정규 교육 세미나

ToBig's 10기 정윤호

Algorithm

Contents

Unit 01 | Coding Concept

코딩이란 ?

Unit 01 | Coding Concept

Logical

Physical

Unit 01 | Coding Concept

코딩을 잘하려면 ?

Unit 01 | Coding Concept

운영체제

어셈블리어

프로그래밍 언어

etc...

자료구조

시스템 구조

데이터베이스

파일처리

알고리즘

소프트웨어공학

시스템 프로그래밍

네트워크

Unit 01 | Coding Concept

운영체제

어셈블리어

프로그래밍 언어

etc...

자료구조

시스템 구조

데이터베이스

파일처리

알고리즘

소프트웨어공학

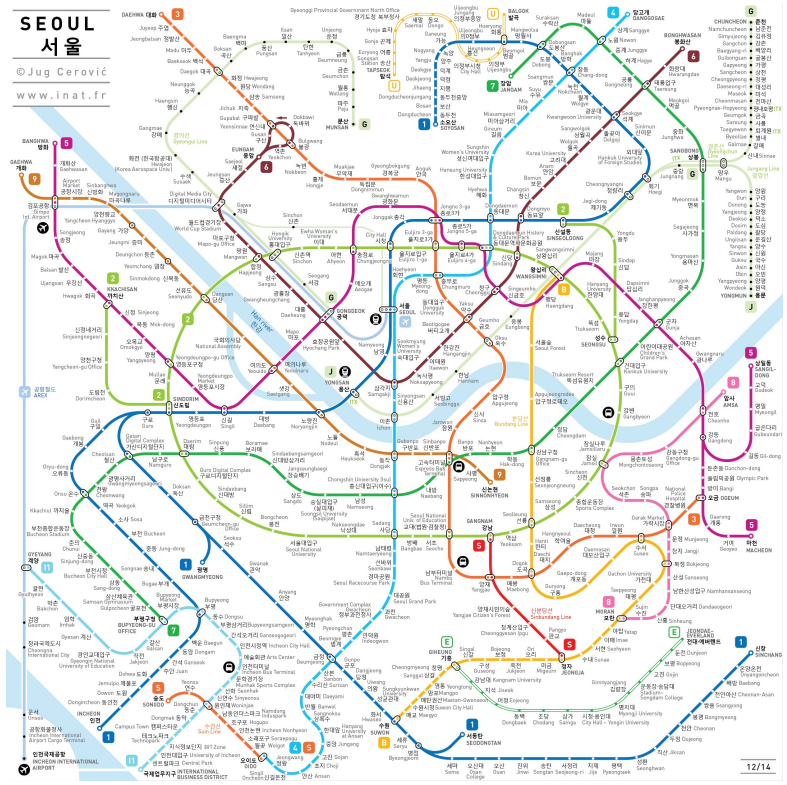
시스템 프로그래밍

네트워크

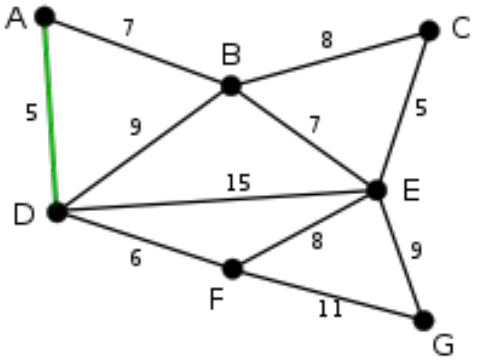
서울 지하철 노선도 앱을 만들어 볼까 ?

