Week-9, Practice Programming

Problem 1

A thief robbing a store and can carry a maximum weight of `W` in his bag. There are `n` items in the store with weights $\{w_1, w_2, w_3, \ldots, w_n\}$ and corresponding values $\{v_1, v_2, v_3, \ldots, v_n\}$. What items should he select to get maximum value? He cannot break an item, either he picks the complete item or doesn't pick it.

Write a function **MaxValue(Items, W)** that accepts a dictionary `Items`, where the key of the dictionary represents the item number (1 to n) and the corresponding value is a tuple `(weight of item, Value of item)`. The function accepts one more integer `W`, which represents the maximum weight capacity of the bag. The function returns the total value of all selected items, which is maximum in all possible selection.

Example Input

```
8 # W - Maximum weight capacity of bag
{1: (2,10),2: (3,20),3: (4,40)} # Items
```

Output

```
60 #total value is 60 which is maximum.
```

Explanation

The thief can pick items 2 and 3 where the total weight of picked items is 3 + 4 = 7 which is less than the maximum capacity of the bag, but he gets maximum value (60) for this selection.

Sample Input

```
30
{1:(10,2), 2:(15,5), 3:(6,8), 4:(9,1)}
```

Output

```
14
```

Solution

```
# Solution code
def MaxValue(Items,W):
    (m,n) = (len(Items),W)
    c = []
    for i in range(m+1):
        row = []
        for j in range(n+1):
            row.append(0)
        c.append(row.copy())
    for i in range(1,m+1):
        for w in range(1,n+1):
```

```
            if Items[i][0] > w:
                c[i][w] = c[i-1][w]
            else:
                c[i][w] = max(c[i-1][w],Items[i][1]+c[i-1][w-Items[i][0]])
    return(int(c[m][n]))



#Suffix Code(Visible)
W = int(input())
Items= eval(input())
print(MaxValue(Items,W))
```

Public Test case

Input 1

```
8
{1: (2,10),2: (3,20),3: (4,40)}
```

Output 1

```
60
```

Input 2

```
30
{1:(10,2), 2:(15,5), 3:(6,8), 4:(9,1)}
```

Output 2

```
14
```

Input 3

```
50
{1:(5,20), 2:(15,50), 3:(16,80), 4:(9,100), 5:(3,200),6:(4,150),7:(2,300),8:(6,120),9:(3,70),10:(6,40)}
```

Output 3

```
1060
```

Private Test Case

Input 1

```
25
{1:(5,20), 2:(1,50), 3:(16,80), 4:(4,100), 5:(5,200),6:(4,150),7:(2,300),8:(6,120),9:(3,70),10:(6,40)}
```

Output 1

```
990
```

Input 2

```
10
{1:(5,20), 2:(1,50), 3:(2,80), 4:(2,100)}
```

Output 2

```
250
```

Input 3

```
10
{1:(5,20), 2:(1,50), 3:(16,80), 4:(4,100), 5:(5,200),6:(4,150),7:(2,300),8:
(6,120),9:(3,70),10:(6,40)}
```

Output 3

```
570
```

Input 4

```
20
{1:(5,20), 2:(1,50), 3:(1,80), 4:(9,100), 5:(3,200)}
```

Output 4

```
450
```

Input 5

```
5
{1:(5,20), 2:(1,50), 3:(1,80), 4:(9,100), 5:(5,2000)}
```

Output 5

```
2000
```

Problem 2

You're given a list of tuples `Activity` for `n` activities, where in each tuple `(activity_name, S, F, P)`, activity `activity_name` is scheduled to be done from start time `S` to finish time `F` and obtains a profit of `P` after the finish.

Find out the maximum profit you can obtained by scheduled activities, but no two activities should be in the subset with overlapping of time frame. If you choose an activity that finishes at time `x` then another activity can be started at time `x`, not before that.

Write a function **MaxProfit(Activity)** that accepts a list of tuples `Activity` for `n` activities and returns the value of maximum profit that can be obtained by scheduled activities.

Sample Input

```
[('A',1,2,40),('B',3,4,5),('C',0,7,6),('D',1,2,3),('E',5,6,8),('F',5,9,2),
('G',10,11,9),('H',0,11,35)]
```

Output

```
62
```

Explanation

Activity schedule `[A, B, E, G]` gives a profit of 62 which is the maximum in all possible schedules

Solution

```
#(Solution code)
def tuplesort(L, index):
    L_ = []
    for t in L:
        L_.append(t[index:index+1] + t[:index] + t[index+1:])
    L_.sort()

    L__ = []
    for t in L_:
        L__.append(t[1:index+1] + t[0:1] + t[index+1:])
    return L__

def MaxProfit(Activity):
    n = len(Activity)
    act = tuplesort(Activity, 2)
    MaxProfit = []
    for a in act:
        MaxProfit.append(a[3])
    for i in range(1, n):
        for j in range(0, i):
            if act[i][1] >= act[j][2] and MaxProfit[i] < MaxProfit[j] +
act[i][3] + 1:
                MaxProfit[i] = MaxProfit[j] + act[i][3]
    return max(MaxProfit)
```

```
# Suffix code(visible)
L = eval(input())
print (MaxProfit(L))
```

Public Test case

Input 1

```
[('A',1,2,40),('B',3,4,5),('C',0,7,6),('D',1,2,3),('E',5,6,8),('F',5,9,2),
('G',10,11,9),('H',0,11,35)]
```

Output 1

```
62
```

Input 2

```
[('A',1,2,2),('B',3,4,15),('C',0,7,16),('D',1,2,3),('E',5,6,8),('F',5,9,2),
('G',10,11,9),('H',0,11,35)]
```

Output 2

```
35
```

Input 3

```
[('A',1,2,2),('B',3,4,15),('C',5,7,16),('D',7,8,3),('E',8,9,8),('F',5,9,2),
('G',10,11,9),('H',0,11,15)]
```

Output 3

```
53
```

Private Test Case

Input 1

```
[('A',1,2,2),('B',3,4,15),('C',5,7,16),('D',7,8,3),('E',8,9,8)]
```

Output 1

```
44
```

Input 2

```
[('A',1,12,2),('B',3,14,15),('C',5,17,16),('D',1,8,3),('E',1,9,8),
('F',1,9,2),('G',1,11,9),('H',1,11,15)]
```

Output 2

```
16
```

Input 3

```
[('A',0,2,2),('B',2,4,15),('C',3,7,16),('D',5,8,3),('E',7,9,8),('F',8,9,2),
('G',10,11,9),('H',10,11,15)]
```

Output 3

```
41
```

Input 4

```
[('A',0,2,2)]
```

Output 4

```
2
```

Input 5

```
[('A',0,2,22),('B',2,4,45),('C',3,7,26),('D',5,8,13)]
```

Output 5

```
80
```