アルゴリズムとデータ構造

第1回講義：イントロダクション

Jie Huang

２年前期

Algorithms and Data Structures

No. 1: Introduction

Jie Huang

First Sem., 2 A. Year

コースの情報

- 担当教員: 黄 捷 (Jie Huang)
  - オフィス 126-B
  - メール j-huang
  - 内線 3510
- 情報はwwwページにて提供
  - http://web-int/~j-huang/Lecture/Algorithms/algorithms.html
- 関連授業
  - これまで: Intro. Programming、C Programming
  - これから: Advanced Algorithms

Course information

- Instructor: Jie Huang
  - Office 126B
  - Mail j-huang
  - In campus phone 3510
- Home page of this course
  - http://web-int/~j-huang/Lecture/Algorithms/algorithms.html
- Related courses
  - Prerequisites: Intro. Programming, C Programming
  - Advanced: Advanced Algorithms

コースの説明

- *コンピュータをいかに効率的かつ効果的に使うかを学ぶ*
  プログラムをいかに効率的かつ効果的に組むかを学ぶ
- 授業の内容：
  - 配列／リスト
  - スタック／キュー
  - 再帰
  - 計算量分析
  - 木、グラフ
  - 整列
  - 探索
  - 動的計画法

Why study algorithms and data structures

- How to use computers effectively and efficiently
- How to make programs effectively and efficiently
- To learn:
  - array / list
  - stack / queue
  - recursion
  - complexity
  - tree, graph
  - sort
  - search
  - dynamic programming

授業のスタイル

- 毎週講義1コマ、演習2コマ
  - 講義：15週
    - クイズ：（講義始め、毎回15分程度）
  - 演習：14週
- 中間試験（第8週）、期末試験（試験期間）
- 成績評価：
  - クイズ（20%）
  - 演習14回分（40%）
  - 中間、期末試験（40%）

合格するためには授業の出席（クイズの提出回数で評価）、演習と試験の両方をパスする必要がある。

Lecture style

- 1 period for lecture, 2 periods for exercises
  - Lecture: 15 weeks
    - Quiz: (20 minutes in the beginning of lecture)
  - Exercise: 14
- Mid-term exam (8th week), final exam (exam week)
- Evaluation:
  - Quizzes (20%)
  - Exercises (40%)
  - Mid-term and final exams (40%)
  - To pass the course, you need the attendance (counted by number of quiz submission), exercises and exam scores.

演習について

- 目的：講義で習った基本的なアルゴリズムについての理解を深め、それをプログラムで実現し、確かめる。
  プログラミングはC言語を用いる。
- 毎週3つの演習問題
  - **問題1**：処理の流れを紙に書いてチェックする問題で、アルゴリズムを理解しているかを確認するための基本問題
    - 必ず演習時間内に解いてTAにチェックしてもらうこと。
  - **問題2：プログラミング問題**
    - 演習時間内に終われるよう頑張ってください。
  - **問題3：問題2の発展問題**
    - プログラミングスキルを高めるための問題で、提出期限までに完成してください。

演習問題の提出について

- 提出期限：次の演習の前（厳守）、期限過ぎた回答は評価しない
- 提出方法
  - 各自のホームディレクトリの下に alg1/を作成する.
  - /alg1/の下に ex01/, ex02/,…,ex12/ などを作成する.
  - 作成したプログラムprog02.c, prog03.c などをそれぞれのディレクトリに置く.
  - ディレクトリとファイルのPermissionを全て705にする.
  - 以上のディレクトリ名、ファイル名などを間違いないよう、十分気をつけてください。間違えた場合、採点できない場合もある。
  - 提出内容は名前、学籍番号を先頭のコメント部分にいれ、プログラム本文と共に送ってください。結果がある場合も必ずコメントアウトして最後の部分に付けてください。
    注：シグネチャなどコンパイルできない部分は入れないでください。

アルゴリズムとは

- アルゴリズムは値あるいはその集合を入力とし、明確に定義された計算手続によって値あるいは値の集合を出力する計算ステップの系列である。
- 問題を解くための解法のことであり、特定のコンピュータや言語に依存しないものであるが、プログラムとして実現するのに向いているものを指す。
  - 整列、探索、…
- アルゴリズムを実装するには
  - 分析
  - 実装（Cなどでプログラムを組む）
  - テスト
  - （実際の問題に）応用

