CS 214
Introduction to Discrete Structures

Chapter 6
Graphs and Trees

Mikel D. Petty, Ph.D.

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Chapter sections and objectives

- 6.1 Graphs and Their Representations
  - Understand and use graph terminology
  - Appreciate the wide range of graph applications
  - Prove (or disprove) two graphs isomorphic
  - Use Euler's formula for graphs
  - Understand the role of K_5 and $K_{3,3}$ in graph planarity
  - Prove elementary properties about graphs
  - Use adjacency matrix and adjacency list representations for graphs

- 6.2 Trees and Their Representations
  - Understand and use tree terminology
  - Prove elementary properties about trees
  - Do preorder, inorder, and postorder tree traversals
  - Use array and pointer representations for trees
- 6.3 Decision Trees Not covered in CS 214
  - Model sorting and searching algorithms with trees
  - Build and use a binary search tree
  - Analyze worst-case for searching and sorting
- 6.4 Huffman Codes Not covered in CS 214

Sample problem 1

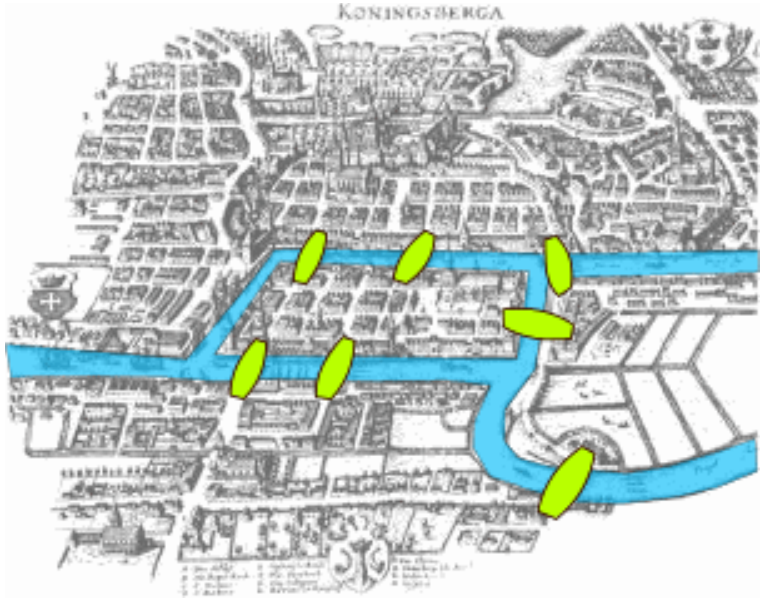
Widgets are made of many parts of the following types: bolt (B), component (C), gear (G), rod (R), and screw (S).
There are many variations of each part type.
Part numbers are given by a leading character B, C, G, R, or S to identify the type, followed by an 8-digit number.
For example: C00347289, B11872432, S45003781.
By the multiplication principle, there are $5 \cdot 10^8$ part numbers.
Each of the 9 characters in a part number uses 1 byte; the total part data list is approx $9 \cdot 5 \cdot 10^8$ bytes, or 4.5 G bytes.

How can this data file be compressed?

The answer to this problem is in § 6.4, not covered.

Sample problem 2

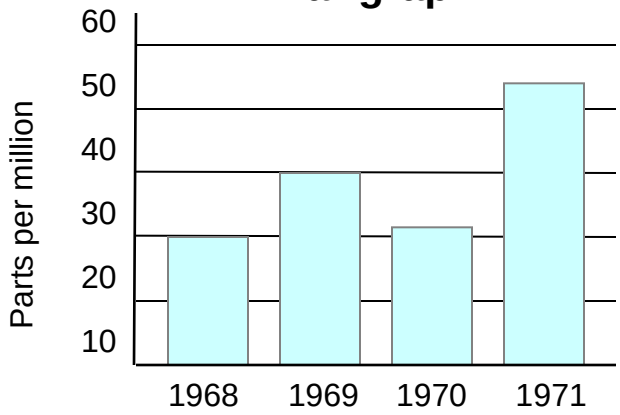
Seven Bridges of Königsberg
Is it possible to find a walk that crosses each of the seven bridges exactly once?



