データ構造とアルゴリズム

演習課題 4

HI4 45 号 山口惺司

提出日: 2024/07/15

1. 問題

バブルソートについて、ダブルバブルソートというアルゴリズムがある。
この方法について、{2, 8, 3, 4, 7, 6, 1, 5}

2. アルゴリズムの説明

まずはバブルソートについて説明する。
バブルソートは、隣同士の要素間でしか比較も交換もできないという条件下でソートするアルゴリズムである。
アルゴリズムは以下の通り。

1. 隣同士のペアを比較して、降順なら入れ替える.（ということを最後尾から順に先頭まで行う）
2. 戦闘を除いて残りの列をソート対象とし、1 に戻る。
3. ソート対象の列の要素が 1 個になったら停止する。

例として{9, 2, 7, 4, 5}というデータが与えられたとき、以下のような経過でバブルソートが行われる。

① {(9, 2, 7, 4, 5)} が与えられた
② {(9, 2, 7, 4, 5)} 4 と 5 を比較して降順なのでそのまま
③ {(9, 2, 7, 4, 5)} 7 と 4 を比較して降順ではないため入れ替え
④ {(9, 2, 4, 7, 5)} 2 と 4 を比較して降順なのでそのまま
⑤ {(9, 2, 4, 7, 5)} 9 と 2 を比較して降順ではないため入れ替え
⑥ {2, (9, 4, 7, 5)} 7 と 5 を比較して降順ではないため入れ替え
⑦ {2, (9, 4, 5, 7)} 4 と 5 を比較して降順なのでそのまま
⑧ {2, (9, 4, 5, 7)} 9 と 4 を比較して降順ではないため入れ替え
⑨ {2, 4, (9, 5, 7)} 5 と 7 を比較して降順なのでそのまま
⑩ {2, 4, (9, 5, 7)} 9 と 5 を比較して降順ではないため入れ替え
⑪ {2, 4, 5, (9, 7)} 9 と 7 を比較して降順ではないため入れ替え
⑫ {2, 4, 5, 7, (9)} ソート対象の列の要素が 1 個になったためソート終了

このバブルソートを改良したものがダブルバブルソートである。
{2, 8, 3, 4, 7, 6, 1, 5}というデータが与えられたとき、以下の経過でダブルバブルソートが行われる。

① {(2, 8, 3, 4, 7, 6, 1, 5)} が与えられた
② {(2, 8, 3, 4, 7, 6, 1, 5)} データの左からバブルソートを右向きに適用。隣同士のペアを比較して昇順でないなら入れ替える
③ {(2, 3, 4, 7, 6, 1, 5), 8} 右端の最大値(8)は固定。その左側(5)から左向きにバブルソートを適用。隣同士のペアを比較して降順でないなら入れ替える
④ {1, (2, 3, 4, 7, 6, 5), 8} 左端の最小値(1)は固定。その右側(2)から②と同様にバブルソートを行う。
⑤ {1, (2, 3, 4, 6, 5), 7, 8} 右端の最大値(7)は固定。その左側(5)から③と同様にバブルソートを行う。
⑥ {1, 2, (3, 4, 5, 6), 7, 8} ②と同様
⑦ {1, 2, (3, 4, 5), 6, 7, 8} ③と同様
⑧ {1, 2, 3, (4, 5), 6, 7, 8} ②と同様
⑨ {1, 2, 3, (4), 5, 6, 7, 8} 中央部が 1 個になったため終了

