Algorithm Design and Analysis

姜海涛

School of Computer Science and Technology

Shandong University

htjiang@sdu.edu.cn

Text Book & Reference Books

- Text Book:
  - Introduction to Algorithms (Third Edition)
    - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
    - The MIT Press (Higher Education Press)
    - Chapter 22-26

- Reference Books
  - 《图算法》：马绍汉编著，山东大学出版社。

Why Study this Course?

- Closely related to our lives(打酒,货郎).
- Help to learn how others analyze and solve problems.
- Help to develop the ability of analyzing and solving problems via computers.
- Very interesting if you can concentrate on this course.

Prerequisite

- Data Structure(二维数组,链表,队列,堆栈)
- C/C++, Java or other programming languages
- A little mathematics(初等数学)

Ready? Let’s begin ……

Lecture 1
Time Complexity and Graph Basis

- Time Complexity
- Graph Basis

What is a Problem?

- A general question to be answered, usually possessing several parameters or free variables, whose value are left to be specified.
- Described by giving:
  - Input(instance): A general description of all its parameters
  - Output(question): A statement of what properties the answer is required to satisfy.

What is an Algorithm?

- Power(x, n)
  - Input: $x \in \mathrm{R}$, $n \in \mathrm{N}^+$
  - Output: $x^n = ?$

$$x^{10} = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x$$

$$x^{10} = ((x^2)^2 \cdot x)^2$$

- An algorithm is a detailed step-by-step method for solving a problem.

Two Algorithms

- Exponentiation(求幂运算):
  - Input: $x \in R$, $n \in N^+$.
  - Question: x^n

```
Algorithm 1-power($x$, $n$)
$y$=$x$;
For $i$ =2 to $n$
    $y$=$y$*$x$;
End For
Return $y$
```

```
Algorithm 2-power($x$, $n$)
//input $x$, $n = (b_{m-1}, b_{m-2}, b_{m-3}, \ldots, b_1, b_0)_2$,
//where $b_{m-1}$=1, m≥2
$y$=$x$;
For $i$ =$m$-2 to 0
    $y$=$y$*$y$;
    if $b_i$=1 then $y$=$y$*$x$;
End For
Return $y$
```

How to Evaluate an Algorithm?

- The Time Complexity
  - Polynomial Time Problem
    (all the problems in this course)
    - How to search a graph?
    - Minimum Spanning Tree Problem
    - Shortest Path Problem
    - Maximum Flow Problem
  - NP-hard Problem
    - Fixed Parameter Tractable
    - W-hard Problem
- The Approximation Factor

An example to illustrate the importance of analyzing the running time of an algorithm

- The Sorting Problem:
- Input(实例): An array of *n* elements *A*[*1*…*n*],
- Output(询问): Sort the entries in *A* in non-decreasing order.
- Assumption: each element comparison takes 10^{-6} seconds on some computing machine.

algorithm	# of element comparisons	n=128 $=2^7$	n=1,048,567 $=2^{20}$
Selection Sort (选择排序)	$\frac{n(n-1)}{2}$	$10^{-6}(128\times127)/2$ ≈ 0.008 seconds	$10^{-6}(2^{20}\times(2^{20}-1))/2$ ≈ 6.4 days
Merge Sort (归并排序)	$n\log n-n+1$	$10^{-6}(128\times7-128+1)$ ≈ 0.0008 seconds	$10^{-6}(2^{20}\times20-2^{20}+1)$ ≈ 20 seconds

The Importance of Analyzing the Running Time of an Algorithm

- Conclusion: Time is undoubtedly an extremely precious resource to be investigated in the analysis of algorithms.
- Question: How to Analyze Running Time?

Asymptotic Running Time(渐近估计)

How to measure the efficiency of an algorithm from the point of view of time?

- Is *actual (exact) running time* a good measure?
- The answer is No. Why?

  1. Actual time is determined by not only the algorithm, but also many other factors;
  2. The measure should be machine or technology independent;
  3. Our estimates of times are *relative* as opposed to *absolute*;
  4. Our main concern is *not* in *small input instances* but the behavior of the algorithms under investigation on *large input instances*.

- Then, is there a better measure? The answer is Yes. It is

asymptotic running time

Elementary Operation

- Elementary Operation(基本操作): Any computational step whose cost is always upper-bounded by a constant amount of time regardless of the input data or the algorithm used.
- Examples of elementary operations:
  - Arithmetic operations: addition, subtraction, multiplication and division.
  - Comparisons and logical operations.
  - Assignments, including assignments of pointers.
- Instance size or Input size(实例长度,输入长度):
  - The number of elements in an array(n).
  - For graph, the number of vertices ($|V|$)and the number of edges($|E|$).

Time Complexity (continued)

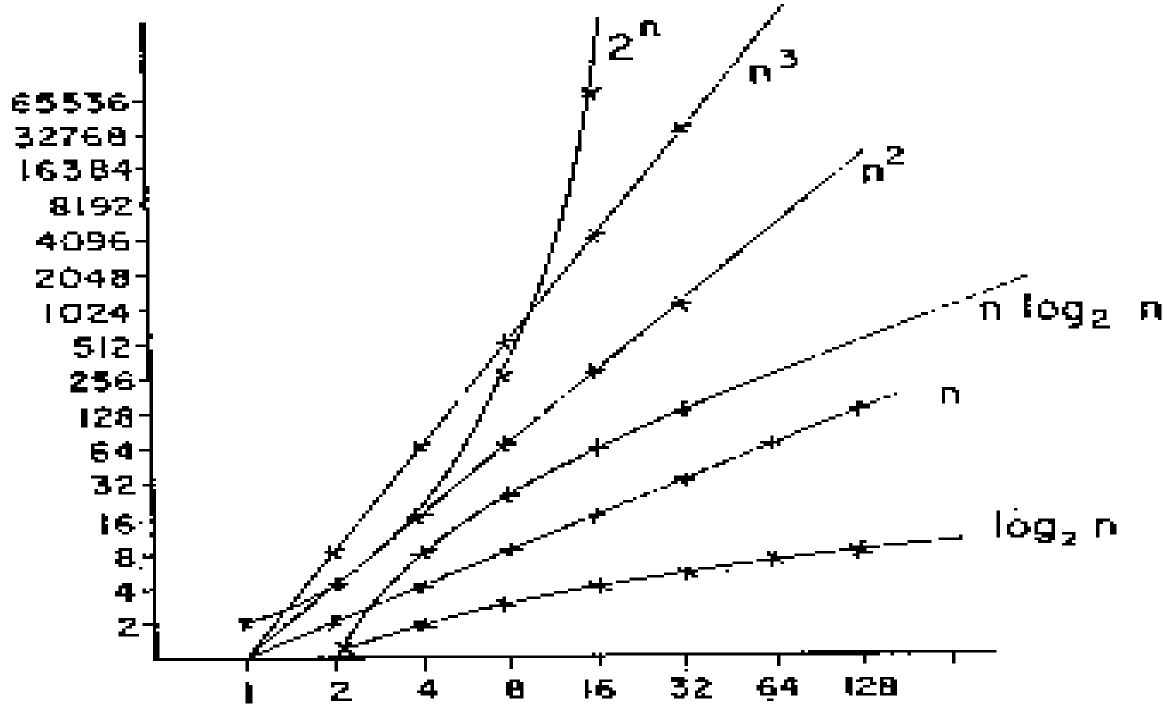
- What’s Time Complexity?
- When analyzing the running time of an algorithm, we are only interest in the elementary operations, the total number of elementary operations is called the time complexity of the algorithm.
- 在算法分析中,总是只考虑算法中的基本操作, 并估计算法执行过程中所需要的基本操作总次数,这个总操作次数称为算法的时间复杂度.
- Once again, we are interested in the running time for large input sizes, or more precisely, we want to discover how fast the running time of an algorithm increases as the size of the input increases (This is called ***order of growth*** of the algorithm，增长趋势).

