Spring
2021

KNAPSACK PROBLEM

Project Report

INSTRUCTOR: SIR EMAD

CLASS ID: 106397

BUSHRA 63759

Problem Statement 1:

Knapsack Problem:

(Fractional deal Greedy Algorithm Approach)

A thief finds much more loot than his bag can fit. Help him to find the most valuable combination

Of items assuming that any fraction of a loot item can be put into his bag.

Problem Description:

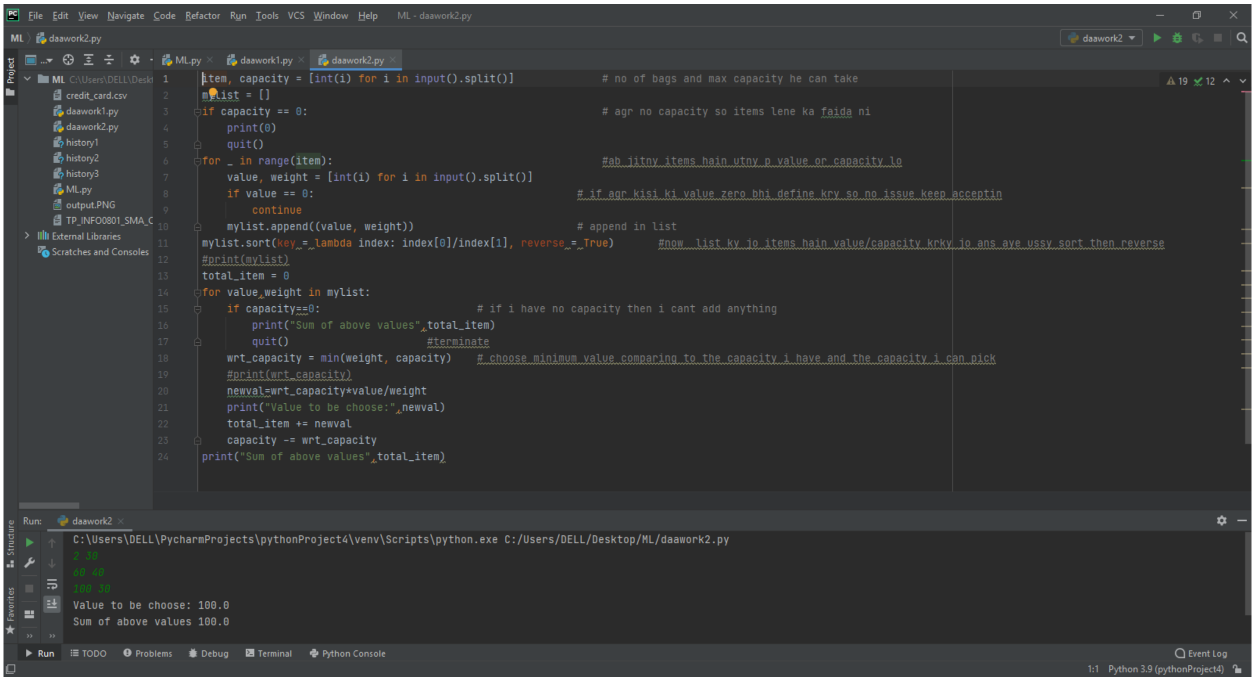
Goal: To implement an algorithm for the fractional knapsack problem

Input: First line of input contains the number of items and the capacity of bag. On behalf of number of inputs the next lines contains the value and weight from which the thief need to find the most valuable combination.

