

優しいアルゴリズム入門

～Tech.C 校内 Version～

稲山 舞衣子

前置き

私は、1年生の頃にプログラムの技術を磨くために競技プログラミングに参加しました。

しかし、競技プログラミングでより難しい問題を解くには、どのプログラミング言語でもアルゴリズムを勉強する必要があります。

アルゴリズムを学べる本は世の中にはたくさんあります。しかし、本自体も難しい内容だったり、プログラムが書いてない本が多く、アルゴリズム初心者、中級者にも理解しづらい内容になっております。

アルゴリズムを勉強する為に本の購入を迷っている方、ゴリゴリのアルゴリズム本の前に、この冊子で簡単なアルゴリズムを1つでも知るだけで本が読みやすくなります。

是非、この冊子をご活用して欲しいと思います。

制作時間が短い為、ページ数が少ないです。

今年中には、ページを増やした続編を作成、Web サイトを作成するつもりです。

よろしくお願いします。

稲山 舞衣子

目次

1. これで指を温めよう！「FizzBuzz」	3
2. これが出来るとプログラム中級者の姿が見える.. 素数判定	5
3. 名前は難しいそうだが、難しくないバブルソート	7
4. 基本情報試験に出るヒープソート	9
5. 使えるようになるとかっこいい！ Bit 全探索.....	13
6. 覚えておくと、便利！「ダイクストラ法」	16
7. ダイクストラ法と一緒に覚える「ベルマンフォード法」	19

1. これで指を温めよう！「FizzBuzz」

問題：

自然数 N を入力する。

1～ N までの数値で、3 で割り切れるものを **Fizz**、5 で割り切れるものを **Buzz**、両方で割り切れるものを **FizzBuzz** 、その他を**その数値自身**が出力されるようにしなさい

条件：

$1 \leq N \leq 100$

例：

入力	出力
16	1
	2
	Fizz
	4
	Buzz
	Fizz
	7
	8
	Fizz
	Buzz
	11
	Fizz
	13
	14
	FizzBuzz
	16



ヒント

プログラムを書く前に、場合分けを試みよう！

場合分けできたら、if 文で書いてみよう！

回答：

Python

```
N = int(input())
for i in range(1,N+1):
    if(i % 3 == 0):
        if(i % 5 == 0):
            print("FizzBuzz")
        else:
            print("Fizz")
    elif(i % 5 == 0):
        print("Buzz")
    else:
        print(i)
```

まとめ

プログラムを書く前に、考えをまとめる

場合分けに困ったら、出題された数値以外（出力の文字列、出力された数値の関連性など）を見てみよう！

Point:

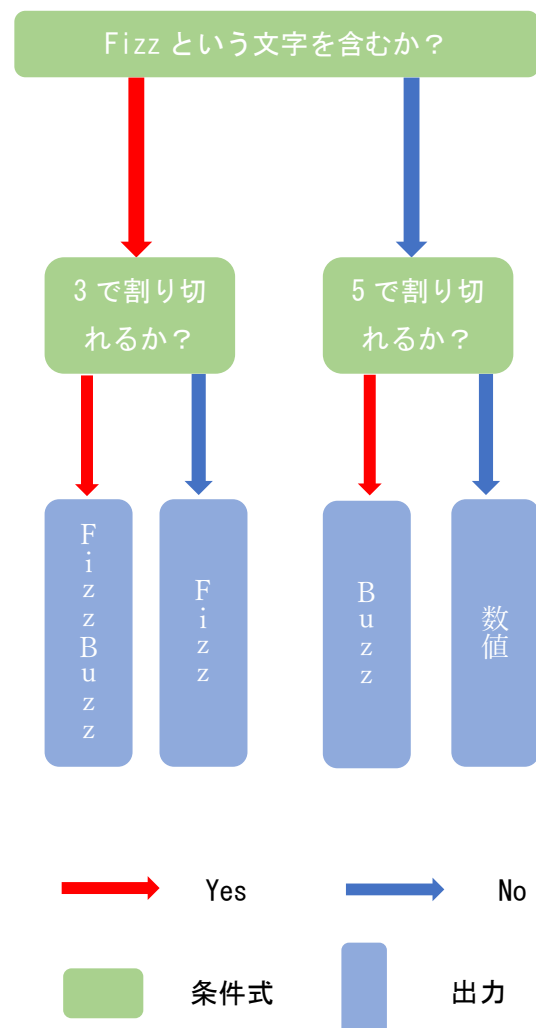
- Python では、文字の入力の受け取りは input 関数を使用する

input 関数で受け取った値の型は、必ず str 型である

その為、数値を受け取った場合は、

int 型をキャスト（強制型変換）しなければならない

- この問題では、出力を以下のように分類すると素早く if else 文を作成することができる



2. これが出来るとプログラム中級者の姿が見える.. 素数判定

問題：

自然数 N を入力する

1~ N までの数値で、素数であるものを出力しなさい

条件：

$1 \leq N \leq 100$

例：

入力	出力
12	2 3 5 7 11



ヒント

関数を使用して、コンパクトに書こう！

素数とは？

→ 1 とその数自身しか割り切れない数

回答：

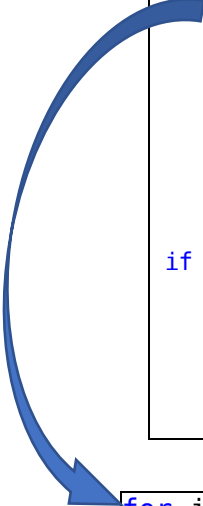
Python

```
import math

def check_primary(num):
    for i in range(2,num):
        if num % i == 0:
            return 0
    return 1

def main():
    N = int(input())
    for i in range(2,N+1):
        if check_primary(i) == 1:
            print(i)

if __name__ == "__main__":
    main()
```



Point:

check_primary 関数は素数判断する関数である

素数だと 1、素数以外だと 0 を返す

main 関数の中の for の range(2, N+1) は
range(2, int(sqrt(i))+1)に変更できる

なぜか？

任意自然数 i 、 i 以下の自然数 a, b を仮定する

$$i = a * b$$

と表すことができ、

$$i = \text{sqrt}(i) * \text{sqrt}(i)$$

と表すことができる

$$a * b = \text{sqrt}(i) * \text{sqrt}(i)$$

であり、

a, b は以下の 3 つの条件のどれか 1 つにあてはまる

1. $a \leq \text{sqrt}(i)$ and $b > \text{sqrt}(i)$
2. $\text{sqrt}(i) = a$ and $\text{sqrt}(i) = b$
3. $b \leq \text{sqrt}(i)$ and $a > \text{sqrt}(i)$

