Chapter 1. 알고리즘과 문제의 분석_원리와 예제 (part 2)

알고리즘과 문제의 분석

알고리즘을 분석하는 목적 : 가능한 알고리즘을 개선하고 또 어떤 문제에 대한 여러 가지의 알고리즘 중에서 더 좋은 것을 선택하기 위함

평가기준

- 정확성
- 수행된 작업의 양
- 사용된 공간의 양
- 단순성과 명확성
- 최적성

정확성

알고리즘이 정확하다 → 작업을 하기 위한 **입력의 특징(선조건; preconditions)** 과 각 **입력에 대해 어떤 결과(후조건; postconditions)** 가 나오는 지에 대한 정확한 기술을 할 수 있어야함
⇒ 입력과 출력 사이의 관계, 즉 선조건을 만족한다면 알고리즘이 종료할 때 후조건을 만족하게 된다라는 사실을 증명 가능

알고리즘의 두 가지 측면

- 해를 구하는 방법론적인 측면
- 구현적인 측면 : 알고리즘을 수행하기 위한 명령어 순서

알고리즘은 보통 작은 모듈들로 구성되므로 작은 모듈들이 정확하다는 것을 증명하면 전체 프로그램이 올바르다는 것을 증명 가능

수행된 작업의 양

알고리즘에 의해 수행된 작업량의 측정 → 같은 문제에 대한 두 알고리즘을 비교하여 어느 것이 더 효율적인가를 결정하는데 도움을 주어야함
  * 수행시간이나 수행되는 명령어나 문장의 수 등은 컴퓨터나 언어에 따라 달라질 수 있으므로 사용하지 않음
  * 오버헤드 등에 무관하게 알로리즘에 의해 사용된 방법의 효율성을 나타낼 수 있는 작업의 측정치가 필요함
⇒ 알고리즘을 수행하는데 필수적인 연산(기본연산)을 추출하여 이 연산이 알고리즘에 의해 몇 번 수행되는지 확인하는 방법을 사용

알고리즘의 복잡도 : 어떤 복잡도 측정치에 의해 측정된 수행된 작업의 양

평균 분석과 최악의 경우 분석

알고리즘에 의해 수행되는 작업의 양을 분석하는 일반적인 방식을 정한 후에는 그 분석 결과를 간결하게 표현하는 방식이 필요
$\rightarrow$ 수행된 단계의 개수가 모든 입력에 대해 같을 수가 없으므로 수행된 작업의 양을 하나의 수로 나타낼 수 없고, 입력의 크기가 같더라도 어떤 입력이 들어오는가에 따라 수행되는 작업량이 달라짐 $\Rightarrow$ 대부분의 경우 최악의 경우 복잡도에 의해 알고리즘의 행동 양식을 설명

- **최악의 경우 복잡도**
  고려중인 문제에 대해 크기가 n인 입력의 집합을 D_n 이라 하고, I를 D_n의 원소라 한다. $t(I)$를 입력 I에 대해 알고리즘에 의해 수행되는 기본 연산의 수라고 할 때, 함수 W를 다음과 같이 정의한다

  $$W(n) = max\{t(I)|I \in D_n\}$$

  함수 $W(n)$을 알고리즘의 최악의 경우 복잡도 라고 부른다. $W(n)$은 크기가 n인 입력에 대해서 알고리즘에 의해 수행되는 기본 연산의 최대 개수이다

최악의 경우 복잡도는 **알고리즘에 의해 수행되는 작업에 대한 상한선** 을 의미

- **평균 복잡도**
  $Pr(I)$를 입력 I가 일어나는 확률이라고 했을 때, 알고리즘의 평균 복잡도는 다음과 같이 정의된다

  $$A(n) = \sum_{I \in D_n} Pr(I)t(I)$$

  $t(I)$는 알고리즘 분석을 통해 결정될 수 있지만, $Pr(I)$는 분석적으로 계산할 수 없고 경험적이나 가정에 의해 결정됨

- 일반화된 탐색 루틴
  데이터에 대한 탐색을 모두 다 하거나 그 목적을 달성할 때까지 수행하는 프로시저

  검색할 자료가 더 이상 없다면,
  실패
  그렇지 않다면
  하나의 자료를 검색한다
  이 자료가 원하는 자료라면
  성공
  그렇지 않다면,
  남은 자료에 대해 탐색을 계속한다

  이러한 방법을 일반화된 탐색 이라고 부르는 이유는 이 루틴이 자료를 이동하거나 자료 구조에 삽입과 삭제와 같은 단순한 연산을 할 때도 이용할 수 있기 때문이다

  - 예제 1) 무순서 배열 순차 탐색

```
E = [정리되지 않은 배열] # E -> 배열
n = len(E) # n -> E의 원소 개수

def seqSearch(E, n, K):
    for i in range(n):
        if K == E[i]:  # 이 부분이 기본 연산
            return i
    return -1
```