Unit 01 | NLP Overview

Logical



Physical



자료구조 : Graph

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

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**

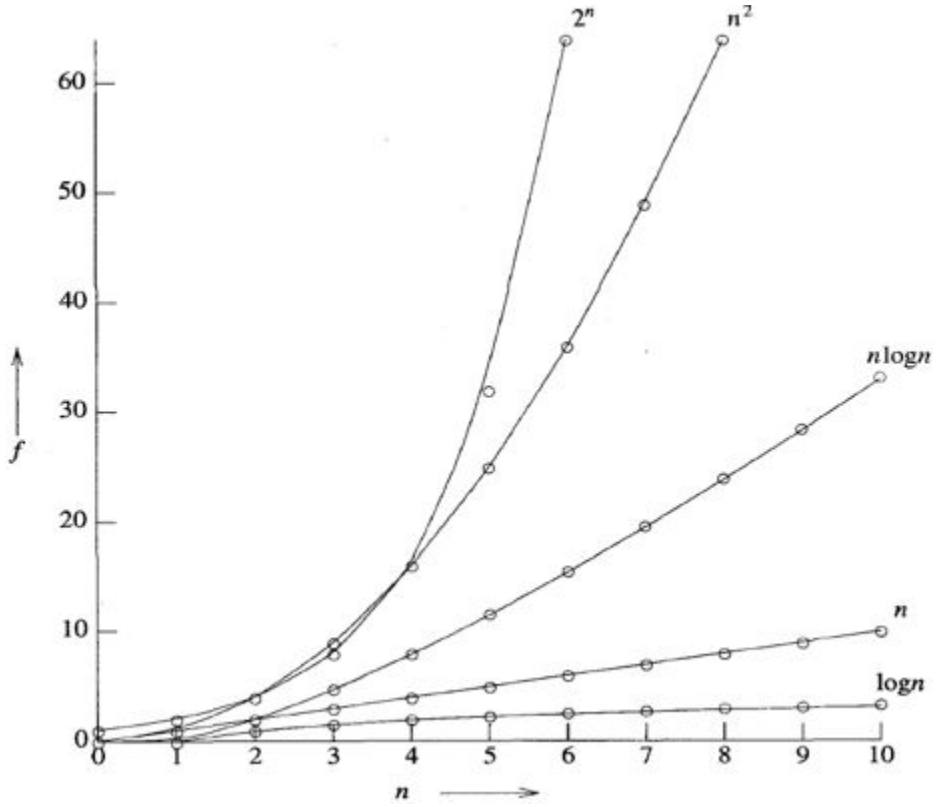
Time Complexity

O(n)

O(nlogn)

O(rowsA*colsA*colsB)

Time Complexity



Time Complexity

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

μs = 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

Time Complexity

list

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

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

Time Complexity

Dict

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

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

Contents

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)

