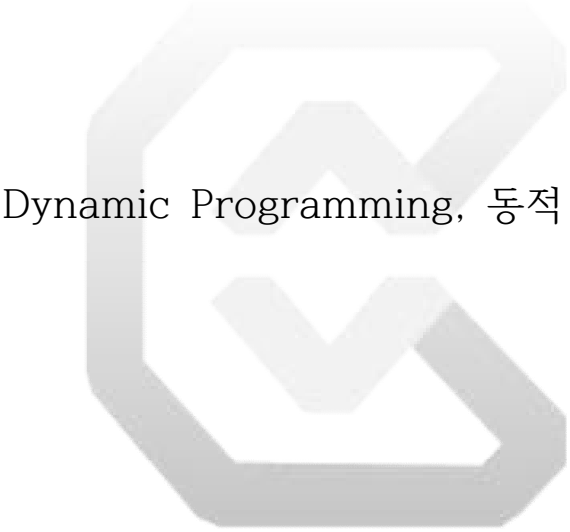
CAPS

C언어 스터디
7.0주차

<DP (Dynamic Programming, 동적 계획법)>

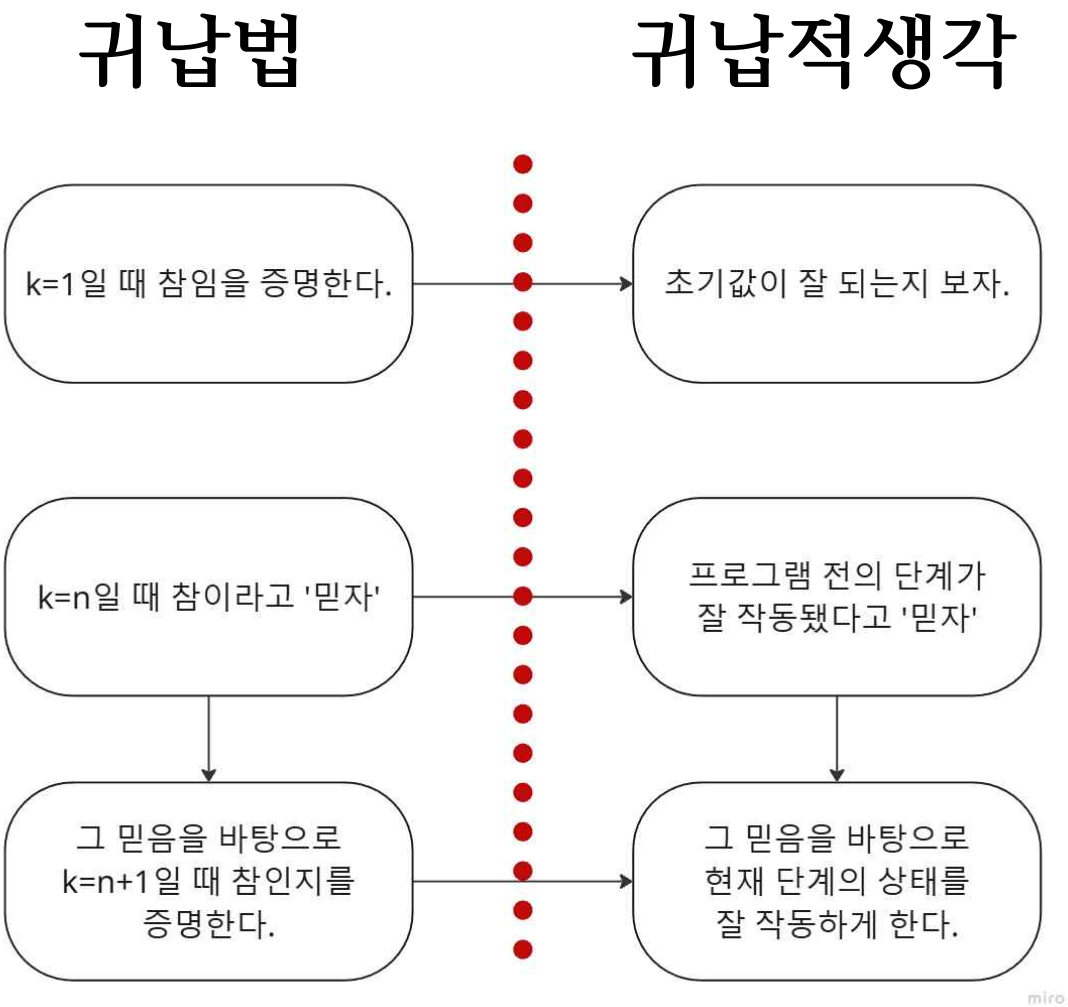


CAPS

귀납적 생각

연결리스트를 잘 하시게 된다면, 향후 다룰 재귀함수 등에 큰 도움이 될 겁니다. 아니? 사실 재귀함수를 잘하시면 연결리스트를 잘하게 될 겁니다. 아니? 사실 이 모든 걸 아우르는 하나의 능력이 있습니다.

