

情報処理技法 (Javaプログラミング)2

第4回 アルゴリズム(2)

人間科学科コミュニケーション専攻
白銀 純子

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

第4回の内容

- アルゴリズム(続き)

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

前回の出席課題の解答

- このプログラムは、自分の名前と相手の名前を入力し、挨拶を出力するプログラムです。(処理)の部分にどのような処理が入るか、下記の選択肢から最も適切なものを選びなさい。

プログラム

```
public static void sayHello(String you, String me) {  
    (a)  
}  
public static void main(String[] args) {  
    String you, me;  
    try {  
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
        // 自分の名前を入力  
        System.out.println("あなたの名前を入力してください");  
        me = br.readLine();  
        // 挨拶の相手の名前の入力  
        System.out.println("挨拶をする人の名前を入力してください");  
        you = br.readLine();  
        // 挨拶処理  
        sayHello(you, me);  
    } catch (IOException e) {  
        System.out.println("標準入力できませんでした");  
    }  
}
```

選択肢

```
(a)  
String message;  
message = "Hello, " + you + "! My name is " + me + "!";  
return message;  
(b)  
System.out.println("Hello, " + you + "!");  
System.out.println("My name is " + me + "!");  
(c)  
String message;  
message = "Hello, " + you + "! My name is " + me + "!";  
(d)  
String message;  
message = "Hello, " + you + "! My name is " + me + "!";  
return message;  
System.out.println(message);
```

解答: b

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

前回の復習

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

アルゴリズムとは

- アルゴリズム: ある問題を解決するときに必要な処理手順
 - プログラムを書くときには、必ずアルゴリズムを考える必要
 - プログラム: アルゴリズムをプログラミング言語を使って記述したもの

Ex. 焼きそば作りのアルゴリズム:

- キャベツや肉などの具をフライパンで炒めなさい。
- フライパンからいったん具を取り出さない。
- めんをフライパンで炒めなさい。
- 具をめんの入っているフライパンに戻しなさい。
- 焼きそばソースを加えてさらに炒めなさい。

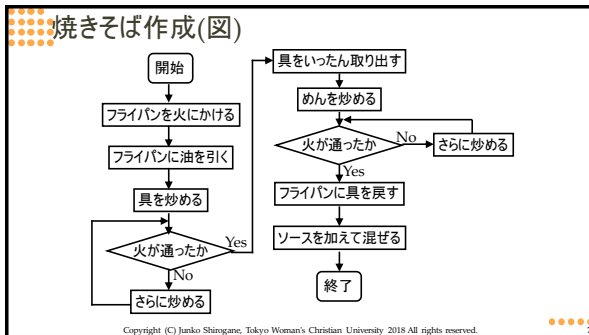
Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

アルゴリズムの表現方法

- 文章で書く
 - 箇条書きで書くことも多い
- 図で書く
- プログラムで書く
 - プログラムが最終段階

プログラムで行う処理手順がわからなければ、まず文章で書き出して、それを詳細にしてい

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.



焼きそば作成(プログラム1)

焼きそばのアルゴリズムをプログラムの表現

```

public static void main(String[] args){
    // 開始
    フライパンを火にかける;
    フライパンに油を引く;
    具を炒める;
    while (火が通っていない){
        // Yes(「火が通ったか?」に対して「No」)
        さらに炒める
    }
    // 「火が通ったか?」に対して「Yes」
    具をいったん取り出す;
    めんを炒める;
    具をいったん取り出す;
}
  
```

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

焼きそば作成(プログラム2)

焼きそばのアルゴリズムをプログラムの表現(続き)

```

while (火が通っていない){
    // Yes(「火が通ったか?」に対して「No」)
    さらに炒める;
}
// 「火が通ったか?」に対して「Yes」
フライパンに具を戻す;
ソースを加えて混ぜる;
// 終了
}
  
```

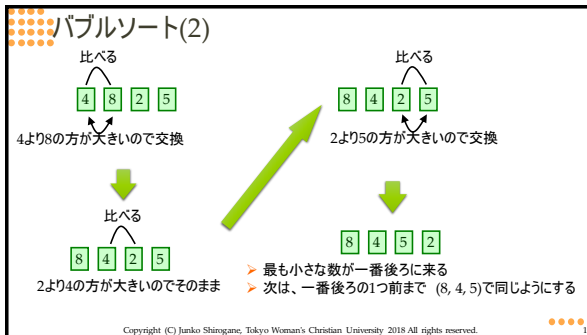
Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

