

アルゴリズムとデータ構造 プログラミング演習

森 立平 mori@c.titech.ac.jp

2016 年 6 月 21 日

今日の目標

- 分割統治法のプログラムを書けるようになる.

今日の主な課題 (提出締切は今週金曜日 23 時 59 分 59 秒)

1. クイックソートのプログラムを完成させる.
2. クイックセレクト, クイックソートの計算量をもとめる.

今日のワークフロー

1. この資料をよく読み分割統治法の考え方, クイックセレクト, クイックソートについて理解する.
2. 課題に取り組む.
3. 教員か TA に課題提出.

1 はじめに

ウェブブラウザで <https://github.com/ryuhei-mori/2016ad> にアクセスしてください.

2 分割統治法とは

分割統治法 (divide-and-conquer) は効率的なアルゴリズムの枠組みで最も代表的なものの一つである. 分割統治法は以下の 3 つのステップからなる再帰的アルゴリズムの枠組みである.

1. 分割: 問題をいくつかの同じ問題のより小さい部分問題へ分割する.
2. 統治: 部分問題を再帰的に解く. ただし, 部分問題のサイズが十分小さい場合は直接的な方法で解く.
3. 結合: 部分問題の解から元の問題の解を計算する.

分割統治法の計算量を見積ろう. 元の問題の (何かしらの方法で定義された) サイズを n とし, サイズ n の問題を解く (何かしらの方法で定義された) 計算量を $T(n)$ とおく. 分割統治法では「統治」のステップで同じサイズの部分問題を解くことが多い. 仮に「統治」のステップで n/a のサイズの部分問題を b 個解く (a と b は整数) とし, 「分割」および「結合」の計算量をそれぞれ $D(n)$, $C(n)$ とすると $T(n)$ は

$$T(n) = bT(n/a) + D(n) + C(n)$$

という漸化式を満たす. この漸化式を解くことで $T(n)$ のオーダーをもとめることができる. この漸化式の解き方はここでは説明しない (厳密な証明は少し長くなってしまふ). また, 同じサイズの部分問題に分割しないアルゴリズムの場合でも似たような漸化式を立てることができる.

3 分割統治法の例

3.1 クイックセレクト

最も単純な分割統治法の例としてクイックセレクトがある。次の問題を考えよう。

入力: 整数の配列 A と非負の整数 k 。

問題: 整数の配列 A の中で $k+1$ 番目に大きい要素を出力せよ。

クイックセレクトはこの問題を解く簡単なアルゴリズムである。クイックセレクトの擬似コードをアルゴリズム 1 に示す。クイックセレクトは分割統治法としては珍しいことに一つの部分問題しか解かない。また、「結合」のステップは存在しない。

アルゴリズム 1 クイックセレクトの擬似コード (入力: 整数の配列 A , 非負の整数 k . 出力: 配列 A の $k+1$ 番目に大きい要素.)

```
if 配列  $A$  の長さが 1 then
    配列の唯一の要素を解として出力する.
else
    配列  $A$  から一つ要素を選択する。以下その要素をピボットと呼ぶ。
    配列  $A$  のピボット以外の要素を「ピボット以下のもの」からなる配列  $A_1$  と「ピボットより大きいもの」からなる配列  $A_2$  の 2 つの配列に分割する。
    配列  $A_2$  の要素の数を  $r$  とおく。
    if  $r == k$  then
        ピボットを解として出力する。
    else if  $r < k$  then
        配列  $A_1$  の中から  $k - r$  番目に大きい要素を解として出力する (再帰呼出)。
    else
        配列  $A_2$  の中から  $k + 1$  番目に大きい要素を解として出力する (再帰呼出)。
    end if
end if
```

3.2 クイックソート

クイックソートは配列を整列させるアルゴリズムである。クイックソートの擬似コードをアルゴリズム 2 に示す。クイックソートはクイックセレクトととても良く似ており、「分割」のステップはまったく同じである。

アルゴリズム 2 クイックソートの擬似コード (入力: 整数の配列 A . 出力: 配列 A を小さい順に並べた配列.)

```
if 配列  $A$  の長さが 1 以下 then
    配列  $A$  を解として出力する。
else
    配列  $A$  から一つ要素を選択する。以下その要素をピボットと呼ぶ。
    配列  $A$  のピボット以外の要素を「ピボット以下のもの」からなる配列  $A_1$  と「ピボットより大きいもの」からなる配列  $A_2$  の 2 つの配列に分割する。
    配列  $A_1$  をソートして  $A'_1$  とする (再帰呼出)。
    配列  $A_2$  をソートして  $A'_2$  とする (再帰呼出)。
     $A'_1$ , ピボット,  $A'_2$  の順番に並べた配列を解として出力する。
end if
```

4 分割統治法についてまとめと動的計画法

ここまでの分割統治法についての説明をまとめる。

- 分割統治法は「分割」、「統治」、「結合」の3つのステップからなる再帰的アルゴリズムの枠組みである。
- 分割統治法の計算量は漸化式を解くことで得られる。
- クイックセレクト、クイックソート、Strassen のアルゴリズムは分割統治法の代表的なアルゴリズムである。

分割統治法はアルゴリズムの枠組みとしてとても重要なものであるが、必ずしも全ての問題に適用できるわけではない。分割統治法が適用できない（適用したとしても効率が良くない）場合として

