提出日：2024/5/31

プログラミング演習　第7回演習レポート

担当教員：杉本　千佳先生

　　　　　　　　　　　　　　　　所属：理工学部　数物・電子情報系学科

　電子情報システムEP

　　　　　　　　　　　　学年・クラス：２年　Fe1

　　　　　　　　　　　　　　学籍番号：2364092

　　　　　　　　　　　　　　　　氏名：熊田　真歩

（１）課題番号：基本課題２

　　　課題名：整列過程におけるデータの比較回数とデータの入れ換え回数を表示し、整列データをファイルに書き込む

（２）プログラムのフローチャート（関数ごと）

ダイアグラム

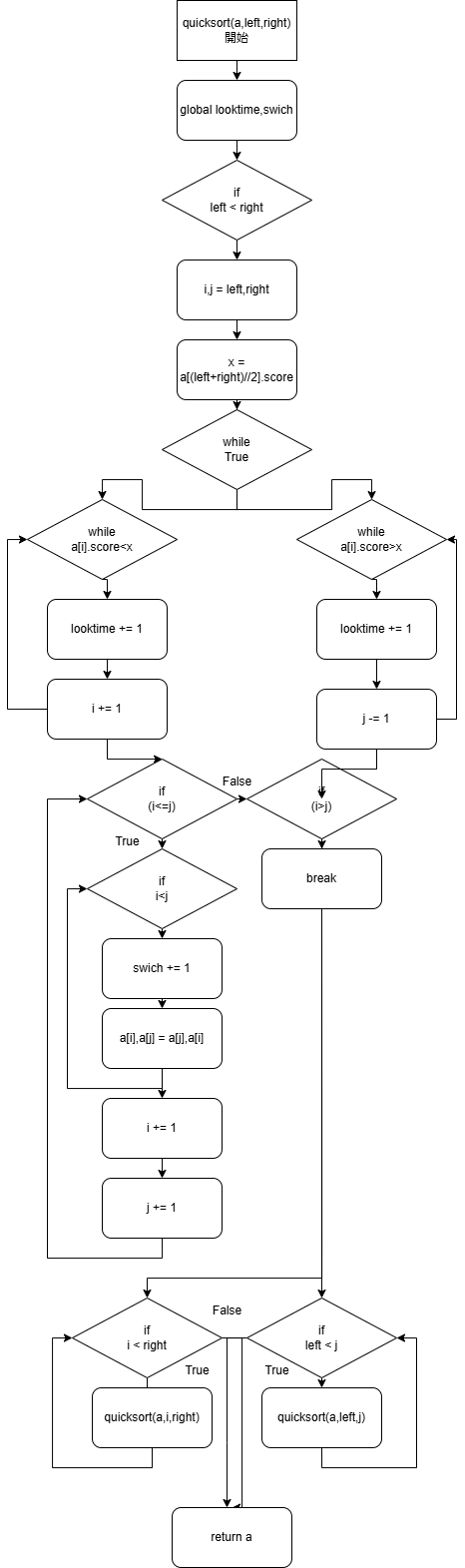
自動的に生成された説明ダイアグラム

自動的に生成された説明・クラスの定義関数　　　　　　　・aditem(item\_list,number:int,name:str,score:int)関数

ダイアグラム

自動的に生成された説明ダイアグラム

自動的に生成された説明・csvreader(filename:str, item\_list)関数　　　　　・outout(item\_list,newfile)関数

・quicksort関数

（３）アルゴリズムが「正しいこと」である説明

　　本プログラムでは学籍番号、名前、成績が記入されたCSVファイルに対してクイック

ソートを用いて成績順に並び替え、ファイルに書き込むという事を行った。ここで、入力フ

ァイル名に関しては入力時のミスによるエラーを回避するために入力式ではなく予め入力

してあるという形を採用した。よって、以下では入力ファイルが存在するという条件のもと、

アルゴリズムの正しさについて議論する。

まず、すべての入力に対してプログラムは停止するということについてだが、上記で述べ

た通り本プログラムでは入力する内容は入出力のファイル名のみであり、入力では存在するCSVファイル入れればよい。しかし、CSVは学籍番号、名前、成績の要素からなるものでなくてはいけない。この条件がそろうCSVファイルであればアルゴリズムはどんな入力に対しても停止する。

