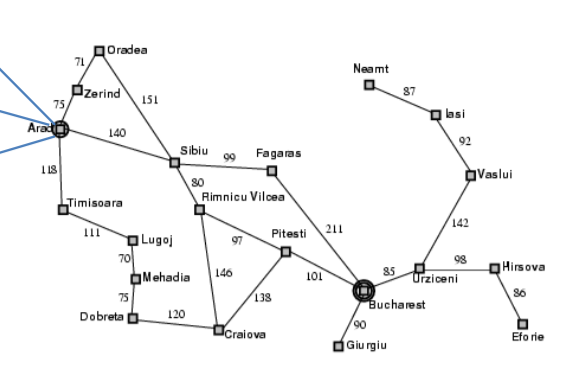
**NAME: Deepa Kumari**

**ROLL NO.: 1814019**

**BATCH: A1**

**DATE: 11/09/2020**

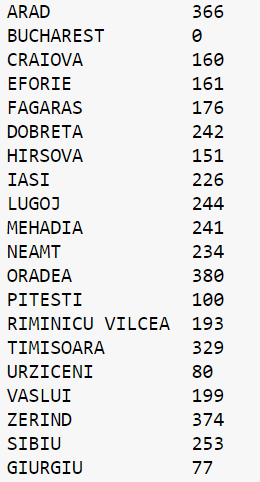
**1. MAP CHOSEN:**

****

**2. START CITY:** ARAD

**END CITY:** BUCHAREST

**HEURISTIC VALUES:**

****

**3. CODE:**

**class Graph:**

**def \_\_init\_\_(self, graph\_dict=None, directed=True):**

**self.graph\_dict = graph\_dict or {}**

**self.directed = directed**

**if not directed:**

**self.make\_undirected()**

**def make\_undirected(self):**

**for a in list(self.graph\_dict.keys()):**

**for (b, dist) in self.graph\_dict[a].items():**

**self.graph\_dict.setdefault(b, {})[a] = dist**

**def connect(self, A, B, distance=1):**

**self.graph\_dict.setdefault(A, {})[B] = distance**

**if not self.directed:**

**self.graph\_dict.setdefault(B, {})[A] = distance**

**def get(self, a, b=None):**

**links = self.graph\_dict.setdefault(a, {})**

**if b is None:**

**return links**

**else:**

**return links.get(b)**

**def nodes(self):**

**s1 = set([k for k in self.graph\_dict.keys()])**

**s2 = set([k2 for v in self.graph\_dict.values() for k2, v2 in v.items()])**

**nodes = s1.union(s2)**

**return list(nodes)**

**class Node:**

**def \_\_init\_\_(self, name:str, parent:str):**

**self.name = name**

**self.parent = parent**

**self.g = 0**

**self.h = 0**

**self.f = 0**

**def \_\_eq\_\_(self, other):**

**return self.name == other.name**

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

**return self.f < other.f**

**def \_\_repr\_\_(self):**

**return ('({0},{1})'.format(self.name, self.f))**

**def astar\_search(graph, heuristics, start, end):**

**open = []**

**closed = []**

**start\_node = Node(start, None)**

**goal\_node = Node(end, None)**

**open.append(start\_node)**

**while len(open) > 0:**

