
    'E': [('H', 5)],
    'F': [('I', 2), ('G', 3)],
    'C': [('A', 4)],
    'D': [('A', 1)],
    'E': [('B', 3)],
    'I': [('F', 2)],
    'G': [('F', 3)]
}



dist = {}
h = {
    'S': 13, 'A': 12, 'B': 4, 'C': 7, 'D': 3, 'E': 8, 'F': 2, 'H': 4, 'I': 9, 'G': 0
}



def bestFit(start, target):
    q = []
    q.append((h[start], start))
    for x in graph:
        dist[x] = 10000
    dist[start] = 0

    while q:
```

```
        print()
        q = sorted(q)
        curr = q.pop(0)
        print(curr)
        curr_dist = dist[curr[1]]
        for node in graph[curr[1]]:
            if curr_dist+node[1] < dist[node[0]]:
                print(node, end=" ")
                dist[node[0]] = curr_dist + node[1]
                q.append((h[node[0]], node[0]))
    print(dist)

print("Niyati's Code for Greedy Best First Search")
root_node = input("Enter Root Node: ")
goal_node = input("Enter Goal Node: ")

bestFit(root_node, goal_node)
```

Output –

```
Niyati's Code for Greedy Best First Search
Enter Root Node: S
Enter Goal Node: G

(13, 'S')
('A', 3) ('B', 2)
(4, 'B')
('E', 3) ('F', 1)
(2, 'F')
('I', 2) ('G', 3)
(0, 'G')

(8, 'E')

(9, 'I')

(12, 'A')
('C', 4) ('D', 1)
(3, 'D')

(7, 'C')
{'S': 0, 'A': 3, 'B': 2, 'E': 5, 'F': 3, 'C': 7, 'D': 4, 'I': 5, 'G': 6}
PS C:\Engineering\3rd Year\Sem VI\PRACTICALS\AI>
```

Niyati_Savant_TE_C31_2103156

Code: A* Search –

```
import heapq


class Node:
    def __init__(self, name, cost=0, parent=None):
        self.name = name
        self.cost = cost
        self.parent = parent
        self.priority = 0

    def __lt__(self, other):
        return self.priority < other.priority



def heuristic(node, goal, heuristic_map):
    return heuristic_map.get(node.name, 0)



def a_star_search(start, goal, graph, heuristic_map):
    open_list = []
    heapq.heappush(open_list, (0, Node(start)))
    visited = set()

    while open_list:
        _, current_node = heapq.heappop(open_list)

        if current_node.name in visited:
            continue
        visited.add(current_node.name)

        if current_node.name == goal:
            path = []
```

```
            while current_node:
                path.append((current_node.name, current_node.priority))  # Include priority in the result
                current_node = current_node.parent
            return path[::-1]

        for neighbor, cost in graph[current_node.name].items():
            if neighbor not in visited:
                neighbor_node = Node(neighbor, current_node.cost + cost, current_node)
                neighbor_node.priority = neighbor_node.cost + heuristic(neighbor_node, goal,
heuristic_map)
                heapq.heappush(open_list, (neighbor_node.priority, neighbor_node))

    return None



# Graph data
graph = {
    'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
    'Bucharest': {'Urziceni': 85, 'Pitesti': 101, 'Giurgiu': 90, 'Fagaras': 211},
    'Craiova': {'Drobeta': 120, 'Rimnicu': 146, 'Pitesti': 138},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Eforie': {'Hirsova': 86},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Giurgiu': {'Bucharest': 90},
    'Hirsova': {'Urziceni': 98, 'Eforie': 86},
    'Iasi': {'Neamt': 87, 'Vaslui': 92},
    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
    'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
    'Neamt': {'Iasi': 87},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Pitesti': {'Rimnicu': 97, 'Craiova': 138, 'Bucharest': 101},
    'Rimnicu': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},
```

```
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu': 80},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},
    'Vaslui': {'Iasi': 92, 'Urziceni': 142},
    'Zerind': {'Arad': 75, 'Oradea': 71}
}

# Heuristic values
heuristic_map = {
    'Arad': 366,
    'Bucharest': 0,
    'Craiova': 160,
    'Drobeta': 242,
    'Eforie': 161,
    'Fagaras': 176,
    'Giurgiu': 77,
    'Hirsova': 151,
    'Iasi': 226,
    'Lugoj': 244,
    'Mehadia': 241,
    'Neamt': 234,
    'Oradea': 380,
    'Pitesti': 100,
    'Rimnicu': 193,
    'Sibiu': 253,
    'Timisoara': 329,
    'Urziceni': 80,
    'Vaslui': 199,
    'Zerind': 374
}
```

Niyati_Savant_TE_C31_2103156

```
cities = [
    "Arad",
    "Bucharest",
    "Craiova",
    "Dobreta",
    "Eforie",
    "Fagaras",
    "Giurgiu",
    "Hirsova",
    "Iasi",
    "Lugoj",
    "Mehadia",
    "Neamt",
    "Oradea",
    "Pitesti",
    "Rimnicu_Vilcea",
    "Sibiu",
    "Timisoara",
    "Urziceni",
    "Vaslui",
    "Zerind"
]

print("Niyati's Code for A* algorithm")
for i in range(len(cities)):
    print(f"{i}. {cities[i]}")
start_city = int(input("Enter no. for  Start City: "))
start_city = cities[start_city]

goal_city = int(input("Enter no. Goal City: "))
goal_city = cities[goal_city]
```

path = a_star_search(start_city, goal_city, graph, heuristic_map)

print(f"Path from {start_city} to {goal_city}:", path)

Output –

```
Niyati's Code for A* algorithm
0. Arad
1. Bucharest
2. Craiova
3. Dobreta
4. Eforie
5. Fagaras
6. Giurgiu
7. Hirsova
8. Iasi
9. Lugoj
10. Mehadia
11. Neamt
12. Oradea
13. Pitesti
14. Rimnicu_Vilcea
15. Sibiu
16. Timisoara
17. Urziceni
18. Vaslui
19. Zerind
Enter no. for  Start City: 0
Enter no. Goal City: 1
Path from Arad to Bucharest: [('Arad', 0), ('Sibiu', 393), ('Rimnicu', 413), ('Pitesti', 417), ('Bucharest', 418)]
PS C:\Engineering\3rd Year\Sem VI\PRACTICALS\AI>
```