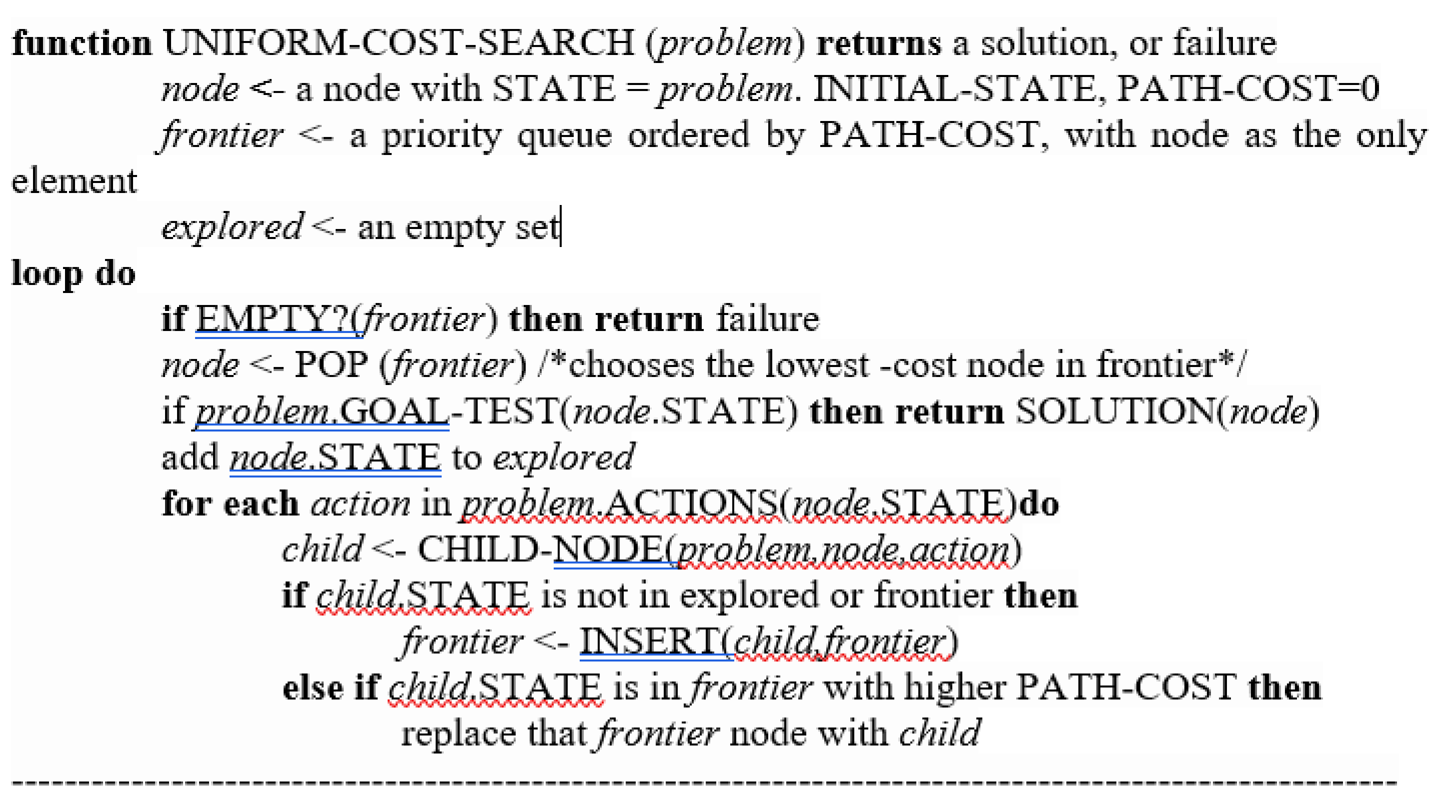
1. Problem Statement for uniform cost search: For the Romania map, the distance between various places are given. If we have to reach from one place to another place there exist several paths. Write a Python Program to find the shortest distance between any two places using a uniform cost search.
   USN - 1NT18CS133

