提出日：2024/7/19

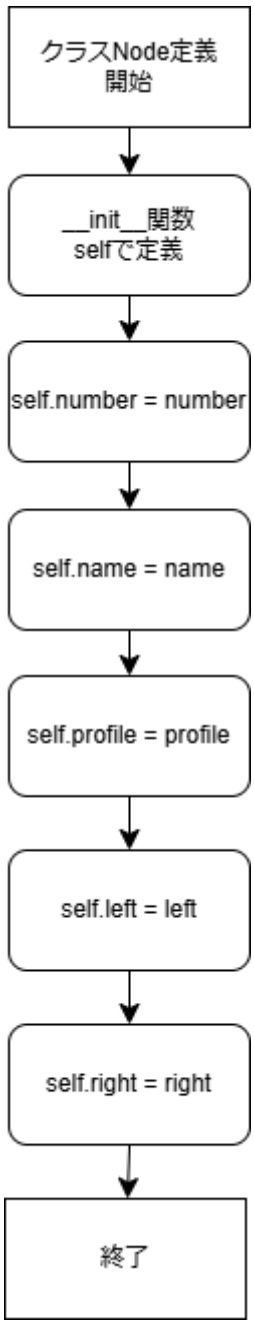
プログラミング演習　第14回演習レポート

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

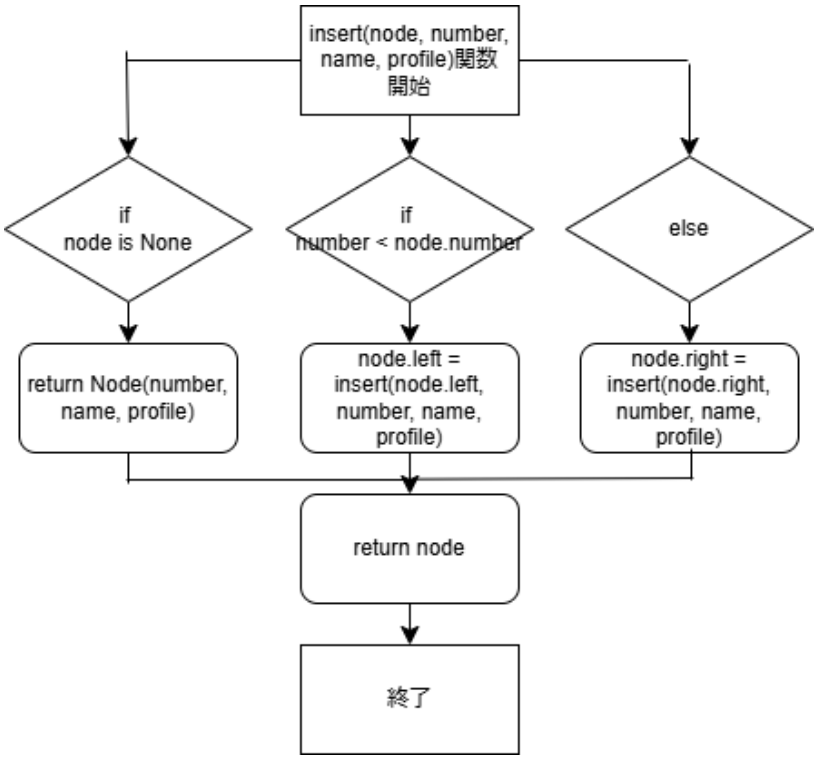
（１）課題番号、課題名：第 14 回基本課題 3　【 木構造を扱うアルゴリズム 】

（２）プログラムのフローチャート

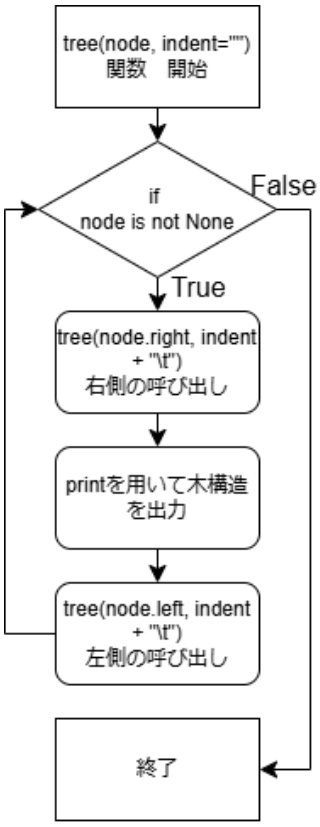
・クラス Node について



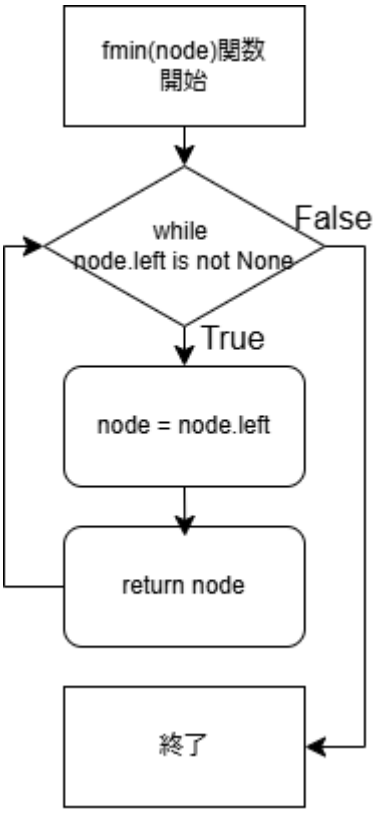
・insert(node, number, name, profile)関数について



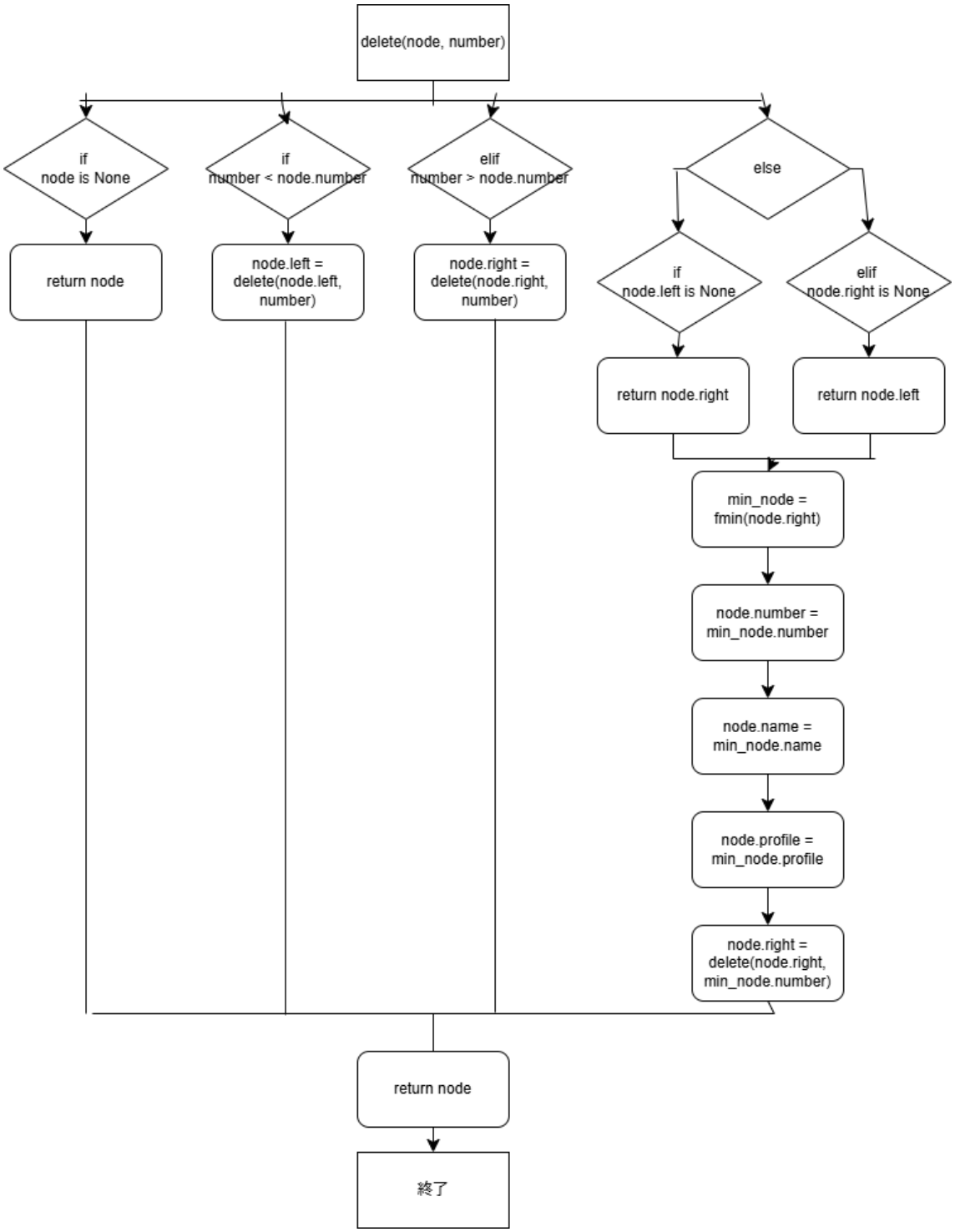
・tree(node, indent="")関数について



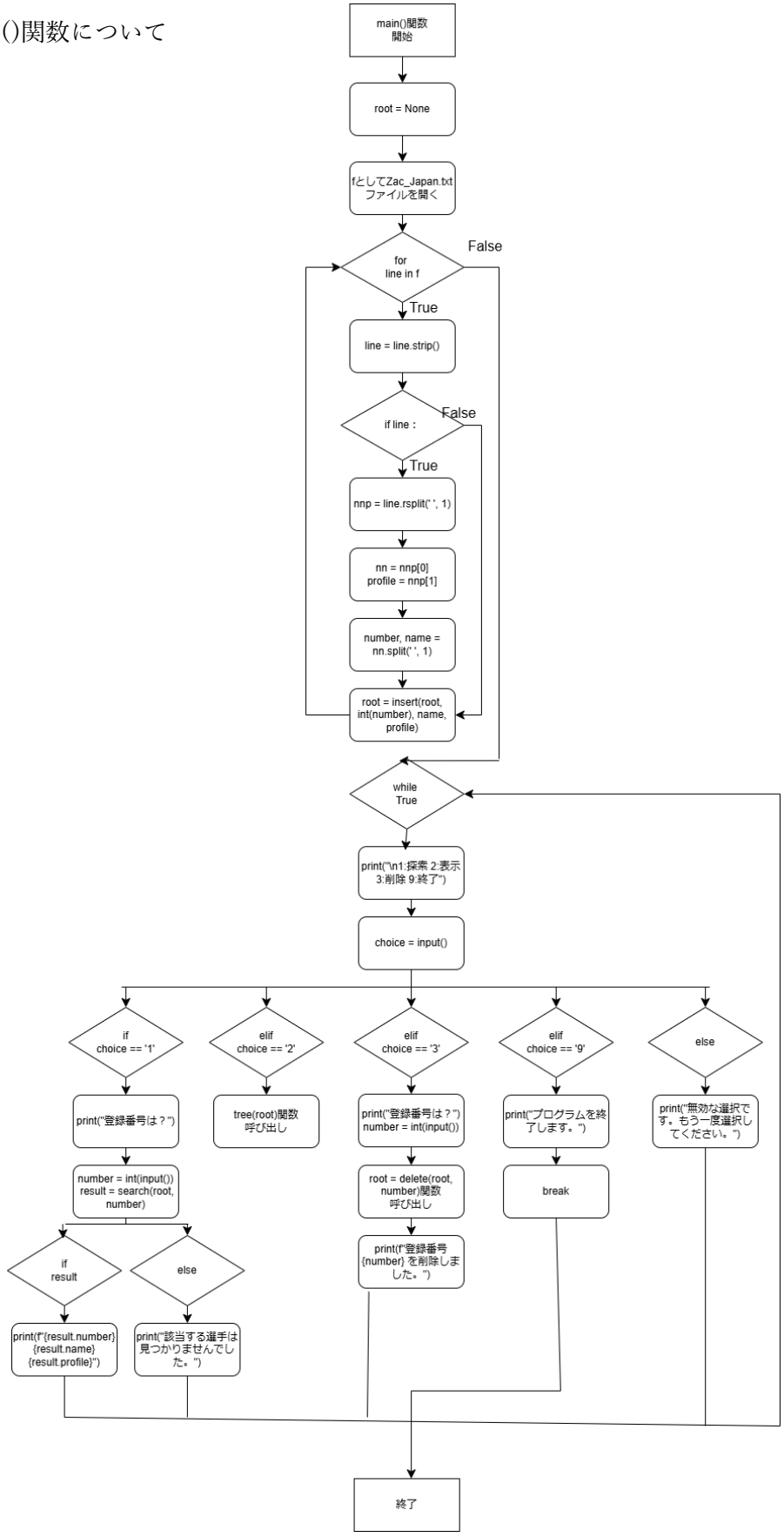
・fmin(node)関数について



・delete(node,number)関数について



・main()関数について


main()関数
開始
root = None
fとしてZac_Japan.txt
ファイルを開く
for
line in f
False
True
line = line.strip()
if line :
False
True
nnp = line.rsplit(' ', 1)
nn = nnp[0]
profile = nnp[1]
number, name =
nn.split(' ', 1)
root = insert(root,
int(number), name,
profile)
while
True
print("\n1:探索 2:表示
3:削除 9:終了")
choice = input()
if
choice == '1'
elif
choice == '2'
elif
choice == '3'
elif
choice == '9'
else
print("登録番号は？")
tree(root)関数
呼び出し
print("登録番号は？")
number = int(input())
print("プログラムを終
了します。")
print("無効な選択です。もう一度選択してください。")
number = int(input())
result = search(root,
number)
root = delete(root,
number)関数
呼び出し
break
if
result
else
print(f"登録番号
{number} を削除しました。")
print(f"{result.number}
{result.name}
{result.profile}")
print("該当する選手は
見つかりませんでした。")
終了

(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、あらかじめファイル名は入力されてあり、そのファイル Zac_Japan.txt に対する木構造を表示しながら項目を削除できるという作りである。プログラムを実行すると 1:探索 2:表示 3:削除 9:終了の選択肢が表示され、適切な番号を選択することでプログラムが実行される。ここで、全ての入力に対してプログラムは停止するということとすべての入力に対して「正しい答え」を出力することについて説明する。初めにここで 1（探索）を選択した場合は以下のように登録番号を入力し、該当する選手の情報を出力する。また、入力した登録番号に該当する選手がいない時はもう一度入力を促される。すなわち停止性と正しい結果の出力ができていると言える。

次に 2(表示)を選択した場合は以下のように現在存在する選手の木構造を出力する。

3（削除）を選択すると以下のように入力した登録番号に該当する選手が削除される。

最後に 9 終了を入力すると、ループから抜けて正しい答えを出力する。以上より、1,2,3,9 のいずれを入力した際もアルゴリズムは停止性かつ正しい答えを出力する。1,2,3,9 以外を入力したときは以下のように 0、負の数、文字列などいずれの場合にもエラーメッセージを出力し、次の入力を求める。

以上より、採用したアルゴリズムは全ての入力に対して停止し、「正しい答え」を出力する。

（４）ソース・プログラムの説明

```
class Node:                                                    #クラス Node の定義
    def __init__(self, number, name, profile, left=None, right=None): #変数の定義
        self.number = number                                              #変数 number 定義
        self.name = name                                                  #変数 name 定義
        self.profile = profile                                            #変数 profile 定義
        self.left = left                                                  #変数 left 定義
        self.right = right                                                #変数 right 定義

def insert(node, number, name, profile):                        #insert 関数定義
    if node is None:                                           #ノードが空の時、
        return Node(number, name, profile)                      #新しいノードを作成し返す
    if number < node.number:                                   #挿入番号が今のノードより小さい時
        node.left = insert(node.left, number, name, profile) #左側の木構造に挿入
    else:                                                      #その他の時
        node.right = insert(node.right, number, name, profile)#右側の木構造に挿入
    return node                                                #ノードに値を返す

def tree(node, indent=""):                                  #木構造を表示する関数
    if node is not None:                                    #ノードが空ではない時
        tree(node.right, indent + "¥t")                      #右側の木構造を出力
        print(indent + str(node.number) + " " + node.name + " " + node.profile)
        tree(node.left, indent + "¥t")                       #左側の木構造を出力

def search(node, number):                                   #ノードを探索する search 関数について
    if node is None or node.number == number:  #ノードが空または指定した番号が存在する時
        return node                                         #ノードに値を返す
    if number < node.number:                                #番号が今のノードより小さい時
        return search(node.left, number)                     #左側について探索関数を呼び出し
    return search(node.right, number)                        #右側について探索関数を呼び出し

def fmin(node):                                #最も左のノードを探索する関数
