INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG

Lab report-5

Course code: CSE-3636

Course Titlle : Artificial Intiligance Lab

Submitted To:

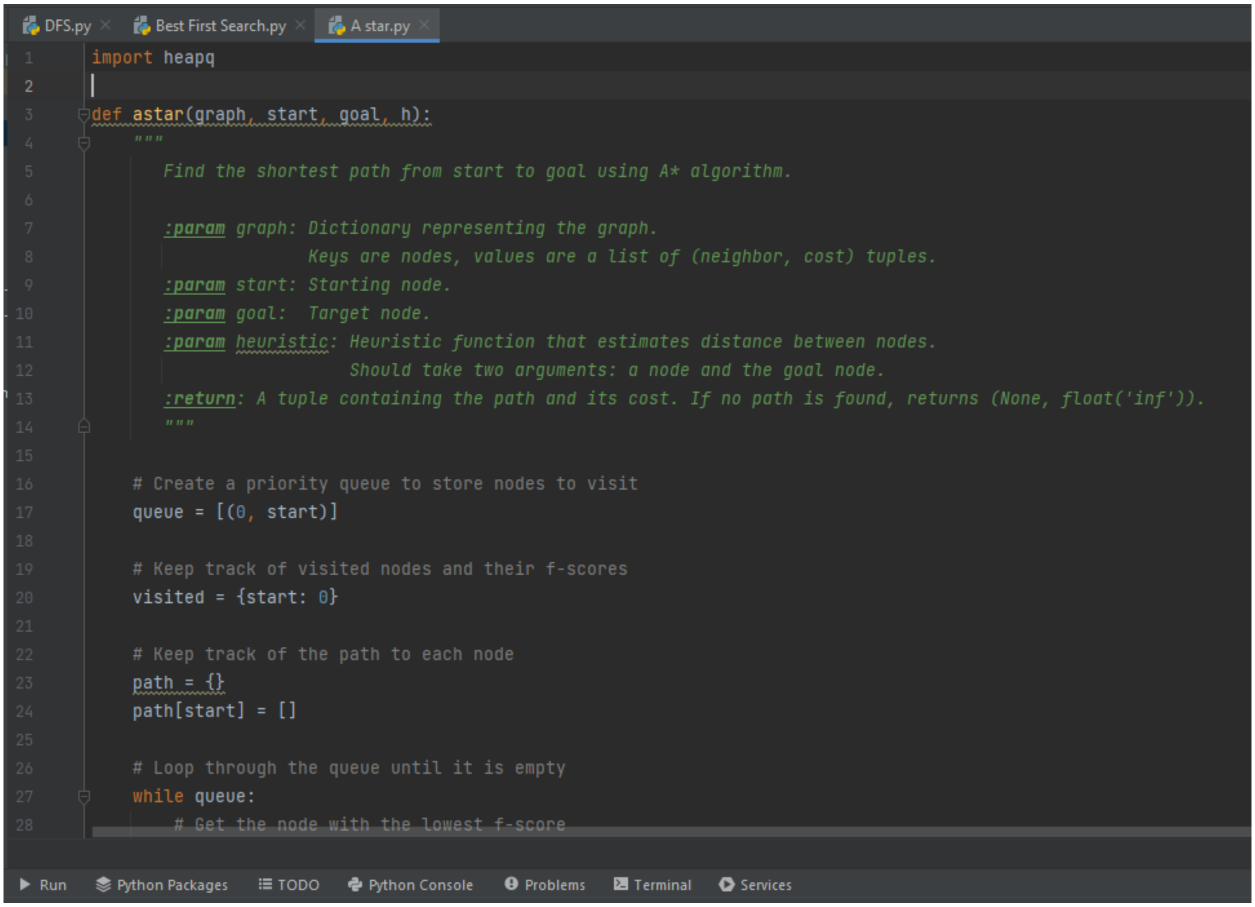
Md Safayet Hossen
Department of CSE
International Islamic University Chittagong

Submitted By:

