

정규 교육 세미나

ToBig's 10기 정운호

Algorithm

Contents

Unit 01 | Coding Concept

Unit 02 | Dynamic Programming

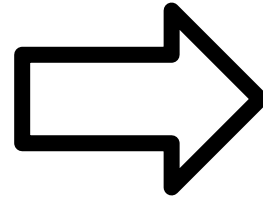
Unit 03 | Assignment

Unit 01 | Coding Concept

코딩이란 ?

Unit 01 | Coding Concept

Logical



Physical



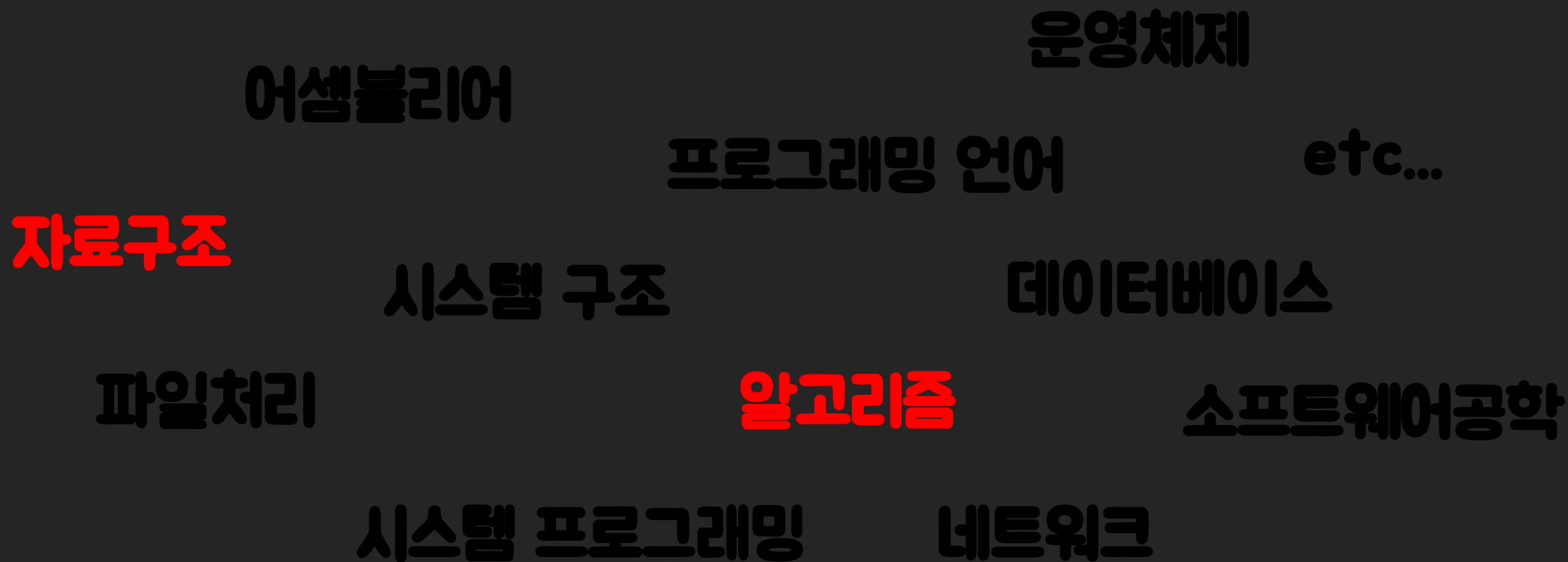
Unit 01 | Coding Concept

코딩을 잘하려면 ?

Unit 01 | Coding Concept

어셈블리어 **운영체제**
자료구조 **프로그래밍 언어** **etc...**
시스템 구조 **데이터베이스**
파일처리 **알고리즘** **소프트웨어공학**
시스템 프로그래밍 **네트워크**

