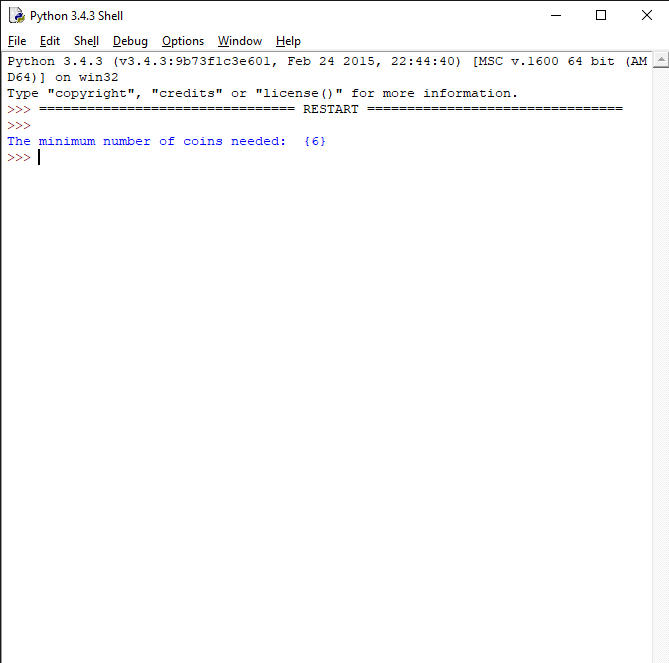
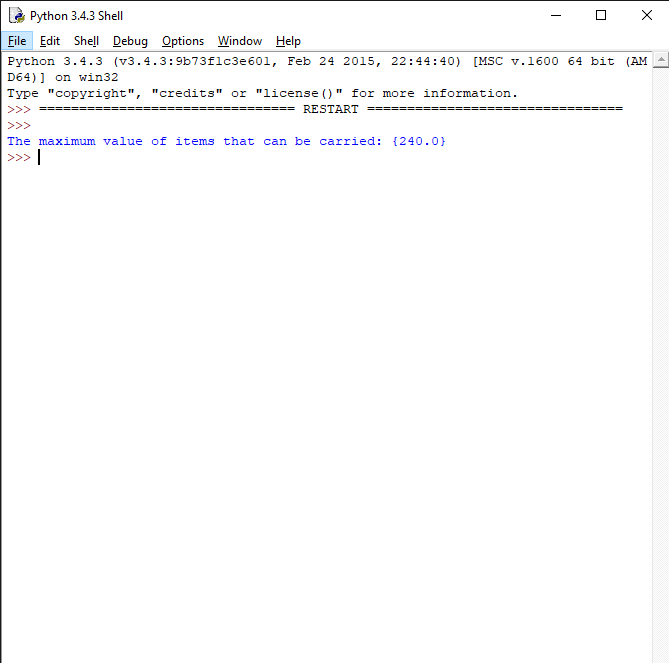
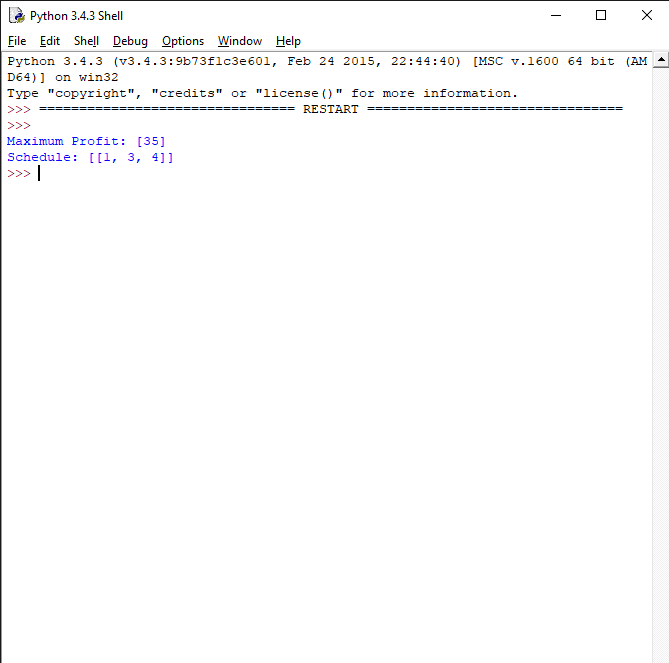
1. Coin Change Problem

2.KNAPSACK PROBLEM

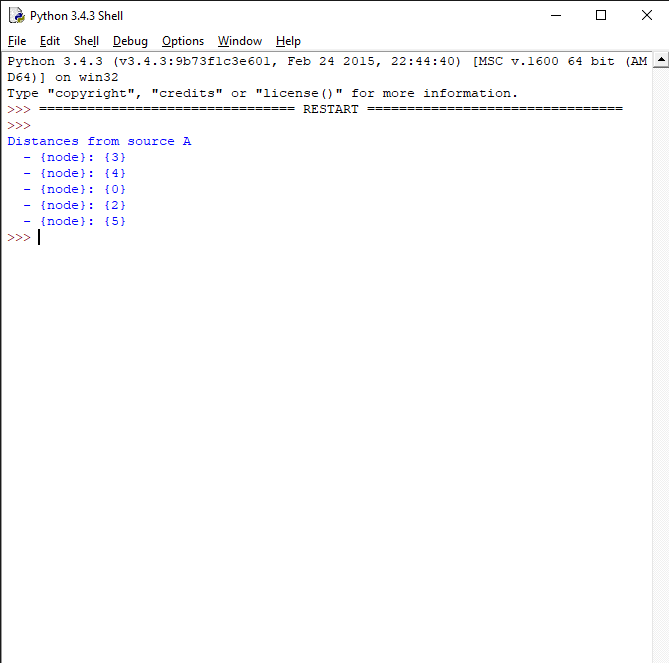


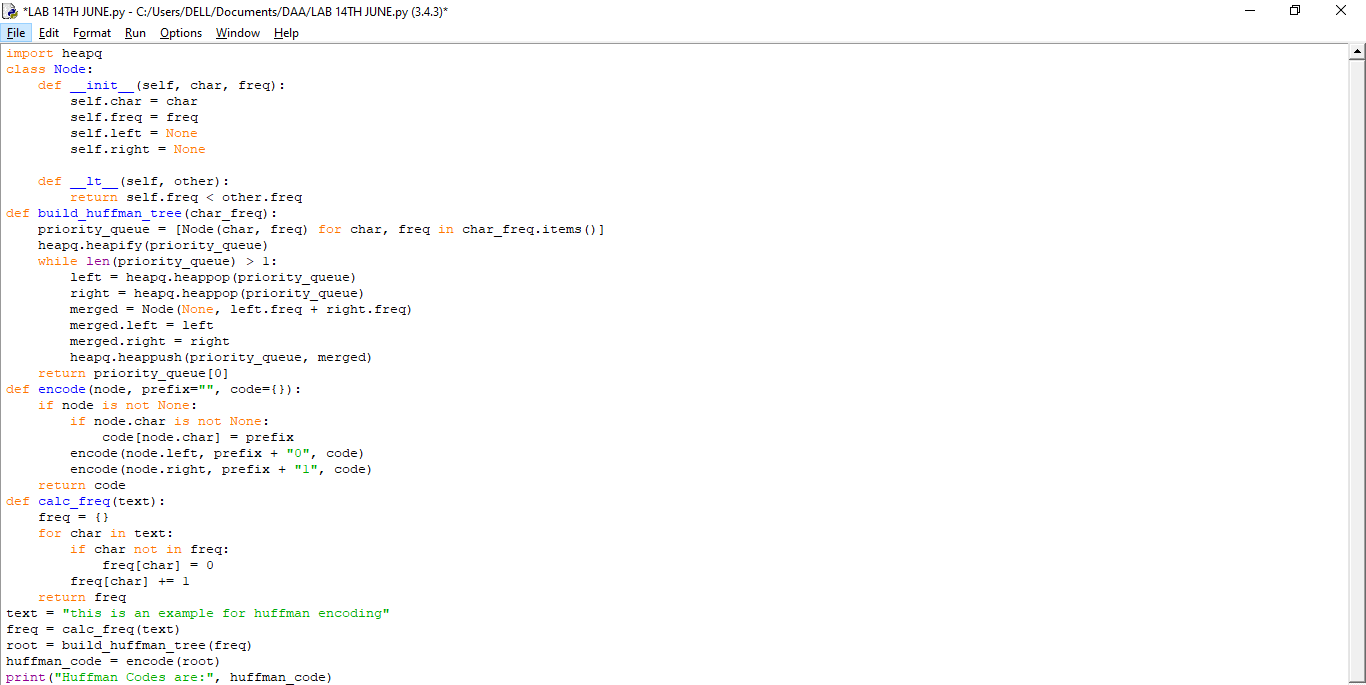


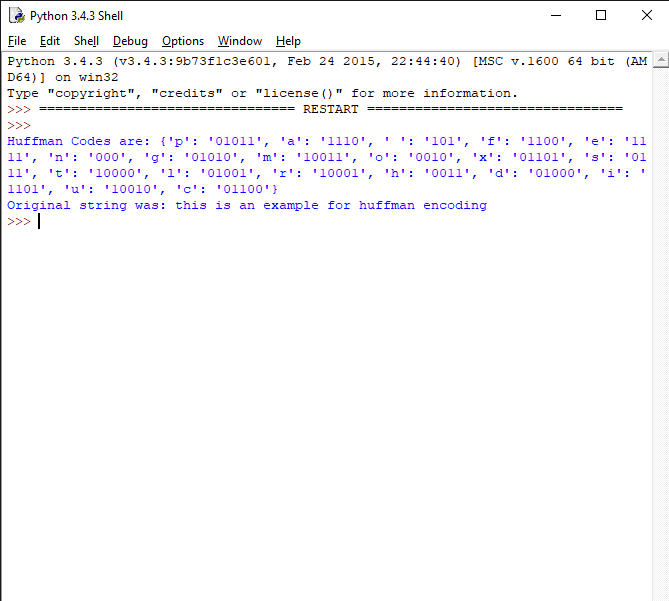
3. JOB SEQUENCING WITH DEADLINES



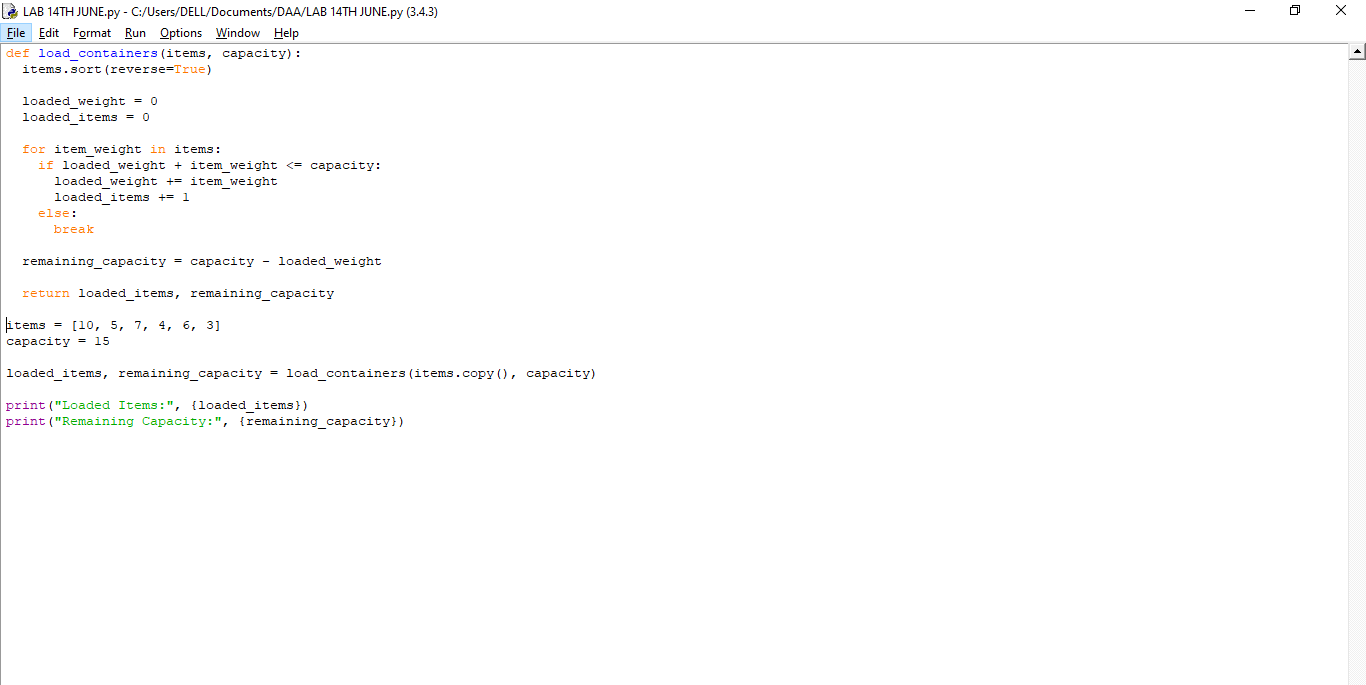
4.SINGLE SOURCE SHORTEST PATHS: DIJKSTRA'S ALGORITHM

