提出日：2024/6/13

プログラミング演習　第 9 回演習レポート

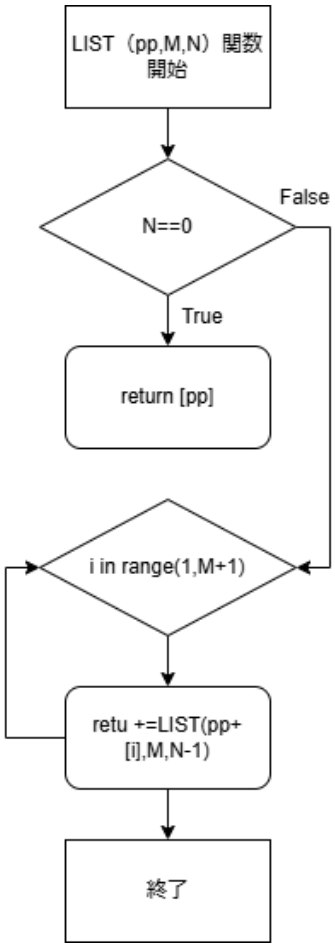
担当教員：杉本　千佳先生
所属：理工学部　数物・電子情報系学科
電子情報システム EP
学年・クラス：２年　Fe1
学籍番号：2364092
氏名：熊田　真歩

（１）課題番号：基本課題２

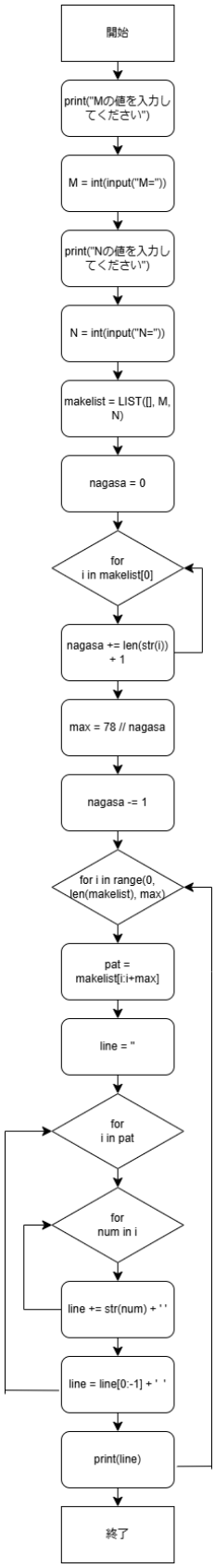
課題名：整列過程におけるデータの比較回数とデータの入れ換え回数を表示し、整列データをファイルに書き込む

（２）プログラムのフローチャート

・LIST(pp,M,N)関数



・メインプログラムのフローチャート



（３）アルゴリズムが「正しいこと」である証明

本プログラムでは、1 桁の正数 N および M を入力し 1 から M を N 個並べるプログラムを作成した。作成した数列は 1 行にできるだけ多く入るように出力した。ここで全ての入力に対してプログラミングは停止することについて考察する。前提条件として本プログラムで入力する M および N の値は自然数である。M や N にどんな自然数を入力してもプログラムは正常に停止する。しかしながらあまりにも M や N の値が大きい際は必ず停止するものの、そのプログラム実行時間はかなり長くなるというデメリットはある。この実行時間については考察で詳しく言及する。次にどんな整数値 M,N を代入してもプログラムは正しい答えを出力することについて考察する。本プログラムでは、N と M にどんな自然数を代入してもプログラムは正しい答えを出力することが様々な数の入力実験により明らかとなった。しかしながら、上でも述べた通りその実行時間は M や N が大きいほど増加した。ここで例として N=M=1 の時の出力結果を表示すると以下のように正しくプログラムが機能していることが分かる。さらに、M=5、N=10 の時の結果もその下に示す。すべての出力結果は表示しきれていないが結果は正しく表示されていることを確認した。しかしながら、M や N が二桁を超える場合、使用可能な RAM をすべて使用した後で、セッションがクラッシュしましたとなりメモリが足りないためにプログラムの実行結果が表示されないことがあった。これについてはメモリの容量のより多いものを使えば解消する問題であるした。

```
1 から M までを並べる 1 桁の正数 M を入力してください
M=1
1 から M までの数を N 個を並べる 正数 N を入力してください
N=1
1
```

```
5 5 5 5 1 2 5 1 1 1  5 5 5 5 1 2 5 1 1 2  5 5 5 5 1 2 5 1 1 3
5 5 5 5 1 2 5 1 1 4  5 5 5 5 1 2 5 1 1 5  5 5 5 5 1 2 5 1 2 1
5 5 5 5 1 2 5 1 2 2  5 5 5 5 1 2 5 1 2 3  5 5 5 5 1 2 5 1 2 4
5 5 5 5 1 2 5 1 2 5  5 5 5 5 1 2 5 1 3 1  5 5 5 5 1 2 5 1 3 2
5 5 5 5 1 2 5 1 3 3  5 5 5 5 1 2 5 1 3 4  5 5 5 5 1 2 5 1 3 5
5 5 5 5 1 2 5 1 4 1  5 5 5 5 1 2 5 1 4 2  5 5 5 5 1 2 5 1 4 3
5 5 5 5 1 2 5 1 4 4  5 5 5 5 1 2 5 1 4 5  5 5 5 5 1 2 5 1 5 1
5 5 5 5 1 2 5 1 5 2  5 5 5 5 1 2 5 1 5 3  5 5 5 5 1 2 5 1 5 4
5 5 5 5 1 2 5 1 5 5  5 5 5 5 1 2 5 2 1 1  5 5 5 5 1 2 5 2 1 2
5 5 5 5 1 2 5 2 1 3  5 5 5 5 1 2 5 2 1 4  5 5 5 5 1 2 5 2 1 5
5 5 5 5 1 2 5 2 2 1  5 5 5 5 1 2 5 2 2 2  5 5 5 5 1 2 5 2 2 3
5 5 5 5 1 2 5 2 2 4  5 5 5 5 1 2 5 2 2 5  5 5 5 5 1 2 5 2 3 1
5 5 5 5 1 2 5 2 3 2  5 5 5 5 1 2 5 2 3 3  5 5 5 5 1 2 5 2 3 4
5 5 5 5 1 2 5 2 3 5  5 5 5 5 1 2 5 2 4 1  5 5 5 5 1 2 5 2 4 2
5 5 5 5 1 2 5 2 4 3  5 5 5 5 1 2 5 2 4 4  5 5 5 5 1 2 5 2 4 5
```