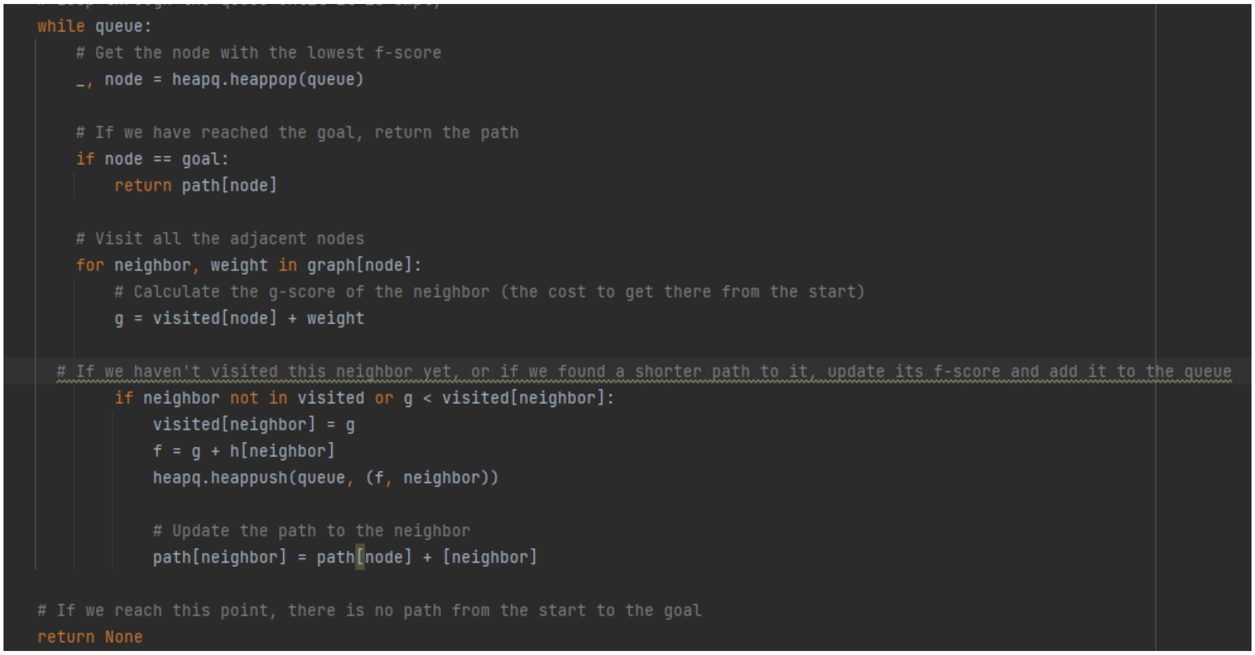
Md.Riaz Ahmed
Id: C201060
Section: 6BM
Semester: 6th

Date of submission : 25/03/23

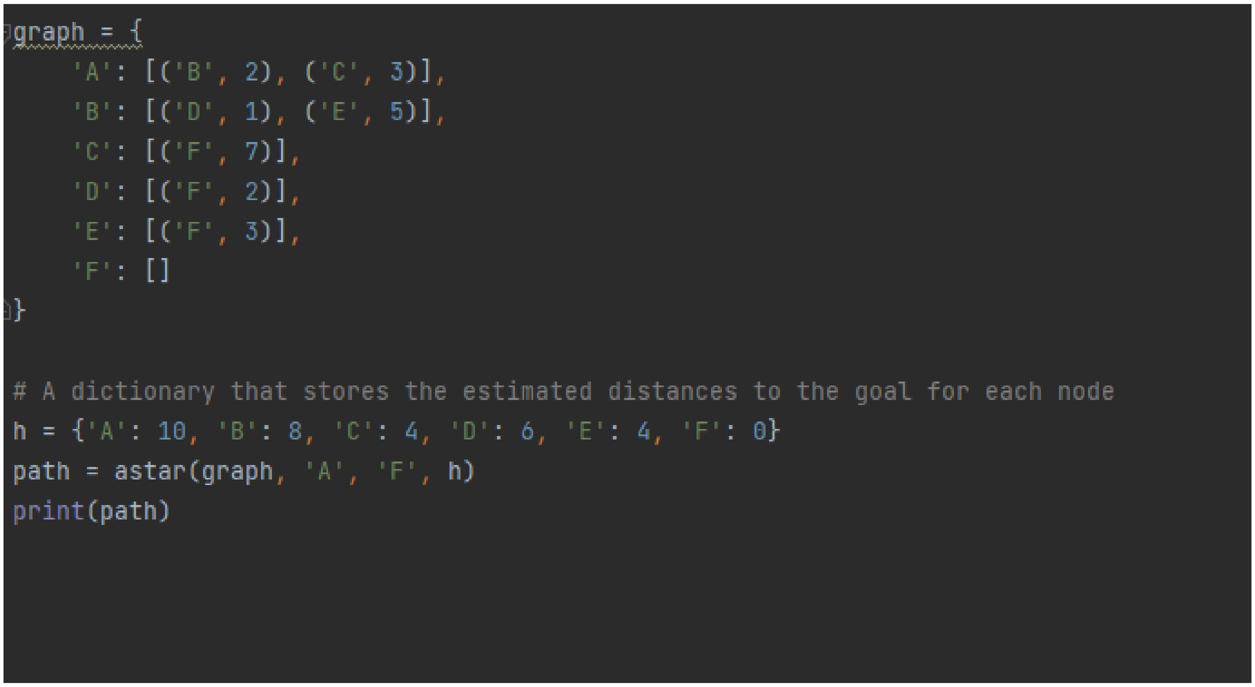
Lab report -5 : A*



```python
import heapq

def astar(graph, start, goal, h):
    """
       Find the shortest path from start to goal using A* algorithm.

       :param graph: Dictionary representing the graph.
                     Keys are nodes, values are a list of (neighbor, cost) tuples.
       :param start: Starting node.
       :param goal:  Target node.
       :param heuristic: Heuristic function that estimates distance between nodes.
                         Should take two arguments: a node and the goal node.
       :return: A tuple containing the path and its cost. If no path is found, returns (None, float('inf')).
       """

    # Create a priority queue to store nodes to visit
    queue = [(0, start)]

    # Keep track of visited nodes and their f-scores
    visited = {start: 0}

    # Keep track of the path to each node
    path = {}
    path[start] = []

    # Loop through the queue until it is empty
    while queue:
        # Get the node with the lowest f-score
```



```python
    while queue:
        # Get the node with the lowest f-score
        _, node = heapq.heappop(queue)

        # If we have reached the goal, return the path
        if node == goal:
            return path[node]

        # Visit all the adjacent nodes
        for neighbor, weight in graph[node]:
            # Calculate the g-score of the neighbor (the cost to get there from the start)
            g = visited[node] + weight

      # If we haven't visited this neighbor yet, or if we found a shorter path to it, update its f-score and add it to the queue
            if neighbor not in visited or g < visited[neighbor]:
                visited[neighbor] = g
                f = g + h[neighbor]
                heapq.heappush(queue, (f, neighbor))

                # Update the path to the neighbor
                path[neighbor] = path[node] + [neighbor]

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



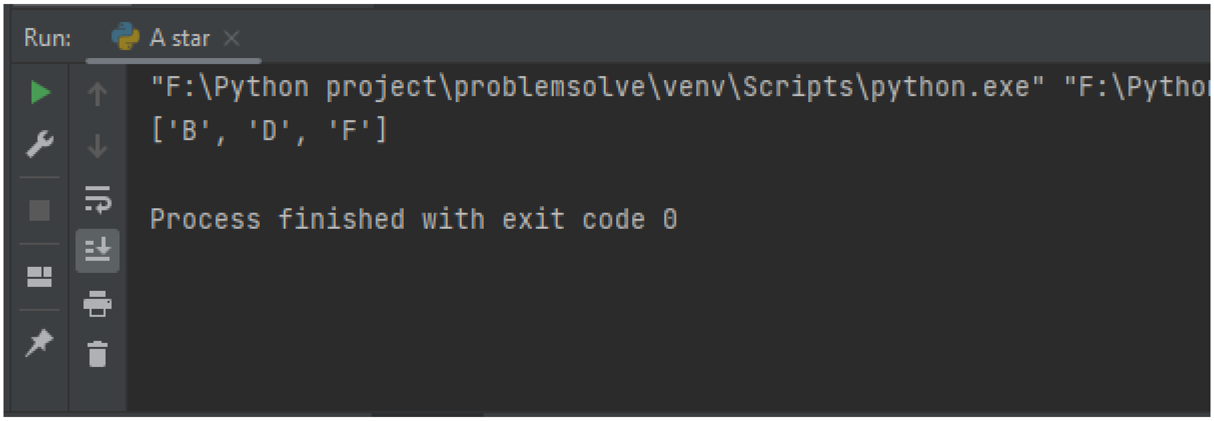
```
graph = {
    'A': [('B', 2), ('C', 3)],
    'B': [('D', 1), ('E', 5)],
    'C': [('F', 7)],
    'D': [('F', 2)],
    'E': [('F', 3)],
    'F': []
}

