

CSC 411

Design and Analysis of Algorithm

Instructor: Minhee Jun
junm@cua.edu

Course Overview

- This course presents
 - Fundamental techniques for analyzing the complexity of algorithms.
 - Fundamental techniques for designing efficient computer algorithms, providing their correctness, and analyzing their complexity.
- **Prerequisites:**
 - CSC 280 – Data Structure
 - CSC 210 – Discrete Math

Course Overview

- General topics include
 - methods for expressing and comparing complexity of algorithms: worst and average cases
 - sorting, selection, and graph algorithms
 - basic algorithm design techniques such as divide-and-conquer, greedy method, backtracking, and dynamic programming.

Course Objective

- Upon completion of this course, students will be able to do the following:
 - Analyze asymptotic runtime complexity of algorithms.
 - Demonstrate a familiarity with major algorithms and data structures.
 - Understand and design algorithms using greedy strategy, divide and conquer approach, and dynamic programming etc.
 - Apply important algorithmic design paradigms and methods of analysis.

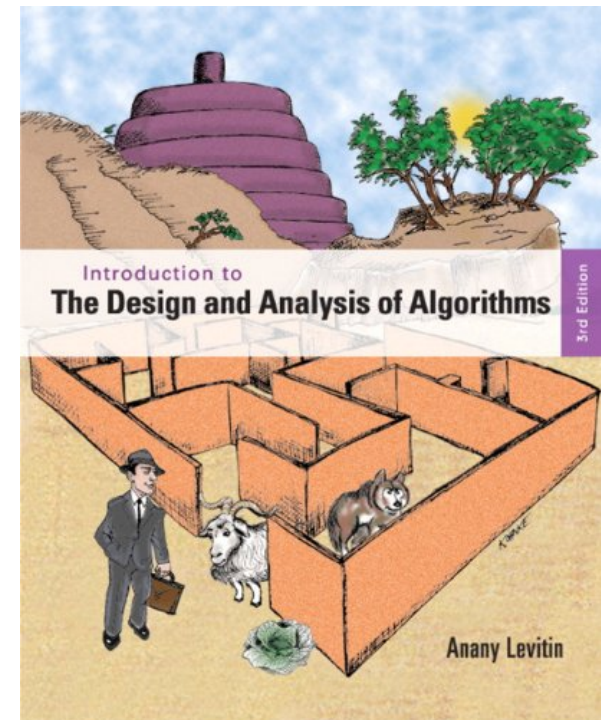
Textbooks

- **Introduction to the Design and Analysis of Algorithms** by Anany V. Levitin

Publisher: Addison Wesley; 3rd edition (October 9, 2011)

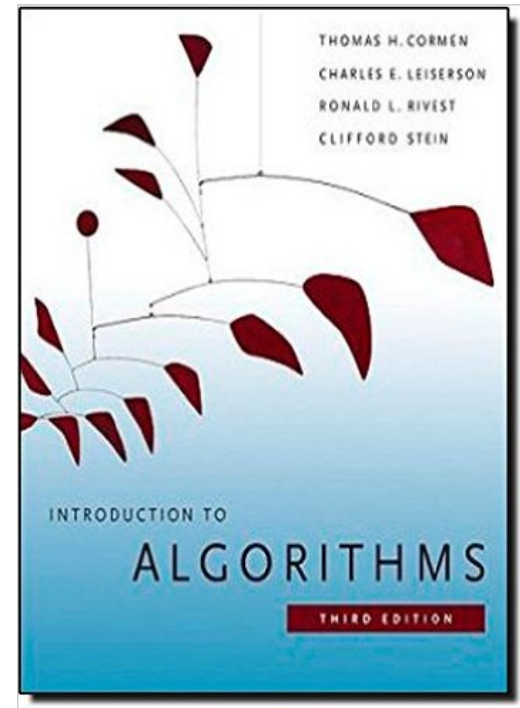
ISBN-10: 0132316811

ISBN-13: 978-0132316811



Textbooks

- **Introduction to Algorithms** by Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L.
Publisher: The MIT Press; 3rd edition (July 31, 2009)
ISBN-10: 0262033844
ISBN-13: 978-0262033848



Homework

- Homework is important.
- Working on your homework is critical to doing well on the quizzes and midterm/final exams.
- The homework assignments will NOT be graded only on correctness. The instructor will record your efforts and the number of problems that were "completed" (no matter it is correctly or incorrectly).
- The solutions to the homework will be made available on Black Board after the homework due date.

