INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG

Lab report-4

Course code: CSE-3636
Course Titlle : Artificial Intiligance Lab

Submitted To:

      Md Safayet Hossen
      Department of CSE
      International Islamic University Chittagong

Submitted By:

      Md.Riaz Ahmed
      Id: C201060
      Section: 6BM
      Semester: 6th

Date of submission : 25/03/23

Lab -4 : Best First search

```python
import heapq
def best_first_search(graph, start, goal):

    """We create a priority queue to use as our queue for the search.
     We add the starting node to the queue with a priority of 0."""
    queue = [(0, start)]

    """We create a set to keep track of the nodes we have visited so far.
     We add the starting node to the set."""
    visited = set()
    visited.add(start)

    # Loop through the queue until it is empty
    """We enter a loop that continues until the queue is empty. In each iteration of the loop,
    we get the node with the lowest cost so far from the queue (using the heappop() method).
    If the node is the goal, we return the cost of the path to the goal."""
    while queue:
        cost, node = heapq.heappop(queue)
        if node == goal:
            return cost
        # Visit all the adjacent nodes
        """We loop through all the adjacent nodes of the current node.
        If a neighbor has not been visited yet,
        we add it to the queue with its cost as the priority and mark it as visited."""
        for neighbor, weight in graph[node]:
            if neighbor not in visited:
                # Add the neighbor to the queue with its estimated cost as the priority
                heapq.heappush(queue, (cost + weight, neighbor))
```

best_first_search()

```
    # If we reach this point, there is no path from the start to the goal
    return None

graph = {
    'A': [('B', 2), ('C', 3)],
    'B': [('D', 1), ('E', 5)],
    'C': [('F', 7)],
    'D': [('F', 2)],
    'E': [('F', 3)],
    'F': []
}
cost = best_first_search(graph, 'A', 'F')
print(cost)
```

Code:

```python
import heapq

def best_first_search(graph, start, goal):


    """We create a priority queue to use as our queue for the search.
     We add the starting node to the queue with a priority of 0."""
    queue = [(0, start)]


    """We create a set to keep track of the nodes we have visited so far.
     We add the starting node to the set."""
    visited = set()
    visited.add(start)


    # Loop through the queue until it is empty
    """We enter a loop that continues until the queue is empty. In each iteration of the loop,
```

```python
        we get the node with the lowest cost so far from the queue (using the heappop() method).
        If the node is the goal, we return the cost of the path to the goal."""
    while queue:
        cost, node = heapq.heappop(queue)
        if node == goal:
            return cost
        # Visit all the adjacent nodes
        """We loop through all the adjacent nodes of the current node.
        If a neighbor has not been visited yet,
        we add it to the queue with its cost as the priority and mark it as visited."""
        for neighbor, weight in graph[node]:
            if neighbor not in visited:
                # Add the neighbor to the queue with its estimated cost as the priority
                heapq.heappush(queue, (cost + weight, neighbor))
                visited.add(neighbor)

    # If we reach this point, there is no path from the start to the goal
    return None


graph = {
    'A': [('B', 2), ('C', 3)],
    'B': [('D', 1), ('E', 5)],
    'C': [('F', 7)],
    'D': [('F', 2)],
    'E': [('F', 3)],
    'F': []
}
cost = best_first_search(graph, 'A', 'F')
print(cost)
```
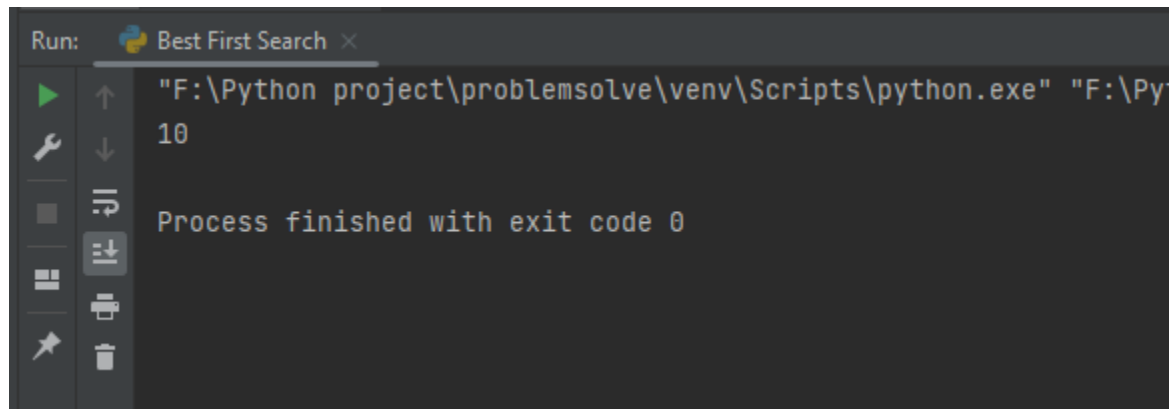
Output:

```
Run:    Best First Search ×
    "F:\Python project\problemsolve\venv\Scripts\python.exe" "F:\Py
    10

    Process finished with exit code 0
```