LAB-9

Coin Change Problem

CODE:

```
def count(S, m, n):
    table = [[0 for x in range(m)] for x in range(n+1)]
    for i in range(m):
        table[0][i] = 1
    for i in range(1, n+1):
        for j in range(m):
            x = table[i - S[j]][j] if i-S[j] >= 0 else 0
            y = table[i][j-1] if j >= 1 else 0
            table[i][j] = x + y
    return table[n][m-1]
arr = [1, 2, 3]
m = len(arr)
n = 4
print(count(arr, m, n))
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/coin change.py
4
>>>
```

Knapsack Problem

CODE:

```
def knapSack(W, wt, val, n):
    if n == 0 or W == 0:
        return 0
    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)
    else:
        return max(
            val[n-1] + knapSack(
```

```
                                W-wt[n-1], wt, val, n-1),
                        knapSack(W, wt, val, n-1))
if __name__ == '__main__':
        profit = [60, 100, 120]
        weight = [10, 20, 30]
        W = 50
        n = len(profit)
        print(knapSack(W, weight, profit, n))
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/knapsack using
 greedy.py
220
>>> |
```

Job Sequencing with Deadlines

CODE:

```
def printjobschedule(array, t):
  m = len(array)
  for j in range(m):
    for q in range(m - 1 - j):
      if array[q][2] < array[q + 1][2]:
        array[q], array[q + 1] = array[q + 1], array[q]
  res = [False] * t
  job = ['-1'] * t
  for q in range(len(array)):
    for q in range(min(t - 1, array[q][1] - 1), -1, -1):
      if res[q] is False:
        res[q] = True
        job[q] = array[q][0]
        break
  print(job)
array = [['a', 7, 202],
     ['b', 5, 29],
```

```
    ['c', 6, 84],
    ['d', 1, 75],
    ['e', 2, 43]]
print("Maximum profit sequence of jobs is- ")
printjobschedule(array, 3)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/job sequence.p
y
Maximum profit sequence of jobs is-
['a', 'c', 'd']
>>>
```

Single Source Shortest Paths: Dijkstra's Algorithm

CODE:

```
class Graph():
        def __init__(self, vertices):
                self.V = vertices
                self.graph = [[0 for column in range(vertices)]
                                                for row in range(vertices)]
        def printSolution(self, dist):
                print("Vertex \t Distance from Source")
                for node in range(self.V):
                        print(node, "\t\t", dist[node])
        def minDistance(self, dist, sptSet):
                min = 1e7
                for v in range(self.V):
                        if dist[v] < min and sptSet[v] == False:
                                min = dist[v]
                                min_index = v
                return min_index
        def dijkstra(self, src):
                dist = [1e7] * self.V
                dist[src] = 0
                sptSet = [False] * self.V
```

```
        for cout in range(self.V):

            u = self.minDistance(dist, sptSet)

            sptSet[u] = True

            for v in range(self.V):

                if (self.graph[u][v] > 0 and

                sptSet[v] == False and

                dist[v] > dist[u] + self.graph[u][v]):

                    dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

g = Graph(9)

g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],

        [4, 0, 8, 0, 0, 0, 0, 11, 0],

        [0, 8, 0, 7, 0, 4, 0, 0, 2],

        [0, 0, 7, 0, 9, 14, 0, 0, 0],

        [0, 0, 0, 9, 0, 10, 0, 0, 0],

        [0, 0, 4, 14, 10, 0, 2, 0, 0],

        [0, 0, 0, 0, 0, 2, 0, 1, 6],

        [8, 11, 0, 0, 0, 0, 1, 0, 7],

        [0, 0, 2, 0, 0, 0, 6, 7, 0]

        ]

g.dijkstra(0)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/dijkstra's alg
orithm single source shortest path.py
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14
>>>
```

Optimal Tree Problem: Huffman Trees and Codes

CODE:

```
import heapq
class Node:
    def __init__(self, symbol=None, frequency=None):
        self.symbol = symbol
        self.frequency = frequency
        self.left = None
        self.right = None
    def __lt__(self, other):
        return self.frequency < other.frequency
def build_huffman_tree(chars, freq):
    priority_queue = [Node(char, f) for char, f in zip(chars, freq)]
    heapq.heapify(priority_queue)
    while len(priority_queue) > 1:
        left_child = heapq.heappop(priority_queue)
        right_child = heapq.heappop(priority_queue)
        merged_node = Node(frequency=left_child.frequency + right_child.frequency)
        merged_node.left = left_child
        merged_node.right = right_child
        heapq.heappush(priority_queue, merged_node)
    return priority_queue[0]
def generate_huffman_codes(node, code="", huffman_codes={}):
    if node is not None:
        if node.symbol is not None:
            huffman_codes[node.symbol] = code
        generate_huffman_codes(node.left, code + "0", huffman_codes)
        generate_huffman_codes(node.right, code + "1", huffman_codes)
    return huffman_codes
chars = ['a', 'b', 'c', 'd', 'e', 'f']
freq = [4, 7, 15, 17, 22, 42]
root = build_huffman_tree(chars, freq)
huffman_codes = generate_huffman_codes(root)
```

```
for char, code in huffman_codes.items():
    print(f"Character: {char}, Code: {code}")
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/huffman code.p
y
Character: f, Code: 0
Character: a, Code: 1000
Character: b, Code: 1001
Character: c, Code: 101
Character: d, Code: 110
Character: e, Code: 111
>>>
```

