Analysis and Design of Algorithms

Chapter 1: Introduction

School of Software Engineering © Yaping Zhu

Introduction of Teacher

- ***Contact Information:***

  Office: Jishi Building, Room 314

  Email: yapingzhu@tongji.edu.cn

- ***Course Information:***

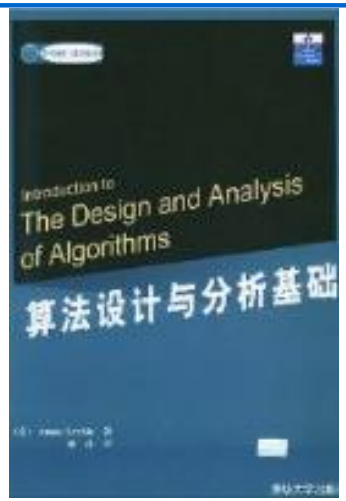
  Slides can be downloaded from:

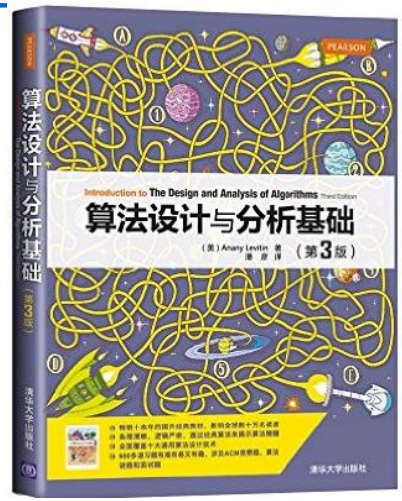
  QQ 群 共享文件

- ***TA Information:***

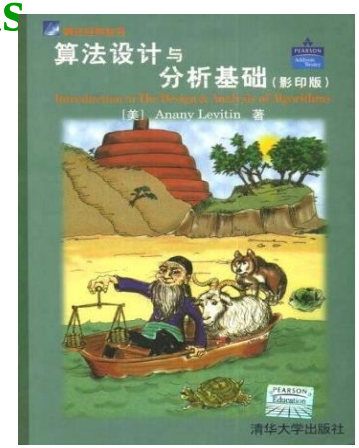
References

算法设计与分析基础(第2版) .
（美）Anany Levitin 著，潘彦译 .
清华大学出版社 .2007年1月.





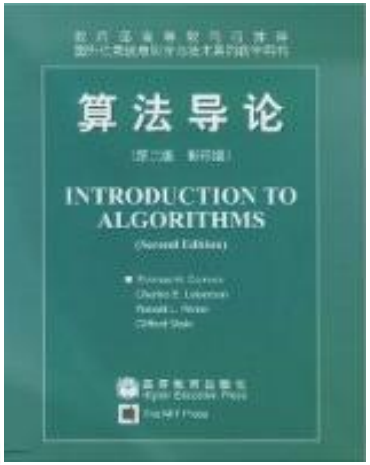
Introduction to the Design and Analysis of Algorithms.
Anany Levitin.
清华大学出版社. 2003年.



References

算法设计与分析(第三版). 王晓东.
电子工业出版社.2007年5月.

Introduction To Algorithms (Mit Press 2nd Edition). Thomas H.Cormen.
高等教育出版社& The MIT Press. 2002年5月.



计算机算法导引: 设计与分析.
卢开澄. 清华大学出版社. 2006年.

Examination

- Homework: 40%, 4 times, and each time 10%, 10%, 10%,10%
- Final examination: 50%
- Attendance: 5% (being absent >=5 times, you will fail this course)
- Class activity: 5% being active in class and answering my questions correctly

Course Prerequisite

- Data Structure
- C, Java or other programming languages
- Discrete Mathematics
- Advanced Mathematics

Chapter 1: Introduction

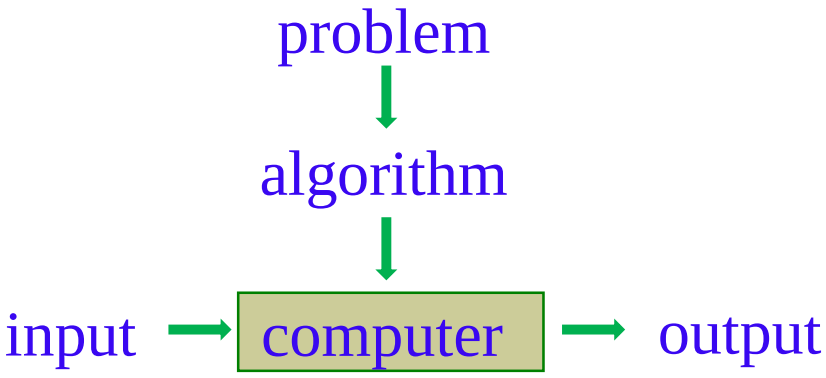
- What's Algorithm
- Fundamentals of algorithmic problem solving
- Important problem types
- Fundamental data structures

What's Algorithm

Definition

- 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.