3. フローチャート

ダブルバブルソートのフローチャートを図 1 に示す。

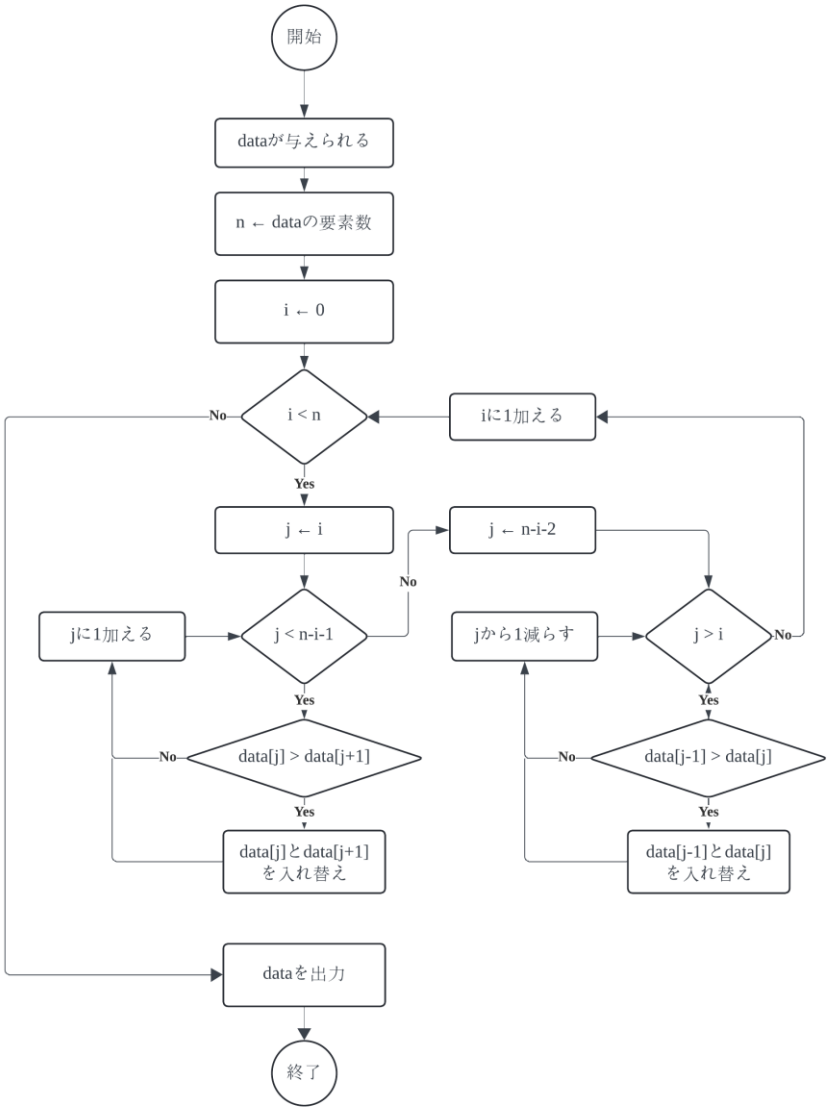

図 1 ダブルバブルソートフローチャート

4. 疑似コード

ダブルバブルソートの疑似コードを以下に示す。

① data が与えられる。
② n=data の要素数。
③ i = 0
④ i < n ではない場合⑭へ進む。
⑤ j = i
⑥ j < n-i-1 ではない場合⑨へ進む。
⑦ data[j] > data[j+1]の場合、data[j]と data[j+1]を入れ替える。
⑧ j に 1 加え⑥へ戻る。
⑨ j = n-i-2

⑩　j > i ではない場合へ⑬進む。
⑪　data[j-1] > data[j]の場合、data[j-1]と data[j]を入れ替える。
⑫　j から 1 減らし⑩へ戻る。
⑬　④へ戻る。
⑭　データを出力し、終了する。

5. プログラム

上記のフローチャートや疑似コードを基に、比較回数と交換回数のカウントを追加したダブルバブルソートのプログラムを以下に示す。

```
def double_bubble_sort(data):
    n = len(data)
    changeCount = 0
    compareCount = 0
    for i in range(n):
        for j in range(i, n-i-1):
            if data[j] > data[j+1]:
                data[j], data[j+1] = data[j+1], data[j]
                changeCount += 1
            compareCount += 1
        for j in range(n-i-2, i, -1):
            if data[j-1] > data[j]:
                data[j], data[j-1] = data[j-1], data[j]
                changeCount += 1
             compareCount += 1

    print("比較回数:", compareCount)
    print("交換回数:", changeCount)
    return data

data = [2, 8, 3, 4, 7, 6, 1, 5]
print("ソート前:", data)
data = double_bubble_sort(data)
print("ソート後:", data)
```

6. 実行結果

実行結果を図 2～9 に示す。

```
ソート前: [2, 8, 3, 4, 7, 6, 1, 5]
比較回数: 28
交換回数: 14
ソート後: [1, 2, 3, 4, 5, 6, 7, 8]
```

図 2 実行結果 1

```
ソート前： [1, 2, 3, 4, 5, 6, 7, 8]
比較回数： 28
交換回数： 0
ソート後: [1, 2, 3, 4, 5, 6, 7, 8]
```

図 3 実行結果 2

```
ソート前： [5]
比較回数： 0
交換回数： 0
ソート後: [5]
```

図 4 実行結果 3

```
ソート前： [5, 1]
比較回数： 1
交換回数： 1
ソート後: [1, 5]
```

図 5 実行結果 4

```
ソート前： [5, 1, 3]
比較回数： 3
交換回数： 2
ソート後: [1, 3, 5]
```

図 6 実行結果 5

```
ソート前： [5, 1, 3, 2]
比較回数： 6
交換回数： 4
ソート後: [1, 2, 3, 5]
```

図 7 実行結果 6

```
ソート前： [2, 1, 4, 2, 5]
比較回数： 10
交換回数： 2
ソート後: [1, 2, 2, 4, 5]
```

図 8 実行結果 7

```
ソート前： [3, 4, 1, 9, 4, 5]
比較回数： 15
交換回数： 4
ソート後: [1, 3, 4, 4, 5, 9]
```

図 9 実行結果 8

7. 考察

全ての実行結果において、正しくソートできているため、作成したプログラムは正しいと言える。
また、与えられたデータの要素数が偶数個でも奇数個でも正しく動作していることがわかる。
図 2, 3 を見ると、同じ要素数では比較回数は同じでも交換回数は与えられたデータによって変わることがわかる。
図 4~9 を見て、与えられたデータの要素数と比較回数を表にすると表 1 のようになる

表 1 与えられたデータの要素数と比較回数の関係

要素数	1	2	3	4	5	6
比較回数	0	1	3	6	10	15

表 1 より、与えられたデータの要素数を n とすると、任意の n においてダブルバブルソートの比較回数は以下の式で求められる。

$$比較回数 = \frac{n(n-1)}{2}$$

これは任意の n においてのバブルソートの比較回数と同じである.

8. 感想

片側からソートするバブルソートを、ダブルバブルソートは往復するようにしてソートをするが面白い発想だなと感じた。
バブルソートもダブルバブルソートも比較回数は同じなので、ダブルバブルソートは本当に効率化できているのか疑問に思った。