Asymptotic Running Time (continued)

- The running time of an algorithm is a function of input size, e.g. $f(n)$;
- In the expression of the function, the costs of elementary operations can be written as their upper-bounds,
  e.g. $f(n) = c_1n^2 + c_2n + c_3$, $c_1, c_2, c_3 > 0$;
- The lower-order terms can be abandoned (or neglected) safely, e.g. $f(n) = c_1n^2$;
- The leading constants can also be abandoned safely,
  e.g. $f(n) = n^2$;

- Once we dispose of lower-order terms and leading constants from a function that expresses the running time of an algorithm, we say that we are measuring the **asymptotic running time** of the algorithm.

Illustration of some typical asymptotic running time functions



We can see: The linear algorithm is obviously slower than the quadratic one and faster than logarithmic one, etc..

We can say: The quadratic algorithm has a *higher order of growth, higher order of asymptotic running time,* or *higher order of time complexity,* than the linear one; etc..

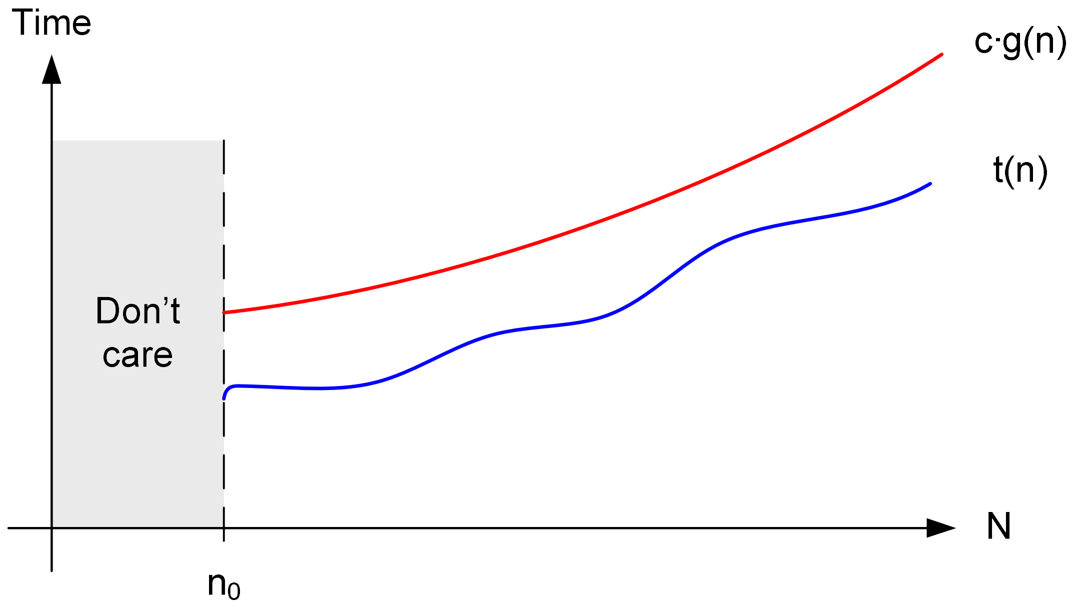
Asymptotic Notations

- To describe the asymptotic running time of an algorithm in a convenient way, notations are introduced. These notations are so-called **Asymptotic Notations**.
- In this course, we use three notations:
  - O(.) : “Big-Oh” – the most used
  - Ω(.) : “Big omega”
  - Θ(.) : “Big theta”

O-notation

- Informal definition of O(.):

  If an algorithm′s running time $t(n)$ is *bounded above* by a function $g(n)$, to within a constant multiple c, for $n > n_0$, we say that the running time of the algorithm is $O(g(n))$.

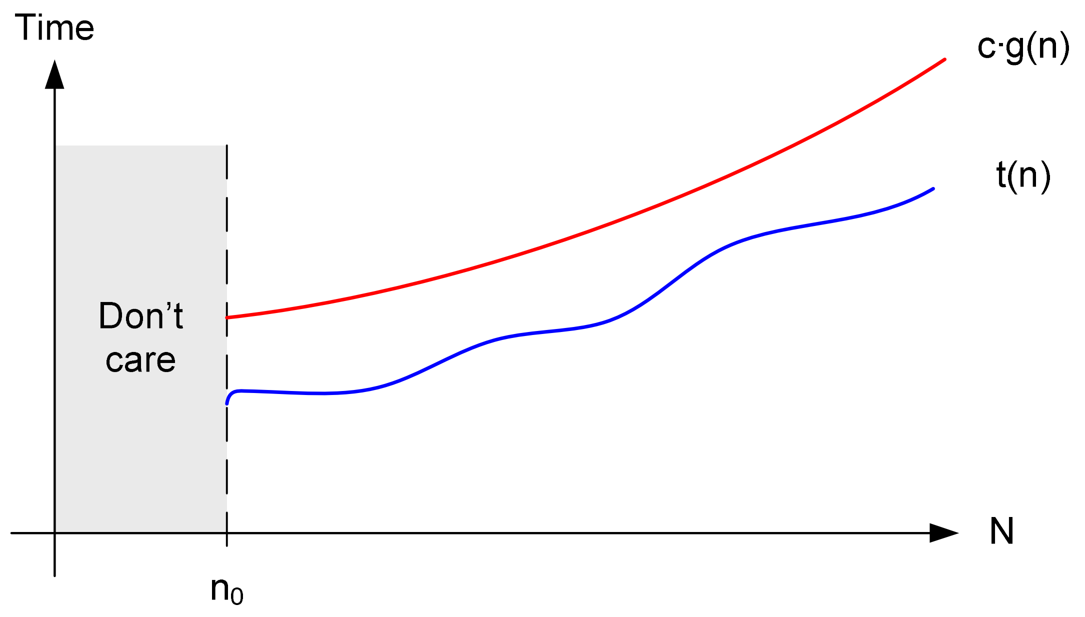


Obviously, O-notation is used to bound the **worst-case** running time of an algorithm

