Experiment No. 9

Aim

To implement a program on **Greedy Best-First Search (GBFS) Algorithm** for finding the optimal path between two nodes in a graph using heuristic information.

Theory

The **Greedy Best-First Search (GBFS)** algorithm is a **heuristic search algorithm** that explores a graph by expanding the node that appears to be **closest to the goal** based on a heuristic function. It is a **best-first** strategy because it always selects the most promising node according to a specific rule — typically the **heuristic estimate** of the distance from the current node to the goal.

Key Concepts:

- **Heuristic Function (h(n))**:
  A function that estimates the cost from node `n` to the goal node.
  Common examples include:
  - Euclidean distance
  - Manhattan distance
  - Straight-line distance
- **Greedy Nature:**
  The algorithm only considers the heuristic value `h(n)` and **ignores the path cost (g(n))**, unlike A* Search which considers both `g(n)` and `h(n)`.
- **Evaluation Function:**
  [
  f(n) = h(n)
  ]
  where:
  - `f(n)` = estimated total cost of the cheapest solution through node `n`
  - `h(n)` = estimated cost from node `n` to goal node

Characteristics:

- **Completeness:** Not guaranteed (may get stuck in loops or dead ends)
- **Optimality:** Not guaranteed (finds a path, not necessarily the shortest)
- **Time Complexity:** (O(b^m)), where `b` is the branching factor and `m` is the maximum depth of the search.

- **Space Complexity:** (O(b^m)), since all nodes are stored in memory.

Algorithm

Greedy Best-First Search Algorithm:

1. **Initialize**
   - Start with the initial node.
   - Create an **open list** (priority queue) containing the start node.
   - Create an **empty closed list** to store visited nodes.
2. **Loop until goal is found or open list is empty:**
   - Select the node `n` from the open list with the **lowest heuristic value** `h(n)`.
   - If `n` is the **goal node**, return the path.
   - Otherwise:
     - Move `n` from the open list to the closed list.
     - For each **neighbor** of `n`:
       - If neighbor not in closed list:
         - Compute `h(neighbor)`.
         - Add neighbor to the open list.
3. **If the open list becomes empty** before finding the goal, return failure (no path found).

Code

Language Used: Python

```
# Greedy Best-First Search Algorithm Implementation in Python

from queue import PriorityQueue

# Define the graph using adjacency lists
graph = {
    'A': [('B', 0), ('C', 0)],
    'B': [('D', 0), ('E', 0)],
    'C': [('F', 0)],
    'D': [],
    'E': [('F', 0)],
    'F': []
}

# Heuristic values for each node (h(n))
heuristic = {
```

```
    'A': 10,
    'B': 8,
    'C': 5,
    'D': 7,
    'E': 3,
    'F': 0
}

def greedy_best_first_search(graph, start, goal, heuristic):
    # PriorityQueue stores nodes as (h(n), node)
    open_list = PriorityQueue()
    open_list.put((heuristic[start], start))
    visited = set()

    # To store the path
    path = []

    while not open_list.empty():
        # Get the node with smallest heuristic value
        h, current = open_list.get()
        path.append(current)
        visited.add(current)

        # Check if goal is reached
        if current == goal:
            print("Goal reached!")
            return path

        # Explore neighbors
        for (neighbor, cost) in graph[current]:
            if neighbor not in visited:
                open_list.put((heuristic[neighbor], neighbor))

    # If goal not found
    print("Goal not reachable.")
    return path

# Example execution
start_node = 'A'
goal_node = 'F'

result_path = greedy_best_first_search(graph, start_node, goal_node,
heuristic)
print("Path Traversed:", result_path)
```

Input/Output Examples

Input:

```
Start Node: A
Goal Node: F
```

Heuristic Table:

Node	h(n)
A	10
B	8
C	5
D	7
E	3
F	0

Output:

```
Goal reached!
Path Traversed: ['A', 'C', 'F']
```

Explanation:

- Start at `A` → `C` has the smaller heuristic (5) than `B` (8).
- From `C` → `F` has heuristic 0, which is the goal.

Analysis of Code and Algorithm

- **Logic Flow:**
  - Uses a **priority queue** (min-heap) to always expand the node with the smallest heuristic.
  - Maintains a **visited set** to avoid revisiting nodes.
  - Stops immediately when the goal node is reached.
- **Time Complexity:**
  (O(b^m)) — where `b` is the branching factor and `m` is the depth of the shallowest goal node.
- **Space Complexity:**
  (O(b^m)) — because all nodes may be stored in the open list at some point.
- **Advantages:**
  - Fast for problems with good heuristic functions.

    - Simple and easy to implement.
- **Limitations:**
    - Not guaranteed to find the optimal path.
    - Can get stuck in local minima.

Real-Life Applications

1. **Pathfinding in Maps and Navigation Systems**
   Used to estimate the shortest or fastest route based on proximity to the destination.
2. **Game AI (Non-Player Character Movement)**
   Helps NPCs or agents find the shortest path toward a target position.
3. **Robot Motion Planning**
   Used to plan paths for autonomous robots by estimating distance to the goal.

Conclusion

In this experiment, we successfully implemented the **Greedy Best-First Search Algorithm** to find a path between two nodes in a graph. The algorithm uses heuristic information to guide its search, always choosing the node that appears closest to the goal. While efficient and intuitive, GBFS does not always guarantee the optimal path.
This practical helped in understanding the role of heuristics in AI search algorithms and how they affect decision-making efficiency.