Homework

- **Late work**

- Homework is due at the beginning of class on the due date. Late work will be accepted with penalty before the solution is posted.

- **Honor Code**

- Collaboration on homework is permitted; copying of solutions or code is not. If you work with other students on your homework, please indicate the name(s) of your collaborator(s) on your homework.

Class Participation

- Requirements include class attendance, active participation in discussions and in-class problem solving.
- Please let the instructor know in advance if you expect to be absent for any reason.
- If you must miss a class due to an emergency or documented illness, email the instructor as soon as possible. You are still responsible for any material covered, assignments given, and homework due during the missed classes unless pre-approved by the instructor.

Feedback

- I like your feedback on how the course is going and how it could be improved.
- I'd like to know what you are thinking during the semester, not only at the end of semester.
- You are welcome to let me know your suggestions in the class, and if you do not feel comfortable with letting me know who made the suggestions, you are welcome to drop me a note in my mailbox or office.

Cell Phone and Computer

- A cell phone or computer can easily turn from “classroom learning tool” into “classroom disruption”.
- **No** cell phone and computer is allowed during our class.

Chapter 1

Introduction to Computer Algorithms

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.
 - Procedural solutions to problems

Notion of algorithm

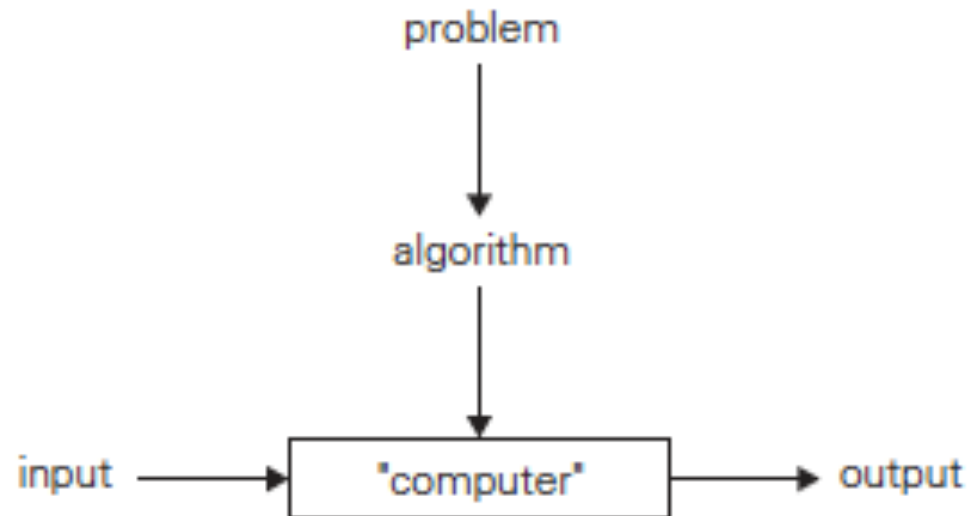


FIGURE 1.1 The notion of the algorithm.

What is an algorithm?

- Recipe, process, method, technique, procedure, routine, ... with following requirements:
 - Finiteness
 - terminates after a finite number of steps
 - Definiteness
 - rigorously and unambiguously specified Input valid inputs are clearly specified
 - Output
 - can be proved to produce the correct output given a valid input
 - Effectiveness
 - steps are sufficiently simple and basic

Example 1: Help Me Cross the River

- Sailor Cat needs to bring a wolf, a goat, and a cabbage across the river.
- The boat is tiny and can only carry one passenger at a time.
- If he leaves the wolf and the goat alone together, the wolf will eat the goat.
- If he leaves the goat and the cabbage alone together, the goat will eat the cabbage.
- Problem:
 - How can he bring all three safely across the river?

