[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.09.03 목요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

[주제 1]
Introduction to Algorithms and Complexity

Data Structure → Algorithm → Theory of Computation

- **어떻게 하면 주어진 복잡한 문제를 이진수 형태의 낮은 수준의 명령어만 이해하는 '단순한' 컴퓨터 상에서 효율적으로 해결할 수 있을까?**
  1. **[Data Structure]** 주어진 추상적인 문제를 어떠한 자료 구조를 사용하여 컴퓨터의 구조에 최적화된 형태로 표현할 수 있을까?
  2. **[Algorithm]** 주어진 추상적인 문제를 어떠한 알고리즘을 사용하여 컴퓨터를 사용하여 가장 효율적으로 해결할 수 있을까?
  3. **[Complexity]** 과연 컴퓨터가 주어진 문제를 효율적으로 해결할 수 있을까 ?
  4. **[Computability]** 과연 컴퓨터가 세상의 모든 문제를 해결할 수 있을까?

✓ 이 과목에서는 [CSE3080 자료구조] 과목에 이어, 1번과 2번을 집중적으로 살펴보고, 3번 문제에 대하여 어느 정도 살펴볼 예정임. 4번 문제는 [CSE3085 자동장치 이론] 과목에서 다룸.

Definition of Algorithm

[Horowitz 1.2]

- An **algorithm** is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:

  1) **Input.** Zero or more quantities from the outside.
  2) **Output.** At least one quantity is produced.
  3) **Definiteness.** Each instruction is clear and unambiguous.
  4) **Finiteness.** If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
  5) **Effectiveness.** Every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper. It is not enough that each operation be definite as in (3); it also must be feasible.

[Horowitz 1.2]를 다시 읽을 것!

어떤 입력 데이터가 들어와도 **유한한 시간 안에** 종료 해야함.

Thoughts on **4) Finiteness:** [Computability]

Input Size

- Problem (Post's correspondence problem)
  - Consider a finite set **P** of ordered pairs of nonempty strings such as **P** = {(a, ab), (b, ca), (ca, a), (abc, c)}.
  - A match of **P** is any string w such that, for some m > 0 and some pairs $(u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m)$ in **P**, $w = u_1 u_2 \ldots u_m = v_1 v_2 \ldots v_m$.
  - Design an algorithm that determine, given **P**, whether **P** has a match.
- Cheolsu's algorithm

a b ca a abc
ab ca a ab c

```
– For i = 1, 2, 3, … do,
    • For each permutation of P of length i, do
        – If it is a match, print 'yes' and exit.
        – If not, continue.
```

- Can this be regarded as **an algorithm**?

무엇보다 어떤 입력 데이터가 들어와도 "**현실적인" 시간 안에** 종료 해야함.

Thoughts on **Efficiency: [Complexity]**

- An algorithm is regarded as efficient or good if there exist a polynomial **P(n)** such that the time required for solving any instance of size **n** is bounded above by **P(n)**.
- **NP-Complete problems:**
  - Nobody has found so far any **good** algorithm for any problem in this class.
  - It has been proved that if a good algorithm exists for some algorithm in this class, then a good algorithm exists for all NP-Complete Problem.
- **Examples**
  - Suppose a CD-ROM can store up to 720MBytes of data. You have a sequence of n files of sizes $s_1, s_2, \ldots, s_n$ Mbytes, to dump into backup CDs. What is the minimum number of necessary CDs to store all the files?
  - Consider n tasks to be executed on CPU. All the tasks must be finished within the time requirement L (seconds). If the i-th task takes s_i seconds, and you can harness multiple processors, what would be the minimum number of processors needed to accomplish this?
  - ➢ Ex. $L = 10$, $n = 6$, and $(s_1, s_2, s_3, s_4, s_5, s_6) = (5, 6, 3, 7, 5, 4)$
    - ✓ (5, 5), (6, 4), (7, 3)

어떻게 하면 좀 더 "효율적으로" 문제를 해결할까?

Efficient Algorithm Design: Example 1

- Sequential search versus binary search
  - **Problem**: Determine whether x is in the sorted array S of n keys.
  - **Inputs**: positive integer n, sorted (nondecreasing order) arrays of keys S indexed from 0 to n - 1, a key x.
  - **Outputs**: the location of x in S (-1 if x is not in S).

  - Sequential search: $T(n) = O(n)$
  - Binary search: $T(n) = O(\log n)$

From Neapolitan

Array Size	Number of Comparisons by Sequential Search	Number of Comparisons by Binary Search
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

  - Why is the binary search more efficient?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	17	19	21	25	26	29	32	35	36	39	40	43	48	51	55

Efficient Algorithm Design: Example 2

- The Fibonacci Sequence
  - **Problem**: Determine the nth term in the Fibonacci sequence.
  - **Inputs**: a nonnegative integer n.
  - **Outputs**: the nth term of the Fibonacci sequence.

$$\boxed{f_0 = 0, \quad f_1 = 1, \quad f_n = f_{n-1} + f_{n-2} \text{ for } n \geq 2}$$

