알고리즘

3장 점화식과 점근적 복잡도 분석

학습 내용

1. 점화식
2. 점화식의 점근적 분석 방법

학습 목표

- 재귀 알고리즘과 점화식의 관계를 이해한다.
- 점화식의 점근적 분석을 이해한다.

학습 내용

1. 점화식

2. 점화식의 점근적 분석 방법

점화식의 이해

- 재귀적(recursive) 성질을 갖는 알고리즘의 복잡도 분석할 때 점화식 이용
- 점화식(recurrence relation)
  - 어떤 함수를 자신보다 더 작은 변수에 대한 동일 함수와의 관계로 표현한 것
  - 즉, 동일한 함수가 등호/부등호 양쪽에 나타나는데, 양쪽에 나타난 함수의 변수 크기가 다르다.
  - 예 :

$$a_n = a_{n-1} + 2$$

$$f(n) = f(n/2) + n$$

$$fib(n) = fib(n-1) + fib(n-2)$$

$$factorial(n) = n * factorial(n-1)$$

점화식과 점근적 분석

- 입력 크기가 작은 경우에는 알고리즘의 효율성 차이가 크지 않음
  - 알고리즘 분석은 입력의 크기가 충분히 큰 경우만 관심의 대상임
- 점근적 분석(asymptotic analysis)
  - 입력의 크기가 충분히 큰 경우에 대한 분석
  - 점근적 표기법(asymptotic notation)을 이용하여 표시
    - $\boldsymbol{O}$-notation, $\boldsymbol{\Omega}$-notation, $\boldsymbol{\Theta}$-notation 등

점화식과 점근적 분석

- 재귀 알고리즘의 수행시간을 점근적 표기법으로 나타내려면
  1) 재귀 알고리즘의 수행시간을 점화식으로 표현한다.
  2) 이렇게 얻은 점화식을 점근적 표기법으로 나타낸다.

점화식을 사용하지 않는 경우

```
sample(A[ ], n)
{
    sum ← 0 ;
    for i ← 0 to n-1
        sum ← sum + A[i];
    return sum ;
}
```

이 알고리즘은 재귀 알고리즘이 아니므로 수행시간을 점화식으로 표현하지 않는다.

- ✓ n에 비례하는 시간이 소요된다.
- ✓ 수행시간 T(n)은?
- ✓ T(n)을 점근적 표기법으로 나타내면?

점화식을 사용하는 경우

재귀 알고리즘의 수행시간은 점화식으로 표현할 수 있다.

병합 정렬 알고리즘 → 4장에서 배움

```
mergeSort(A[ ], p, r) ▷ A[p … r]을 정렬한다.
{
        if (p < r) then {
                q ← (p+r)/2;   ------------  ①    ▷ p, r의 중간 지점 계산
                mergeSort(A, p, q);  --------  ②    ▷ 전반부 정렬
                mergeSort(A, q+1, r); -------  ③    ▷ 후반부 정렬
                merge(A, p, q, r);   ---------  ④    ▷ 병합
        }
}
```

✓입력 크기가 n인 병합정렬 수행시간
= 크기가 $n/2$인 병합정렬을 2번 수행한 시간 + 후처리 시간

⇩

1) 수행시간의 점화식: $T(n) = 2T(n/2)$ + 후처리 시간

⇩

2) 점근적 표기: $T(n) = \Theta(n \log n)$

학습 내용

1. 점화식

2. 점화식의 점근적 분석 방법

점화식의 점근적 분석 방법

- 점화식으로부터 점근적 복잡도를 구하는 방법
  (1) 반복 대치(repeated substitution)
    - 더 작은 문제에 대한 함수로 반복해서 대치해 나가는 해법

  (2) 추정 후 증명(guess and verification)
    - 결론을 추정하고 수학적 귀납법을 이용하여 증명하는 방법

  (3) 마스터 정리(master theorem)
    - 점화식의 형식이 $\mathbf{T(n) = aT(n/b) + f(n)}$ 인 경우, 복잡도를 바로 구할 수 있는 theorem

(1) 반복 대치

$T(n)$ 을 $T(n-1) + c$ 로 대치하는 과정을 반복하여 점화식을 전개

n! 을 구하는 알고리즘

```
factorial(n)
{
        if (n = 1)  return  1;
        return   n * factorial(n − 1);
}
```

1) 수행시간 $T(n)$을 점화식으로 표현하면 다음과 같다.