Unit 01 | Coding Concept

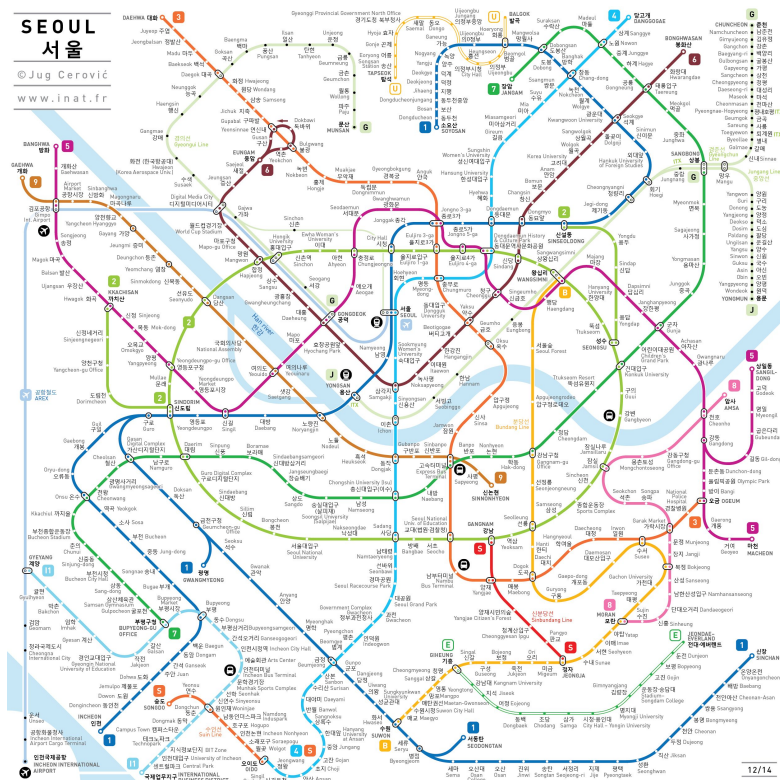


Unit 01 | Coding Concept

서울 지하철 노선도 앱을 만들어 볼까 ?

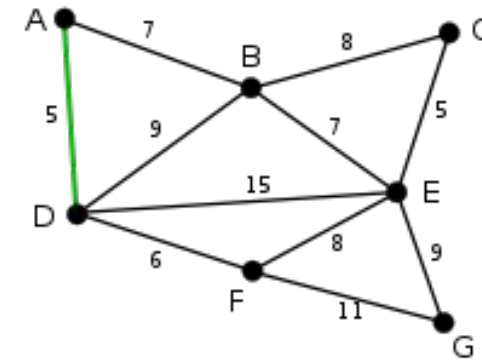
Unit 01 | NLP Overview

Logical



Physical

자료구조 : Graph



알고리즘 : 최단거리 알고리즘(Dijkstra Algorithm, Floyd-Warshall Algorithm 등) 등

Unit 01 | Coding Concept

자료구조

Stack

Tree

etc...

Array

Linked List

Hash Table

Queue

Graph

알고리즘

Greedy

Backtracking

Divide & Conquer

Sorting

etc...

Dynamic Programming

Branch & Bound

알고리즘

Definition) An algorithm is a finite set of instructions that if followed, accomplishes a particular task. All algorithms must satisfy the following criteria :

