# Algorithm Assignment_05

*Dynamic Programming and Greedy algorithm*

20182592 장원범

# Algorithm Assignment 05

*Dynamic Programming and greedy algorithm*

## Problem

P1) Implement an algorithm of Fibonacci numbers using dynamic programming and get the answers when feeding $n$ =5 and $n$ =10 into the algorithm as inputs (find the $n$-th number).

[Note that you should store the results we've calculated and return the values when needed.]

P2) Implement a matrix-chain multiplication algorithm using dynamic programming and perform the algorithm for a chain of three matrices containing randomly chosen positive integers whose sizes are 5×3, 3×7, and 7×10, respectively (output should be a matrix of size 5×10). Display your output matrix, optimal chain order, and the minimum number of computations.

[Note that you are required to feed a sequence of dimensions of the matrices into the algorithm (i.e., $\langle 5,3,7,10 \rangle$).]

P3) Implement an algorithm for the fractional knapsack problem by a greedy strategy.

A. Assume that we have a knapsack with max weight capacity of 16, and our objective is to fill the knapsack with items such that the benefit (value) is maximum. Consider the following items and their associated weight and value:

B. Sort the items in decreasing order of value / weight, and only the last item in the sorted list need to be broken up as in a greedy approach the current item is guaranteed to be the optimum one to take. [Note that the last item denotes an item in the list that makes the knapsack of maximum capacity when filling with (fraction of) the item.]

C. Display the maximum value and its associated items (with their fraction numbers) for the problem.

| ITEM | WEIGHT | VALUE |
|---|---|---|
| 1 | 6 | 60 |
| 2 | 10 | 20 |
| 3 | 3 | 12 |
| 4 | 5 | 80 |
| 5 | 1 | 30 |
| 6 | 3 | 60 |

Table 1. Item weight / value in problem 3

# Analysis

Problem 1.

Fibonacci is a typical dynamic programming problem. $fibo(n)$ is divided into sub-problems: $fibo(n-1)+fibo(n-2)$; the answers to each sub-problem are overlapped. Therefore, it is solved by dynamic programming. I use dynamic programming using top-down method with memorization for this problem.

Problem 2.

Matrix chain problem is to get optimal order of matrix multiplication to reduce flops. One matrix chain can divided into multiple chains and each chain has optimal solution. Concatenating chain, we consider all combination of multiplication order. The answers to the sub-problems overlap when seeking answers to the problem. So I use dynamic programming using bottom-up approach.

Problem 3.

Knapsack problem can divide into 0-1 knapsack problem and fraction knapsack problem. Problem 3 is fraction knapsack problem and It is known that the problem can be solved by dynamic programming. So I use dynamic programming using bottom-up approach.

# Results

```
Qustion 1: Fibonacci numbers using dynamic programming

n =   5: fibonacci(  5) =   5
n =  10: fibonacci( 10) =  55

--------------------------------------------------------------------------------
```

Figure 1: Result of problem 1

```
Question2: Implement a matrix-chain multiplication algorithm using dynamic programming

Input Matrix 1
|    2    4    3|
|    3    0    2|
|    4    3    3|
|    4    0    0|
|    2    2    2|

Input Matrix 2
|    3    2    4    0    2    3    4|
|    4    2    0    3    4    3    1|
|    0    2    1    2    2    0    3|

Input Matrix 3
|    4    4    4    1    2    2    3    1    0    0|
|    3    1    4    2    1    3    3    4    3    0|
|    4    1    0    3    2    1    4    4    1    3|
|    2    3    3    3    3    1    0    3    0    4|
|    3    1    0    4    0    4    4    1    2    0|
|    0    4    1    3    3    3    0    2    2    1|
|    1    2    3    4    1    2    1    3    2    3|

result matrix
|  321  311  295  387  213  327  289  317  195  186|
|  178  154  151  222  113  169  171  199  118  121|
|  368  346  326  445  242  365  340  377  229  222|
|  176  160  140  208  116  164  172  188  112  108|
|  216  204  194  262  142  216  198  220  134  130|

index and corresponding dimension
0 [dim: 5] / 1 [dim: 3] / 2 [dim: 7] / 3 [dim: 10]

computation order displayed using index is
2 [dim: 7] -> 1 [dim: 3]

minimum number of computations: 360
--------------------------------------------------------------------------------
```

Figure 2: Result of problem 2

In 'computation order displayed using index' is order of multiplication order of matrix chain. If dimension of matrix multiplication is $[5 \times 3]@[3 \times 7]@[7 \times 10]$, array of dimension is [5,3,7,10]. If result is $2[dim: 7] -> 1[dim: 3]$, the order of multiplication is $([5 \times 3]@([3 \times 7]@[7 \times 10]))$

* @ refer matrix multiplication

```
Qustion 3: Implement an algorithm for the fractional knapsack problem

Maximum weight: 16.000000

      Item     Weight      Value  value / weight
         5   1.000000         30       30.000000
         6   3.000000         60       20.000000
         4   5.000000         80       16.000000
         1   6.000000         60       10.000000
         3   3.000000         12        4.000000
         2  10.000000         20        2.000000

Max Value: 234.000000

      Item       Used
         5   1.000000
         6   3.000000
         4   5.000000
         1   6.000000
         3   1.000000
         2   0.000000

      Item Remained weight
         5   0.000000
         6   0.000000
         4   0.000000
         1   0.000000
         3   2.000000
         2  10.000000
-----------------------------------------------------------------------------------------
```

Figure 3: Result of problem 3