
# Ahsanullah University of Science & Technology

## Department of Computer Science & Engineering

**Course No.** : CSE 4108
**Course Name** : Artificial Intelligence Lab
**Assignment No.** : 05

**Submitted To :**

Md. Siam Ansary
Department of CSE, AUST

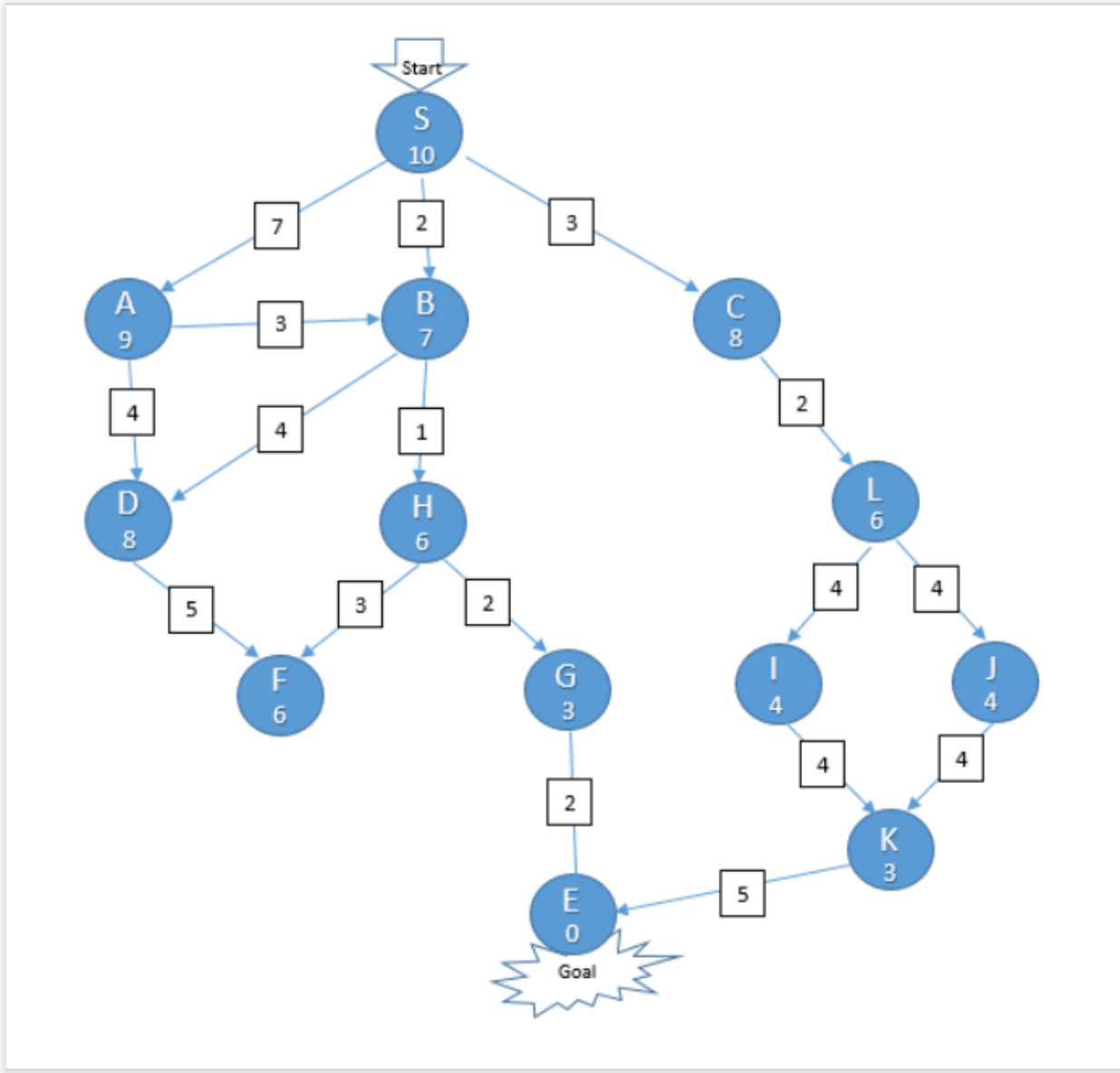
Mr. Ashek Seum
Department of CSE, AUST

**Submitted By:**

Name : Sanjida Akter Ishita
ID No. : 170204089
Session : Fall 2020
Section : B
Lab group : B2

**Date of Submission:** September 21, 2021

**QUESTION 01:** For the graph below, implement Greedy Best First Search Algorithm and A* Search Algorithm using Python.



**Answer:** The two variants of Best First Search are **Greedy Best First Search** and **A* Best First Search**. The Greedy BFS algorithm selects the path which appears to be the best, it can be known as the combination of depth-first search and breadth-first search. Greedy BFS makes use of Heuristic function and search and allows us to take advantages of both algorithms.

**A* Algorithm** in **Python** or in general is basically an artificial intelligence problem used for the path finding and the Graph traversals. This **algorithm** is flexible and can be used in a wide range of contexts.

The **A* search algorithm** uses the heuristic path cost, the starting point's cost, and the ending point.