- 算法是一系列解决问题的明确指令，也就是说，对于符合一定规范的输入，能够在有限时间内获得要求的输出。



What's Algorithm

Some points for algorithms

- Each step of an algorithm must be unambiguous.
- Different algorithms for a certain problem.
- Different representations to describe a certain algorithm.
- Different ideas and different execution speed for different algorithms.

What's Algorithm

Properties of algorithms

- **Input:** 0 or more valid input values, to provide the initialization conditions
- **Output:**
  - produce the correct output given a valid input
  - at least one value is produced by the algorithm
  - desired input/output relationship specified by the problem
- **Tangibility(确定性):**
  - each instruction/step is clear
  - precisely and unambiguously specified

Example: 不符合确定性的运算

- 5/0
- 将6或7与x相加
- 未赋值变量参与运算

Example of Algorithm

- ***Problem:* Computing the Greatest Common Divisor of two integers**
  - **gcd(*m*, *n*)**: the largest integer that divides both *m* and *n*

- ***Algorithm I***
  - **Euclid's algorithm（欧几里得算法）:**

    gcd(*m*, *n*) = gcd(*n*, *m* mod *n*) iteratively while $n \neq 0$

    gcd(*m*, 0) = *m*

    gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

  - **Natural language**

    Step 1: If *n* = 0, return the value of *m* as the answer and stop;
    otherwise, proceed to Step 2.

    Step 2: Divide *m* by *n* and assign the value of the remainder to *r*.

    Step 3: Assign the value of *n* to *m* and the value of *r* to *n*.
    Go to Step 1.

Example of Algorithm

- **Pseudocode（伪代码）:**

a mixture of a natural language and programming language-like structures.

- omits declarations of variables
- use indentation to show the scope of such statements as for, if, and while.
- Use "←" for assignment, " // " for annotation

Algorithm Euclid(*m*, *n*)
//Computes gcd(*m*, *n*) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers *m* and *n*
//Output: Greatest common divisor of *m* and *n*
while $n \neq 0$ **do**
$r \leftarrow m \bmod n$
$m \leftarrow n$
$n \leftarrow r$
return *m*

Example of Algorithm

Algorithm II

Consecutive Integer Algorithm (连续整数检测算法)

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

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

Step 3: Divide *n* by *t*. If the remainder of this division is 0, return the value of *t* as the answer and stop; otherwise, proceed to Step 4.

Step 4: Decrease the value of *t* by 1. Go to Step 2.

当它的一个输入为0时，错误。准确定义输入的值域。

Example of Algorithm

Algorithm II

Consecutive Integer Algorithm

```
Algorithm ConsecutiveInteger(m, n)
          //使用连续整数检测法计算gcd(m, n)
          //输入：两个不全为0的非负整数m, n
          //输出：m, n的最大公约数
        if n=0 return m
          t=min{m,n}
          while t>0 do
              if (m mod t)==0
                 if (n mod t)==0
                    return t
                 else t=t-1
          return t
```

Example of Algorithm

Algorithm III

Middle-school procedure

Step 1: Find the prime factors （质因数） of m. ?

Step 2: Find the prime factors of n. ?

Step 3: Identify all the common factors in the two prime expansions found in Step 1 and Step 2. (If p is a common factor occurring p_m and p_n times in m and n, respectively, it should be repeated in $\min\{p_m, p_n\}$ times.)

Step 4: Compute the product of all the common factors and return it as the gcd of the numbers given.

