분할 정복

Data Structures and Algorithms

목차

- 아우스터리츠 전투
- 분할 정복
- 응용

아우스터리츠 전투

아우스터리츠 전투: 배경

- 오스트리아-러시아 동맹군의 규모 : 8만명
- 프랑스 군의 규모 : 6만 5천명
- 프랑스 군이 오스트리아-러시아 동맹 군을 유인하여 반으로 분할, 가운데에서 반으로 나뉜 동맹군을 공격하여 대파한 전투.
- 프랑스의 승리. 동맹군 15,000명을 사살하고 11,000명의 포로를 잡았습니다. 프랑스는 9,000명의 사상자 발생.

Battle of Austerlitz 설명 보기: https://youtu.be/XmVoizt4orc

분할 정복

분할 정복 알고리즘 소개

- 문제를 더 이상 나눌 수 없을 때까지 나누고, 이렇게 나누어진 문제들을 각각 풂으로써 결국 전체 문제의 답을 얻는 알고리즘

- 문제를 쪼개는 요령이나 규칙은 없음
- 개발자의 창의에 달려 있음

접근 방법

1. 분할(Divide) :
   - 문제가 분할이 가능하면 2개 이상의 하위 문제로 나눔

2. 정복(Conquer) :
   - 하위 문제가 분할 가능한 상태면 1번 재수행
   - 불가능하면 하위 문제 풀기

3. 결합(combine) :
   - 2 과정에서 정복된 답을 취함

- 복잡도는 ?

응용

응용: 병합정렬

1. (분할) 정렬할 데이터 집합을 반으로 나눔
   - 나누어진 하위 집합의 크기가 2 이상이면 1을 반복

2. (정복) 같은 집합에서 나뉜 하위 데이터 집합 둘을 병합
   - 단, 병합을 할 때 원소 순서에 맞춰 정렬

3. (결합) 데이터 집합이 다시 하나가 될 때까지 2을 반복

병합정렬

- 주어진 데이터 집합

5	1	6	4	8	3	7	9	2

병합 정렬으로 정렬해보자

응용: 거듭 제곱(Exponentiation)

- 복잡도 : 단순 구현 시 ???

$$C^2 = C \times C$$

$$C^3 = C \times C \times C$$

$$C^n = C \times C \times C \times \dots \times C$$

```
int Power( int Base, int Exponent )
{
    int i=0;
    int Result = 1; /*  C^0은 1이므로. */

    for ( i=0; i<Exponent; i++ )
        Result *= Base;

    return Result;
}
```

거듭 제곱(Exponentiation)

$$C^8 = C^8$$

짝수

$$C^8 = C^4 \times C^4$$

$$C^8 = C^2 \times C^2 \times C^2 \times C^2$$

$$C^8 = C \times C \times C \times C \times C \times C \times C \times C \times C$$

$$C^7 = C^7$$

홀수

$$C^7 = C^3 \times C^3 \times C$$

$$C^7 = (C \times C \times C) \times (C \times C \times C) \times C$$

일반화

???

거듭 제곱(Exponentiation)

- 복잡도 : 분할 정복 기반 ???

```
long int Power( int Base, int Exponent )
{
    if ( Exponent == 1 )
        return Base;
    else if ( Base == 0 )
        return 1;
    if ( Exponent % 2 == 0 )
    {
        long int                         ???
        return NewBase * NewBase;
    }
    else
    {
        long int                          ???
        return (NewBase * NewBase) * Base;
    }
}
```

응용: 피보나치

- 피보나치 수
  - 이탈리아의 수학자 레오나르도 피보나치(Leonardo Fibonacci. 1170~1250)의 저서 "계산의 책(Liber Abbaci)"에 소개된 문제

단순한 구현

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n > 1 \end{cases}$$

```
int Fibonacci( int n )
{
    if ( n== 0 )
        return 0;
    if ( n == 1 || n == 2 )
        return 1;
    return Fibonacci( n - 1) +
Fibonacci( n - 2 );
}
```

복잡도: $O(2^n)$

응용: 피보나치

- 분할 정복 알고리즘 방법
  - n번째 피보나치 수를 구할 때 n/2번째 피보나치 수를 찾아 제곱하면 되고, n/2번째 피보나치 수를 구하려면 (n/2)/2번째 피보나치 수를 제곱
    - ➔ n번째 피보나치 수를 구할 때 log2n회만 제곱하면 된다.

$$A^n A^m = A^{n+m}$$

$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n}$$

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n}$$

복잡도: $O(\log_2 n)$