JavaScript 다이나믹 프로그래밍

다이나믹 프로그래밍 이해하기

다이나믹 프로그래밍 이해하기 | 코딩 테스트에서 자주 등장하는 다이나믹 프로그래밍 이해하기

강사 나동빈

JavaScript 다이나믹 프로그래밍

다이나믹 프로그래밍 이해하기

JavaScript 동적 계획법 다이나믹 프로그래밍

다이나믹 프로그래밍

JavaScript 동적 계획법
다이나믹 프로그래밍

- 통상적으로 메모리를 더 사용하여 **시간 복잡도를 개선**할 때 많이 사용된다.
- 구체적으로, 시간 복잡도가 비효율적인 알고리즘이 있을 때 **부분 문제의 반복**이 발생하는 경우 적용하면 효과적이다.
- 다이나믹 프로그래밍 문제를 해결하기 위해 **점화식**을 찾는 것이 **핵심**적인 과정이다.

다이나믹 프로그래밍의 사용 조건

JavaScript
동적 계획법
다이나믹
프로그래밍

- 다이나믹 프로그래밍은 일반적으로 아래의 두 조건을 만족할 때 사용할 수 있다.

1. **최적 부분 구조(optimal substructure)**

- 큰 문제를 유사한 형태의 작은 문제로 나눌 수 있으며, 작은 문제의 답을 모아 큰 문제를 해결한다.

2. **반복되는 부분 문제(overlapping sub-problem)**

- 동일한 작은 문제를 반복적으로 해결해야 한다.

점화식과 최적 부분 구조

JavaScript
동적 계획법
다이나믹
프로그래밍

- **피보나치 수열 예시**: $[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \dots]$
- **점화식**: 인접한 항으로 현재 값을 결정하는 관계식을 의미한다.

→ 일반적으로 **최적 부분 구조**를 만족한다는 특징이 있다.

- 피보나치 수열의 점화식: $a_n = a_{n-1} + a_{n-2}\ (a_1 = 1, a_2 = 1)$

JavaScript 동적 계획법 점화식의 구성요소
다이나믹 프로그래밍

- 점화식의 **기본적인 구성 요소**는 다음과 같다.

1. 초기항
2. 인접한 항과의 관계

- 점화식은 **[재귀 함수]**로 표현할 수 있다.
- 재귀 함수는 **[종료 조건]**이 있어야 하는데, 이것이 점화식의 초기항과 같은 역할을 수행한다.

```
// 피보나치 함수(Fibonacci Function)을 재귀함수로 구현
function fibo(x) {
  if (x == 1 || x == 2) {
      return 1;
  }
  return fibo(x - 1) + fibo(x - 2);
}

console.log(fibo(4));
```

$$a_n = a_{n-1} + a_{n-2}\ (a_1 = 1, a_2 = 1)$$

JavaScript 동적 계획법 점화식을 코드로 구현하는 방법
다이나믹 프로그래밍

- 점화식을 **재귀 함수** 코드로 구현할 수 있다.

1. 점화식을 **초기항**은 **종료 조건**과 같은 역할을 수행한다.
2. 점화식의 내용은 $f(x)$의 반환 값에 들어간다.

```
// 피보나치 함수(Fibonacci Function)을 재귀함수로 구현
function fibo(x) {
  if (x == 1 || x == 2) {
      return 1;
  }
  return fibo(x - 1) + fibo(x - 2);
}

console.log(fibo(4));
```

