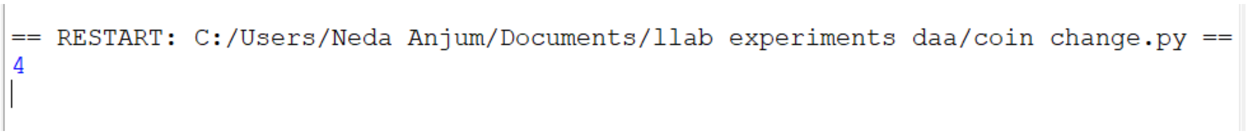
Lab9(14/6/24):

1.cion change problem

Code:

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

output:



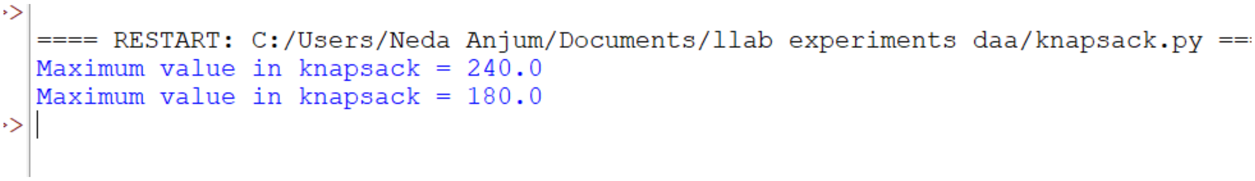
```
== RESTART: C:/Users/Neda Anjum/Documents/1lab experiments daa/coin change.py ==
4
```

2.knapsack problem

Code:

```
def knapsack(val, wei, W):
    n = len(val)
    ratios = [(val[i] / wei[i], val[i], wei[i]) for i in range(n)]
    ratios.sort(reverse=True)
     total_val = 0
    current_wei = 0
    for ratio, val, wei in ratios:
        if current_wei + wei <= W:
            total_val += val
            current_wei += wei
        else:
            fraction = (W - current_wei) / wei
            total_val += val * fraction
            break
 return total_val
val1 = [60, 100, 120]
wei1 = [10, 20, 30]
W1 = 50
print("Maximum value in knapsack =",knapsack(val1, wei1, W1))
val2 = [40, 100, 50, 60]
wei2 = [20, 10, 40, 30]
W2 = 50
print("Maximum value in knapsack =",knapsack(val2, wei2, W2))
```

output:



```
>
==== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/knapsack.py ==
Maximum value in knapsack = 240.0
Maximum value in knapsack = 180.0
>
```

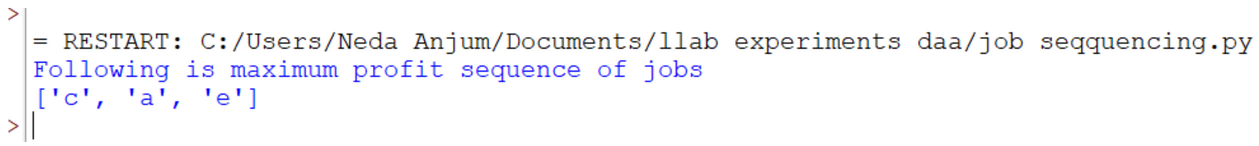
3.job sequencing with deadline

Code:

```
def JobSequencing(arr, t):
        n = len(arr)
        for i in range(n):
                for j in range(n - 1 - i):
                        if arr[j][2] < arr[j + 1][2]:
```

```
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    result = [False] * t
    job = ['-1'] * t
    for i in range(len(arr)):
        for j in range(min(t - 1, arr[i][1] - 1), -1, -1):
            if result[j] is False:
                result[j] = True
                job[j] = arr[i][0]
                break
    print(job)
if __name__ == '__main__':
    arr = [['a', 2, 100],
            ['b', 1, 19],
            ['c', 2, 27],
            ['d', 1, 25],
            ['e', 3, 15]]
    print("Following is maximum profit sequence of jobs")
    JobSequencing(arr, 3)
```

Output:



```
>
 = RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/job seqquencing.py
 Following is maximum profit sequence of jobs
 ['c', 'a', 'e']
>
```