　次にすべての入力に対してアルゴリズムは「正しい答え」を出力することについて議論する。上記と同じ通り、ここですべての入力とは条件を満たすCSVが存在し、それが入力ファイルとして存在しているという条件下のもと議論する。課題ではデータ数が8,288，2240個の時に対してプログラムを実行し、いずれの場合も正しい出力結果が得られた。すなわち、データ数の量によらずこのプログラムは正しい答えを出力することが分かる。すなわちこの条件に当てはまるCSVファイルであればすべての入力に対してアルゴリズムは正しい答えを出力する。

（４）ソース・プログラムの説明

import csv           #csvファイルを用いるための設定

looktime = 0         #looktime（比較回数）の初期値は0

swich = 0            #swich（入れ替え回数）の初期値は0

class Item:          #クラス、Itemの定義

  def \_\_init\_\_ (self, number:int, name:str, score:int):#クラスの設定

   self.number = number       #生徒番号の設定

   self.name = name           #名前の設定

   self.score = score         #成績の設定

def aditem(item\_list,number:int,name:str,score:int):#クラスに要素を追加する関数

  item\_list.append(Item(number,name,score))#リストに要素を追加

def csvreader(filename:str, item\_list):  #CSVファイルを読み込む関数

  with open(filename,'r') as f:          #指定されたファイルを開く

    csvr = csv.reader(f)                 #CSVファイルの読み込み

    next(csvr)                           #読み込みをスキップ

    for row in csvr:                     #CSVファイルの1行ずつに対して

      number = int(row[0])               #学籍番号はint型

      name = str(row[1])                 #名前はstr型

      score = int(row[2])                #成績はint型

      aditem(item\_list, number, name, score)   #これらをリストに追加

def outout(item\_list,newfile):            #出力関数

  global looktime,swich                   #looktimeとswichをグローバル変数とする

  looktime = 0                            #looktimeの初期値は0

  swich = 0                               #swichの初期値は0

  alist = quicksort(item\_list,0, len(item\_list)-1)  　　　　　 #alistの値をクイックソートで並び　　　　　　　　　　　替えた関数とする

  f = open(newfile,"w")                                      #newfileをfとして開く

  quicksort(item\_list,0, len(item\_list)-1)                   #クイックソート関数の呼び出し

  print("比較回数="+str(looktime),"入れ替え回数="+str(swich)) #比較回数と入れ替え回数の出力

  for item in alist:                                       #alistのそれぞれのitemに対して

    f.write(f"{item.number:4},{item.name:16},{item.score}\n")#ファイルに内容を書き込む

  print("出力ファイル名："+str(newfile))                      #出力ファイル名の出力

def quicksort(a:list, left:int,right:int):   #順序を入れ替えるクイックソート関数

    global looktime,swich                    #looktime（比較回数）,swich（入れ替え回数）をグロ　　　　　ーバル変数とする

    if left < right:                         #rightがleftより大きい時

        i, j = left, right                   #i=left,j=rightとする

        x = a[(left + right) // 2].score     #リストの中央にあるスコアの値をxに代入

        while True:                          #条件が正しい間

            while(a[i].score < x):           #i番目リストのスコアがリストの中央のスコアより小さ　　　　　い時（中央より左側について）

                looktime += 1                #比較回数を1回増やす

                i += 1                       #iをインクリメントすることで比較対象を次の値へとずら　　　　　　　す（右にずらす）

            while(x < a[j].score):           #i番目リストのスコアがリストの中央のスコアより小さ　　　　　　　　い時（中央より右側について）