종료 조건이 없으면 무한 루프

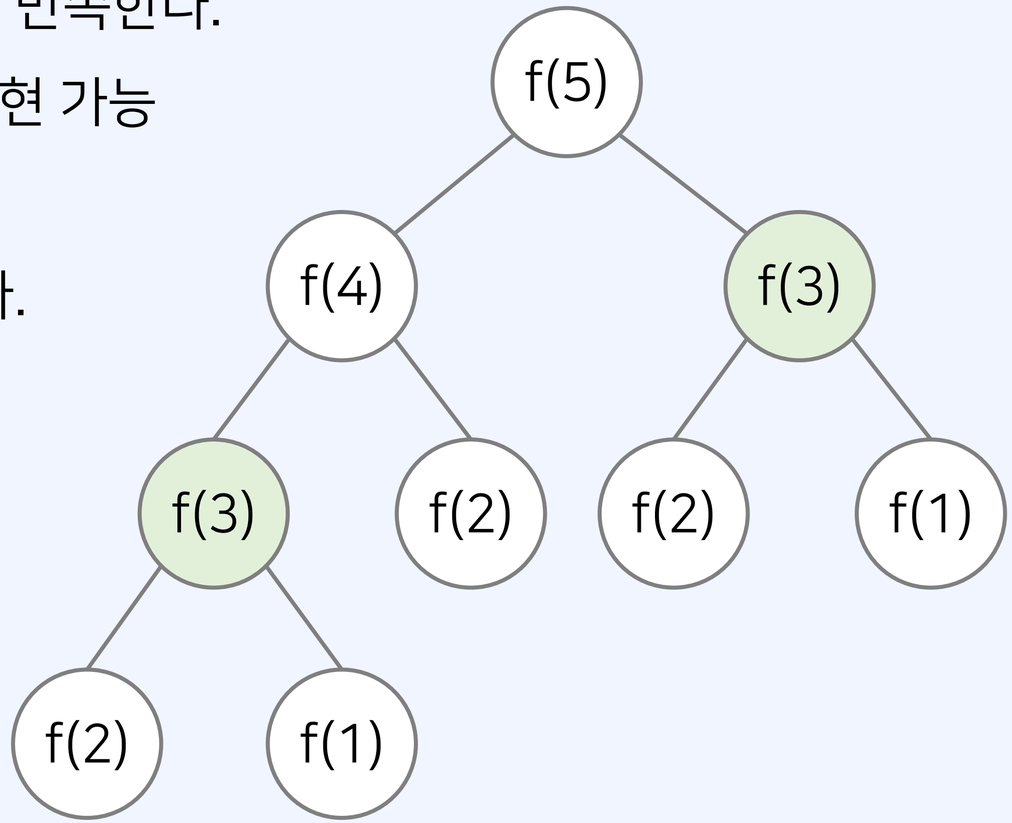
실질적인 점화식 부분

JavaScript 동적 계획법 피보나치 수열 문제
다이나믹 프로그래밍

- 피보나치 수열 문제는 다음의 조건을 만족한다.

1. **최적 부분 구조** → 점화식 형태로 표현 가능
2. **반복되는 부분 문제**

- 이미 해결한 문제를 또 해결해야 한다.



JavaScript 동적 계획법 다이나믹 프로그래밍의 일반 형태

- 피보나치 수열의 점화식을 그대로 재귀 함수로 구현하면 어떻게 될까?

→ **중복되는 부분 문제**가 발생한다. (이미 구한 값을 불필요하게 반복 계산)

- 다이나믹 프로그래밍은 이 문제를 해결할 수 있도록 해준다.

```
d = new Array(100).fill(0); // 한 번 계산된 결과를 메모이제이션(Memoization)하기 위한 리스트 초기화
function fibo(x) { // 피보나치 함수(Fibonacci Function)를 재귀함수로 구현(탑다운 다이나믹 프로그래밍)
  if (x == 1 || x == 2) {// 종료 조건(1 혹은 2일 때 1을 반환)
    return 1;
  }
  // 이미 계산한 적 있는 문제라면 그대로 반환
  if (d[x] != 0) {
    return d[x];
  }
  // 아직 계산하지 않은 문제라면 점화식에 따라서 피보나치 결과 반환
  d[x] = fibo(x - 1) + fibo(x - 2);
  return d[x];
}
console.log(fibo(99));
```

종료 조건 이 없으면 무한 루프

한 번 해결한 문제는 여러 번 해결하지 않도록

실질적인 점화식 부분

JavaScript 동적 계획법 다이나믹 프로그래밍의 일반 형태

다이나믹 프로그래밍

- 다이나믹 프로그래밍(재귀 함수)의 대표적인 코드 형식은 다음과 같다.

```
function dp() {
  ① 종료 조건
  ② 이미 해결한 문제라면, 정답을 그대로 반환
  ③ 점화식에 따라 정답 계산
}
```

JavaScript 동적 계획법 다이나믹 프로그래밍 문제 해결 과정

- 다이나믹 프로그래밍 문제 해결 접근 순서는 다음과 같다.

