서울시립대학교 소모임 알

컴퓨터과학부 김희중

Dynamic Programming

Dynamic Programming

Dynamic Programming

DP, 동적계획법 이라고도 한다.

- 큰 문제를 작은 문제로 나눠서 푸는 알고리즘
- Dynamic, 동적 이라는 단어 뜻과 아무런 관련이 없다.
- 이 용어를 만든 Richard Bellman은 Dynamic이라는 단어가 멋있어서 사용했다고 한다.

그리디 알고리즘과의 비교

Dynamic Programming은 최적의 해를 구한다.

- 그리디 알고리즘은 순간순간의 상황에 당장의 최선의 선택을 한다. 부분에서는 최적의 해가 될 수 있지만 항상 전체적으로 최적의 해를 구할 수는 없다.
- 반면 동적계획법은 항상 최적의 해를 구한다.

Dynamic Programming

Dynamic Programming의 속성

- 두 가지 속성을 만족해야 Dynamic Programming으로 문제를 해결할 수 있다.

1. Overlapping Subproblems : 반복되는 부분문제, 같은 재귀함수 콜의 반복

2. Optimal Substructure : 최적 부분구조, 전체 문제의 최적해가 부분 문제의 최적해로부터 만들어지는 구조

Overlapping Subproblems

반복되는 부분문제

- 피보나치 수

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ ($n \geq 2$)

Overlapping Subproblems

반복되는 부분문제

- 피보나치 수

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ ($n \geq 2$)

- 문제 : N번째 피보나치 수를 구한다.

- 작은 문제 : N-1번째 피보나치 수를 구한다, N-2번째 피보나치 수를 구한다.

Overlapping Subproblems

반복되는 부분문제

- 문제 : N번째 피보나치 수를 구한다.
- 작은 문제 : N-1번째 피보나치 수를 구한다, N-2번째 피보나치 수를 구한다.

- 문제 : N-1번째 피보나치 수를 구한다.
- 작은 문제 : N-2번째 피보나치 수를 구한다, N-3번째 피보나치 수를 구한다.

- 문제 : N-2번째 피보나치 수를 구한다.
- 작은 문제 : N-3번째 피보나치 수를 구한다, N-4번째 피보나치 수를 구한다.

Overlapping Subproblems

반복되는 부분문제

- 큰 문제와 작은 문제는 상대적이다.
- 큰 문제와 작은 문제를 같은 방법으로 풀 수 있다.
- 문제를 작은 문제로 쪼갤 수 있다.

Optimal Substructure

최적 부분구조

- 문제의 답을 작은 문제의 정답에서 구할 수 있다.
- Ex) 정문에서 후문을 가는 가장 빠른 길은 건설공학관, 제2 공학관을 거쳐야 한다면.
- 건설공학관에서 후문을 가장 빠르게 가려면 제2 공학관을 거쳐야 한다.

Optimal Substructure

최적 부분구조

- 문제 : N번째 피보나치 수를 구한다.
- 작은 문제 : N-1번째 피보나치 수를 구한다, N-2번째 피보나치 수를 구한다.
- **문제의 답을 작은 문제의 정답을 합하는 것으로 구할 수 있다.**

- 문제 : N-1번째 피보나치 수를 구한다.
- 작은 문제 : N-2번째 피보나치 수를 구한다, N-3번째 피보나치 수를 구한다.
- **문제의 답을 작은 문제의 정답을 합하는 것으로 구할 수 있다.**

Optimal Substructure

최적 부분구조

- Optimal Substructure를 만족한다면 크기에 상관 없이 어떠한 문제의 정답은 일정하다.

- 10번째 피보나치 수를 구하면서 구한 4번째 피보나치 수
- 9번째 피보나치 수를 구하면서 구한 4번째 피보나치 수
- …
- 5번째 피보나치 수를 구하면서 구한 4번째 피보나치 수
- 4번째 피보나치 수를 구하면서 구한 4번째 피보나치 수

- 는 항상 같다.