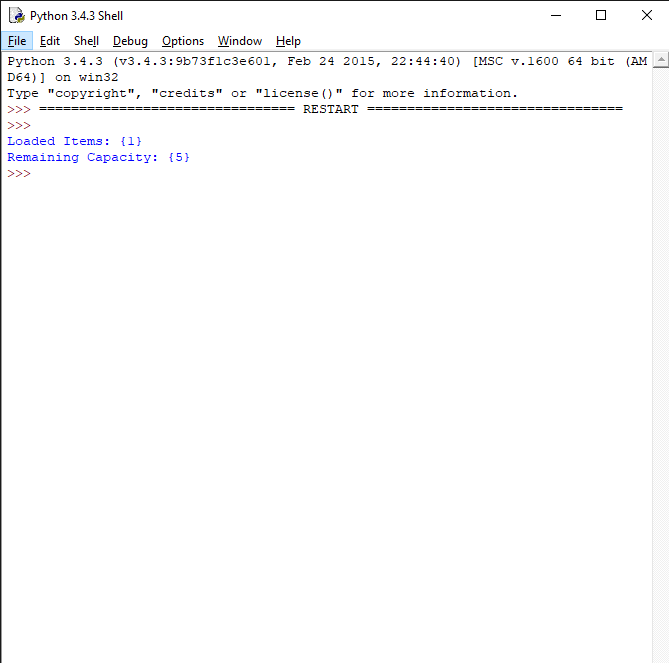


5. OPTIMAL TREE PROBLEM: HUFFMAN TREES AND CODES



6. CONTAINER LOADING





7. KRUSKAL ALGORITHM

class DisjointSet:

def \_\_init\_\_(self, vertices):

self.parent = {v: v for v in vertices}

self.rank = {v: 0 for v in vertices}

def find(self, v):

if self.parent[v] != v:

self.parent[v] = self.find(self.parent[v])

return self.parent[v]

def union(self, u, v):

root\_u = self.find(u)

root\_v = self.find(v)

if root\_u != root\_v:

if self.rank[root\_u] > self.rank[root\_v]:

self.parent[root\_v] = root\_u

elif self.rank[root\_u] < self.rank[root\_v]:

self.parent[root\_u] = root\_v

else:

self.parent[root\_v] = root\_u

self.rank[root\_u] += 1

def kruskal(vertices, edges):

disjoint\_set = DisjointSet(vertices)

edges.sort(key=lambda edge: edge[2])

mst = []

mst\_weight = 0

for edge in edges:

u, v, weight = edge

if disjoint\_set.find(u) != disjoint\_set.find(v):

disjoint\_set.union(u, v)

mst.append(edge)

mst\_weight += weight

return mst, mst\_weight

vertices = ['A', 'B', 'C', 'D', 'E', 'F']

edges = [

('A', 'B', 4),

('A', 'C', 4),

('B', 'C', 2),

('B', 'D', 6),

('C', 'D', 8),

('C', 'E', 10),

('D', 'E', 12),

('D', 'F', 8),

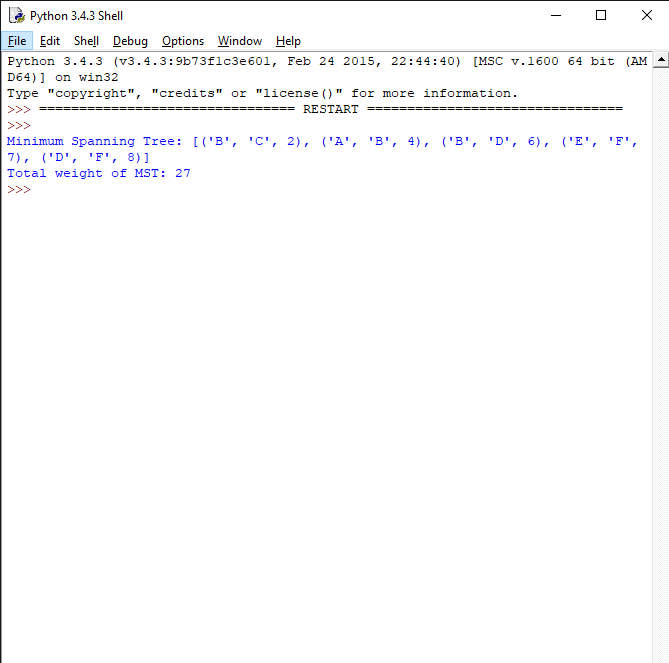
('E', 'F', 7)

