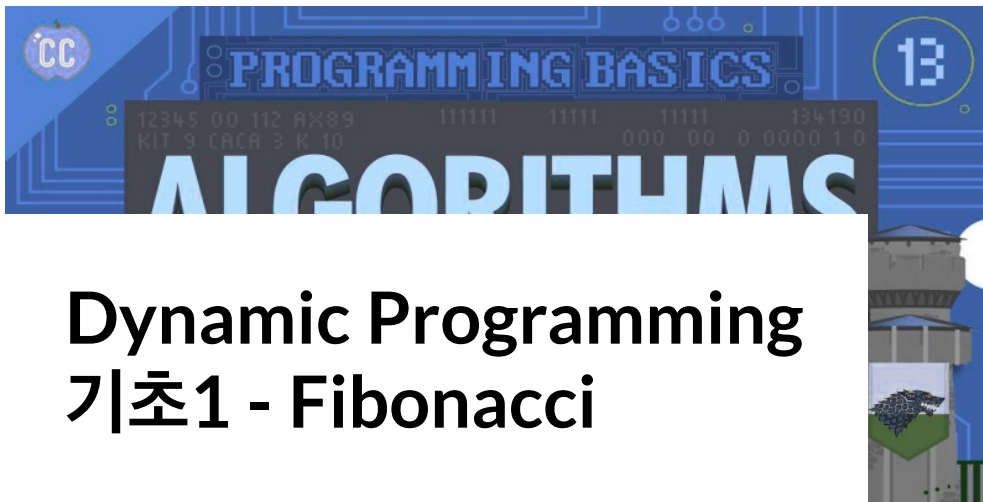


**YABOONG***Safety zone is unsafe.*

Dynamic Programming 기초1 - Fibonacci

BY YABOONG ON FEBRUARY 5, 2018

개요

피보나치 수열의 3가지 구현방법을 알아본다.

- Simple Recursion
- Recursion with Memoization
- Bottom-Up Approach

[Sample Code](#)

피보나치 수열

```
0, 1, 1, 2, 3, 5, 8, 13, 21...
```

– YABOONG



오스카 윈들러는 흔해 빠진 기회주의자요 부패한 사업가였다. 그러나 거대한 악이 세상을 점령하는 것처럼 보일 때 그 악에 대항해서 사람의 생명을 구한 것은 귀족도 지식인도 종교인도 아닌 부패한 기회주의자 오스카 윈들러였다.

– LATEST POSTS

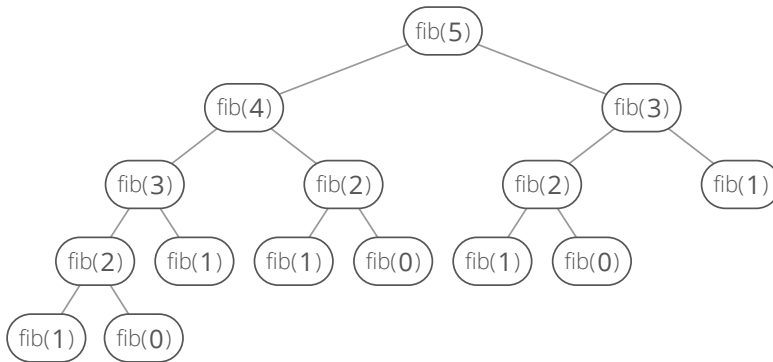
스프링 카프카 Batch Consumer - 의도치 않은 리스너 호출

n th value = $(n-1)$ th + $(n-2)$ th 의 방식으로 전개되는 수열이다. 재귀함수로 구현할 수 있다.

```
private static long fibSimple(int n) {
    return (n < 2) ? n : fib(n-1) + fib(n-2);
}
```

기존 Recursion 의 문제점

Recursive 하게 짠 fib() 함수는 아래와 같은 방식으로 호출이 일어난다.



$n=5$ 일 때 벌써 함수 호출을 15번이나 해야 한다. n 이 작을때는 문제 없지만 n 이 커졌을 때 2가지 문제점을 발견할 수 있다.