                looktime += 1                #比較回数を1回増やす

                j -= 1                       #jを負にインクリメントすることで比較対象を次の値へ　　　　　　とずらす（左にずらす）

            if (i <= j):                     #iの値がj以下になったら

                if i < j:                    #jの値がiの値よりも大きい時

                   swich += 1               #入れ替え回数の値を1回増やす

                  a[i], a[j] = a[j], a[i]　#リストのi番目のデータとj番目のデータを入れ替える

                i += 1                       #iの値を増やす

                j -= 1                       #jの値を減らす

            if i > j:                        #iの値がjの値よりも大きい時

                break                        # ループ終了

        if left < j:                         #最も左の値がjよりも小さい時

            quicksort(a, left, j)            #quicksort(a, left, j)関数の呼び出し

        if i < right:                        #最も左の値がiよりも大きい時

            quicksort(a , i, right)          #quicksort(a , i, right)関数の呼び出し

        return a                             # aに値を返す

item\_list = []                               #item\_listの初期値は空

csvreader('data\_8.csv', item\_list)           #入力ファイル

outout(item\_list,"output\_8.txt")             #出力ファイル

（５）考察

　本プログラムでは前回に引き続き、CSVファイルの内容の書き換えを行った。しかし、

第6回とは異なり、CSVファイルのデータの入れ替えを行う際にクイックソートを用いて

並び替えを行った。ここで前回の入れ替え方法とクイックソートを用いた入れ替え方法を

比較してそのメリット・デメリットについて考察する。以下にdata\_8.csv, data\_288.csv,

data\_2240.csvの第6回と第7回の入れ替え方法でファイルの内容を並び替えた時の比較回

数と入れ替え回数を示す。

|  |  |  |
| --- | --- | --- |
|  | 第6回のアルゴリズム | クイックソート(7回） |
| data\_8.csv | 比較：21　替え：5 | 比較：18　替え：5 |
| data\_288.csv | 比較：41041　替え：282 | 比較：3074　替え：818 |
| data\_2240.csv | 比較：2505441  替え：2216 | 比較：27147  替え：11494 |

　表から明らかにわかることとして、第6回で採用したアルゴリズムは比較回数がクイッ

クソートに比べ多い。しかし、入れ替え（表中の替え）回数はクイックソートの方がより多

い傾向にある。ここで「効率の良いプログラム」とはこの表中の比較回数のより少ないもの

である。比較して入れ替えなくとも、作業に要する時間は比較すればかかることになるから

である。ここで、クイックソートではなぜ比較回数がより少なくて済むのかという点につい

て考察する。クイックソートは中央の値をピンで固定することで、二つのグループに分割し、

比較を行う。これにより比較回数は初めからずらしていくプログラムに比べ比較回数がよ

り少なくなることが分かる。表からも明らかなように、このクイックソートはそのデータ数

が多いほど効果を発揮する。膨大なデータ数を扱う際には不可欠なアルゴリズムであるこ

とが分かる。また、本プログラムではクイックソートはリストの中の数を扱ったため、right

の値はlen(item\_list)-1と－１をしたことに注意した。そしてleftの値は0とした。

　プログラムの改善の余地について考察する。本プログラムでは、CSVファイルが学籍番

号、名前、成績の三要素からなるものでないと正常に稼働しない。この条件以外のものでも

稼働するようなプログラムは非常に複雑であるが、この条件を満たさない場合はエラーメ

ッセージを表示する等の処理が加わるとより汎用性の高いプログラムとなると考えた。

（６）参考文献

・クイックソートに関して

①アルゴリズム、スライドサンプルプログラム

②Zenn：ダーマン「Pythonでクイックソート完全解説」

https://zenn.dev/yutabeee/articles/e8fb2847cfc980　　2024/05/31アクセス

③スズメの本棚「Pythonでクイックソートを実装してみた」

https://tech-shelf.hatenablog.com/entry/algorithm/quicksort　　2024/05/31アクセス

（７）感想

　前回のプログラムではクラスの内部に関数を作成することを試みてアルゴリズムを組んだが、本プログラムではそれを改良し関数をクラスの外部に作成した。結果論であるが、関数はクラスの外部に設定しする方がより明確でコードを組むのも容易であった。クイックソートはアルゴリズムのサンプルプログラムのものをそのまま用いたが、意味の理解がはっきりできていなかったクイックソートの使い方が明確にわかるようになり、やはり実際に習ったプログラムを活用してコードを書いてみる重要性を感じた。また、比較回数と入れ替え回数ではクイックソートのメリットを実感でき良い学びへとつながった。しかしながら、クイックソートに関してはアルゴリズムを自分で作成したのではなく、アルゴリズムのサンプルプログラムを用いたため、自分でクイックソートを作れるようになったかというと怪しい、というか書けない。すなわち、これを自力でしっかりかけるようにすることは今後の課題であると思った。