The Design and Analysis of Algorithm

Lecture 0
Introduction

课程简介

- 算法设计与分析 (Design and Analysis of Algorithms)
- 任课教师: 朱山风 (zhusf@fudan.edu.cn)
- 时间地点： H4301 周三 11~13小节
- TA: 张益松 (15210240031@fudan.edu.cn)
- 教材： Introduction to Algorithms, 2nd Edition, Cormen, T.H., Leiserson, C.E., Rivest, R.L., and C. Stein. MIT Press
- 参考书： Introduction to The Design and Analysis of Algorithms 2nd Edition, Anany Levitin, Addison Wesley
- 授课方式: 3学时/周,
- 课件网址:

Self Introduction

Name 朱山风

Email: zhusf@fudan.edu.cn

Education Background

- **B. Sc. Department of Computer Science, Wuhan University, 1996**
- **M.Phil. Department of Computer Science, Wuhan University, 1999**
- **Ph.D. Department of Computer Science, City University of Hong Kong, 2003**

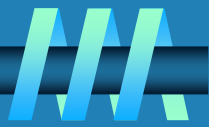
Working Experience

- **2013-2014 Department of CS, UIUC, USA**
- **2004-2008 Bioinformatics Center，Institute for Chemical Research, Kyoto University, Japan**
- **2008,7 School of Computer Science, Fudan University.**

Research Interest

- **Data Mining and Text Mining for Bioinformatics,**
- **Information Retrieval**

Policy and Grade



Policy

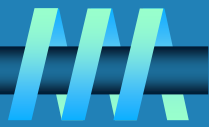
- Please power off the Cell Phone
- Don’t chat with each other unless discussions are required
- Cooperation is encouraged but plagiarism is prohibited

Grade

- Homework 15%
- Mid-term 35%
- Final 50%

You are welcome to give suggestions and feedbacks.

Topics we're going to cover



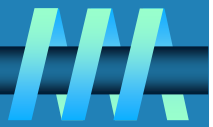
Problem Type

- Sorting Algorithm
- Graph Algorithm
- Hashing
- Network Flow

Design and Analysis

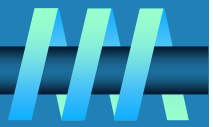
- Analysis framework
- Brute-force algorithm/Exhaustive Search
- Divide and Conquer
- Dynamic programming
- Greedy algorithm

Content



- Why study Algorithms?
- What is an Algorithm?
- An example: greatest common divisor problem
- Fundamentals of Algorithmic Problem Solving
- Problem Types
- Fundamental Data Structures.

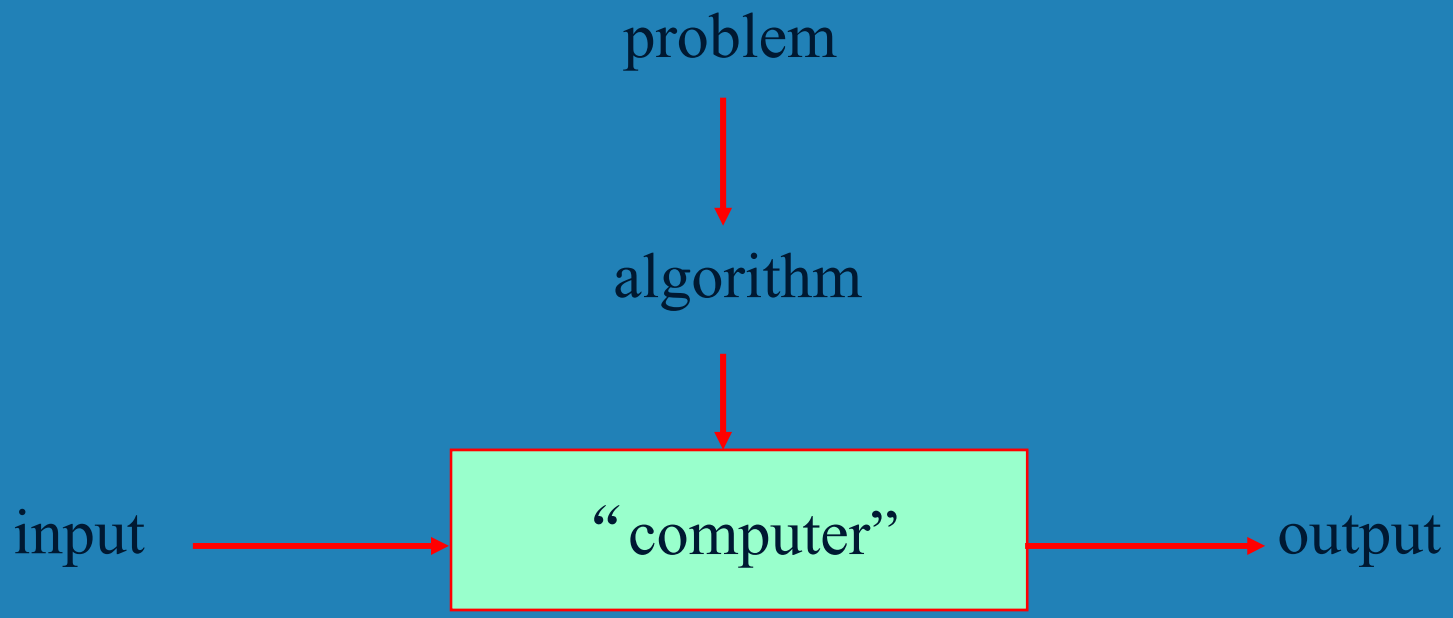
Why study algorithms?