Tool: PyCharm

Language: Python

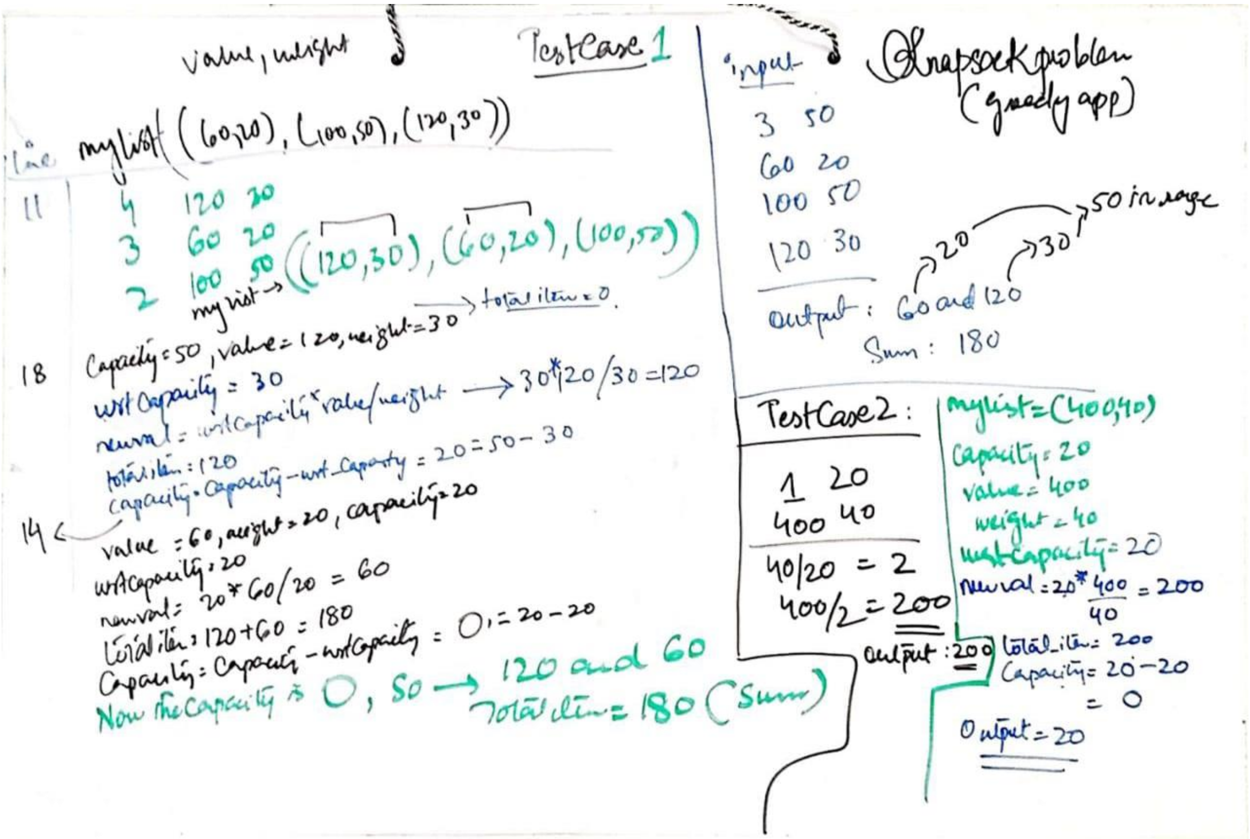
Code and Output:



```python
item, capacity = [int(i) for i in input().split()]          # no of bags and max capacity he can take
mylist = []
if capacity == 0:                                           # agr no capacity so items lene ka faida ni
    print(0)
    quit()
for _ in range(item):                                       #ab jitny items hain utny p value or capacity lo
    value, weight = [int(i) for i in input().split()]
    if value == 0:                                      # if agr kisi ki value zero bhi define kry so no issue keep acceptin
        continue
    mylist.append((value, weight))                      # append in list
mylist.sort(key = lambda index: index[0]/index[1], reverse = True)      #now  list ky jo items hain value/capacity krky jo ans aye ussy sort then reverse
#print(mylist)
total_item = 0
for value,weight in mylist:
    if capacity==0:                         # if i have no capacity then i cant add anything
        print("Sum of above values",total_item)
        quit()                      #terminate
    wrt_capacity = min(weight, capacity)    # choose minimum value comparing to the capacity i have and the capacity i can pick
    #print(wrt_capacity)
    newval=wrt_capacity*value/weight
    print("Value to be choose:",newval)
    total_item += newval
    capacity -= wrt_capacity
print("Sum of above values",total_item)
```

```
C:\Users\DELL\PycharmProjects\pythonProject4\venv\Scripts\python.exe C:/Users/DELL/Desktop/ML/daawork2.py
2 30
60 40
100 30
Value to be choose: 100.0
Sum of above values 100.0
```

Working for the test cases



value, weight

TestCase 1

line mylist((60,20), (100,50), (120,30))

11 | 4 120 30
3 60 20
2 100 50
mylist→((120,30), (60,20), (100,50))
→ total item = 0.