1. 문제 이해하기
2. 점화식 찾아내기 → 일반적으로 가장 **핵심**적인 부분이다.
3. 구현 방식(상향식/하향식) 결정하기
4. 점화식을 실제 코드로 구현하기

JavaScript 동적 계획법 다이나믹 프로그래밍 문제 접근 방법

JavaScript
동적 계획법
다이나믹
프로그래밍

- 다이나믹 프로그래밍 문제는 두 가지 방법으로 접근할 수 있다.

1. **상향식**: 반복문을 이용해 초기 항부터 계산한다.
2. **하향식**: 재귀 함수로 큰 항을 구하기 위해 작은(이전) 항을 호출하는 방식이다.

→ 이미 구한 함수 값을 담는 테이블을 흔히 **DP 테이블**이라고 한다.

JavaScript 동적 계획법 피보나치 수열(하향식)
다이나믹 프로그래밍

```
// 한 번 계산된 결과를 메모이제이션(Memoization)하기 위한 리스트 초기화
d = new Array(100).fill(0);

// 피보나치 함수(Fibonacci Function)를 재귀함수로 구현(탑다운 다이나믹 프로그래밍)
function fibo(x) {
  // 종료 조건(1 혹은 2일 때 1을 반환)
  if (x == 1 || x == 2) {
    return 1;
  }
  // 이미 계산한 적 있는 문제라면 그대로 반환
  if (d[x] != 0) {
    return d[x];
  }
  // 아직 계산하지 않은 문제라면 점화식에 따라서 피보나치 결과 반환
  d[x] = fibo(x - 1) + fibo(x - 2);
  return d[x];
}

console.log(fibo(99));
```

JavaScript 동적 계획법 피보나치 수열(상향식)
다이나믹 프로그래밍

```
// 앞서 계산된 결과를 저장하기 위한 DP 테이블 초기화
d = new Array(100).fill(0);

// 첫 번째 피보나치 수와 두 번째 피보나치 수는 1
d[1] = 1;
d[2] = 1;
n = 99;

// 피보나치 함수(Fibonacci Function) 반복문으로 구현(보텀업 다이나믹 프로그래밍)
for (let i = 3; i <= n; i++) {
  d[i] = d[i - 1] + d[i - 2];
}

console.log(d[n]);
```

JavaScript 동적 계획법 창고 털기 문제 예시

다이나믹 프로그래밍

- N개의 창고가 있을 때, 얻을 수 있는 식량의 최댓값을 계산해 보자.
- 이때, 최소한 한 칸 이상 떨어진 창고들만 선택하여 털 수 있다.

	창고 1	창고 2	창고 3	창고 4
예시)	1	3	1	5

- 현재 예시에서는 **두 번째 창고**와 **네 번째 창고**를 선택했을 때 최댓값인 8을 얻을 수 있다.

JavaScript 동적 계획법 창고 털기 문제 예시
다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

- 각 위치까지의 **최적의 해**를 일종의 수열에서의 각 항으로 볼 수 있다.
- 왼쪽부터 하나씩 창고를 본다고 가정하자.

→ 차례대로 $a_1, a_2, a_3, \cdots, a_N$으로 이해하자.

- 혹은 함수 $f(\cdot)$ 형태로 표현하기도 하며 그 의미는 다음과 같다.
- $f(1)$: 1번 창고까지 처리했을 때, 최대 식량 값(optimal solution)
- ...
- $f(N)$: N개의 창고까지 모두 처리했을 때, 최대 식량 값(optimal solution)