$$T(n) = T(n-1) + c \quad (c\text{는 상수})$$
$$T(1) \le c$$

2) 점화식의 점근적 복잡도를 오른쪽과 같은 반복 대치 방법으로 구할 수 있다.

$$\begin{aligned} T(n) &= T(n-1) + c \\ &= T(n-2) + c + c \\ &= T(n-3) + c + c + c \\ &\dots \\ &= T(1) + c + c + \dots + c \\ &= T(1) + (n-1)\, c \\ &\le c + (n-1)\, c \\ &= cn \end{aligned}$$

$\rightarrow$ $\boxed{T(n) = O(n)}$

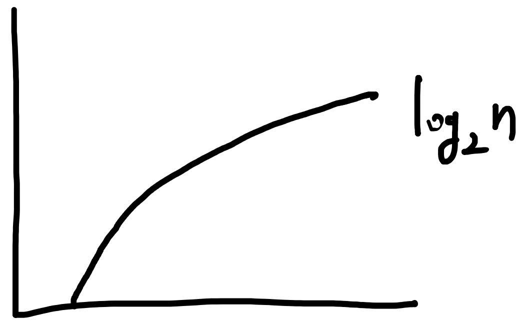
반복 대치

$T(n) = T(n-1) + c$ (c는 상수)
$T(1) \leq c$

$$
\begin{aligned}
T(n) &= T(n-1) + c \\
&= T(n-2) + c + c \\
&= T(n-3) + c + c + c \\
&\cdots \\
&= T(1) + c + c + \ldots + c \\
&= T(1) + (n-1)\,c \\
&\leq c + (n-1)\,c \\
&= cn
\end{aligned}
$$

➔ $T(n) \leq cn$ 이므로
$T(n) = O(n)$

참고: 로그 함수



$$\log_2 2 = 1$$

$$\log_2 2^3 = 3$$

$$\log_2 1 = 0$$

$$\log_c ab = \log_c a + \log_c b$$

$$\log_c \frac{a}{b} = \log_c a - \log_c b$$

$$\log_c a^b = b \log_c a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b \frac{1}{a} = \log_b a^{-1} = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

참고: 로그 함수

$$\Theta(\log_2 n) = \Theta(\log_{10} n) = \Theta(\log n) \quad ?$$

$$\log_2 n \neq \log_{10} n$$

$$\log_2 n = \frac{\log_{10} n}{\log_{10} 2} = \frac{1}{\log_{10} 2} \log_{10} n$$

$$\Theta(\log_2 n) \qquad\qquad \Theta(\log_{10} n)$$

반복 대치

$$T(n) \le 2T(n/2) + n$$
$$T(1) = 1$$

$$\begin{aligned}
T(n) &\le 2T(n/2) + n \\
&\le 2(2T(n/2^2) + n/2) + n = 2^2T(n/2^2) + 2n \\
&\le 2^2(2T(n/2^3) + n/2^2) + 2n = 2^3T(n/2^3) + 3n \\
&\cdots \\
&\le 2^kT(n/2^k) + kn = nT(1) + kn = n + n\log_2 n
\end{aligned}$$

➔ $T(n) = O(n \log_2 n)$

$n/2^k = 1$
$n = 2^k$
$\log_2 n = \log_2 2^k$
$\log_2 n = k$

점화식의 점근적 분석 방법

- 점화식으로 표현된 수행시간으로부터 점근적 복잡도를 구하는 방법
  - (1) 반복 대치(repeated substitution)
    - 더 작은 문제에 대한 함수로 반복해서 대치해 나가는 해법
  - (2) 추정 후 증명(guess and verification)
    - 결론을 추정하고 수학적 귀납법을 이용하여 증명하는 방법
  - (3) 마스터 정리(master theorem)
    - 점화식의 형식이 $\mathbf{T(n) = aT(n/b) + f(n)}$ 인 경우, 복잡도를 바로 구할 수 있는 theorem

(2) 추정 후 증명

$$T(n) = 2T(n/2) + n$$

<추정> $\mathbf{T(n) = O(n \log n)}$,

즉, 충분히 큰 n에 대해 $T(n) \le cn \log n$ 인 양의 상수 c가 존재

<증명>

$T(1) \le c\ 1 \log 1$ 은 불가능하므로

1) 경계 조건 : 2에 대해 성립, 즉, $T(2) \le c2 \log 2$ 인 c가 존재함

2) 귀납 가정 : $n/2$에 대해 성립, 즉 $T(n/2) \le c(n/2)\ \log (n/2)$ 이라고 가정

3) n에 대해 성립함을 증명. 즉, $T(n) \le cn \log n$ 인 c가 존재함을 증명

추정 후 증명

$T(n) = 2T(n/2) + n$

<추정> $\mathbf{T(n) = O(n \log n)}$,

즉, 충분히 큰 n에 대해 $T(n) \le cn \log n$ 인 양의 상수 c가 존재

<증명>

1) 경계 조건 : 2에 대해 성립, 즉, $T(2) \le c2 \log 2$ 인 c가 존재함

2) 귀납 가정 : $n/2$에 대해 성립, 즉 $T(n/2) \le c(n/2) \log (n/2)$ 이라고 가정

3) n에 대해 성립함을 다음과 같이 증명

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&\le 2\,(c(n/2)\log(n/2)) + n \\
&= cn \log(n/2) + n = cn\,(\log n - \log 2) + n = cn \log n - cn \log 2 + n \\
&= cn \log n + (-\,c \log 2 + 1)\, n \\
&\le cn \log n
\end{aligned}
$$

➔ 이 식 $T(n) \le cn \log n$ 을 만족하는 c가 존재 (c가 1/log 2 이상이면 됨)

➔ 따라서 $T(n) = O(n \log n)$

추정 후 증명

$$T(n) = 2T(n/2) + 1$$

<추정> **$T(n) = O(n)$,**

즉, 충분히 큰 n에 대해 $T(n) \le cn$ 인 양의 상수 c가 존재

<증명>

귀납 가정 : $T(n/2) \le cn/2$

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &\le 2cn/2 + 1 \\ &= cn + 1 \end{aligned}$$

➔ $cn + 1 \le cn$ 를 보일 수 없으므로 더 이상 진행 불가!

추정 후 증명

$$T(n) = 2T(n/2) + 1$$

<추정> $\mathbf{T(n) = O(n)}$,

단, 이번에는 앞에서와 달리 $T(n) \le cn - 2$ 인 c가 존재함을 증명하자.

<증명>

귀납 가정 : $T(n/2) \le cn/2 - 2$

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &\le 2(cn/2 - 2) + 1 \\ &= cn - 3 \\ &\le cn - 2 \end{aligned}$$

점화식의 점근적 분석 방법

- 점화식으로 표현된 수행시간으로부터 점근적 복잡도를 구하는 방법
  - (1) 반복 대치(repeated substitution)
    - 더 작은 문제에 대한 함수로 반복해서 대치해 나가는 해법
  - (2) 추정 후 증명(guess and verification)
    - 결론을 추정하고 수학적 귀납법을 이용하여 증명하는 방법
  - (3) 마스터 정리(master theorem)
    - 점화식의 형식이 $T(n) = aT(n/b) + f(n)$ 인 경우, 복잡도를 바로 구할 수 있는 theorem

(3) 마스터 정리

- $\mathbf{T(n) = aT(n/b) + f(n)}$ 형태의 점화식(단, $a \geq 1$, $b > 1$)은 마스터 정리에 의해 바로 점근적 복잡도를 알 수 있다.
- $n^{\log_b a} = h(n)$이라 하면 $T(n)$의 점근적 복잡도는 다음과 같다.

① 어떤 양의 상수 ε에 대하여 $f(n)/h(n) = O(1/n^{\varepsilon})$이면,
$T(n) = \Theta(h(n))$이다.
② 어떤 양의 상수 ε에 대하여 $f(n)/h(n) = \Omega(n^{\varepsilon})$이고,
어떤 상수 $c(< 1)$와 충분히 큰 모든 n에 대해
$af(n/b) \leq cf(n)$이면 $T(n) = \Theta(f(n))$이다.
③ $f(n)/h(n) = \Theta(1)$이면 $T(n) = \Theta(h(n) \log n)$이다.

마스터 정리

- $T(n) = aT(n/b) + f(n)$ 형태가 의미하는 바는 다음과 같다.

  크기 n인 문제를 푸는 시간 =

  크기 n/b인 문제 a개를 푸는 시간 + 나머지 오버헤드 $f(n)$

- 예) 병합정렬의 수행시간

  $T(n) = 2T(n/2) + n$

  ➔ $a = 2$

  $b = 2$

  $f(n) = n$

마스터 정리의 근사 버전

- $a \geq 1$, $b > 1$ 에 대해, 점화식 $T(n) = aT(n/b) + f(n)$ 에서 $n^{\log_b a} = h(n)$이라 하면 $T(n)$의 개략적인 점근적 복잡도는 다음과 같다.