- (1) Input
- (2) Output
- (3) Definiteness
- (4) Finiteness
- (5) Effectiveness

Time Complexity

문제를 해결하는데 걸리는 시간과 입력의 함수 관계
문제의 성격과 입력 특성에 의해 결정

Time Complexity

Big-O

Big-Ω

Big-θ

Time Complexity

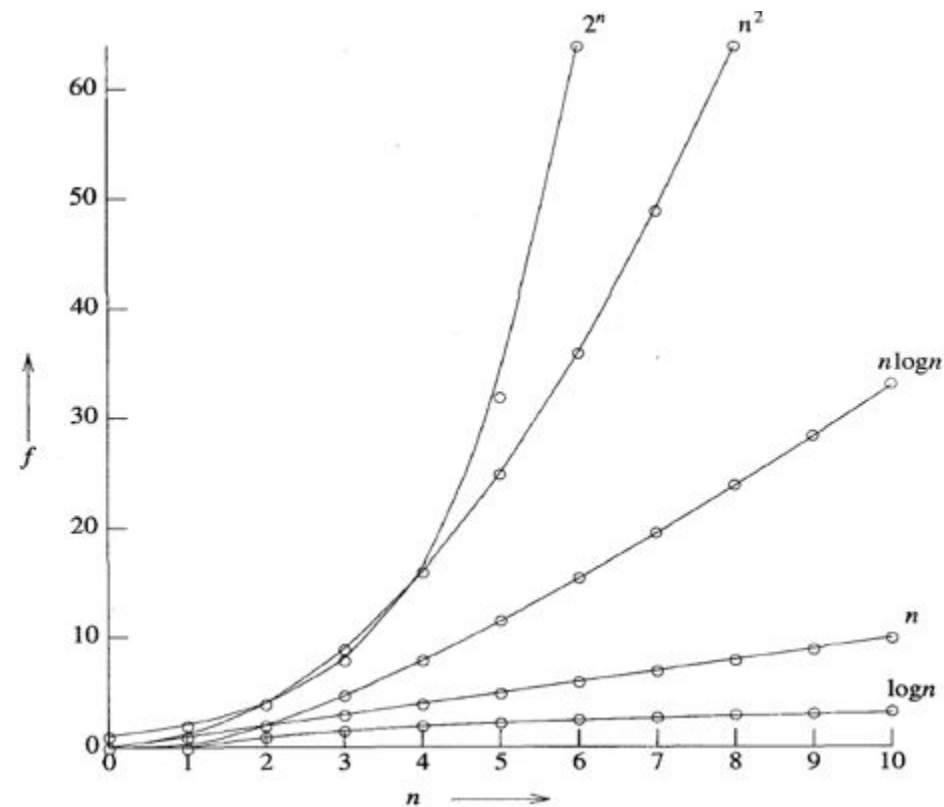
$O(n)$

$O(n \log n)$

$O(\text{rowsA} * \text{colsA} * \text{colsB})$

Unit 01 | Coding Concept

Time Complexity



Unit 01 | Coding Concept

Time Complexity

	$f(n)$						
n	n	$n \log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01 μ s	.03 μ s	.1 μ s	1 μ s	10 μ s	10 s	1 μ s
20	.02 μ s	.09 μ s	.4 μ s	8 μ s	160 μ s	2.84 h	1 ms
30	.03 μ s	.15 μ s	.9 μ s	27 μ s	810 μ s	6.83 d	1 s
40	.04 μ s	.21 μ s	1.6 μ s	64 μ s	2.56 ms	121 d	18 m
50	.05 μ s	.28 μ s	2.5 μ s	125 μ s	6.25 ms	3.1 y	13 d
100	.10 μ s	.66 μ s	10 μ s	1 ms	100 ms	3171 y	4×10^{13} y
10^3	1 μ s	9.96 μ s	1 ms	1 s	16.67 m	3.17×10^{13} y	32×10^{283} y
10^4	10 μ s	130 μ s	100 ms	16.67 m	115.7 d	3.17×10^{23} y	
10^5	100 μ s	1.66 ms	10 s	11.57 d	3171 y	3.17×10^{33} y	
10^6	1 ms	19.92 ms	16.67 m	31.71 y	3.17×10^7 y	3.17×10^{43} y	

μ s = microsecond = 10^{-6} seconds; ms = milliseconds = 10^{-3} seconds
s = seconds; m = minutes; h = hours; d = days; y = years

Figure 1.9: Times on a 1-billion-steps-per-second computer

Unit 01 | Coding Concept

Time Complexity

list

Operation	Example	Big-O	Notes
Index	<code>l[i]</code>	$O(1)$	
Store	<code>l[i] = 0</code>	$O(1)$	
Length	<code>len(l)</code>	$O(1)$	
Append	<code>l.append(5)</code>	$O(1)$	
Pop	<code>l.pop()</code>	$O(1)$	<code>l.pop(-1)</code> 과 동일
Clear	<code>l.clear()</code>	$O(1)$	<code>l = []</code> 과 유사
Slice	<code>l[a:b]</code>	$O(b-a)$	<code>l[:]</code> : $O(\text{len}(l)-0) = O(N)$
Extend	<code>l.extend(...)</code>	$O(\text{len}(...))$	확장 길이에 따라
Construction	<code>list(...)</code>	$O(\text{len}(...))$	요소 길이에 따라
check <code>==</code> , <code>!=</code>	<code>l1 == l2</code>	$O(N)$	비교
Insert	<code>l.insert(i, v)</code>	$O(N)$	<code>i</code> 위치에 <code>v</code> 를 추가
Delete	<code>del l[i]</code>	$O(N)$	
Remove	<code>l.remove(...)</code>	$O(N)$	
Containment	<code>x in/not in l</code>	$O(N)$	검색
Copy	<code>l.copy()</code>	$O(N)$	<code>l[:]</code> 과 동일 - $O(N)$
Pop	<code>l.pop(i)</code>	$O(N)$	<code>l.pop(0):O(N)</code>
Extreme value	<code>min(l)/max(l)</code>	$O(N)$	검색
Reverse	<code>l.reverse()</code>	$O(N)$	그대로 반대로
Iteration	<code>for v in l:</code>	$O(N)$	
Sort	<code>l.sort()</code>	$O(N \log N)$	
Multiply	<code>k*l</code>	$O(k \cdot N)$	<code>[1,2,3] * 3</code> » $O(N^{**2})$

<https://wayhome25.github.io/python/2017/06/14/time-complexity/>

Unit 01 | Coding Concept

Time Complexity

Dict

Operation	Example	Big-O	Notes
Index	<code>d[k]</code>	$O(1)$	
Store	<code>d[k] = v</code>	$O(1)$	
Length	<code>len(d)</code>	$O(1)$	
Delete	<code>del d[k]</code>	$O(1)$	
get/setdefault	<code>d.method</code>	$O(1)$	
Pop	<code>d.pop(k)</code>	$O(1)$	
Pop item	<code>d.popitem()</code>	$O(1)$	
Clear	<code>d.clear()</code>	$O(1)$	<code>s = {}</code> or <code>= dict()</code> 유사
View	<code>d.keys()</code>	$O(1)$	<code>d.values()</code> 동일
Construction	<code>dict(...)</code>	$O(\text{len}(...))$	
Iteration	<code>for k in d:</code>	$O(N)$	