Algorithm: Uniform-Cost Search is similar to Dijikstra's algorithm .
In this algorithm from the starting state we will visit the adjacent states and will choose the least costly state then we will choose the next least costly state from the all un-visited and adjacent states of the visited states, in this way we will try to reach the goal state (note we wont continue the path through a goal state ), even if we reach the goal state we will continue searching for other possible paths( if there are multiple goals) . We will keep a priority queue which will give the least costliest next state from all the adjacent states of visited states.



```
function UNIFORM-COST-SEARCH (problem) returns a solution, or failure
        node <- a node with STATE = problem. INITIAL-STATE, PATH-COST=0
        frontier <- a priority queue ordered by PATH-COST, with node as the only
element
        explored <- an empty set
loop do
        if EMPTY?(frontier) then return failure
        node <- POP (frontier) /*chooses the lowest -cost node in frontier*/
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE)do
                child <- CHILD-NODE(problem,node,action)
                if child.STATE is not in explored or frontier then
                        frontier <- INSERT(child,frontier)
                else if child.STATE is in frontier with higher PATH-COST then
                        replace that frontier node with child
-----------------------------------------------------------------------------------------------------
```

In [1]:

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

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

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

    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 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
    return c

def ucs(start, goal, g):
    frontier = PQueue()
    # pushing path and cost to pqueue
    frontier.push(start, 0)
    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]
        if goal == end:
            break
        for node, weight in g[end].items():
            # adding edge weight(cost) to total cost
            new_cost = cost + weight
            new_path = path + "->" + node
            # adding new path and cost to pqueue
            frontier.push(new_path, new_cost)

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

```
Arad 0
Arad->Zerind 75
Arad->Timisoara 118
Arad->Sibiu 140
Arad->Zerind->Oradea 146
Arad->Zerind->Arad 150
Arad->Zerind->Oradea->Zerind 217
Arad->Sibiu->Rimnicu Vilcea 220
Arad->Zerind->Arad->Zerind 225
Arad->Timisoara->Lugoj 229
Arad->Timisoara->Arad 236
Arad->Sibiu->Fagaras 239
Arad->Zerind->Arad->Timisoara 268
Arad->Sibiu->Arad 280
Arad->Zerind->Oradea->Zerind->Oradea 288
Arad->Zerind->Arad->Sibiu 290
Arad->Sibiu->Oradea 291
Arad->Zerind->Oradea->Zerind->Arad 292
Arad->Zerind->Arad->Zerind->Oradea 296
Arad->Zerind->Oradea->Sibiu 297
Arad->Timisoara->Lugoj->Mehadia 299
Arad->Zerind->Arad->Zerind->Arad 300
Arad->Sibiu->Rimnicu Vilcea->Sibiu 300
Arad->Timisoara->Arad->Zerind 311
Arad->Sibiu->Rimnicu Vilcea->Pitesti 317
Arad->Sibiu->Fagaras->Sibiu 338
Arad->Timisoara->Lugoj->Timisoara 340
Arad->Timisoara->Arad->Timisoara 354
Arad->Sibiu->Arad->Zerind 355
Arad->Zerind->Oradea->Zerind->Oradea->Zerind 359
Arad->Sibiu->Oradea->Zerind 362
Arad->Sibiu->Rimnicu Vilcea->Craiova 366
Arad->Zerind->Arad->Zerind->Oradea->Zerind 367
Arad->Zerind->Oradea->Zerind->Arad->Zerind 367
Arad->Timisoara->Lugoj->Mehadia->Lugoj 369
Arad->Zerind->Arad->Sibiu->Rimnicu Vilcea 370
Arad->Timisoara->Lugoj->Mehadia->Dobreta 374
Arad->Zerind->Arad->Zerind->Arad->Zerind 375
Arad->Timisoara->Arad->Sibiu 376
Arad->Zerind->Oradea->Sibiu->Rimnicu Vilcea 377
Arad->Zerind->Arad->Timisoara->Lugoj 379
Arad->Sibiu->Rimnicu Vilcea->Sibiu->Rimnicu Vilcea 380
Arad->Timisoara->Arad->Zerind->Oradea 382
Arad->Timisoara->Arad->Zerind->Arad 386
Arad->Zerind->Arad->Timisoara->Arad 386
Arad->Zerind->Arad->Sibiu->Fagaras 389
Arad->Zerind->Oradea->Sibiu->Fagaras 396
Arad->Sibiu->Arad->Timisoara 398
Arad->Sibiu->Rimnicu Vilcea->Sibiu->Fagaras 399
Arad->Zerind->Oradea->Zerind->Arad->Timisoara 410
Arad->Sibiu->Rimnicu Vilcea->Pitesti->Rimnicu Vilcea 414
Arad->Sibiu->Fagaras->Sibiu->Rimnicu Vilcea 418
Arad->Sibiu->Rimnicu Vilcea->Pitesti->Bucharest 418
```

In [ ]: