알고리즘및실습

제4장 도사 정리

1. 개요

재귀 알고리즘의 시간 복잡도를 분석하는 것은 간단하지 않다. 1장에서 분할 정복의 시간 복잡도를 분석한 것처럼 재귀 알고리즘의 재귀 호출 트리를 분석하여야 한다. 비재귀적 비용이 $O(1)$이면 재귀 호출 트리에 있는 전체 노드의 개수가 시간 복잡도를 결정한다. 예를 들어 2장에서 살펴본 cansum 문제의 경우에는 전체 노드의 개수가 n^m보다 적고, 비재귀적 비용이 $O(1)$이므로 cansum의 빅O는 $O(n^m)$이다.

비재귀적 비용이 $O(1)$이 아니면 분석이 복잡해진다. 분할 정복처럼 각 레벨마다 비용을 분석하여 전체 비용을 계산하는 방법도 있지만 모든 재귀 알고리즘을 이와 같이 분석할 수 있는 것은 아니다. **도사 정리**(the master theorem)는 재귀 알고리즘의 **점화식**(recurrence)을 이용하여 시간 복잡도를 분석하여 주는 분석 도구이다.

도사 정리를 이용하기 위해서는 분석하고자 하는 재귀 알고리즘이 표준 점화식에 해당해야 한다. 표준 점화식에 해당하면 각 레벨마다 비용을 분석하여 시간 복잡도를 계산할 수 있는데, 직접 분석할 필요 없이 표준 점화식의 3가지 유형에 따른 분석 결과를 활용해 쉽게 재귀 알고리즘의 시간 복잡도를 제시해 주는 도구이다. 이 장에서는 표준 점화식, 도사 정리를 사용하는 방법을 살펴본다.

도사 정리는 블랙박스처럼 내부 원리를 이해하지 않고 기계적으로 활용할 수 있지만 도사 정리가 성립하는 이유를 알면 재귀 알고리즘에 대한 이해를 높일 수 있으므로 내부 원리를 이해하기 위해 도사 정리의 증명도 살펴본다.

2. 정수 곱셈

도사 정리를 사용하기 위해서는 분석하고자 하는 재귀 알고리즘의 점화식을 구하여야 한다. 1장에서 살펴본 두 개의 n자리 수 곱셈 문제를 해결하는 분할 정복 알고리즘의 점화식을 구하여 보자. 우리는 1장에서 두 개의 n자리를 각 두 개의 $n/2$자리로 나누어 다음과 같이 분할 정복하는 알고리즘을 살펴보았고, Karatsuba 알고리즘도 살펴보았다.

$$X = 10^{n/2}a + b, Y = 10^{n/2}c + d, X \times Y = 10^n(ac) + 10^{n/2}(ad + bc) + bd$$

이 알고리즘은 4번의 $n/2$자리 수 곱셈, 3번의 덧셈, 2번의 자리 이동 연산이 필요하다. 덧셈과 자리 이동 연산의 시간 복잡도는 $O(n)$이다. 따라서 이 알고리즘의 점화식은 다음과 같다.

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

여기서 4는 재귀 호출의 수이고, $\frac{n}{2}$은 재귀 호출할 때 소문제의 크기이며, $O(n)$은 비재귀적 부분의 비용이다. 더 정확하게 점화식을 제시하기 위해서는 일반적인 경우 뿐만 아니라 기저 경우(base case)에 대한 제시도 필요하다. 이 알고리즘의 기저 경우는 $T(1) = O(1)$이다. 점화식을 다른 말로 재귀식이라고도 한다.

Karatsuba 알고리즘은 앞서 제시한 알고리즘과 다르게 다음과 같이 진행된다.

$$X = 10^{n/2}a + b, Y = 10^{n/2}c + d,$$
$$A = ac, B = bd, C = (a+b)(c+d) - A - B, X \times Y = 10^n A + 10^{n/2}C + B$$

이 알고리즘은 3번의 $n/2$자리 수 곱셈, 6번의 덧셈, 2번의 자리 이동 연산이 필요하다. 덧셈은 오히려 늘어났지만 여전히 비재귀적 부분의 비용은 $O(n)$이며, 이 알고리즘의 점화식은 다음과 같다.

$$T(n) \leq 3T\left(\frac{n}{2}\right) + O(n)$$

점화식만 보면 일반 분할 정복 곱셈 알고리즘보다 Karatsuba가 더 효과적이다는 것은 쉽게 인지할 수 있지만 얼마나 더 효율적인지는 식으로부터는 쉽게 알기 어렵다. 참고로 합병 정렬 알고리즘의 점화식은 $T(n) \leq 2T\left(\frac{n}{2}\right) + O(n)$이므로 두 알고리즘이 모두 합병 정렬보다는 느리다는 것을 알 수 있다. 물론 서로 다른 문제를 해결하는 알고리즘을 비교할 이유는 없지만 점화식의 비교를 통해 시간 복잡도의 수준이 $O(n \log n)$과 $O(n^2)$ 사이에 있을 것이라는 것은 짐작할 수 있다.

3. 표준 점화식

재귀 알고리즘의 점화식이 다음과 같이 표현되면 표준 점화식이라 한다.

- 조건 1. 다음을 만족하는 n과 독립적인 n_0와 c가 존재해야 한다.

$$T(n) \leq c, \text{ if } n \leq n_0$$

- 조건 2. $n > n_0$이면 점화식은 다음과 같이 표현되어야 한다.

$$T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$$

  여기서 $a(\geq 1)$, $b(> 1)$, d는 n과 독립적인 상수이다.

위 조건의 의미를 좀 더 구체적으로 살펴보자. 조건 1은 기저 경우를 제시할 수 있어야 하며, 입력 크기가 충분히 작으면 $O(1)$에 해결할 수 있어야 한다는 것을 의미한다. 조건 2에서 a는 재귀 호출의 수이고, b는 입력 크기가 줄어드는 정도를 나타내며, d는 비재귀적 과정에서 소요되는 시간 복잡도의 차수를 나타낸다.

조건 2의 형태로 재귀 알고리즘을 표현하기 위해서는 같은 레벨에서 이루어지는 모든 재귀 호출의 입력 크기는 항상 같은 수준으로 줄어야 한다 또 비재귀적 부분의 비용이 다차 시간이어야 한다. 예를 들어 비재귀적 부분의 비용이 지수 시간이면 도사 정리를 적용할 수 없다. 합병 정렬 알고리즘은 표준 점화식으로 표현할 수 있으며, 표준 점화식에서 $a = 2$, $b = 2$, $d = 1$이다.

4. 도사 정리

실제 도사 정리는 다음에 제시된 정리보다 조금 더 복잡하다. 이에 다음에 제시된 정리는 단순화한 도사 정리라 한다.

정리 4.1 (도사 정리)**.** 표준 점화식으로 표현할 수 있는 재귀 알고리즘의 시간 복잡도는 다음과 같다.

- 경우 1. $a = b^d$: $T(n) = O(n^d \log n)$
- 경우 2. $a < b^d$: $T(n) = O(n^d)$
- 경우 3. $a > b^d$: $T(n) = O(n^{\log_b a})$

경우 1에서 기저가 바뀌더라도 그 차이가 일정한 상수만큼의 차이[1]만 발생하며, 이 차이는 빅O에 의해 무시되므로 경우 1에서 log의 기저는 중요하지 않다. 하지만 경우 3에서 기저는 중요하다. 경우 2는 비재귀적인 부분의 비용 자체가 알고리즘의 시간 복잡도로 제시되고 있다. 즉, 경우 2 형태의 알고리즘에서 재귀 비용은 무시된다는 것을 의미한다. 이 이유와 도사 정리에서 왜 a와 b^d를 비교하는 이유는 도사 정리의 증명을 살펴보면 쉽게 이해할 수 있다. 도사 정리의 증명을 살펴보기 전에 몇 가지 재귀 알고리즘에 도사 정리를 적용해 알고리즘의 시간 복잡도를 분석하여 보자.

- 예1) 합병 정렬: $a = 2$, $b = 2$, $d = 1$
  $b^d = 2 = a$이므로 경우 1에 해당한다. 따라서 합병 정렬의 시간 복잡도는 $O(n \log n)$이다.
- 예2) 이진 검색: $a = 1$, $b = 2$, $d = 0$
  $b^d = 1 = a$이므로 경우 1에 해당한다. 따라서 이진 검색의 시간 복잡도는 $O(\log n)$이다.
- 예3) 일반 분할 정복 곱셈 알고리즘: $a = 4$, $b = 2$, $d = 1$
  $b^d = 2 < a = 4$이므로 경우 3에 해당한다. 따라서 이 알고리즘의 시간 복잡도는 $O(n^{\log_2 4}) = O(n^2)$이다.
- 예4) Karatsuba 곱셈 알고리즘: $a = 3$, $b = 2$, $d = 1$
  $b^d = 2 < a = 3$이므로 경우 3에 해당한다. 따라서 이 알고리즘의 시간 복잡도는 $O(n^{\log_2 3}) = O(n^{1.59})$이다.
- 예5) 일반 분할 정복 행렬 곱셈 알고리즘: $a = 8$, $b = 2$, $d = 2$
  $b^d = 4 < a = 8$이므로 경우 3에 해당한다. 따라서 이 알고리즘의 시간 복잡도는 $O(n^{\log_2 8}) = O(n^3)$이다.
- 예6) Strassen 행렬 곱셈 알고리즘: $a = 7$, $b = 2$, $d = 2$
  $b^d = 4 < a = 7$이므로 경우 3에 해당한다. 따라서 이 알고리즘의 시간 복잡도는 $O(n^{\log_2 7}) = O(n^{2.81})$이다.
- 예7) 가상의 알고리즘: $a = 2$, $b = 2$, $d = 2$
  $b^d = 4 > a = 2$이므로 경우 2에 해당한다. 따라서 이 알고리즘의 시간 복잡도는 $O(n^2)$이다.

합병 정렬처럼 두 개의 $n/2$ 크기로 나누어 문제를 해결하는 분할 정복 알고리즘의 경우 비재귀적 부분이 $O(n)$이면 시간 복잡도가 $O(n \log n)$이지만 비재귀적 부분이 $O(n^2)$이면 시간 복잡도가 $O(n^2)$이 된다. 보통 입력 크기가 n인 것을 전수조사나 일반 알고리즘으로 해결하였을 때 시간 복잡도가 $O(n^2)$인 경우 절반으로 나누어 분할 정복하는 알고리즘을 고안하였을 때 비재귀적 부분이 $O(n)$ 이하가 아니면 분할 정복하는 것이 아무 의미가 없다.

5. 도사 정리의 증명

도사 정리의 증명을 이해하기 위해 먼저 1장에서 살펴본 합병 정렬의 시간 복잡도 분석을 다시 생각하여 보자. 우리는 먼저 합병 정렬 알고리즘의 재귀 호출 트리를 그려 보았다. 레벨이 내려가면서 입력 크기가 항상 절반으로 줄어들기 때문에 총 레벨 수는 $\log_2 n + 1$이다. 또 비재귀적 부분의 비용은 cn이었다. 구체적으로 각 레벨 j에서 2^j개의 합병이 필요하고, 입력 배열의 크기가 $n/2^j$이므로 각 레벨 j에서 연산 수는 $2^j \times c\left(\frac{n}{2^j}\right) = kn$이다.

[1] $\log_b n = \frac{\log_a n}{\log_a b}$이다.

합병 정렬에서 사용한 이 분석을 일반화 해보자. a가 재귀 호출의 수이면 각 레벨에서 a^j개의 재귀 호출이 이루어지며, 각 재귀 호출의 입력 크기는 $\frac{n}{b^j}$이다. 따라서 특정 레벨 j에서 필요한 일은 다음과 같다.

$$\leq a^j \times c \times \left(\frac{n}{b^j}\right)^d = cn^d \left(\frac{a}{b^d}\right)^j$$

모든 레벨에서 소요되는 일을 합하면 전체 비용이 되며, 이는 다음과 같다.

$$T(n) \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$

따라서 a와 b^d의 관계에 의해 이 식의 좌우된다는 것을 쉽게 알 수 있다. 이를 이용한 것이 바로 도사 정리이다.

표준 점화식에서 a는 재귀 호출 수이며, a가 커질수록 재귀 호출 수가 기하급수적으로 늘어난다. 이 때문에 a를 RSP(Rate of Subproblem Proliferation)이라 하며, 시간 복잡도에 나쁜 영향을 주는 요소이다. 반면에 b^d는 문제의 크기가 축소되는 비율이며, b가 클수록 재귀 호출 트리의 높이가 낮아지기 때문에 더 빨리 재귀 호출이 종료하게 된다. 이 때문에 b^d를 RWS(Rate of Work Shrinkage)라 하며, 시간 복잡도에 좋은 영향을 주는 요소이다.

RSP와 RWS의 관계는 크게 다음 3가지 경우로 나눌 수 있다.

- 경우 1. RSP=RWS: 각 레벨에서 소요되는 전체 비용이 같은 경우이다.
- 경우 2. RSP<RWS: 레벨이 증가할수록 비용이 줄어드는 경우이다.
- 경우 3. RSP>RWS: 레벨이 증가할수록 비용이 증가하는 경우이다.

경우 2는 대부분의 일이 루트 레벨에서 이루어지는 경우이며, 반대로 경우 3은 대부분의 일이 가장 높은 레벨(단말 레벨)에서 이루어지는 경우이다.

각 경우를 수학적으로 좀더 자세히 분석하여 보자. 경우 1은 $a = b^d$인 경우이며, 이 경우 $\frac{a}{b^d} = 1$이므로 전체 비용은 다음과 같다.

$$T(n) \leq cn^d(\log_b n + 1) = O(n^d \log n)$$

경우 2는 $\frac{a}{b^d} < 1$인 경우이며, 경우 3은 $\frac{a}{b^d} > 1$이다. $r \neq 1$일 때 $\sum_{i=0}^{k} r^i = \frac{r^{k+1}-1}{r-1}$이며, $r < 1$이면 $\frac{r^{k+1}-1}{r-1} \leq \frac{1}{1-r} = c$이며, $r > 1$이면 $\frac{r^{k+1}-1}{r-1} \leq \frac{r^{k+1}}{1-r} = r^k \frac{r}{r-1}$이다. 이것을 이용하면 경우 2의 분석도 다음과 같이 간단하다.

$$T(n) \leq cn^d c' = O(n^d)$$

경우 3은 다음과 같다.

$$T(n) \leq cn^d \left(\frac{a}{b^d}\right)^{\log_b n} = cn^d a^{\log_b n} b^{-d \log_b n} = cn^d a^{\log_b n} n^{-d} = ca^{\log_b n} = cn^{\log_b a} = O(n^{\log_b a})$$

6 점화식과 시간 복잡도

재귀 함수의 점화식이 표준 점화식에 해당하면 도사 정리를 블랙 박스 도구처럼 사용하여 알고리즘의 시간 복잡도를 분석할 수 있다. 도사 정리를 이용하지 않고 점화식 자체를 해결하여 시간 복잡도를 분석할 수 있다. 예를 들어 이진 검색의 점화식은 다음과 같다.

$$T(n) = T(\frac{n}{2}) + 1, T(1) = 1$$

이 점화식을 직접 풀어 시간 복잡도를 분석하여 보자. 몇 가지 항에 대해 점화식의 값을 구해보고, 얻은 값을 통해 해를 추정한 다음, 추정한 해를 귀납법을 이용하여 증명하여 점화식을 해결할 수 있다. 이진 검색의 점화식의 값을 구해보면 $T(2) = T(1)+1 = 2, T(4) = T(2)+1 = 3, T(8) = T(4)+1 = 4$와 같다. 이 값들을 통해 $T(n) = \log n+1$으로 추정할 수 있다. 이 추정이 맞다는 것을 귀납법을 이용하여 증명해야 한다.

- 귀납 출발. $T(1) = \log 1 + 1 = 0 + 1 = 1$이므로 성립한다.
- 귀납 가정. $n(\geq 1)$이 2의 거듭제곱일 때 $T(n) = \log n + 1$이 성립한다고 가정하자.
- 귀납 절차. $T(2n) = \log 2n + 1$이 성립한다는 것을 귀납 가정을 이용하여 다음과 같이 증명할 수 있다.

$$\begin{aligned} T(2n) &= T(\frac{2n}{2}) + 1 = T(n) + 1 = \log n + 1 + 1 \\ &= \log n + \log 2 + 1 = \log 2n + 1 \end{aligned}$$

물론 n이 2의 거듭제곱인 경우에 대해서만 성립한다.

점화식이 주어졌을 때 도사 정리를 이용하지 않고 이와 같은 방법으로 시간 복잡도를 분석할 수 있다.

퀴즈

1. 다음 중 도사 정리를 적용할 수 있는 점확식은?
   ① $T(n) = nT\left(\frac{n}{2}\right) + O(n)$
   ② $T(n) = \frac{1}{2}T\left(\frac{n}{2}\right) + O(n)$
   ③ $T(n) = 2T(n) + O(n^2)$
   ④ $T(n) = 2T\left(\frac{n}{2}\right) + O(1)$
2. 어떤 재귀 알고리즘의 표준 점화식이 다음과 같다.

$$T(n) \leq 3T(\frac{n}{3}) + O(n)$$

   이 알고리즘의 시간 복잡도는?
   ① $O(n \log n)$
   ② $O(n^3)$
   ③ $O(n)$
   ④ $O(n^2)$
3. 어떤 재귀 알고리즘의 표준 점화식이 다음과 같다.

$$T(n) \leq T(\frac{n}{2}) + O(n)$$

   이 알고리즘의 시간 복잡도는?
   ① $O(n \log n)$
   ② $O(n^2)$
   ③ $O(n)$
   ④ $O(\log n)$
4. 2장에서 살펴본 cansum 알고리즘은 도사 정리로 시간 복잡도를 분석할 수 없다. 그 이유로 틀린 것은?
   ① 재귀 호출하는 수가 입력 중 하나인 배열 크기만큼 필요하기 때문에 표준 점화식에서 a가 n과 독립적인 상수가 아니다.

② 비재귀 과정에서 소요되는 비용이 상수 비용이다.

③ 재귀 호출 과정에서 m이 감소하는데, 배열에 있는 값에 따라 감소하는 비율이 다르다. 표준 점화식에서 b를 결정할 수 없다.

④ 재귀 트리의 높이가 최초 목표값과 배열에 있는 값에 의해 결정된다.

연습문제

1. 하노이탑 문제는 다음과 같은 알고리즘을 이용하여 해결할 수 있다.

```
1: procedure HANOITOWER(n, src, dest, tmp)
2:     if n = 1 then
3:         MOVE(1, src, dest)
4:     else
5:         HANOITOWER(n − 1, src, tmp, dest)
6:         MOVE(1, src, dest)
7:         HANOITOWER(n − 1, tmp, dest, src)
```

   이 알고리즘이 점화식을 제시하고, 이 알고리즘의 시간 복잡도를 제시하라.

2. 주어진 다음 각 점화식을 도사 정리에 적용하여 알고리즘의 시간 복잡도를 제시하라.
   참고. 표준 점화식의 비재귀 부분 $f(n)$이 다차 시간이 아니라 $n^k \log^p n$ 형태일 수 있다. 이 경우 $\log_b a > k$ 이면 경우 3에 해당하며, $\log_b a = k$이면 경우 2이고, $\log_b a < k$이고 $p >= 0$이면 경우 1에 해당하고, $T(n) = O(n^k log^p n)$이다.

   ① $T(n) = 3T(n/2) + O(n^2)$

   ② $T(n) = 4T(n/2) + O(n^2)$

   ③ $T(n) = 2T(n/2) + O(1)$

   ④ $T(n) = 3T(n/4) + O(n \log n)$

   ⑤ $T(n) = 6T(n/3) + O(n^2 \log n)$

   ⑥ $T(n) = 2T(n/4) + n^{0.5}$