バブルソートの改良アルゴリズムの正当性証明

廣瀬佳典

2020/7/4

この記事では、バブルソートとその改良アルゴリズムを定義した上で、そのアルゴリズムの正当性を証明するものである。

1　最小値を求める

要素数が $n \neq 0$ の配列 $A_1, ..., A_n$ の最小値を求めるアルゴリズムは以下のとおりである。

```
1  x := A_1
2  for integer i from 2 to n:
3      if x > A_i:
4          x := A_i
5  print x
```

これは次のように書き換えることができる。

```
1  x_1 := A_1
2  for integer i from 2 to n:
3      if x_{i-1} > A_i:
4          x_i := A_i
5      else:
6          x_i := x_{i-1}
7  print x_n
```

[証明]

各 i について $x_i \in \{A_1, ..., A_i\}$ であるので $x_n \in \{A_1, ..., A_n\}$ 。また、 $y \in \{A_1, ..., A_i\}$ について $x_i \leq y$ であるので $y \in \{A_1, ..., A_n\}$ について $x_i \leq y$ である。従って、 $x_n = \min(A_1, ..., A_n)$ 。

2　並べ替え

ここでは、ソートにおいて重要な概念である並べ替えについて定義する。並べ替えの概念は、配列が重複を持つ場合があるので必要となる。配列が重複を持たない場合は配列の集合が一致することが並べ替えであることの必要十分条件となる。

2.1 入れ替え

長さが $n \geq 2$ の配列 $A_1, ..., A_n$ について配列 $B_1, ..., B_n$ が次の性質を満たすならば、 $B_1, ..., B_n$ は $A_1, ..., A_n$ の入れ替えであるとする:

$$\text{ある2 つの整数 } i, j(1 \leq i < j \leq n) \text{ と任意の整数 } k \text{ について、} \tag{1}$$
$$B_i = A_j, \tag{2}$$
$$B_j = A_i, \tag{3}$$
$$k \neq i \text{ and } k \neq j \Longrightarrow B_k = A_k. \tag{4}$$

2.2 並べ替え

配列 $B_1, ..., B_n$ が配列 $A_1, ..., A_n$ の並べ替えであるということは、 $B_1, ..., B_n$ が $A_1, ..., A_n$ に有限回入れ替えを行ったもののことを指す。

3 バブルソート

バブルソートの正当性について議論する。なお、ここではバブルソートの安定性については議論しない。

3.1 問題

要素数が $n \geq 2$ の配列 $A_1, ..., A_n$ について、次の性質を満たす配列 $B_1, ..., B_n$ を求めること:

$$\text{任意の正の整数 } i, j \text{ について } i < j \Longrightarrow B_i \leq B_j, \tag{5}$$
$$\text{配列 } B_1, ..., B_n \text{は配列 } A_1, ..., A_n \text{の並べ替え}. \tag{6}$$

ちなみにプログラムのソートにおいて配列長が 0 または 1 の場合は自明なので扱わないことにする。

3.2 バブルソートのステップ

配列長が n の配列 $A_1, ..., A_n$ のバブルソートのステップ i は配列 $S_{i1}, ..., S_{in}$ と表記し、次の性質を満たすものとする:

$$S_{ii} \leq S_{i,i+1}, ..., S_{ii} \leq S_{in}, \tag{7}$$
$$\text{任意の正の整数 } j, k \text{ について } j \leq k \leq i \Longrightarrow S_{ji} \leq S_{ki}, \tag{8}$$
$$\text{配列 } S_{i1}, ..., S_{in} \text{は配列 } A_1, ..., A_n \text{の並び替え}. \tag{9}$$

バブルソートのステップ $n-1$ においてはソートが完了していることに注意。

3.3 バブルソートのアルゴリズム

基本的なバブルソートは比較を $n(n-1)/2$ 回行う。

```
1 x_1, ..., x_n := A_1, ..., A_n
2 for i from 1 to n-1:
3     for j from n-1 to i step -1:
4         if x_j > x_{j+1}:
5             x_j, x_{j+1} := x_{j+1}, x_j
6 B_1, ..., B_n := x_1, ..., x_n
```