What is an algorithm

- An algorithm is a well-defined computational procedure (sequence of computational steps) that takes some values as input, and produces some values as output.
- A tool to solve well-defined computational problems, which can be implemented by programming but not depends on a specified programming language.
- Ex. sorting, searching, …
- To implement an algorithm
  - analysis
  - implementation by programming
  - test
  - applying for real problems

データ構造とは、

- 情報を保存するためのデータの構成法で、アルゴリズムと密接に関連するものである。
  本コースで扱うものとして以下のものがある
  - 配列
  - リスト
  - スタック
  - キュー
  - 木
  - グラフ
  - …

What is a data structure

- A way of organizing data so that it can be used by algorithms efficiently
- Data structures treated in this course
  - array
  - list
  - stack
  - queue
  - tree
  - graph
  - …

アルゴリズムとデータ構造

- ***アルゴリズム＋データ構造＝プログラム***

（N. Wirth, Pascal言語の発明者）

- アルゴリズムを実現するためには、適切なデータ構造が必要である。また、データ構造はアルゴリズムの最終産物もしくは副産物として存在する。

- アルゴリムズの理解には、データ構造の諸性質を明らかにする必要がある。

Algorithms and data structures

- ***algorithm + data structure = program***

  (Niklaus Wirth, the inventor of Pascal)
- An algorithm needs to organize data involved in the computation that leads to data structures. (Data structures exist as the byproducts or end products of algorithms.)
- Algorithms and data structures go hand in hand. Simple algorithms can give rise to complicated data structures and, conversely, complicated algorithms can use simple data structures.
- We must study data structures in order to understand the algorithms.

疑似コード(pseudocode)

- プログラム言語ほど厳密ではなく、既存のプログラム言語の構文と自然言語に近い表現を組み合わせた架空（疑似）のプログラム（アルゴリズム）言語である。
- 疑似コードはあいまいさがなく、簡単にプログラムコードに置き換えられることが必要である。
- どのプログラム言語の構文に基づくか決まったルールがなく、自然言語による表現は内容の理解しやすさを優先する。

Pseudocode

- Pseudocode is an informal high-level description which is based on some programming languages and augmented with nature language for easier understand.
- There is no standard for pseudocode syntax.
- A good pseudocode language must avoid any ambiguities and easy to be replaced by programming languages.

アルゴリズム例:最大公約数を求めるプログラム

疑似コード

```
gcd(a, b)
    n = min(a, b)
    for i = n down to 1
        if a,bがともにiで割り切れる
            return i
```

注）a, bの小さい値から虱潰しに探す
非効率的なアルゴリズム、ループ数は最大min(a,b)である。

```
/* GCD function in C */
int gcd(int a, int b)
{
  int i, r;
  if (a < b) i = a;
  else i = b;
  for(;i>=1;i--) {
    r = a%i+b%i;
    if(r ==0 ) break;
  }
  return i;
}
```

Example of algorithm: greatest common divisor

Pseudocode:

```
gcd(a, b)
    n = min(a, b)
    for i = n down to 1
        if i divides both a and b
            return i
```

Note: a simple inefficient algorithm searching from the minimum of a and b one-by-one, with a maximum loops number of min(a,b).

```
/* GCD function in C */

int gcd(int a, int b)
{
  int i, r;
  if (a < b) i = a;
  else i = b;
  for(;i>=1;i--) {
    r = a%i+b%i;
    if(r ==0 ) break;
  }
  return i;
}
```

ユークリッド互除法のアルゴリズム

- 原理
  - 紀元前300頃、ユークリッドの「原論」に記される最古と言われるアルゴリズム
  - 2つの自然数a, b (a>=b)の最大公約数はaをb割った余りrとbの最大公約数に等しいという性質を利用する
  - aとbを素因数分解するよりはるかに速く最大公約数を求めることができる。

- アルゴリズム
  1. 入力をa, bとする
  2. bが0なら、aを出力としてアルゴリズムを終了する
  3. aをbで割った余りをrとする。
  4. rを新たにbとし、元のbをaとしてステップ2に戻る
- 最悪でもbの桁数の 5 倍回繰り返すだけで最大公約数が求まる。

Euclidean algorithm (division)

- Principle
  - Described in Euclid's Elements in 300 BC, to be said the oldest algorithm.
  - GCD of two numbers does not change if the larger number is replaced by its difference with the smaller number.
  - It can find the GCD efficiently without having to compute the prime factors which is a computationally difficult problem.

- Algorithm procedure
  1. Input a, b (a >= b)
  2. If b eques to 0, output a and quit.
  3. Let r be the reminder when divide a by b.
  4. Replace b with r, and a with b, go back to step 2.
- The number of steps is never more than five times the number of digits of the smaller number.