① $\lim_{n \to \infty} \frac{f(n)}{h(n)} = 0$ 이면 $T(n) = \Theta(h(n))$이다.

② $\lim_{n \to \infty} \frac{f(n)}{h(n)} = \infty$ 이고, 충분히 큰 모든 n에 대해 $af(\frac{n}{b}) < f(n)$ 이면 $T(n) = \Theta(f(n))$이다.

③ $f(n)/h(n) = \Theta(1)$이면 $T(n) = \Theta(h(n) \log n)$이다.

마스터 정리의 직관적 의미

① h(n)이 더 무거우면 h(n)이 수행시간을 결정한다.

② f(n)이 더 무거우면 f(n)이 수행시간을 결정한다.

③ h(n)과 f(n)이 같은 무게이면 h(n)에 $\log n$을 곱한 것이 수행시간이 된다.

- h(n)
  - 반복적인 자기호출 끝에 마지막으로 크기 1인 문제를 만나는 횟수
- f(n)
  - 최상위 레벨(문제크기 n)에서 발생하는 자기호출 이외의 오버헤드
  - 크기가 다른 문제들 간의 관계를 반영하기 위한 비용
  - 가장 상위 레벨의 오버헤드만 의미하므로 하위 레벨에서의 오버헤드는 반영하고 있지 않음 ➔ 하위레벨의 오버헤드 합이 레벨이 내려가면서 적어도 증가하지는 않아야 함 $af(n/b) \leq cf(n)$

마스터 정리의 적용 예

- $T(n) = 2T(n/3) + c$

마스터 정리의 적용 예

- $T(n) = 2T(n/4) + n$

마스터 정리의 적용 예

- $T(n) = 2T(n/2) + n$

요약

- 재귀 알고리즘의 수행시간에 대해 점근적 복잡도를 구하는 방법
  1) 재귀 알고리즘의 수행시간을 점화식으로 표현
  2) 점화식을 점근적 표기로 나타냄
- 점화식으로부터 점근적 표기를 구하는 방법
  - 반복 대치는 점화식을 더 작은 변수에 대한 점화식으로 대치하는 작업을 반복하여 점근적 복잡도를 구한다.
  - 추정후 증명은 점화식의 점근적 복잡도를 추정한 후 이를 수학적 귀납법을 이용하여 증명한다.
  - 마스터 정리는 $T(n) = aT(n/b) + f(n)$ 형태로 된 점화식의 점근적 복잡도를 복잡한 계산이나 증명 없이 편리하게 구할 수 있다.

문제1

- 입력의 크기가 n일 때 다음 알고리즘의 점근적 수행시간은?

```
sample(A[], n) {
    if (n=1) return 1;
    sum ← 0;
    for i ← 1 to n
        sum ← sum + A[i];
    tmp ← sum + sample(A, n-1);
    return tmp;
}
```

(1) 수행시간 T(n)을 점화식으로 표현

(2) T(n)을 점근적 표기로 나타냄

문제2

- 입력의 크기가 n일 때 다음 알고리즘의 점근적 수행시간은?

```
sample(A[], n) {
    if (n=1) return 1;
    sum ← 0;
    for i ← 1 to n
        sum ← sum + A[i];
    tmp ← sum + sample(A, n/3);
    return tmp;
}
```

(1) 수행시간 T(n)을 점화식으로 표현

(2) T(n)을 점근적 표기로 나타냄

문제3

- 입력의 크기가 n일 때 다음 알고리즘의 점근적 수행시간은?

```
sample(A[], p, r) {
        if (p=r) return 1;
        sum ← 0;
        for i ← 1 to n
                sum ← sum + A[i];
        q ← (p+r)/2  ;   // 정수나눗셈
        tmp ← sum + sample(A, p, q) + sample(A, q+1, r);
        return tmp;
}
```

(1) 수행시간 T(n)을 점화식으로 표현

(2) T(n)을 점근적 표기로 나타냄

문제4

- 입력의 크기가 n일 때 다음 알고리즘의 점근적 수행시간은?

```
sample(A[], p, r) {
    if (p=r) return 1;
    q ← (p+r)/2  ;   // 정수나눗셈
    tmp ← sample(A, p, q) + sample(A, q+1, r);
    return tmp;
}
```

(1) 수행시간 T(n)을 점화식으로 표현

(2) T(n)을 점근적 표기로 나타냄