Dynamic Programming

1. Iterative

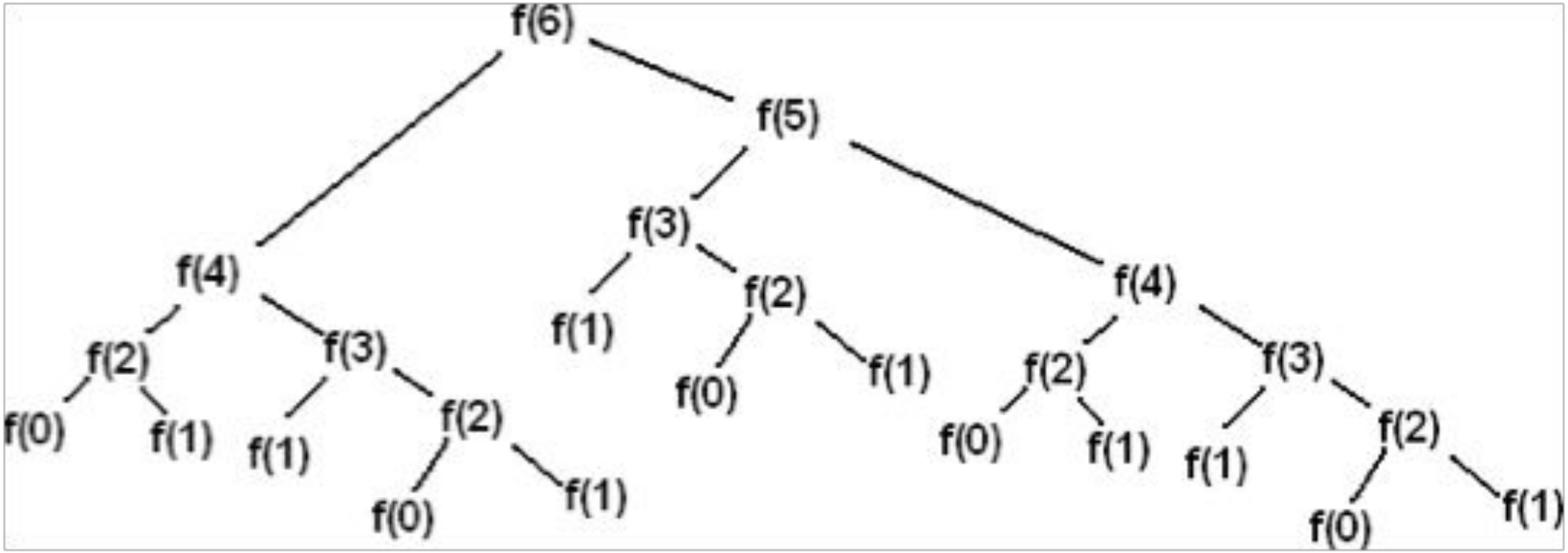
2. Recursive

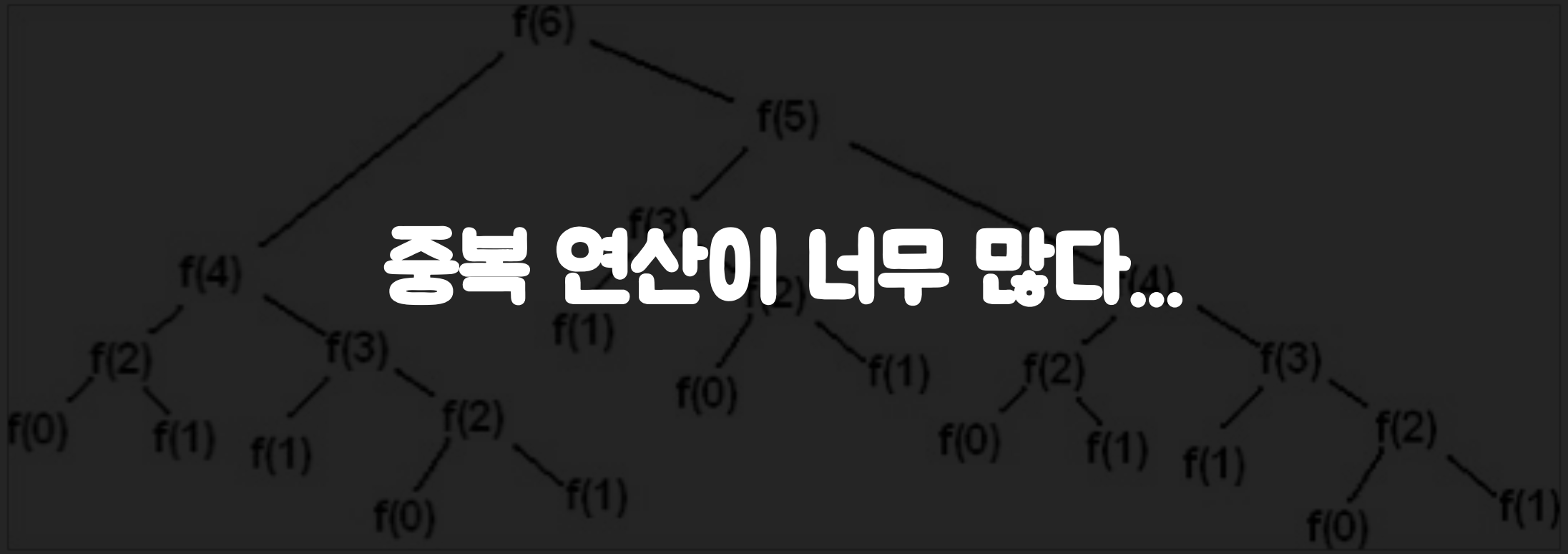
피보나치 수열

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

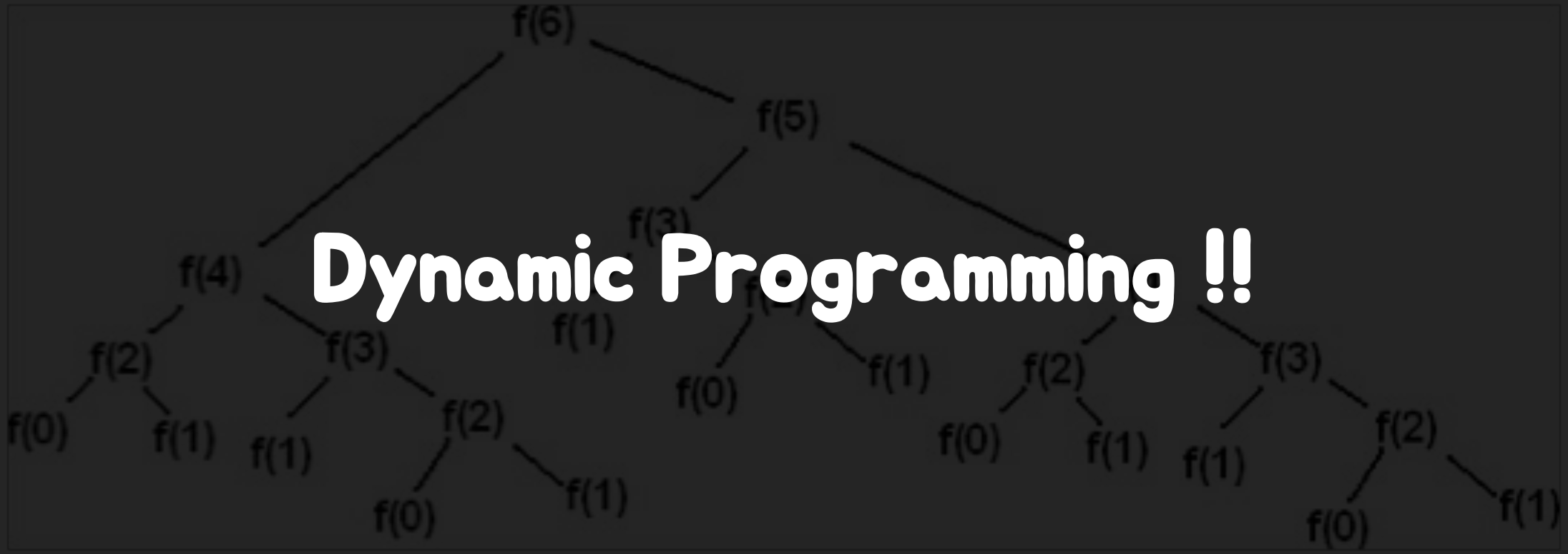
피보나치 수열

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


f(6)
f(5)
f(4)
f(3)
f(4)
f(2)
f(3)
f(1)
f(2)
f(2)
f(3)
f(0)
f(1)
f(1)
f(2)
f(0)
f(1)
f(0)
f(1)
f(1)
f(0)
f(1)
f(0)
f(1)


f(6)
f(5)
f(4)
f(3)
f(2)
f(3)
f(2)
f(0)
f(1)
f(1)
f(2)
f(0)
f(1)
f(0)
f(1)
f(1)
f(0)
f(1)
f(2)
f(0)
f(1)
f(1)
f(2)
f(0)
f(1)

중복 연산이 너무 많다...



Dynamic Programming !!

Dynamic Programming

1. 정의

2. 초기화

3. 점화식(dp식)

Dynamic Programming

1. 정의 dp[i] = i번째 피보나치 수

2. 초기화 dp[0] = 0, dp[1] = 1

3. 점화식 dp[i] = dp[i-1] + dp[i-2]