Dynamic Programming

Dynamic Programming

- Dynamic Programming에서는 각 부분 문제를 한번만 풀어야 한다.
- Optimal Substructure를 만족하기 때문에 같은 문제는 구할 때 마다 답이 같다.
- 따라서, 정답을 한 번 구했으면 정답을 배열에 메모한다.
- => Memoization

피보나치 수

Fibonacci

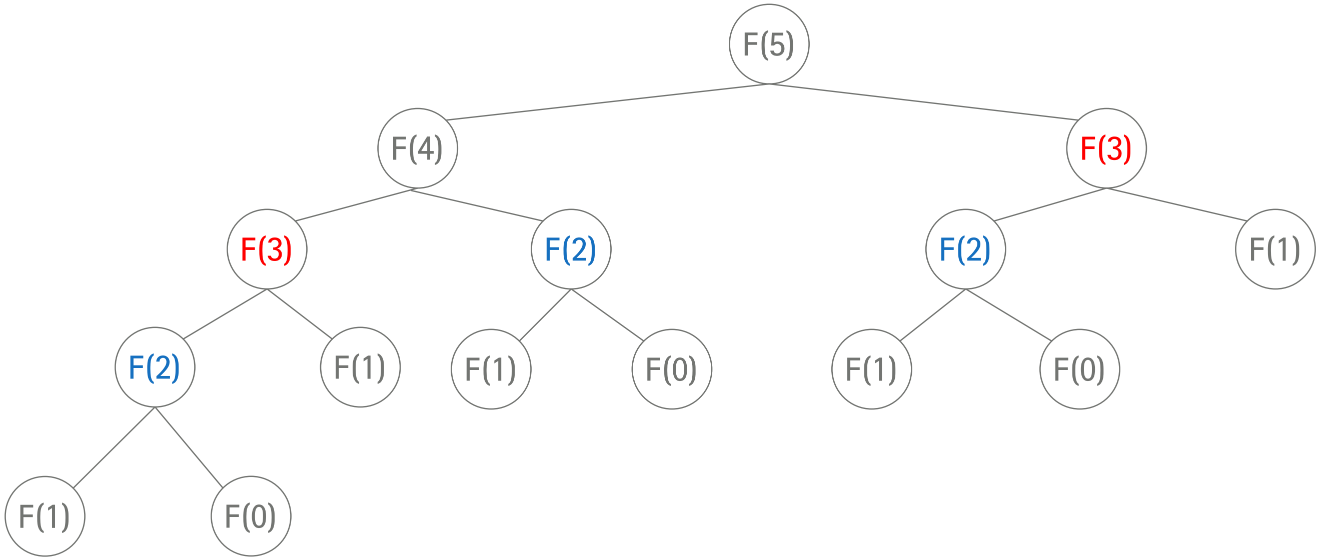
- 피보나치수를 구하는 재귀함수 구현의 예

```
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

피보나치 수

Fibonacci

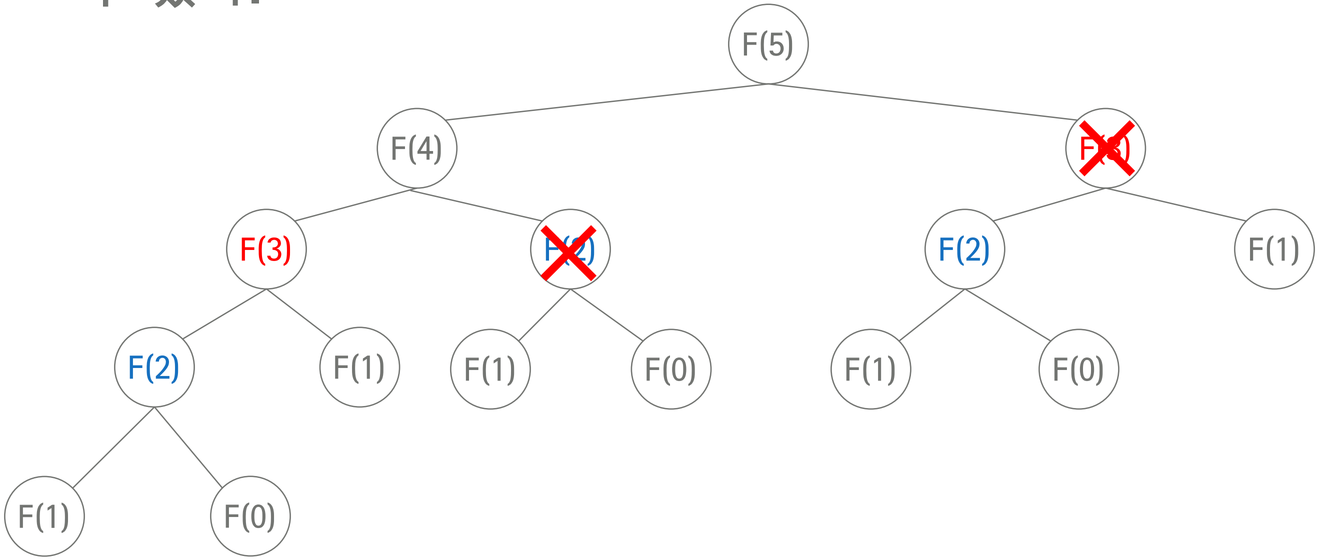
- 하지만... 이런 구현은 재귀함수의 반복되는 호출 문제가 있다.



피보나치 수

Fibonacci

- 한번 답을 구할 때, Memoization을 통해 중복 호출을 막을 수 있다.



Top-down

Dynamic Programming

- 방금 같은 방법을 Top-down 이라 한다.
- 문제를 작은 문제로 나눠서 작은 문제를 해결한다.
- 해결 된 작은 문제들을 통해 전체 문제를 해결한다.
- 재귀함수 + 메모이제이션!

Bottom-up

Dynamic Programming

- 문제의 크기가 작은 문제부터 차례대로 해결한다.
- 문제의 크기를 점점 키워가며 문제를 해결한다.
- 이를 반복하면 풀어야 하는 문제의 크기까지 도달한다.
- 반복문 + 배열!

Combination

조합

Combination

- n개 중 k개를 순서와 상관 없이 선택하는 경우의 수 (nCk)

- nCk을 구하는 방법은 여러가지가 있다.

$$\binom{n}{k} = \frac{P(n,k)}{k!} = \frac{n!}{k! \cdot (n-k)!}$$

- 위와 같이 factorial을 통해서 구할 수 있지만 조합의 수는 점화식이 존재한다.

조합

Combination

- 파스칼의 삼각형에서 볼 수 있듯 조합의 개수는 아래와 같은 점화식이 존재한다.

$$
\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & 1 & & 1 & & & \\
