LAB-8

1. Coin Change Problem

```
def coinChange(coins, amount):
    coins.sort(reverse=True)
    result = 0
    for coin in coins:
        count = amount // coin
        amount -= coin * count
        result += count
    return result
coins = [1, 5, 10, 25]
amount = 37
print("Minimum number of coins required:", coinChange(coins, amount))
```

Output

```
"C:\Users\jacin\PycharmProjects\DAA-Design analysis of algorithm\.venv\Scripts\python.exe" C:\Users\jacin\AppData\
Minimum number of coins required: 4

Process finished with exit code 0
```

2. Knapsack Problem

```
def fractionalKnapsack(W, arr, N):
    arr.sort(key=lambda x: x[0]/x[1], reverse=True)
    finalvalue = 0.0
    for i in range(N):
        if arr[i][1] <= W:
            W -= arr[i][1]
            finalvalue += arr[i][0]
        else:
            finalvalue += arr[i][0] * (W / arr[i][1])
            break
    return finalvalue
W = 50
arr = [[60, 10], [100, 20], [120, 30]]
N = len(arr)
print("Maximum value in the knapsack =", fractionalKnapsack(W, arr, N))
```

Output

```
"C:\Users\jacin\PycharmProjects\DAA-Design analysis of algorithm\.venv\Scripts\python.exe" C:\Users\jacin\Ap
Maximum value in the knapsack = 240.0

Process finished with exit code 0
```

3. Job Sequencing with Deadlines

```python
def job_sequencing(jobs):
    n = len(jobs)
    max_deadline = max(job[0] for job in jobs)
    jobs.sort(key=lambda x: x[1], reverse=True)
    dp = [[0] * (max_deadline + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, max_deadline + 1):
            if jobs[i - 1][0] > j:
                dp[i][j] = dp[i - 1][j]
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - jobs[i - 1][0]] + jobs[i - 1][1])
    seq = []
    i, j = n, max_deadline
    while i > 0 and j > 0:
        if dp[i][j]!= dp[i - 1][j]:
            seq.append(i - 1)
            j -= jobs[i - 1][0]
        i -= 1
    seq.reverse()
    return dp[n][max_deadline], seq
jobs = [[2, 100], [1, 19], [2, 27], [1, 25], [3, 15]]
max_profit, seq = job_sequencing(jobs)
print("Maximum profit:", max_profit)
print("Optimal sequence:", seq)
```

Output

```
"C:\Users\jacin\PycharmProjects\DAA-Design analysis of algorithm\.venv\Scripts\python.exe" "C:\Users\jacin\App
Maximum profit: 125
Optimal sequence: [0, 2]

Process finished with exit code 0
```

4. Single Source Shortest Paths: Dijkstra's Algorithm

```python
import heapq
iPair = tuple
1 usage
class Graph:
    def __init__(self, V: int): # Constructor
        self.V = V
        self.adj = [[] for _ in range(V)]
    14 usages
    def addEdge(self, u: int, v: int, w: int):
        self.adj[u].append((v, w))
        self.adj[v].append((u, w))
    1 usage
    def shortestPath(self, src: int):
        pq = []
        heapq.heappush( *args: pq, (0, src))
        dist = [float('inf')] * self.V
        dist[src] = 0
```

```python
        dist[src] = 0
        while pq:
            d, u = heapq.heappop(pq)
            for v, weight in self.adj[u]:
                if dist[v] > dist[u] + weight:
                    dist[v] = dist[u] + weight
                    heapq.heappush( *args: pq, (dist[v], v))
        for i in range(self.V):
            print(f"{i} \t\t {dist[i]}")
if __name__ == "__main__":
    V = 9
    g = Graph(V)
    g.addEdge( u: 0,  v: 1,  w: 4)
    g.addEdge( u: 0,  v: 7,  w: 8)
    g.addEdge( u: 1,  v: 2,  w: 8)
    g.addEdge( u: 1,  v: 7,  w: 11)
```

```
    g = Graph(V)
    g.addEdge( u: 0,  v: 1,  w: 4)
    g.addEdge( u: 0,  v: 7,  w: 8)
    g.addEdge( u: 1,  v: 2,  w: 8)
    g.addEdge( u: 1,  v: 7,  w: 11)
    g.addEdge( u: 2,  v: 3,  w: 7)
    g.addEdge( u: 2,  v: 8,  w: 2)
    g.addEdge( u: 2,  v: 5,  w: 4)
    g.addEdge( u: 3,  v: 4,  w: 9)
    g.addEdge( u: 3,  v: 5,  w: 14)
    g.addEdge( u: 4,  v: 5,  w: 10)
    g.addEdge( u: 5,  v: 6,  w: 2)
    g.addEdge( u: 6,  v: 7,  w: 1)
    g.addEdge( u: 6,  v: 8,  w: 6)
    g.addEdge( u: 7,  v: 8,  w: 7)
    g.shortestPath(0)
```