Solution:

1. **CROSS THE RIVER WITH THE GOAT. LEAVE THE GOAT ON THE OTHER SIDE.**
2. **CROSS BACK AND GET THE CABBAGE. DROP THE CABBAGE ON THE OTHER SIDE AND TAKE THE GOAT WITH YOU.**
3. **SWITCH THE GOAT WITH THE WOLF AND DROP THE WOLF WITH THE CABBAGE**
4. **THEN GO BACK TO GET THE GOAT.**

algorithm

a sequence of unambiguous
instructions to solve the problem

Example 2: Euclid's algorithm

- Euclid's algorithm for finding the greatest common divisor
- **Problem:** $\text{gcd}(12, 3)=?$ $\text{gcd}(10, 15)=?$
- Can you find a solution?
 - Sure, the answer is 3 and 5.
- Can you write the code to take any 2 integers as inputs, and output the gcd?
 - Before you write the code, can you describe your solution in a step by step manner?

Euclid's Algorithm

- **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
 - $\gcd(k, 0) = ?$ k
 - $\gcd(0,0) = ?$ undefined

Euclid's Algorithm

- Solution1:
 - $\text{gcd}(m, n) = \text{gcd}(n, m \bmod n)$, assuming $m \geq n$
- Euclid's algorithm is based on *repeated* application of equality: $\text{gcd}(m, n) = \text{gcd}(n, m \bmod n)$, *until the second number becomes 0*, which makes the problem trivial.
- How do you write the algorithm?
 - Step by step instructions to solve the problem

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 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.

```
while  $n \neq 0$  do  
     $r \leftarrow m \bmod n$   
     $m \leftarrow n$   
     $n \leftarrow r$   
return  $m$ 
```

Any Other Solutions?

- **Problem:** Find $\text{gcd}(m, n)$, the greatest common divisor of two nonnegative, not both zero integers m and n
- An algorithm designer always ask
 - Is there *another* solution?
 - Is there a *better* solution?
 - Less time
 - Less memory
 - More robust

Example 3: New World puzzle

- There are four people who want to cross a rickety bridge; they all begin on the same side. You have 17 minutes to get them all across to the other side. It is night, and they have one flashlight. A maximum of two people can cross the bridge at one time. Any party that crosses, either one or two people, must have the flashlight with them. The flashlight must be walked back and forth; it cannot be thrown, for example.
- Person 1 takes 1 minute to cross the bridge, person 2 takes 2 minutes, person 3 takes 5 minutes, and person 4 takes 10 minutes.
- A pair must walk together at the rate of the slower person's pace. (Note: According to a rumor on the Internet, interviewers at a well-known software company located near Seattle have given this problem to interviewees.)

Solution:

Let 1, 2, 5, 10 be labels representing the men of the problem, f represent the flashlight's location, and the number in the parenthesis be the total amount of time elapsed. The following sequence of moves solves the problem:

$$\begin{array}{c} \hline (0) \\ \hline f,1,2,5,10 \end{array} \quad \begin{array}{c} f,1,2 \\ \hline (2) \\ \hline 5,10 \end{array} \quad \begin{array}{c} 2 \\ \hline (3) \\ \hline f,1,5,10 \end{array} \quad \begin{array}{c} f,2,5,10 \\ \hline (13) \\ \hline 1 \end{array} \quad \begin{array}{c} 5,10 \\ \hline (15) \\ \hline f,1,2 \end{array} \quad \begin{array}{c} f,1,2,5,10 \\ \hline (17) \\ \hline \end{array}$$

Algorithm:

Problem:

- Input: a list $\langle a \rangle$ of crossing times for n people
 - Numbered $t_1, t_2, t_3, \dots, t_n$
- Output: total time to cross

Solution:

- Strategy?
 - use 1 & 2 as shuttles and send the others across in pairs.
 - Describe your solution in a step-by-step instruction.