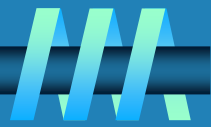
- **Theoretical importance**
  - **the core of computer science**
- **Practical importance**
  - **A practitioner's toolkit of known algorithms**
  - **Framework for designing and analyzing algorithms for new problems**

What is an algorithm?

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.



What is an algorithm?



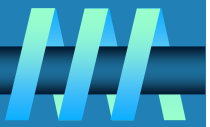
- Recipe, process, method, technique, procedure, routine,… with following requirements:

1. **Finiteness**
   - terminates after a finite number of steps
2. **Definiteness**
   - rigorously and unambiguously specified
3. **Input**
   - valid inputs are clearly specified
4. **Output**
   - can be proved to produce the correct output given a valid input
5. **Effectiveness**
   - steps are sufficiently simple and basic

Example of computational problem: sorting

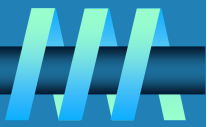
- **Statement of problem:**
  - ***Input:*** **A sequence of *n* numbers $\langle a_1, a_2, \ldots, a_n \rangle$**
  - ***Output:*** **A reordering of the input sequence $\langle a'_1, a'_2, \ldots, a'_n \rangle$ so that $a'_i \leq a'_j$ whenever $i < j$**
- **Instance: The sequence <5, 3, 2, 8, 3>**
- **Algorithms:**
  - **Selection sort**
  - **Insertion sort**
  - **Merge sort**
  - **(many others)**

Some Well-known Computational Problems



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

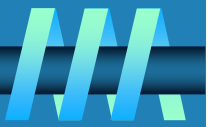
The greatest common divisor problem



Problem: Find gcd(*m*,*n*), the greatest common divisor of two nonnegative, not both zero integers *m* and *n*

Examples: gcd(60,24) = 12, gcd(60,0) = 60, gcd(0,0) = ?

Method 1 for computing gcd(*m*,*n*)



Consecutive integer checking algorithm

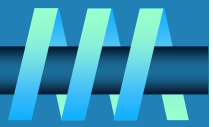
Step 1 Assign the value of min{*m*,*n*} to *t*

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

Method2 for gcd(*m*,*n*) [cont.]



Middle-school procedure

Step 1 Find the prime factorization of *m*

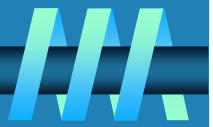
Step 2 Find the prime factorization of *n*

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*)

Is this an algorithm?

Sieve of Eratosthenes



Input: Integer $n \geq 2$

Output: List of primes less than or equal to n

for $p \leftarrow 2$ **to** n **do** $A[p] \leftarrow p$

for $p \leftarrow 2$ **to** $\lfloor n \rfloor$ **do**

    if $A[p] \neq 0$ //p hasn't been previously eliminated from the list

        $j \leftarrow p * p$

        while $j \leq n$ **do**

            $A[j] \leftarrow 0$ //mark element as eliminated

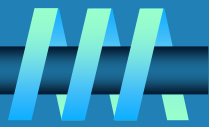
            $j \leftarrow j + p$

Example: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

ALGORITHM *Sieve*(n)

```
//Implements the sieve of Eratosthenes
//Input: An integer n ≥ 2
//Output: Array L of all prime numbers less than or equal to n
for p ← 2 to n do A[p] ← p
for p ← 2 to ⌊√n⌋ do    //see note before pseudocode
    if A[p] ≠ 0    //p hasn't been eliminated on previous passes
        j ← p * p
        while j ≤ n do
            A[j] ← 0    //mark element as eliminated
            j ← j + p
//copy the remaining elements of A to array L of the primes
i ← 0
for p ← 2 to n do
    if A[p] ≠ 0
        L[i] ← A[p]
        i ← i + 1
return L
```

Euclid's Algorithm



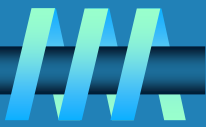
Euclid's algorithm is based on repeated application of equality

$$\gcd(m,n) = \gcd(n, m \bmod n)$$

until the second number becomes 0, which makes the problem trivial.