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
```

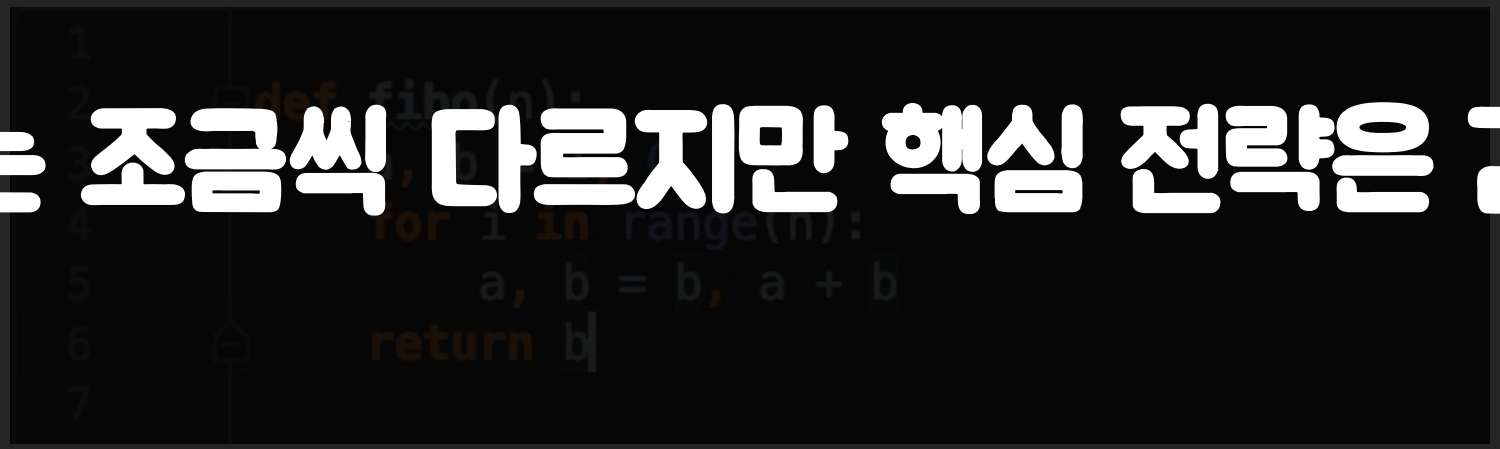
Dynamic Programming

```
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
```

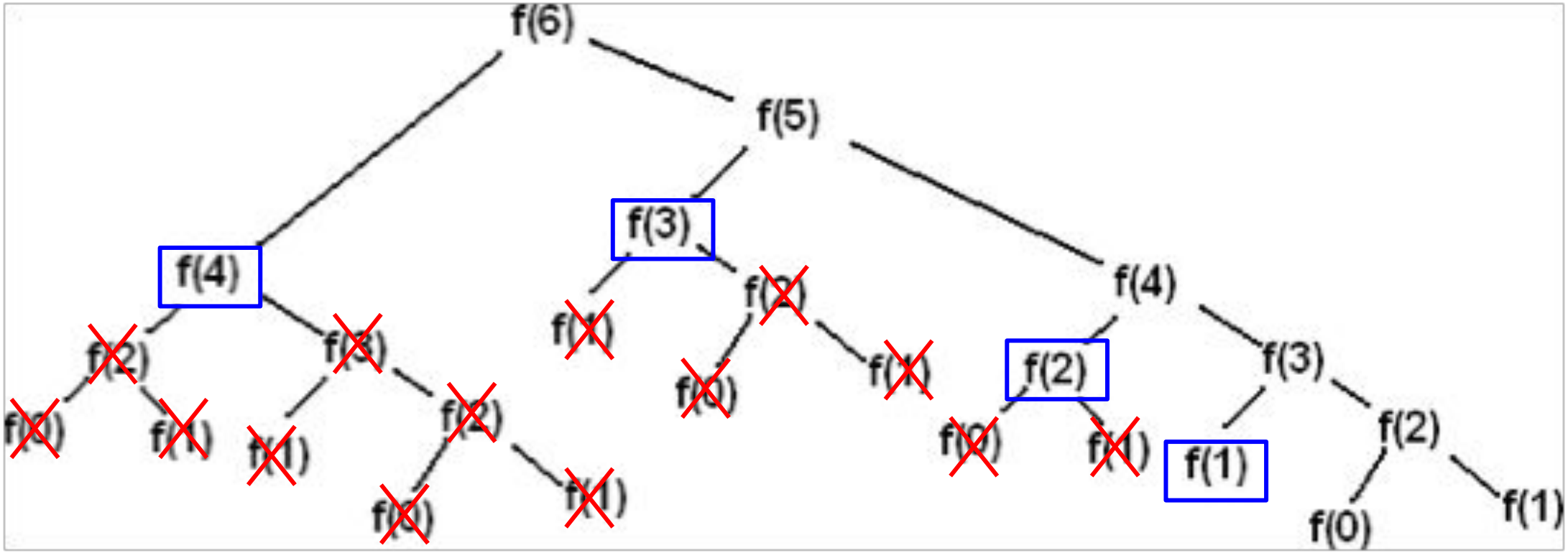
Dynamic Programming

