ASSIGNMENT - 2

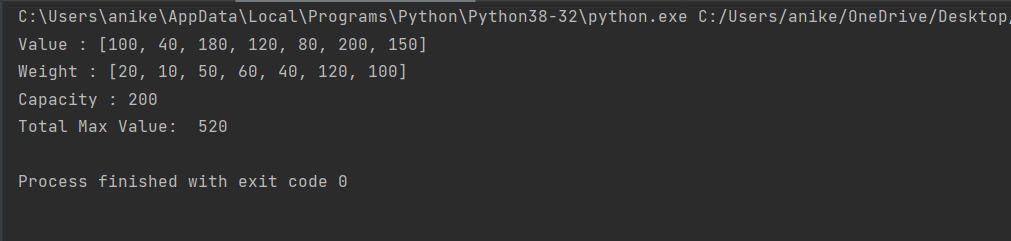
ANIKEIT SETHI(190001003)

Code: -

# 0-1 knapsack algorithm using branch and bound  
  
data\_weight = [20, 40, 60, 100, 120, 10, 50]  
data\_value = [100, 80, 120, 150, 200, 40, 180]  
capacity = 200  
  
values = []  
  
for i in range(len(data\_value)):  
 cost = data\_value[i] // data\_weight[i]  
 values.append([cost, data\_value[i], data\_weight[i]])  
  
values.sort(reverse=True)  
  
  
data\_cost = [i[0] for i in values]  
data\_weight = [i[2] for i in values]  
data\_value = [i[1] for i in values]  
  
  
class State(object):  
 def \_\_init\_\_(self, level, benefit, weight, token):  
  
 self.level = level  
 self.benefit = benefit  
 self.weight = weight  
 self.token = token  
 self.available = self.token[:self.level] + [1] \* (len(data\_value) - level)  
 self.ub = self.upperbound()  
  
 def upperbound(self):  
 upperbound = 0  
 weight\_accumulate = 0  
 for i in range(len(data\_weight)):  
 if data\_weight[i] \* self.available[i] <= capacity - weight\_accumulate:  
 weight\_accumulate += data\_weight[i] \* self.available[i]  
 upperbound += data\_value[i] \* self.available[i] # Anikeit Sethi

else:  
 upperbound += data\_value[i] \* (capacity - weight\_accumulate) / data\_weight[i] \* self.available[i]  
 break  
 return upperbound  
  
 def develop(self):  
 level = self.level + 1  
 if self.weight + data\_weight[self.level] <= capacity:  
 left\_weight = self.weight + data\_weight[self.level]  
 left\_benefit = self.benefit + data\_value[self.level]  
 left\_token = self.token[:self.level] + [1] + self.token[self.level + 1:]  
 left\_child = State(level, left\_benefit, left\_weight, left\_token)  
 else:  
 left\_child = None  
  
 right\_child = State(level, self.benefit, self.weight, self.token)  
 if left\_child is not None:  
 return [left\_child, right\_child]  
 else:  
 return [right\_child]  
  
  
Root = State(0, 0, 0, [0] \* len(data\_value))  
waiting\_States = []  
current\_state = Root  
while current\_state.level < len(data\_value):  
 waiting\_States.extend(current\_state.develop())  
 waiting\_States.sort(key=lambda x: x.ub)  
 current\_state = waiting\_States.pop()  
best\_solution = current\_state  
  
print("Value :", data\_value)  
print("Weight :", data\_weight)  
print("Capacity :", capacity)  
  
print("Total Max Value: ", best\_solution.benefit)

Output: -



Graph: -

