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

## 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}

## 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

## 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。

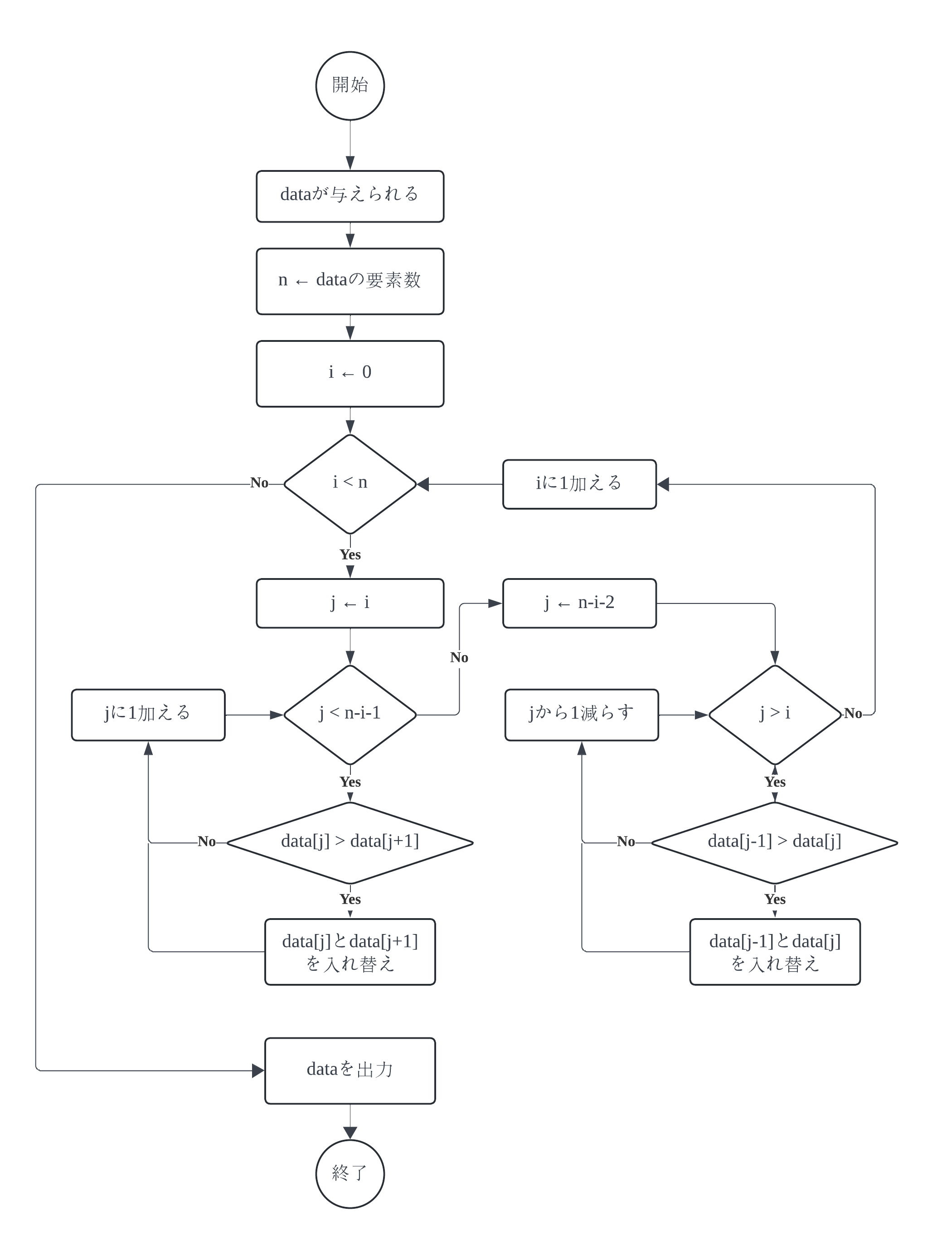


図1 ダブルバブルソートフローチャート

## 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。

## 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)

## 実行結果

実行結果を図2～9に示す。

時計と文字の加工写真

低い精度で自動的に生成された説明

図2 実行結果1

文字が書かれている

中程度の精度で自動的に生成された説明

図3 実行結果2

ダイアグラム, QR コード

自動的に生成された説明

図4 実行結果3

文字が書かれている

中程度の精度で自動的に生成された説明

図5 実行結果4

時計 が含まれている画像

自動的に生成された説明

図6 実行結果5

時計と文字の加工写真

低い精度で自動的に生成された説明

図7 実行結果6

時計 が含まれている画像

自動的に生成された説明

図8 実行結果7

時計 が含まれている画像

自動的に生成された説明

図9 実行結果8

## 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。

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.

## 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。