有名なアルゴリズム

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

- ### 並べ替え(ソート)
- ソート: 複数の数を小さい or 大きい順に並べること
 - バブルソート
 - 選択ソート
 - 併合ソート
 - クイックソート
 - etc.
- Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

- ### バブルソート(1)
- 前から2つずつ、数の大きさを比較して、小さい数を後ろに送っていく
 - 最後まで調べると、最も小さな数が一番後ろにある
 - この作業を、並べ替える数の個数だけ繰り返すと、数が大きい順に並ぶ
 - $x_1 < x_2$ ならば、 x_1 と x_2 を入れ替える
 - $x_2 < x_3$ ならば、 x_2 と x_3 を入れ替える
 - ...
 - $x_{n-1} < x_n$ ならば、 x_{n-1} と x_n を入れ替える
 - n 個の数でこの処理が終わった後に n 番目に最も小さな数
 - $n-1$ 回この処理を繰り返すことで、大きい順に数を並べ替え
- Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.



選択ソート(1)

- 変数*t*: 並べ替えをする数の中で最も小さい数を入れておく変数
- 変数*i*: 並べ替えをする数の中で、*t*が最初から何番目の位置にあるかを表す変数
- 変数*j*: 何回繰り返したかを数えるための変数
- 並べ替えをする数は*n*個

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved. 14

選択ソート(2)

- t*に並べ替えをする数の一番最初の数を代入する
- i*に1を代入する
 - 1: 並べ替えをする数の一番最初の数の位置
- j*に2を代入し、1ずつ増やしながら*n*になるまで以下を繰り返す
 - t*が*i*番目の数より大きいならば、*j*番目の数を*t*に代入し、*i*に*j*の値を代入する
 - この処理を1回することにより*i*の値を1増やす
 - j*の値は、現在調べている数の位置になる
- 並べ替えをする数の一番最後の数と*t*を入れ替える

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved. 15

選択ソート(2)

ステップ1:
➢ *t*に4を代入する
➢ *i*に1を代入する

ステップ2:
➢ *j*に2を代入する
➢ *t*の値(4)と8を比べる
➢ *t*の値の方が小さいのでそのまま

ステップ3:
➢ *j*の値を1増やす(3になる)
➢ *t*の値(4)と2を比べる
➢ *t*の値の方が大きいので*t*に2を代入し、*i*に*j*の値(3)を代入する

ステップ4:
➢ *j*の値を1増やす
➢ *t*の値(2)と5を比べる
➢ *t*の値の方が小さいのでそのまま

ステップ5:
➢ *t*の値(2)を最後に置く
➢ これまで最後だった数(5)を*i*番目に入れる

➢ 最小の数が一番後ろに来る
➢ 次は、一番後ろの1つ前まで(8, 4, 5)で同じようにする

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved. 16

アルゴリズムの良し悪し

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved. 17

良いアルゴリズム

- そのアルゴリズムによるプログラムを実行するときの計算時間や記憶領域の使用量の少なさ
- アルゴリズムのわかりやすさ・作りやすさ・修正の容易さ

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved. 18

アルゴリズムの計算時間

- CPUそのものの速さや、コンパイラに大きく依存
- ファイルの入出力時の記憶装置とのアクセスの速度に依存
- メインメモリが少ない場合、メインメモリとHDDの間でデータが頻繁に行き来(スラッシング)

これらを除いても、同じ結果を出す複数のアルゴリズムで、計算時間に違いが出る

➡ アルゴリズムの計算量

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved. 19

アルゴリズムの計算量

- CPUの速さなど、アルゴリズムには関係ない要因を除いた、アルゴリズムそのものの計算時間
 - Ex. for文で繰り返す回数, 足し算・かけ算などの回数, etc.