- 部分問題への分割はできるのだがサイズ n/a の部分問題へ分割できない

という状況がある。しかし、一方で

- 異なる部分問題の間で計算が重複している

という性質が満たされている場合には、計算の重複を避けることで効率的なアルゴリズムが設計できる。これが動的計画法の考え方である。分割統治法と動的計画法を理解すれば世の中のアルゴリズムの半分くらいが理解できるといっても過言ではない。

5 今日の課題

計算量

1. クイックセレクトの C 言語のプログラムをコンパイルして動かせ。またプログラムを1行ずつ読んで内容を理解せよ。
2. クイックセレクトのプログラムを参考にしてクイックソートの C 言語のプログラムを作成せよ。「分割」のステップはクイックセレクトと同じであるが、部分問題を解いた後の再配置を防ぐため、あらかじめピボットを適切な場所に配置してから再帰呼び出しせよ。
3. クイックセレクトとクイックソートの計算量のオーダーをもとめよ。ただし「分割」のステップで必ず配列を半分づつに分割すると仮定せよ。
4. クイックセレクトとクイックソートの計算量のオーダーをもとめよ。ただし「分割」のステップで必ず配列を長さ1と長さ $n-1$ に分割すると仮定せよ。
5. 発展的課題 (やらなくてもよいが): 改良版クイックセレクトの最悪計算量をもとめよ。

実際の C 言語のプログラムは以下ようになる。

Listing 1: クイックセレクトの C 言語プログラム

```
#include <stdio.h>

//B[start], B[start+1], ..., B[end-1]の中で k+1 番目に大きい値を返す関数
int quickselect(int B[], int start, int end, int k){
    int m, i, tmp;
    int pivot;

    //サイズが1の場合はその唯一の要素を返す
```

```

    if(end == start+1) return B[start];

/*
B[start]をピボットとし，Bの前半にピボット以下の要素，
Bの後半にピボットより大きい要素を配置する
*/
    pivot = B[start];
    m = start+1;
    for(i = start+1; i < end; i++){
        if(B[i] <= pivot){
            tmp = B[m];
            B[m] = B[i];
            B[i] = tmp;
            m++;
        }
    }

// ピボットより大きい要素が k 個あった場合
    if(end == m + k) return pivot;
// ピボットより大きい要素が k 個より少ない場合
    else if(end < m + k) return quickselect(B, start+1, m, k - (end - m) - 1);
// ピボットより大きい要素が k 個より多い場合
    else return quickselect(B, m, end, k);
}

int main(){
    int i;
    int A[] = { 6, 1, 20, 3, 8, 9, 2, 7, 10, 6 };

    for(i = 0; i < 10; i++){
        printf("%d th: %d\n", i, quickselect(A, 0, 10, i));
    }

    return 0;
}

```

余談: クイックセレクトの改良

クイックセレクトおよびクイックソートにおいては「分割」で半分の長さの配列に分割できる場合が一番計算量が小さくなる。また、厳密に半分でなくても $0.4n$ と $0.6n$ の分割や $0.0001n$ と $0.9999n$ の分割でもやはり計算量のオーダーは最適となる。逆に 1 と $n-1$ や \sqrt{n} と $n-\sqrt{n}$ のように極端に偏った分割をしてしまうとクイックセレクトおよびクイックソートは非効率となる。もし入力となる配列の要素がランダムな順番に並んでいるとすると、このように偏った分割をしてしまうことはほとんどないため、クイックセレクト、クイックソートの計算量は上の課題の 3 でもとめたものになる。これを平均計算量と呼ぶ。一方でアルゴリズムの計算量が最も大きくなってしまような入力に対する計算量を最悪計算量と呼ぶ。単に計算量を言うときは最悪計算量のことを指している。以下ではクイックセレクトのピボットの選択を改良することで最悪計算量を平均計算量と等しくできを示す。これは若干発展的な内容なので理解しなくてもよい。

クイックセレクトにおいてピボットを選択するときにはできるだけ中央値に近いものを選びたい。そこで一旦サイズ n の配列をサイズ 5 ずつに分割して、それぞれ 5 つの要素の中で中央値をとる。その中央値を集めて長さ $\lceil n/5 \rceil$ の配列を作る。そしてその長さ $\lceil n/5 \rceil$ の配列の中央値をクイックセレクトを再帰呼び出しすることで計算し、それをピボットとする。擬似コードをアルゴリズム 3 に示す。

アルゴリズム 3 改良版クイックセレクトの擬似コード (入力: 整数の配列 A , 非負の整数 k . 出力: 配列 A の $k+1$ 番目に大きい要素.)

```
if 配列  $A$  の長さが 1 then
    配列の唯一の要素を解として出力する.
else
    配列  $A$  を長さ 5 ずつに分割してそれぞれ中央値を計算し, 長さ  $\lceil n/5 \rceil$  の配列  $A'$  を作る.
    配列  $A'$  の中央値をピボットとして選択 (再帰呼出).
    配列  $A$  のピボット以外の要素を「ピボット以下のもの」からなる配列  $A_1$  と「ピボットより大きいもの」からなる配列  $A_2$  の 2 つの配列に分割する.
    配列  $A_2$  の要素の数を  $r$  とおく.
    if  $r == k$  then
        ピボットを解として出力する.
    else if  $r < k$  then
        配列  $A_1$  の中から  $k - r$  番目に大きい要素を解として出力する (再帰呼出).
    else
        配列  $A_2$  の中から  $k + 1$  番目に大きい要素を解として出力する (再帰呼出).
    end if
end if
```