연결리스트에서의 중요한 건 '귀납적인 생각'입니다. dp, 재귀 모두 '귀납적인 생각'이 중요하거든요. 사실 알고리즘을 배우실 때 가장 처음 만나고 가장 어려운 벽이 귀납적인 생각일 겁니다.



또 펙토리얼

```
#include<iostream>
using namespace std;
long long factorial[20];
int main(void) {
	factorial[0] = 1; // 초기값을 잘 세팅하고
	for (int i = 1; i <= 20; i++) {
		long long tmp = factorial[i - 1]; // 전의 계산이 잘 맞다고 '믿자'
		factorial[i] = i * tmp; //그럼 현재 상태를 잘 계산하자.
	}

	cout << factorial[5] << "\n"; //120
	cout << factorial[2] << "\n"; //2
}
```

수학

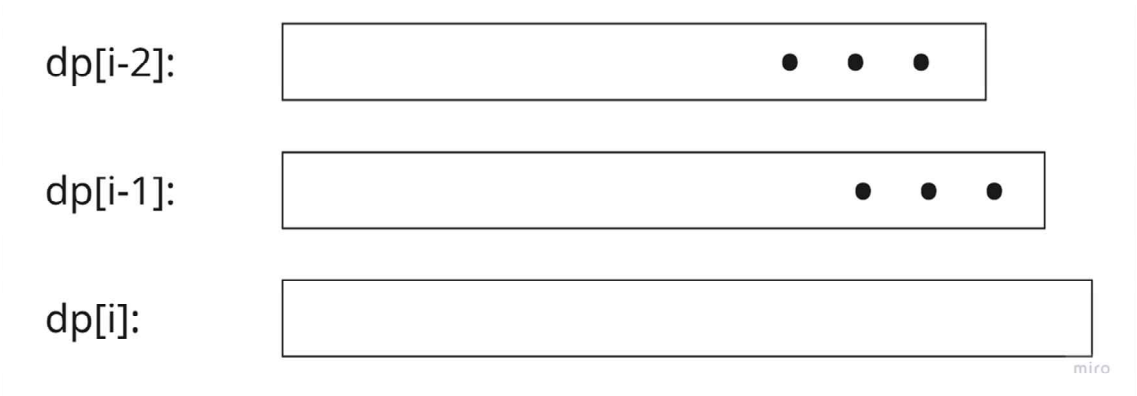
mod (% 연산) 은 곱하기와 더하기에 대해서 분배법칙이 성립합니다.
즉,

$$(a + b \times c)\%k$$
$$= a\%k + (b \times c)\%k$$
$$= a\%k + (b\%k) \times (c\%k)$$

이 성립합니다. 오버플로우를 방지하기 위해 매번 더하기와 곱하기마다 계속 mod 처리를 해 주세요!

문제를 풀어보자.

https://www.acmicpc.net/problem/11726



어떤 n이 i인 상태에서, i-1과 i-2 의 경우의 수가 잘 저장되어 있다고 '믿자'.

그러면 i인 상태를 만들기 위해서는
1. i-2에서 가로로 두 줄
2. i-1에서 세로 한 줄
이 있다면 I인 상태를 만들 수 있다.
"여기서 i-2에서 세로 두 줄인 상태는요?"
좋은 질문입니다. 하지만 해당 경우는 dp[i-1]에 끝에 한 줄을 추가한 경우에 포함됩니다.
즉,