18 Capacity = 50, value = 120, weight = 30
wrt Capacity = 30
newval = wrtCapacity*value/weight → 30*120/30 = 120
total item : 120
capacity = capacity - wrt_capacity = 20 = 50 - 30

14 ← value = 60, weight = 20, capacity = 20
wrtCapacity = 20
newval = 20*60/20 = 60
total item = 120+60 = 180
Capacity = Capacity - wrtCapacity = 0 = 20-20
Now the Capacity is 0, so → 120 and 60
Total item = 180 (Sum)

Knapsack problem
(greedy app)

input
3 50
60 20
100 50
120 30

output : 60 and 120 →20 →30 →50 in range
Sum : 180

TestCase2 :

1 20
400 40

40/20 = 2
400/2 = 200

output : 200

mylist=(400,40)
Capacity = 20
value = 400
weight = 40
wrtCapacity = 20
newval = $20 * \frac{400}{40} = 200$
total_item = 200
Capacity = 20-20
= 0
Output = 20

Problem Statement 2:

Knapsack Problem:

(Dynamic Programming)

You are given a set of bars of gold and your goal is to take as much gold as possible into your bag. There is just one copy of each bar and for each bar you can either take it or not (hence you cannot take a fraction of a bar).

Problem Description:

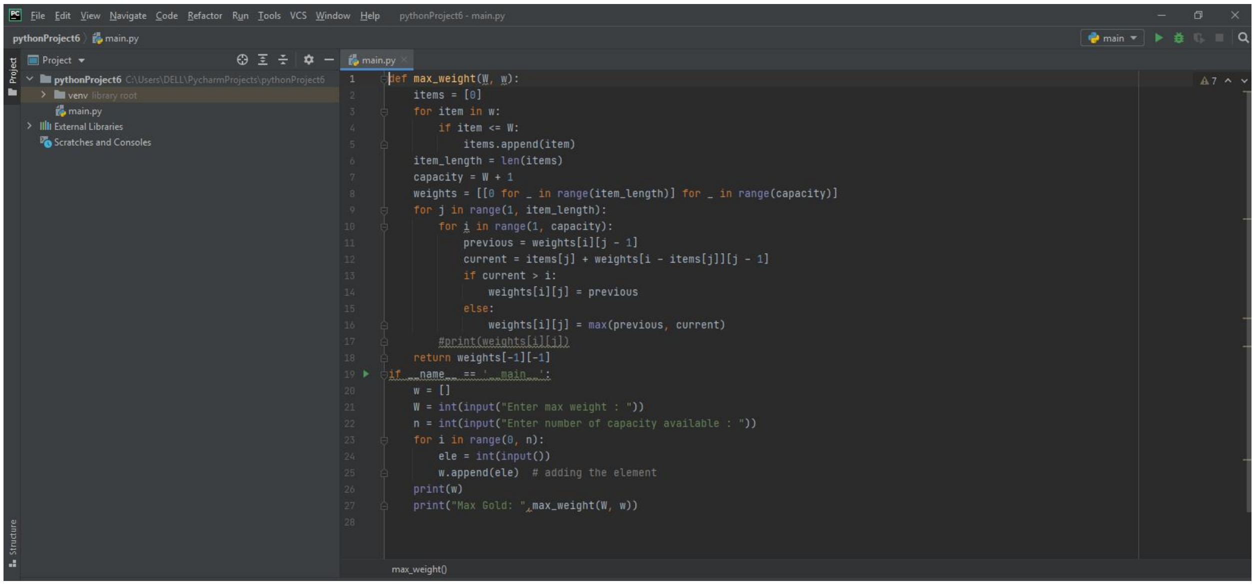
Goal: To implement an algorithm for the maximum weight of gold that fits into a bag of capacity.

Input: Firstly, asking for max weight and number of capacity available and take inputs for the capacity in the form of list.