<https://wayhome25.github.io/python/2017/06/14/time-complexity/>

Contents

Unit 01 | Coding Concept

Unit 02 | Dynamic Programming

Unit 03 | Assignment

Unit 02 | Dynamic Programming

Dynamic Programming

1. 큰 문제의 해답에 작은 문제들의 해답이 포함되어 있어 연산시에 중복이 발생하는 경우, 큰 문제를 작은 문제로 나누어 해결함으로써 중복되는 연산을 해결하는 문제 해결 전략
2. 주로 Optimization Problem을 해결하기 위한 방법

Dynamic Programming

1. 정의

2. 초기화

3. 점화식(dp식)

Unit 02 | Dynamic Programming

Dynamic Programming

1. Iterative

2. Recursive

Unit 02 | Dynamic Programming

피보나치 수열

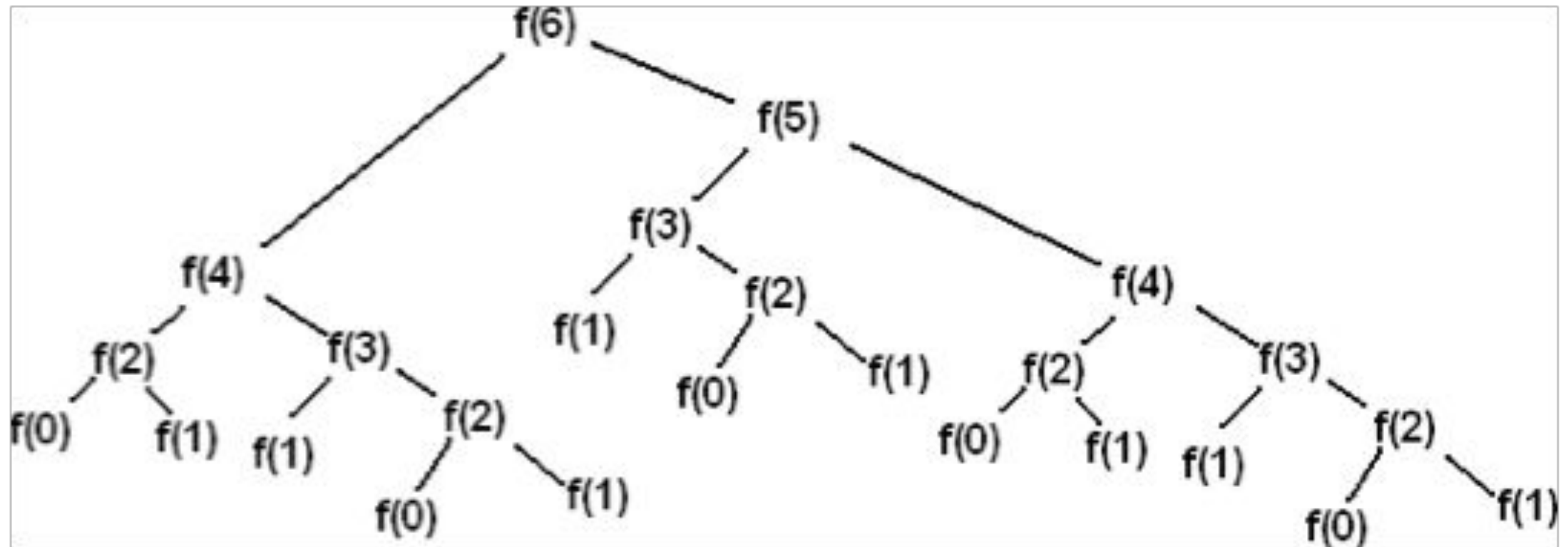
$$fibonacci(n) = \begin{cases} 0 & \text{if } n \text{ is } 0, \\ 1 & \text{if } n \text{ is } 1, \\ fibonacci(n-1) + fibonacci(n-2) & \text{otherwise.} \end{cases}$$