60=2*2*3*5，24=2*2*2*3，gcd(60,24)=2*2*3=12

Example of Algorithm

- ***Algorithm III***
  - **Sieve of Eratosthenes (埃拉托色尼筛选法)，产生一个不大于给定整数n的连续质数序列**

n=25

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	3		5		7		9		11		13		15		17		19		21		23		25
2	**3**		5		7				11		13				17		19				23		25
2	3		**5**		7				11		13				17		19				23		

Example of Algorithm

Algorithm III

Algorithm Sieve(n)

for $p \leftarrow 2$ **to** n **do** $A[p] \leftarrow p$
for $p \leftarrow 2$ **to** $\lfloor \sqrt{n} \rfloor$ **do** //see note before pseudocode
  if $A[p] \neq 0$ //p hasn't been eliminated on previous passes
    $j \leftarrow p * p$
    while $j \leq n$ **do**
      $A[j] \leftarrow 0$ //mark element as eliminated
      $j \leftarrow j + p$
//copy the remaining elements of A to array L of the primes
$i \leftarrow 0$
for $p \leftarrow 2$ **to** n **do**
  if $A[p] \neq 0$
    $L[i] \leftarrow A[p]$
    $i \leftarrow i + 1$
return L

Fundamental of algorithmic problem solving

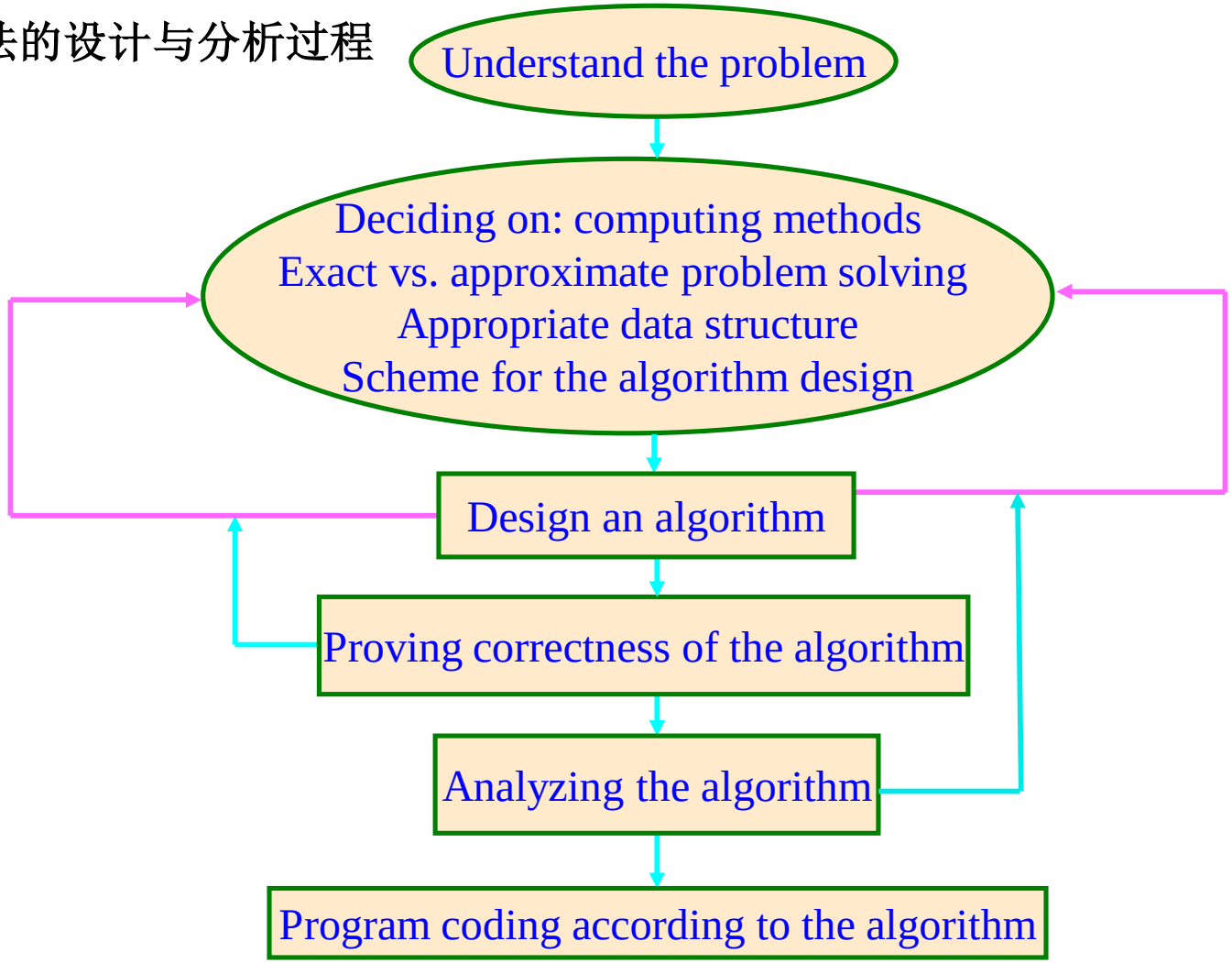
- Understanding the problem（理解问题, 考虑到所有可能的input. ）
- Ascertaining the capabilities of the computational device
  Sequential algorithm（顺序算法）和 Parallel algorithm（并行算法）
- Choosing the exact and approximate problem solving
  在精确解法和近似解法之间做出选择.
- Algorithm design techniques
  积累传统的算法是基础.