# A dictionary that stores the estimated distances to the goal for each node
h = {'A': 10, 'B': 8, 'C': 4, 'D': 6, 'E': 4, 'F': 0}
path = astar(graph, 'A', 'F', h)
print(path)
```

Code:

```
import heapq


def astar(graph, start, goal, h):
  """
    Find the shortest path from start to goal using A* algorithm.


    :param graph: Dictionary representing the graph.
              Keys are nodes, values are a list of (neighbor, cost) tuples.
    :param start: Starting node.
    :param goal:  Target node.
    :param heuristic: Heuristic function that estimates distance between nodes.
                Should take two arguments: a node and the goal node.
    :return: A tuple containing the path and its cost. If no path is found, returns (None, float('inf')).
    """

  # Create a priority queue to store nodes to visit
  queue = [(0, start)]

  # Keep track of visited nodes and their f-scores
  visited = {start: 0}

  # Keep track of the path to each node
  path = {}
  path[start] = []

  # Loop through the queue until it is empty
  while queue:
      # Get the node with the lowest f-score
      _, node = heapq.heappop(queue)

      # If we have reached the goal, return the path
      if node == goal:
          return path[node]

      # Visit all the adjacent nodes
      for neighbor, weight in graph[node]:
          # Calculate the g-score of the neighbor (the cost to get there from the start)
          g = visited[node] + weight

    # If we haven't visited this neighbor yet, or if we found a shorter path to it, update its f-score and add it to the queue
          if neighbor not in visited or g < visited[neighbor]:
              visited[neighbor] = g
              f = g + h[neighbor]
              heapq.heappush(queue, (f, neighbor))

              # Update the path to the neighbor
              path[neighbor] = path[node] + [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': []
}

# A dictionary that stores the estimated distances to the goal for each node
h = {'A': 10, 'B': 8, 'C': 4, 'D': 6, 'E': 4, 'F': 0}
path = astar(graph, 'A', 'F', h)
```

```
print(path)
```

Output:



```
"F:\Python project\problemsolve\venv\Scripts\python.exe" "F:\Pytho
['B', 'D', 'F']

Process finished with exit code 0
```