숫자 만들기

n일 때의 경우의 수를 A(n)이라고 하면

n = 1) A(1) = 1

1

n = 2) A(2)=2

1 + 1

2

n = 3) A(3)=4

1 + 1 + 1

2 + 1

1 + 2

3

n = 4) A(4)=A(3)+A(2)+A(1)

n=3일 때의 각 경우에 숫자1을 더하면, n= 4 인 경우를 만족합니다.

n=2일 때의 각 경우에 숫자2을 더하면, n= 4 인 경우를 만족합니다.

n=1일 때의 각 경우에 숫자3을 더하면, n= 4 인 경우를 만족합니다.

따라서, A(4)를 구할 때 모든 경우의 수를 일일이 구하는 것이 아니라, 이미 구해놨던 A(1), A(2), A(3)의 결과값을 이용하는 거죠. 기록해 놨다가 재사용하여 계산의 번거로움 즉, 계산의 반복을 피할 수 있습니다.

만약 A(1), A(2), A(3)의 결과를 재사용 하지 않는다면, A(4)를 구하는 과정에서 불필요하게 A(1), A(2), A(3)의 경우의 수를 계산하는 꼴이 됩니다.

정리

A(1) A(2) A(3) 의 값은 사람이 직접 구해야하는 기본값들이라 일일이 손으로 값을 알아야 하고,

A(n) (n>=4) 부터는 **A(n-1) + A(n-2) + A(n-3) = A(n)** 의 점화식으로 A(n)의 결과값을 계속 저장하고 재사용하는 것입니다.(Bottom up 방식: n이 커지는 방식)

저장을 해야하니, 배열을 사용하는 것이 좋습니다.

DP는 한마디로 정리하면, **저장해놓은 값을 재활용해서 연산을 줄이자!(메모리공간을 더 쓰고, CPU연산을 줄이자)** 라고 말할 수 있습니다.

참고)

알고리즘 연산을 할 때, DP를 사용하는 기준은 컴퓨터의 초당연산횟수를 초과하냐 아니냐로 결정된다고 합니다. CPU는 **1억번/초 (10의 8승)**의 연산을 할 수 있습니다. 따라서, 이중 for문의 각 for문이 **10의 4승**을 초과하는 숫자라면, 내부 실행문들과 총연산횟수를 합해서 계산했을 때 1억을 가뿐히 넘기므로 반드시 DP(배열같은 메모리공간에 기록하고 연산은 줄이고)로 풀어야 합니다.

참고)

A(n-1) + A(n-2) + A(n-3) **=** A(n) 이 아닌 **A(n) = A(n-1) + A(n-2) + A(n-3)** 으로 점화식을 사용하는 것은 (Top down방식: n이 줄어드는 방식) 재귀함수에서 쓸 수 있는 **분할 정복**입니다.