



알고리즘 설계 – 1장

분할정복법 (Divide-and-Conquer)

References:

1. R. Neapolitan and K. Naimipour, Foundations of Algorithms using Java Pseudocode, Jones and Bartlett Pub.



알고리즘 설계의 전체 목차

- 분할정복법
- 동적 프로그래밍
- 탐욕적인 방법
- 계산복잡도



분할정복법의 목차

- 분할정복법의 소개
- 이진 검색
- 빠른 정렬
- 행렬곱셈 알고리즘
- 언제 분할정복법을 사용하는가?



1. 분할정복법의 소개

- 개념

- 문제의 범위를 2개 이상의 더 작은 범위로 나눈다.
- 작은 범위에 대해 해를 바로 얻을 수 있으면 OK.
- 작은 범위가 여전히 크다면 더욱 범위를 나눈다.

- 특징

- 하향식 접근방법
- recursion 사용



설계 전략

- 분할(Divide): 해결하기 쉽도록 문제를 여러 개의 작은 부분으로 나눈다.
- 정복(Conquer): 나눈 작은 문제를 각각 해결한다.
- 통합(Combine): 해결된 해답을 모은다.

2. 이진 검색(Binary Search)

- 문제: 크기가 n 인 정렬된 배열 S 에 x 가 있는지를 결정하라.
- 입력: 자연수 n , 오름차순으로 정렬된 배열 $S[0..n-1]$, 검색 항목 x
- 출력: x 가 S 의 어디에 있는지의 위치. x 가 S 에 없다면 -1
- 설계전략:
 - x 가 배열의 중간에 위치하고 있는 항목과 같으면, "빙고", 찾았다! 그렇지 않으면:
 - 분할: 배열을 반으로 나누어서 x 가 중앙에 위치한 항목보다 작으면 왼쪽에 위치한 배열 반쪽을 선택하고, 그렇지 않으면 오른쪽에 위치한 배열 반쪽을 선택한다.
 - 정복: 선택된 반쪽 배열에서 x 를 찾는다.
 - 통합: (필요 없음)

알고리즘

```
index location (index low, index high) {
    index mid;
    if (low > high)        return -1;           // 찾지 못했음
    else {
        mid = (low + high) / 2                 // 정수 나눗셈 (나머지 버림)
        if (x == S[mid])
            return mid;                       // 찾았음
        else if (x < S[mid])
            return location(low, mid-1);       // 왼쪽 반을 선택
        else
            return location(mid+1, high);      // 오른쪽 반을 선택
    }
}
...
locationout = location(0, n-1);
...
```



관찰

- 입력 파라미터 S , x 는 알고리즘 수행 중 불변
- 함수를 재귀호출(recursive call)할 때 마다 변하지 않는 파라미터를 가지고 다니는 것은 낭비



복잡도 (n 이 2^k 일 경우)

- 시간복잡도(단위 연산: x 와 $S[mid]$ 의 비교)
 - $W(n) = W(n/2) + 1$
 - $W(1) = 1$
- 해
 - $W(1) = 1$
 - $W(2) = W(1) + 1 = 2$
 - $W(4) = W(2) + 1 = 3$
 - $W(8) = W(4) + 1 = 4$
 - ...
 - $W(2^k) = k + 1$
 - $W(n) = \log_2 n + 1$



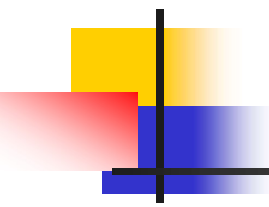
귀납법에 의한 증명

- 귀납출발점
 - $n = 1$ 이면, $W(1) = 1 = \log_2 1 + 1$.
- 가정
 - 2의 거듭제곱인 양의 정수 n 에 대해서, $W(n) = \log_2 n + 1$ 이라고 가정
- 귀납단계
 - $W(2n) = \log_2(2n) + 1$ 임을 보이면 된다.
 - $W(2n) = W(n) + 1 = \log_2 n + 1 + 1$
 $= \log_2 n + \log_2 2 + 1$
 $= \log_2(2n) + 1$

n이 2의 거듭제곱이 아닐 경우

n	왼쪽 부분배열의 크기	mid	오른쪽 부분배열의 크기
짝수	$n/2 - 1$	1	$n/2$
홀수	$(n-1)/2$	1	$(n-1)/2$

- 다음 단계에서 찾아야 할 항목의 개수는 $\lfloor n/2 \rfloor$.
- 시간 복잡도
 - $W(n) = W(\lfloor n/2 \rfloor) + 1$
 - $W(1) = 1$
- 해: $W(n) = \lfloor \log_2 n \rfloor + 1 \leftarrow$ 귀납법으로 풀어볼 것.