```
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
```

Dynamic Programming



코드는 조금씩 다르지만 핵심 전략은 같다 !


f(6)
f(5)
f(3)
f(4)
f(4)
f(2)
f(3)
f(2)
f(3)
f(2)
f(1)
f(1)
f(1)
f(1)
f(1)
f(1)
f(0)

Time Complexity

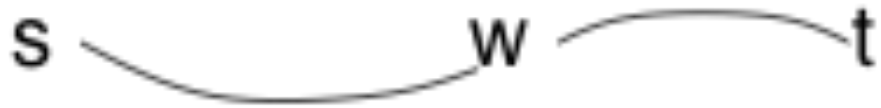
$O(2^n) \rightarrow O(n)$

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 substances

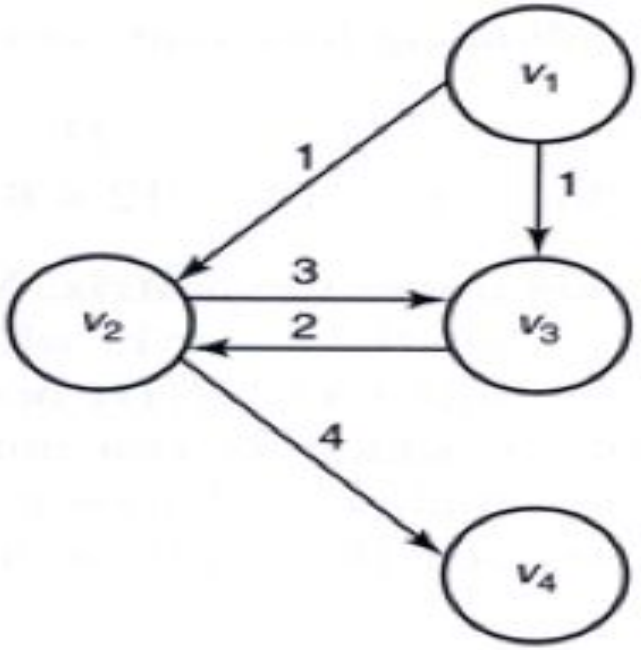
Example) Shortest Path

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.



Dynamic Programming

Exception) Shortest Path



[v1, v3, v2, v4] is the longest simple path.
However, [v1, v3] is not a longest simple path.
[v1, v2, v3] is the longest simple path.

Dynamic Programming

1. 정의

2. 초기화

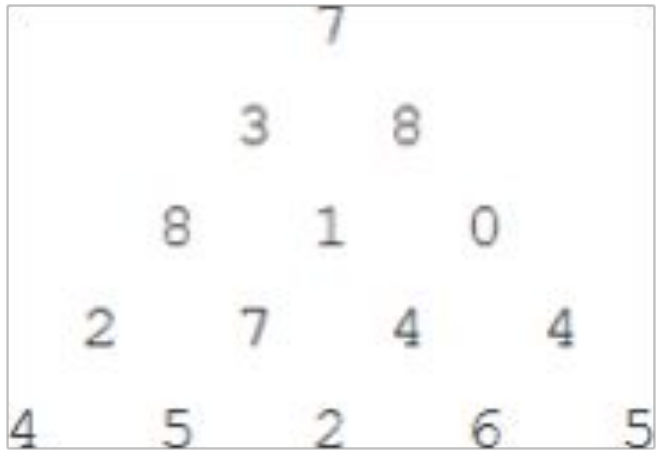
3. 점화식(dp식)

Dynamic Programming

1. 정의
2. 초기화
3. 점화식(dp식)

수학적 귀납법과 닮아있다 !

숫자 삼각형



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

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]

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]

Time Complexity

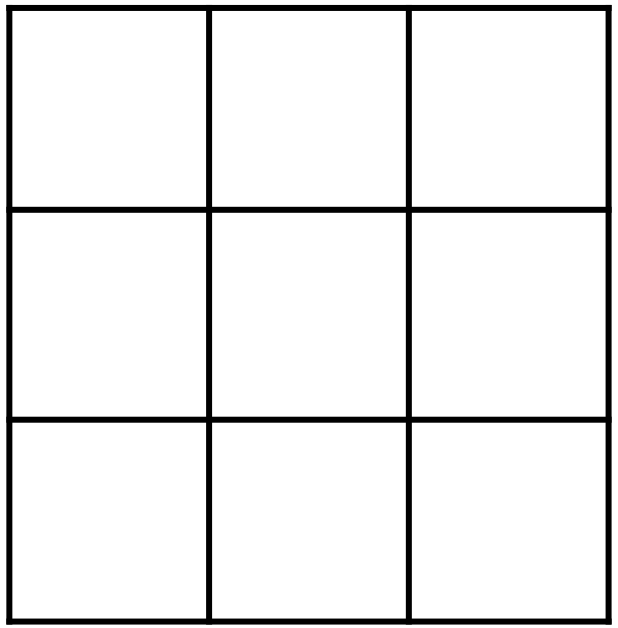
$O(2^n) \rightarrow O(n^2)$

Contents

〈 과제 1〉