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

Two descriptions of Euclid's algorithm



Step 1 If $n = 0$, return m and stop; otherwise go to Step 2

Step 2 Divide m by n and assign the value fo the remainder to r

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

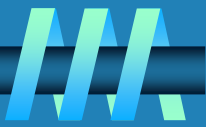
while $n \neq 0$ **do**

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return m



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

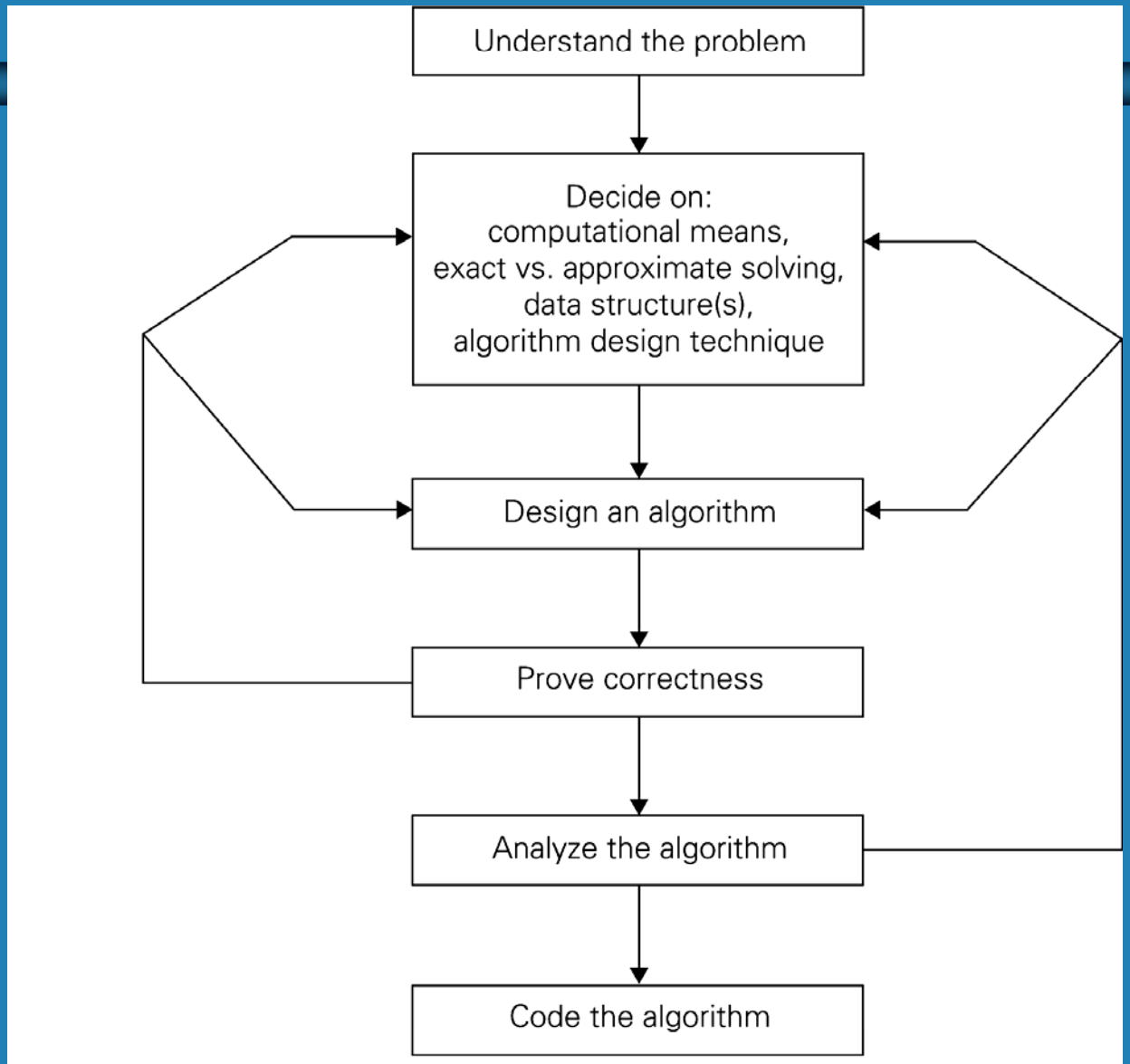
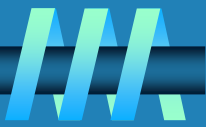


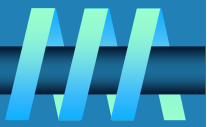
FIGURE 1.2 Algorithm design and analysis process

Fundamentals of Algorithmic Problem Solving



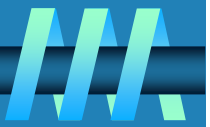
- How to design algorithms
- How to express algorithms
- Proving correctness
- Efficiency
  - Theoretical analysis
  - Empirical analysis
- Optimality

Algorithm design strategies



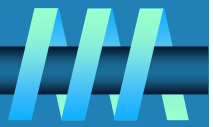
- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Greedy approach
- Dynamic programming
- Backtracking and Branch and bound
- Space and time tradeoffs

Analysis of Algorithms



- How good is the algorithm?
  - Correctness
  - Time efficiency
  - Space efficiency
- Does there exist a better algorithm?
  - Lower bounds
  - Optimality

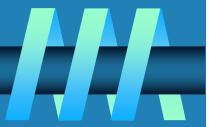
Two main issues related to algorithms



- How to design algorithms?

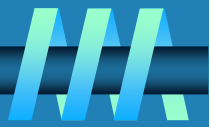
- How to analyze algorithm efficiency?

Important problem types



- sorting
- searching
- string processing
- graph problems
- combinatorial problems
- geometric problems
- numerical problems

Fundamental data structures



- list
  - array
  - linked list
  - string
- stack
- queue
- priority queue
- graph
- tree
- set and dictionary

