1. We have the Map of Romania. In this map, the distance between various places in Romania is given. If we have to reach from one place to another place there exist several paths. Write a Python Program to find the shortest distance between any two places using a A* search algorithm.
   USN - 1NT18CS133

Algorithm: // A* Search Algorithm

1. Initialize the open list
2. Initialize the closed list put the starting node on the open list (you can leave its f at zero)
3. while the open list is not empty a) find the node with the least f on the open list, call it "q"

   b) pop q off the open list

   c) generate q's 8 successors and set their parents to q d) for each successor

```
i) if successor is the goal, stop search
  successor.g = q.g + distance between
                      successor and q
  successor.h = distance from goal to
  successor (This can be done using many
  ways, we will discuss three heuristics-
  Manhattan, Diagonal and Euclidean
  Heuristics)

  successor.f = successor.g + successor.h

ii) if a node with the same position as
      successor is in the OPEN list which has a
     lower f than successor, skip this successor

   iii) if a node with the same position as
       successor is in the CLOSED list which has
       a lower f than successor, skip this successor
       otherwise, add the node to the open list
```

end (for loop)

e) push q on the closed list end (while loop)

In [1]:
```
class PQueue():
    def __init__(self):
        self.dict = {}
        self.keys = []
        self.sorted = False

    def _sort(self):
        self.keys = sorted(self.dict, key=self.dict.get, reverse=True)
        self.sorted = True

    def push(self, k, v):
        self.dict[k] = v
        self.sorted = False

    def pop(self):
        try:
            if not self.sorted:
                self._sort()
            key = self.keys.pop()
            value = self.dict[key]
            self.dict.pop(key)
            return key, value
        except:
            return None

def heuristics(path):
    h = {}
    with open(path, 'r') as file:
        for line in file:
            k, v = line.split(", ")
            h[k] = int(v)
            #print(h)
    return h

def path_costs(path):
    c = {}
    with open(path, 'r') as file:
        for line in file:
            line = line.split(", ")
            v = int(line.pop())
            e1 = line.pop()
            e2 = line.pop()
            if e1 not in c:
                c[e1] = {}
            if e2 not in c:
                c[e2] = {}
            c[e1][e2] = c[e2][e1] = v
            #print(c)
    return c

def a_star(start, goal, h, g):
    frontier = PQueue()
    # pushing path and cost to pqueue
    frontier.push(start, h[start])
    while True:
        # poping path with least cost
        path, cost = frontier.pop()
        print(path+ " " +str(cost))
        # splitting out end node in path
        end = path.split("->")[-1]
        # removing heuristic value of end node from cost
        cost -= h[end]
        if goal == end:
            break
        for node, weight in g[end].items():
            # adding edge weight(cost) and node heuristic to total cost
            new_cost = cost + weight + h[node]
            new_path = path + "->" + node
            # adding new path and cost to pqueue
            frontier.push(new_path, new_cost)


a_star('Arad', 'Bucharest', heuristics('./heuristics.txt'), path_costs('./paths.txt'))
```

```
Arad 366
Arad->Sibiu 393
Arad->Sibiu->Rimnicu Vilcea 413
Arad->Sibiu->Fagaras 415
Arad->Sibiu->Rimnicu Vilcea->Pitesti 417
Arad->Sibiu->Rimnicu Vilcea->Pitesti->Bucharest 418
```

In [ ]: