Group No. - 26

Group Members:

1. Samarjoy Pandit
2. Saurabh Datir
3. Vishal Gauba
4. Saksham Rastogi
5. Akanksha Jojwan

Problem Statement:

There is a thief who has to visit the given `n` cities in a car and pick up from total items `m`. The distance between each city is given in distance matrix `d`. The items have an associated profit of `p_k` and each item has weight `w_k`. The maximum carrying capacity of the car is `W`. Thief cannot visit a city more than once. Find out the path that the thief should take in order to maximise his profit and minimise the time taken to reach back to the original city.

- Total distance

$$D = \sum_{i=0,j=0}^{n,n} \pi(i,j) * d_{i,j}$$

- Total Profit

$$P = \sum_{i=0}^{n} \theta(i) * d_i$$

- Total Time

  i = `city travelling from`

  j = `city travelling to`

$$T = \pi(i,j) * (\sum_{i=0,j=0}^{n,n} d(i,j)/v + \sum \theta(k) * t(k))$$

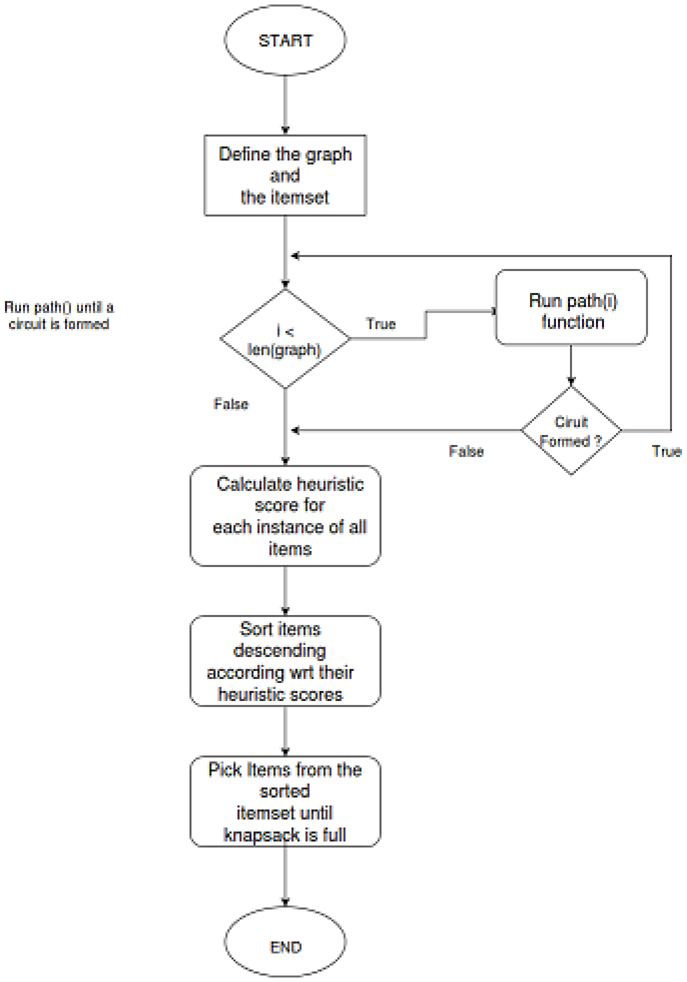
- Weight

$$\sum w_k <= W$$

Assumptions:

- Can pick same item from multiple cities
- Cannot visit a city more than once
- If knapsack is full, may come back to origin city

Flowchart



Screenshot

```
Path taken by thief: [1, 7, 6, 5, 2, 4, 3, 1]
Distance covered: 290
Items picked:
        City: 1         Items: 1, 2, 3, 4, 5, 6, 7
        City: 2         Items: 3, 5, 7
        City: 4         Items: 4
        City: 5         Items: 1, 2, 3
Total Weight: 600
Total Profit: 6150
```

