NMTTNT_Tuan3_19110413

November 27, 2021

1 Tên : Vòng Vĩnh Phú

2 MSSV : 19110413

[ ]:
```
graph = {
  "Arad": {
    "Zerind": { "cost": 75 },
    "Timisoara": { "cost": 118 },
    "Sibiu": { "cost": 140 }
  },
  "Zerind": {
    "Oradea": { "cost": 71 },
    "Arad": { "cost": 75 }
  },
  "Timisoara": {
    "Lugoj": { "cost": 111 },
    "Arad": { "cost": 118 }
  },
  "Sibiu": {
    "Fagaras": { "cost": 99 },
    "Rimnicu Vilcea": { "cost": 80 },
    "Oradea": { "cost": 151 },
    "Arad": { "cost": 140 }
  },
  "Oradea": {
    "Sibiu": { "cost": 151 },
    "Zerind": { "cost": 71 }
  },
  "Lugoj": {
    "Mehadia": { "cost": 70 },
    "Timisoara": { "cost": 111 }
  },
  "Fagaras": {
    "Bucharest": { "cost": 211 },
    "Sibiu": { "cost": 99 }
  },
  "Rimnicu Vilcea": {
```

```
    "Pitesti": { "cost": 97 },
    "Craiova": { "cost": 146 },
    "Sibiu": { "cost": 99 }
  },
  "Mehadia": {
    "Drobeta": { "cost": 75 },
    "Lugoj": { "cost": 70 }
  },
  "Pitesti": {
    "Bucharest": { "cost": 101 },
    "Rimnicu Vilcea": { "cost": 97 },
    "Craiova": { "cost": 138 }
  },
  "Craiova": {
    "Pitesti": { "cost": 138 },
    "Rimnicu Vilcea": { "cost": 146 },
    "Drobeta": { "cost": 120 }
  },
  "Drobeta": {
    "Craiova": { "cost": 120 },
    "Mehadia": { "cost": 75 }
  },
  "Bucharest": {
    "Fagaras": { "cost": 211 },
    "Pitesti": { "cost": 101 },
    "Giurgiu": { "cost": 90 },
    "Urziceni": { "cost": 85 }
  },
  "Giurgiu": {
    "Bucharest": { "cost": 90 }
  },
  "Urziceni": {
    "Bucharest": { "cost": 85 },
    "Hirsova": { "cost": 98 },
    "Vaslui": { "cost": 142 }
  },
  "Hirsova": {
    "Eforie": { "cost": 70 },
    "Urziceni": { "cost": 98 }
  },
  "Vaslui": {
    "Iasi": { "cost": 92 },
    "Urziceni": { "cost": 142 }
  },
  "Iasi": {
    "Neamt": { "cost": 87 },
    "Vaslui": { "cost": 92 }
```

```
  },
  "Neamt": {
    "Iasi": { "cost": 87 }
  }
}

heuristic = {
  "Arad": 366,
  "Bucharest": 20,
  "Craiova": 160,
  "Drobeta": 242,
  "Eforie": 161,
  "Fagaras": 176,
  "Giurgiu": 77,
  "Hirsova": 0,
  "Iasi":226,
  "Lugoj": 244,
  "Mehadia": 241,
  "Neamt": 234,
  "Oradea": 380,
  "Pitesti": 100,
  "Rimnicu Vilcea": 193,
  "Sibiu": 253,
  "Timisoara": 329,
  "Urziceni": 10,
  "Vaslui": 199,
  "Zerind": 374,
}
Start = 'Arad'
Goal = 'Hirsova'
```

[ ]:
```
def Quicksort(array,Map, sortBy): # QuickSort (python version)
    less = []
    equal = []
    greater = []
    if len(array) > 1:
        pivot =Map[array[0]][sortBy]
        for city in array:
            cost =Map[city][sortBy]
            if cost < pivot:
                less.append(city)
            if cost == pivot:
                equal.append(city)
            if cost > pivot:
                greater.append(city)
        return Quicksort(less,Map, sortBy) + equal +␣
↪Quicksort(greater,Map,sortBy) # Merge array
```

```
    else:
        return array
```

[ ]:
```
def GBFS(graph,hn,start, goal):
  queue = []
  queue.append(start)
  # create map between 2 city with heuristic function dictionary
  map = []
  for city in graph.keys():
    map.append((city, {'from': None, 'heuristic': hn[city]}))
  map = dict(map)
  while True:
    curCity = queue.pop(0) # dequeue the city have least heuristic cost
    if curCity == goal: #path found
      # trace back the path
      path = [curCity]
      while curCity != start:
        curCity = map[curCity]['from']
        path.append(curCity)
      path.reverse()
      return path
    for city in graph[curCity].keys(): # explore the city
      if map[city]['from'] == None: # if city wasn't explored add it into the␣
↪map
        queue.append(city) # add city into queue
        map[city]['from'] = curCity # add city explore into the map
    if len(queue) == 0:
      raise Exception("No way Exception")
    queue = Quicksort(queue, map, 'heuristic') #sort by heuristic cost increase
```

[ ]:
```
greedy=GBFS(graph,heuristic,Start,Goal)
print("Greedy best first search solution from",Start,"to",Goal)
print(greedy)
```

```
Greedy best first search solution from Arad to Hirsova
['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Urziceni', 'Hirsova']
```

[ ]:
```
def AStar(graph,hn,start, goal):
  queue = []
  queue.append(start)
  # create map between 2 city with heuristic cost + path cost (F(n) = H(n)+␣
↪G(n)) dictionary
  map = []
  for city in graph.keys():
    map.append((city, {'from': None, 'total_cost': hn[city]}))
  map = dict(map)
  while True:
```

```
    curCity = queue.pop(0) # dequeue the city have least total cost
    curCityTotalCost = map[curCity]['total_cost'] - hn[curCity] # cost from␣
↪start to current
    if curCity == goal: #path found
      # trace back the path
      path = [curCity]
      cost = map[curCity]['total_cost']
      while curCity != start:
        curCity = map[curCity]['from']
        path.append(curCity)
      path.reverse()
      return path,cost
    for city in graph[curCity].keys(): # explore the city
      cityTotalCost = map[city]['total_cost']
      # total cost from start to city explore
      totalCost = graph[curCity][city]['cost'] + curCityTotalCost + hn[city]
      # if city wasn't explored or have the better way add it into the map
      if map[city]['from'] == None or totalCost < cityTotalCost :
        if queue.count(city) != 0: # if city in open/close set (queue) delete it
          queue.remove(city)
        queue.append(city) # add it ino queue
        map[city]['from'] =  curCity # add city explore into the map/update new␣
↪path
        map[city]['total_cost'] = totalCost # update new total cost path
    if len(queue) == 0:
      raise Exception("No way Exception")
    queue=Quicksort(queue,map,'total_cost') #sort by total cost increase
```

[ ]:
```
AstarPath,cost=AStar(graph,heuristic,Start,Goal)
print("A* solution from",Start,"to",Goal)
print(AstarPath,"Total cost = ",cost)
```

```
A* solution from Arad to Hirsova
['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest', 'Urziceni',
'Hirsova'] Total cost =  601
```