データ構造とアルゴリズム

演習課題3

提出日2024/07/01

HI4 45号 山口惺司

1. 問題

挿入ソートについて、ダブル挿入ソートという改良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}が与えられたとき、ダブル挿入ソートを利用してソートせよ。

1. アルゴリズムの説明

以下のような経過でダブル挿入ソートが行われる。

1. {2, 8, 3, 4, 7, 6, 1, 5} が与えられた(偶数個の場合のみ)
2. {2, 8, 3, (4, 7), 6, 1, 5} 中央の2つを大小順に入れ替え
3. {2, 8, 3, (4, 7), 6, 1, 5} 中央部の両側の2つを大小順に入れ替え
4. {2, 8, (3, 4, 7), 6, 1, 5} 中央部の左側を挿入ソートで中央部が昇順になるよう挿入
5. {2, 8, (3, 4, 6, 7), 1, 5} 中央部の右側を挿入ソートで中央部が昇順になるよう挿入
6. {2, 1, (3, 4, 6, 7), 8, 5} 操作③
7. {2, (1, 3, 4, 6, 7), 8, 5} 操作④
8. {2, (1, 3, 4, 6, 7, 8), 5} 操作⑤
9. {2, (1, 3, 4, 6, 7, 8), 5} 操作③
10. {(1, 2, 3, 4, 6, 7, 8), 5} 操作④
11. {(1, 2, 3, 4, 5, 6, 7, 8)} 操作⑤
12. {(1, 2, 3, 4, 5, 6, 7, 8)} 終了
13. フローチャート