최악의 경우 분석

$W(n) = n$, K가 배열의 마지막에 있거나 배열에 없는 경우

이 경우 K가 n개의 모든 원소와 비교된다

평균 분석

가정: 배열의 모든 원소는 서로 다르고, 위치에 대한 확률은 모두 같다

$A_{succ}(n) = \sum_{i=0}^{n-1} Pr(I_i|succ)t(I_i) = \sum_{i=0}^{n-1}(\frac{1}{n})(i+1) = (\frac{1}{n})\frac{n(n+1)}{2} = \frac{n+1}{2}$ → 배열에 있는 경우

$A_{fail}(n) = n$ → 배열에 없는 경우

$A(n) = Pr(succ)A_{succ}(n) + Pr(fail)A_{fail}(n) = q((n+1)/2) + (1-q)n = n(1-q/2) + q/2$

→ 배열에 있는 경우와 없는 경우가 혼합

$t(I)$: 입력 I에 대해서 알고리즘에 의해 수행되는 비교 횟수 ($0 \leq i < n$ 에 대해 $t(I_i) = i+1$)

- 예제 2) 행렬 곱셈

```
# A : (m, n) 행렬
# B : (n, p) 행렬
# C = AB

def matmul(A, B, m, n, p):
    C = list()
    for i in range(m):
        for j in range(p):
            C[i][j] = 0
            for k in range(n):
                c[i][j] = A[i][k] + B[k][j]
    return C
```

이 경우 mnp만큼의 곱셈이 필요

공간 사용도

프로그램 실행에 필요한 공간의 양은 구현에 따라 다르지만 단순히 알고리즘을 검토해서 공간 사용도에 대한 판정을 내릴 수 있음

* 입력 그대로의 형태(수의 배열 혹은 행렬)만 필요하다면 부가적으로 사용된 공간을 분석
* 부가적인 공간의 양이 입력의 크기에 대해 상수라면 알고리즘은 **자체 공간 작업**이 가능하다고 말한다
* 입력이 어려 형태로 표현될 수 있다면 사용되는 부가적인 공간은 물론 입력 자체에 필요한 공간까지 고려

단순성

문제를 해결하는 단순하고 직관적인 방법은 대부분 효율적이지 못하지만 단순한 알고리즘은 정확성을 쉽게 확인하거나 구현, 디버깅 측면에서 편리

최적성

아무리 뛰어나다고 해도 어떤 문제에 대한 알고리즘을 더 이상 개선할 수 없음$\rightarrow$ 각 문제는 본래부터 가지고 있는 복잡성이 있음(그 문제를 풀기위한 최소한의 작업량이 존재)

어떤 문제의 복잡도를 분석하기 위해서는 알고리즘의 클래스(알고리즘이 수행되는 데 허용된 연산의 유형)와 복잡도 측정치를 선택해야 함
 * 어떤 알고리즘이 최적이라는 말은 그 클래스 내에 더 적은 수의 연산으로 수행되는 알고리즘이 존재하지 않는다는 것을 의미(최악의 경우에)
 * 지금까지 개발된 알고리즘뿐만 아니라 아직까지 개발되지 않은 모든 가능한 알고리즘을 포함
 * **최적** 이라는 것은 알려진 것 중 가장 좋은 것 이 아니라 가능한 것 중 가장 좋은 것 을 뜻함

하한선과 문제의 복잡도

최적의 알고리즘 증명 $\Rightarrow$ 문제를 해결하기 위해 필요한 연산의 개수에 대한 하한선을 설정하는 정리를 증명 가능

어떤 문제를 풀기 위해 필요하고도 충분한 작업의 양은 얼마인가?
 * 효율적이라고 생각되는 알고리즘 A를 개발하고 입력의 크기 n에 대해서 A의 최악의 경우 복잡도 함수 $W_A(n)$을 찾는다
 * 어떤 함수 F에 대하여, 해당 클래스 내의 어떠한 알고리즘도 적어도 $F(n)$ 단계를 수행해야 하는 크기가 n인 입력이 존재한다는 정리를 증명한다
 $\Rightarrow$ 함수 W_A와 F가 같다면 그 알고리즘 A는 (최악의 경우에서) 최적이다
 $\Rightarrow$ 그렇지 않다면 더 좋은 알고리즘이 있든지 아니면 더 좋은 하한선이 존재

점근적인 증가율에 의한 함수 분류

알고리즘의 분석은 그 알고리즘에 의해 수행되는 모든 단계를 세는 것이 아니기 때문에 부정확할 수 밖에 없음 $\rightarrow$ 기본 연산의 개수만 가지고 비교
(두 알고리즘의 시간 복잡도 함수가 상수배만큼 차이가 난다면 그 둘 사이를 비교하는 것은 무의미)