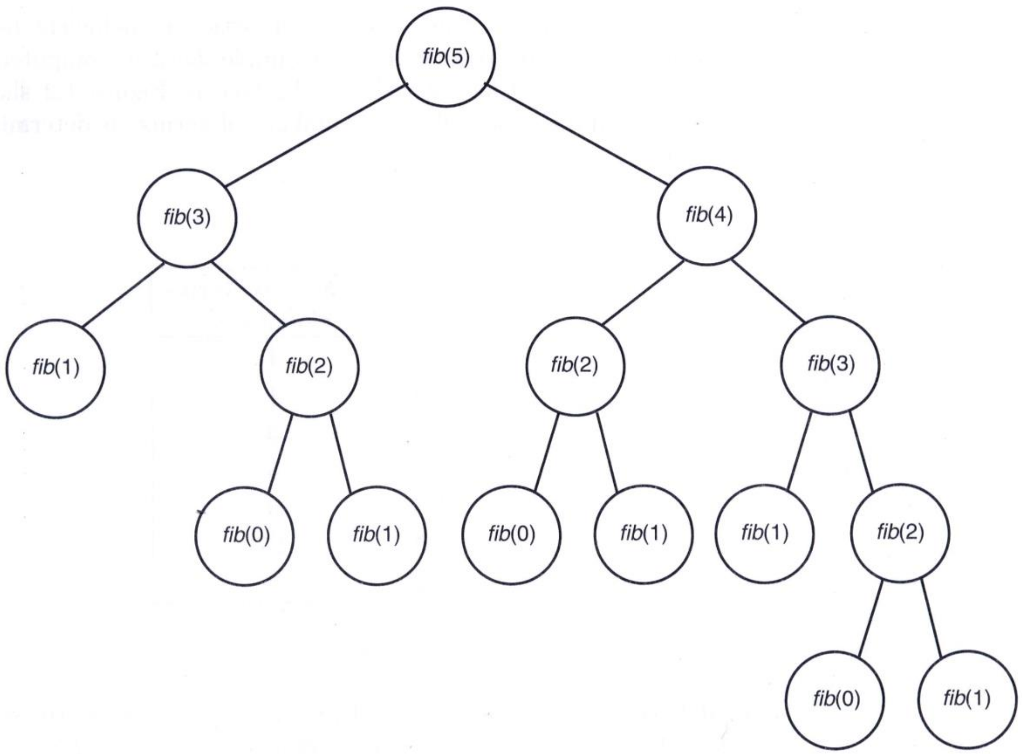
```
<recursive: divide-and-conquer>
int fib (int n) {
  if (n == 0) return 0;
  else if (n == 1) return 1;
  else return fib(n-1) + fib(n-2);
}
```

```
<iterative: dynamic programming>
int fib(int n) {
  index i;
  int f[0 .. n];

  f[0] = 0;
  if (n > 0) {
    f[1] = 1;
    for (i = 2; i <= n; i++)
      f[i] = f[i-1] + f[i-2];
  }
  return f[n];
}
```

- Recursive: $T(n) = O(2^n)$
- Iterative: $T(n) = O(n)$

- Why is the iterative version more efficient?



$$T(n) > 2^{\frac{n}{2}} \text{ for } n \geq 2$$

Mathematical induction을 써서 증명해볼 것!

- Linear versus exponential

From Neapolitan

n	$n+1$	$2^{n/2}$	Execution Time Using Algorithm 1.7	Lower Bound on Execution Time Using Algorithm 1.6
40	41	1,048,576	41 ns*	1048 μs†
60	61	1.1×10^9	61 ns	1 s
80	81	1.1×10^{12}	81 ns	18 min
100	101	1.1×10^{15}	101 ns	13 days
120	121	1.2×10^{18}	121 ns	36 years
160	161	1.2×10^{24}	161 ns	3.8×10^7 years
200	201	1.3×10^{30}	201 ns	4×10^{13} years

Order of Algorithms

- **Big *O***

For given two functions $f(n)$ and $g(n)$, $g(n) = O(f(n))$ if and only if there exists some positive real constant c and some nonnegative integer N such that $g(n) \leq c \cdot f(n)$ for all $n \geq N$.

> We say that $g(n)$ is big O of $f(n)$.

– Example:

```
x = x + 1;
for (i = 1; i <= n; i++)
  y = y + 2;
for (i = n; i >=1; i--)
  for (j = n; j >= 1; j--)
     z = z + 1;
```

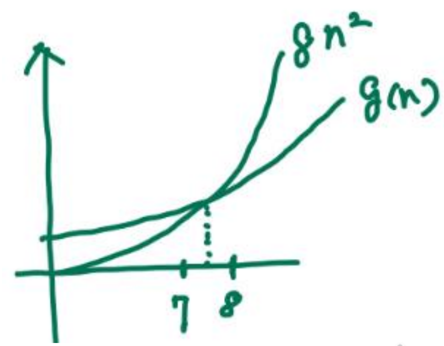
Time complexity: $c_0 + c_1 n + c_2 n^2 = O(n^2)$

비용 : $g(n) = c_0 + c_1 n + c_2 n^2$

예 : $g(n) = 5 + 6n + 7n^2$

$\rightarrow$ $g(n) \leq \underbrace{8 \cdot n^2}_{c \cdot f(n)}$ for all $n \geq \underbrace{8}_{N}$

$\Rightarrow$ $g(n) = O(n^2)$



- **Note 1:** The big *O* puts an asymptotic upper bound on a function.
  - **Asymptotic** analysis (from Wikipedia)

    If f(n) = n^2 + 3n, then as n becomes very large, the term 3n becomes insignificant compared to n^2. The function f(n) is said to be "asymptotically equivalent to n^2, as n → ∞". This is often written symbolically as f(n) ~ n^2, which is read as "f(n) is asymptotic to n^2".

    계산 비용이 $0.01n^2$인 알고리즘과 $100n$인 알고리즘 중 어떤 것이 더 "효율적"인가?

  - (Tight) **upper bound**
    - $37\log n + 0.1n = O(n)$
    - $n^2 + 10n = O(n^2)$
    - $4(\log n)^2 + n\log n + 100n = O(n\log n)$
    - $n^2 + 10n = O(n^{200})$???
    - …

Growth Rates of Some Common Complexity Functions