1.2 Algorithm design and analysis process

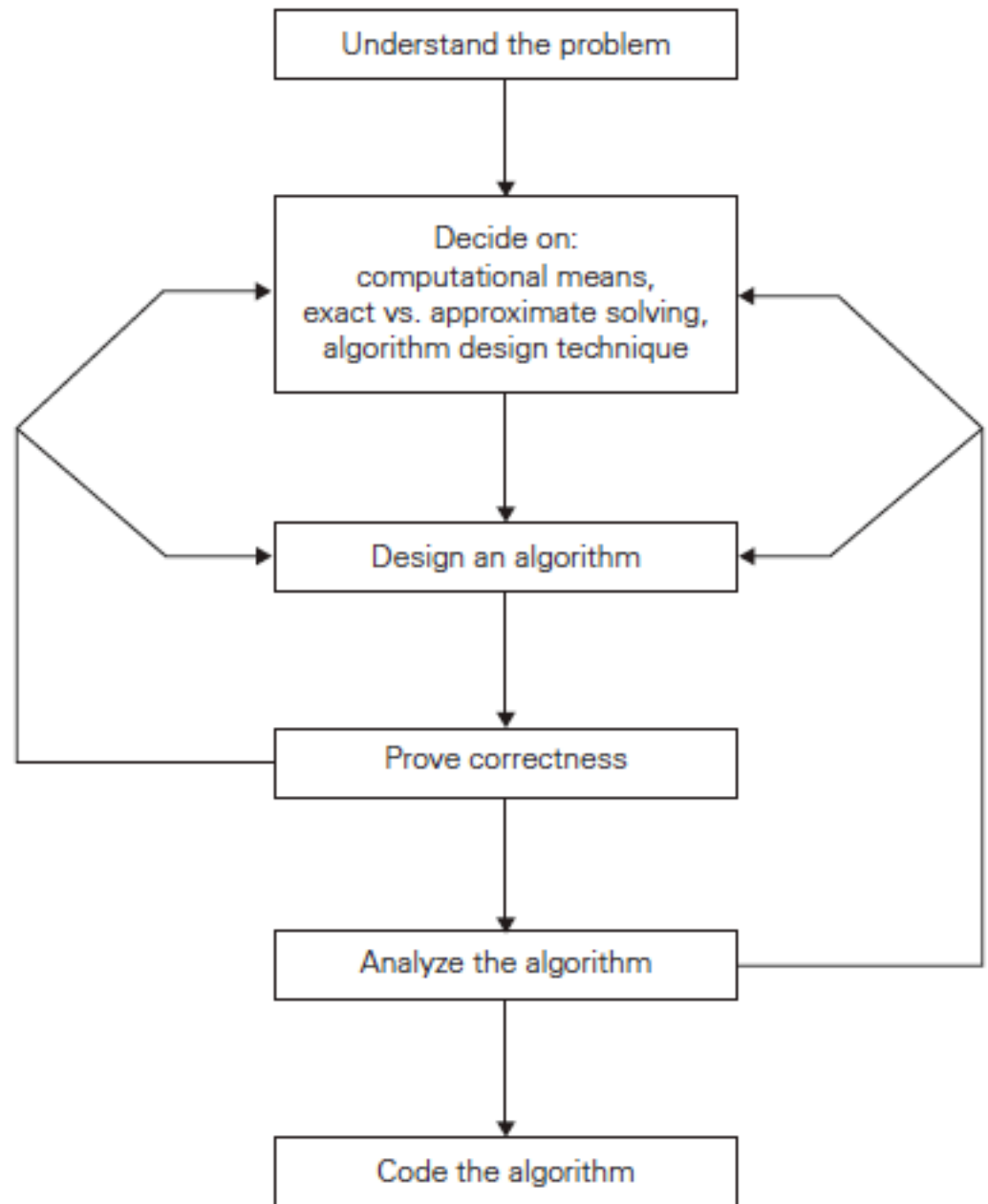


FIGURE 1.2 Algorithm design and analysis process.