算法是问题的程序化解决方案

- Designing an algorithm and data structures
- Methods of specifying an algorithm: pseudocode（伪代码）
- Proving an algorithm correctness
- Analyzing an algorithm:
  Time efficiency（时间效率）和 Space efficiency（空间效率）
- Coding an algorithm

Fundamental of algorithmic problem solving

算法的设计与分析过程



一个好的算法是不懈努力和反复修正的结果。

Important problem types

Sort（排序）

- The sorting problem is to rearrange the items of a given list in nondecreasing order.

- In the case of records, we need to choose a piece of information to guide sorting. Such a specially chosen piece of information is called a key（键）.

Important problem types

Sort（排序）

Two properties of sorting algorithms deserve special mention.

- A sorting algorithm is called stable（稳定） if it preserves the relative order of any two equal elements in its input.
- The second notable feature of a sorting algorithm is the amount of extra memory the algorithm requires. An algorithm is said to be in-place (在位) if it does not require extra memory, except, possibly, for a few memory units.

Important problem types

Searching（查找）

- The searching problem deals with finding a given value, called a search key, in a given set.

- For searching, there is no single algorithm that fits all situations best. Some algorithms work faster than others but require more memory; some are very fast but applicable only to sorted arrays; and so on.

- Searching has to be considered in conjunction with two other operations: an addition to and deletion from the data set of an item (在数据集合中添加和删除元素).

Important problem types

String processing（字符串处理）

- A string is a sequence of characters from an alphabet (字母表中的符号所构成的序列).

- String matching (字符串匹配)--- searching for a given word in a text.

Important problem types

Graph problem（图问题）

- A graph can be thought of as a collection of points called vertices（顶点）, some of which are connected by line segments called edges（边）.

Important problem types

- ***Graph problem（图问题）***
  - Basic graph algorithms
    - graph-traversal algorithms (图的遍历算法)
    - shortest-path algorithms (最短路径算法)
    - topological sorting for graphs with directed edges (有向图的拓扑排序)

Important problem types

Graph problem（图问题）

The most well-known examples

- The traveling salesman (TSP, 旅行商问题) is the problem of finding the shortest tour through *n* cities that visits every city exactly once. In addition to obvious applications involving route planning.

- The graph-coloring problem (图填色问题) seeks to assign the smallest number of colors to the vertices of a graph so that no two adjacent vertices are the same color. This problem arises in several applications, such as event scheduling.

Important problem types

Combinatorial problem（组合问题）

- the TSP and the graph-coloring problem
- These are problems that ask, explicitly (明确) or implicitly, to find a combinatorial object – such as a permutation (排列), a combination (组合), or a subset (分组) – that satisfies certain constraints (满足某些限制).
- A desired combinatorial object may also be required to have some additional property (附加属性) such as a maximum value or a minimum cost.

Important problem types

Geometric problem（几何问题）

- The closest-pair problem (最近邻对问题) is self-explanatory : given *n* points in the plane, find the closest pair among them.
- The convex-hull problem (凸包问题) asks to find the smallest convex polygon that would include all the points of a given set.

Important problem types

Numerical problem（数值问题）

- Numerical problems are problems that involve mathematical objects of continuous nature:
  - solving equations and systems of equations (解方程)
  - computing definite integrals (定积分)
  - evaluating functions (评估函数)
  - and so on.