- 재귀호출이 지나치게 많아져서 memory 가 부족해 질 수 있다.
 - 함수가 호출되고 return 하기 전까지는 memory 의 stack 영역에 돌아갈 곳을 기록해 둔다. n 이 커지면 커질 수록 stack 에 함수 호출에 대한 기록이 더 많이 쌓이는데 return 하기 까지 또다른 재귀적 호출이 있기 때문에 문제가 생길 수 있다.
- 느리다.
 - $O(2^n)$ 의 시간 복잡도를 가진다.

JUNE 7, 2020

스프링 - 생성자 주입을
사용해야 하는 이유, 필드
인젝션이 좋지 않은
이유

AUGUST 29, 2019

자바 제네릭 이해하기

Part 1

JANUARY 19, 2019

스프링 - Spring
Cloud Config 예제

NOVEMBER 25, 2018

디자인패턴 - 프록시 패턴

OCTOBER 17, 2018

– CATEGORIES

ALGORITHMS

DATA-SCIENCE

DATA-STRUCTURES

DATABASE

DESIGN-PATTERN

DOCKER FOR-FUN

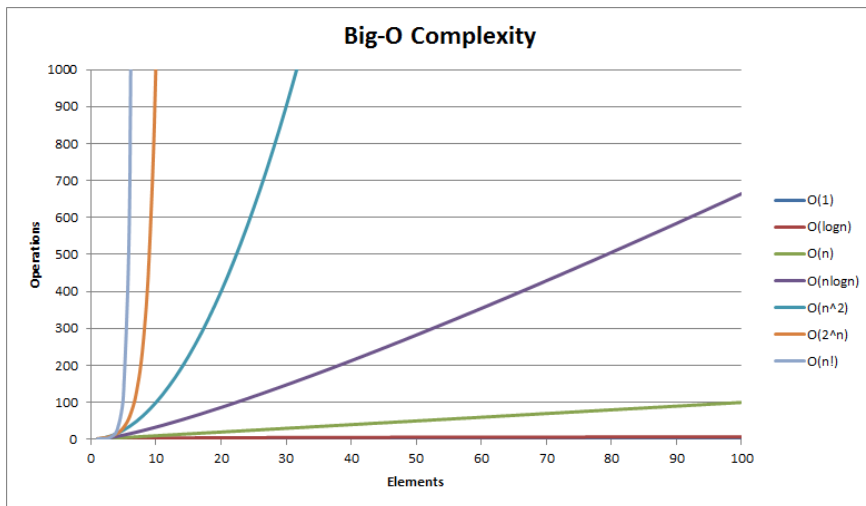
FRAMEWORKS

FUNCTIONAL-PROGRAMMING

JAVA JENKINS

PYTHON SCALA

SPARK SPRING



$O(2^n)$ 의 시간 복잡도를 가지는 알고리즘은 factorial 다음으로 최악의 성능을 가진다. Input 이 크다면 반드시 피해야 하는 방법이다.

해결방법1 - Memoization

Memoization 은 Dynamic Programming 의 한 방법으로 한번 계산한 값을 반복적으로 계산하지 않도록 기록해 두는 것이다.

기존 recursion 방식은 재귀호출 자체가 문제라기 보다는 불필요한 호출을 여러번 하게 된 다는 것이다. fib(5) 를 구하기 위해 fib(3) 은 2회, fib(2) 는 3회, fib(1) 은 5회... 호출하게 된다. Memoization 의 아이디어는 처음 fib(n) 을 계산할 때 어딘가에 기록해 두고 또 사용하게 될 경우에는 기록해 둔 결과값만 가지고 오는 것이다.

Memoization 을 사용했을 때의 recursion tree 는 아래 그림과 같다.