O-notation

- Formal definition of O(.):

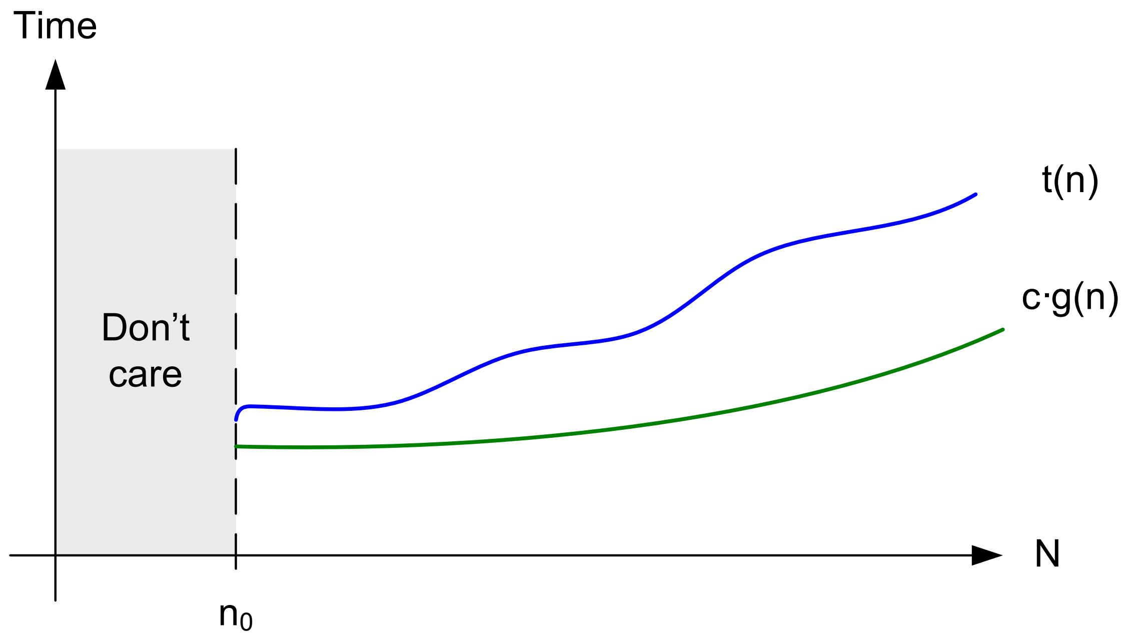
  A founction $t(n)$ is said to be in $O(g(n))$, if there exist some $c > 0$, $n_0 > 0$, such that $t(n) \leq cg(n)$, for all for $n > n_0$.



Ω-notation

- Informal definition of Ω(.) :

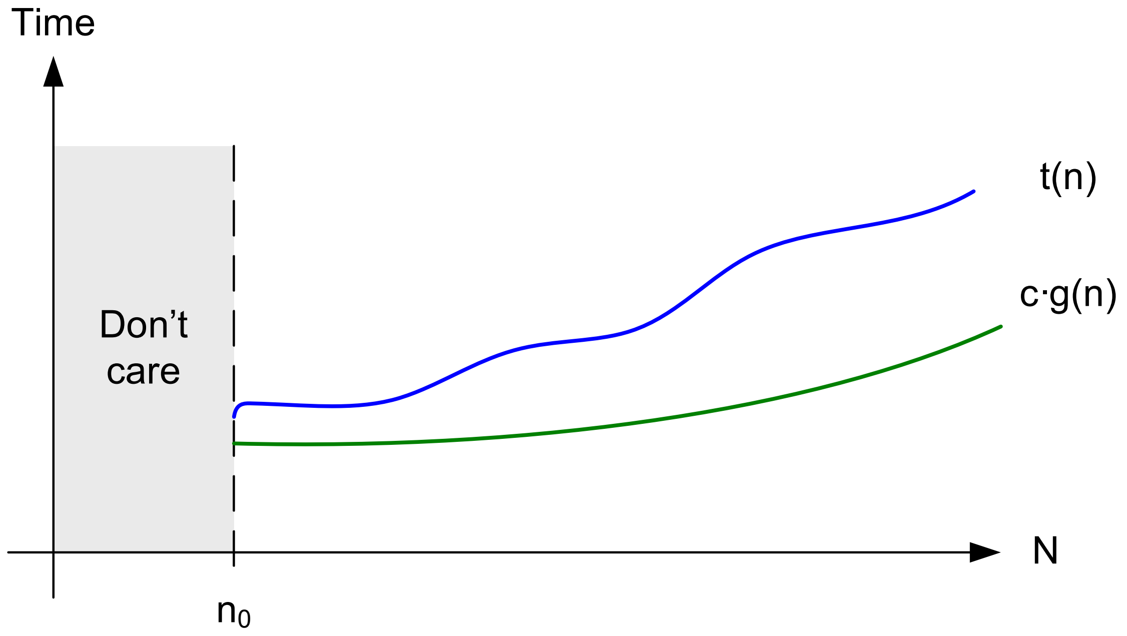
  If an algorithm′s running time $t(n)$ is *bounded blow* by a function $g(n)$, to within a constant multiple c, for $n > n_0$, we say that the running time of the algorithm is $\Omega(g(n))$.



Ω-notation

- Formal definition of Ω(.) :

  A founction $t(n)$ is said to be in $\Omega(g(n))$, if there exist some $c > 0$, $n_0 > 0$, such that $t(n) \geq cg(n)$, for all $n > n_0$.