Source Code

```python
"""
@authors:
    Samarjoy Pandit
    Saurabh Datir
    Vishal Gauba
    Saksham Rastogi
    Akanksha Jojwan
"""

# An adjacency list representation of the graph of cities
Graph = {
            1:[(2, 45), (3, 30), (4, 35), (5, 20), (6, 80), (7, 50)],
            2:[(1, 25), (4, 50), (5, 25), (6, 30), (7, 45)],
            3:[(1, 30), (4, 30), (6, 65)],
            4:[(1, 55), (2, 30), (3, 50), (6, 60), (7, 25)],
            5:[(1, 20), (2, 35), (6, 20)],
            6:[(1, 80), (2, 30), (3, 65), (4, 50), (5, 20), (7, 35)],
            7:[(1, 30), (2, 15), (4, 25), (6, 55)]
        }

# The itemsets as key-value pairs. In the 'value', the first and second elements of the list is the profit and weight resp. The third is the tuple of cities in which the item is available
ItemSet = {

            1:[ 400, 40, (1, 2, 3, 4, 5, 6, 7)],
            2:[ 500, 35, (3, 5, 7)],
            3:[ 700, 200, (1, 2, 4, 6)],
            4:[ 200, 20, (4, 5, 6, 7)],
            5:[ 550, 65, (1, 2, 3)]
}

# We maintain global variables to maintain our state during graph traversal for shortest path calculation
Visited = [0]*len(Graph)
ShortestPath = []
PathLen = []
VMax = 20
VMin = 10
R = 0.1
W = 600

def GetPath(index):

    global Graph, Visited, PathLen, ShortestPath

    ShortestPath = ShortestPath + [index,]

    if Visited != [1]*len(Graph):
        Visited[index-1] = 1
        min = float("inf")
        small = ()

```

```
        # find the unvisited city with minimum distance for ith node in the graph and recursively call path() on that node
        for e in Graph[index]:
            if Visited[e[0] - 1] == 0 and e[1] < min:
                small = e

        if small != ():
            GetPath(small[0])
            PathLen = PathLen + [small[1]+ (PathLen[-1] if len(PathLen) > 0 else 0) , ]
        # else if all cities from the ith node are visited, then check if we can return to node 1 from the ith node
        else:
            if ShortestPath[0] in [E[0] for E in Graph[index]]:
                ShortestPath = ShortestPath + [ShortestPath[0],]
                for E in Graph[index]:
                    if E[0] == ShortestPath[0]:
                        PathLen = PathLen + [E[1] + (PathLen[-1] if len(PathLen) > 0 else 0), ]
            return
    else:
        return

for j in range(1, len(Graph) + 1):
    GetPath(j)
    # if the thief returns to his origin city then break
    if ShortestPath[0] == ShortestPath[-1]:
        break

# CityDistance contains the distance of the ith city along the path from the end of the circuit
CityDistance = [PathLen[-1] + PathLen[0] - c for c in PathLen]
ThiefBag = []
# Time is calculated assuming average velocity throughout the tour.
Time = 2*PathLen[-1]*(VMax + VMin)

for item,value in ItemSet.items():
    for k in value[2]:
        # Here 'k' is the cities in which the item is present
        profit = int(value[0] - (0.25*value[0]*(CityDistance[ShortestPath.index(k)]/PathLen[-1])) - (R*Time*value[1]/W))
        ThiefBag = ThiefBag + [[item, k, value[1], profit, value[0]],]

#We sort the items by their profit, and then keep picking till the knapsack's weight limit is reached
ThiefBag.sort(key = lambda x: int(x[3]))
# Here, we need the items sorted descending by their heuristic profit
ThiefBag.reverse()

c = 0
i = 0
ItemsPicked = []
TotalProfit = 0

```

```python
# Calculating total profit based on the items picked by the thief
while i < len(ThiefBag):
    if c + ThiefBag[i][2] <= W:
        ItemsPicked = ItemsPicked + [[ThiefBag[i][0], ThiefBag[i][1]],]
        c = c + ThiefBag[i][2]
        TotalProfit = TotalProfit + ThiefBag[i][4]

    i = i + 1

ItemsPicked.sort(key = lambda x: int(x[0]))
values = sorted(set(map(lambda x:x[0], ItemsPicked)))

Arr = []

for i in values:
    templist = []
    for j in ItemsPicked:
        if j[0] == i:
            templist = templist + [j[1],]

    templist.sort()
    Arr = Arr + [[i, templist],]

print()
print("Path taken by thief: " + str(ShortestPath))
print("Distance covered: " + str(PathLen[-1]))
print("Items picked: ")
for i in Arr:
    print("\tCity: " + str(i[0]) + "\t\tItems: " + ', '.join(str(e) for e in i[1]))

print("Total Weight: " + str(c))
print("Total Profit: " + str(TotalProfit))
```