SPRING-CLOUD

- TAGS

ALGORITHMS

ARRAY AWS

BINARY-SEARCH-TREE

BINARY-TREE

BREADTH-FIRST-
SEARCH

CLUSTERING

DATA-ENGINEERING

DATA-SCIENCE

DATA-STRUCTURES

DATABASE

DEPTH-FIRST-SEARCH

DESIGN-PATTERN

DEVOPS DOCKER

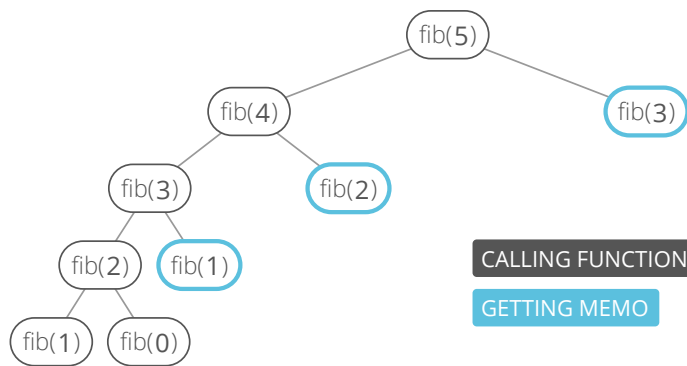
DYNAMIC-
PROGRAMMING

ELASTIC-BEANSTALK

EMR FILE-SYSTEM

FOR-FUN

FUNCTIONAL-
PROGRAMMING



```

private static long fibMemoization(int n, long[] memo) {
    if (memo[n] != 0) return memo[n]; // 기록해 둔 값
    memo[n] = (n == 1 || n == 2) ? 1 : fibMemoization(n-1, memo) + fibMemoization(n-2, memo);
    return memo[n];
}

```

Memoization 을 사용하는 호출의 경우 상수시간이 걸리고, 첫번째 fib(n) 을 계산 할 때만 재귀적 호출이 두번 $f(n-1) + f(n-2)$ 이루어 지기 때문에 $O(2n + c) = O(n)$ 의 time complexity 를 가진다. 기존 $O(2^n)$ 에서 $O(n)$ 으로의 개선은 큰 변화다.

해결방법2 - Bottom-Up

Recursion 을 사용하지 않는 방법. 상대적으로 memory 를 적게 사용한다. Recursion 이 top-down 이라면, bottom-up 은 말 그대로 작은 것 부터 순차적으로 풀어나간다.

풀이방법은 간단하다. 1st, 2nd, 3rd element 는 직접 지정해주고, 4th element 부터는 $array[n-1] + array[n-2]$ 를 계산한다.

```

private static long fibBottomUp(int n) {
    long[] bottomUp = new long[n+1];
    bottomUp[1] = 1;
    bottomUp[2] = 1;
    for (int i=3; i<=n; i++) bottomUp[i] = bottomUp[i-1] + bottomUp[i-2];
    return bottomUp[n];
}

```

GARBAGE-
COLLECTION

GENERICs GRAPH

HEAP HTTP4S

JAVA JENKINS

KAFKA

LINKED-LIST

MEMORY-
MANAGEMENT

NORMALIZATION

OOP PYTHON

QUEUE SCALA

SEMINAR

SORTING SPARK

SPRING

SPRING-CLOUD

SPRING-CLOUD-
CONFIG

STACK TREE

이 방법 역시 from 3 to n 까지 한 번의 loop 만 수행하면 되므로 $O(n)$ 의 시간복잡도를 가진다.

Comparison

위 세가지 방법을 thread 에서 동시에 실행 시키고 execution time 을 milliseconds 단위로 측정해 보았다.

```
public class Fibonacci {
    public static void main(String[] args) throws
        int N = 50;
        long[] memo = new long[N+1];

        Thread fibSimpleThread = new Thread(() ->
            long startTime = System.currentTimeMillis();
            System.out.format("fibSimple: %d %n", N);
            System.out.format("fibSimple elapsed: %d ms",
                System.currentTimeMillis() - startTime);
        });

        Thread fibMemoizationThread = new Thread(
            long startTime = System.currentTimeMillis();
            System.out.format("fibMemoization: %d %n", N);
            System.out.format("fibMemoization elapsed: %d ms",
                System.currentTimeMillis() - startTime);
        );

        Thread fibBottomUpThread = new Thread(() ->
            long startTime = System.currentTimeMillis();
            System.out.format("fibBottomUp: %d %n", N);
            System.out.format("fibBottomUp elapsed: %d ms",
                System.currentTimeMillis() - startTime);
        );

        fibSimpleThread.start();
        fibMemoizationThread.start();
        fibBottomUpThread.start();
    }

    private static long fibSimple(int n) {
        return (n < 2) ? n : fibSimple(n-1) + fibSimple(n-2);
    }

    private static long fibMemoization(int n, long[] memo) {
        if (memo[n] != 0) return memo[n];
        memo[n] = (n == 1 || n == 2) ? 1 : fibMemoization(n-1, memo) + fibMemoization(n-2, memo);
        return memo[n];
    }
}
```

```
private static long fibBottomUp(int n) {  
    long[] bottomUp = new long[n+1];  
    bottomUp[1] = 1;  
    bottomUp[2] = 1;  
    for (int i=3; i<=n; i++)  
        bottomUp[i] = bottomUp[i-1] + bottomUp[i-2];  
    return bottomUp[n];  
}
```