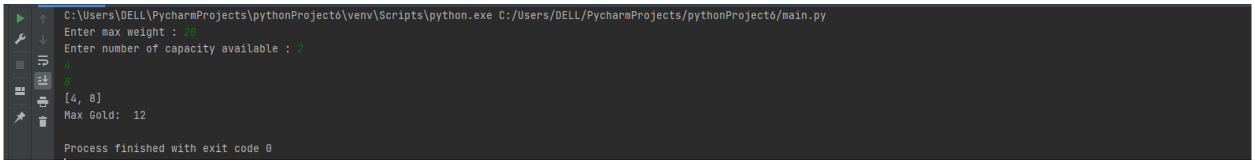
Tool: PyCharm

Language: Python

Code and Output:

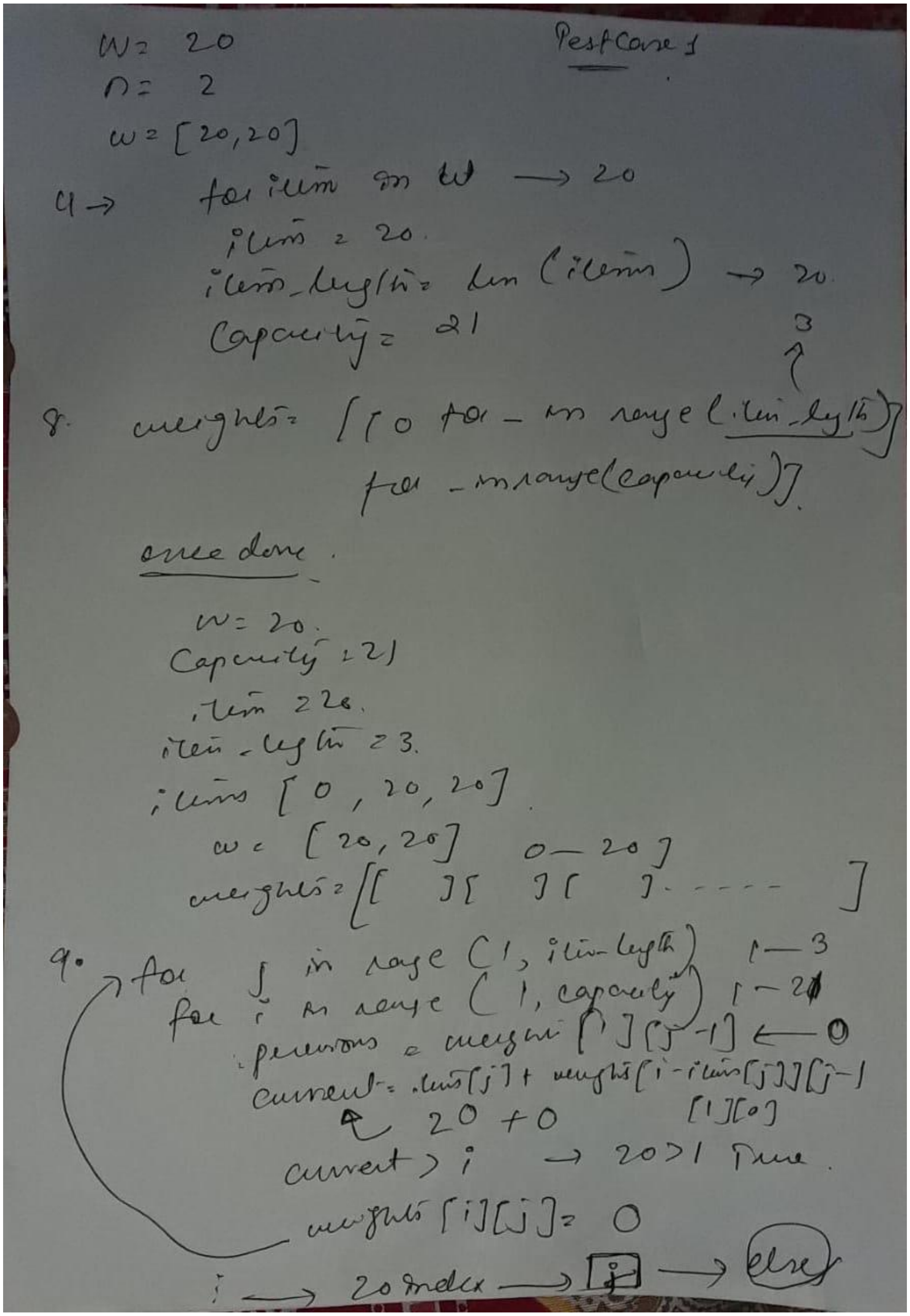


```python
def max_weight(W, w):
    items = [0]
    for item in w:
        if item <= W:
            items.append(item)
    item_length = len(items)
    capacity = W + 1
    weights = [[0 for _ in range(item_length)] for _ in range(capacity)]
    for j in range(1, item_length):
        for i in range(1, capacity):
            previous = weights[i][j - 1]
            current = items[j] + weights[i - items[j]][j - 1]
            if current > i:
                weights[i][j] = previous
            else:
                weights[i][j] = max(previous, current)
        #print(weights[i][j])
    return weights[-1][-1]
if __name__ == '__main__':
    w = []
    W = int(input("Enter max weight : "))
    n = int(input("Enter number of capacity available : "))
    for i in range(0, n):
        ele = int(input())
        w.append(ele)  # adding the element
    print(w)
    print("Max Gold: ",max_weight(W, w))