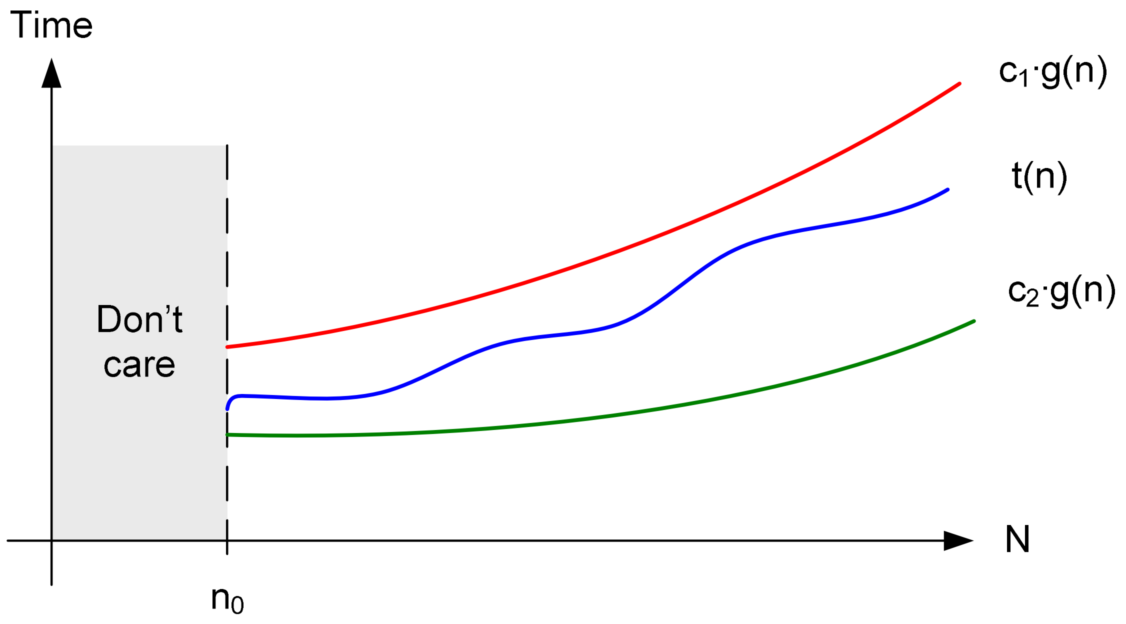


Obviously, Ω -notation is used to bound the **best-case** running time of an algorithm

Θ-notation

- Informal definition of Θ(.) :

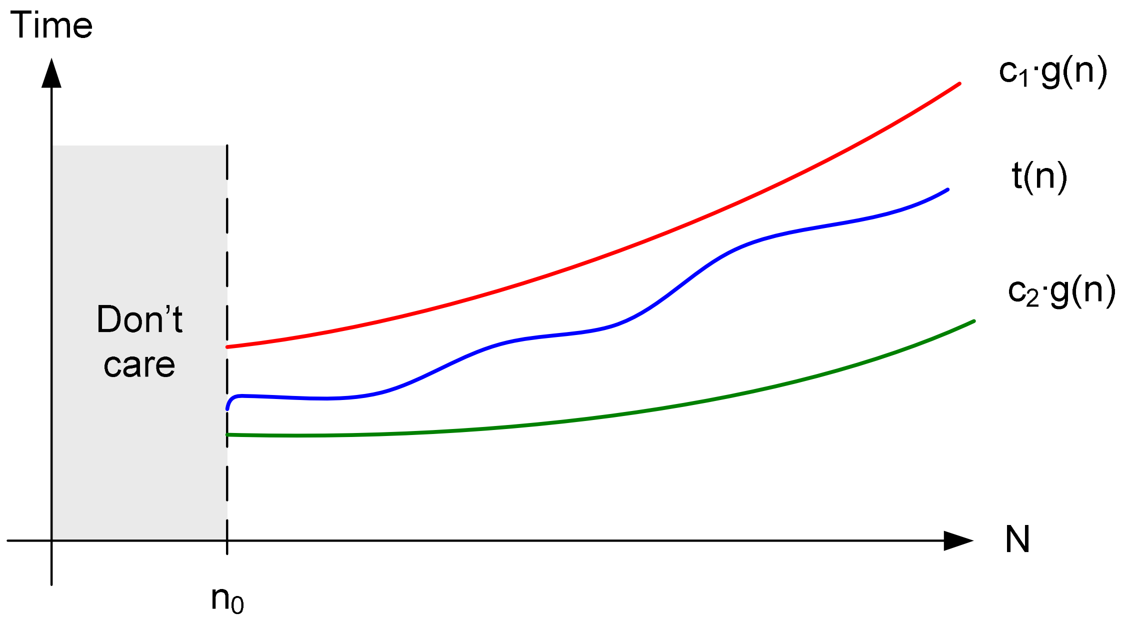
  If an algorithm′s running time $t(n)$ is *bounded above* and *blow* by a function $g(n)$, to within a constant multiple c_1 and c_2, for $n > n_0$, we say that the running time of the algorithm is $\Theta(g(n))$.



Θ-notation

- Formal definition of Θ(.) :

  A function $t(n)$ is said to be in $\Theta(g(n))$ if there exist $c_1 > 0$, $c_2 > 0$, and $n_0 > 0$, such that $c_2 g(n) \le t(n) \le c_1 g(n)$, for all $n > n_0$.



Three Types of Analysis

- Best-Case Analysis: too optimistic
- Average-Case Analysis: too difficult, e.g. the difficulty to define "average case", the difficulty related with mathematic
- Worst-Case Analysis: very useful and practical. We will adopt this approach.

Evaluate the two algorithms

- Exponentiation(求幂运算):
- Input: $x \in R$, $n \in Z^+$.
- Question: x^n

```
Algorithm 1-power($x$, $n$)
$y$=$x$;
For $i$ =2 to $n$
    $y$=$y$*$x$;
End For
Return $y$
```

```
Algorithm 2-power($x$, $n$)
//input $x$, $n$ = $(b_{m-1}, b_{m-2}, b_{m-3}, \ldots , b_1, b_0)_2$,
//where $b_{m-1}$=1, m≥2
$y$=$x$;
For $i$ =$m$-2 to 0
    $y$=$y$*$y$;
    if $b_i$=1 then $y$=$y$*$x$;
End For
Return $y$
```

- *Time Complexity is* $O(n)$
- *Time Complexity is* $O(\log n)$

Example of Three Types of Analysis

$INSERTION-SORT(A)$	cost	times
1 for $j = 2$ to n	c_1	$n-1$
2 do $key \leftarrow A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1..j-1]$		
3 $i \leftarrow j-1$	c_3	$n-1$
4 while $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_j$
5 do $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6 $i \leftarrow i-1$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $A[i+1] \leftarrow key$	c_7	$n-1$