これは次のように書き換えることができる。

```
1 S_0 := [A_1, ..., A_n]
2 for i from 1 to n-1:
3     [x_1, ..., x_n] := S_{i-1}
4     for j from n-1 to i step -1:
5         if x_j > x_{j+1}:
6             x_j, x_{j+1} := x_{j+1}, x_j
7     S_i := [x_1, ..., x_n]
8 B_1, ..., B_n := S_{n-1}
```

このコードは、配列 $A_1, ..., A_n$ 、つまりステップ 0 からステップ $n-1$ を計算している。ステップ $i-1$ からステップ i への計算を抜き出す。

```
1 [x_1, ..., x_n] := S_{i-1}
2 for j from n-1 to i step -1:
3     if x_j > x_{j+1}:
4         x_j, x_{j+1} := x_{j+1}, x_j # 入れ替えが発生
5 S_i := [x_1, ..., x_n]
```

この計算によって $S_{i-1,i}, ..., S_{i-1,n}$ から $S_{ii}, ..., S_{in}$ への並び替えが発生し、$S_{ii} = \min(S_{i-1,i}, ..., S_{i-1,n})$ となる。

[証明]

$S_{ii} \neq \min(S_{i-1,i}, ..., S_{i-1,n}) = m$ とする。$S_{i-1,i} = m$ であった場合は $S_{i-1,i}$ と $S_{i-1,i+1}$ の入れ替えが発生しないので矛盾する。$S_{i-1,i} \neq m$ で $S_{i-1,i+1} = m$ の場合は $S_{i-1,i+1}$ と $S_{i-1,i+2}$ の入れ替えが発生せず $S_{i-1,i}$ と $S_{i-1,i+1}$ の入れ替えが発生するので矛盾する。$S_{i-1,i} \neq m$ かつ $S_{i-1,i+1} \neq m$ の場合でもある正の整数 j について $S_{i-1,j} = m$ となり、$S_{i-1,j-1}$ と $S_{i-1,j}$ の入れ替えが発生し、その後連続的に入れ替えが起こり $S_{i-1,i}$ と $S_{i-1,j}$ の入れ替えまでが発生するので矛盾する (入れ替えが途中で止まった場合は $S_{i-1,j}$ が止まったところの一つ手前も m であるのでそこが j となる)。境界的なケースにおいては場合分けしたケース自体が存在しない場合もあるがそのケースにおいてもすべての場合で矛盾することに変わりはない。従って $S_{ii} = \min(S_{i-1,i}, ..., S_{i-1,n})$ である。

4　バブルソートの改良アルゴリズム

バブルソートのステップ $i-1$ において、$S_{i-1,i}=\min(S_{i-1,i},...,S_{i-1,n})$ であった場合は $S_{i-1,i}$ は動くことがない。これと同様にステップ i の計算においてソートされていることが分かっていれば多くのステップを省略することができる。そこで次のアルゴリズムを導入する。

```
1  x_1,...,x_n := A_1,...,A_n
2  i = 1
3  while i < n:
4      i_next := n
5      for j from n-1 to i step -1:
6          if x_j > x_{j+1}:
7              x_j, x_{j+1} := x_{j+1}, x_j # 入れ替えが発生
8              i_next := j+1
9      i := i_next
10 B_1,...,B_n := x_1,...,x_n
```

ステップ $i-1$ においてステップ i を計算するとともに計算後の配列がどこまでソートされているかを i_{next} で探索している。このアルゴリズムの正当性を示す。

[証明]

ステップ $i-1$ を考える。このアルゴリズムの j ループの終わりまで到達した時、つまり8行目の終わりにおいて $x_j=\min(S_{i-1,j},...,S_{i-1,n})$ である。$j<k$ のときに入れ替えが発生しない場合、$1\leq l\leq m\leq k\Longrightarrow x_l\leq x_m$ であり $k<l\leq n\Longrightarrow x_k\leq x_l$ であるので、ステップ i の計算完了時にステップ k であることも示される。ここで、i_{next} は $k+1$ となる。

謝辞

質問サイト teratail において本アルゴリズムを教えてくださった LouiS0616 様、hope_mucci 様、並びに teratail の皆様に感謝申し上げます。