ダブル挿入ソートのフローチャートを図1に、挿入ソート関数のフローチャートを図2に示す。ダイアグラム

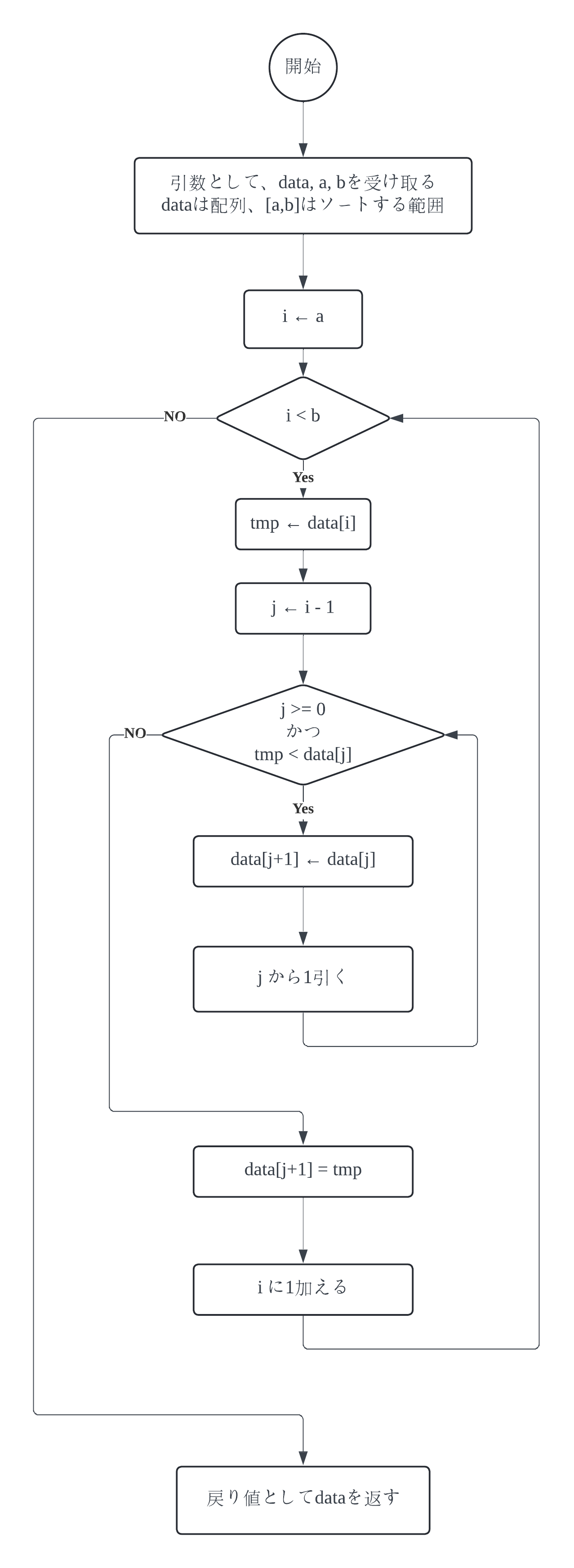
自動的に生成された説明

図1 ダブル挿入ソートのフローチャート　 図2 挿入ソート関数のフローチャート

1. 疑似コード

ダブル挿入ソートと挿入ソート関数の疑似コードを以下に示す。

ダブル挿入ソート：

1. dataが与えられる。
2. n=dataの要素数、m=dataの要素数の半分
3. data[m] < data[m-1]であればdata[m]とdata[m-1]を入れ替え
4. i = 1
5. i < mでなければ⑩へ進む
6. data[m+i] < data[m-i-1]であればdata[m+i] と data[m-i-1]を入れ替え
7. 挿入ソート関数にdata, [m-i, m+i]を渡す
8. 挿入ソート関数にdata, [m-i, m+i+1]を渡す
9. iに1加えて⑤に戻る
10. dataを出力する

挿入ソート関数：

1. data, 範囲[a, b]が与えられる
2. i = a
3. i < bでなければ⑧へ進む
4. tmp = data[i], j = i -1
5. j >= 0 かつ j >= a-1でなければへ進む
6. tmp < data[j]であればdata[j+1] = data[j], jから1減らし、⑤へ戻る、そうでなければ⑦へ進む
7. data[j+1] = tmp, ③へ戻る
8. dataを返す
9. プログラム

作成したフローチャートと疑似コードを基に、比較回数と交換回数をカウントする処理を加えたPythonプログラムを以下に示す。

def double\_insertion\_sort(data):　 #ダブル挿入ソート関数

    chcount, cocount = 0, 0 #比較回数と交換回数

    n = len(data)

    m = int(n / 2)

    if data[m] < data[m - 1]:

        data[m], data[m - 1] = data[m - 1], data[m]

        chcount += 1 # 交換回数を増やす

    for i in range(1, m):

        if data[m + i] < data[m - i - 1]:

            data[m + i], data[m - i - 1] = data[m - i - 1], data[m + i]

            chcount += 1 # 交換回数を増やす

        a, b = insertion\_sort(data, m - i, m + i)

        chcount += a # 交換回数を増やす

        cocount += b # 比較回数を増やす

        a, b = insertion\_sort(data, m - i, m + i + 1)

        chcount += a # 交換回数を増やす

        cocount += b # 比較回数を増やす

    print("交換回数:", chcount)

    print("比較回数:", cocount)

    return data

def insertion\_sort(data, a, b): #挿入ソート関数

    chcount, cocount = 0, 0

    for i in range(a, b):

        tmp = data[i]

        j = i - 1

        while j >= 0 and j >= a - 1:

            cocount += 1  # 比較回数を増やす

            if tmp < data[j]:

                data[j + 1] = data[j]

                j -= 1

                chcount += 1  # 交換回数を増やす

            else:

                break

        data[j + 1] = tmp

    return chcount, cocount

def main(): #main関数

    data = [2, 8, 3, 4, 7, 6, 1, 5] #与えられたデータ

    print("元のデータ:", data)

    sorted\_data = double\_insertion\_sort(data)

    print("ソート後のデータ:", sorted\_data)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. 実行結果

実行結果を図3～図7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3 [2, 8, 3, 4, 7, 6, 1, 5]を与えたときの実行結果

カレンダー が含まれている画像

自動的に生成された説明

図4 [1, 2, 3, 4, 5, 6]を与えたときの実行結果

時計 が含まれている画像

自動的に生成された説明

図5 [0, -1, 2, 3, -2, 2, 9, 4]を与えたときの実行結果

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6 [2, 8, 3, 4, 7, 6, 1, 5]を与えたときの実行結果

グラフ が含まれている画像

自動的に生成された説明

図7 [3, 5, 2, 1, 0]を与えたときの実行結果

1. 考察

図3~6では正しくソートできていることがわかる。

図7では与えられたデータの要素数が奇数であったため、正しくソートできなかった。

また、図3, 4, 6を見ると、与えられたデータが同じ要素数でも比較回数と交換回数が違うことが分かった。

1. 感想

コードが挿入ソートと比べて明らかに長くなっているが、挿入ソートよりダブル挿入ソートの方がより効率化されていることが驚きである。