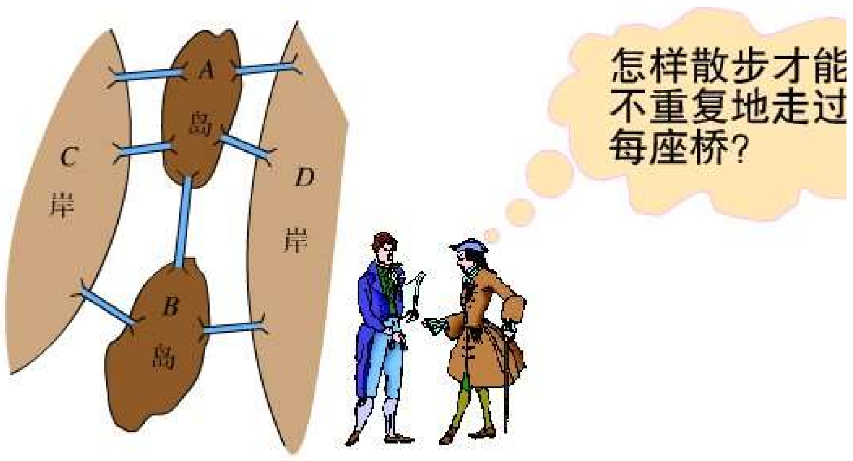
* t_j stands for the number of times the while loop test in line 5 is executed for that value of j

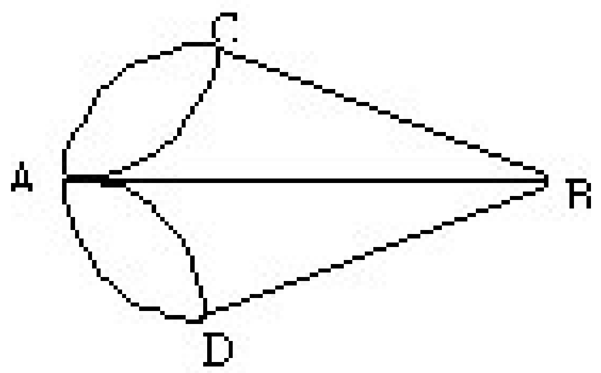
Graph Preliminary Knowledge

- In this class, try to answer three questions:
  - What is a graph?
  - How to represent a graph?
  - Basic properties of a graph

Graph Preliminary Knowledge

- The origination of graph
  (Maybe the first graph!) 哥尼斯堡七桥问题.

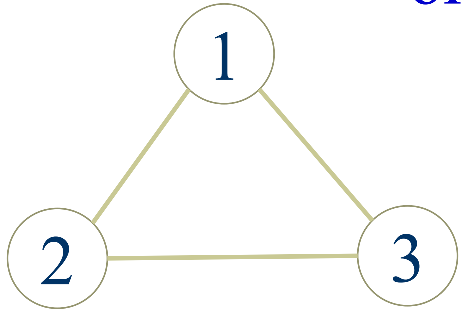




Graph Basics (Undirected)

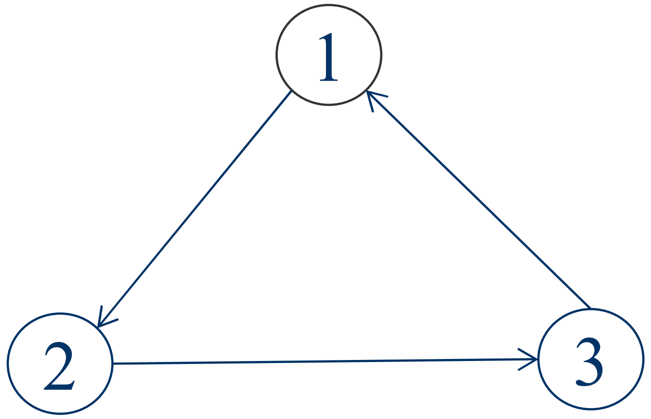
- A Graph G is a pair (V, E), where V is a finite set of objects and E is a set of pairs on V
  - Denoted as $G = (V, E)$
  - Element of V: vertex; V: vertex set
  - Element of E: edge(arc); E: edge set
- Undirected Graph : each pair in E is unordered, i.e., if $(u, v) \in E$, then $(u, v) = (v, u)$.
- Undirected graph: $G_1 = (V=\{1, 2, 3\}, E=\{(1, 2), (2, 3), (3,1)\})$