Unit 02 | Dynamic Programming

피보나치 수열

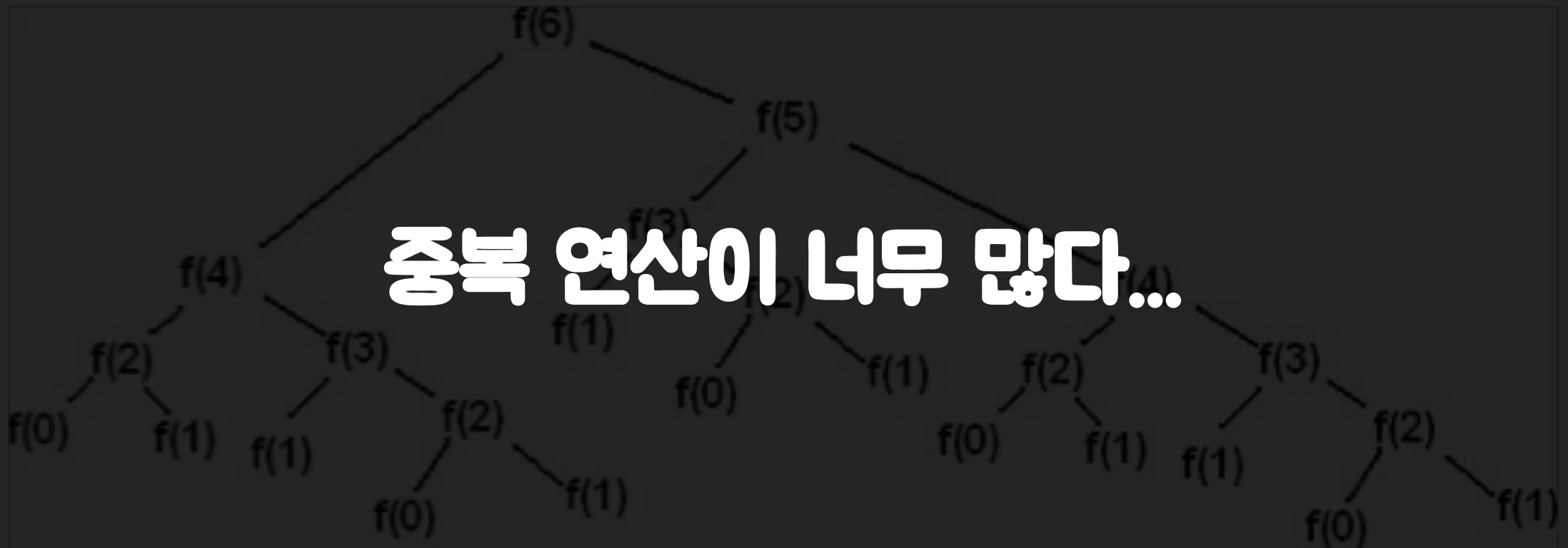
```
1  
2  def fibo(n):  
3      if n <= 1 :  
4          return n  
5      return fibo(n-1) + fibo(n-2)  
6  
7
```

Unit 02 | Dynamic Programming



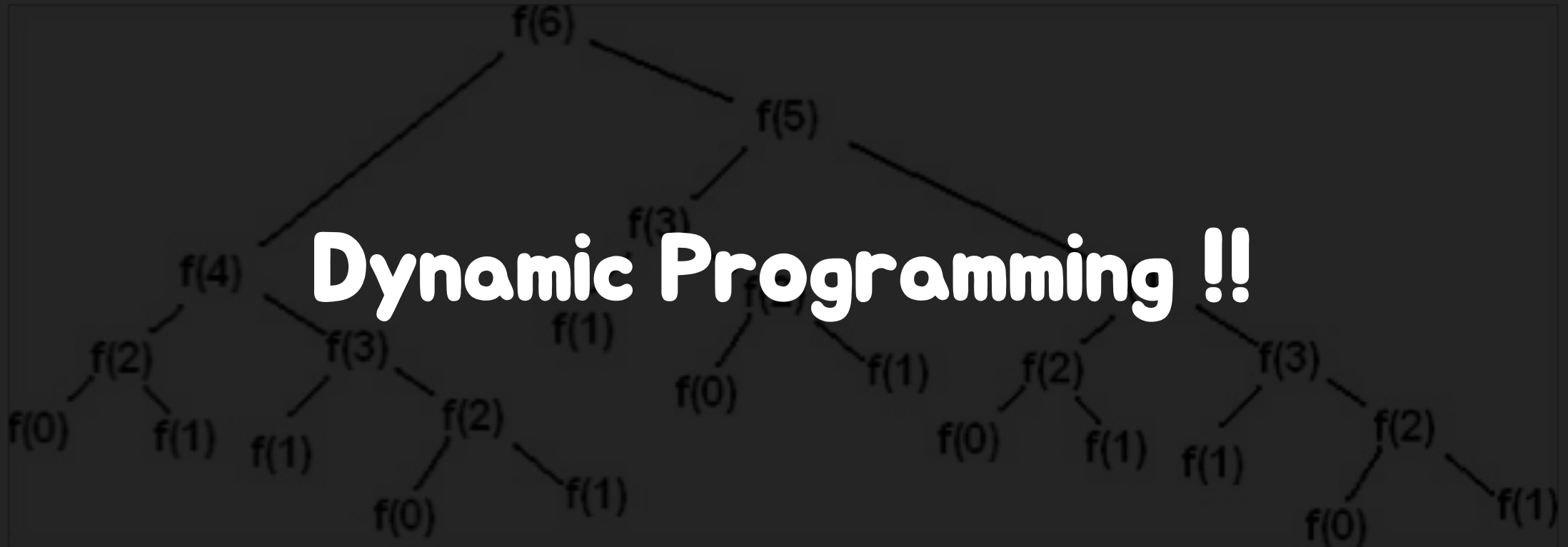
Unit 02 | Dynamic Programming

중복 연산이 너무 많다...



Unit 02 | Dynamic Programming

Dynamic Programming !!



