Ex.no.3 **Greedy best first and A* Search algorithm**

Date:

Aim:

To model the Greedy best first and A* Search algorithm in generic ways

Greedy Best First Search: Greedy Best-First Search is an algorithm used for searching the shortest path from a starting node to a goal node in a graph or tree. It is called "greedy" because it makes decisions based solely on the heuristic evaluation of each node, always choosing the node that appears closest to the goal according to the heuristic function.

A* Search: A* search is an algorithm that finds the shortest path between nodes by using a cost function \(f(n) = g(n) + h(n) \), where \(g(n) \) is the cost from the start node to \(n \), and \(h(n) \) is the estimated cost from \(n \) to the goal. It uses a priority queue to explore nodes with the lowest estimated total cost first. A* is both complete and optimal, given an admissible heuristic. It is commonly used in pathfinding and graph traversal applications

Algorithm: Greedy Best First Search

STEP 1: Start.

STEP 2: Import the `PriorityQueue` class.

STEP 3: Initialize the `open_list` as a priority queue. For A*, initialize `g_cost` to track the path cost from the start to each node.

STEP 4: While the `open_list` is not empty, pop the node with the lowest priority. If this node is the goal, reconstruct the path and return it.

STEP 5: For each neighbor of the current node:

- **Greedy Best-First Search**: Add the neighbor to `open_list` with its heuristic value if not visited.
- *A Search**: Calculate the tentative `g_cost`. If it's lower than the previous cost, update the `g_cost`, calculate the `f_score` (tentative `g_cost` + heuristic), and add the neighbor to `open_list`.

STEP 6: If the goal is reached, return the path (and path cost for A*). If not, return that no path exists.

STEP 7: Stop.

Program:

```
from queue import PriorityQueue

def greedy_best_first_search(graph, start, goal, heuristic):
    open_list = PriorityQueue()
    open_list.put((heuristic[start], start))
    visited = set()
    came_from = {}
    while not open_list.empty():
        _, current_node = open_list.get()
        if current_node == goal:
            path = []
            while current_node in came_from:
                path.append(current_node)
                current_node = came_from[current_node]
            path.append(start)
            path.reverse()
            return path
        visited.add(current_node)
        for neighbor in graph[current_node]:
            if neighbor not in visited:
                visited.add(neighbor)
                came_from[neighbor] = current_node
                open_list.put((heuristic[neighbor], neighbor))
    return None

def a_star_search(graph, start, goal, heuristic):
    open_list = PriorityQueue()
```

```
    open_list.put((heuristic[start], start))
    g_cost = {start: 0}
    came_from = {}
    while not open_list.empty():
        _, current_node = open_list.get()
        if current_node == goal:
            path = []
            while current_node in came_from:
                path.append(current_node)
                current_node = came_from[current_node]
            path.append(start)
            path.reverse()
            # Calculate the path cost
            path_cost = 0
            for i in range(len(path) - 1):
                path_cost += graph[path[i]].get(path[i + 1], 0)
            return path, path_cost
        for neighbor, cost in graph[current_node].items():
            tentative_g_cost = g_cost[current_node] + cost
            if neighbor not in g_cost or tentative_g_cost < g_cost[neighbor]:
                g_cost[neighbor] = tentative_g_cost
                f_score = tentative_g_cost + heuristic[neighbor]
                open_list.put((f_score, neighbor))
                came_from[neighbor] = current_node
    return None, float('inf')

graph = {
    'A': {'B': 1, 'C': 10},
```

```
    'B': {'D': 1, 'E': 5},
    'C': {'F': 2, 'E': 1},
    'D': {'G': 1},
    'E': {'G': 1},
    'F': {'G': 1},
    'G': {}
}

heuristic = {
    'A': 7,
    'B': 6,
    'C': 1,
    'D': 4,
    'E': 2,
    'F': 2,
    'G': 0
}
start = 'A'
goal = 'G'

# Greedy Best-First Search
path_gbfs = greedy_best_first_search(graph, start, goal, heuristic)
print("Greedy Best-First Search path:", path_gbfs)

# A* Search
path_astar, path_cost_astar = a_star_search(graph, start, goal, heuristic)
print("A* Search path:", path_astar)
print("A* Search path cost:", path_cost_astar)
```

Output:

Greedy Best-First Search path: ['A', 'C', 'E', 'G']

A* Search path: ['A', 'B', 'D', 'G']

A* Search path cost: 3

Rubrics	Marks
Observation (20)	
Record (5)	
Total (25)	

Result:

Thus, the Greedy best first and A*Search are executed and output is verified successfully