以上から

$$\min(a, b) \leq \text{sqrt}(i)$$

という条件式を作ることができる

i が 2 から $\text{sqrt}(i)$ までの数値を 1 つ以上約数に持たないと、 i は素数だと言える

3. 名前は難しいそうだが、難しくないバブルソート

隣り合うふたつの要素の値を比較して、降順、昇順にするために交換を行うアルゴリズムである

本冊子では降順を紹介している

簡単なアルゴリズムだが、この次の章で紹介するヒープソート の為の準備体操だと思っていただきたい

問題：

以下のような list 型変数 data ある。

あなたは、バブルソートを用いて、data の中身を降順にしたものを表示しなさい

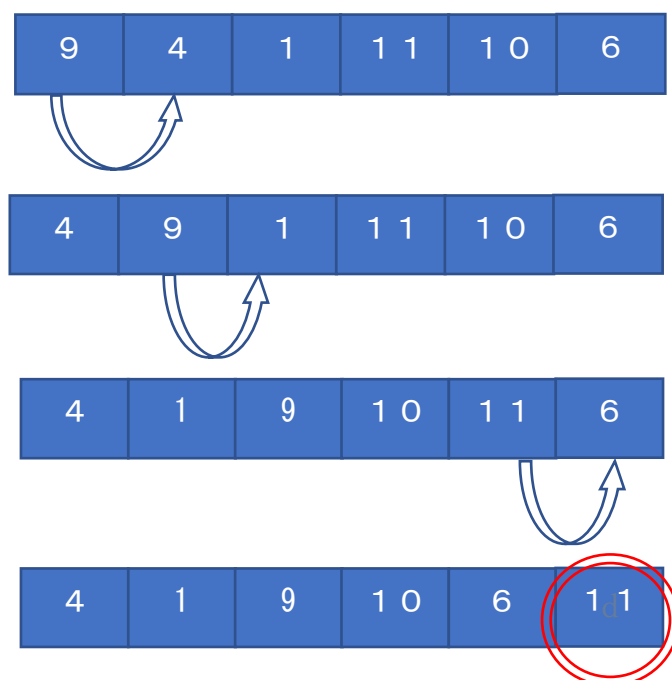
最も運賃が安くなる乗り方をすると、運賃の合計はいくらになるでしょうか？

例 1 回目のバブルソートの動作を表す図：

1 回目のバブルソートで、配列の後ろの値（**最大値**）がソート済みとなる

回数を重ねるごとに、配列の後ろの値から順番にソート済みとなる

これが、配列の一番最初の値（最小値）が決まるまで繰り返す



回答：

Python

```
data = [9,4,1,11,10,6]

for i in range(len(data)-1):
    for j in range(len(data)-1-i):
        if data[j] > data[j+1]:
            data[j],data[j+1] = data[j+1],data[j]
print(data)
```


Point:

list 型変数 data にソート前の値が格納されている

2 個の for 文のうち、外側の for 文は前のページで述べたソートの回数 (バブルソートが最後の値まで処理し終わったら、また配列の最初から (バブルソートを繰り返す) を表す
中側の for 文は、完全にバブルソートが済んだ data の値以外にバブルソートを繰り返す
為である

前のページで述べたが、バブルソートは配列の後ろからソートが完全に完了する

(降順、昇順の両方でも同じである)



まとめ

無駄 for 文、while 文を削ろう！

無駄な for 文、while 文はメモリを消費し、プログラムの処理速度を低下させる

すぐに、スマートなプログラムを書くことは不可能である

プログラムを書く前に紙に設計を書き、プログラムを書いた後も見直すことをオ
ススメする

4. 基本情報試験に出るヒープソート

ヒープソート とは、二分ヒープを用いてソートを行うアルゴリズム

二分ヒープとは？

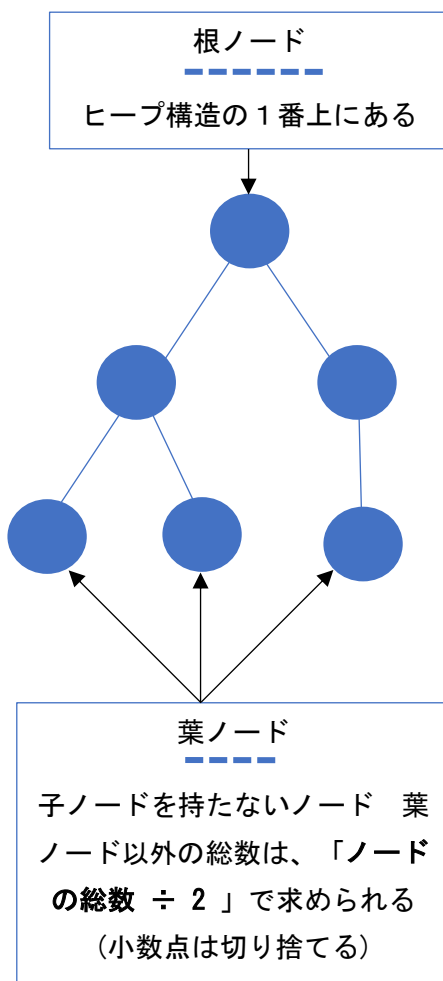
二分木を用いたヒープ構造体

二分木とは？

木構造の一種で、あるノードが持つ子ノードの数が最大で2、最小で0個持つものである

(図 1 参照)