Dynamic Programming

1. 정의

2. 초기화

3. 점화식(dp식)

Dynamic Programming

1. 정의 $dp[i] = i$ 번째 피보나치 수
2. 초기화 $dp[0] = 0, dp[1] = 1$
3. 점화식 $dp[i] = dp[i-1] + dp[i-2]$

Unit 02 | Dynamic Programming

Dynamic Programming

```
1
2  def fibo(n):
3      dp = [1,1]
4      if n >= 3:
5          for i in range(2,n):
6              dp.append(dp[i-1] + dp[i-2])
7      return dp[n-1]
8
```

Unit 02 | Dynamic Programming

Dynamic Programming

```
1
2     dp = {1:1, 2:1}
3     def fibo(n):
4         if n == 0 :
5             return 0
6         if n not in dp:
7             dp[n] = fibo(n-1) + fibo(n-2)
8         return dp[n]
9
```


Unit 02 | Dynamic Programming

Dynamic Programming

```
1
2  def fibo(n):
3      a, b = 1, 0
4      for i in range(n):
5          a, b = b, a + b
6      return b
7
```

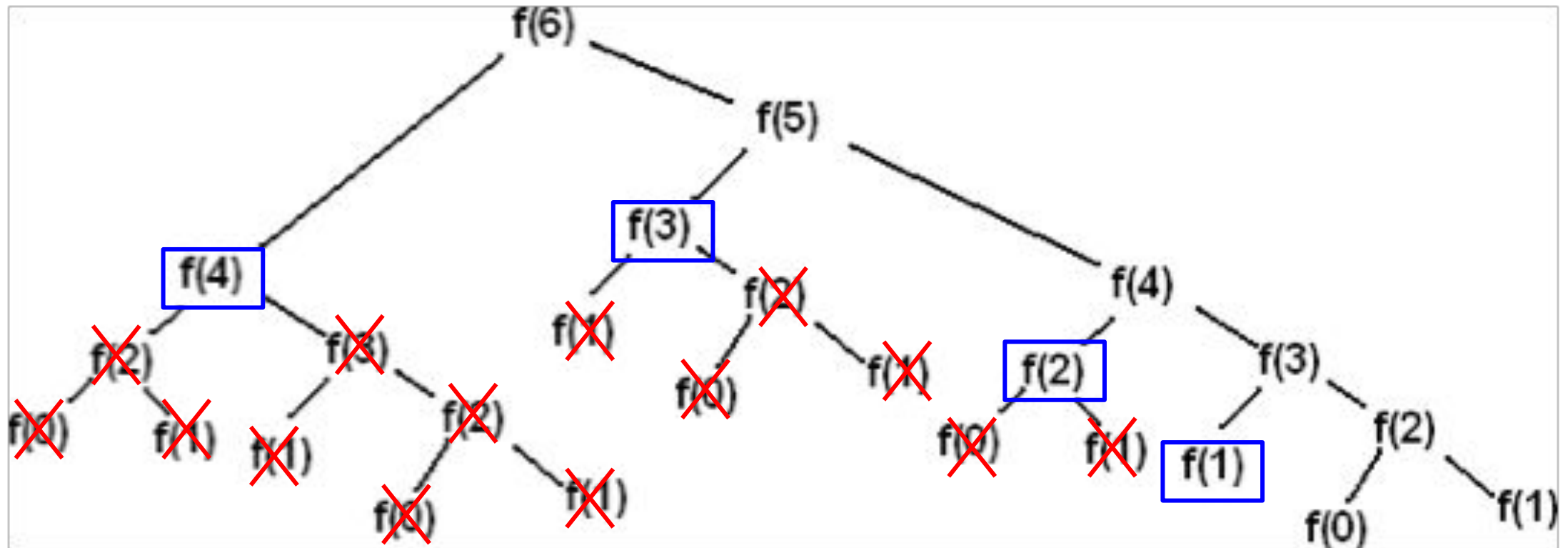
Unit 02 | Dynamic Programming

Dynamic Programming

코드는 조금씩 다르지만 핵심 전략은 같다 !

```
1  
2 def fibo(n):  
3     a, b = 1, 1  
4     for i in range(n):  
5         a, b = b, a + b  
6     return b  
7
```

Unit 02 | Dynamic Programming



Unit 02 | Dynamic Programming

Time Complexity

$$O(2^n) \rightarrow O(n)$$

Unit 02 | Dynamic Programming

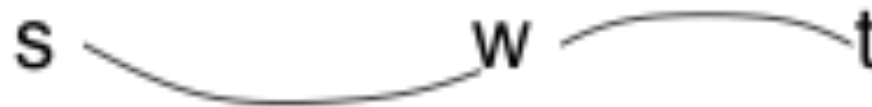
Dynamic Programming

Definition) The **principle of optimality** is said to apply in a problem if an optimal solution to an instance of the problem always contains optimal solutions to all subproblems

Example) Shortest Path

If $P(s,t)$ is a shortest path, $P(s,w)$ and $P(w,t)$ are also shortest paths.