정의와 점근적 표기법

- 집합 $O(g)$
  - g가 음이 아닌 정수로부터 양의 실수로 사상되는 함수일 때, $O(g)$는 $c > 0$인 어떤 실수 상수와 음이 아닌 어떤 정수 상수 n_0에 대해서, $n \geq n_0$인 모든 n에 대해 $f(n) \leq cg(n)$을 만족하는 함수 f의 집합을 말한다 (이 때 함수 f도 음이 아닌 정수로부터 양의 실수로 사상되는 함수이다)
  - 모든 n에 대해서 항상 $f(n) > g(n)$이더라도 함수 f는 $O(g)$에 속할 수 있다 $\rightarrow$ 중요한 것은 f는 g의 상수배를 넘지 못한다는 것이고 n값이 작은 경우는 고려하지 않음
  - 집합 $o(g)$는 "big oh of g" 혹은 "oh of g"라고 부른다
  - 극한값이 0인 경우를 포함해서 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$ 이면, $f \in O(g)$
    즉, f를 g로 나눈 값에 대한 극한값이 존재하고 그 값이 무한대가 아니라면 f는 g보다 빨리 증가하지 않는다

- L`Hopital의 법칙
  - f와 g 가 다음 식을 만족하면서 미분 가능이고, 각각의 도함수를 $f\prime$와 $g\prime$라고 할 때, $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ 이면 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f\prime(n)}{g\prime(n)}$
- $\Omega(g)$
  - g가 음이 아닌 정수로부터 양의 실수로 사상되는 함수일 때 $\Omega(g)$는 $C > 0$인 어떤 실수 상수와 음이 아닌 어떤 정수 상수 n_0에 대해, $n \geq n_0$인 모든 n에 대해서 $f(n) \geq c \cdot g(n)$을 만족하는 함수 f의 집합을 말한다 이 때 함수 f도 음이 아닌 정수로부터 양의 실수로 사상되는 함수이다)
  - 극한값이 ∞인 경우를 포함해서 $\lim_{n\to\infty} \frac{f(n)}{g(n)}$이면, 함수 $f \in \Omega(g)$이다
- 집합 $\Theta(g)$, g의 점근적 차수
  - g가 음이 아닌 정수로부터 양의 실수로 사상되는 함수일 때 $\Theta(g) = O(g) \cap \Omega(g)$이다. 즉, 집합 $O(g)$와 $\Omega(g)$ 양 쪽 모두에 포함된 함수의 집합이다
  - 정확성을 위해 점근적 차수*(asymptotic order)* 라는 말이나 점근적 복잡도*(asymptotic complexity)*라는 말을 쓰기도 함
  - $0 < c < \infty$인 어떤 상수 c에 대해 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$라면, $f \in \Theta(g)$
- 집합 $o(g)$와 $\omega(g)$
  - g가 음이 아닌 정수로부터 양의 실수로 사상되는 함수일 때,
  1. $o(g)$는 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$을 만족하는 함수 f의 집합
  2. $\omega(g)$는 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$을 만족하는 함수 f의 집합

O, Ω와 Θ의 성질

- 만약 $f \in O(g)$이고, $g \in O(h)$라면, $f \in O(h)$이다. 즉, O은 이행적이다. 마찬가지로 $\Omega, \Theta, o, \omega$도 이행적이다.
- 보조정리들
  - $f \in O(g)$는 $g \in \Theta(f)$이기 위한 필요충분조건이다
  - 만약 $f \in \Theta(g)$이면, $g \in \Theta(f)$이다
  - Θ는 함수에 대해 동치 관계를 정의한다. 집합 $\Theta(f)$는 복잡도 부류라고 불리는 동치 부류이다
  - $O(f + g) = \mathrm{O}(max(f, g))$이다. Ω, Θ에 대해서도 비슷한 등식이 성립한다.
- 많이 쓰이는 정리들
  - $lg(n)$은 어떠한 $\alpha > 0$에 대해서도 $o(n^\alpha)$에 속한다. 즉, 로그함수는 어떠한 α값에 대해서도 n^α보다 매우 천천히 증가한다
  - n^k는 어떠한 $k > 0$에 대해서도 $o(2^n)$에 속한다. 즉, n^k은 지수함수 2^k보다 매우 천천히 증가한다

자주 발생하는 합산에 대한 점근적 차수

d를 음이 아닌 상수, r이 1이 아닌 양의 상수라 할 때,

1. 다항 급수의 합은 지수 값을 1 증가시킨다. 차수가 d인 다항급수의 합은 $\sum_{i=1}^{n} i^d$ 형태의 합이고, 이는 $\Theta(n^{d+1})$에 속한다
2. 기하 급수이 합은 가장 큰 항의 Θ에 속한다. 기하 급수의 합은 $\sum_{i=a}^{b} r^i$의 형태이고, 이러한 법칙은 $0 < r < 1$이거나 $r > 1$일 때 적용되지만 $r = 1$일때는 적용되지 않는다. a, b는 두 값이 동시에 상수는 아니며, 일반적으로 상한값 b는 n에 대한 함수, 하한값 a는 상수이다
3. 로그 급수의 합은 $\Theta(\text{항의 개수} \times \text{가장 큰 항의 로그})$ 내에 속한다. 로그 급수는 $\sum_{i=1}^{n} \log(i)$의 형태이며 이런 종류의 합은 $\Theta(n \log(n))$에 속한다. 점근적 차수를 말할 때 로그의 밑은 상관이 없다

4. 다항-로그 급수의 합은 $\sum_{i=1}^{n} i^d \log(i)$의 형태의 합으로써 이는 $\Theta(n^{d+1} \log(n))$내에 속한다