Data structure [B01]

김종규, PhD

2017-03-08

Fibonacci 관련 질문

- f_n 을 계산하는데 2^n 번 더하기 하는 것이 맞나요?
  - F_n: f_n 을 계산하는데 더하기 횟수
  - $F_0 = 0$
  - $F_1 = 0$
  - $F_2 = 1$
  - $F_3 = 2 = F_2 + F_1 + 1$
  - $F_n = F_{n-1} + F_{n-2} + 1$

$\longrightarrow$ 2^n 에 가까운 수

Fibonacci

- f_n 을 구하는 공식은 없나요?

$$f_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

$\longrightarrow$ 이산수학에서 배울 내용

알고리즘

- 개념: 컴퓨터로 원하는 일을 수행하도록 하는 방법
  - 무엇에 대해서 계산할 것인지가 결정되어야 함 (예: 자연수 n 에 대응되는 Fibonacci number f_n. 계산 결과는 자연수)
  - 어떻게 계산하는지를 명확하게 설명할 수 있어야 함 (예: $f_0 = 1$, $f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$)
  - 모든 가능한 입력값에 대해서 결과를 출력해야 함 $\longrightarrow$ 자연수에 대해서는 결과를 얻을 수 있어야 함

$\longrightarrow$ 이런 조건을 만족하는 방법은 여러 가지가 있을 수 있다. (예: `fib` 와 `fib_fast`)

알고리즘의 선택

- `fib` 와 `fib_fast` 중 어떤 것을 선택할까?
- f_n 을 비교하여 빠르게 계산할 수 있는 알고리즘을 선택
  - 그런데 n 이 1 이면 `fib` 이 빠름 (1ns vs 6ns)
  - n 이 45 이면 `fib_fast` 가 빠름 (수백 초 vs 0.1 초 미만)

$\longrightarrow$ n 이 커질 수록 차이가 더 벌어질 것임

$\therefore$ n 값이 클 때 f_n 을 빠르게 계산하는 알고리즘은 선택해야 함

$\longrightarrow$ 그런데, 얼마나 큰 것이 큰 것일까?

Growth function

- 두 개의 알고리즘 `f`, `g` 이 있다고 가정
  - 입력의 크기를 n 으로 나타냄
  - n 의 크기를 갖는 입력에 대해서 알고리즘을 수행하는데 걸리는 시간: $f(n), g(n) > 0$
  - n 에 관계 없이 $f(n) < g(n)$ 이라면 어떤 알고리즘을 선택할까? $\longrightarrow$ 당연히 `f`
  - $n < n_0$ 이면 $f(n) > g(n)$ 이지만 $n > n_0$ 이면 $f(n) < g(n)$ 이라고 보장할 수 있다. 어떤 알고리즘을 선택할까? $\longrightarrow$ 이론적으로는 `f`
- 이론적으로 좋은 알고리즘은 $n \to \infty$ 일 때 계산 결과를 빨리 도출할 수 있는 알고리즘이다. $\longrightarrow$ 이 개념을 어떻게 구체화 시킬까?

Growth function

- 두 개의 알고리즘을 수행하는데 걸리는 시간 (millisecond)
  - $f(n) = 1,000,000,000 \times n$
  - $g(n) = 0.000000001 \times 2^n$
- $f(1)$ 은 약 10 일 정도 소요됨. $g(1)$ 은 약 2 picoseconds
- 어떤 알고리즘이 좋을까?
  - $n_0 > 65$ 이고 $n > n_0$ 라면?
  - $n = 66$:
    - $f(n) = 66,000,000,000$
    - $g(n) = 73,786,976,295$

Growth function

- 결론: f 가 g 에 비해서 훨씬 좋은 알고리즘이다.
  - $f(n) = 1,000,000,000 \times n$
  - $g(n) = 0.000000001 \times 2^n$
- 앞에 곱해준 값 (coefficient) 는 별로 중요하지 않다
- n 이 증가할 때 얼마나 빨리 증가하는지가 중요하다

$\longrightarrow$ coefficient 는 무시하고 함수의 증가속도만 고려하도록 하자

Big-O: $O(g(n))$

- $O(g(n))$: 여러 함수의 집합
  - 어떤 상수 c, n_0 를 지정하여
  - $0 \leq f(n) \leq c \cdot g(n), n > n_0$ 를 만족시키도록

$\longrightarrow$ $O(g(n)) = \{f(n) | 0 \leq f(n) \leq c \cdot g(n), n > n_0\}$

- 예
  - $f(n) = n$, $g(n) = 1,000,000,000 \times n$
  - $f(n), g(n) \in O(n)$

Big-Omega: $\Omega(g(n))$

- $\Omega(n)$: 여러 함수의 집합
  - 어떤 상수 c, n_0 를 지정하여
  - $f(n) \geq c \cdot g(n) \geq 0, n > n_0$ 를 만족시키도록

$\longrightarrow$ $\Omega(g(n)) = \{f(n) | f(n) \geq c \cdot g(n) \geq 0, n > n_0\}$

$\longrightarrow$ 이보다 좋을 수는 없다는 것을 보장

Big-Theta: $\Theta(g(n))$

- Big-O, Big-Omega 를 모두 증명한 경우

Example

- 정의에 따라 다음은 모두 참이다
  - $f(n) \in \Theta(n)$
  - $g(n) \in \Theta(2^n)$
  - $f(n) \in \Omega(n)$
  - $g(n) \in \Omega(2^n)$
  - $f(n) \in O(n)$
  - $g(n) \in O(2^n)$

Example

- $h(n) = n^2$ 일 때 $h(n) \in O(n)$ 인가?
- 맞다고 가정 $\longrightarrow n_0, c$ 가 존재
  - $n > n_0 \longrightarrow h(n) < cn$
  - $n > n_0 \longrightarrow n^2 < cn$
  - $n > n_0 \longrightarrow n < c$

  $\therefore$ $n > n_0$ 이면서 동시에 $n < c$

$\longrightarrow$ 모순 $\longrightarrow \therefore h(n) \notin O(n)$

Example

- $f(n), g(n) \in O(n)$ 라고 가정
- $f(n), g(n) \in O(n^2)$ 일까?

  $\longrightarrow$ 당연히 $n > 0$ 이면 $n < n^2$ 이니까
- Big-O 가 보장하는 것 $\longrightarrow$ 이보다 나쁠 수는 없다는 정도는 보장된다

Big-O: $O(g(n))$

- 어떤 알고리즘이 $O(n^2)$ 라는 사실을 증명하였다. 다음 중 거짓인 것은?
  - 이 알고리즘은 $g(n) = n^2$ 의 시간보다는 빨리 결과를 도출할 것이다 $\longrightarrow$ 참
  - 이 알고리즘이 $g(n) = n$ 의 시간이 걸리는 알고리즘보다 빨리 결과를 도출하는 경우는 절대 없다 $\longrightarrow$ 거짓
  - 이 알고리즘은 $O(n^3)$ 에도 속한다 $\longrightarrow$ 참
  - 이 알고리즘은 $O(2^n)$ 에도 속한다 $\longrightarrow$ 참

Reading

- 이번 주: Chapter 1-3
- 다음 주: Chapter 10

Wrap-up

- Regarding performance of algorithms, we are interested in solving large problems
- For sufficiently large problem, the execution time is dominated by its growth and denote it as a growth function
- We prefer slow growing algorithms to fast growing ones
- We can classify growth functions using Big-O, Big-Omega and Big-Theta analysis
- Each gives upper, lower and exact bounds for growth functions