    while node.left is not None: #左のノードが空になるまで
        node = node.left                       #ノードを左に移す
    return node                                #ノードに値を返す
```

```
def delete(node, number):                          #ノードを削除する関数
    if node is None:                               #ノードが空である時
        return node                                #ノードに値を返す
    if number < node.number:                       #今のノードが番号よりも大きい時
        node.left = delete(node.left, number)  #左側のノードを削除
    elif number > node.number:                     #番号が今のノードよりも大きい時
        node.right = delete(node.right, number)#右側のノードを削除
    else:                                          #それ以外の時
        if node.left is None:                      #左側のノードが空の時
            return node.right                      #右側のノードに値を返す
        elif node.right is None:                   #右側のノードが空の時
            return node.left                       #左側のノードに値を返す

        min_node = fmin(node.right)                #最も左側のノードの呼び出し
        node.number = min_node.number              #番号を最も左側に設定
        node.name = min_node.name                  #名前を最も左側に設定
        node.profile = min_node.profile            #プロフィールを最も左側に設定
        node.right = delete(node.right, min_node.number) #右側の木構造を削除する

    return node                                    #ノードに値を返す

def main():                                        #main()関数
    root = None                                    #初期ノードは空
    with open('Zac_Japan.txt', 'r', encoding='utf-8') as f: #読み取り専用でファイルを開く
        for line in f:                                      #ファイルの一行ずつに関して
            line = line.strip()                             #行の空白を削除
            if line:                                        #行が空では無い時
                nnp = line.rsplit(' ', 1)                   #名前とプロフィールに分割
                nn = nnp[0]                                 #名前を取得
                profile = nnp[1]                            #プロフィールを取得
                number, name = nn.split(' ', 1)             #番号と名前の取得
                root = insert(root, int(number), name, profile) #変数 root にデータを格納

    while True:                                    #break するまで繰り返す
        print("¥n1:探索 2:表示 3:削除 9:終了") #選択肢の出力
        choice = input()                           #選んだ番号の格納
```

```
if choice == '1':                    # 1、すなわち探索を選んだ時
    print("登録番号は？")              #登録番号を聞く
    number = int(input())            #登録番号を入力
    result = search(root, number)     #search 関数を呼び出す
    if result:                       #結果が空では無い時
        print(f"{result.number} {result.name} {result.profile}") #該当するデータ
                                                                  の出力
    else:                                                        #結果が見つからない時
        print("該当する選手は見つかりませんでした。")                  #その旨を出力
elif choice == '2':                                              #2、すなわち表示を選んだ時
    tree(root)                                                   #tree 関数を呼び出し
elif choice == '3':                                              #3、すなわち削除を選択した時
    print("登録番号は？")                                          #登録番号を聞く
    number = int(input())                                        #番号を入力
    root = delete(root, number)                                  #delete 関数の出力
    print(f"登録番号 {number} を削除しました。")                    #削除したとの旨を出力
elif choice == '9':                                              #9、すなわち終了を選んだ時
    print("プログラムを終了します。")                               #終了を知らせる旨を出力
    break                                                        #ループ終了
else:                                                            #番号が不適切であった時
    print("無効な選択です。もう一度選択してください。")#エラーメッセージを出力
```