JavaScript 동적 계획법 창고 털기 문제 예시

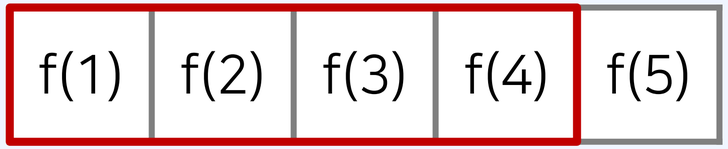
다이나믹 프로그래밍

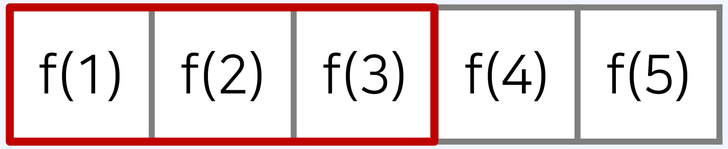
JavaScript
동적 계획법
다이나믹
프로그래밍

- 각 [최적의 해]를 수열에서의 각 항으로 보자. 어떤 특성이 있는지 확인할 수 있다.

$$f(5) = \max\big(f(4), f(3) + arr(5)\big)$$

- $f(3)$ = 위치 3까지의 해
- $f(4)$ = 위치 4까지의 해





- 4번째($f(4)$)를 털었다면, 5번째는 털 수 없다.
- 3번째($f(3)$)까지만 고려했다면, 5번째는 털 수 있다.

JavaScript 동적 계획법 창고 털기 문제 예시
다이나믹 프로그래밍

- 이 문제도 결국 아래와 같은 두 조건을 만족한다.

1. **최적 부분 구조(optimal substructure)**

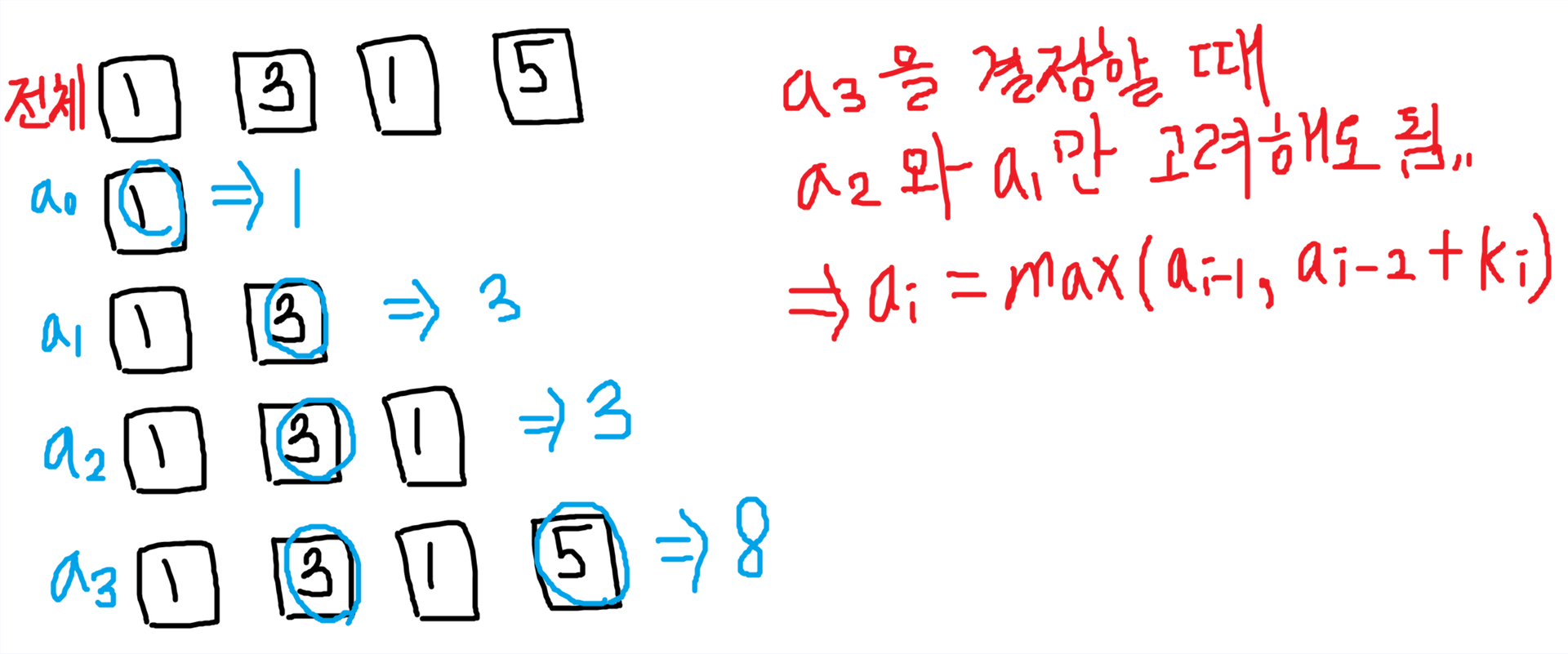
- 큰 문제는 동일한 구조의 작은 문제의 조합으로 해결 가능하다.
- **점화식** 그 자체로 이해할 수 있다.