## Implementation of Greedy Best First Search Algorithm using Python

```
import copy

h = {'S': 10, 'A': 9, 'B': 7, 'C': 8, 'D': 8, 'H': 6, 'L': 6, 'F': 6, 'G': 3, 'I': 4, 'J': 4, 'K': 3, 'E': 0}

traversalPath = {'S': ['A', 'B', 'C'], 'A': ['B', 'D'], 'B': {'H', 'D'}, 'C': ['L'], 'D': ['F'], 'H': ['F', 'G'],
                 'L': ['I', 'J'], 'G': ['E'], 'I': ['K'], 'J': ['K'], 'K': ['E']}

def bestFirstSearch(start, final):
      path = []
      t = []
      priorityQueue = [[[start], h[start]]]
      visited = []
      while priorityQueue != []:
         t = priorityQueue.copy()
         path.append(priorityQueue.pop(0))
         node = path[-1][0][-1]
         visited.append(node)
         print(t)

         if node == final:
           finalPath = copy.deepcopy(path[-1])
           print("Final Path", finalPath[0:1])
           return "Found"
         for neighbor in traversalPath[node]:
            if neighbor not in visited:
                newPath = copy.deepcopy(path[-1])
                newPath[0].append(neighbor)
                newPath[1] = h[neighbor]
                priorityQueue.append(newPath)
```

```
        priorityQueue.sort(key=lambda x: x[1])
        print("Visited", visited)


bestFirstSearch('S', 'E')
```

# Output:

```
H:\Python\Python39\python.exe "G:/4.1/LAB/AI Lab/Assignment/Assignment05/170204089.py"
[['S'], 10]]
Visited ['S']
[['S', 'B'], 7], [['S', 'C'], 8], [['S', 'A'], 9]]
Visited ['S', 'B']
[['S', 'B', 'H'], 6], [['S', 'C'], 8], [['S', 'B', 'D'], 8], [['S', 'A'], 9]]
Visited ['S', 'B', 'H']
[['S', 'B', 'H', 'G'], 3], [['S', 'B', 'H', 'F'], 6], [['S', 'C'], 8], [['S', 'B', 'D'], 8], [['S', 'A'], 9]]
Visited ['S', 'B', 'H', 'G']
[['S', 'B', 'H', 'G', 'E'], 0], [['S', 'B', 'H', 'F'], 6], [['S', 'C'], 8], [['S', 'B', 'D'], 8], [['S', 'A'], 9]]
Final Path [['S', 'B', 'H', 'G', 'E']]

Process finished with exit code 0
```

# Implementation of A* Search Algorithm using Python

```python
class Graph:
    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list

    def get_neighbors(self, v):
        return self.adjacency_list[v]

    def h(self, n):
      H = {
          'S': 10,
          'A': 9,
          'B': 7,
          'C': 8,
          'D': 8,
          'H': 6,
          'L': 6,
          'F': 6,
          'G': 3,
          'I': 4,
          'J': 4,
          'K': 3,
          'E': 0
      }
      return H[n]

    def a_star_algorithm(self, start_node, stop_node):
       open_list = set([start_node])
       closed_list = set([])
```

```
    closed_list = set([])
    li = []
    g = {}
    g[start_node] = 0
    parents = {}
    parents[start_node] = start_node

    while len(open_list) > 0:
        n = None

        for v in open_list:
            if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                n = v

        if n == None:
            print('Path does not exist!')
            return None

        if n == stop_node:
            li.append(n)
            print("Visited: ", li)
            reconst_path = []


            while parents[n] != n:
              reconst_path.append(n)
              n = parents[n]
```

```
        reconst_path.append(start_node)


        reconst_path.reverse()


        print('Final Path: {}'.format(reconst_path))

        return reconst_path


    for (m, weight) in self.get_neighbors(n):
         if m not in open_list and m not in closed_list:
                open_list.add(m)
                parents[m] = n
                g[m] = g[n] + weight
         else:
           if g[m] > g[n] + weight:
            g[m] = g[n] + weight
            parents[m] = n


            if m in closed_list:
               closed_list.remove(m)
               open_list.add(m)
```

```
            li.append(n)
            print("Visited: ", li)
            open_list.remove(n)
            closed_list.add(n)


        print('Path does not exist!')
        return None


adjacency_list = {

 'S': [('A', 7), ('B', 2), ('C', 3)],

 'A': [('B', 3), ('D', 4)],

 'B': [('D', 4), ('H', 1)],

 'C': [('L', 2)],

 'D': [('F', 5)],

 'H': [('F', 3), ('G', 2)],

 'L': [('I', 4), ('J', 4)],

 'F': [],
```

```
 'G': [('E', 2)],

 'I': [('K', 4)],

 'J': [('K', 4)],

  'K': [('E', 5)],
   'E': []
}

graph1 = Graph(adjacency_list)

graph1.a_star_algorithm('S', 'E')
```

## Output:

```
H:\Python\Python39\python.exe "G:/4.1/LAB/AI Lab/Assignment/Assignment05/170204089_2.py"
Visited:  ['S']
Visited:  ['S', 'B']
Visited:  ['S', 'B', 'H']
Visited:  ['S', 'B', 'H', 'G']
Visited:  ['S', 'B', 'H', 'G', 'E']
Final Path: ['S', 'B', 'H', 'G', 'E']

Process finished with exit code 0
```

## Analyzing Code and Output:

In greedy best first search, we used priority queue and sorted them according to the value of h(n) as f(n) = h(n). And for A* search, we took both the value of h(n) and g(n). The only difference between Greedy BFS and A* is in the evaluation function. For Greedy BFS the evaluation function is f(n) = h(n) while for A* the evaluation function is f(n) = g(n) + h(n).