  or $G_1 = (V=\{1, 2, 3\}, E=\{(2, 1), (3, 2), (1,3)\})$

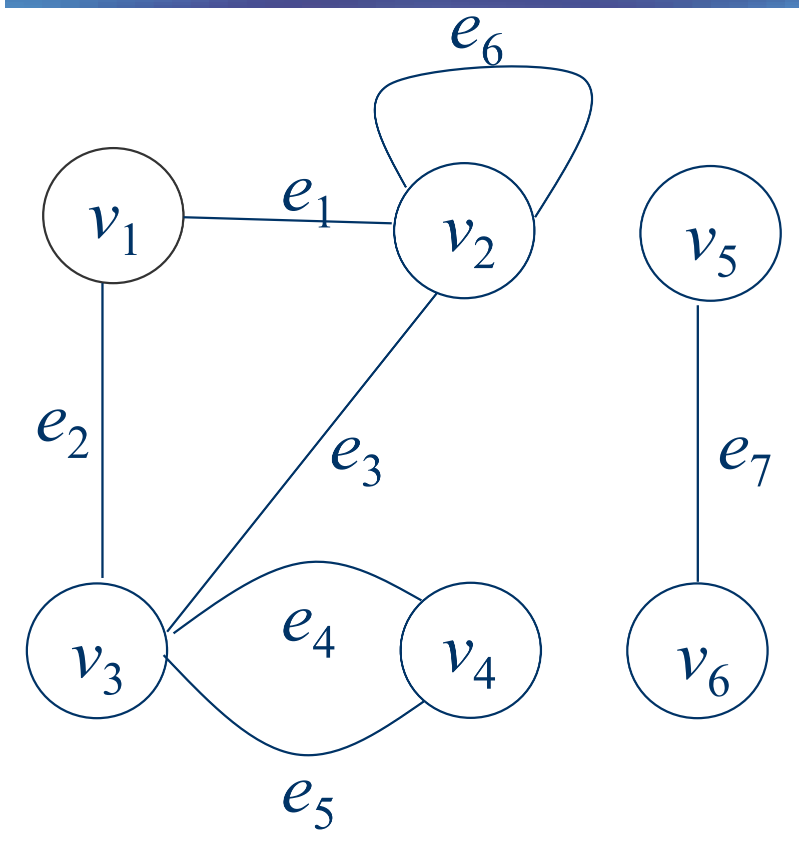


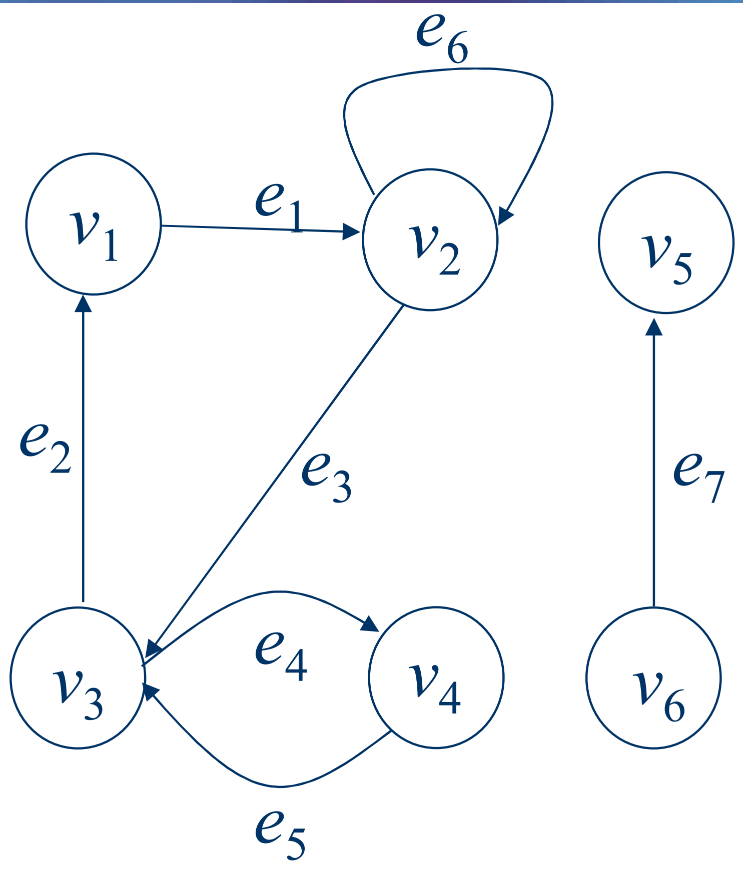
Graph Basics (Directed)

- Directed Graph: each pair in E is ordered, i.e., for each pair of vertices, u, v, $<u, v> \in E$ does **not** necessarily mean $<v, u> \in E$, and vice versa.
- Directed graph: $G_2 = (V=\{1, 2, 3\}, E=\{<1, 2>, <2, 3>, <3,1>\})$



Schematic Representation





Can you give the representation in format $G = (V, E)$ for above graphs?

Graph Basics (Adjacency)

- Adjacency: In a graph $G = (V, E)$, if $(u, v) \in E$, we say that vertex v is adjacent to vertex u.
  - If G is an undirected graph, vertex u is adjacent to vertex v, too. The adjacency relation is symmetric.

    We also say (u , v) is incident with u and v, vice versa.
  - If G is a directed graph and if $<u, v> \in E$, vertex v is adjacent to vertex u. *Now, $<u, v>$ is from (or leaves) vertex u to (or enters) vertex v.*

Graph Basics(Multiple, Simple)

- Repeated edges: edges e and f have the same endpoints.
- Self-loop: an edge from a vertex to itself.
- Simple graph: a graph without repeated edges and without self-loops.
- Weighted graph: each edge e has a weight $w(e)$, $w(e)\in R$.

Graph Basics (Degree)

- Degree of a vertex *v*:
  - For undirected graph:
    - Number of edges incident with *v*.(Self-loop?)
    - Denote as $d_G(v)$ or simply, $d(v)$.
  - For directed graph:
    - In-degree: number of edges entering (or incident to) *v*.
    - Out-degree: number of edges leaving (or incident from) *v*.
    - Denote as $id_G(v)$, $od_G(v)$ or simply, $id(v)$, $od(v)$ respectively.
- Isolated vertex: a vertex whose degree is 0.

Graph Properties

- Theorem 1.1 (Handshaking Lemma)
  - Given an undirected graph $G = (V, E)$, $\sum_{v \in V} d(v) = 2|E|$, where $|E|$ is the number of edges in G.
  - Proof: can you give?
  - How about a directed graph? (sum of in-degrees equals to sum of out-degrees, i.e., the number or arcs)
- Corollary 1.2
  - Given an undirected graph $G = (V, E)$, the number of vertices with odd degrees is even.
  - Proof: can you give?

Graph Basics (Cycle)

- A path is a sequence of vertices $p = <v_0, v_1, v_2, \ldots, v_k>$ such that $(v_i,v_{i+1}) \in E$, $i=0,1,\ldots,k$-1.
  - 路径: $v_0, v_1, \ldots, v_k$ are distinct (except that $v_0 = v_k$).
  - 通路: otherwise
- A cycle is a path $c = <v_0, v_1, v_2, \ldots, v_k>$ with $v_0 = v_k$.
  - 圈和回路.
  - For undirected graph, a cycle contains at least 3 edges, that is, $k \geq 3$ and $v_1, v_2, \ldots, v_k$ are distinct.
  - For directed graph, a cycle contains at least 2 edge.
  - A self-loop is a cycle of length 1.
- Acyclic graph: a graph with no cycles.

Graph Basics (Sub-graph)

- A graph $G' = (V', E')$ is a sub-graph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.
  - A graph $G' = (V', E')$ is a vertex induced sub-graph of $G = (V, E)$ if $V' \subseteq V$ and $E' = \{(u, v) \in E : u, v \in V'\}$.

Graph Basics (Connected)

- An undirected graph is connected if every pair of vertices is connected by a path.
  - Connected component of a graph: a sub-graph that is connected and is not included in any other connected sub-graphs with more edges or vertices.
  - *Any connected, undirected graph $G = (V, E)$ satisfies that* $| E | \geq | V | -1$.
- A directed graph is strongly connected if every two vertices are reachable from each other.
  - Strongly connected components of a directed graph

Special graphs

- A complete graph is an undirected graph in which every pair of vertices is adjacent.
- A bipartite graph is a graph $G = (V, E)$ in which V can be partitioned into two sets V_1 and V_2 such that $(u, v) \in E$ implies either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$.
- A tree is a connected, acyclic, undirected graph.($|E|=|V|-1$; a unique path between any pair of vertices.)
- A forest is an acyclic, undirected graph.