```

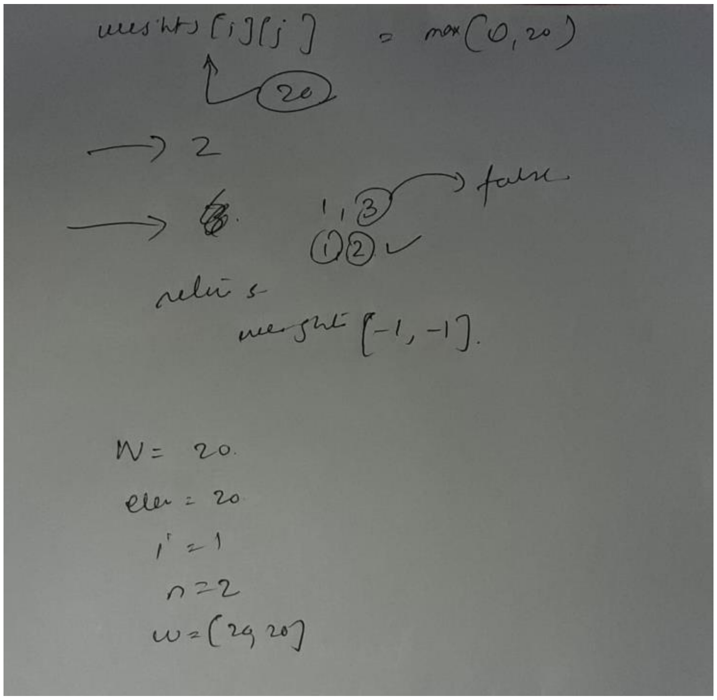


```
C:\Users\DELL\PycharmProjects\pythonProject6\venv\Scripts\python.exe C:/Users/DELL/PycharmProjects/pythonProject6/main.py
Enter max weight : 20
Enter number of capacity available : 2
4
8
[4, 8]
Max Gold:  12

Process finished with exit code 0
```

Working for test Cases:



```
W= 20                                   Testcase 1
n= 2
w= [20,20]
4 →  for item in W → 20
       items = 20.
       items_length= len (items) → 20.
                                     3
       Capacity= 21                  ↑
8.  weights= [[0 for _ in range(item_length)]
                 for _ in range(capacity)]].

  once done.

     w= 20.
    Capacity =21
     item =20.
    item_length =3.
    items [0, 20, 20].
      w= [20, 20]     0— 20]
     weights= [[  ][  ][  ]. - - - -  ]
9. for j in range (1, item_length)  1— 3
    for i in range (1, capacity)  1— 20
       previous = weight[i][j-1] ← 0
       current= items[j] + weights[i-items[j]][j-1
                 20 + 0            [1][0]
       current > i  → 20>1 True.
       weights[i][j]= 0
   i ⟶ 20 index ⟶ [j] ⟶ else
```



weights[i][j] = max(0, 20)

↳ ⑳

→ 2

→ ~~8~~. 1, ③ → false

①② ✓

return :-

weight [-1, -1].

W = 20.

ele = 20

i = 1

n = 2

w = [29, 20]

References:

https://www.tutorialspoint.com/data_structures_algorithms/greedy_algorithms.htm

https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/

https://www.geeksforgeeks.org/dynamic-programming/

https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/