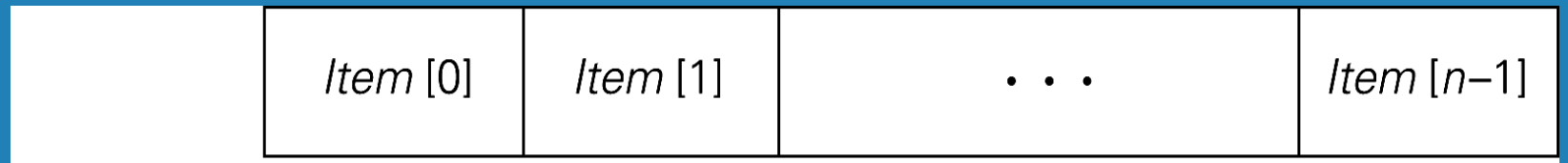


FIGURE 1.3 Array of n elements

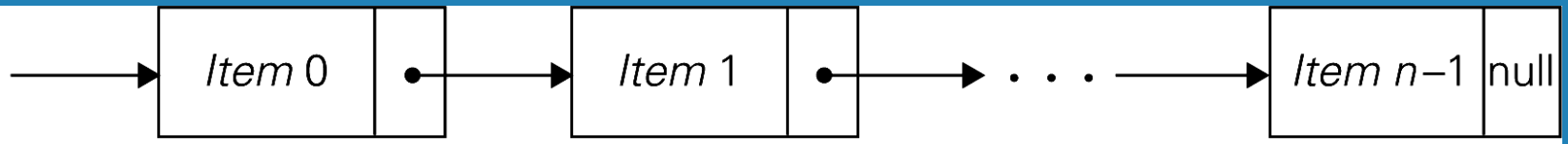


FIGURE 1.4 Singly linked list of n elements

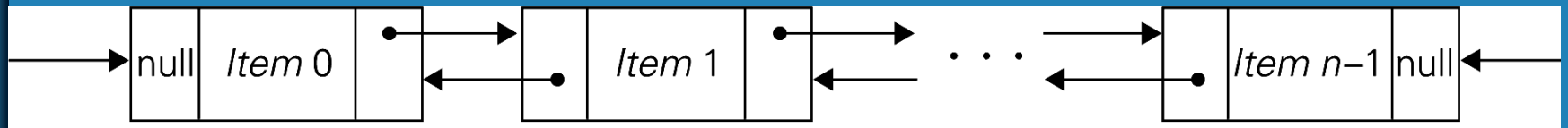


FIGURE 1.5 Doubly linked list of n elements

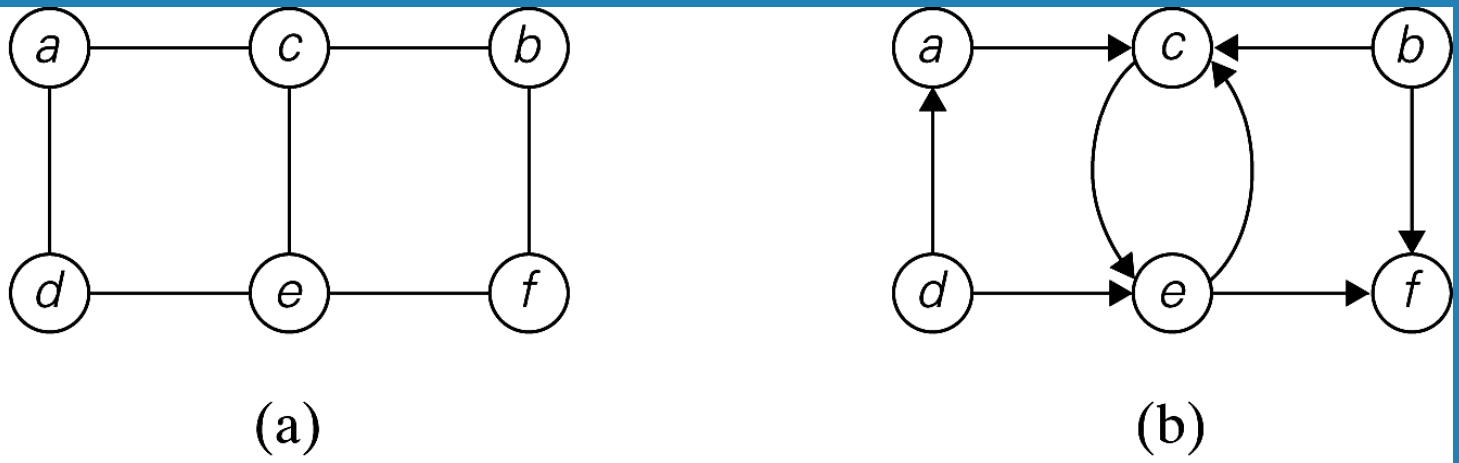


FIGURE 1.6 (a) Undirected graph. (b) Digraph.

