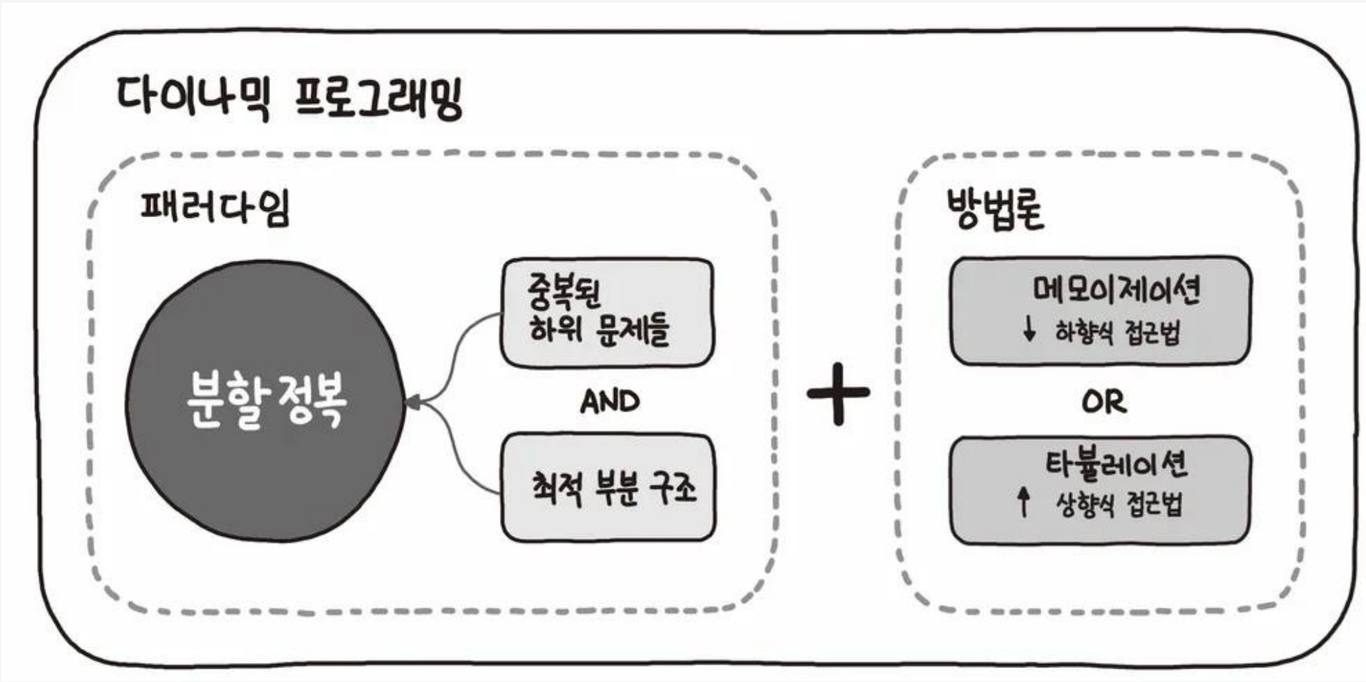
DP

1330574 길태은


그들을
데려와라
'무사히'
넷플릭스 시리즈
D.P.
디피
8월 27일 공개 | NETFLIX

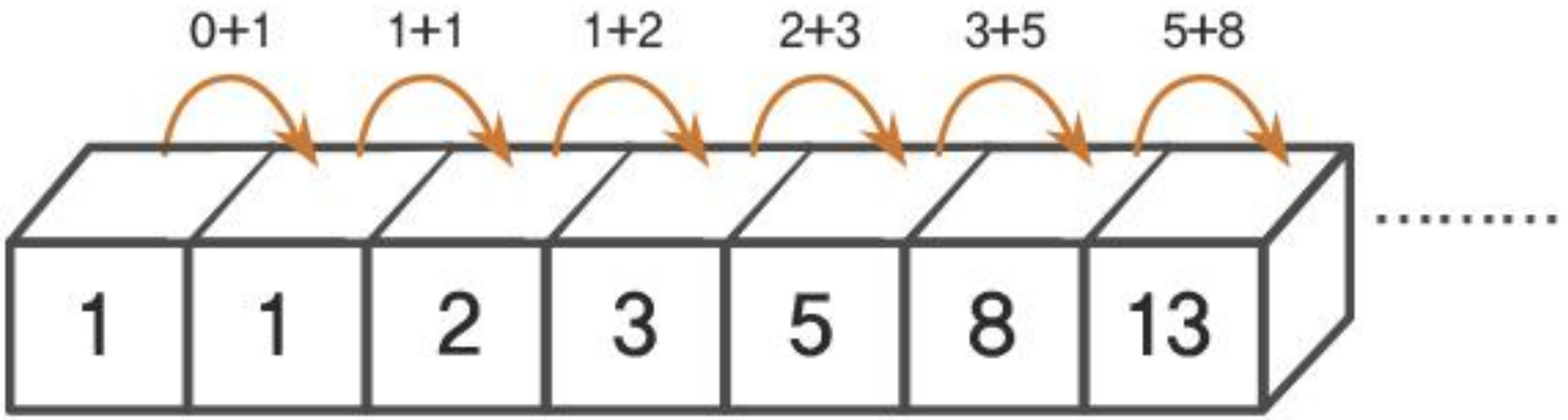
DP - Dynamic Programming 동적 계획법



하위 문제들이 중복되어 있는데
답을 구하기 위해서 했던 계산을
계속해야 하는 종류의 문제의 구조
(최적 부분 구조)

- 답을 기록해서 재활용!