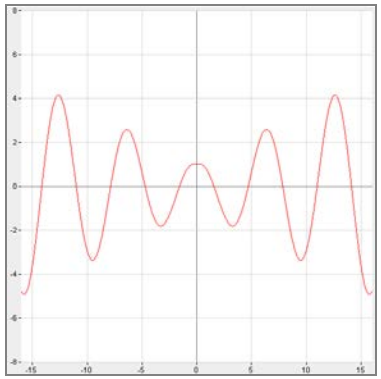
6.1 Graphs

Non-graphs

Bar graph



Function graph

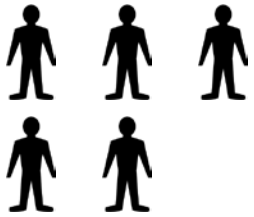


Picture graph

Each figure represents 100,000

New York

Philadephia



Pie chart

Figure 6.2

Graph

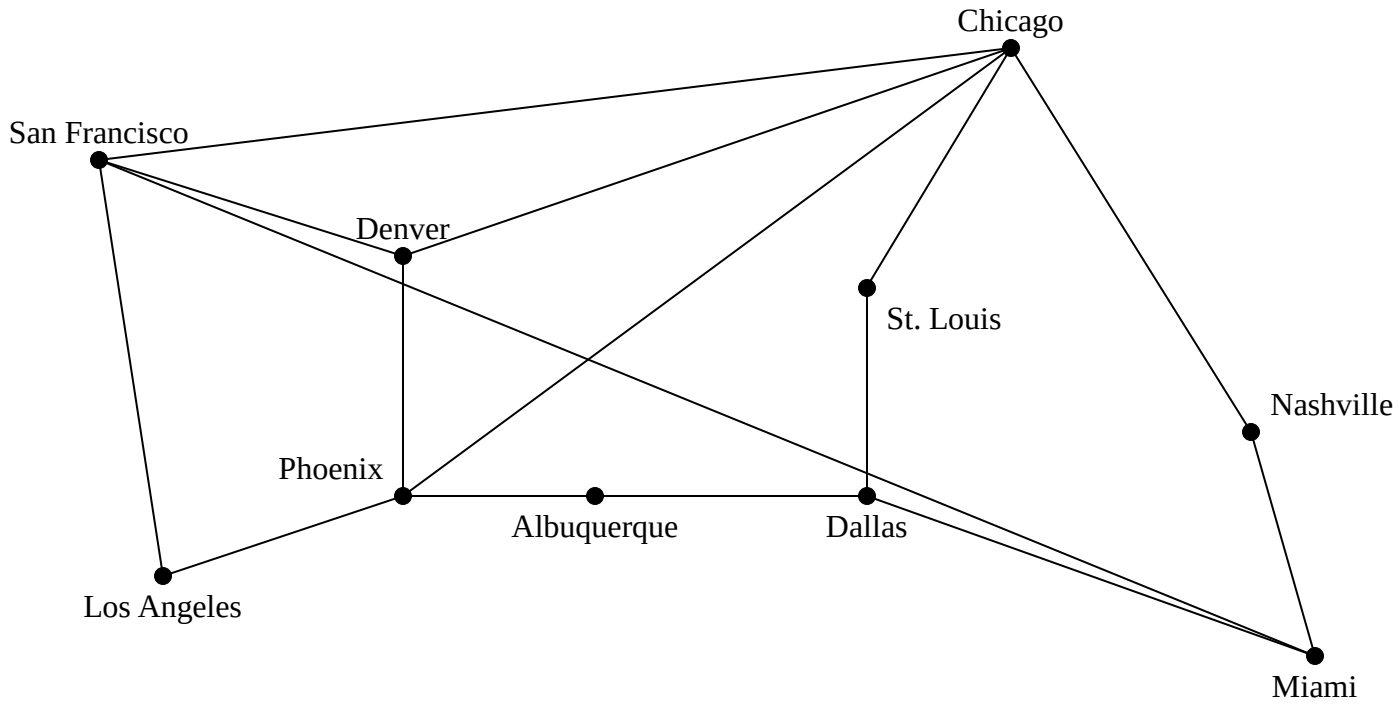


Figure 6.1

Graph definition (informal)

- Definition
  - A graph is a nonempty finite set of nodes and a finite set of arcs that each connect two nodes
- Comments
  - Alternate terms: node aka vertex, arc aka edge
  - Informal definition depends on visual representation