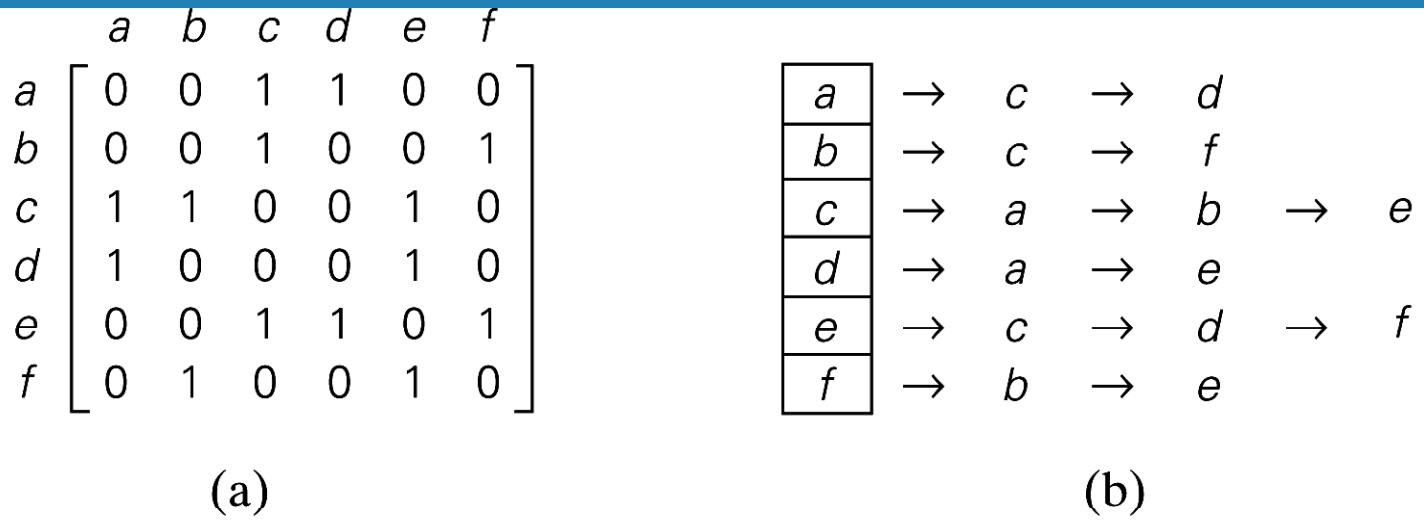
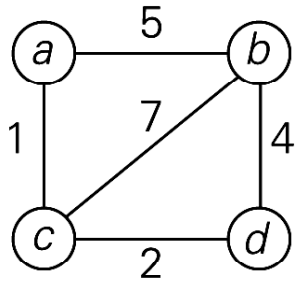


FIGURE 1.7 (a) Adjacency matrix and (b) adjacency lists of the graph in Figure 1.6a



(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)

FIGURE 1.8 (a) Weighted graph. (b) Its weight matrix. (c) Its adjacency lists.