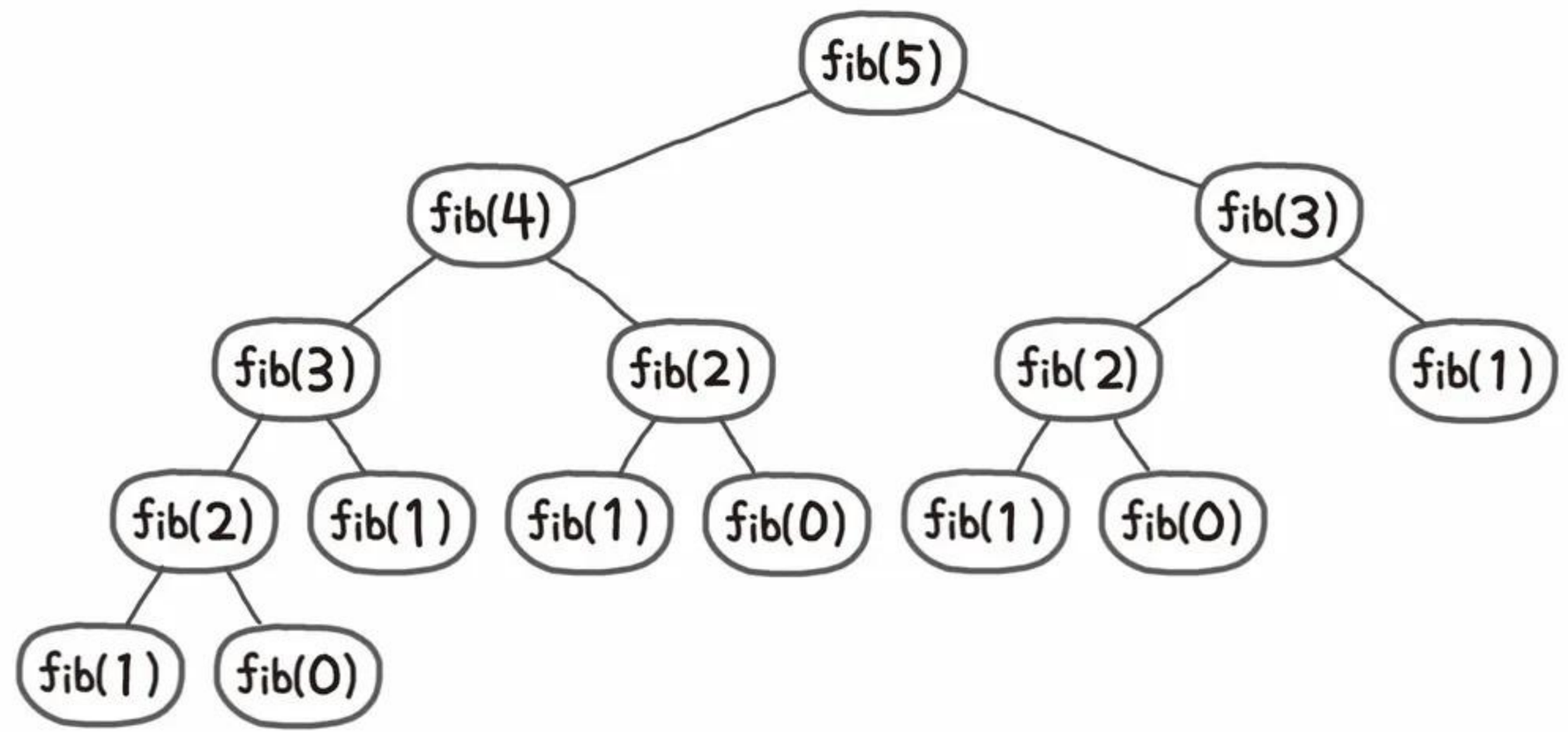
피보나치 수열



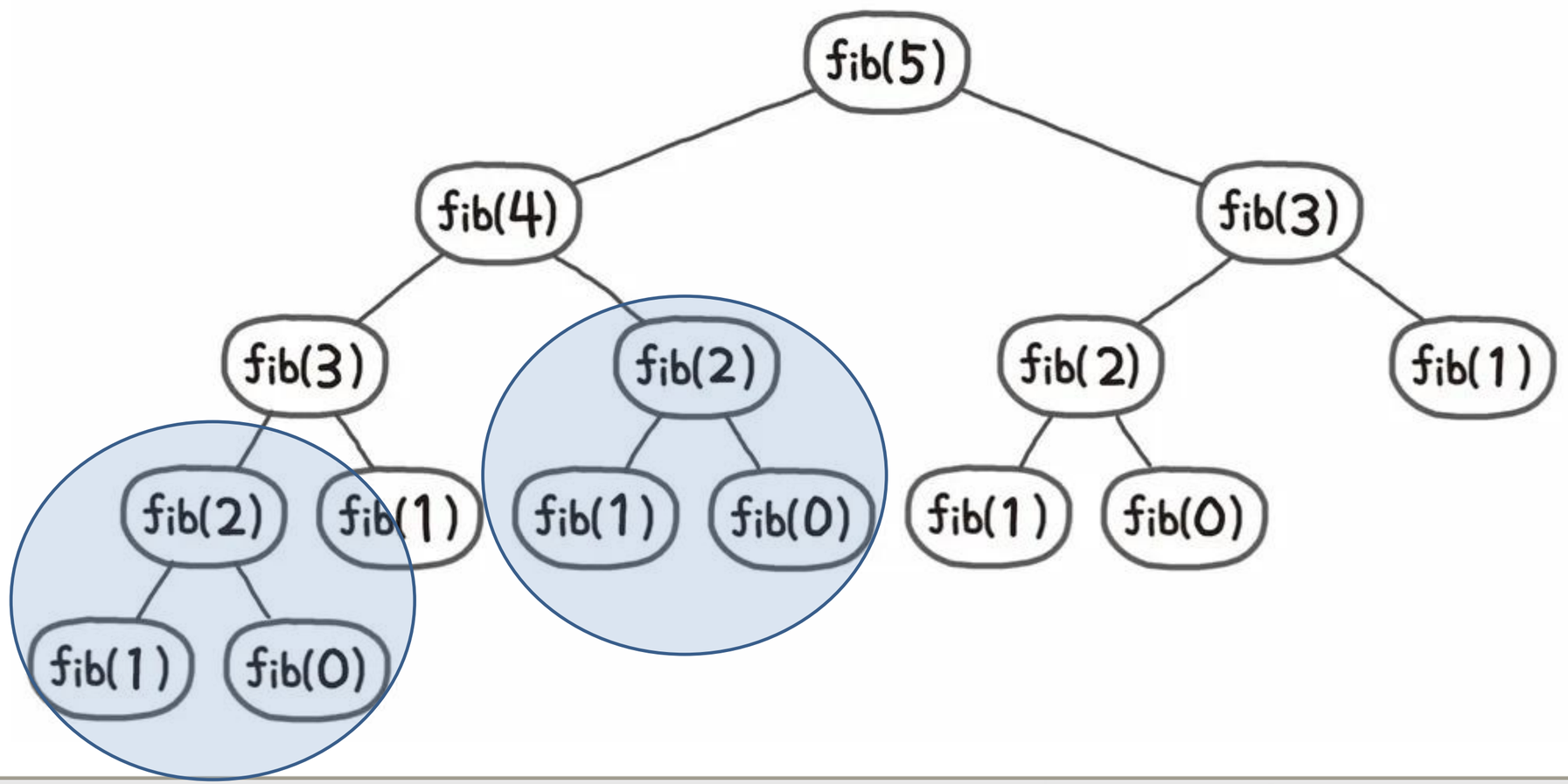
피보나치 수열의 재귀 구조

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

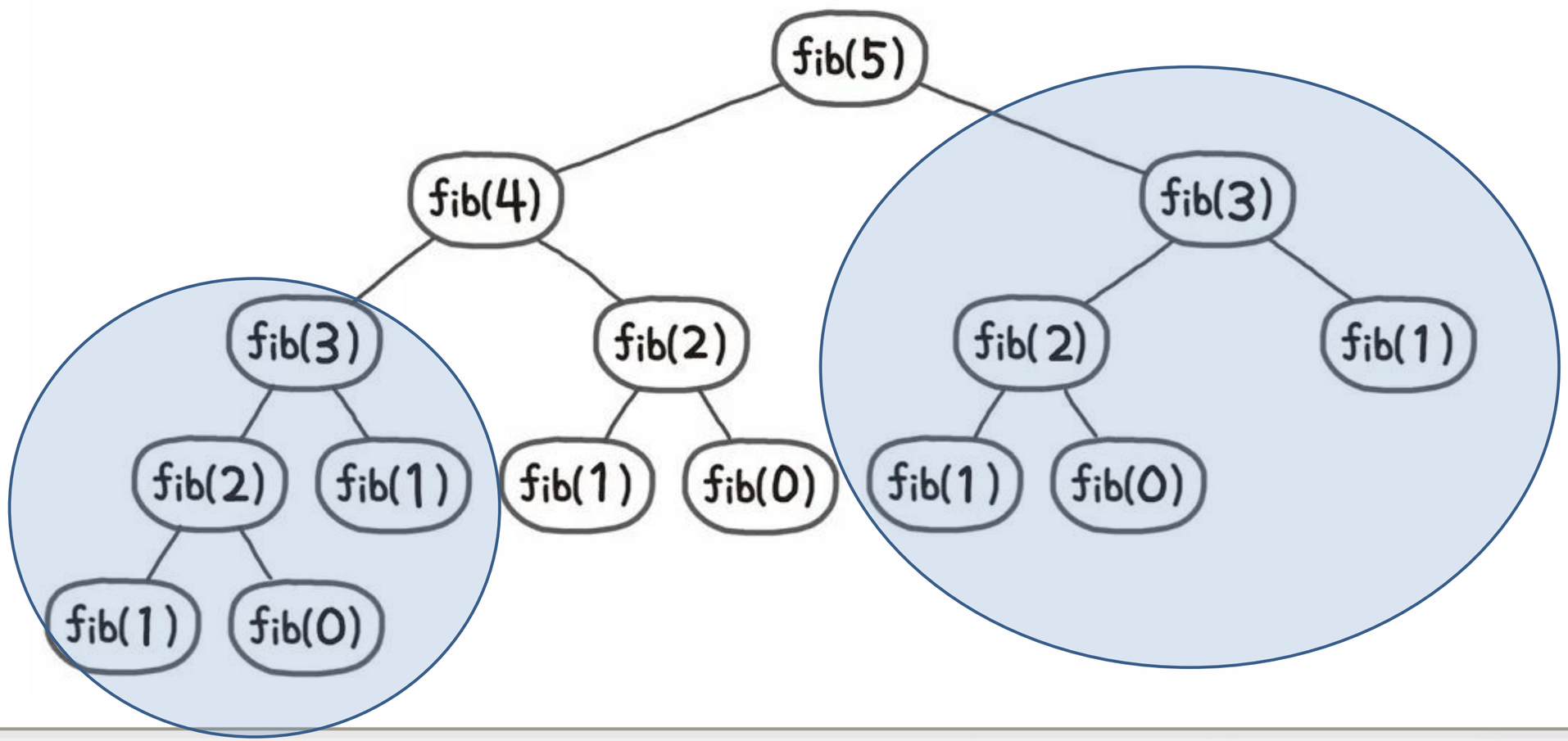