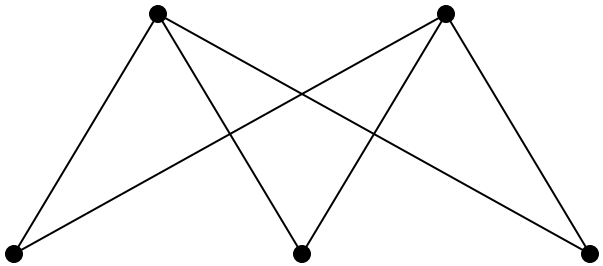
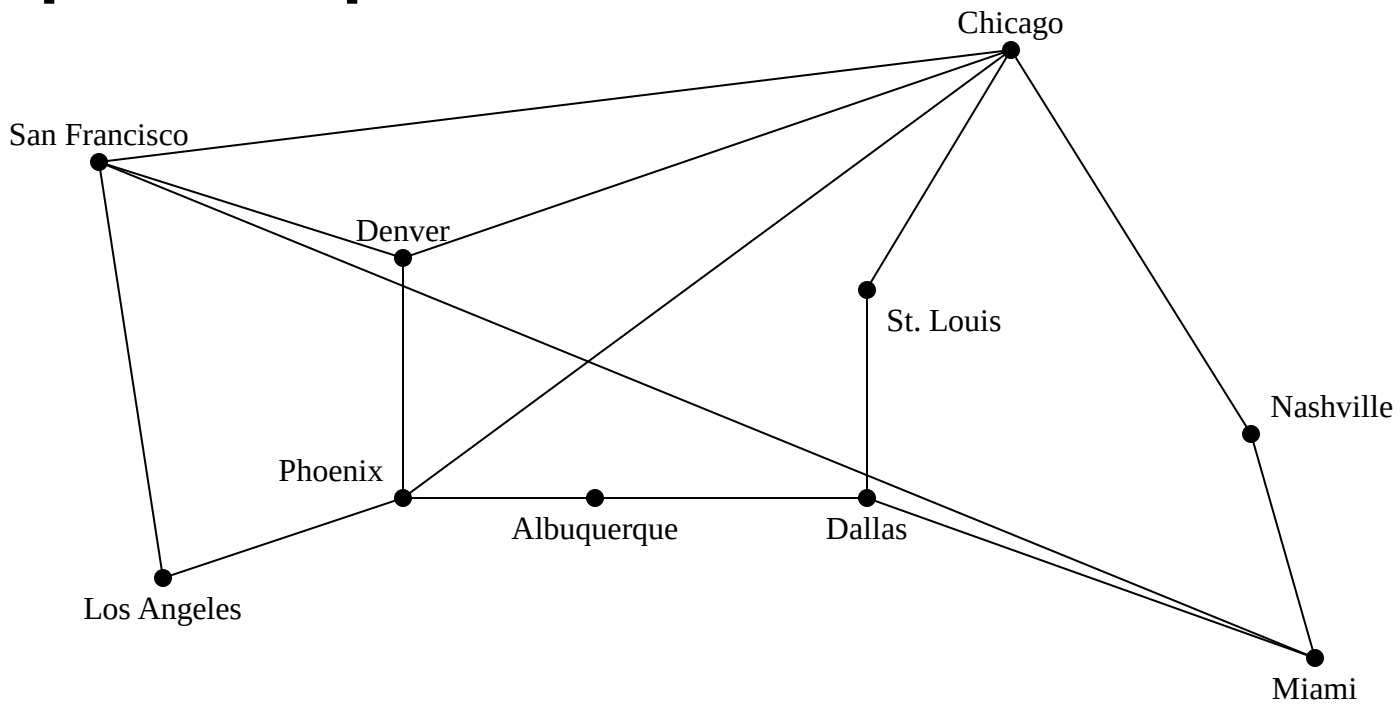


Figure 6.12

Graph example



Nodes {San Francisco, Los Angeles, Denver, Phoenix, Albuquerque, St. Louis, Dallas, Chicago, Nashville, Miami}