Output

```
C:\Users\vinot\PycharmProjects
0        0
1        4
2        12
3        19
4        21
5        11
6        9
7        8
8        14
```

5. Optimal Tree Problem: Huffman Trees and Codes

```python
from collections import Counter, deque
2 usages
class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None
    def __lt__(self, other):
        return self.freq < other.freq
1 usage
def build_tree(freq_list):
    if len(freq_list) == 1:
        return freq_list[0]
    freq_list = deque(sorted(freq_list))
    while len(freq_list) > 1:
        left_node = freq_list.popleft()
```

```python
        right_node = freq_list.popleft()
        internal_node = Node( char: None, left_node.freq + right_node.freq)
        internal_node.left = left_node
        internal_node.right = right_node
        freq_list.append(internal_node)
        freq_list = deque(sorted(freq_list))
    return freq_list[0]
1 usage
def build_huffman_tree(data):
    freq_list = [Node(ch, data.count(ch)) for ch in set(data)]
    return build_tree(freq_list)
3 usages
def generate_codes(node, code, code_dict):
    if node is None:
        return
```

```
    generate_codes(node.left, code + '0', code_dict)
    generate_codes(node.right, code + '1', code_dict)
1 usage
def huffman_encode(data):
    tree = build_huffman_tree(data)
    code_dict = {}
    generate_codes(tree, '', code_dict)
    encoded_data = ''.join([code_dict[ch] for ch in data])
    return encoded_data, tree
data = 'BCAADDDCCACACAC'
encoded_data, tree = huffman_encode(data)
print('Original data:', data)
print('Encoded data:', encoded_data)
```

Output

```
Original data: BCAADDDCCACACAC
Encoded data: 1000111110110110100110110110

Process finished with exit code 0
```

6. Container Loading

```
def container_loading(containers, items):
    containers.sort(reverse=True)
    items.sort(reverse=True)
    loaded_items = []
    for item in items:
        for container in containers:
            if item <= container:
                loaded_items.append(item)
                containers.remove(container)
                break
    return loaded_items
containers = [10, 20, 30, 40, 50]
items = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
loaded_items = container_loading(containers, items)
print("Loaded items:", loaded_items)
```

Output

```
Loaded items: [50, 40, 30, 20, 10]

Process finished with exit code 0
```

7.Minimum Spanning Tree

8.

```
1 usage
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    15 usages
    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])
    7 usages
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    1 usage
    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
```

Kruskal's Algorithms

```
1 usage
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    5 usages
    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])
    5 usages
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    1 usage
    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
```



```
        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
    1 usage
    def kruskal_mst(self):
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []
        for node in range(self.V):
```

```python
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i += 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                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
```

```python
        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)
g = Graph(4)
g.add_edge( u: 0,  v: 1,  w: 10)
g.add_edge( u: 0,  v: 2,  w: 6)
g.add_edge( u: 0,  v: 3,  w: 5)
g.add_edge( u: 1,  v: 3,  w: 15)
g.add_edge( u: 2,  v: 3,  w: 4)
g.kruskal_mst()
```

Output

```
C:\Users\vinot\PycharmProjects\pythonP
Edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Spanning Tree 19

Process finished with exit code 0
```