The reverse is not true. That is, there may be a shortest path which does not contain w .

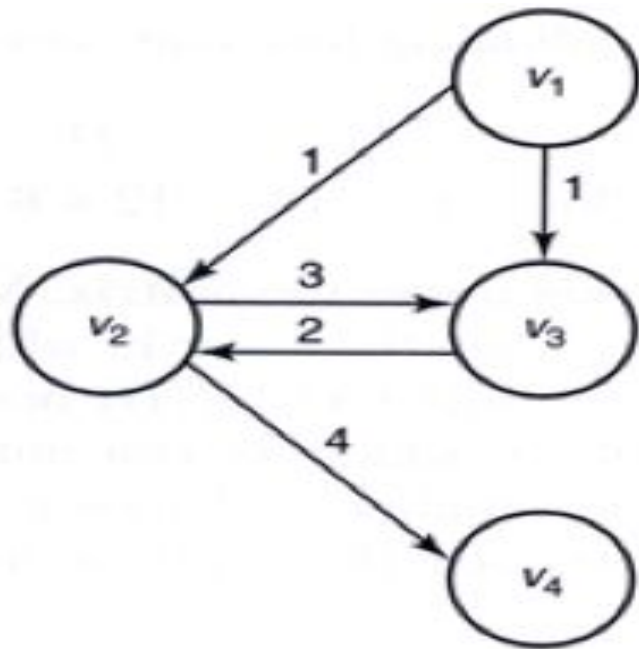


Unit 02 | Dynamic Programming

Dynamic Programming



Exception) Shortest Path



$[v_1, v_3, v_2, v_4]$ is the longest simple path.
However, $[v_1, v_3]$ is not a longest simple path.

$[v_1, v_2, v_3]$ is the longest simple path.

Dynamic Programming

1. 정의

2. 초기화

3. 점화식(dp식)

Dynamic Programming

1. 정의

수학적 귀납법과 닮아있다 !

2. 초기화

3. 점화식(dp식)

Unit 02 | Dynamic Programming

숫자 삼각형



맨 위의 숫자에서 한 번에 한 칸씩 아래로 내려가 맨 아래 줄까지 닿는 경로 중 숫자의 합이 최대인 경로의 숫자들의 합을 구하시오.

Unit 02 | Dynamic Programming

Brute Force

1. 모든 경우의 수를 확인

2. $O(2^n)$

Dynamic Programming

1. 정의 $dp[i][j] = (0,0)$ 에서 (i,j) 까지 합이 최대인 경로의 합
2. 초기화 $dp[0][0] = a[0][0]$
3. 점화식 $dp[i][j] = \max(dp[i-1][j-1], dp[i-1][j]) + a[i][j]$

Unit 02 | Dynamic Programming

Dynamic Programming

1. 정의 $dp[i][j]$ = (0,0)에서 (i,j)까지 합이 최대인 경로의 합

코드 구현은 과제 1번 !

2. 초기화 $dp[0][0] = a[0][0]$

3. 점화식 $dp[i][j] = \max(dp[i-1][j-1], dp[i-1][j]) + a[i][j]$

Unit 02 | Dynamic Programming

Time Complexity

$$O(2^n) \rightarrow O(n^2)$$

Contents

Unit 01 | Coding Concept

Unit 02 | Dynamic Programming

Unit 03 | Assignment

Unit 03 | Assignment

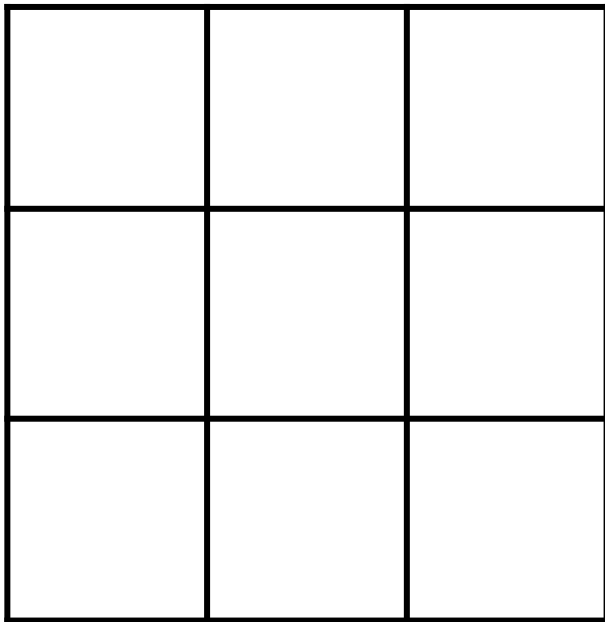
< 과제 1 >



이 그림은 크기가 5인 숫자 삼각형이다. 맨 위층의 숫자 7부터 시작해 아래에 있는 수 중 하나를 선택해, 내려올 때 선택되어 왔던 모든 수의 합이 최대가 되는 경로의 합을 구하는 알고리즘을 완성하시오.