2. **부분 문제의 중복(overlapping sub-problem)**

- 해결했던 부분 문제를 또 해결해야 하는 경우가 발생한다.
- 이미 해결한 문제를 **[캐싱]** 혹은 **[메모이제이션]**하여 해결 할 수 있다.

JavaScript 동적 계획법 창고 털기 문제 예시

다이나믹 프로그래밍



JavaScript 동적 계획법 창고 털기 문제 예시
다이나믹 프로그래밍

```
// 정수 N을 입력 받기
n = 4;
// 모든 식량 정보 입력받기
array = [1, 3, 1, 5];

// 앞서 계산된 결과를 저장하기 위한 DP 테이블 초기화
d = new Array(100).fill(0);

// 다이나믹 프로그래밍(Dynamic Programming) 진행 (보텀업)
d[0] = array[0];
d[1] = Math.max(array[0], array[1]);
for (let i = 2; i < n; i++) {
  d[i] = Math.max(d[i - 1], d[i - 2] + array[i]);
}

// 계산된 결과 출력
console.log(d[n - 1]);
```

JavaScript 동적 계획법 1로 만들기 문제 예시

다이나믹 프로그래밍

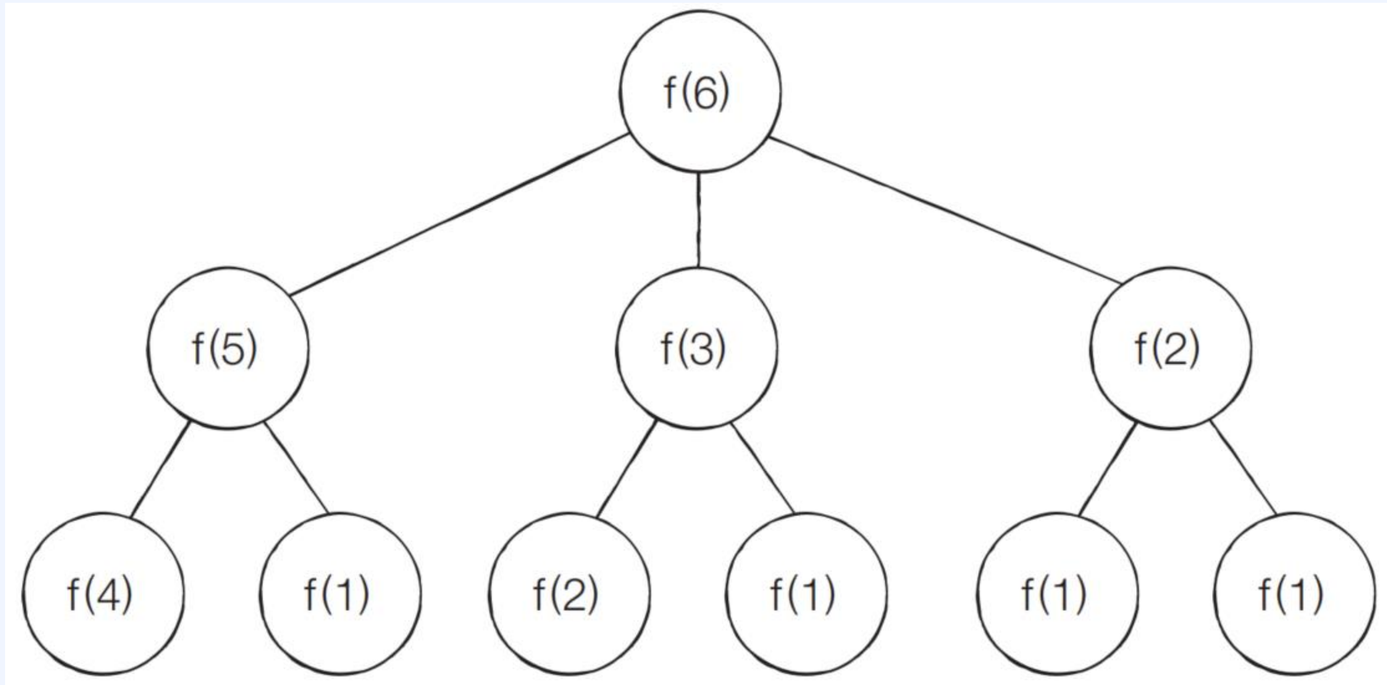
- 정수 X가 주어졌을 때, 정수 X에 사용할 수 있는 연산은 다음과 같이 **4가지**다.
  1. X가 5로 나누어 떨어지면, 5로 나눈다.
  2. X가 3으로 나누어 떨어지면, 3으로 나눈다.
  3. X가 2로 나누어 떨어지면, 2로 나눈다.
  4. X에서 1을 뺀다.
- 정수 X가 주어졌을 때, 연산 4개를 적절히 사용해서 값을 1로 만들고자 한다.
- 연산을 사용하는 횟수의 최솟값을 출력하여라.
- 예를 들어 정수가 26이면 다음과 같이 계산해서 3번의 연산이 최솟값이다.
  - 26 → 25 → 5 → 1