ユークリッド互除法のアルゴリズム

再帰呼び出しを利用した疑似コード

```
gcd(a,b)
    もし a=0 ならば
        return b
    そうでないならば
        return gcd(b%a,a)
```

引き算だけで実現したプログラム

```
int gcd(int a, int b)
{
  int t;
  while (a > 0) {
    if (a < b) {
      t=a; a=b; b=t;
    }
    a = a - b;
  }
  return b;
}
```

Euclidean algorithm (division)

Pseudocode by recursion

```
gcd(a,b)
    if a=0 then
        return b
    else
        return gcd(b%a,a)
```

A subtraction-based version (Euclid's original version)

```
int gcd(int a, int b)
{
  int t;
  while (a > 0) {
    if (a < b) {
      t=a; a=b; b=t;
    }
    a = a – b;
  }
  return b;
}
```

ユークリッド互除法の証明

2つの正の整数a, b (a>=b) を以下のように表す (p_iは素数）

$$a = \prod_i p_i^{n_i} \qquad b = \prod_i p_i^{m_i}$$

最大公約数 (Greatest Common Divisor) は

$$\gcd(a,b) = \prod_i p_i^{\min(n_i,m_i)} = \prod_{n_i \geq m_i} p_i^{m_i} \cdot \prod_{n_i < m_i} p_i^{n_i}$$

最小公倍数 (Least Common Multiple) は

$$\mathrm{lcm}(a,b) = \prod_i p_i^{\max(n_i,m_i)} = \prod_{n_i \geq m_i} p_i^{n_i} \cdot \prod_{n_i < m_i} p_i^{m_i}$$

従って

$$a/b = q + r/b = \prod_{n_i \geq m_i} p_i^{n_i - m_i} + \left(\prod_{n_i < m_i} p_i^{n_i} \right) /b$$

$$\gcd(a,b) = \gcd(r,b)$$

Prove for Euclid's algorithm

Factorize the natural numbers a, b (a>=b) as follows (p_i is prime number)

$$a = \prod_i p_i^{n_i} \qquad b = \prod_i p_i^{m_i}$$

The Greatest Common Divisor is given by

$$\gcd(a,b) = \prod_i p_i^{\min(n_i,m_i)} = \prod_{n_i \geq m_i} p_i^{m_i} \cdot \prod_{n_i < m_i} p_i^{n_i}$$

Then

$$r = \prod_{n_i < m_i} p_i^{n_i}$$

$$\gcd(r,b) = \prod_{n_i \geq m_i} p_i^{m_i} \cdot \prod_{n_i < m_i} p_i^{n_i} = \gcd(a,b)$$

アルゴリズム例：多項式を求めるプログラム

- 式の通りに計算を行う方法

```
double p(double a[], int
   n, double x)
  int i,j;
  double tmp, result;
  result = a[0];
  for(i=1;i<=n;i++){
    tmp=1;
    for(j=0;j<i;j++){
      tmp*=x;
    }
    result+=tmp*a[i];
  }
  return result;
}
```

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_0$$

- 漸近式を使う方法
  （ホーナー法）

```
double p(double a[], int
  n, double x)
  int i;
  double result;
  result = a[n];
  for(i=n-1;i==0;i--)
    result =
  result*x+a[i];
  return result;
}
```

$$p(x) = (((a_n x + a_{n-1})x + a_{n-2})x + \ldots)x + a_0$$

Example of algorithm: calculation of polynomial

- Direct calculation

```
double p(double a[], int n,
   double x)
  int i,j;
  double tmp, result;
  result = a[0];
  for(i=1;i<=n;i++){
    tmp=1;
    for(j=0;j<i;j++){
      tmp*=x;
    }
    result+=tmp*a[i];
  }
  return result;
}
```

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_0$$

- Recurrence relation (Horner's method)

```
double p(double a[], int n,
   double x)
  int i;
  double result;
  result = a[n];
  for(i=n-1;i==0;i--)
    result = result*x+a[i];
  return result;
}
```

$$p(x) = (((a_n x + a_{n-1})x + a_{n-2})x + \ldots)x + a_0$$

プログラムファイルの作成、コンパイル、実行

- emacs
  - emacs euclid.c
- gccでコンパイル
  - gcc –o euclid euclid.c
- プログラム名で実行
  - euclid
- makefileによる一括処理

来週のトピックス

- 基本データ構造
  - 配列
  - リスト
  - その他

Topic of next week

- Data structures
  - array
  - list
  - others