[図 1]



ヒープ構造体とは？

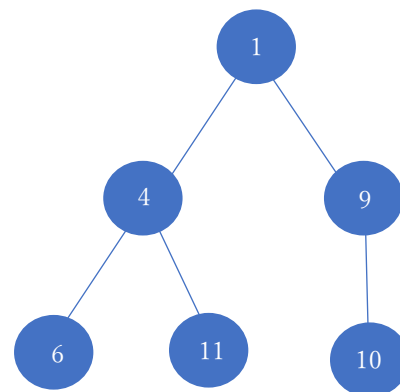
各々のノードはその子ノードよりも小さい又は等しくなるように配置される

この方法を**最小ヒープ**という

各々のノードをその子ノードより大きい又は等しくなるように配置する方法を**最大ヒープ**という

本冊子では、最小ヒープを取り上げている (図 2 参照)

[図 2]



ヒープソート の考え

- 1.ソート後の値を入れるための配列を作成する
- 2.ソートされる前のデータを2分木ヒープ構造にする
3. 根ノードが最小値になるように並び替えを行う
(最大ヒープは、最大値にする)
- 4.並び替えが完了した根ノードの値を1で作成した配列に入れる。
5. 1～4を繰り返して、1で作成した配列にソートされる前の配列と同じ個数入っていれば、完了

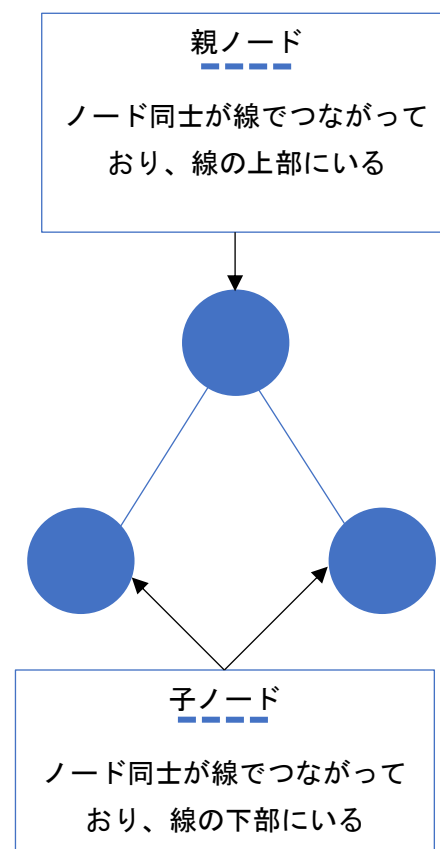
問題：

以下のような list 型変数 data がある

あなたは、ヒープソート を用いて、data の中身を降順にしたものを表示しなさい

この問題は、ヒープソート の考え方を知らないとは解くことができない

次ページでコードを説明するので、その前に前のページで紹介しきれなかったノードの説明をする



回答：

Python

```
data = [1,4,9,6,11,10]

def my_heap(data,parent_index):

    data_min = parent_index

    data_left = i*2 +1

    data_right = i* 2+2

    data_size = len(data) - 1

    if data_left <= data_size and data[data_min] > data[data_left]:

        data_min = data_left

    if data_right <= data_size and data[data_min] > data[data_right]:

        data_min = data_right

    if data_min != parent_index:

        data[parent_index],data[data_min] = data[data_min],data[parent_index]

        my_heap(data,data_min)

for i in range((len(data)//2),0,-1):

    my_heap(data,i)

heaped_list = []

for i in range(len(data)):

    data[0],data[-1] = data[-1],data[0]

    heaped_list.append(data.pop())

    my_heap(data,0)

print(heaped_list)
```

Point:

変数 data ... ソートする前の配列

関数 my_heap

注意

この関数内で一番小さいノード とは、
親ノード、左側の子ノード、右側の子ノードの中で一番数値が小さいものである

list 型の変数 data と int 型の変数 i を受け取り、
それらを利用して、以下の変数を関数内で作成する

int 型変数 data_min ... 初期値として親ノードのインデックス、一番数値が小さいノード
のインデックスを表す

int 型変数 data_left ... 左側の子ノードのインデックスを表す

int 型変数 data_right ... 右側の子ノードのインデックスを表す

int 型変数 data_size ... data のインデックスの最大値を表す
(data_size は、2ページ前の「ヒープソート の考え4」にあるように data の値を捨てて、
ソート後の値を格納する配列に値を入れる為、data の長さは毎回変化する)

int 型変数 data_right ... 右側の子ノードを表す

左側の子ノードのインデックスが data のインデックスの最大値より小さく、且一番値が
小さいノードの数値が左側の子ノードより大きい時、
一番値が小さいノードのインデックスを左側の子ノードに変更する

右側の子ノードのインデックスが data のインデックスの最大値より小さく、且一番値が
小さいノードの数値が左側の子ノードより大きい時、
一番値が小さいノードのインデックスを右側の子ノードに変更する

親ノードのインデックスが初期値ではないとき(変更されていた場合)
親ノードと一番値が小さいノードの値を変更し、my_heap 関数を繰り返す

1つ目の for 文は、葉ノード以外を my_heap 関数を使用して、並び替える処理を行っている
なぜ、葉ノード以外並び替えるのか？

葉ノードは子ノードを持たない つまり、葉ノードの親ノードで my_heap 関数を使用し並び替
えすることで、葉ノードも並び替えられるからである

ソート後の値を入れるための list 型変数 heaped_list を入れる

2つ目の for 文は、
list 型変数 data の最初の数値と最後の数値を入れ替える
(最小ヒープでは、根ノードが最小値になる)
data の最小値を data から無くして、heaped_list に入れる
data の要素が無くなるまで my_heap 関数でまた並び替える

5. 使えるようになるとかっこいい！ Bit 全探索

問題：

長さが任意の数列 N があります。

数列の中の数値を足し合わせて、数値 M と同じになる数値の組み合わせはあるでしょうか？

(問題には出題してないが、その数値自身だけでも良い)

ある場合は、**個数**を入力してください。ない場合は、**0**を出力してください。

条件：

$2 \leq N.length \leq 10$

$2 \leq M \leq 50$

$1 \leq N_i \leq 50$

$N.length$... 数列 N の長さ

N_i ... 数列 N の i 番目の数値

入力の例

M

$N_1 N_2 N_3 \dots N_{N.length-1}$

入力①	出力①
73	3
1 4 8 16 21 25 27	
入力②	出力②
47	4
16 2 4 24 17 8 19	

回答：

Python

```
ans_num = int(input())
nums_list = list(map(int,input().split()))

cnt = 0

for i in range(2**len(nums_list)):
    nums_sum = 0
    for j in range(len(nums_list)):
        if((i>>j&1) == 1):
            nums_sum += nums_list[j]
    if nums_sum == ans_num:
        cnt+= 1
print(cnt)
```

Point:

```
list(map(int,input().split()))
```

→ 1 行に複数の数値を list 型の変数として受け取ることができる

(数値と数値の間に空白を開けることに注意！)

```
for i in range(2**len(nums_list)):
```

```
    for j in range(len(nums_list)):
```

```
        if((i>>j&1) == 1):
```

で、数列 N(集合と数列の違いは数値の順番が決まっているかである)の部分集合を列挙できる

例：

N = [1, 3, 2]

N の部分集合は以下になる

[], [1], [3], [1, 3], [2], [1, 2], [3, 2], [1, 3, 2]

Bit 全搜索のプログラム上は、以上の部分集合を以下のように扱っている

[[0, 0, 0], [1, 0, 0], [0, 1, 0], [1, 1, 0], [0, 0, 1], [1, 0, 1], [0, 1, 1], [1, 1, 1]]