피보나치 수열의 재귀 트리



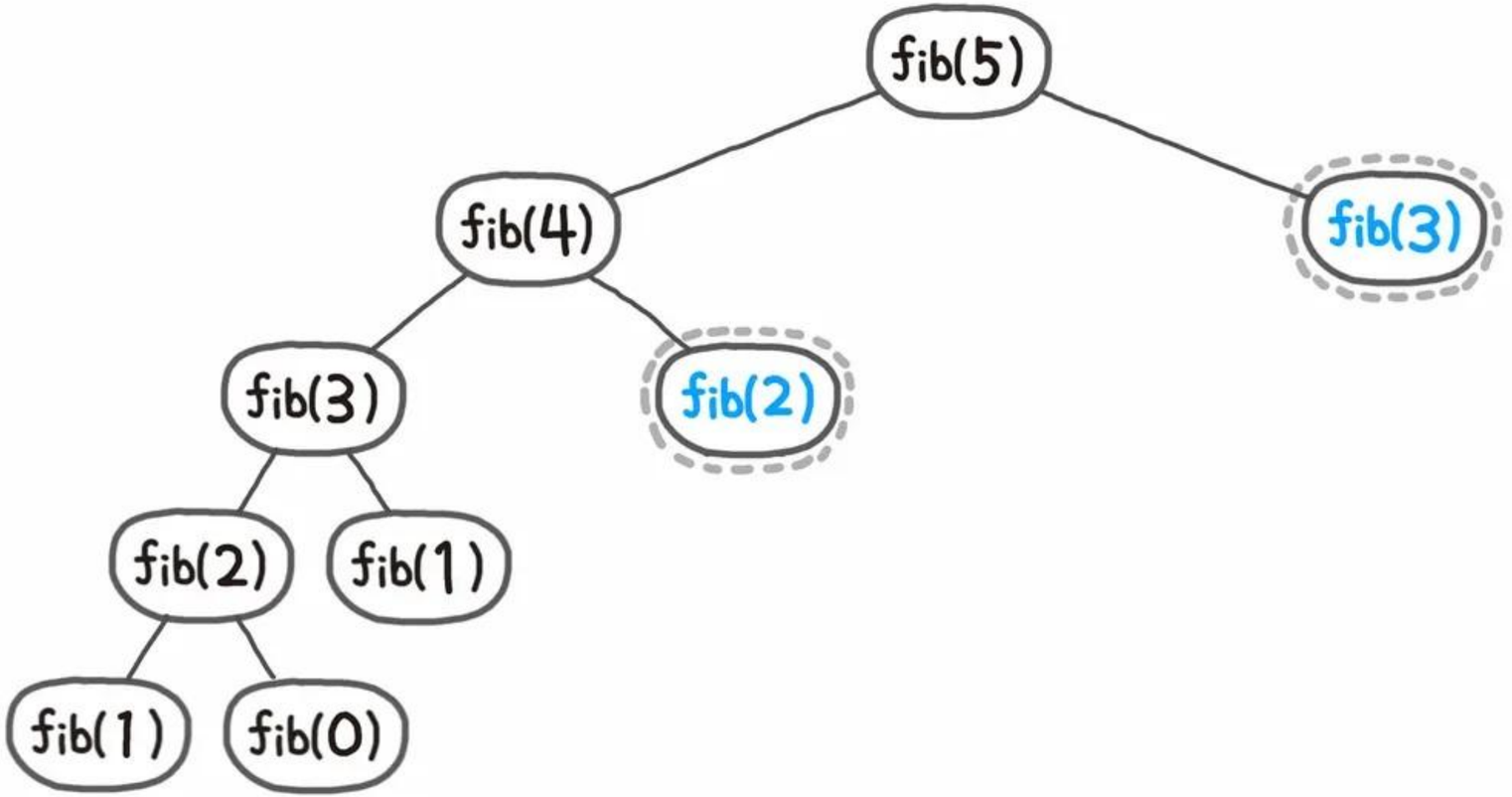
피보나치 수열의 재귀 트리



피보나치 수열의 재귀 트리



피보나치 수열의 재귀 트리

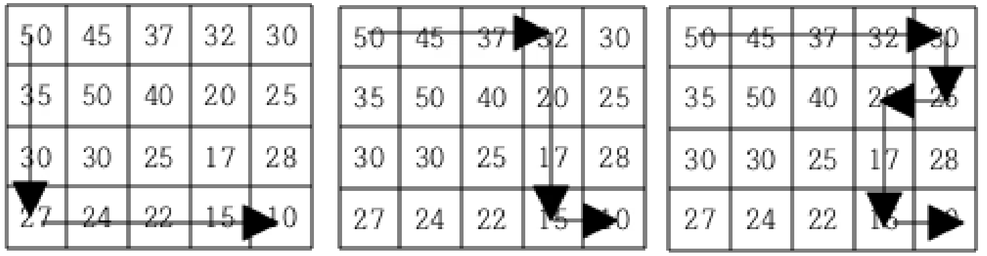


1520번 내리막 길

여행을 떠난 세준이는 지도를 하나 구하였다. 이 지도는 아래 그림과 같이 직사각형 모양이며 여러 칸으로 나뉘어져 있다. 한 칸은 한 지점을 나타내는데 각 칸에는 그 지점의 높이가 쓰여 있으며, 각 지점 사이의 이동은 지도에서 상하좌우 이웃한 곳끼리만 가능하다.

50	45	37	32	30
35	50	40	20	25
30	30	25	17	28
27	24	22	15	10

현재 제일 왼쪽 위 칸이 나타내는 지점에 있는 세준이는 제일 오른쪽 아래 칸이 나타내는 지점으로 가려고 한다. 그런데 가능한 힘을 적게 들이고 싶어 항상 높이가 더 낮은 지점으로만 이동하여 목표 지점까지 가고자 한다. 위와 같은 지도에서는 다음과 같은 세 가지 경로가 가능하다.