 & & 1 & & 2 & & 1 & & \\
 & 1 & & 3 & & 3 & & 1 & \\
1 & & 4 & & 6 & & 4 & & 1
\end{array}
\equiv
\begin{array}{ccccccccc}
 & & & & {}_0C_0 & & & & \\
 & & & {}_1C_0 & & {}_1C_1 & & & \\
 & & {}_2C_0 & & {}_2C_1 & & {}_2C_2 & & \\
 & {}_3C_0 & & {}_3C_1 & & {}_3C_2 & & {}_3C_3 & \\
{}_4C_0 & & {}_4C_1 & & {}_4C_2 & & {}_4C_3 & & {}_4C_4
\end{array}
$$

$${}_nC_k = {}_{n-1}C_{k-1} + {}_{n-1}C_k$$

이친수

2193

https://www.acmicpc.net/problem/2193

- N자리의 이친수를 만들 때
- 현재 자리에
- 1. ‘0’이 오거나
- 2. ‘1’이 오거나…
- 이 두가지의 가지를 만들어낸다.

피보나치 함수

1003

https://www.acmicpc.net/problem/1003

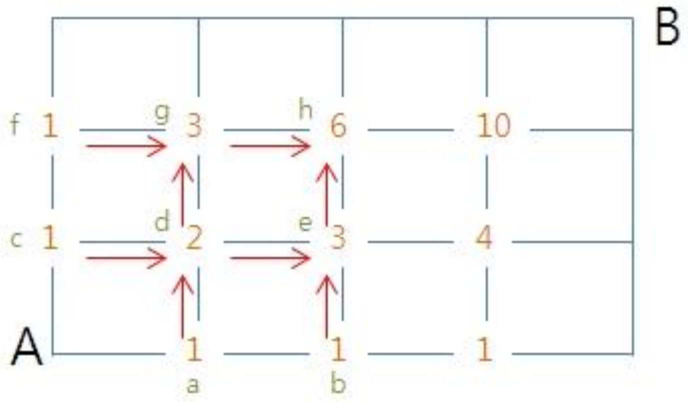
- 피보나치 수열 간의 점화식이 있다.
- 그렇다면 피보나치 함수호출 횟수의 점화식은?

점프

1890

https://www.acmicpc.net/problem/1890

- 고등학교 때 바둑판에서 경로의 수를 세던 문제를 떠올리자.



가장 긴 증가하는 부분 수열

11053

https://www.acmicpc.net/problem/11053

- 가장 긴 증가하는 부분 수열 (Longest Increasing Subsequence, 이하 LIS) 문제는 DP의 가장 대표적인 문제
- LIS : {10, 20, 10, 30, 20, 50}
- LIS의 길이 : 4

11053

https://www.acmicpc.net/problem/11053

- Hint : 점화식을 다음과 같이 정한다.
- An = n번째 숫자로 시작하는 LIS의 길이
- An = n번째 숫자를 마지막으로 하는 LIS의 길이