CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

NAAC GRADE A+
Accredited University

Experiment 3

Student Name: Rahul Saxena

UID: 24MCI10204

Branch: MCA (AI-ML)

Section/Group: MAM - 3(B)

Semester: II

Subject Name: Artificial Intelligence Lab [24CAP-674]

Aim: Implement Best-First Search Algorithm with a chosen heuristic.

Steps:

1. **Define the Heuristic Function:**
   - Choose a heuristic function (e.g., Manhattan Distance, Euclidean Distance) to estimate the cost from the current node to the goal.
   - Implement the heuristic function in Python.
2. **Initialize Data Structures:**
   - Use a priority queue (e.g., heapq) to store nodes based on their heuristic values.
   - Create a dictionary (came_from) to keep track of the path by storing the parent of each node.
3. **Start the Search:**
   - Push the start node into the priority queue with its heuristic value.
   - Mark the start node as visited by adding it to the came_from dictionary.
4. **Explore Nodes:**
   - Pop the node with the lowest heuristic value from the priority queue.
   - If the popped node is the goal, stop the search.
   - Otherwise, explore its neighbors and add them to the priority queue if they haven't been visited.
5. **Reconstruct the Path:**
   - Once the goal is reached, backtrack from the goal to the start using the came_from dictionary.
   - Reverse the path to get the correct order from start to goal.

Source Code:

```
import heapq
import networkx as nx
import matplotlib.pyplot as plt

def heuristic(a, b):
    return 1  # Assigning a heuristic of 1 for simplicity (change as needed)

# Best-First Search Algorithm
def best_first_search(graph, start, goal):
    frontier = []
```

```
    heapq.heappush(frontier, (heuristic(start, goal), start))
    came_from = {}
    came_from[start] = None

    while frontier:
        _, current = heapq.heappop(frontier)

        if current == goal:
            break

        for next_node in graph[current]:
            if next_node not in came_from:
                priority = heuristic(next_node, goal)
                heapq.heappush(frontier, (priority, next_node))
                came_from[next_node] = current

    path = []
    current = goal
    while current != start:
        path.append(current)
        current = came_from[current]
    path.append(start)
    path.reverse()

    return path

if __name__ == "__main__":
    graph = {
        'A': ['B', 'C'],
        'B': ['D', 'E'],
        'C': ['F'],
        'D': [],
        'E': ['G'],
        'F': ['G'],
        'G': []
    }

    start = 'A'
    goal = 'G'

    path = best_first_search(graph, start, goal)
    print("Path found:", path)

    # Generate the graph using networkx
    G = nx.DiGraph(graph)
```

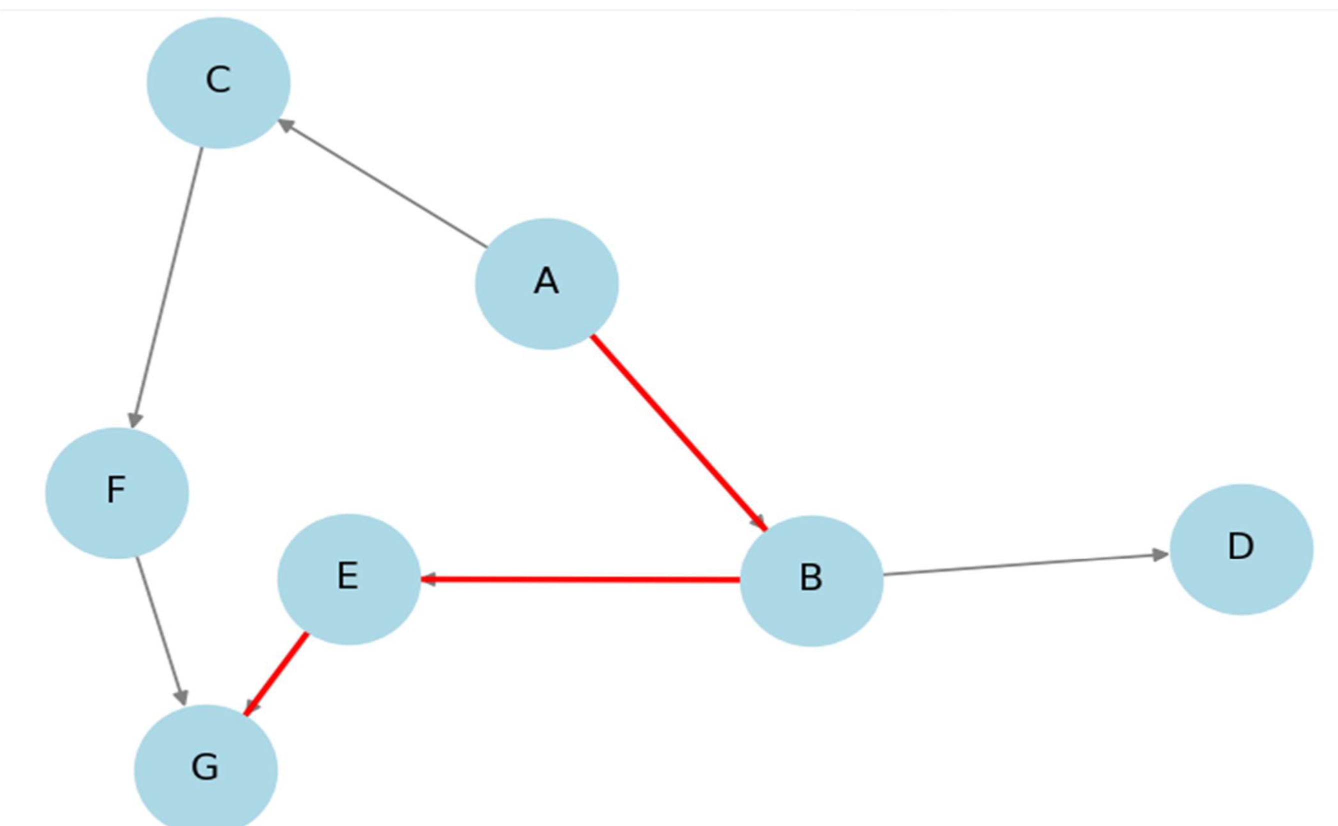
```
pos = nx.spring_layout(G)

plt.figure(figsize=(6, 4))
nx.draw(G, pos, with_labels=True, node_color='lightblue', edge_color='gray', node_size=2000, font_size=12)
path_edges = [(path[i], path[i+1]) for i in range(len(path)-1)]
nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='red', width=2)

plt.title("Best-First Search Path")
plt.show()
```

Output:

```
PS D:\MCA\Semester 2\AI Practice> & C:/Users/saxen/AppData/Local/Programs/Python/Pyt
irstSearch.py"
Path found: ['A', 'B', 'E', 'G']
```



Learning Outcome:

1. Understand Heuristic-Based Search:
    - Learn how heuristic functions guide the search process by estimating the cost to the goal.
    - Recognize the importance of choosing an appropriate heuristic for different problems.
2. Work with Priority Queues:
    - Gain experience using priority queues (e.g., heapq in Python) to prioritize nodes based on their heuristic values.
3. Implement Graph Traversal:
    - Learn how to traverse a graph using a heuristic-driven approach.
    - Understand the difference between Best-First Search and other search algorithms like Breadth-First Search (BFS) or Depth-First Search (DFS).
4. Path Reconstruction:
    - Practice reconstructing the path from the goal to the start using a came_from dictionary.x