지도가 주어질 때 이와 같이 제일 왼쪽 위 지점에서 출발하여 제일 오른쪽 아래 지점까지 항상 내리막길로만 이동하는 경로의 개수를 구하는 프로그램을 작성하시오.

DFS?

```python
M, N = map(int, input().split())
matrix = [list(map(int, input().split())) for _ in range(M)]
count = 0
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

def dfs(x, y, M, N, matrix):
    global count
    if x == M - 1 and y == N - 1:
        count +=1
        return

    for i in range(4):
        nx, ny = x + dx[i], y + dy[i]
        if 0 <= nx < M and 0 <= ny < N and matrix[nx][ny] < matrix[x][y]:
            dfs(nx, ny, M, N, matrix)

dfs(0, 0, M, N, matrix)
print(count)
```

DFS?

문제	결과
3 1520	시간 초과

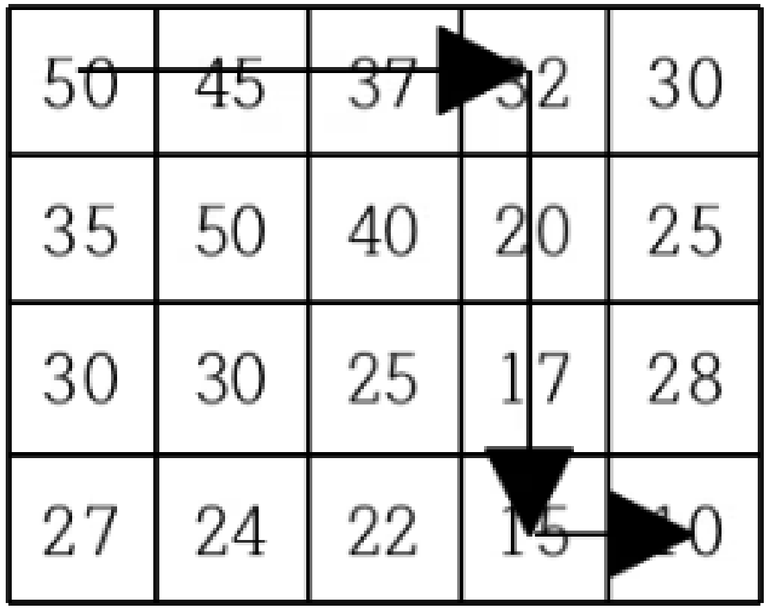
입력

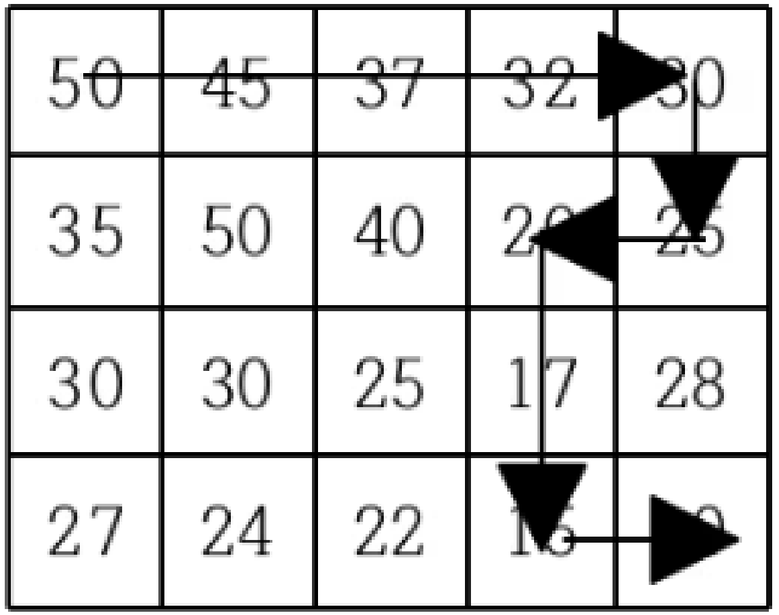
첫째 줄에는 지도의 세로의 크기 M과 가로의 크기 N이 빈칸을 사이에 두고 주어진다. 이어 다음 M개 줄에 걸쳐 한 줄에 N개씩 위에서부터 차례로 각 지점의 높이가 빈 칸을 사이에 두고 주어진다. M과 N은 각각 500이하의 자연수이고, 각 지점의 높이는 10000이하의 자연수이다.