JavaScript 동적 계획법 1로 만들기 문제 예시

다이나믹 프로그래밍

JavaScript
동적 계획법
다이나믹
프로그래밍

- 피보나치 수열 문제를 도식화한 것처럼 함수가 호출되는 과정을 그림으로 그려보자.
- 다음의 조건을 만족한다.
  - **최적 부분 구조**
  - **중복되는 부분 문제**



JavaScript 동적 계획법 1로 만들기 문제 예시

다이나믹 프로그래밍

- 다이나믹 프로그래밍 문제를 해결할 때는 **점화식**을 세우는 것이 가장 중요하다.
- 이걸 위해서는 각 항을 어떻게 정의할 수 있는지가 중요하다.
- $f(x) = x$번째 항 = x까지 보았을 때 최적의 해(문제에서 요구하는 바)

[정의] $f(x)$: x를 1로 만들기 위해 필요한 연산의 최소 개수

- $f(x)$을 구하기 위해서 인접한 항들을 이용할 수 있는가?

JavaScript 동적 계획법 1로 만들기 문제 예시

다이나믹 프로그래밍

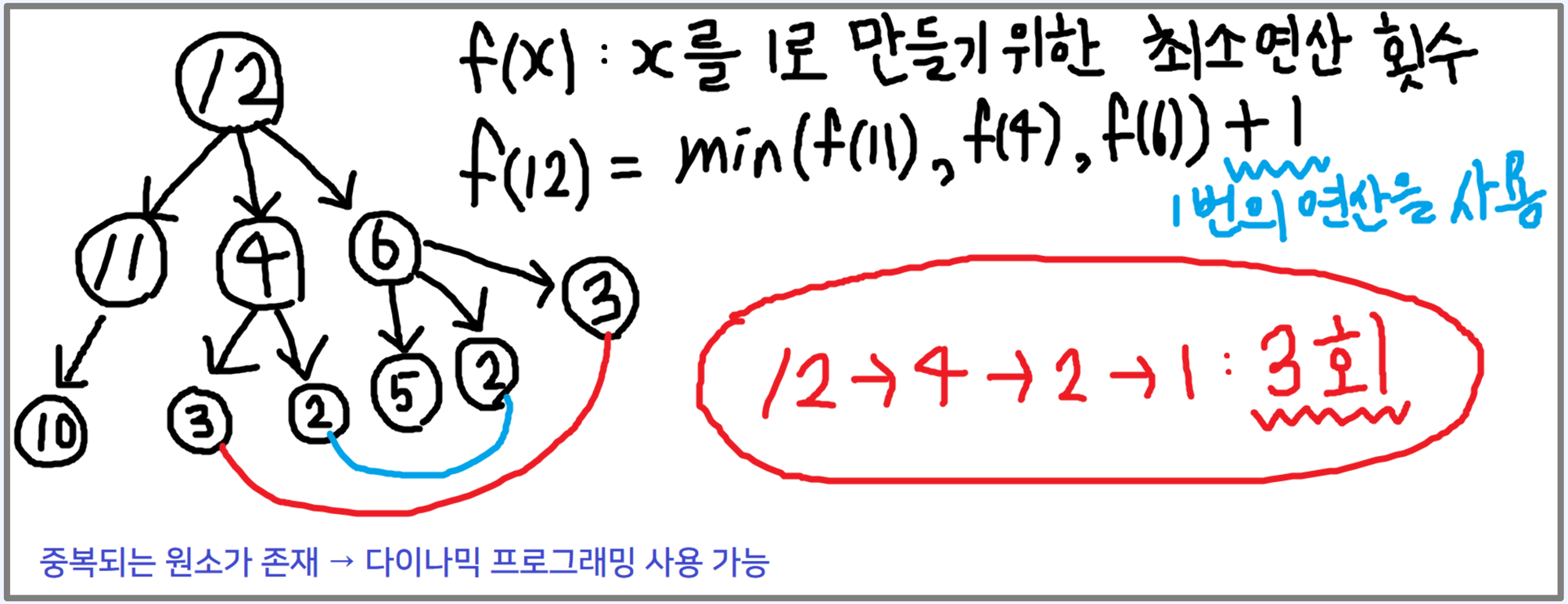
- 다이나믹 프로그래밍 문제를 해결할 때는, 경험적으로 시도해보는 것도 좋은 방법이다.

→ 일단 시도하다 보면 **점화식**이 보이는 경우가 종종 있다.

- $f(1) = 0$ (**이유**: 이미 값이 1이다.)
- $f(2) = 1$ (**이유**: 1을 빼거나 2로 나눌 수 있다.)
- $f(3) = 1$ (**이유**: 3으로 나눌 수 있다.)
- $f(4) = 2$ (**이유**: 1을 뺀 뒤에 $f(3)$ 혹은 2로 나누고 $f(2)$)
- $f(5) = 1$ (**이유**: 5로 나눌 수 있다.)
