챕터 5. Dynamic Programming (동적 계획법) - (1,2) 동적계획과 이항계수

동적계획법: Dynamic Programming

- 문제를 더 작은 문제로 분할하되, 상향식으로 문제를 해결한다.
- 1953년, Richard Bellman 교수가 제안
- Programming: 여기서는 '계획'을 의미
    - TV프로그램, 오늘 행사의 프로그램 안내.
- Memoization: 메모이제이션
    - 가장 작은 입력 사례의 해답을 테이블에 저장하고 필요할 때 꺼내 쓴다.

*참고 자료

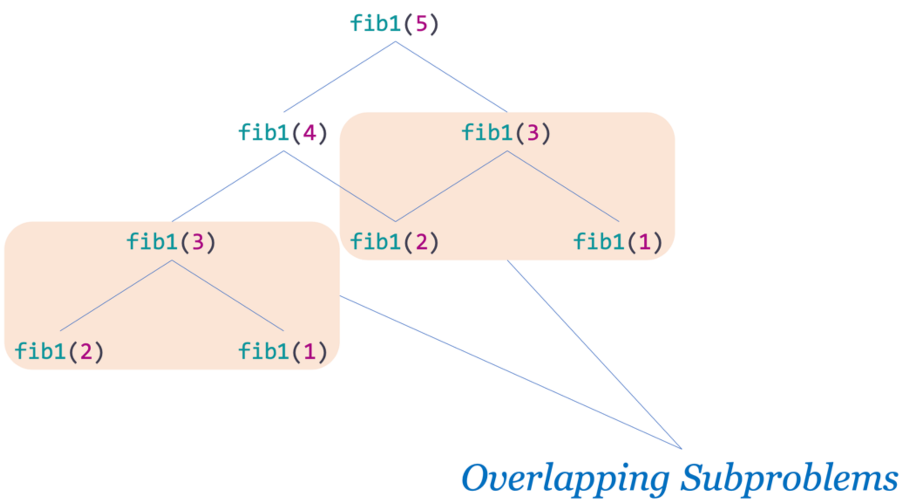
동적계획법으로 문제 풀기

1. 문제를 해결할 수 있는 재귀 관계식을 구한다.
2. 가장 작은 입력사례로부터 상향식 방법으로 문제를 해결한다.

분할정복법 vs 동적 계획법

- 문제를 작은 사례로 분할하여 해결한다는 점에서 동일
- 분할정복: 재귀 호출을 통해 분할하여 정복(Top-Down)
- 동적계획: 메모이제이션을 통해 상향식으로 정복(Bottom-up)

    ex) 피보나치 수열



이항 계수 문제

- 이항 계수의 정의

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}, \text{ for } 0 \le k \le n.$$

- 문제점: n!, k!의 값은 매우 크기 때문에 계산이 어렵다.

이항 계수의 재귀적 정의: 분할정복(Divide-and-Conquer)

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \text{ or } k = n \end{cases}$$

이항계수 (Divide-and-Conquer)

Algorithm 3.1: Binomial Coefficient (Divide-and-Conquer)

```python
def bin (n, k):
    if (k == 0 or n == k):
        return 1
    else:
        return bin(n - 1, k - 1) + bin(n - 1, k)
```

```python
for n in range(10):
    for k in range(n + 1):
        print(bin(n, k), end = " ")
    print()
```

Algorithm 3.1의 문제점

- 피보나치 항 구하기의 재귀적(recursive) 방법과 같은 문제
- 중복 호출을 없앨 수 있는 방법은? 반복적(iterative) 방법

Top-Down →

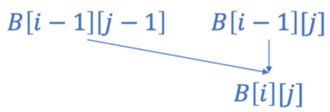
```
bin(4, 2)
    bin(3, 1)
        bin(2, 0)
        bin(2, 1)
            bin(1, 0)
            bin(1, 1)
    bin(3, 2)
        bin(2, 1)
            bin(1, 0)
            bin(1, 1)
        bin(2, 2)
```

← **Bottom-Up**

이항 계수의 성질: 파스칼의 삼각형

B[i][j]	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

j, k (columns); *i, n* (rows)



- You may recognize the array in this figure as *Pascal's triangle*.

Bottom - up이 가능 -> tabulation

1단계: 재귀 관계식을 찾는다.