$O(4^{N*M})$

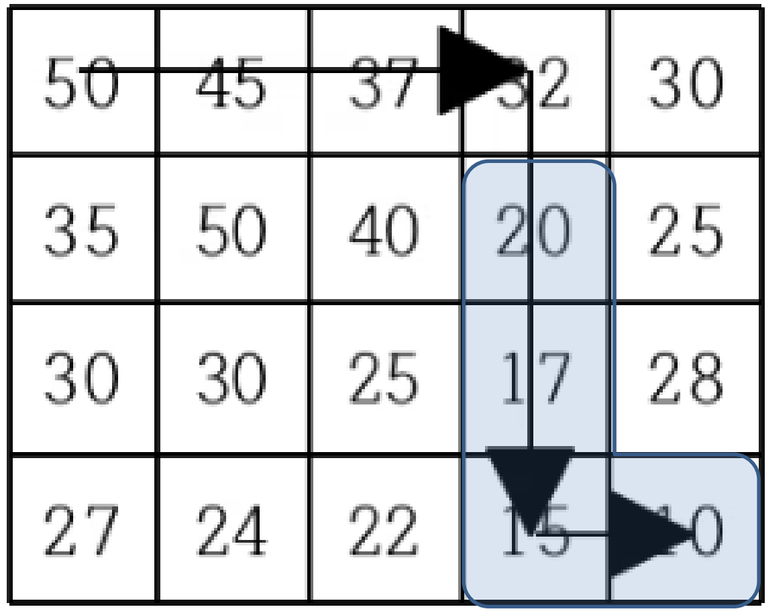
정답 비율
28.396%

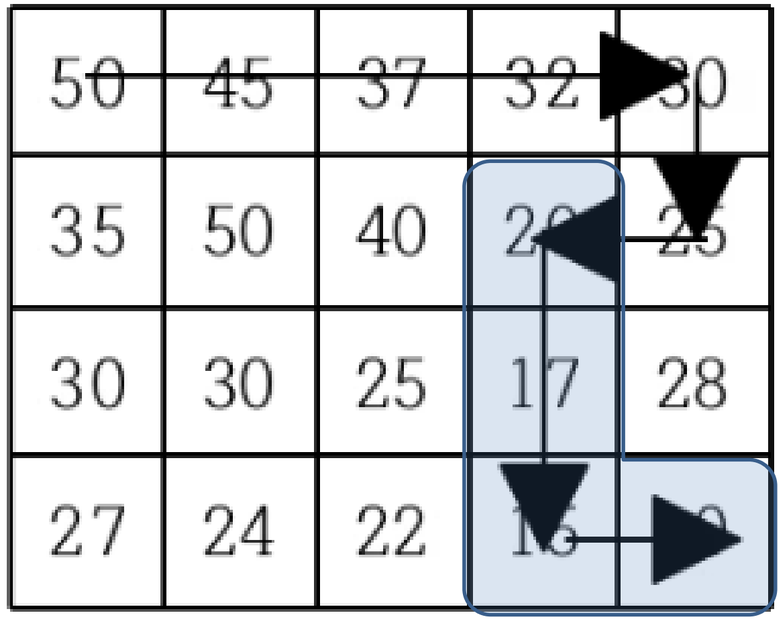
예시


50 45 37 32 30
35 50 40 20 25
30 30 25 17 28
27 24 22 15 10

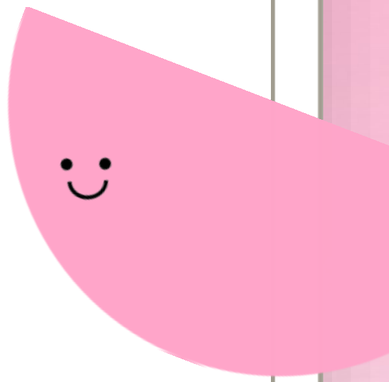

50 45 37 32 30
35 50 40 25
30 30 25 17 28
27 24 22

예시


50 45 37 32 30
35 50 40 20 25
30 30 25 17 28
27 24 22 15 10


50 45 37 32 30
35 50 40 20 25
30 30 25 17 28
27 24 22 15 10

```python
N, M = map(int, input().split())
graph = [list(map(int, input().split())) for _ in range(N)]

dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

dp = [[-1] * M for _ in range(N)]

def dfs(x, y):
    # 이미 방문한 적 있다면 그 값을 바로 반환
    if dp[x][y] != -1:
        return dp[x][y]

    dp[x][y] = 0  # 방문 표시 대용

    # 도착점에 도달하면 1을 반환
    if x == N-1 and y == M-1:
        return 1

    for i in range(4):
        nx, ny = x + dx[i], y + dy[i]
        if 0 <= nx < N and 0 <= ny < M and graph[nx][ny] < graph[x][y]:
            dp[x][y] += dfs(nx, ny)

    return dp[x][y]

dfs(0, 0)
print(dp[0][0])
```



```
[[3, 2, 2, 2, 1],
 [1, -1, -1, 1, 1],
 [1, -1, -1, 1, -1],
 [1, 1, 1, 1, 0]]
```

5215. 햄버거 다이어트 D3

평소 햄버거를 좋아하던 민기는 최근 부쩍 늘어난 살 때문에 걱정이 많다.