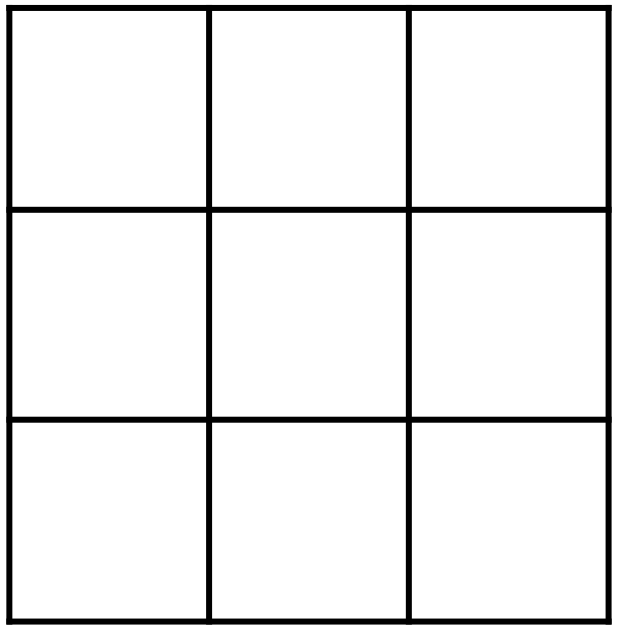
이 그림은 크기가 5인 숫자 삼각형이다. 맨 위층의 숫자 7부터 시작해 아래에 있는 수 중 하나를 선택해, 내려올 때 선택되어 왔던 모든 수의 합이 최대가 되는 경로의 합을 구하는 알고리즘을 완성하시오.

< 과제 2>



투빅이는 평면상 NxM 크기의 집에 있다. 집은 1x1 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.

< 과제 2>



입력은 첫째 줄에 집의 크기 N, M이 주어진다. 둘째 줄부터 N개 줄에는 총 M개의 숫자가 주어지며, r번째 줄의 c번째 수는 (r,c)에 놓여져 있는 금괴의 개수이다.

〈 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.

< 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 = 310이다. 반면에 ((ep1 ep2)(ep3 ep4)) 순서로 합치게 되면 최종파일을 만드는 데에 필요한 비용의 합은 70 + 80 + 150 = 300이다.
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.

〈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.acmicpc.net
- https://www.ics.uci.edu/~pattis/ICS-33/lectures/complexitypython.txt

〈IMG〉

- https://medium.com/@sngho.jeong/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=OpulXJzyHIXn-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..0l10.19089.20208..20413...1.0..1.105.705.3j4.....1....1..gws-wiz-img.KmTzIvn0gaU#imgrc=mtT3VLbCDePXoM:
- https://wayhome25.github.io/python/2017/06/14/time-complexity/

들어주셔서 감사합니다.