Fundamental data structures

Data structure (数据结构)：

对相关的数据项进行组织的特殊架构

- ***Linear data structures （线性数据结构）***
- ***Graphs（图）***
- ***Trees（树）***
- ***Sets and Directions（集合与字典）***

Fundamental data structures

Linear data structures（线性数据结构）

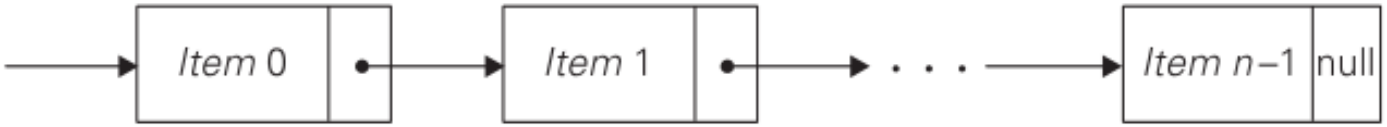
- Array (数组): a sequence of *n* items of the same data type that are stored contiguously in computer memory and made accessible by specifying a value of the array's index (连续存储并且通过特定的下标进行搜索).

- Linked list (链表): a sequence of zero or more elements called nodes (节点), each containing two kinds of information: some data and one or more links called pointers (指针) to other nodes of the linked list, (A special pointer called "null" is used to indicate the absence of a node's successor.)

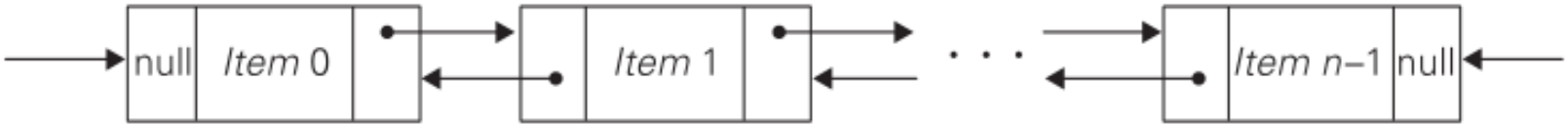
Fundamental data structures

- **Linked list (链表):**

- Singly linked list (单链表): each node except the last one contains a single pointer to the nest element (两个元素之间只有一个指针).



- Doubly linked list (双向链表): in which every node, except the first and the last, contains pointers to both its successor (后继节点) and its predecessor (前节点) .



Fundamental data structures

- **Linked list (链表):**

  On the positive side, linked lists do not require any preliminary (预留的) reservation of the computer memory, and insertions and deletions can be made quite efficiently in a linked list by reconnecting a few appropriate pointers (使用指针可以高效地删除或者添加节点).

Fundamental data structures

- The array and linked list are two principal choices in representing a more abstract data structure called a linear list or simply a list.
- A list is a finite sequence of data items, i.e., a collection of data items arranged in a certain linear order.
- The basic operations performed on this data structure are searching for, inserting, and deleting an element.

Fundamental data structures

- **Two special types of lists:**
  - Stack (堆栈)
  - Queue (队列)

Fundamental data structures

- Stack (堆栈): a list in which insertions and deletions can be done only at the end (只能在末尾进行插入和删除).

  - When elements are added to a stack (进栈) and deleted from it (出栈), the structure operates in a "last-in-first-out" (LIFO-后进先出) fashion – exactly like a stack of plates if we can add or remove a plate only from the top (添加和一处元素都是在端部进行).

  - Stacks have a multitude of applications, in particular, they are indispensable for implementing recursive algorithms (递归算法).

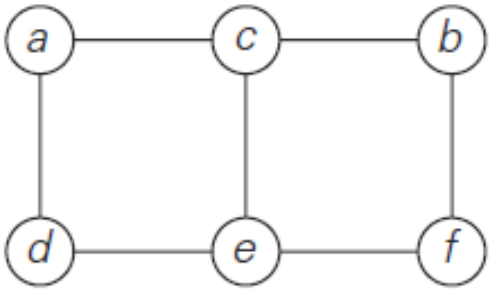
Fundamental data structures

- Queue (队列): a list from which elements are deleted from one end of the structure called the front (出队), and new elements are added to the other end called the rear (入队). (从队头移除元素，在队尾添加元素)
  - A queue operates in a "first-in-first-out" (FIFO-先进先出) fashion – akin to a queue of customers served by a single teller in a bank.
  - Queues also have many important applications, including several algorithms for graph problems.
  - Priority queue (优先队列): require selection of an item of the highest priority among a dynamically changing set of candidates (要求在动态变化的候选集合中选择优先级最高的项).