그렇다고 햄버거를 포기할 수 없었던 민기는 햄버거의 맛은 최대한 유지하면서 정해진 칼로리를 넘지 않는 햄버거를 주문하여 먹으려고 한다.

민기가 주로 이용하는 햄버거 가게에서는 고객이 원하는 조합으로 햄버거를 만들어서 준다.

하지만 재료는 미리 만들어서 준비해놓기 때문에 조합에 들어가는 재료를 잘라서 조합해주지는 않고, 재료를 선택하면 준비해놓은 재료를 그대로 사용하여 조합해준다.

민기는 이 가게에서 자신이 먹었던 햄버거의 재료에 대한 맛을 자신의 오랜 경험을 통해 점수를 매겨놓았다.

민기의 햄버거 재료에 대한 점수와 가게에서 제공하는 재료에 대한 칼로리가 주어졌을 때,

민기가 좋아하는 햄버거를 먹으면서도 다이어트에 성공할 수 있도록 정해진 칼로리 이하의 조합 중에서 민기가 가장 선호하는 햄버거를 조합해주는 프로그램을 만들어보자.

(단 여러 재료를 조합하였을 햄버거의 선호도는 조합된 재료들의 맛에 대한 점수의 합으로 결정되고, 같은 재료를 여러 번 사용할 수 없으며, 햄버거의 조합의 제한은 칼로리를 제외하고는 없다.)

[입력]

첫 번째 줄에 테스트 케이스의 수 T가 주어진다.

각 테스트 케이스의 첫 번째 줄에는 재료의 수, 제한 칼로리를 나타내는 N, L($1 \le N \le 20$, $1 \le L \le 10^4$)가 공백으로 구분되어 주어진다.

다음 N개의 줄에는 재료에 대한 민기의 맛에 대한 점수와 칼로리를 나타내는 T_i, K_i($1 \le T_i \le 10^3$, $1 \le K_i \le 10^3$)가 공백으로 구분되어 주어진다.

```
1                   //Test Case 갯수
5 1000              //Test Case #1, N = 5, L = 1000
100 200             //Test Case #1, 첫번째 재료 정보
300 500
250 300
500 1000
400 400
```

[출력]

각 줄마다 "#T" (T는 테스트 케이스 번호)를 출력한 뒤, 주어진 제한 칼로리 이하의 조합중에서 가장 맛에 대한 점수가 높은 햄버거의 점수를 출력한다.

모든 조합을 구해서

제한 칼로리 이상은
계산하지 말고

제한 칼로리 이하면
점수가 현재 답보다 높을 때
갱신하자!

```python
from itertools import combinations

T = int(input())
for tc in range(1, T+1):
    N, L = map(int, input().split())
    ingredients = [list(map(int, input().split())) for _ in range(N)]

    ans = 0

    for i in range(1, N+1):
        for comb in combinations(ingredients, i):
            sum_of_scores = 0
            sum_of_calories = 0
            for score, calorie in comb:
                sum_of_scores += score
                sum_of_calories += calorie
            if sum_of_calories > L:
                continue
            ans = max(ans, sum_of_scores)

    print(f'#{tc} {ans}')
```

DP를 쓰면

```python
T = int(input())

for tc in range(1, T + 1):
    N, L = map(int, input().split())
    taste_calorie = [list(map(int, input().split())) for _ in range(N)]
    dp = [0] * (L + 1)
    for taste, calorie in taste_calorie:
        for i in range(L, calorie - 1, -1):
            dp[i] = max(dp[i], dp[i - calorie] + taste)
            print(dp)
    print(f"#{tc} {max(dp)}")
```

최대 칼로리 -> 현재 재료 칼로리

이미 저장된 점수 dp[i] vs

현재 재료 칼로리를 뺀 점수 dp[I – calorie]에
현재 재료 점수 taste를 더한 합
더 큰 값을 갱신 -> 제한 칼로리 내 최대 점수만 기록해갈 수 있음!

Thank you!