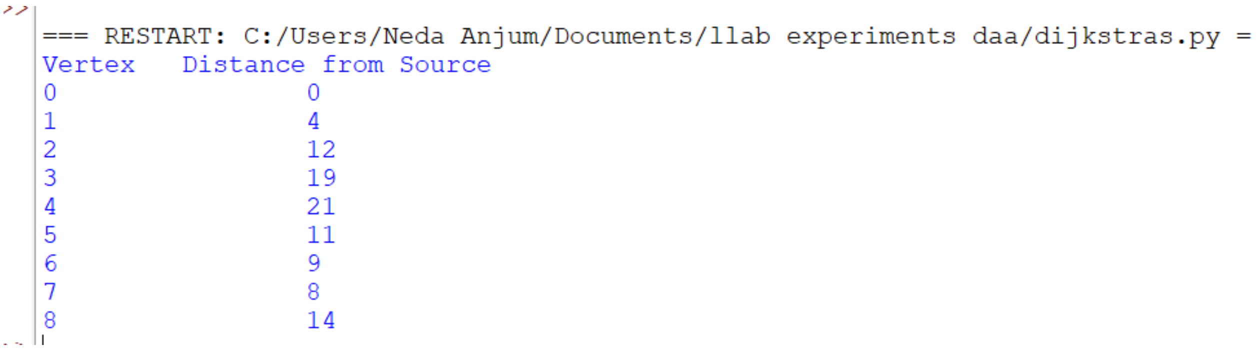
4.Dijkstras algorithm:

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:



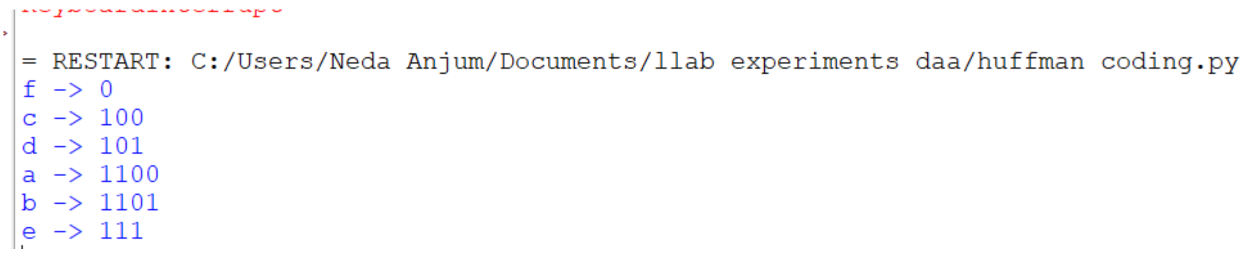
```
=== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/dijkstras.py =
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14
```

5.Huffman coding:

Code:

```
import heapq

class node:

        def __init__(self, freq, symbol, left=None, right=None):

                self.freq = freq

                self.symbol = symbol
        self.left = left


        self.right = right
        self.huff = ''


    def __lt__(self, nxt):
        return self.freq < nxt.freq
def printNodes(node, val=''):
    newVal = val + str(node.huff)
    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)
    if(not node.left and not node.right):
        print(f"{node.symbol} -> {newVal}")
chars = ['a', 'b', 'c', 'd', 'e', 'f']
freq = [5, 9, 12, 13, 16, 45]
nodes = []
for x in range(len(chars)):
    heapq.heappush(nodes, node(freq[x], chars[x]))


while len(nodes) > 1:
    left = heapq.heappop(nodes)
    right = heapq.heappop(nodes)
    left.huff = 0
    right.huff = 1
    newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)
    heapq.heappush(nodes, newNode)
printNodes(nodes[0])
```

output:



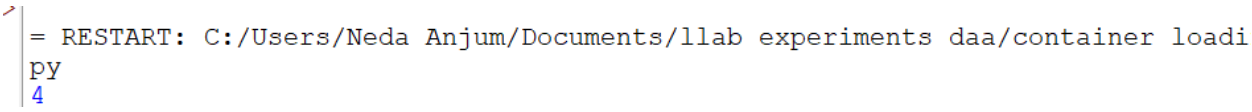
```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/huffman coding.py
f -> 0
c -> 100
d -> 101
a -> 1100
b -> 1101
e -> 111
```