Container Loading

CODE:

```
cont = [[ 0 for i in range(1000)]
                    for j in range(1000)]
def num_of_containers(n, x):
        count = 0
        cont[1][1] = x
        for i in range(1, n + 1):
                for j in range(1, i + 1):
                        if (cont[i][j] >= 1):
                                count += 1
                                cont[i + 1][j] += (cont[i][j] - 1) / 2
                                cont[i + 1][j + 1] += (cont[i][j] - 1) / 2
        print(count)
n = 3
x = 5
num_of_containers(n, x)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/container load
ing.py
4
>>>
```

Kruskal's Algorithms

CODE:

```
class Graph:
```

```
def __init__(self, vertices):
    self.V = vertices
    self.graph = []
def addEdge(self, u, v, w):
    self.graph.append([u, v, w])
def find(self, parent, i):
    if parent[i] != i:
        parent[i] = self.find(parent, parent[i])
    return parent[i]
def union(self, parent, rank, x, y):
    if rank[x] < rank[y]:
        parent[x] = y
    elif rank[x] > rank[y]:
        parent[y] = x
    else:
        parent[y] = x
        rank[x] += 1
def KruskalMST(self):
    result = []
    i = 0
    e = 0
    self.graph = sorted(self.graph,
                        key=lambda item: item[2])
    parent = []
    rank = []
    for node in range(self.V):
        parent.append(node)
        rank.append(0)
    while e < self.V - 1:
        u, v, w = self.graph[i]
        i = i + 1
```

```
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.union(parent, rank, x, y)
        minimumCost = 0
        print("Edges in the constructed MST")
        for u, v, weight in result:
            minimumCost += weight
            print("%d -- %d == %d" % (u, v, weight))
        print("Minimum Spanning Tree", minimumCost)
if __name__ == '__main__':
    g = Graph(4)
    g.addEdge(0, 1, 10)
    g.addEdge(0, 2, 6)
    g.addEdge(0, 3, 5)
    g.addEdge(1, 3, 15)
    g.addEdge(2, 3, 4)
    g.KruskalMST()
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/kruskal's algo
rithm.py
Edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Spanning Tree 19
>>>
```

Prims Algorithm

CODE:

```
import heapq
class Graph:
    def __init__(self, V):
```

```
        self.V = V
        self.adj = [[] for _ in range(V)]
    def add_edge(self, u, v, w):
        self.adj[u].append((v, w))
        self.adj[v].append((u, w))
    def prim_mst(self):
        pq = []
        src = 0
        key = [float('inf')] * self.V
        parent = [-1] * self.V
        in_mst = [False] * self.V
        heapq.heappush(pq, (0, src))
        key[src] = 0
        while pq:
            u = heapq.heappop(pq)[1]
            if in_mst[u]:
                continue
            in_mst[u] = True
            for v, weight in self.adj[u]:
                if not in_mst[v] and key[v] > weight:
                    key[v] = weight
                    heapq.heappush(pq, (key[v], v))
                    parent[v] = u
        for i in range(1, self.V):
            print(f"{parent[i]} - {i}")
if __name__ == "__main__":
    V = 5
    g = Graph(V)
    (0, 1, 2), (0, 3, 6), (1, 2, 3), (1, 3, 8), (1, 4, 5), (2, 4, 7), (3, 4, 9)
    g.add_edge(0, 1, 2)
    g.add_edge(0, 3, 6)
```

```python
    g.add_edge(1, 2, 3)
    g.add_edge(1, 3, 8)
    g.add_edge(1, 4, 5)
    g.add_edge(2, 4, 7)
    g.add_edge(3, 4, 9)
    g.prim_mst()
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/prims algorith
m.py
0 - 1
1 - 2
0 - 3
1 - 4
>>>
```

Boruvka's Algorithm

CODE:

```python
from collections import defaultdict
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
```

```
        else:
            parent[yroot] = xroot
            rank[xroot] += 1
    def boruvkaMST(self):
        parent = []
        rank = []
        cheapest = []
        numTrees = self.V
        MSTweight = 0
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
            cheapest = [-1] * self.V
        while numTrees > 1:
            for i in range(len(self.graph)):
                u, v, w = self.graph[i]
                set1 = self.find(parent, u)
                set2 = self.find(parent, v)
                if set1 != set2:
                    if cheapest[set1] == -1 or cheapest[set1][2] > w:
                        cheapest[set1] = [u, v, w]
                    if cheapest[set2] == -1 or cheapest[set2][2] > w:
                        cheapest[set2] = [u, v, w]
            for node in range(self.V):
                if cheapest[node] != -1:
                    u, v, w = cheapest[node]
                    set1 = self.find(parent, u)
                    set2 = self.find(parent, v)
                    if set1 != set2:
                        MSTweight += w
                        self.union(parent, rank, set1, set2)
```

```
                print("Edge %d-%d with weight %d included in MST" %
                    (u, v, w))
                numTrees = numTrees - 1
        cheapest = [-1] * self.V
    print("Weight of MST is %d" % MSTweight)
g = Graph(4)
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)
g.boruvkaMST()
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/Boruvka's Algo
rithm.py
Edge 0-3 with weight 5 included in MST
Edge 0-1 with weight 10 included in MST
Edge 2-3 with weight 4 included in MST
Weight of MST is 19
>>>
```