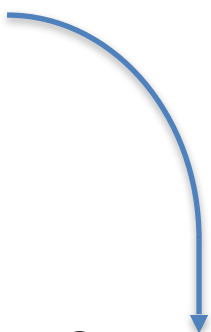
Correctness

- How do you prove the correctness of an algorithm?
 - A common technique is to use **mathematical induction**.
 - Direct prove by using some definition
 - Indirect Argument: contradiction and contraposition
- How do you prove the incorrectness of an algorithm?
 - You just need **one instance** of its input for which the algorithm fails. I.e., given a counter example.

Analyzing an Algorithm

- Efficiency
 - Time efficiency
 - How fast the algorithm run?
 - The Big-O notation
 - Space efficiency
 - How much extra memory the algorithm needs?
- Simplicity
 - Easier to understand and easier to program
 - Fewer bug
- Generality
 - Generality of the problems the algorithm solves
 - The set of inputs it accepts

Analysis of Algorithms

- How **good** is the algorithm?
 - Correctness
 - Efficiency (Time, Space)
 - Simplicity
 - Generality
 - Does there exist a better algorithm?
 - Lower bounds
 - Optimality
- 

What you will learn in this course?

Two main issues related to algorithms

- How to design algorithms
- How to analyze algorithm efficiency

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

1.3 Important Problem Types

- Sorting
 - Rearrange the items of a given list in nondecreasing order
- Searching
 - Deals with finding a given value, called a search key
 - Example: straightforward sequential search
- String Processing
 - String: a sequence of characters from an alphabet
 - String Matching

1.3 Important Problem Types

- Graph Problems
 - Traveling salesman problem: finding the shortest tour through n cities that visits every city exactly once
: how can one reach all the points in a network?
 - Shortest path problem: what is the best route between two cities?
 - Graph-colouring problem: assign the smallest number of colors to the vertices of a graph so that no two adjacent vertices are the same color

1.3 Important Problem Types

- Combinatorial Problems
 - Find a combinatorial object - permutation, combination, subset - subject to constraints
 - Example: traveling salesman problem, graph-coloring problem
- Geometric Problems
 - Closest-Pair: given n points in the plane, find the closest pair among them
 - Convex-Hull: find the smallest convex polygon that would include all the points of a given set
- Numerical Problems
 - Solving systems of equations, computing definite integrals, evaluating functions, etc.

Sorting Example

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

Algorithm Design Technique

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

Algorithm Design Strategies

- Brute force
 - A straightforward approach to solving a problem, usually directly based on the problem's statement
- Divide and conquer
 - Divide a problem into smaller instances, solve smaller instances (perhaps recursively), combine
- Decrease and conquer
 - Exploit relationship between the problem and a smaller instance reduced by some factor (often 1)
- Transform and conquer
 - Transform the problem to a simpler instance, another representation or an instance with a known solution

More Algorithm Design Strategies

- Greedy approach
 - Make locally optimal steps which (hopefully) lead to a globally optimal solution for an optimization problem
- Dynamic programming
 - Technique for solving problems with overlapping sub-domains
- Backtracking and Branch and bound
 - A way of tackling difficult optimization and combinatorial problems without exploring all state-space
- Space and time tradeoffs
 - Preprocess the input and store additional information to accelerate solving the problem

1.4 Fundamental data structures

- Linear Data Structure
 - array
 - linked list
 - string
- Special Type of List
 - stack
 - queue
 - priority queue
- Graphs
 - weighted graphs
 - paths and cycles
- Trees
 - rooted trees
 - ordered trees
- Set and Dictionaries

Linear Data Structure



FIGURE 1.3 Array of n elements.