Unit 03 | Assignment

< 과제 2 >



투빅이는 평면상 $N \times M$ 크기의 집에 있다. 집은 1×1 크기의 방으로 나누어져 있고, 각 방에는 금괴들이 놓여져 있다. 집의 가장 왼쪽 윗 방은 $(1,1)$ 이고, 가장 오른쪽 아랫 방은 (N,M) 이다. 투빅이는 현재 $(1,1)$ 에 있고 (N,M) 으로 이동하려고 한다. 투빅이가 (r,c) 에 있으면, $(r+1,c)$, $(r,c+1)$, $(r+1,c+1)$ 로 이동할 수 있고 각 방을 방문할 때마다 방에 놓여져 있는 금괴를 모두 가져갈 수 있다. 또 집 밖으로 나갈 수는 없다. 투빅이가 (N,M) 으로 이동할 때, 가져올 수 있는 금괴 개수의 최댓값을 구하시오.

Unit 03 | Assignment

〈 과제 2〉

입력은 첫째 줄에 집의 크기 N , M 이 주어진다. 둘째 줄부터 N 개 줄에는 총 M 개의 숫자가 주어지며, r 번째 줄의 c 번째 수는 (r, c) 에 놓여져 있는 금괴의 개수이다.

Unit 03 | Assignment

< 과제 3 >

만화가 오다 에이치로는 원피스를 여러 에피소드로 나누어 쓰는데, 각 장은 각각 다른 파일에 저장하곤 한다. 원피스의 모든 장을 쓰고 나서는 각 에피소드가 쓰여진 파일을 합쳐서 하나의 임시파일을 만들고, 이 임시파일이나 원래의 파일을 계속 두 개씩 합쳐서 소설의 여러 에피소드들이 연속이 되도록 파일을 합쳐나가고, 최종적으로는 하나의 파일로 합친다. 두 개의 파일을 합칠 때 필요한 비용이 두 파일 크기의 합이라고 가정할 때, 최종적인 한 개의 파일을 완성하는데 필요한 비용의 총 합을 계산하시오.

Unit 03 | Assignment

< 과제 3 >

예를 들어 ep1, ep2, ep3, ep4가 연속적인 네 개의 에피소드를 수록하고 있는 파일이고, 각 파일 크기가 각각 40, 30, 30, 50 이라고 하자. ((ep1(ep2 ep3))ep4) 순서로 합치게 되면 최종파일을 만드는 데에 필요한 비용의 합은 $60 + 100 + 150 = 310$ 이다. 반면에 ((ep1 ep2)(ep3 ep4)) 순서로 합치게 되면 최종파일을 만드는 데에 필요한 비용의 합은 $70 + 80 + 150 = 300$ 이다.

입력은 첫째 줄에 원피스를 구성하는 에피소드의 수를 나타내는 양의 정수가 주어진다. 둘째 줄에는 에피소드 1부터 에피소드 k까지 수록된 파일의 크기를 나타내는 양의 정수가 순서대로 k개 주어진다.

Unit 03 | Assignment

〈Reference〉

- <http://www.datamarket.kr/xe/>(투빅스 9기 이잉걸 파이썬 심화)
- 서강대학교 알고리즘 입문 장직현 교수님 수업 자료
- 서강대학교 scsc 코딩 워크샵 최윤영 수업 자료
- <https://www.ocmicpc.net>
- <https://www.ics.uci.edu/~pattis/ICS-33/lectures/complexitypython.txt>

Unit 04 | Assignment

〈IMG〉

- <https://medium.com/@snghojeong/c-memoization-techniques-eddd3d2>
- 서강대학교 알고리즘 입문 장직현 교수님 수업 자료
- 서강대학교 scsc 코딩 워크샵 최윤영 수업 자료
- https://ko.wikipedia.org/wiki/%ED%81%AC%EB%9F%AC%EC%8A%A4%EC%BB%AC_%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98
- <http://seoulmetro.co.kr/kr/index.do?device=PC>
- <http://www.edaily.co.kr/news/read?newsId=01167686619214168&mediaCodeNo=257>
- https://www.google.com/search?biw=908&bih=685&tbm=isch&sa=1&ei=OpulXJzyHlXn-AafpY2YCg&q=%EC%BB%B4%ED%93%A8%ED%84%B0&oq=%EC%BB%B4%ED%93%A8%ED%84%B0&gs_l=img.3..010.19089.20208..20413...1.0..1.105.705.3j4.....1...1..gws-wiz-img.KmTzlvnOgaU#imgsrc=mtT3VLbCDePXoM
- <https://wayhome25.github.io/python/2017/06/14/time-complexity/>

Q & A

들어주셔서 감사합니다.