6.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:



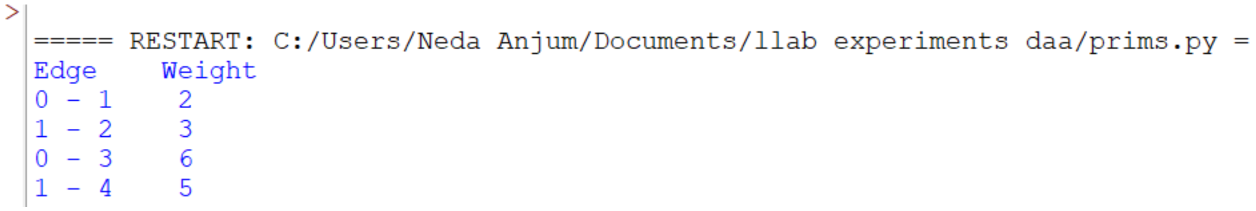
```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/container loadi
py
4
```

7.Minimum spanning tree(prims):

Code:

```
def printMST(self, parent):
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])
    def minKey(self, key, mstSet):
        min = sys.maxsize

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

        return min_index
    def primMST(self):
        key = [sys.maxsize] * self.V
        parent = [None] * self.V
        key[0] = 0
        mstSet = [False] * self.V

        parent[0] = -1
        for cout in range(self.V):
            u = self.minKey(key, mstSet)
            mstSet[u] = True
            for v in range(self.V):
                if self.graph[u][v] > 0 and mstSet[v] == False \
                and key[v] > self.graph[u][v]:
                    key[v] = self.graph[u][v]
                    parent[v] = u


        self.printMST(parent)

if __name__ == '__main__':

    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.primMST()
```

output:



```
>
===== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/prims.py =
Edge    Weight
0 - 1     2
1 - 2     3
0 - 3     6
1 - 4     5
```

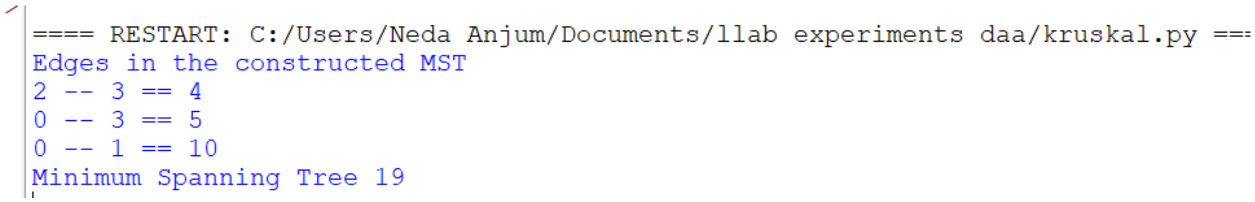
8.kruskal's algorithm:

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/Neda Anjum/Documents/llab experiments daa/kruskal.py ===
Edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Spanning Tree 19
```

9.Boruvka's algorithm:

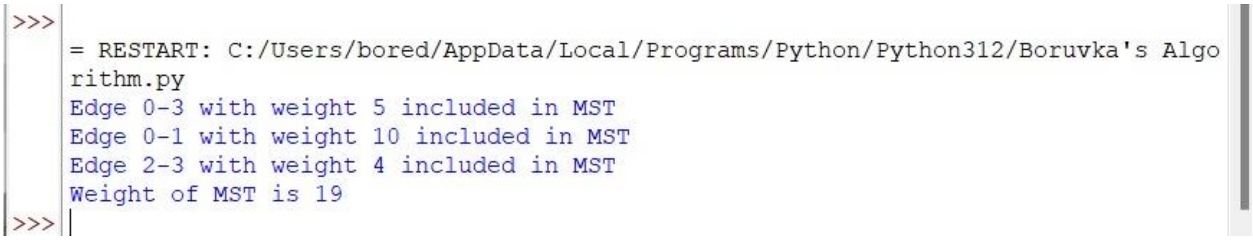
Code:

```
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
>>>
```