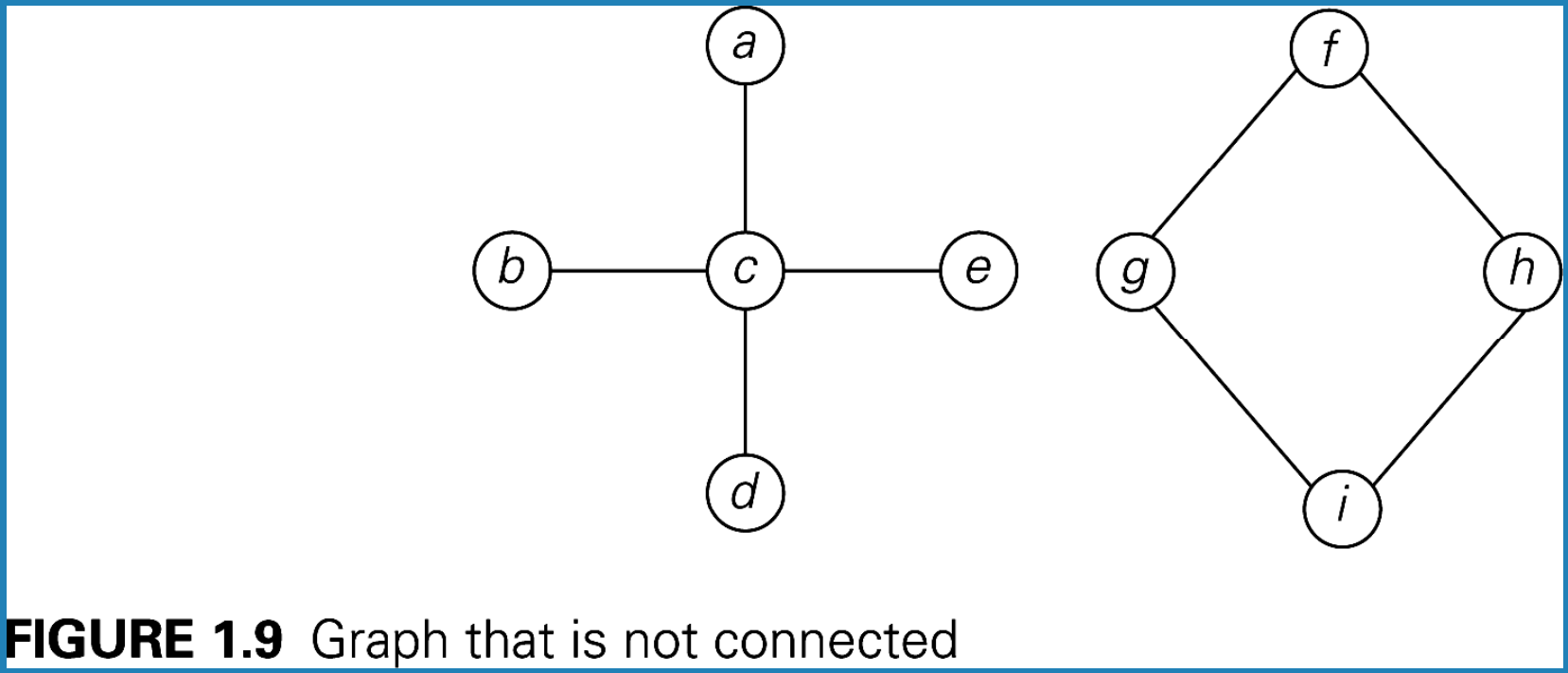


FIGURE 1.9 Graph that is not connected

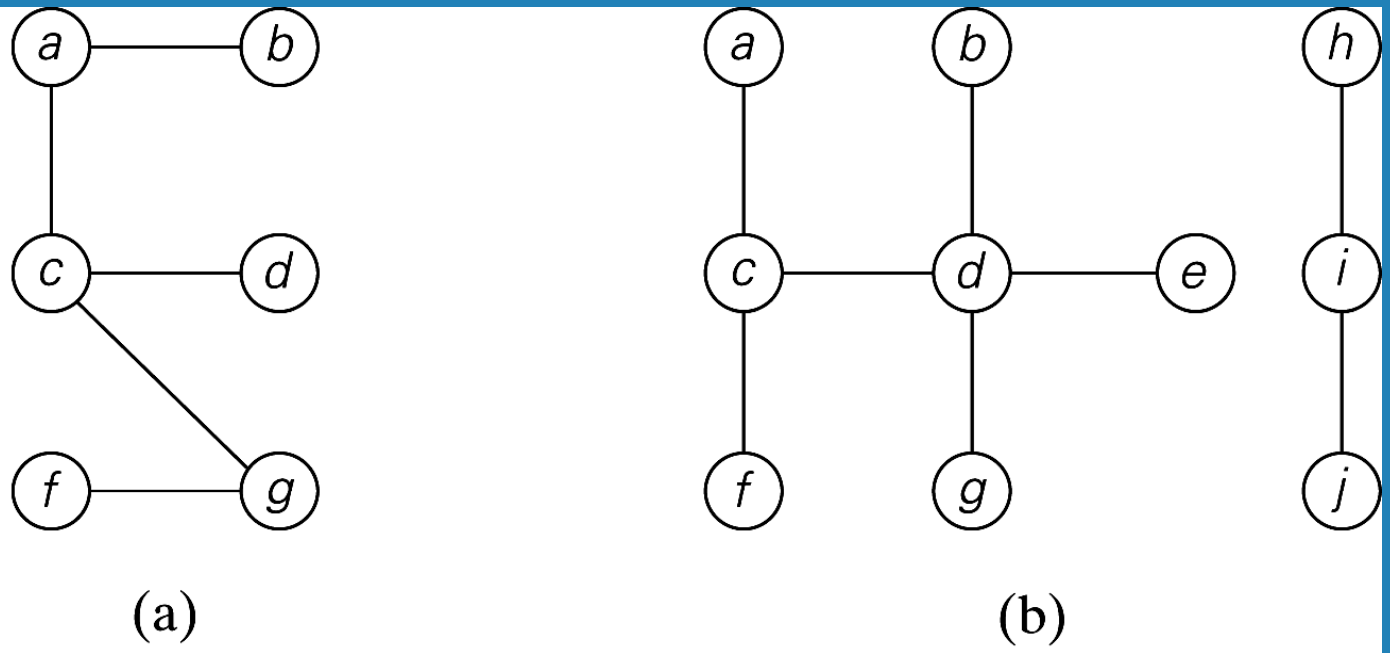


FIGURE 1.10 (a) Tree. (b) Forest.

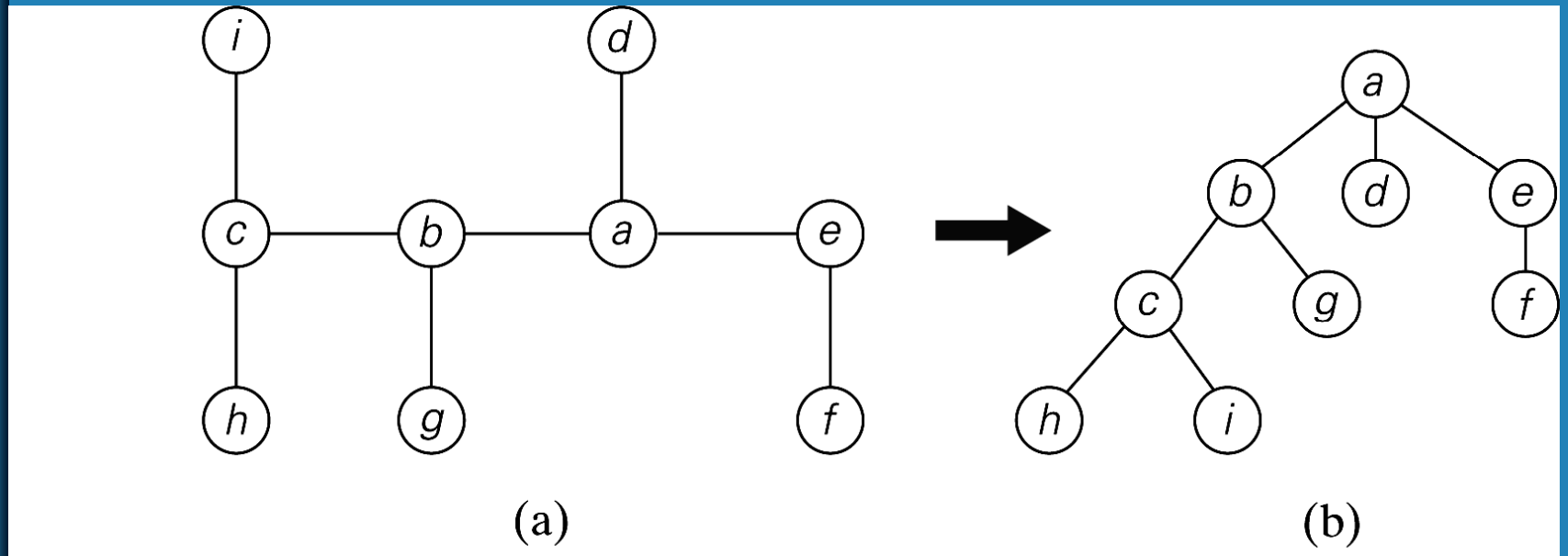


FIGURE 1.11 (a) Free tree. (b) Its transformation into a rooted tree.

