Experiment 04

Greedy Best FS:

```
import heapq

def gbfs(graph, start, goal, heuristic):
    visited = set()
    priority_queue = [(heuristic[start], start)]  # Priority queue sorted by heuristic value
    path = {start: None}

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

    if current_node == goal:
    return construct_path(path, start, goal)

    visited.add(current_node)

    for neighbor in graph[current_node]:
    if neighbor not in visited:
        heapq.heappush(priority_queue, (heuristic[neighbor], neighbor))
        path[neighbor] = current_node

    return None

def construct_path(path, start, goal):
    current_node = goal
    path_sequence = []
    while current_node:
    path_sequence.insert(0, current_node)
    current_node = path[current_node]

    return path_sequence

graph = {
    'A': ['S', 'T', 'Z'],
    'S': ['A', 'F', 'O', 'R'],
    'T': [],
    'Z': [],
    'F': ['S', 'B']
}
```

```
start_node = input("Enter the start node: ")
goal_node = input("Enter the goal node: ")

heuristic = {
        'A': 366,
        'S': 253,
        'F': 176 ,
        'T': 329,
        'O': 380,
        'Z': 374,
        'R': 193,
        'B': 0
}

gbfs_path = gbfs(graph, start_node, goal_node, heuristic)

if gbfs_path:
        print('Path:', gbfs_path)
        print(f"Goal '{goal_node}' found using GBFS.")
else:
        print(f"Goal '{goal_node}' not found using GBFS.")
```

OUTPUT:

```
Enter the start node:  A
Enter the goal node:  B

Path: ['A', 'S', 'F', 'B']
Goal 'B' found using GBFS.
```

A* Search:

```
import heapq

def astar(graph, start, goal, heuristic, cost):
    visited = set()
    priority_queue = [(heuristic[start], 0, start)]  # Priority queue sorted by f-value (heuristic + cost)
    path_cost = {start: 0}
    path = {start: None}

    while priority_queue:
        print("OPEN LIST: ",priority_queue)
        _, current_cost, current_node = heapq.heappop(priority_queue)

        if current_node == goal:
            return construct_path(path, start, goal)

        visited.add(current_node)

        for neighbor, edge_cost in graph[current_node].items():
            total_cost = path_cost[current_node] + edge_cost

            if neighbor not in visited or total_cost < path_cost[neighbor]:
                path_cost[neighbor] = total_cost
                heapq.heappush(priority_queue, (total_cost + heuristic[neighbor], total_cost,
neighbor))
                path[neighbor] = current_node

    return None

def construct_path(path, start, goal):
    current_node = goal
    path_sequence = []
    while current_node:
        path_sequence.insert(0, current_node)
        current_node = path[current_node]

    return path_sequence

# Example usage
```

```
graph = {
    'A': {'B': 1, 'C': 2},
    'B': {'D': 3, 'E': 4},
    'C': {'F': 1},
    'D': {},
    'E': {},
    'F': {}
}

start_node = input("Enter the start node: ")
goal_node = input("Enter the goal node: ")

heuristic = {
    'A': 3,
    'B': 2,
    'C': 4,
    'D': 1,
    'E': 1,
    'F': 0
}

cost = {
    'A': 0,
    'B': 1,
    'C': 2,
    'D': 3,
    'E': 4,
    'F': 5
}

astar_path = astar(graph, start_node, goal_node, heuristic, cost)
print()
if astar_path:
    print('Path:', astar_path)
    print(f"Goal '{goal_node}' found using A*.")
else:
    print(f"Goal '{goal_node}' not found using A*.")
```

OUTPUT:

```
Enter the start node: A
```

```
Enter the goal node: F
OPEN LIST:  [(3, 0, 'A')]
OPEN LIST:  [(3, 1, 'B'), (6, 2, 'C')]
OPEN LIST:  [(5, 4, 'D'), (6, 2, 'C'), (6, 5, 'E')]
OPEN LIST:  [(6, 2, 'C'), (6, 5, 'E')]
OPEN LIST:  [(3, 3, 'F'), (6, 5, 'E')]

Path: ['A', 'C', 'F']
Goal 'F' found using A*.

Process finished with exit code 0
```