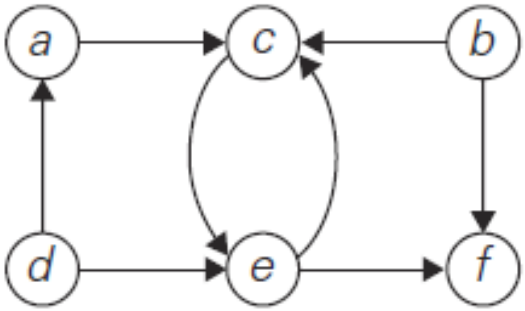
Fundamental data structures

Graphs（图）

- Graph G = <V, E>　V– vertices (顶点)； E– edges (边)
  - A graph G is called undirected if every edge in it is undirected.



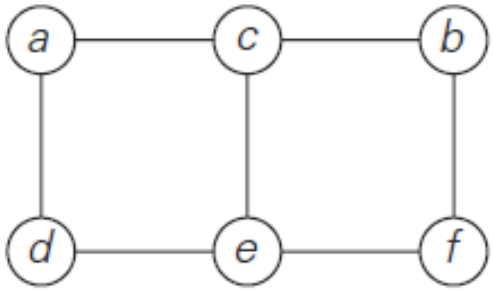
  - A graph whose every edge is directed is called directed.



Fundamental data structures

Graph Representation

- Graphs for computer algorithms are usually represented in one of two ways: the adjacency matrix (邻接矩阵) and adjacency lists (邻接链表).
  - The adjacency matrix of a graph with n vertices is an $n \times n$ boolean matrix with one row and one column for each of the vverticesv.

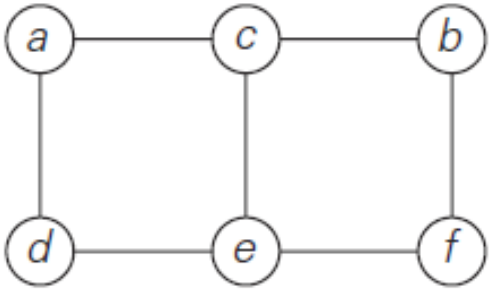


	a	b	c	d	e	f
a	0	0	1	1	0	0
b	0	0	1	0	0	1
c	1	1	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	1	0	1
f	0	1	0	0	1	0

Fundamental data structures

Graph Representation

- The adjacency lists of a graph or a digraph is a collection of linked lists, one for each vertex, that contain all the vertices adjacent to the list's vertex.

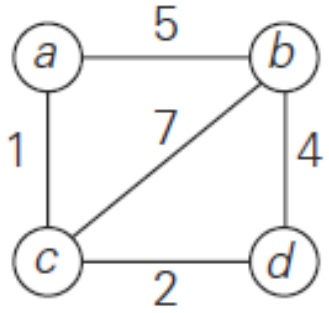


a	→	c	→	d		
b	→	c	→	f		
c	→	a	→	b	→	e
d	→	a	→	e		
e	→	c	→	d	→	f
f	→	b	→	e		

Fundamental data structures

Weighted Graphs

- A weighted graph (加权图) (or weighted digraph) is a graph (or digraph) with numbers assigned to its edges. These numbers are called weights or costs.



(a)

	a	b	c	d
a	∞	5	1	∞
b	5	∞	7	4
c	1	7	∞	2
d	∞	4	2	∞

(b)

a	→ b, 5 → c, 1
b	→ a, 5 → c, 7 → d, 4
c	→ a, 1 → b, 7 → d, 2
d	→ b, 4 → c, 2

(c)

(a)加权图；(b)邻接矩阵；(c)邻接链表

Fundamental data structures

Paths and Cycles

- Two important properties of graphs for a great number of applications: connectivity (连通性) and acyclicity (无环性).
- A path from vertex *u* to vertex *v* of a graph *G* can be defined as a sequence of adjacent (connected by an edge) vertices that starts with *u* and ends with *v*.
- Simple path: if all vertices of a path are distinct.
- Directed path: a sequence of vertices in which every consecutive pair of the vertices is connected by an edge directed from the vertex listed first to the vertex listed next.