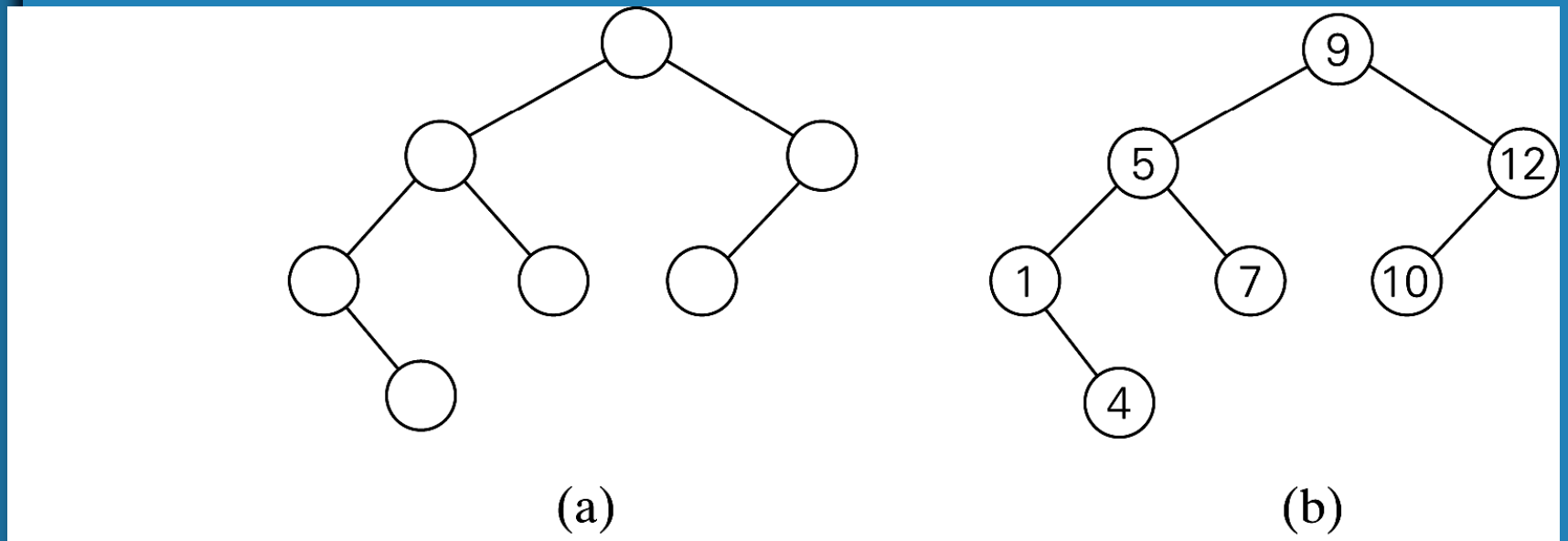


FIGURE 1.12 (a) Binary tree. (b) Binary search tree.