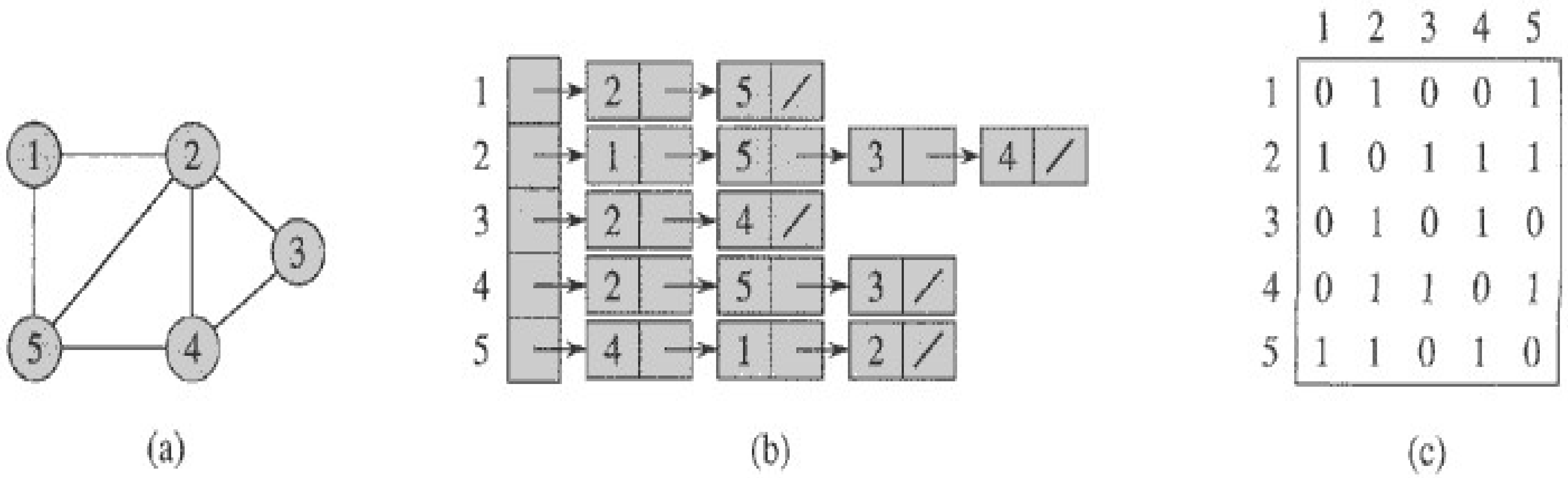
Representations of graphs

- Two standard ways:
  - A collection of adjacency lists
  - An adjacency matrix
- Sparse graph: $|E|$ is much less than $|V|^2$
  - Adjacency-list representation is preferred
- Dense graph: $|E|$ is close to $|V|^2$
  - Adjacency-matrix representation is preferred

Adjacency-list representation

- consist of an array *Adj* of $|V|$ lists, one for each vertex in V.
- adjacency list $Adj[u]$ contains all the vertices v such that there is an edge $(u, v) \in E$, i.e., all vertices that are adjacent to u.
- vertices in $Adj[u]$ can be stored in arbitrary order.

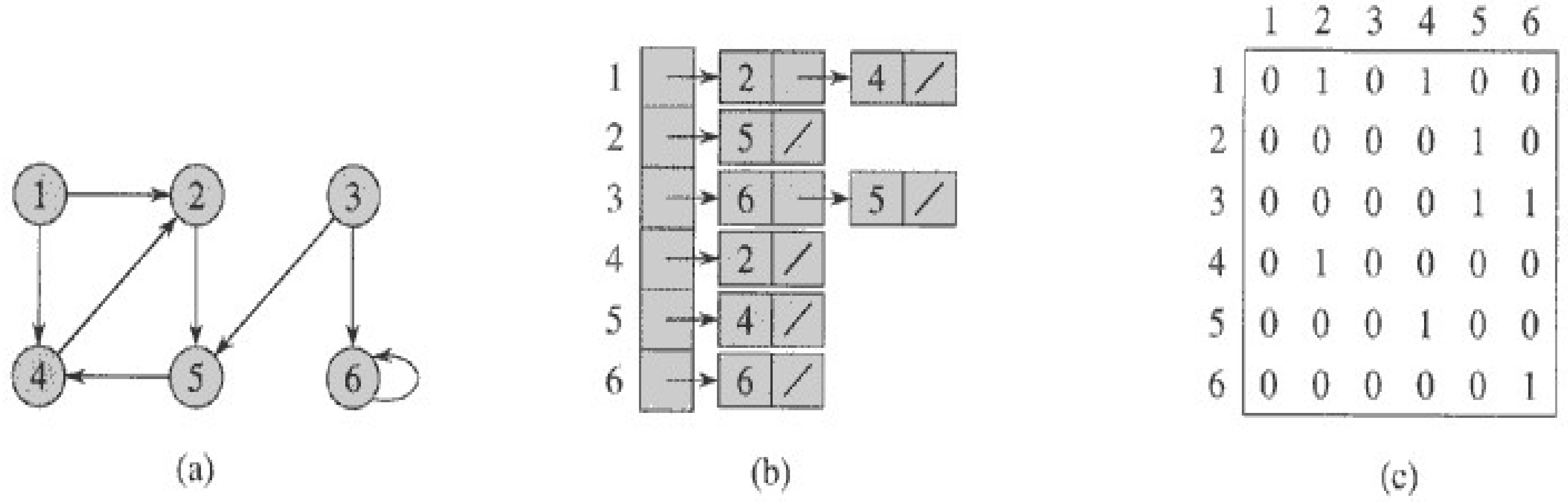
Representations of graphs (Examples)



$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Representations of graphs (Examples)



$$\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}$$

Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Adjacency-list representation

- Some questions.
  - How to compute the degree of each vertex?
  - What is the total length of all the adjacency-lists?
  - How to determine whether a given edge is present in the graph or not?

Adjacency-list representation

- sum of the lengths of all the adjacency lists is
  - $2|E|$ if G is an undirected graph
  - $|E|$ if G is a directed graph
- Memory needed to store an adjacency-list representation of a graph is $O(|V|+|E|)$
- Weighted graph can be represented by adjacency lists.

Adjacency-list representation (advantage vs disadvantage)

- Advantage:
  - when the graph is sparse, uses only $O(|V|+|E|)$ memory.
- Disadvantage:
  - no quicker way to determine if a given edge (u, v) is present in the graph than searching v in the adjacency list $Adj[u]$. $O(|V|)$