2. 빠른 검색(Quick Sort)

- 문제: 15 22 13 27 12 10 20 25
- 분할 단계
 - 기준(pivot) 값보다 작은 값은 왼쪽으로, 큰 값은 오른쪽에 배치
 - 10 13 12 15 22 27 20 25
- 정복 단계
 - 부분 배열을 정렬
 - 10 13 12 15 20 22 25 27

알고리즘(1)

- 문제: n 개의 정수를 오름차순으로 정렬
- 입력: 정수 $n > 0$, 크기가 n 인 배열 $S[0..n-1]$
- 출력: 오름차순으로 정렬된 배열 $S[0..n-1]$
- 알고리즘:

```
void quicksort (index low, index high) {  
    index pivotpoint;  
    if (high > low) {  
        pivotpoint = partition(low, high);  
        quicksort(low, pivotpoint-1);  
        quicksort(pivotpoint+1, high);  
    }  
}
```

분할 알고리즘

- 문제: 빠른 정렬을 하기 위해서 배열 S 를 둘로 나눈다.
- 입력: (1) 첨자 low , $high$, (2) 첨자 low 에서 $high$ 까지의 S 의 부분배열
- 출력: 첨자 low 에서 $high$ 까지의 S 의 부분배열의 기준점 $pivotpoint$
- 알고리즘:

```
index partition (index low, index high) {  
    index i, j, pivotpoint;  
    keytype pivotitem;  
    pivotitem = S[low];           // pivotitem을 위한 첫번째 항목을 고른다  
    j = low;  
    for (i = low + 1; i <= high; i++)  
        if (S[i] < pivotitem) { j++; exchange S[i] and S[j]; }  
    pivotpoint = j;  
  
    exchange S[low] and S[pivotpoint]; // pivotitem 값을 pivotpoint에 넣는다  
    return pivotpoint;  
}
```

분할 알고리즘의 작동 예

i	j	S[0]	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]
–	–	15	22	13	27	12	10	20	25
1	0	15	22	13	27	12	10	20	25
2	1	15	22	13	27	12	10	20	25
3	1	15	13	22	27	12	10	20	25
4	2	15	13	22	27	12	10	20	25
5	3	15	13	12	27	22	10	20	25
6	3	15	13	12	10	22	27	20	25
7	3	15	13	12	10	22	27	20	25
–	3	10	13	12	15	22	27	20	25



분할 알고리즘의 분석

- 단위연산
 - $S[i]$ 와 key 와의 비교
- 입력크기
 - 부분배열이 가지고 있는 항목의 수
 - $n = high - low + 1$
- 분석
 - 첫번째 항목만 제외하고 모든 항목을 한번씩 비교
 - $T(n) = n - 1$

빠른 정렬 알고리즘의 분석 - 최악(1)

- 최악의 경우
 - 이미 오름차순으로 정렬이 되어 있는 배열을 정렬하려는 경우
 - 크기가 n 인 배열의 경우, 크기가 0 인 부분배열은 왼쪽에 오고, 크기가 $n-1$ 인 부분배열은 오른쪽에 오도록 하여 계속 쪼개진다.
 - 복잡도: $W(n) = W(0) + W(n-1) + n - 1$
 - $W(0)$: 왼쪽 배열을 정렬하는 시간 = 0
 - $W(n-1)$: 오른쪽 배열을 정렬하는 시간
 - $n - 1$: 분할 시간
 - 수정된 복잡도
 - $W(n) = W(n-1) + n - 1, n > 0$
 - $W(0) = 0$



빠른 정렬 알고리즘의 분석 - 최악(2)

$$W(n) = W(n-1) + n-1$$

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

$$W(n-2) = W(n-3) + n-3$$

...

$$W(2) = W(1) + 1$$

$$W(1) = W(0) + 0$$

$$W(0) = 0$$

$$W(n) = 1 + 2 + \dots + (n-1) = n(n-1)/2 \in \theta(n^2)$$

빠른 정렬 알고리즘의 분석 - 평균(1)

$$\begin{aligned} A(n) &= \sum_{p=1}^n \frac{1}{n} [A(p-1) + A(n-p)] + n - 1 \\ &= \frac{2}{n} \sum_{p=1}^n A(p-1) + n - 1 \end{aligned}$$

A(n)과 A(n-1)에 대해 n과 n-1을 각각 곱한 후, 빼기

$$nA(n) = 2 \sum_{p=1}^n A(p-1) + n(n-1)$$

$$(n-1)A(n-1) = 2 \sum_{p=1}^{n-1} A(p-1) + (n-1)(n-2)$$

$$nA(n) - (n-1)A(n-1) = 2A(n-1) + 2(n-1),$$

$$\frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

$$\begin{aligned} a_n &= \frac{A(n)}{n+1}, \text{ then} \\ a_n &= a_{n-1} + \frac{2(n-1)}{n(n+1)} \\ a_0 &= 0 \end{aligned}$$



빠른 정렬 알고리즘의 분석 - 평균(2)

$$a_n = a_{n-1} + \frac{2(n-1)}{n(n+1)}$$

$$a_0 = 0, \text{ then}$$

$$a_n = \sum_{i=1}^n \frac{2(i-1)}{i(i+1)} = 2 \left(\sum_{i=1}^n \frac{i}{i(i+1)} - \sum_{i=1}^n \frac{1}{i(i+1)} \right)$$