N 이 50만 되어도 실행 시간은 아래와같이 크게 차이 난다. 단순히 재귀로 접근한 방법보다 memoization, bottom-up approach 를 사용한 dynamic programming 방식이 (N=50 일 때) 약 2880 배 빠르다.

```
fibMemoization: 12586269025  
fibMemoization elapsed time: 25 ms  
  
fibBottomUp: 12586269025  
fibBottomUp elapsed time: 25 ms  
  
fibSimple: 12586269025  
fibSimple elapsed time: 72173 ms  
  
Process finished with exit code 0
```

계산 결과가 맞는지 [여기](#) 로 가면 n=300 까지의 피보나치 수를 확인할 수 있다.

참고한 자료

- [CJ Dojo Youtube Channel - What Is Dynamic Programming and How To Use It](#)

SHARE   

TAGS: DYNAMIC-PROGRAMMING JAVA

— COMMENTS

YABOONG.GITHUB.IO의 다른 댓글.

2년 전 • 댓글 9건

**스트링 - 생성자 주
입을 사용해야 하는
이유, ...**

3년 전 • 댓글 5건

**디자인패턴 - 템플릿
메소드 패턴**

3

D

표

Sponsored**12살인데 이렇게 크다고? "매일 '이것' 먹었어요!"**

아이클타임

나이, 학력무관 요새 이자격증 "뜬다" 月소득 '깜짝!'

에듀이지 손해평가사

왜 몰랐지?? "똥보유전자" 공략하니 요요없이 초고속 짹~빠졌다!!

제노핏 유전자다이어트

임플란트 시술 전 체크사항은?

석플란트치과병원

시흥동 의 치과 임플란트 비용에 놀라실지도 모릅니다

MyMedia

배우 윤세아의 피부관리 비법 '화제!' '기미, 주름, 모공' 이걸로 해결!?

천수윤珍

세계에서 가장 아름다운 여성 top 30

BuHamster.com

댓글 커뮤니티  개인정보 보호정책

 1 로그인 ▾

 추천


 Tweet

 공유

인기순 ▾

토론 시작

다음으로 로그인

또는 디스커스에 가입하세요. 



이름

1등으로 댓글 달기

 구독  당신의 사이트에 Disqus 추가하기Disqus 추가추가

Sponsored

새로운 거래 전쟁. 시장의 상황을 미리 파악하세요. 오늘 바로 계좌를
개설하세요

IC Markets

나이, 학력무관 요새 이자격증 "뜯다" 月소득 '깜짝!'

에듀이지 손해평가사

중년탈모 모발 재생효과 '이것' 2주만에 효과 보여?

모모단 프리미엄

임플란트 합리적으로 하려면?

석플란트치과병원

유저 사이에서 소문 난 인기 RPG의 습격! 재미는 물론 그래픽이 최
고!

Raid: Shadow Legends | 지금 바로 다운로드 받으세요

시흥동 의 치과 임플란트 비용에 놀라실지도 모릅니다

MyMedia

PREVIOUS

Docker 로 Http4s 프로젝트 Aws Elastic Beanstalk 에 배포해보기 - 1

JANUARY 19, 2018

NEXT

Array & ArrayList - Java

FEBRUARY 8, 2018

— YABOONG

오스카 윈들러는 흔해빠진 기회주의자요 부패한 사업가였다. 그러나 거대한 악이 세상을 점령하는 것처럼 보일 때 그 악에 대항해서 사람의 생명을 구한 것은 귀족도 지식인도 종교인도 아닌 부패한 기회주의자 오스카 윈들러였다.

📍 SEOUL, SOUTH KOREA

[HTTPS://GITHUB.COM/YABOONG](https://github.com/yaboong)



© YABOONG ALL RIGHTS RESERVED.
POWERED BY JEKYLL.