アルゴリズムとデータ構造

分割統治法とソート

森 立平
mori@c.titech.ac.jp

2019 年 6 月 25 日

今日のメッセージ

- **分割統治法の時間計算量のは漸化式を解けば得られる**
- **クイックソートはピボットを適切に選べば $O(n \log n)$ 時間で計算できる**
- **中央値の中央値アルゴリズムを使えば $O(n)$ 時間で k 番目に小さい値が計算できる**

今日の目標

- クイックソート、クイックセレクト、中央値の中央値アルゴリズムを理解する

今日の演習の目標

- クイックソート、クイックセレクトなどのプログラムを書けるようになる

今日の主な演習課題 (提出締切は来週火曜日正午 (12:00))

1. クイックソートとクイックセレクトのプログラムを書く

今日の演習時間のワークフロー

1. この資料をよく読み、`alg2019/sort` にある課題をやる

1 クイックソート

クイックソートは次の漸化式に基づく。

$$\mathrm{Qsort}(A) = \begin{cases} [\,], & \text{if } |A| = 0 \\ \mathrm{Qsort}(B) \circ [a_p] \circ \mathrm{Qsort}(C), & \text{where } (B, C) := \mathrm{partition}(A \setminus a_p, a_p), \end{cases}$$

ここで、a_p は配列 A の要素の一つであり、$\mathrm{partition}(A, a)$ は配列 A を a 以下の値からなる配列 B と a より大きい値からなる配列 C へ分割する関数である。a_p の選択の仕方によって時間計算量は大きく変化する。

2 クイックソートの時間計算量

他のソートアルゴリズムと同様に、クイックソートの時間計算量を「比較の回数と代入の回数の和」と定義する。クイックソートの時間の時間計算量はピボットの選び方に大きく依存する。

もし、ピボットが常に最小値であった場合、時間計算量 $T(n)$ は

$$T(n) \leq T(n-1) + cn$$

を満たす (c は定数)。よって、$T(n) = O(n^2)$ となる。もし、ピボットが常に中央値であった場合、時間計算量 $T(n)$ は

$$T(n) \leq 2T(n/2) + cn$$

を満たすので、$T(n) = O(n \log n)$ となる。もしも中央値を $O(n)$ 時間で選択することができれば、クイックソートの時間計算量は $O(n \log n)$ となる。もし、ピボットが常に αn 番目に小さい場合、

$$T(n) \leq T(\alpha n) + T((1-\alpha)n) + cn$$

を満たす。この場合も $T(n) = O(n \log n)$ であることを示す。ある定数 $d > 0$ について帰納法で $T(n) \leq dn \log n$ を示す。$T(n) \leq dn \log n$ が $n < \lceil 1/\alpha \rceil$ に対して成り立つように $d > 0$ をとることができる。このとき、$T(m) \leq dm \log m$ が $\lceil 1/\alpha \rceil \leq m < n$ について成り立っているとすると,

$$\begin{aligned} T(n) &\leq T(\alpha n) + T((1-\alpha)n) + cn \\ &\leq d\alpha n \log(\alpha n) + d(1-\alpha)n \log((1-\alpha)n) + cn \\ &= dn \log n - dn \left(-\alpha \log \alpha - (1-\alpha) \log(1-\alpha)\right) + cn \end{aligned}$$

ここでバイナリエントロピー関数 $h(\alpha) := -\alpha \log \alpha - (1-\alpha) \log(1-\alpha)$ は $\alpha \in (0,1)$ について正である。よって d を十分大きく取り直せば、$dh(\alpha) \geq c$ となる。よって、$T(n) \leq dn \log n$ が得られる。よって、任意の自然数 n について $T(n) \leq dn \log n$ である。

よって αn 番目に小さい値を $O(n)$ 時間で選択することができれば、クイックソートの時間計算量は $O(n \log n)$ となる。ランダムにピボットを選ぶのも有効で平均時間計算量が $O(n \log n)$ となることが示せる。

3 クイックセレクト

配列の中で $k+1$ 番目に小さい要素を計算する問題を「選択問題」と呼ぶ。選択問題はソートを用いれば $O(n \log n)$ 時間で解くことができるが、もっと効率のよい $O(n)$ 時間のアルゴリズムが存在する。

$$\text{Qselect}(A,k) = \begin{cases} (B,C) := \text{partition}(A, a_p) \text{と置く} & \\ a_p, & \text{if } |B| = k+1 \\ \text{Qselect}(C, k-|B|), & \text{if } |B| \leq k \\ \text{Qselect}(B \setminus a_p, k), & \text{otherwise.} \end{cases}$$

```
// *p と *q の値を入れ替える
void swap(int *p, int *q){
  int tmp;
  tmp = *p;
  *p = *q;
  *q = tmp;
}

/*
A[0], A[1], ..., A[n-1] の中でk+1番目に小さい値を返す関数。
n >= 1 && k >= 0 && k < n は保証されている。
ただし、Aの中身は並び換えてしまう。
```

```
*/
int quick_select(int A[], int n, int k){
  int i, j, pivot;

// 先頭の要素をピボットとする
  pivot = A[0];
  for(i = j = 1; i < n; i++){
    if(A[i] <= pivot){
      swap(A + i, A + j);
      j++;
    }
  }
  if(j == k + 1) return pivot;
  else if(j < k + 1) return quick_select(A + j, n - j, k - j);
  else return quick_select(A + 1, j - 1, k);
}
```