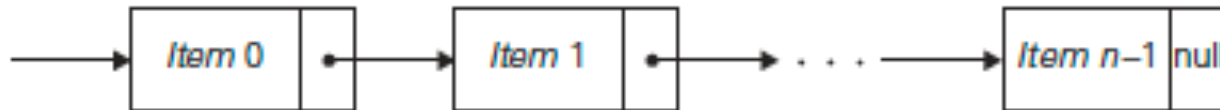


FIGURE 1.4 Singly linked list of n elements.

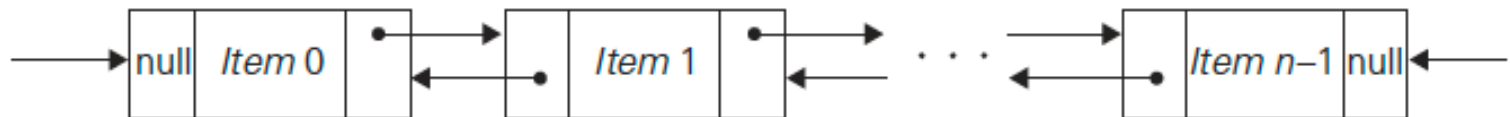


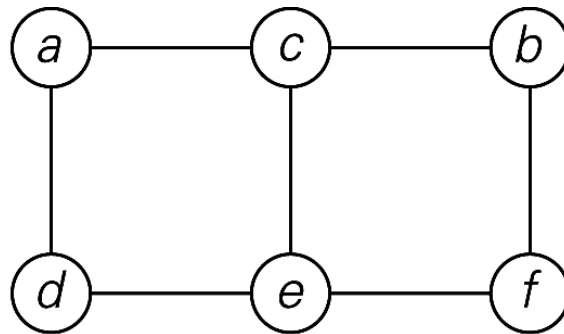
FIGURE 1.5 Doubly linked list of n elements.

Special Type of List

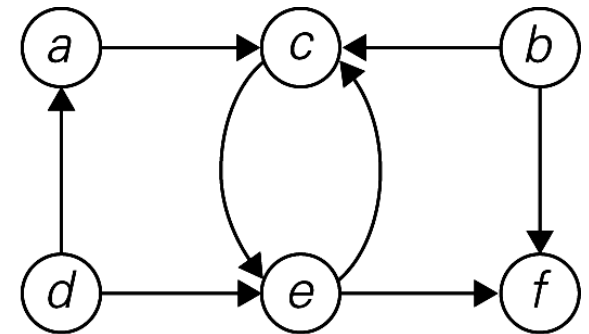
- Stack
 - Insertions and deletions can be done only at the end of the structure (called the **top**)
 - Last-in-first-out (LIFO)
- Queue
 - Elements are deleted from one end of the structure (called the **front**), and new elements are added to the other end (called the **rear**)
 - First-in-first-out (FIFO)
- Priority Queue
 - A data structure that seeks to satisfy the needs of applications
 - Example: finding and deleting the largest element and, adding a new element

Graphs

- A graph $G = (V, E)$
- Vertices or nodes V
- Edges E



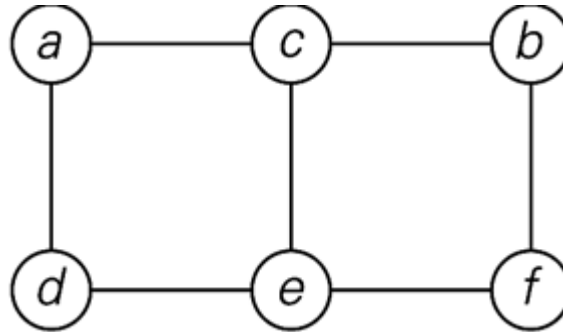
(a)



(b)

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

Graph Representation: Adjacency matrix & Adjacency list



	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
<i>a</i>	0	0	1	1	0	0
<i>b</i>	0	0	1	0	0	1
<i>c</i>	1	1	0	0	1	0
<i>d</i>	1	0	0	0	1	0
<i>e</i>	0	0	1	1	0	1
<i>f</i>	0	1	0	0	1	0

(a)