오른쪽 항은 무시할 수 있고, $(\frac{1}{2} + \dots + \frac{1}{n} \approx \ln n)$,
이므로 $a_n = 2 \ln n$ (Appendix A의 A.9 참조). 따라서,

$$\begin{aligned} A(n) &\approx (n+1)2 \ln n = (n+1)2(\ln 2)(\lg n) \\ &\approx 1.38(n+1) \lg n \in \Theta(n \lg n) \end{aligned}$$

3. 행렬 곱셈

- 단순한 행렬곱셈 알고리즘
 - 문제: $n \times n$ 크기의 행렬의 곱을 구하시오.
 - 입력: 양수 n , $n \times n$ 크기의 행렬 A 와 B
 - 출력: 행렬 A 와 B 의 곱인 C
 - 알고리즘:

```
void matrixmult (int n, const int A[], const int B[], int C[]) {  
    index i, j, k;  
    for (i = 0; i < n; i++)  
        for (j = 0; j < n; j++) {  
            C[i][j] = 0;  
            for (k = 0; k < n; k++)  
                C[i][j] = C[i][j] + A[i][k] * B[k][j];  
        }  
}
```

시간 복잡도 = $\Theta(n^3)$

Strassen의 방법

- 문제: n 이 2의 거듭제곱이고, 각 행렬을 4개의 부분행렬로 나눈다고 가정하자. 두 $n \times n$ 행렬 A 와 B 의 곱 C :

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

행렬곱: 8번
행렬합: 4번

- Strassen의 기법

$$C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

- $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
- $M_2 = (A_{21} + A_{22})B_{11}$
- $M_3 = A_{11}(B_{12} - B_{22})$
- $M_4 = A_{22}(B_{21} - B_{11})$
- $M_5 = (A_{11} + A_{12})B_{22}$
- $M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$
- $M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

행렬곱: 7번
행렬합: 18번

Strassen 알고리즘

- 문제: n 이 2의 거듭제곱일 때, $n \times n$ 크기의 두 행렬의 곱을 구하시오.
- 입력: 정수 n , $n \times n$ 크기의 행렬 A 와 B
- 출력: 행렬 A 와 B 의 곱인 C
- 알고리즘:

```
void strassen (int n, n*n_matrix A, n*n_matrix B, n*n_matrix& C) {  
    if (n <= 임계점)  
        단순한 알고리즘을 사용하여  $C = A * B$ 를 계산;  
    else {  
        A를 4개의 부분행렬  $A_{11}, A_{12}, A_{21}, A_{22}$ 로 분할;  
        B를 4개의 부분행렬  $B_{11}, B_{12}, B_{21}, B_{22}$ 로 분할;  
        스트라센의 방법을 사용하여  $C = A * B$ 를 계산;  
        // 되부르는 호출의 예: strassen(n/2, A11+A12, B11+B22, M1)  
    } }
```

- 용어: 임계점(threshold)이란? 단순한 알고리즘보다 Strassen의 알고리즘을 사용하는 편이 더 좋을 것이라고 예상되는 지점.

분석(1)

- 단위연산: 곱셈하는 연산
- 입력크기: 행과 열의 수, n
- 모든 경우 시간복잡도 분석: 임계값을 1이라고 하자.
- 복잡도
 - $T(n) = 7T(n/2)$ for $n > 1$, n 은 2의 배수
 - $T(1) = 1$
- 해
 - $T(n) = 7^{\log n} = n^{\log 7} \approx n^{2.81} \in \Theta(n^{2.81})$

분석(2)

- 단위연산: 덧셈/뺄셈하는 연산
- 입력크기: 행과 열의 수, n
- 모든 경우 시간복잡도 분석: 위에서와 마찬가지로 임계값을 1이라고 하자.
- 복잡도
 - $T(n) = 7T(n/2) + 18(n/2)^2$
 - $T(1) = 0$
 - $T(n) = 6n^{\log 7} - 6n^2 \approx 6n^{2.81} - 6n^2 \in \Theta(n^{2.81})$

실험

- 실험 환경
 - HW: i7-4790 (Quad Core 8 threads, 3.6GHz)
 - SW: Java 1.8
 - 행렬: $C[n][n] = A[n][n] * B[n][n]$
 - Threshold of Strassen: $n = 512$

- 실험 결과

n	Classic	Strassen
256	0.160초	0.310초
512	0.265초	0.284초
1024	7.457초	1.373초
2048	103.694초	9.641초
4096	1005.106초	70.641초
8192	9333.979초	479.506초

5. 분할정복법을 사용할 수 없는 경우

- 크기가 n 인 입력이 2개 이상의 조각으로 분할되며, 분할된 부분들의 크기가 거의 n 에 가깝게 되는 경우
 - 예: 분할정복법으로 n 번째 피보나치 항 구하기
 - $T_n = T_{n-1} + T_{n-2} + 1 = 2^{n/2}$

⇒ 시간복잡도: 지수(exponential) 시간
- 크기가 n 인 입력이 거의 n 개의 조각으로 분할되며, 분할된 부분의 크기가 n/c 인 경우. 여기서 c 는 상수이다.

⇒ 시간복잡도: $\Theta(n^{\log n})$