もしも $O(n)$ 時間で中央値が計算できたとすると、クイックセレクトの時間計算量 $T(n)$ は n を 2 の冪とすると

$$T(n) = T(n/2) + cn = c\left(n + \frac{n}{2} + \frac{n}{4} + \cdots + 2\right) + T(1) \leq 2cn + T(1) = O(n)$$

である。クイックソートと同様に大体中央にあるようなピボットを選んだ場合も $O(n)$ 時間であることはすぐに分かる。

4 クイックソート、クイックセレクトの実装方法

前章のクイックセレクトの実装ではピボットから配列を 2 つに分割するプログラムは以下のようになっていた。

```
pivot = 「適切に選ぶ」;
for(i = j = 1; i < n; i++){
  if(A[i] <= pivot){
    swap(A + i, A + j);
    j++;
  }
}
/*
 A[1], ..., A[j-1] は A[i] 以下
 A[j], ..., A[n] は A[i] より大きい
*/
```

ピボットとして中央値が選べれば半分のサイズの配列に分割することができ、アルゴリズムの時間計算量は小さくなる。ピボットをランダムに選択すれば高い確率で中央値に近い要素が選択できる。しかし、もし全ての要素が等しかった場合、どのようなピボットの選び方をしても常に偏った分割をしてしまう。よって従来のクイックソートを少し改良して次の漸化式を考えよう。

$$\mathrm{Qsort}(A) = \begin{cases} [\,], & \text{if } |A| = 0 \\ \mathrm{Qsort}(B) \circ C \circ \mathrm{Qsort}(D), & \text{where } (B, C, D) := \mathrm{partition}_3(A \setminus a_p, a_p), \end{cases}$$

ここで、a_p は配列 A の要素の一つであり、$\mathrm{partition}_3(A, a)$ は配列 A を a **より小さい**値からなる配列 B、a と**等しい**値からなる配列 C と a **より大きい**値からなる配列 D へ分割する関数である。クイックセレクトについても同様の改善が考えられる。そうすると全ての要素が等しい場合はすぐに計算が終わることが分かる。

5　中央値の中央値アルゴリズム

乱数を用いないで決定的に $O(n)$ 時間で中央値を計算することもできる。まず、n 個の要素を 5 つづつ $\lceil n/5 \rceil$ 個のグループに分ける。各グループの中央値を計算して、中央値を集めて長さ $\lceil n/5 \rceil$ の配列を作る。この長さ $\lceil n/5 \rceil$ の配列の中央値を再帰で計算し、その結果をピボットとして選択する。アルゴリズムを以下に示す。

アルゴリズム 1 中央値の中央値アルゴリズムの擬似コード (入力: 整数の配列 A，非負の整数 k. 出力: 配列 A の $k+1$ 番目に小さい要素．)

---

if 配列 A の長さが 5 以下 **then**
　解を計算して出力する.
else
　配列 A を長さ 5 づ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 B と「ピボットより大きいもの」からなる配列 C の 2 つの配列に分割する.
　配列 B 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 C の中から $k - r$ 番目に小さい要素を解として出力する (再帰呼出).
　else
　　配列 B の中から $k+1$ 番目に小さい要素を解として出力する (再帰呼出).
　end if
end if

このアルゴリズムを「中央値の中央値アルゴリズム」と呼ぶ。中央値の中央値アルゴリズムの時間計算量 $T(n)$ は次の漸化式を満たす (切り捨て、切り上げの影響は無視することにする)。

$$T(n) \leq T(n/5) + T(7n/10) + cn$$

ある定数 $d > 0$ について $T(n) \leq dn$ を帰納法で示す。帰納法の仮定を用いると

$$T(n) \leq \frac{d}{5}n + \frac{7d}{10}n + cn = \frac{9d + 10c}{10}n.$$

よって $d \geq 10c$ となるような d を選べば、$T(n) \leq dn$ を帰納法で示せる。

6　演習課題

1. クイックセレクトのソースコードを参考にしてクイックソートのプログラムを書け。
2. クイックセレクト及びクイックソートのプログラムを改良して、partition_3 を用いるようにせよ。重複した要素を持つ配列に対しても、適切にピボットを選べばそれぞれ $O(n)$ 時間、$O(n \log n)$ 時間で動くようにすること。ただし、ソートする配列以外に別の配列を使ってはならない。
3. [発展的課題] 中央値の中央値アルゴリズムのプログラムを書け。