（５）考察

本プログラムでは存在するサッカー選手のテキストデータについて探索、表示、削除、終了を選択することでそれに応じた結果を出力するということを実装した。初めに本プログラムの効率について考察する。プログラムを実行した際、プログラム実行中に待機することはなく、一瞬で結果の出力が行われた。これは、本プログラムでは本プログラムでは先週および先々週で考察した結果を活かして比較的効率の良いとされる２分探索を用いたためである。また、データ量も多くなかったため、２分探索の最小計算量 O(logn)で計算できたと考えられるためである。また、関数を多く用いることで複雑ではなく、簡潔なアルゴリズムを採用したことで実行時間の短縮につながったと考えた。具体的には再帰関数を適切に用いたことがこれに当たる。

また、アルゴリズムを考える際に、与えられた選択肢以外または登録されている番号以外を用いたときにはエラーメッセージを出力し再び選択肢や番号を求めるというアルゴリズ

ムを採用した。これにより数字以外の入力を行ってもエラーとなることなくプログラムを実行できた。すなわち、汎用性の高いプログラムであると言える。更に簡潔なプログラムであるため、選択肢を容易に増やすことができ、柔軟性にも優れたアルゴリズムであると考えた。

最後に本プログラムで改善の余地があるかについて考察する。アルゴリズムの説明の項でも説明したようにどのような入力を行ってもエラーが表示されることなくプログラムを実行できるため、入力に関しては問題ない。しかしながら、テキストファイルが別のものとなった際に中身の様式が異なるとエラーまたは正しく木構造が出力されない可能性がある。あくまで本プログラムはテキストファイル Zac_Japan.txt のみを対象としたものと考えれば採用したアルゴリズムは改善の余地がないと言えると考えた。異なるテキストデータを用いる時には適宜修正を行えば良い。

（６）感想

木構造を自分で考える時間よりもプログラムが出力してくれる時間の方が圧倒的に速くてプログラムを書く意義を感じた。やはりコンピュータの計算の速さはすごいなと感心した。と同時に先週の講義を活かし 2 分探索が実装できるようになって良かった。夏休みにも忘れないように時々コードを書こうと思えるくらい面白かった。