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved. 20

アルゴリズムのわかりやすさ

- 一旦完成したプログラム: 機能の追加などのためにプログラムの修正が必要なことも多い

アルゴリズムの修正

人がプログラムを読んで理解する

プログラムを読んで理解する = 書かれてあるアルゴリズムの理解が必要

- アルゴリズムが難解 = 修正が難しい
- アルゴリズムが簡単 = 修正が容易

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved. 21

アルゴリズムを使う状況

- 多くの場合、計算時間の速いアルゴリズムとわかりやすいアルゴリズムは対立関係
 - 計算時間が速ければ、わかりにくいアルゴリズム
 - わかりやすければ、計算時間が遅い
 - 速さをとるか、わかりやすさをとるかは、状況に応じて判断
 - めったに使わないプログラムや頻繁に修正するプログラム: わかりやすいアルゴリズム
 - よく使うプログラムや計算時間に制約があるプログラム: 速いアルゴリズム

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved. 22

並べ替えアルゴリズムの比較(1)

- 結果が出るまでの基本処理の回数(アルゴリズムの計算量)
 - バブルソート: $N(N-1)/2$
 - クイックソート: $N\log_2(N)$

※ $\log_2(N)$: N を2^kとしたときの「k」

N	$N(N-1)/2$ (バブル)	$N\log_2(N)$ (クイック)
8	28	24
32	496	160
64	2016	384
128	8128	896

➡ Nが大きければ大きいほど、クイックソートの方が速い

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved. 23

並べ替えアルゴリズムの比較(2)

- 計算量: 入力(N: 並べ替えの場合は数の個数)に対して行われる基本処理の回数
 - Nが十分に大きなとき: 計算式の中の最も大きな項だけに着目して、大まかに計算
 - 各項の比例定数や次数の低い項は無視
 - バブル: $N(N-1)/2 = N^2/2 - N/2$
 - N^2 のみに注目
 - クイック: $N\log_2 N$
 - $N\log_2 N$ のみに注目

アルゴリズムの計算量は、正確な計算量ではなく、Nが大きくなればどの程度の割合で計算量が増えるかを大まかに知ることが重要なため

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved. 24

並べ替えアルゴリズムの比較(3)

- 計算時間の速いアルゴリズム: N や $\log N$ などのみで計算量が計算できるアルゴリズム
- 計算時間の遅いアルゴリズム: N^2 , N^3 , ..., N^k や $N!$ (1から N までをかけあわせた数), 2^N など、多くのかけ算を計算に必要とするアルゴリズム
 - N^2 , N^3 などの計算を必要とするアルゴリズム: 多項式時間アルゴリズム
 - $N!$ や 2^N などの計算を必要とするアルゴリズム: 指数時間アルゴリズム

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

25

準備

- 授業のページから2つのファイルをダウンロード
 - BubbleSort.java
 - バブルソートをするプログラム
 - QuickSort.java
 - クイックソートをするプログラム

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

26

BubbleSort.java(前半1)

```
int num = 10000;
int i;
int[] random = new int[num];
for (i = 0; i < num; i++) {
    random[i] = (int) (num * Math.random());
}
try {
    File file = new File("BubbleSort.txt");
    FileWriter fw = new FileWriter(file);
    PrintWriter pw = new PrintWriter(fw);

    pw.println("並べ替え前");
    for (i = 0; i < num; i++) {
        pw.write(Integer.toString(random[i]) + ", ");
    }
    Calendar cal1 = Calendar.getInstance();
    long beforeTime = cal1.getTimeInMillis();
```

並べ替えをする
数の個数

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

27

BubbleSort.java(前半2)

```
int num = 10000;
int i;
int[] random = new int[num];
for (i = 0; i < num; i++) {
    random[i] = (int) (num * Math.random());
}
try {
    File file = new File("BubbleSort.txt");
    FileWriter fw = new FileWriter(file);
    PrintWriter pw = new PrintWriter(fw);

    pw.println("並べ替え前");
    for (i = 0; i < num; i++) {
        pw.write(Integer.toString(random[i]) + ", ");
    }
    Calendar cal1 = Calendar.getInstance();
    long beforeTime = cal1.getTimeInMillis();
```

並べ替えをする数を
乱数で作成

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

28

BubbleSort.java(前半3)

```
int num = 10000;
int i;
int[] random = new int[num];
for (i = 0; i < num; i++) {
    random[i] = (int) (num * Math.random());
}
try {
    File file = new File("BubbleSort.txt");
    FileWriter fw = new FileWriter(file);
    PrintWriter pw = new PrintWriter(fw);

    pw.println("並べ替え前");
    for (i = 0; i < num; i++) {
        pw.write(Integer.toString(random[i]) + ", ");
    }
    Calendar cal1 = Calendar.getInstance();
    long beforeTime = cal1.getTimeInMillis();
```