$$dp[i] = dp[i-1] + dp[i-2]$$

다음과 같은 수식이 만들어집니다. dp를 풀 때 모두 딱딱하게 수식을 쓸 필요는 없지만, '과거 상태'가 '현재 상태'를 만들어 내는 걸 목표로 합시다.

```
#include<iostream>
using namespace std;
int DP[1001];
int main(void) {
	ios::sync_with_stdio(0);
	cin.tie(0);
	cout.tie(0);
	DP[1] = 1; DP[2] = 2;
	for (int i = 3; i < 1001; i++)DP[i] = (DP[i - 1] + DP[i - 2]) % 10007;
	int tmp; cin >> tmp; cout << DP[tmp];
}
```

https://www.acmicpc.net/problem/2748

이 문제는 아예 식을 주네요?

```
#include<iostream>
using namespace std;
long long dp[91];
int main(void) {
	ios::sync_with_stdio(0);
	cin.tie(0);
	dp[1] = 1;
	dp[2] = 1;
	for (int i = 3; i <= 90; i++)dp[i] = dp[i - 1] + dp[i - 2];
	int N; cin >> N;
	cout << dp[N];
}
```

https://www.acmicpc.net/problem/2579

이 문제는 중요합니다. dp를 조금 더 똑똑하게 써 봅시다.

참고로 max(a,b), min(a,b) 라는 함수를 자주 쓰게 될 겁니다. 잘 익혀봅시다.

dp는 단순히 1차원으로 국한되지 않습니다.

2차원으로 만들어서 '물통' 문제와 같이 어떤 상태를 index를 통해 표현해 낼 수 있습니다.

계단 오르기 문제를 1차원dp로 푼 사람들이 많지만, 2차원으로 dp를 구성해 푼 풀이를 드리겠습니다.

```cpp
#include<iostream>
#define ll long long
using namespace std;
ll dp[2][301];
int arr[301];
int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N;
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> arr[i];
    dp[0][1] = arr[1];
    dp[1][2] = arr[2] + arr[1];
    dp[0][2] = arr[2];
    for (int i = 3; i <= N; i++) {
        dp[0][i] = max(dp[0][i - 2], dp[1][i - 2]) + arr[i];
        dp[1][i] = dp[0][i - 1] + arr[i];
    }
    cout << max(dp[1][N], dp[0][N]);
}
```

dp는 재귀, 수학, 그리디 등과 같이 여러분들을 계속 쫓아다닐 유형입니다. 아주 기초적인 dp는 끝났으니, 그래도 적어도 여러분들은 뭘 해도 잘 해낼 수 있을 겁니다.

문제)

1. 10872 팩토리얼
   https://www.acmicpc.net/problem/10872
   https://github.com/sungjoonyoung/BOJ/blob/main/10000/10872_1.cpp
2. 11726 2×n 타일링
   https://www.acmicpc.net/problem/11726
   https://github.com/sungjoonyoung/BOJ/blob/main/10000/11726.cpp
3. 2748 피보나치 수 2
   https://www.acmicpc.net/problem/2748
   https://github.com/sungjoonyoung/BOJ/blob/main/00000/2748.cpp
4. 9095 1, 2, 3 더하기
   https://www.acmicpc.net/problem/9095
   https://github.com/sungjoonyoung/BOJ/blob/main/00000/9095.cpp
5. 11727 2×n 타일링 2
   https://www.acmicpc.net/problem/11727

https://github.com/sungjoonyoung/BOJ/blob/main/10000/11727.cpp

6. 2579 계단 오르기
   https://www.acmicpc.net/problem/2579
   https://github.com/sungjoonyoung/BOJ/blob/main/00000/2579.cpp
7. 1149 RGB거리
   https://www.acmicpc.net/problem/1149
   https://github.com/sungjoonyoung/BOJ/blob/main/00000/1149.cpp
8. 11051 이항 계수 2 (nCr = n-1Cr-1 + n-1Cr 인 거 기억하시나요?)
   https://www.acmicpc.net/problem/11051
   https://github.com/sungjoonyoung/BOJ/blob/main/10000/11051.cpp

(기초 언어)

1. 클래스 1 밀기
https://solved.ac/class?class=1

(자료 구조)

2. 바킹독 알고리즘 블로그 0x00~0x08강 보면서 클래스 2 밀기
https://blog.encrypted.gg/category/%EA%B0%95%EC%A2%8C/%EC%8B%A4%EC%A0%84%20%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98?page=2

(알고리즘)

3. 클래스3부터 쭉 밀면서, doit 알고리즘 책이랑 바킹독 블로그 같이 병행하기
https://product.kyobobook.co.kr/detail/S000217191101