]

mst, mst\_weight = kruskal(vertices, edges)

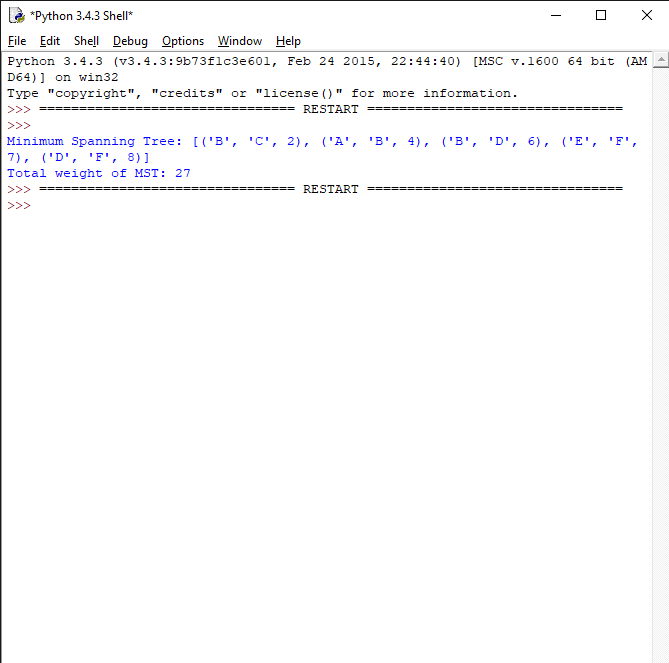
print("Minimum Spanning Tree:", mst)

print("Total weight of MST:", mst\_weight)



8.PRIM’S ALGORITHM





9. BORUVKA'S ALGORITHM

class DisjointSet:

def \_init\_(self, vertices):

self.parent = {v: v for v in vertices}

self.rank = {v: 0 for v in vertices}

def find(self, v):

if self.parent[v] != v:

self.parent[v] = self.find(self.parent[v])

return self.parent[v]

def union(self, u, v):

root\_u = self.find(u)

root\_v = self.find(v)

if root\_u != root\_v:

if self.rank[root\_u] > self.rank[root\_v]:

self.parent[root\_v] = root\_u

elif self.rank[root\_u] < self.rank[root\_v]:

self.parent[root\_u] = root\_v

else:

self.parent[root\_v] = root\_u

self.rank[root\_u] += 1

def boruvka(vertices, edges):

# Initialize disjoint set

disjoint\_set = DisjointSet(vertices)

mst = [] # Minimum Spanning Tree

mst\_weight = 0 # Weight of the MST

num\_components = len(vertices)

while num\_components > 1:

cheapest = {v: None for v in vertices}

for edge in edges:

u, v, weight = edge

root\_u = disjoint\_set.find(u)

root\_v = disjoint\_set.find(v)

if root\_u != root\_v:

if cheapest[root\_u] is None or cheapest[root\_u][2] > weight:

cheapest[root\_u] = edge

if cheapest[root\_v] is None or cheapest[root\_v][2] > weight:

cheapest[root\_v] = edge

for v in vertices:

if cheapest[v] is not None:

u, v, weight = cheapest[v]

if disjoint\_set.find(u) != disjoint\_set.find(v):

disjoint\_set.union(u, v)

mst.append(cheapest[v])

mst\_weight += weight

num\_components -= 1

return mst, mst\_weight

vertices = ['A', 'B', 'C', 'D', 'E', 'F']

edges = [

('A', 'B', 4),

('A', 'C', 4),

('B', 'C', 2),

('B', 'D', 6),

('C', 'D', 8),

('C', 'E', 10),

('D', 'E', 12),

('D', 'F', 8),

('E', 'F', 7)

]

mst, mst\_weight = boruvka(vertices, edges)

print("Minimum Spanning Tree:", mst)

print("Total weight of MST:", mst\_weight)