（４）ソース・プログラムの説明

```
def LIST(pp, M, N):#再帰的に組み合わせを生成する関数
    if N == 0:      #N 個の数字を並べ終えたとき
        return [pp]#組み合わせをリスト pp に返す
    retu = []       #結果を格納するリスト retu を初期化
    for i in range(1, M + 1):      #1 から M までの数値で数の組み合わせを作成
        retu += LIST(pp+[i],M,N-1)#結果を retu に追加
    return retu                    #組み合わせリスト retu を返す
```

```
print("1 から M までを並べる 1 桁の正数 M を入力してください")
M = int(input("M="))
print("1 から M までの数を N 個を並べる 正数 N を入力してください")
N = int(input("N="))
makelist = LIST([], M, N)#LIST 関数を呼び出し 1 から M までの数を N 個並べた組み合わせリスト
                          makelist を生成
nagasa = 0               #表示する文字の長さの初期値を 0 とする
for i in makelist[0]:
    nagasa += len(str(i)) + 1 #数字の後ろにスペースを 1 つ追加
max = 78 // nagasa            #1 行に出力できる組み合わせ数
nagasa -= 1                  #最後のスペースの削除
for i in range(0, len(makelist), max):
    pat = makelist[i:i+max]           #1 行に表示できる個数ごとにリスト化
    line = ''                        #表示する行を格納する文字列
    for i in pat:
        for num in i:
            line += str(num) + ' '    #数字を文字列に変換しスペースで区切って line に追加
        line = line[0:-1] + '  '      #スペースを取り除き、改行文字を追加
    print(line)                      #行 line を表示
```

（５）考察

本プログラムでは、1 桁の正数 N および M を入力し 1 から M を N 個並べるプログラムを作成した。作成した数列は 1 行にできるだけ多く入るように出力した。ここでアルゴリズムの正しさを実証するために M や N に様々な値を代入してプログラムの出力を行った。この際、N や M の値を大きくしていくとプログラムの実行にかかる時間は段々と増加した。具体的には M=3、N=3 の時の実行時間は 5 秒だったのに対し、M=5、N=１０の時の実行時間は 17 分 57 秒であるということである。更には、N=10、M=10 付近以上の値からは使用可能な RAM をすべて使用した後で、セッションがクラッシュしましたというメッセージが表示されてしまいプログラムの出力結果が表示されなかった。これはプログラムの書き方による問題であると考えられる。これは、本プログラムでは、作成したすべての文字の組み合わせをリストに入れ保存しているためであると考えた。そのため、M と N の値が大きくなると、生成される組み合わせの数が指数関数的に増加し、それを全てメモリに保持するため大量のメモリが必要になる。更に、N の大きさが大きい時は再帰呼び出しによりスタックメモリを大量に使用する仕組みになっているため、メモリとスタックメモリの

いずれかが容量オーバーになった可能性が高いと考察した。こういった観点から本プログラムには改善の余地がある。具体的には join やマッピングを使用することでメモリの使用量を大幅に減らすことができると考えた。

また、本プログラムでは作成した文字列を一行にひとつずつ表示させるのではなく、できるだけ一行に多く表示させるように工夫した。ここで一行ごとにリストを作成し、リストの中身を表示させることで一行に 78 文字と統一した文字数での出力を実現した。

（６）感想

Join やマッピングはまだ使いこなせる自信がなく使い慣れた for 文やリストを用いてプログラムの作成を行ったが予想よりもメモリの消費量が多く、実行時間もかなり長いというデメリットの大きいプログラムとなってしまった。マッピングを使う事による簡潔さやメモリの節約の効果の偉大さを実感した。やはり使えるコードのバラエティーを増やすことは不可欠なようだ。