並べ替え
数を並べ替える直前の時間を計測

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

29

BubbleSort.java(後半1)

```
for (i = 0; i < num; i++) {
    for (j = 0; j < num - i - 1; j++) {
        if (random[j] > random[j + 1]) {
            temp = random[j + 1];
            random[j + 1] = random[j];
            random[j] = temp;
        }
    }
    Calendar cal2 = Calendar.getInstance();
    long afterTime = cal2.getTimeInMillis();
    System.out.println("おかしな時間(バブルソート): " + (afterTime - beforeTime) + "ミリ秒");

    pw.println("バブルソート後の数");
    for (i = 0; i < num; i++) {
        pw.write(Integer.toString(random[i]) + ", ");
    }
    fw.close();
    pw.close();
} catch (IOException e) {}
```

バブルソートをする処理部分

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

30

BubbleSort.java(後半2)

```
for (i = 0; i < num; i++) {
    for (j = 0; j < num - i - 1; j++) {
        if (random[j] > random[j + 1]) {
            temp = random[j + 1];
            random[j + 1] = random[j];
            random[j] = temp;
        }
    }
}

Calendar cal2 = Calendar.getInstance();
long afterTime = cal2.getTimeInMillis();
System.out.println("かかった時間(バブルソート): " + (afterTime - beforeTime) + "ミリ秒");

pw.println("バブルソート並べ替え後");
for (i = 0; i < num; i++) {
    pw.write(Integer.toString(random[i]) + " ");
}
fw.close();
pw.close();
} catch (IOException e) {} }
```

数を並べ替えた直後の時間を計測

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved.

31

QuickSort.java(前編)

```
public static void sort(int[] rand, int first, int last) {
    int i, j, temp, x = rand[(first + last) / 2];

    i = first;
    j = last;
    while (true) {
        while (rand[i] < x) {
            i = i + 1;
        }
        while (x < rand[j]) {
            j = j - 1;
        }
        if (i >= j) {
            break;
        }
        temp = rand[i];
        rand[i] = rand[j];
        rand[j] = temp;
        i = i + 1;
        j = j - 1;
    }
}
```

クイックソートの
処理部分
(メソッドとして定義)

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved.

32

QuickSort.java(中編1)

```
if (first < i - 1) { sort(rand, first, i - 1); }
if (j + 1 < last) { sort(rand, j + 1, last); }
}

public static void main(String[] args) {
    int i, j, temp, num = 10000;
    int[] random = new int[num];
    for (i = 0; i < num; i++) {
        random[i] = (int) (num * Math.random());
    }
    try {
        File randFile = new File("QuickSort.txt");
        FileWriter fw = new FileWriter(randFile);
        PrintWriter pw = new PrintWriter(fw);

        pw.println("並べ替え前");
        for (i = 0; i < num; i++) {
            pw.print(random[i] + " ");
        }
    }
}
```

クイックソートの
処理部分
(メソッドとして定義～続き～)

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved.

33

QuickSort.java(中編2)

```
if (first < i - 1) { sort(rand, first, i - 1); }
if (j + 1 < last) { sort(rand, j + 1, last); }
}

public static void main(String[] args) {
    int i, j, temp, num = 10000;
    int[] random = new int[num];
    for (i = 0; i < num; i++) {
        random[i] = (int) (num * Math.random());
    }
    try {
        File randFile = new File("QuickSort.txt");
        FileWriter fw = new FileWriter(randFile);
        PrintWriter pw = new PrintWriter(fw);

        pw.println("並べ替え前");
        for (i = 0; i < num; i++) {
            pw.print(random[i] + " ");
        }
    }
}
```

並べ替えをする
数の個数

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved.