**open.sort()**

**current\_node = open.pop(0)**

**closed.append(current\_node)**

**if current\_node == goal\_node:**

**path = []**

**while current\_node != start\_node:**

**path.append(current\_node.name + ': ' + str(current\_node.f))**

**current\_node = current\_node.parent**

**path.append(start\_node.name + ': ' + str(start\_node.f))**

**return path[::-1]**

**neighbors = graph.get(current\_node.name)**

**for key, value in neighbors.items():**

**neighbor = Node(key, current\_node)**

**if(neighbor in closed):**

**continue**

**neighbor.g = current\_node.g + graph.get(current\_node.name, neighbor.name)**

**neighbor.h = heuristics.get(neighbor.name)**

**neighbor.f = neighbor.g + neighbor.h**

**if(add\_to\_open(open, neighbor) == True):**

**open.append(neighbor)**

**return None**

**def add\_to\_open(open, neighbor):**

**for node in open:**

**if (neighbor == node and neighbor.f > node.f):**

**return False**

**return True**

**graph = Graph()**

**n=int(input("Enter number of edges\n"))**

**inp=0**

**print("Enter graph edges in order of start to end index and its path cost")**

**while inp!=n:**

**start=input()**

**end=input()**

**cost=int(input())**

**graph.connect(start,end,cost)**

**inp+=1**

**graph.make\_undirected()**

**heuristics = {}**

**nodes=int(input("Enter number of nodes: "))**

**for i in range(nodes):**

**node=input()**

**heuristic=int(input())**

**heuristics[node]=heuristic**

**start=input("Enter start node")**

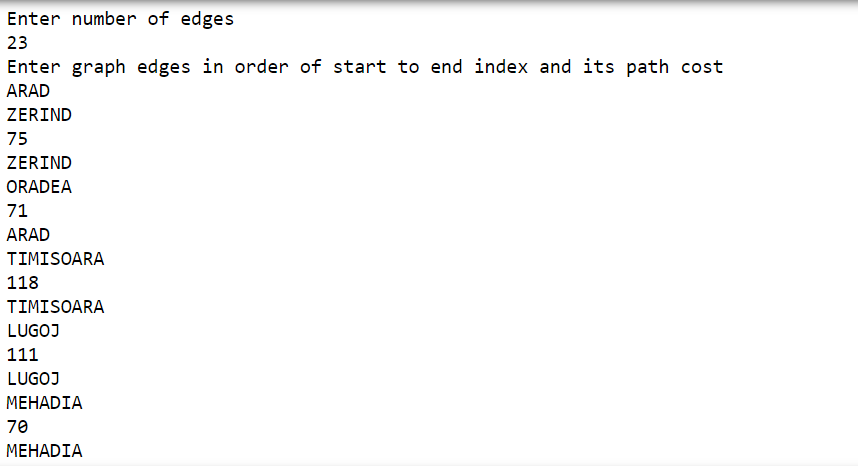
**end=input("Enter end node")**

**path = astar\_search(graph, heuristics, start, end)**

**print(path)**

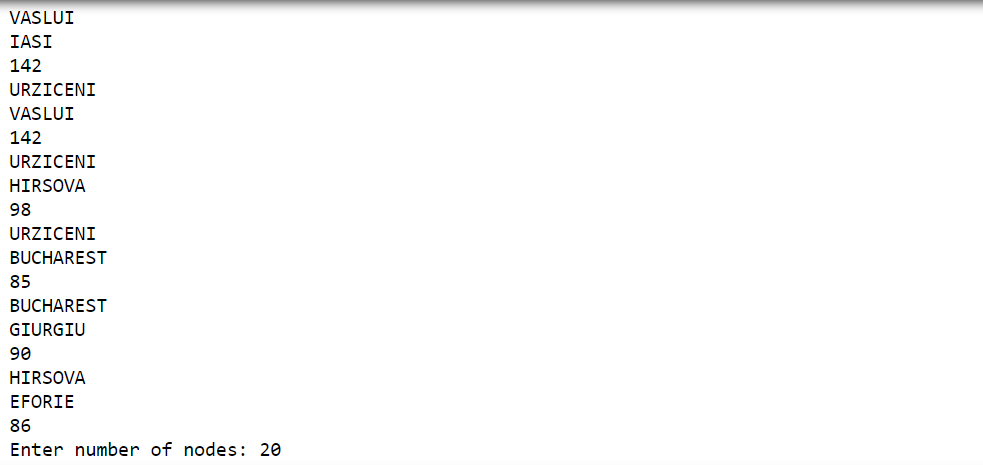
**print("Path cost", path[-1])**

**4. OUTPUT:**

****

****

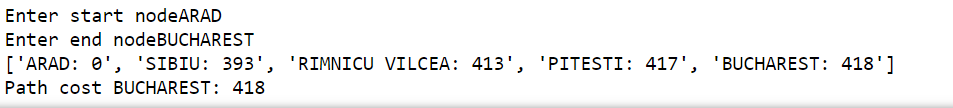
****

****

****

****

****

****