Fundamental data structures

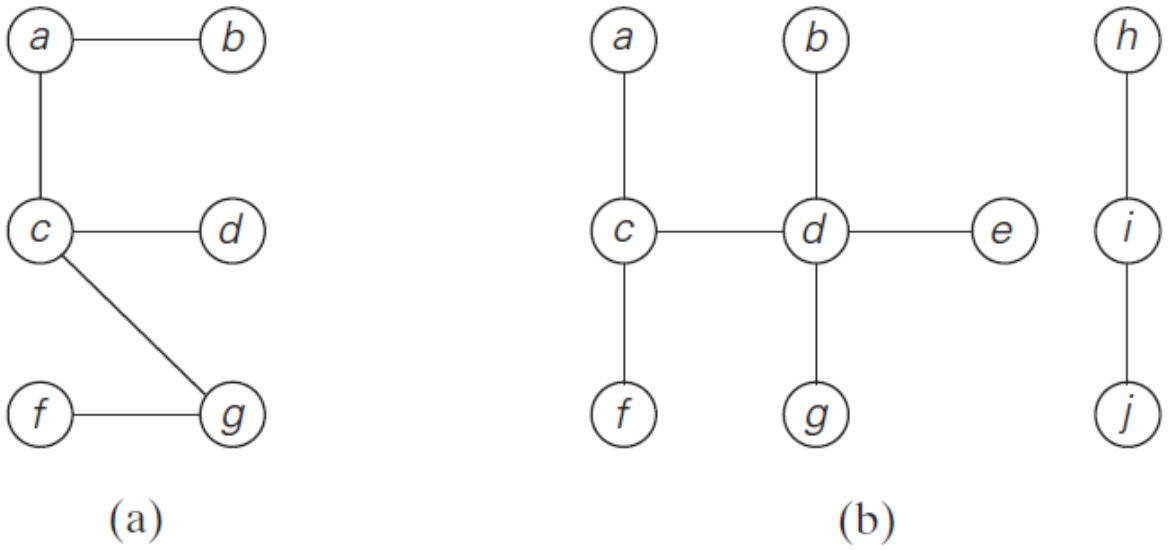
Paths and Cycles

- Connected graph: if for every pair of its vertices *u* and *v* there is a path from *u* to *v*.
- If a graph is not connected, such a model will consist of several connected pieces that are called connected components (连通分量) of the graph.

Fundamental data structures

Trees（树）

- Tree: a connected acyclic (连通无回路) graph.
- A graph that has no cycles but is not necessarily connected is called a forest: each of its connected components is a tree.



(a)树；(b)森林

Fundamental data structures

Rooted Trees（有根树）

- Property: For every two vertices in a tree, there always exists exactly one simple path from one of these vertices to the other.
- This property makes it possible to select an arbitrary vertex in a free tree and consider it as the root of the so-called rooted tree.

Ordered Trees（有序树）

- An ordered tree is a rooted tree in which all the children of each vertex are ordered.

Fundamental data structures

Sets and Directions（集合与字典）

- A set（集合）can be described as an unordered collection (possibly empty) of distinct items called elements of the set.
- In computing, the operations we need to perform for a set or a multiset most often are searching for a given item, adding a new item, and deleting an item from the collection. A data structure that implements these three operations is called the dictionary（字典）.

Contents of Algorithm

Algorithm Design Techniques/Strategies

- Brute force 蛮力法
- Divide and conquer 分治法
- Decrease and conquer 减治法
- Transform and conquer 变治法
- Greedy approach 贪心算法
- Dynamic programming 动态规划
- Back tracking 回溯法
- Branch and bound 分支界限法
- …

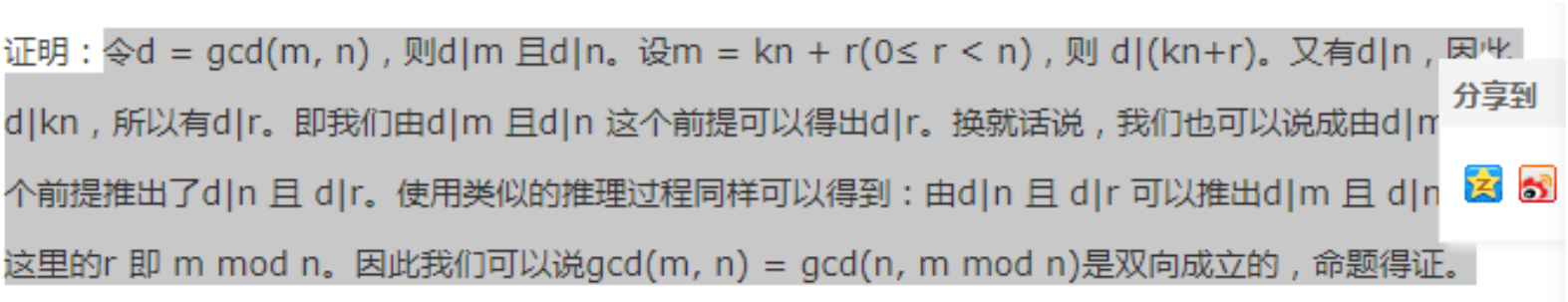
Summary

- 算法的定义：在有限时间内，对问题求解的一个清晰的指令序列。算法的每个输入确定了该算法求解问题的一个实例。
- 算法的特点：输入，输出，确定性，有穷性，和可行性。
- 算法可以用自然语言或者伪代码表示，或计算机程序实现
- 一个好的算法常常是不懈努力和反复修改的结果。
- 算法操作的是数据，所以数据结构很重要。