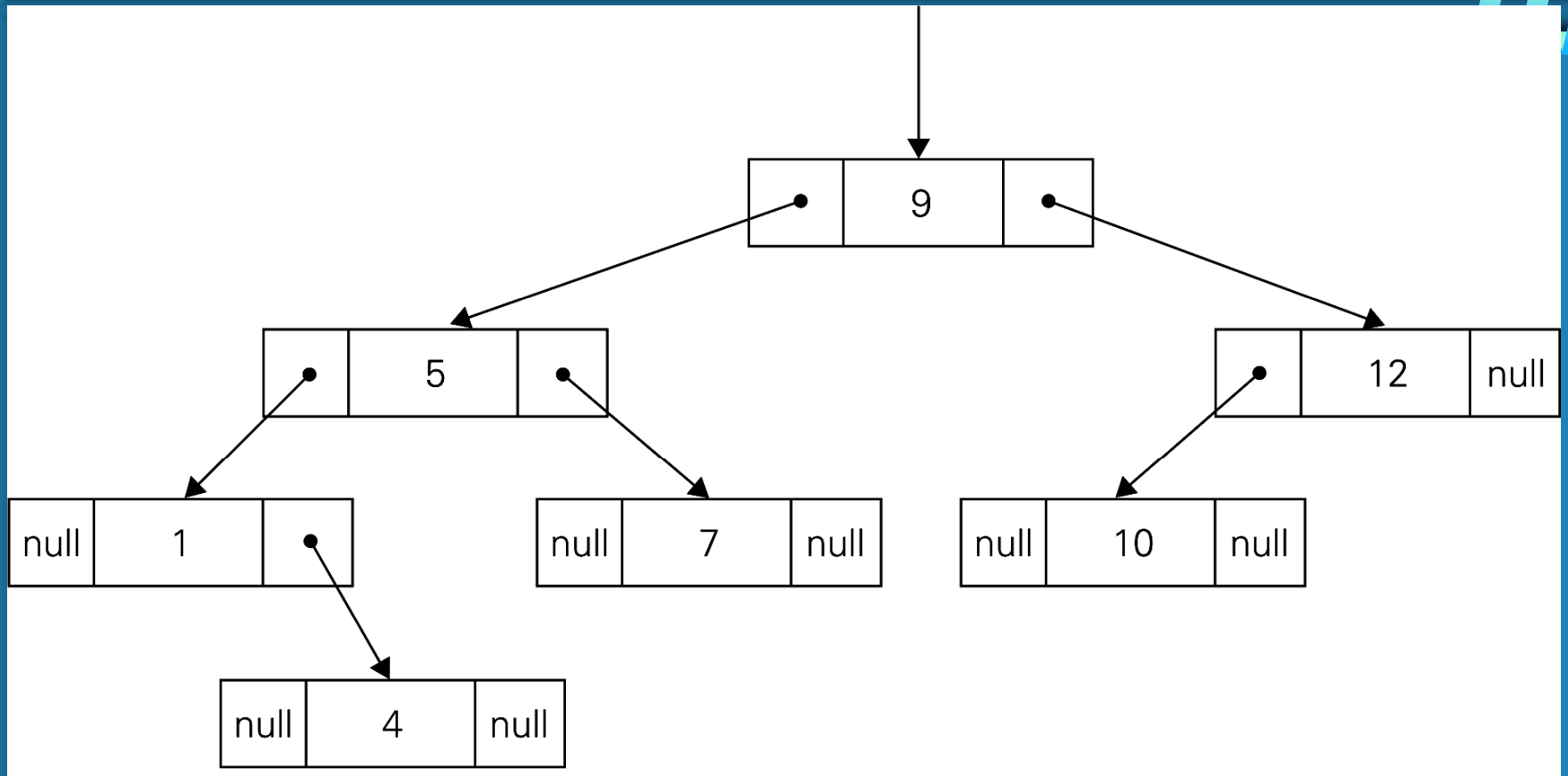
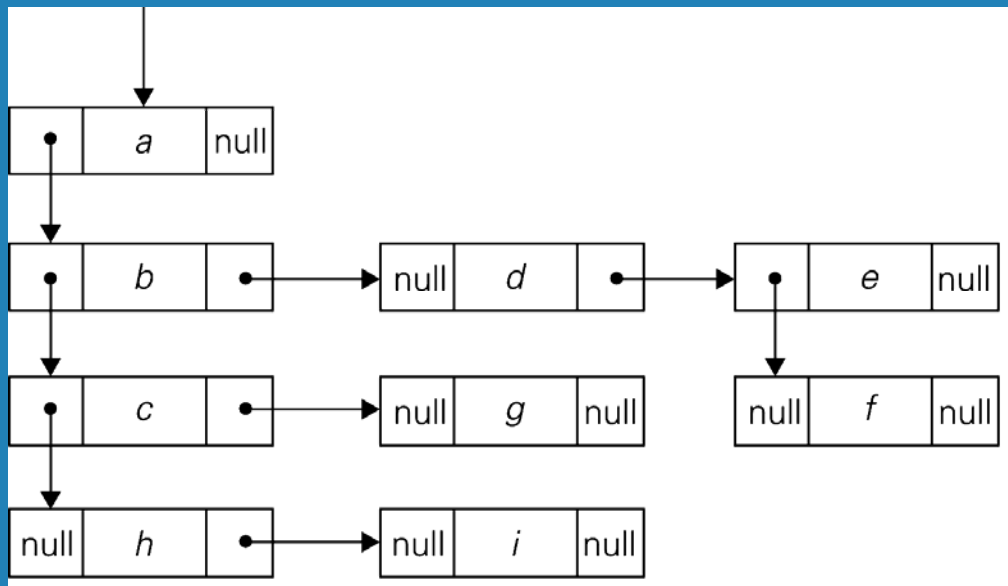
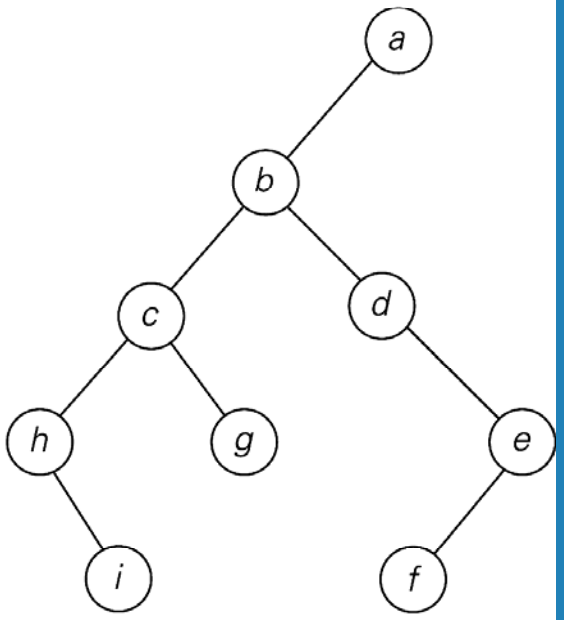


FIGURE 1.13 Standard implementation of the binary search tree in Figure 1.12b



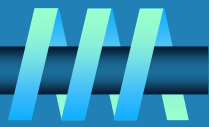
(a)



(b)

FIGURE 1.14 (a) First child–next sibling representation of the graph in Figure 1.11b. (b) Its binary tree representation.

Summarization



- Why study Algorithms?
- What is an Algorithm?
- An example: greatest common divisor problem
- Fundamentals of Algorithmic Problem Solving
- Problem Types
- Fundamental Data Structures.