34

QuickSort.java(中編3)

```
if (first < i - 1) { sort(rand, first, i - 1); }
if (j + 1 < last) { sort(rand, j + 1, last); }
}

public static void main(String[] args) {
    int i, j, temp, num = 10000;
    int[] random = new int[num];
    for (i = 0; i < num; i++) {
        random[i] = (int) (num * Math.random());
    }
    try {
        File randFile = new File("QuickSort.txt");
        FileWriter fw = new FileWriter(randFile);
        PrintWriter pw = new PrintWriter(fw);

        pw.println("並べ替え前");
        for (i = 0; i < num; i++) {
            pw.print(random[i] + " ");
        }
    }
}
```

並べ替えをする数を
乱数で作成

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved.

35

QuickSort.java(後編1)

```
Calendar cal1 = Calendar.getInstance();
long beforeTime = cal1.getTimeInMillis();

sort(rand, 0, num - 1);

Calendar cal2 = Calendar.getInstance();
long afterTime = cal2.getTimeInMillis();
System.out.println("かかった時間(クイックソート): " + (afterTime - beforeTime) + "ミリ秒");

pw.println("バブルソート並べ替え後");
for (i = 0; i < num; i++) {
    pw.print(random[i] + " ");
}
fw.close();
pw.close();
} catch (IOException e) {} }
```

数を並べ替える直前の時間を計測

Copyright (C) Junko Shingane, Tokyo Woman's Christian University 2018 All rights reserved.

36

QuickSort.java(後編1)

```

Calendar cal1 = Calendar.getInstance();
long beforeTime = cal1.getTimeInMillis();

sort(random, 0, num - 1);

Calendar cal2 = Calendar.getInstance();
long afterTime = cal2.getTimeInMillis();
System.out.println("QuickSortのメソッドを呼び出す処理にかかった時間(マイクロ秒): " + (afterTime - beforeTime) + "マイクロ秒");

pw.println("QuickSortのメソッドを呼び出す処理");
for (i = 0; i < num; i++) {
    pw.print(random[i] + " ");
}
pw.close();
fw.close();
}
catch (IOException e) {
}
}

```

QuickSortのメソッドを呼び出す処理

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

QuickSort.java(後編3)

```

Calendar cal1 = Calendar.getInstance();
long beforeTime = cal1.getTimeInMillis();

sort(random, 0, num - 1);

Calendar cal2 = Calendar.getInstance();
long afterTime = cal2.getTimeInMillis();
System.out.println("QuickSortのメソッドを呼び出す処理にかかった時間(マイクロ秒): " + (afterTime - beforeTime) + "マイクロ秒");

pw.println("QuickSortのメソッドを呼び出す処理");
for (i = 0; i < num; i++) {
    pw.print(random[i] + " ");
}
pw.close();
fw.close();
}
catch (IOException e) {
}
}

```

QuickSortのメソッドを呼び出す処理

QuickSortのメソッドを呼び出す処理にかかった時間を計測

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

ファイルの役割

- BubbleSort.java
 - 「BubbleSort.txt」に、並び替え前と後の数を保存
- QuickSort.java
 - 「QuickSort.txt」に、並び替え前と後の数を保存

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

やってみよう!(1)

- 配ったカードを、順番をバラバラにして...
 1. バブルソートで昇順に並び替え(練習)
 2. バブルソートで昇順に並び替え、かかった時間を計測(本番)
 3. 自分のやりやすい方法で昇順に並び替え、かかった時間を計測
 4. 2. と3. の時間を比較(どちらが時間がかかっているか??)

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

やってみよう!(2)

- 授業のページから2つのファイルをダウンロードし、コンパイルと実行
 - BubbleSort.java
 - QuickSort.java
- 実行した結果、かかった時間が表示されるので、どちらが速いかを比較
- 2つのファイルの中に書いてある、「並び替えをする数の個数」をいろいろな数に変更し、実行しなおして比較

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

第1回課題

- プログラム作成課題
 - 提出先: junko@cis.twcu.ac.jp
 - 第1回の授業で連絡した、連絡用のメールアドレスとは異なるので注意すること
 - このメールアドレスへの質問は受け付けないので注意すること
 - 提出期限: 10月29日(月) 18:00
 - 詳細は授業のページに
 - <http://www.cis.twcu.ac.jp/~junko/Programming/Java2/>

Copyright (C) Junko Shiragane, Tokyo Woman's Christian University 2018 All rights reserved.

次回

- 課題の質問受け付け
 - 前回の復習問題の解答はなし