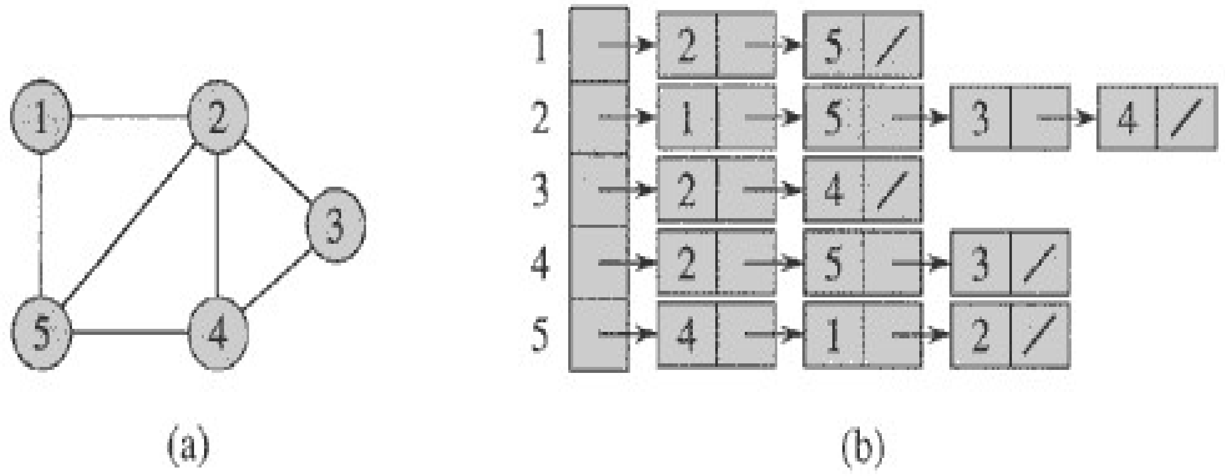
Adjacency-matrix representation

- Consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if (i,j)} \in E, \\ 0 & \text{otherwise.} \end{cases} \quad \text{or} \quad a_{ij} = \begin{cases} 1 & \text{if (i,j)} \in E, \\ \infty & \text{otherwise.} \end{cases}$$

- A is symmetric along the main diagonal for undirected graphs, that is, $A^{\mathrm{T}} = A$.
- For directed graph, generally, $A^{\mathrm{T}} \neq A$. Can you imagine what directed graph satisfies $A^{\mathrm{T}} = A$?
- Adjacency-matrix can be used to represent weighted graphs.

Representations of graphs (Examples)



$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0
\end{array}
$$

(c)

Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Representations of graphs (Examples)

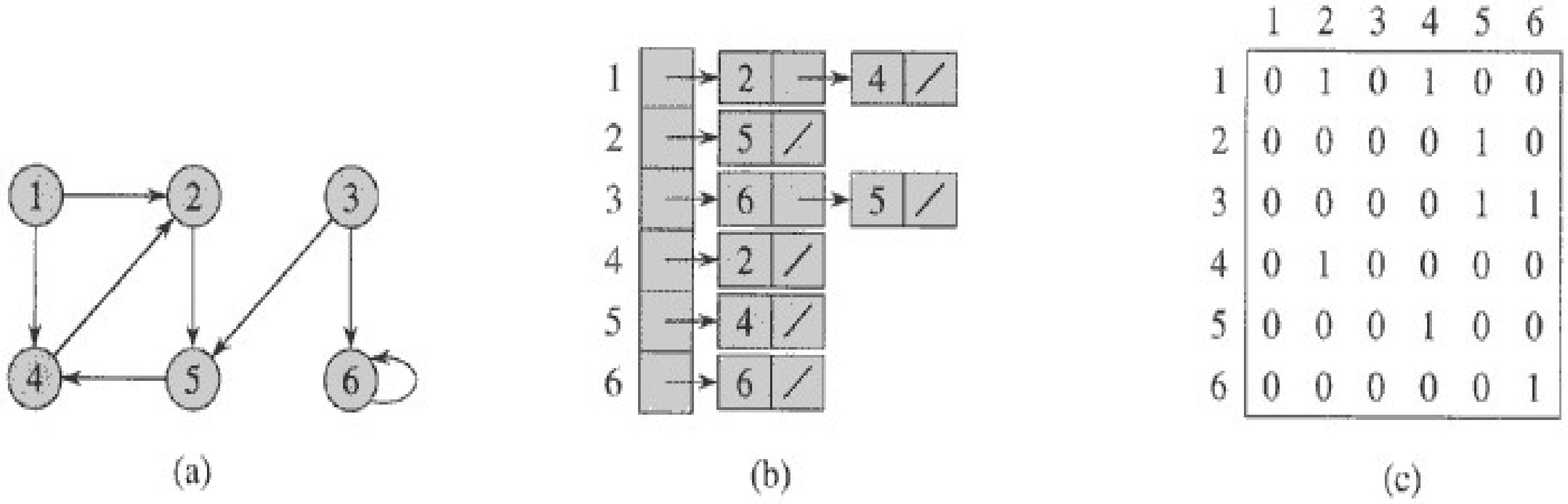


Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Properties of adjacency-matrix representation

- Lemma1.3: Suppose that A is the adjacency-matrix representation of an undirected graph G, then the element $A^k[i, j]$ of $A^k = A \cdot A \cdot \ldots \cdot A$ is the number of paths(通路) of length k that connect vertices i and j.
  - Proof by induction on k.
    - Basic step: $k = 1$, $A^1 = A$, $A[i, j]$ is the number of paths of length 1 that connect vertices i and j.
    - Hypothesis : $A^k[i, j]$ is the number of paths of length k that connect vertices i and j.
    - Inductive step: $A^{k+1}=A \cdot A^k$, we have $A^{k+1}[i, j] = \Sigma_{l \in V} A[i, l] \cdot A^k[l, j]$. For $A[i, l]$ is the number of paths of length 1 that connect vertices i and l, $A^k[l, k]$ is the number of paths of length k that connect vertices l and j. So, $A[i, l] \cdot A^k[l, j]$ is the number of paths of length k+1 that connect vertices i and j via l. Consider all vertices l, $1 \leq l \leq n$, we get our conclusion.

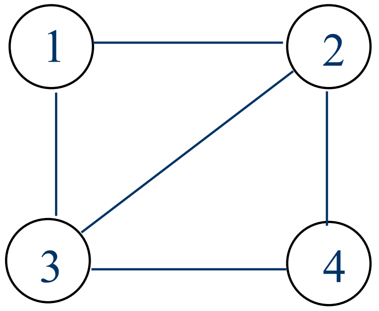
Properties of adjacency-matrix representation

- Corollary 1.4 $A^2[i,i] = \sum_{j=1}^{n} A[i,j] = \sum_{j=1}^{n} A[j,i] = d_G(i)$

Proof: For symmetry of adjacency-matrix,

$A[i, l]$=A$[l, i]$ = 0 or 1, so

$$A^2[i,i] = \sum_{l=1}^{n} A[i,l] \bullet A[l,i] = \sum_{j=1}^{n} A[i,j] = \sum_{j=1}^{n} A[j,i] = d_G(i)$$



$A^2[1, 1]$: 2: (1, 2, 1), (1, 3, 1)

$A^2[2, 2]$: 3: (2, 1, 2), (2, 3, 2), (2, 4, 2)

Adjacency-matrix representation (Advantage vs disadvantage)

- Advantage:
  - Easily or quickly to determine if an edge is in the graph or not. $O(1)$
- Disadvantage:
  - Uses more memory to store a graph. $O(|V|^2)$