- 이항 계수의 재귀적 정의를 찾았다.

$$B[i][j] = \begin{cases} B[i-1][j-1] + B[i-1][j] & 0 < j < i \\ 1 & j = 0 \; or \; j = i \end{cases}$$

2단계: 상향식 방법으로 해답을 구한다.

- 파스칼의 삼각형이 가진 특성을 이용한다.

- B[i][j] = 1, j = 0 or j = 1

- B[i][j] = B[i - 1][j - 1] + B[i - 1][j], 0 < j < i

이항 계수의 행렬을 보면 symetric(대칭적) 한 것을 알 수 있다. 따라서 뒤에 것은 계산할 필요 X.

▪ Using *Tabulation*

```
typedef unsigned long long longint;
vector<longint> F;

longint fib3(int n) {
    F.resize(n + 1);
    if (n <= 1)
        F[n] = n;
    else {
        F[0] = 0; F[1] = 1;
        for (int i = 2; i <= n; i++)
            F[n] = F[n - 1] + F[n - 2];
    }
    return F[n];
}
```

이항 계수의 시간 복잡도와 성능 개선

- Algorithm 3.2(D.P)는 Algorithm 3.1(D&C)보다 훨씬 효율적
    - D&C의 시간 복잡도 $\in \Theta(\binom{n}{k})$, D.P의 시간 복잡도 $\in \Theta(nk)$

연습문제 3.4: 효율적인 이항계수 계산

- 다음 성질을 이용하면 성능을 더 개선할 수 있다.
- $\binom{n}{k} = \binom{n}{n-k}$: k가 $n/2$보다 클 경우에 적용
- 2차원 리스트(배열)를 사용할 필요가 있는가?
    - 1차원 리스트(배열)만으로도 구현이 가능하다.

```
LongInteger bin3(int n, int k) {
    vector<LongInteger> B(n + 1);
    if (k > n / 2)
        k = n - k;
    B[0] = 1;
    for (int i = 1; i < n + 1; i++) {
        int j = min(i, k);
        while (j > 0) {
            B[j] = (B[j] + B[j - 1]) % 10007;
            j -= 1;
        }
    }
    return B[k];
}
```

```cpp
// Binomial Coefficient: Tabulation
/* input case
10000 5000
*/

#include <iostream>
#include <vector>
using namespace std;

typedef unsigned long long LongInteger;
typedef vector<vector<LongInteger> > matrix_t;

// LongInteger bin2(int n, int k) {
//     matrix_t B(n + 1, vector<LongInteger>(n + 1));
//     for (int i = 0; i <= n; i++) {
//         for (int j = 0; j <= min(i, k); j++) {
//             if (j == 0 || j == i)
//                 B[i][j] = 1;
//             else
//                 B[i][j] = (B[i - 1][j] + B[i - 1][j - 1]) % 10007;
//         }
//     }
//     return B[n][k];
// }

LongInteger bin3(int n, int k) {
    vector<LongInteger> B(n + 1);
    if (k > n / 2)
        k = n - k;
    B[0] = 1;
    for (int i = 1; i < n + 1; i++) {
        int j = min(i, k);
        while (j > 0) {
            B[j] = (B[j] + B[j - 1]) % 10007;
            j -= 1;
        }
    }
    return B[k];
}

int main () {
    int n, k;
    cin >> n >> k;

    LongInteger result = bin3(n, k);

    cout << result << endl;

    return 0;
}
```

```cpp
// Binomial Coefficient: Memoization
/* input case
2000 500
*/

#include <iostream>
#include <vector>
using namespace std;

typedef unsigned long long LongInteger;
typedef vector<vector<LongInteger> > matrix_t;
LongInteger call_count = 0;
matrix_t B;

LongInteger binom(int n, int k) {
    call_count++;
    if (k == 0 || k == n) {
        return 1;
    } else if (B[n][k] != -1) {
        return B[n][k];
    } else {
        B[n][k] = (binom(n - 1, k - 1) + binom(n - 1, k)) % 10007;
        return B[n][k];
    }
}

int main () {
    int n, k;
    cin >> n >> k;

    B.assign(n + 1, vector<LongInteger>(k + 1, -1));
    LongInteger result = binom(n, k);

    cout << result << endl;
    cout << call_count << endl;
}
```