9. Prims Algorithm

```python
import sys
1 usage
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for _ in range(vertices)] for _ in range(vertices)]
    1 usage
    def print_mst(self, parent):
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(f"{parent[i]}-{i} \t{self.graph[i][parent[i]]}")
    1 usage
    def min_key(self, key, mst_set):
        min_val = sys.maxsize
        min_index = 0
        for v in range(self.V):
            if key[v] < min_val and mst_set[v] == False:
```

```python
                min_index = v
        return min_index
    1 usage
    def prim_mst(self):
        key = [sys.maxsize] * self.V
        parent = [None] * self.V
        key[0] = 0
        mst_set = [False] * self.V
        parent[0] = -1
        for _ in range(self.V):
            u = self.min_key(key, mst_set)
            mst_set[u] = True
            for v in range(self.V):
                if (
                    self.graph[u][v] > 0
                    and mst_set[v] == False
                    and key[v] > self.graph[u][v]
```

```python
                    and mst_set[v] == False
                    and key[v] > self.graph[u][v]
                ):
                    key[v] = self.graph[u][v]
                    parent[v] = u
        self.print_mst(parent)
g = Graph(5)
g.graph = [
    [0, 2, 0, 6, 0],
    [2, 0, 3, 8, 5],
    [0, 3, 0, 0, 7],
    [6, 8, 0, 0, 9],
    [0, 5, 7, 9, 0],
]
g.prim_mst()
```

Output

```
C:\Users\vinot\PycharmProjects\pythonProj
Edge    Weight
0-1     2
1-2     3
0-3     6
1-4     5

Process finished with exit code 0
```

10. Boruvka's Algorithm

```
1 usage
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    15 usages
    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])
    7 usages
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    1 usage
    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
```

```python
        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
    1 usage
    def boruvka_mst(self):
        parent = []
        rank = []
        cheapest = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
```

```python
            parent.append(node)
            rank.append(0)
            cheapest = [-1] * self.V
        num_trees = self.V
        mst_weight = 0
        print("---------Forming MST------------")
        while num_trees > 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):
```

```python
                    u, v, w = cheapest[node]
                    set1 = self.find(parent, u)
                    set2 = self.find(parent, v)
                    if set1 != set2:
                        mst_weight += w
                        self.union(parent, rank, set1, set2)
                        print("Added edge [" + str(u) + " - " + str(v) + "]\n"
                              + "Added weight: " + str(w) + "\n")
                        num_trees -= 1
            cheapest = [-1] * self.V
        print("-----------------------------------")
        print("The total weight of the minimal spanning tree is: " + str(mst_weight))
g = Graph(9)
g.add_edge( u: 0,  v: 1,  w: 4)
g.add_edge( u: 0,  v: 6,  w: 7)
g.add_edge( u: 1,  v: 6,  w: 11)
g.add_edge( u: 1,  v: 7,  w: 20)
```

```python
g.add_edge( u: 0,  v: 6,  w: 7)
g.add_edge( u: 1,  v: 6,  w: 11)
g.add_edge( u: 1,  v: 7,  w: 20)
g.add_edge( u: 1,  v: 2,  w: 9)
g.add_edge( u: 2,  v: 3,  w: 6)
g.add_edge( u: 2,  v: 4,  w: 2)
g.add_edge( u: 3,  v: 4,  w: 10)
g.add_edge( u: 3,  v: 5,  w: 5)
g.add_edge( u: 4,  v: 5,  w: 15)
g.add_edge( u: 4,  v: 7,  w: 1)
g.add_edge( u: 4,  v: 8,  w: 5)
g.add_edge( u: 5,  v: 8,  w: 12)
g.add_edge( u: 6,  v: 7,  w: 1)
g.add_edge( u: 7,  v: 8,  w: 3)
g.boruvka_mst()
```

OUTPUT:

```
C:\Users\vinot\PycharmProjects\pythonPr
---------Forming MST------------
Added edge [0 - 1]
Added weight: 4

Added edge [2 - 4]
Added weight: 2

Added edge [3 - 5]
Added weight: 5
```

```
Added edge [4 - 7]
Added weight: 1

Added edge [6 - 7]
Added weight: 1

Added edge [7 - 8]
Added weight: 3
```

```
Added edge [0 - 6]
Added weight: 7

Added edge [2 - 3]
Added weight: 6
```

```
----------------------------------
The total weight of the minimal spanning tree is: 29
```