`for i in range(2**len(nums_list)):` ... 以下のように考えると理解し易い

数列 N のすべての要素についての部分集合に〈含む(1)／含まない(0)〉の 2 つの可能性の組み合わせとして定義している

`for j in range(len(nums_list))` ... 部分集合の中の 1 つの集合の数値にアクセスしようとしている

`i>>j` ... i の j ビットを右シフトする

`x&1` ... 任意の数 x と 1 の論理積

例：

x を 7 とする

7 を 2 進数にすると 0b111、6 を 2 進数にすると 0b000 なる

(先頭の 0b は 2 進数であることを表す)

それぞれを 1 と論理積する (`x&1` x に 6、7 があてはまる) と、以下ようになる

0b111	[x = 6 の場合]	0b110
&) 0b001		&) 0b001
-----		-----
0b001		0b000

`x&1` の答えは、1 か 0 のどちらかになる

その為、

`if((i>>j&1) == 1)` ... 数列の中の数値が部分集合に含まれているかを判断している

(含まれていたら 1、含まれていなかったら 0 を返す)

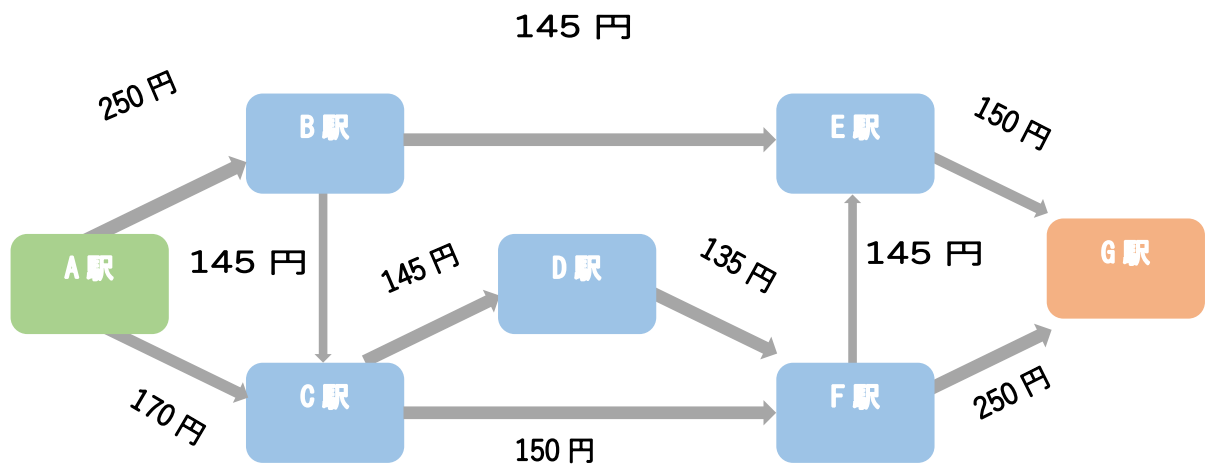
6. 覚えておくと、便利! 「ダイクストラ法」

問題：

以下の路線図がある。

あなたは、A 駅にいて、G 駅に行きたいです。

最も運賃が安くなる乗り方をすると、運賃の合計はいくらになるでしょうか？



また、上の問題のように、経路とコストが提示され、最短経路を求める問題を「**最短経路問題**」という

ダイクストラ法は構造を理解するまでが大変なので、まず回答プログラムを見て抽象的にダイクストラ法を理解することがまず大切である

回答：

Python

```
stations = [
    [[1,250],[2,170]],
    [[2,145],[4,145]],
    [[3,145],[5,150]],
    [[5,135]],
    [[6,150]],
    [[4,145],[6,250]],
    []
]

fare = [float("inf")] * 7
fare[0] = 0
station_num = [i for i in range(7)]
while len(station_num) > 0:
    num_min = station_num[0]
    for i in station_num:
        if fare[i] < fare[num_min]:
            num_min = i
    minnum = station_num.pop(station_num.index(num_min))
    for i in stations[minnum]:
        if fare[i[0]] > fare[minnum] + i[1]:
            fare[i[0]] = fare[minnum] + i[1]
print(fare[-1])
```

Point:

ここでは、各頂点を各駅、コストを運賃とする

以下のような 2 次元配列 stations を作成する

```
stations = [  
    [[i 番目の頂点と繋がっている頂点の index],[繋がっている頂点までのコスト]]  
    .....  
]
```

各頂点までのコストを ∞ (無限)とした list 型変数 fare を作成する

fare の 1 番最初の値(最初の頂点のコスト)を 0 にする

頂点のインデックスを要素とする list 型変数 station_num を作成する

while 文の中身

station_num の一番初めの数値を変数 num_min とおく

1 番目の for 文の中身

num_min を index とする配列 fare の数値より、小さい値が fare の中にあるか
探索する

小さい値があったら、num_min をその数値に置き換える

index が num_min であるの数値を station_num から捨て、変数 minnum とする

2 番目の for 文の中身

i を index にもつ駅までの最短距離(コスト)を更新できるかを確認する

更新できたら、その駅を index にもつ fare の値をそのコストに更新する

while 文は、station_num の要素がなくなるまで続ける

7. ダイクストラ法と一緒に覚える「ベルマンフォード法」

「ベルマンフォード法」は一つ前に紹介した「ダイクストラ法」と同じく、「最短経路問題」に使用できます。

ダイクストラ法との違いは、コストに負の数を扱えることである

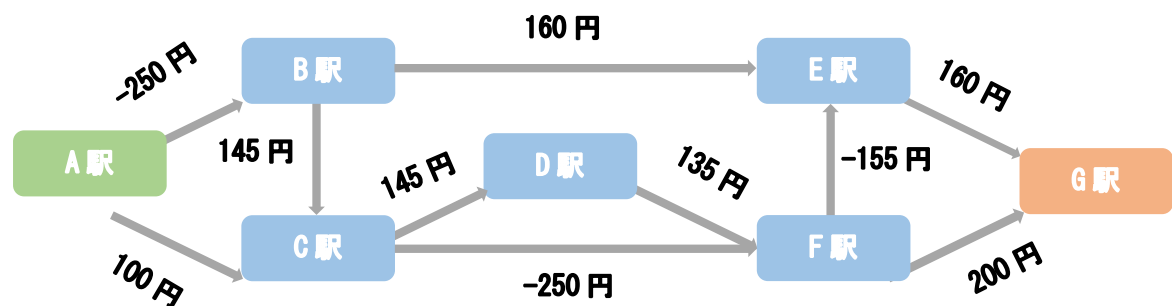
問題：

あなたは、某乗り物をしようして、所持金を上げていくゲームをしています。

今、あなたは所持金 1000 円で A 駅にいて、G 駅に行きたいです。

最も所持金が減らない乗り方をすると、所持金はいくらになるでしょうか？

注意：-〇〇〇円は、〇〇〇円もらえるということです。



回答：

Python

```
stations = [ [0,1,-250],[0,2,100],[1,2,145],[1,4,160],[2,3,145],[2,5,-250],[3,5,135],[4,6,160],[5,4,-155],[5,6,200]]
my_money = 1000

fare = [float("inf")] * 7
fare[0] = 0
flag = True
while flag:
    flag = False
    for i in stations:
        if fare[i[1]] > fare[i[0]] + i[2]:
```

```
        fare[i[1]] = fare[i[0]] + i[2]

        flag = True

print(1000-fare[-1])
```

Point:

ここでは、各頂点を各駅、コストを運賃とする

以下のような 2 次元配列 stations を作成する

```
stations = [
    [index, 繋がっている頂点の index、繋がっている頂点までのコスト]
    .....
]
```

各頂点までのコストを ∞ (無限)とした list 型変数 fare を作成する

fare の 1 番最初の値(最初の頂点のコスト)を 0 にする

boolean 型の変数 flag の値を True にする

while 文の中身

 flag を False にする 1 番目の for 文の中身

 for 文の中身

 i を index にもつ駅までの最短距離(コスト)を更新できるかを確認する

 更新できたら、その駅を index にもつ fare の値をそのコストに更新する

 更新できなくなったら、while 文を抜ける