- $f(6) = 2$ (**이유**: 1을 뺀 뒤에 $f(5)$ 혹은 2로 나누고 $f(3)$ 혹은 3으로 나누고 $f(2)$)
- ...

JavaScript 동적 계획법 1로 만들기 문제 예시

다이나믹 프로그래밍



JavaScript 동적 계획법 1로 만들기 문제 예시

다이나믹 프로그래밍

- 결과적으로 **점화식**을 세울 수 있다.
- 현재의 예시에서 +1 항은 하나의 연산을 사용하는 행위로 이해할 수 있다.

$$f(i) = \min(f(i-1), f(i/2), f(i/3), f(i/5)) + 1$$

JavaScript 동적 계획법 1로 만들기 문제 예시

다이나믹 프로그래밍

```
// 정수 X를 입력받기
x = 26;
// 앞서 계산된 결과를 저장하기 위한 DP 테이블 초기화
d = new Array(30001).fill(0);

// 다이나믹 프로그래밍(Dynamic Programming) 진행 (보텀업)
for (let i = 2; i <= x; i++) {
  // 현재의 수에서 1을 빼는 경우
  d[i] = d[i - 1] + 1;
  // 현재의 수가 2로 나누어 떨어지는 경우
  if (i % 2 == 0)
    d[i] = Math.min(d[i], d[parseInt(i / 2)] + 1);
  // 현재의 수가 3으로 나누어 떨어지는 경우
  if (i % 3 == 0)
    d[i] = Math.min(d[i], d[parseInt(i / 3)] + 1);
  // 현재의 수가 5로 나누어 떨어지는 경우
  if (i % 5 == 0)
    d[i] = Math.min(d[i], d[parseInt(i / 5)] + 1);
}

console.log(d[x]);
```