20171248 안재형
CAUSWE 2021
Algorithm Course - Class#2
(Prof.Eunwoo Kim)

Assignment #4

Results(Output)

Problem #1

```
5
55
```

Problem #2

```
Original Matrix :
[27, 66, 63]
[5, 63, 51]
[26, 90, 57]
[15, 34, 78]
[7, 96, 3]
-------------------------
[70, 19, 55, 94, 49, 94, 23]
[10, 91, 60, 95, 70, 97, 75]
[90, 6, 34, 8, 68, 6, 12]
-------------------------
[15, 88, 80, 49, 1, 45, 9, 7, 53, 64]
[97, 95, 57, 72, 93, 41, 27, 49, 83, 19]
[36, 89, 73, 15, 7, 13, 68, 11, 65, 69]
[53, 76, 88, 83, 23, 54, 95, 59, 97, 16]
[82, 1, 67, 91, 71, 52, 73, 78, 30, 43]
[97, 59, 83, 42, 27, 3, 65, 97, 38, 32]
[26, 69, 48, 70, 23, 57, 89, 66, 93, 7]
-------------------------
Minimum number of computation :  360
Optimal chain order :
1 2
0 2
Output Matrix :
[3465939, 3758082, 4186335, 3550968, 2040150, 2174553, 3576099, 3147492, 3653835, 2111844]
[2726568, 2900343, 3199337, 2781129, 1662482, 1701824, 2787247, 2469617, 2858760, 1590112]
[4180554, 4443924, 4898660, 4204254, 2480612, 2543372, 4297564, 3809486, 4373862, 2379172]
[2268949, 2574764, 2935621, 2460432, 1351486, 1563837, 2346781, 2030236, 2439785, 1630696]
[3389289, 3403512, 3633715, 3229998, 2053150, 1881763, 3449459, 3118402, 3471105, 1550684]
```

Problem #3

```
The bag's maximum value is  234.0
item # 5  :  1.0
item # 6  :  1.0
item # 4  :  1.0
item # 1  :  1.0
item # 3  :  0.3333333333333333
```

Codes

Problem #1

```
memo = []


def fibonacci(n):
    if len(memo) < (n+1):
        for i in range(n-len(memo)+1):
            memo.append(None)

    if memo[n] is not None:
        return memo[n]
    else:
        result = None
        if n == 0:
            result = 0
        elif n == 1:
            result = 1
        else:
            result = fibonacci(n - 2) + fibonacci(n - 1)
        memo[n] = result
        return result


print(fibonacci(5))
print(fibonacci(10))
```

Problem #2

```
import random
matrix = [
    [[random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)]],

    [[random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
        random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
        random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
        random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)]],
```

```
    [[random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)],
     [random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99),
      random.randint(1, 99), random.randint(1, 99), random.randint(1, 99)]]
]

memo_count = [[None for col in range(len(matrix))]
              for row in range(len(matrix))]
memo_way = [[None for col in range(len(matrix))]
            for row in range(len(matrix))]


def M(i, j):
    # Matrix들의 sequence of dimensions는 matrix에서 직접 추출하여 feed됩니다.
    if memo_count[i][j] != None:
        return memo_count[i][j]

    result = None
    result_k = None
    if i == j:
        result = 0
        result_k = i
    else:
        for k in range(i, j):
            count = M(i, k) + M(k+1, j) + \
                (len(matrix[i]) * len(matrix[k+1]) * len(matrix[j][0]))
            if result is None:
                result = count
                result_k = k
            elif count < result:
                result = count
                result_k = k

    memo_count[i][j] = result
    memo_way[i][j] = result_k
    return result


def matrix_multiple(x, y):
    result = [[0 for col in range(len(y[0]))]
              for row in range(len(x))]
    for i in range(len(x)):
        for j in range(len(y[0])):
            for k in range(len(y)):
                result[i][j] += x[i][k] * y[k][j]
```

```
    return result


def calculate(i, j):
    if i == j:
        return matrix[i]
    elif (j-i) == 1:
        print(i, j)
        return matrix_multiple(matrix[i], matrix[j])
    else:
        result = matrix_multiple(
            calculate(i, memo_way[i][j]),
            calculate(memo_way[i][j]+1, j)
        )
        print(i, j)
        return result


print("Original Matrix : ")
for i in matrix:
    for j in i:
        print(j)
    print("-------------------------")

print("Minimum number of computation : ", M(0, len(matrix) - 1))

print("Optimal chain order : ")
calculated_result = calculate(0, len(matrix) - 1)

print("Output Matrix : ")
for i in range(len(calculated_result)):
    print(calculated_result[i])
```

Problem #3

```
bag = [
    # index, weight
]
max_capacity = 16
items = [
    # index, weight, value
    [1, 6, 60],
    [2, 10, 20],
    [3, 3, 12],
    [4, 5, 80],
    [5, 1, 30],
    [6, 3, 60],
]
sorted_items = sorted(items, key=lambda item: item[2]/item[1],
reverse=True)


def current_capacity():
    result = max_capacity
    for i in bag:
        result -= i[1]
    return result


def current_value():
    result = 0
    for i in bag:
```

```
        index = i[0]-1
        result += items[index][2] / items[index][1] * i[1]
    return result


def put_in_the_bag():
    for i in sorted_items:
        if current_capacity() <= 0:
            break
        elif current_capacity() < i[1]:
            item = [
                i[0],
                current_capacity(),
            ]
            bag.append(item)
            break
        else:
            item = [
                i[0],
                i[1],
            ]
            bag.append(item)


put_in_the_bag()
print("The bag's maximum value is ", current_value())
for i in bag:
    print("item #", i[0], " : ", i[1] / items[i[0]-1][1])
```