アルゴリズムの設計と解析

黄　潤和

Goal

【授業の概要と目的（何を学ぶか)】

The goal of this course is to enhance students' knowledge of data strcture and skill of applying associated algorithms. This course will cover the content review of learned data structures and algorithms related tree and graph, and plus algorithm analysis and design techniques.

到達目標:

The objectives of this course are to make students firmly laying good foundation of data structures and algorithms, and one-step further comprehensively understanding algorithm analysis and having design and implementation skills in Python, Java, or other programming language.

【テキスト（教科書)】
"Introduction to
The design and Analysis of Algorithms", Anany Levitin,
Pubisher: Pearson,
ISBN-13: 978-0-13-231681-1

【参考書】
書名: Introduction to Algorithms, Third Edition
著者: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
出版社: The MIT Press
出版年: 2009 年

【成績評価の方法と基準】
中間課題 (20%) と定期試験 (80%)

Contents (L1 - Introduction)

- What is an Algorithm?
- How to design?
- How to analyze?

Why study algorithms?

- Algorithms play the central role both in the
  - science
  - practice

  Of computing

- From a practical standpoint

  - you have to know a standard set of important algorithms

  - you should be able to design new algorithms

- From theoretical standard

  - the study of algorithms is the core of computer science

  related to many other fields

  useful in developing analytical skills

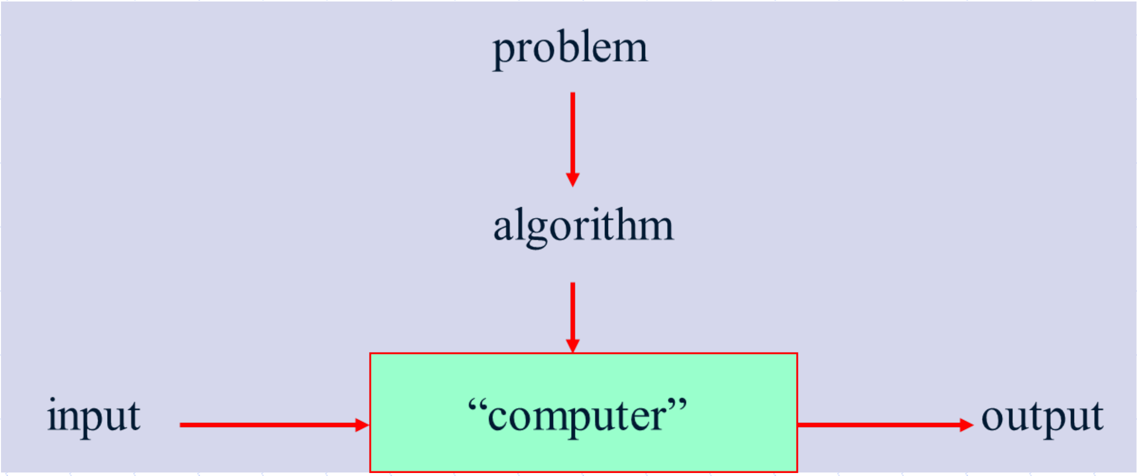
Introduction

What is an Algorithm?

ambiguous
あいまいな

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

合理的な



Two main approaches

1. from typical problem types
2. from algorithm design techniques

1. from typical problem types

(a number of algorithms to a problem type)

- Sorting, searching, graphs, …
  - Bubble sort, quick sort, merge sort, heap sort
- Merits:
  - Enables the efficiency comparison between different algorithms
- Disadvantages:
  - Sacrifices the conceptual clearness on algorithm design techniques

Some Well-known Computational Problems

- **Sorting**
- **Searching**
- **Shortest paths in a graph**
- **Minimum spanning tree**
- **Primality testing**
- **Traveling salesman problem**
- **Knapsack problem**
- **Chess**
- **Towers of Hanoi**
- **Program termination**

e.g. Sorting problem
There are many different algorithms

	最悪実行時間	最良実行時間	平均実行時間	その場ソートか否か	その他の特徴
挿入ソート	$O(n^2)$	$O(n)$	$O(n^2)$	○	
マージソート	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	×	分割統治法
ヒープソート	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	○	ヒープはデータ構造としても有用
クイックソート	$O(n^2)$	$O(n \lg n)$	$O(n \lg n)$	○	• 分割統治法 • 実用上は最も高速

e.g. Greatest common factor　最大公約数

Problem:　Find gcd(*m*,*n*),

the greatest common divisor of two nonnegative,

not both zero integers *m* and *n*

e.g.: gcd(60,24) = 12, gcd(60,0) = 60, ...

(1) Euclid's algorithm: it is based on repeated application of equality

gcd(*m*,*n*) = gcd(*n*, *m* mod *n*)

until the second number becomes 0,

which makes the problem trivial.

e.g.: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

Other methods

(to the same problem: Greatest common factor)

(2) Brute force solution

Step 1 Assign the value of min{*m,n*} to *t*

Step 2 Divide *m* by *t*. If the remainder is 0, go to Step 3; otherwise, go to Step 4

Step 3 Divide *n* by *t*. If the remainder is 0, return *t* and stop; otherwise, go to Step 4

Step 4 Decrease *t* by 1 and go to Step 2

(3) Finding the prime factors

Step 1 Find the prime factorization of *m*

Step 2 Find the prime factorization of *n*

Step 3 Find all the common prime factors

Step 4 Compute the product of all the common prime factors and return it as gcd*(m,n)*

60 = 2x2x3x5

24 = 2x2x2x3

Common prime factors are: 2,2,3

gdc(60,24) = 2x2x3 = 12

2. from algorithm design techniques

Some well-known algorithm design techniques

- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Brute force
- Greedy approach
- Dynamic programming
- Backtracking and branch-and-bound
- Space and time tradeoffs

A design technique to solve different problems

e. g. Divide and Conquer technique (分割統治) which is used in many different algorithms for solving different problems.

