**Experiment 4**

**AIM - Implement Greedy Best First Search / A\* search algorithm in Python.**

**Code: Greedy Best First Search –**

graph = {

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

'A': [('C', 4), ('D', 1)],

'B': [('E', 3), ('F', 1)],

'E': [('H', 5)],

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

'C': [('A', 4)],

'D': [('A', 1)],

'E': [('B', 3)],

'I': [('F', 2)],

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

}

dist = {}

h = {

'S': 13, 'A': 12, 'B': 4, 'C': 7, 'D': 3, 'E': 8, 'F': 2, 'H': 4, 'I': 9, 'G': 0

}

def bestFit(start, target):

q = []

q.append((h[start], start))

for x in graph:

dist[x] = 10000

dist[start] = 0

while q:

print()

q = sorted(q)

curr = q.pop(0)

print(curr)

curr\_dist = dist[curr[1]]

for node in graph[curr[1]]:

if curr\_dist+node[1] < dist[node[0]]:

print(node, end=" ")

dist[node[0]] = curr\_dist + node[1]

q.append((h[node[0]], node[0]))

print(dist)

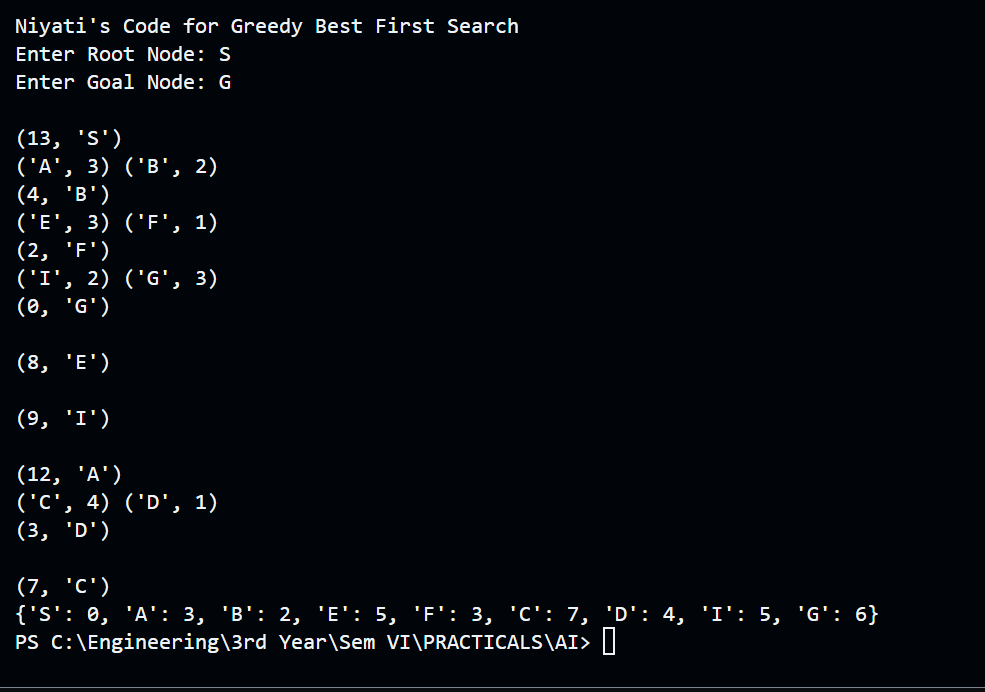
print("Niyati's Code for Greedy Best First Search")

root\_node = input("Enter Root Node: ")

goal\_node = input("Enter Goal Node: ")

bestFit(root\_node, goal\_node)

**Output –**



**Code: A\* Search –**

import heapq

class Node:

def \_\_init\_\_(self, name, cost=0, parent=None):

self.name = name

self.cost = cost

self.parent = parent

self.priority = 0

def \_\_lt\_\_(self, other):

return self.priority < other.priority

def heuristic(node, goal, heuristic\_map):

return heuristic\_map.get(node.name, 0)

def a\_star\_search(start, goal, graph, heuristic\_map):

open\_list = []

heapq.heappush(open\_list, (0, Node(start)))

visited = set()

while open\_list:

\_, current\_node = heapq.heappop(open\_list)

if current\_node.name in visited:

continue

visited.add(current\_node.name)

if current\_node.name == goal:

path = []

while current\_node:

path.append((current\_node.name, current\_node.priority)) # Include priority in the result

current\_node = current\_node.parent

return path[::-1]

for neighbor, cost in graph[current\_node.name].items():

if neighbor not in visited:

neighbor\_node = Node(neighbor, current\_node.cost + cost, current\_node)

neighbor\_node.priority = neighbor\_node.cost + heuristic(neighbor\_node, goal, heuristic\_map)

heapq.heappush(open\_list, (neighbor\_node.priority, neighbor\_node))

return None

# Graph data

graph = {

'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},

'Bucharest': {'Urziceni': 85, 'Pitesti': 101, 'Giurgiu': 90, 'Fagaras': 211},

'Craiova': {'Drobeta': 120, 'Rimnicu': 146, 'Pitesti': 138},

'Drobeta': {'Mehadia': 75, 'Craiova': 120},

'Eforie': {'Hirsova': 86},

'Fagaras': {'Sibiu': 99, 'Bucharest': 211},

'Giurgiu': {'Bucharest': 90},

'Hirsova': {'Urziceni': 98, 'Eforie': 86},

'Iasi': {'Neamt': 87, 'Vaslui': 92},

'Lugoj': {'Timisoara': 111, 'Mehadia': 70},

'Mehadia': {'Lugoj': 70, 'Drobeta': 75},

'Neamt': {'Iasi': 87},

'Oradea': {'Zerind': 71, 'Sibiu': 151},

'Pitesti': {'Rimnicu': 97, 'Craiova': 138, 'Bucharest': 101},

'Rimnicu': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},

'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu': 80},

'Timisoara': {'Arad': 118, 'Lugoj': 111},

'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},

'Vaslui': {'Iasi': 92, 'Urziceni': 142},

'Zerind': {'Arad': 75, 'Oradea': 71}

}

# Heuristic values

heuristic\_map = {

'Arad': 366,

'Bucharest': 0,

'Craiova': 160,

'Drobeta': 242,

'Eforie': 161,

'Fagaras': 176,

'Giurgiu': 77,

'Hirsova': 151,

'Iasi': 226,

'Lugoj': 244,

'Mehadia': 241,

'Neamt': 234,

'Oradea': 380,

'Pitesti': 100,

'Rimnicu': 193,

'Sibiu': 253,

'Timisoara': 329,

'Urziceni': 80,

'Vaslui': 199,

'Zerind': 374

}

cities = [

"Arad",

"Bucharest",

"Craiova",

"Dobreta",

"Eforie",

"Fagaras",

"Giurgiu",

"Hirsova",

"Iasi",

"Lugoj",

"Mehadia",

"Neamt",

"Oradea",

"Pitesti",

"Rimnicu\_Vilcea",

"Sibiu",

"Timisoara",

"Urziceni",

"Vaslui",

"Zerind"

]

print("Niyati's Code for A\* algorithm")

for i in range(len(cities)):

print(f"{i}. {cities[i]}")

start\_city = int(input("Enter no. for Start City: "))

start\_city = cities[start\_city]

goal\_city = int(input("Enter no. Goal City: "))

goal\_city = cities[goal\_city]

path = a\_star\_search(start\_city, goal\_city, graph, heuristic\_map)

print(f"Path from {start\_city} to {goal\_city}:", path)

**Output –**