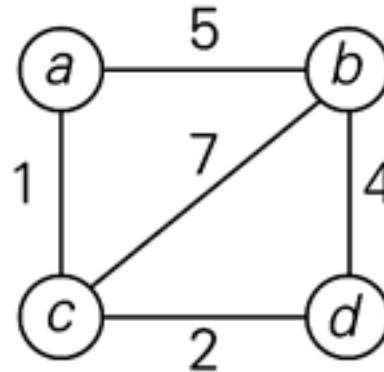
<i>a</i>	→	<i>c</i>	→	<i>d</i>	
<i>b</i>	→	<i>c</i>	→	<i>f</i>	
<i>c</i>	→	<i>a</i>	→	<i>b</i>	→ <i>e</i>
<i>d</i>	→	<i>a</i>	→	<i>e</i>	
<i>e</i>	→	<i>c</i>	→	<i>d</i>	→ <i>f</i>
<i>f</i>	→	<i>b</i>	→	<i>e</i>	

(b)

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

Weighted Graphs

- A simple weighted graph



	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>a</i>	∞	5	1	∞
<i>b</i>	5	∞	7	4
<i>c</i>	1	7	∞	2
<i>d</i>	∞	4	2	∞

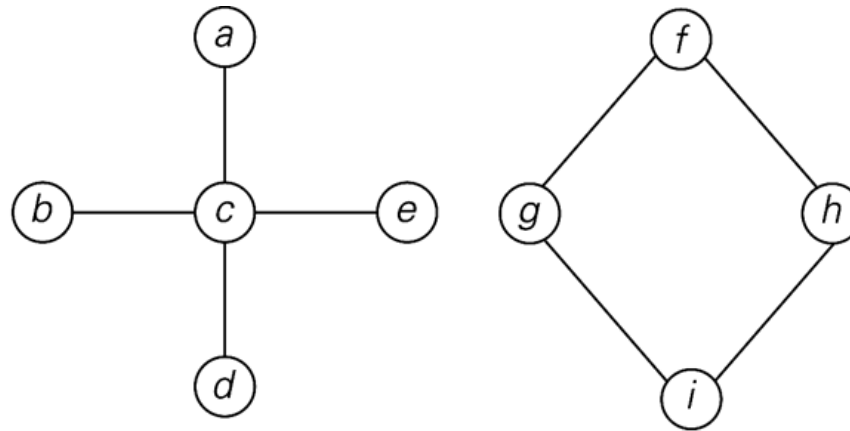
weight matrix

<i>a</i>	$\rightarrow b, 5 \rightarrow c, 1$
<i>b</i>	$\rightarrow a, 5 \rightarrow c, 7 \rightarrow d, 4$
<i>c</i>	$\rightarrow a, 1 \rightarrow b, 7 \rightarrow d, 2$
<i>d</i>	$\rightarrow b, 4 \rightarrow c, 2$

weighted adjacency list

Paths and Cycles

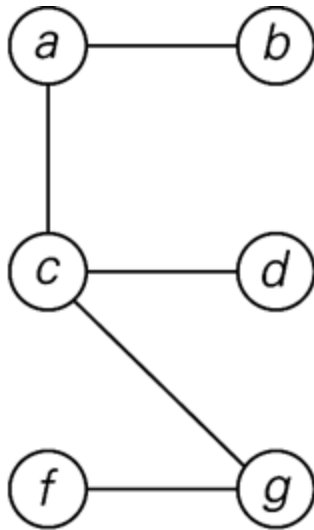
- Is this graph?