For example

- Searching
- Sorting
- Matrix multiplication
- ......

Which is better?

Two main issues:

(1) How to design algorithms?

(solve the problem)

(2) How to analyze algorithms?

(evaluate/optimize the algorithms)

Algorithm design techniques

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch and bound
- ……

Analysis of algorithms

- How good is the algorithm?
  - correctness
  - space efficiency, usually in terms of the amount of fast memory, but disk volume and network bandwidth shall be taken as another sort of limited resource for tasks related to big data.
  -

- Does there exist a better algorithm?
  - lower bounds
  - optimality

For example: sorting

- Rearrange the items of a given list in ascending order.
  - Input: A sequence of n numbers $<a_1, a_2, \ldots, a_n>$
  - Output: A reordering $<a'_1, a'_2, \ldots, a'_n>$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.
- Why sorting?
  - Help searching
  - Algorithms often use sorting as a key subroutine.
- Sorting key
  - A specially chosen piece of information used to guide sorting. E.g., sort student records by student ID.

An example:

Approach→

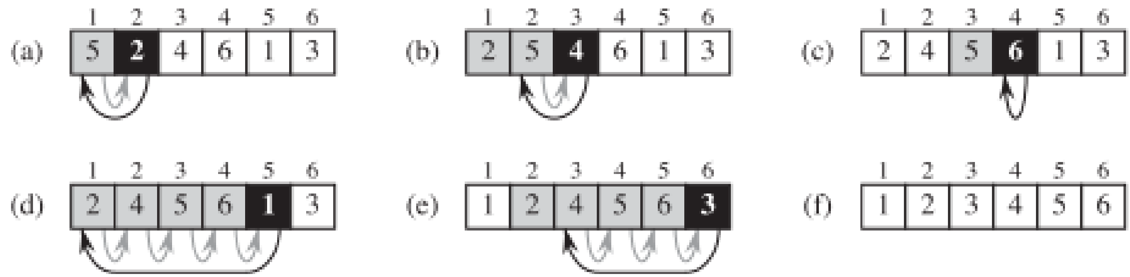


Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key moves to in line 8. **(f)** The final sorted array.

Pseudo code →

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1 \,.\,.\, j-1]$.	0	$n-1$
$i = j - 1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

the best case occurs
if the array is already sorted,
$t_j = 1$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

the best-case running time is $\theta(n)$

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8)\,. \end{aligned}$$

A worst-case running time of $\theta(n^2)$

$O(n^2)$

bound on worst-case running time of insertion sort

O-notation

The Θ-notation asymptotically bounds a function from above and below. When we have only an ***asymptotic upper bound***, we use O-notation. For a given function $g(n)$, we denote by $O(g(n))$ (pronounced "big-oh of g of n" or sometimes just "oh of g of n") the set of functions

$$O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \}\,.$$

We use O-notation to give an upper bound on a function, to within a constant

Efficiency is very much depended on data structure

Apart from the linked list, there are other often used data structure.

Often used data structures

- Array
  - Sequentially ordered, random access, update
- linked list
  - Fast insertion, deletion
- Stack
  - First in last out
- Queue
  - First in first out
- Graph
  - Advanced modeling, but costly
- Tree
  - Divide and conquer
- set and dictionary
  - Implemented as array, list, or tree

About this course

Textbook

Anany Levitin, Addison-Wesley, 2011

Reference book

Thomas H. Cor-men, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press, 2009
Anany Levitin, Addison-Wesley, 2011

About this course

- Teaching plan

  It is expected to have slight adjustments

- Evaluation

  - Mid-term: exercise problems (20%)
  - Exams: final exam (80%)

Exercise 1-1

- What is the output of Test1(200) ?
  Test1（200）の出力結果は何ですか？
  Test1は次のアルゴリズムです。

Algorithm Test1(n)
    $b \leftarrow 0$
    for $i \leftarrow 1$ to n do
        if i mod 6 = 0 then $b \leftarrow b + 1$
                else if i mod 9 = 0 then $b \leftarrow b + 10$
    return b

Exercise 1-2

What are the output of Test2(100)?
Test2（100）の出力結果は何ですか？
Test2は次のアルゴリズムです。

Algorithm Test2(n)
  $b \leftarrow 0$
  for $i \leftarrow 1$ to n do
    for $j \leftarrow 1$ to i do
      $b \leftarrow b + 1$
  return b

Exercise 1-3

What are the output of Test3(1000) ?
Test3（1000 ）の出力結果は何ですか？
Test3は次のアルゴリズムです。

Algorithm Test3(n)
    $i \leftarrow 1$
    $b \leftarrow 0$
    while $i < n$ do
        $b \leftarrow b + 1$
        $i \leftarrow 2i$
    return b