思考题

1. Prove the equality gcd(m, n) = gcd(n, m mod n) for every pair of positive integers m and n.

   http://blog.csdn.net/deserthero2013/article/details/51161696


证明：令d = gcd(m, n)，则d|m 且d|n。设m = kn + r(0≤ r < n)，则 d|(kn+r)。又有d|n，因此
d|kn，所以有d|r。即我们由d|m 且d|n 这个前提可以得出d|r。换就话说，我们也可以说成由d|m
个前提推出了d|n 且 d|r。使用类似的推理过程同样可以得到：由d|n 且 d|r 可以推出d|m 且 d|n
这里的r 即 m mod n。因此我们可以说gcd(m, n) = gcd(n, m mod n)是双向成立的，命题得证。

1. What does Euclid's algorithm do for a pair of numbers in which the first number is smaller than the second one? What is the largest number of times this can happen during the algorithm's execution on such an input?

上机练习

1-1. Computing *gcd(m, n)*

1) Compose a program using Euclid's algorithm
2) Compose a program using Consecutive Integer Algorithm
3) Find gcd(31415, 14142) by applying Euclid's algorithm
4) Estimate how many time faster it will be to find gcd(31415, 14142) by Euclid's algorithm compared with the algorithm based on checking consecutive integers from min{*m*,*n*} down to gcd(*m*,*n*)

1-2. find the binary representation of a positive decimal integer

Compose a program

上机练习

1-3. Element uniqueness problem

1) a) Compose a program using *UniqueElement* algorithm on P63

b) Check its efficiency in worst case, best case, and average case, in your program

2) a) Compose a program using the method in which the array is sorted firstly

b) Check its efficiency in worst case, best case, and average case, in your program

代码要求

1-1. GCD (1)

```
int gcd_Euclid (int m, int n)
{  // computes the greatest common divisor of two integers m and n
      using Euclid algorithm;
   // Input: two integers;
   // Output: their GCD


}
```

假定m, n 都是自然数，使用欧几里德算法求m, n的最大公约数，作为返回值;

注意：特殊情况 $m < n$

代码要求

1-1. GCD (2)

```
int gcd_ConsecutInteger (int m, int n)
{  // computes the greatest common divisor of two integers m and n
using Consecutive Integer Algorithm ;
    // Input: two integers;
    // Output: their GCD
}
```

假定m, n 都是自然数，使用连续整数递减法求m, n的最大公约数，作为返回值；

注意：特殊情况 $m < n$

代码要求

1-2. binary representation of a positive decimal integer

```
int convert_decimal_to_binary(int dec_number)
{  // find the binary representation of a positive decimal integer;
   //Input: a positive decimal integer;
   //Output: binary integer;
}
```

结果输出到屏幕上（cout，printf），结果输出到一行
注意：要求不可以有前置的0，例如，结果不能是 00001111000
而是 1111000

代码要求

1-3. Element uniqueness problem (1)

```
bool  UniqueElement (int A[0,…, n-1], int size)
 {  // Determines whether all the elements in a given array A are
       distinct using the definition based algorithm;
    // Input: an array A[0,…, n-1]) ;
    // Output: returns true if all the elements are distinct, and false
       otherwise;
}
```

代码要求

1-3. Element uniqueness problem (2)

```
bool  UniqueElement_sort (int A[0,…, n-1], int size)
 {  // Determines whether all the elements in a given array A are
       distinct using sorting firstly;
    // Input: an array A[0,…, n-1]) ;
    // Output: returns true if all the elements are distinct, and false
       otherwise;

}
```

代码要求

提交代码注意：

所有算法写入到一个c/cpp文件中，

文件命名：　学号_姓名_algo_assignment1.c

文档要求

对三个练习分别给出

1. 算法本身的思路，可以用伪代码或者自然语言描述
2. 程序中用于计算算法复杂度的方法，用文字或者公式描述清楚