- Connectivity
 - A graph is **connected** if for every pair of vertices *u* and *v*, there is a path from *u* to *v*
- Acyclicity
 - A **cycle** is a path of a positive length that starts and ends at the same vertex and does not traverse the same edge more than once
 - A graph is **acyclic** if a graph has no cycles

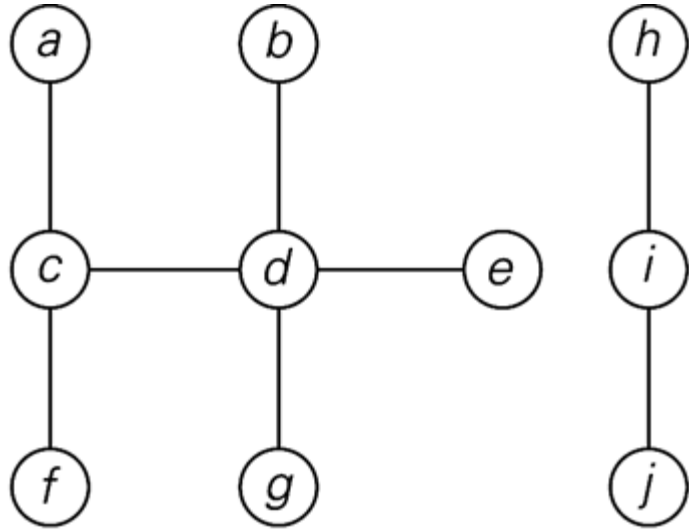
Trees

$$|E| = |V| - 1$$



Tree

$$|E| < |V| - 1$$



Forest

- Tree: a connected acyclic graph
- Forest: a graph the is acyclic but is not necessarily connected

Rooted Trees

- Place its root on the top (level 0), the vertices adjacent to the root below it (level 1), ... the vertices k edges apart from the root below (level k)

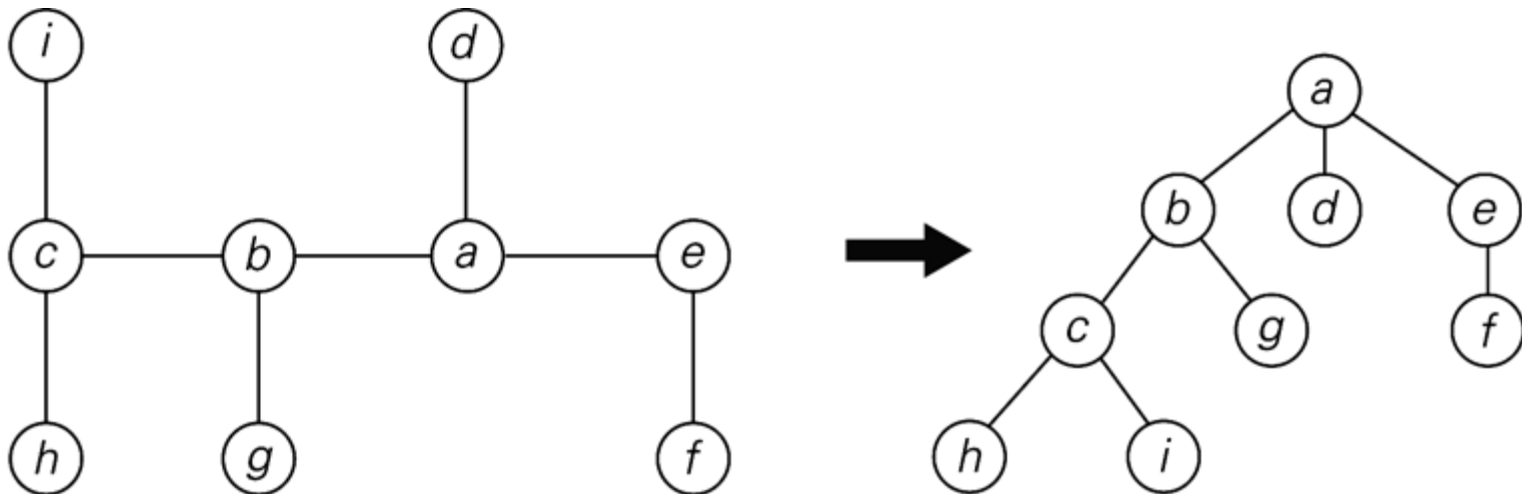


FIGURE 1.11 (a) Free tree. (b) Its transformation into a rooted tree.