Arcs {Dallas–St. Louis, Denver–Phoenix, Nashville–Miami, … }

Example 1, Figure 6.1

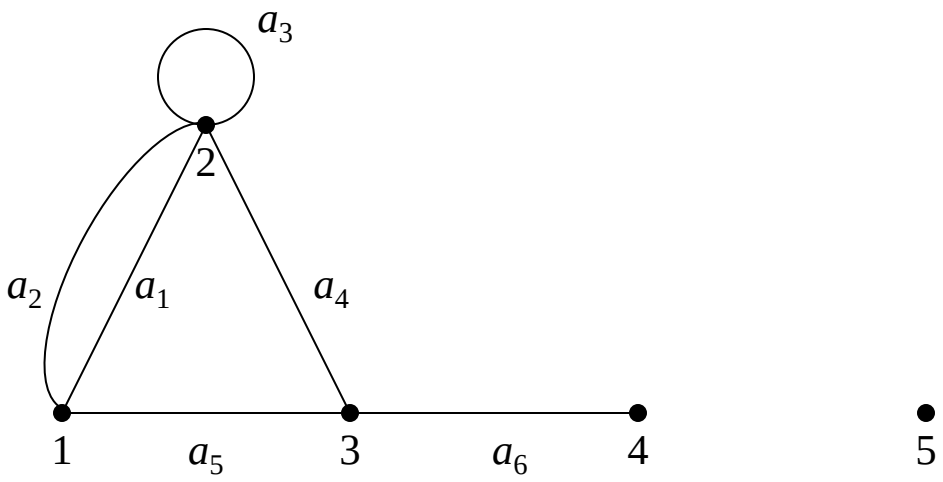
Graph definition (formal)

- Definition
  - A graph is an ordered triple (N, A, g) where
    N = a nonempty finite set of nodes
    A = a finite set of arcs
    g = a function associating with each arc $a \in A$
    an unordered pair of nodes x–y
- Comments
  - If $g(a) = x$–y, x and y are the endpoints of a
  - For function g, domain A,
    codomain $\{\{x, y\} \mid x, y \in N\}$, with $\{x, y\}$ written x–y
  - Does not depend on visual representation

Graph definition (alternate formal)

- Definition
  - A graph G is an ordered G = (V, E) where
    $V = \{v_1, v_2, \ldots, v_n\}, n > 0$
    $E = \{ \{v_i, v_j\} \mid v_i, v_j \in V \wedge v_i \neq v_j \}$
- Comment
  - No function *g* mapping edges to vertices, rather edges are sets of exactly two vertices

Graph example



Nodes $\{1, 2, 3, 4, 5\}$

Arcs $\{a_1, a_2, a_3, a_4, a_5, a_6\}$

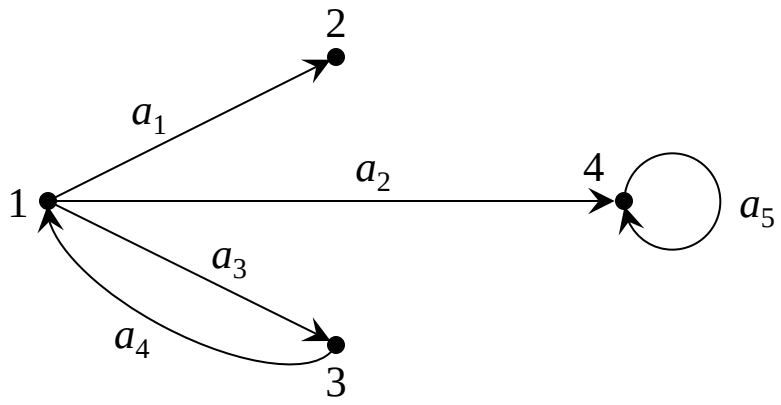
Function $g(a_1) = 1–2$, $g(a_2) = 1–2$, $g(a_3) = 2–2$,
$g(a_4) = 2–3$, $g(a_5) = 1–3$, $g(a_6) = 3–4$

Examples 2 and 3, Figure 6.3

Directed graph definition (formal)

- Definition
  - A directed graph is an ordered triple (N, A, g) where
    N = a nonempty finite set of nodes
    A = a finite set of arcs
    g = a function associating with each arc $a \in A$
    an ordered pair of nodes (x, y)
- Comments
  - aka digraph
  - If $g(a) = (x, y)$, x is the initial point and y is the terminal point of a
  - For function g, domain N, codomain $N \times N$
  - Does not depend on visual representation

Directed graph example



Nodes $\{1, 2, 3, 4\}$

Arcs $\{a_1, a_2, a_3, a_4, a_5\}$

Function $g(a_1) = (1, 2)$, $g(a_2) = (1, 4)$, $g(a_3) = (1, 3)$,
$g(a_4) = (3, 1)$, $g(a_5) = (4, 4)$

Example 4, Figure 6.4

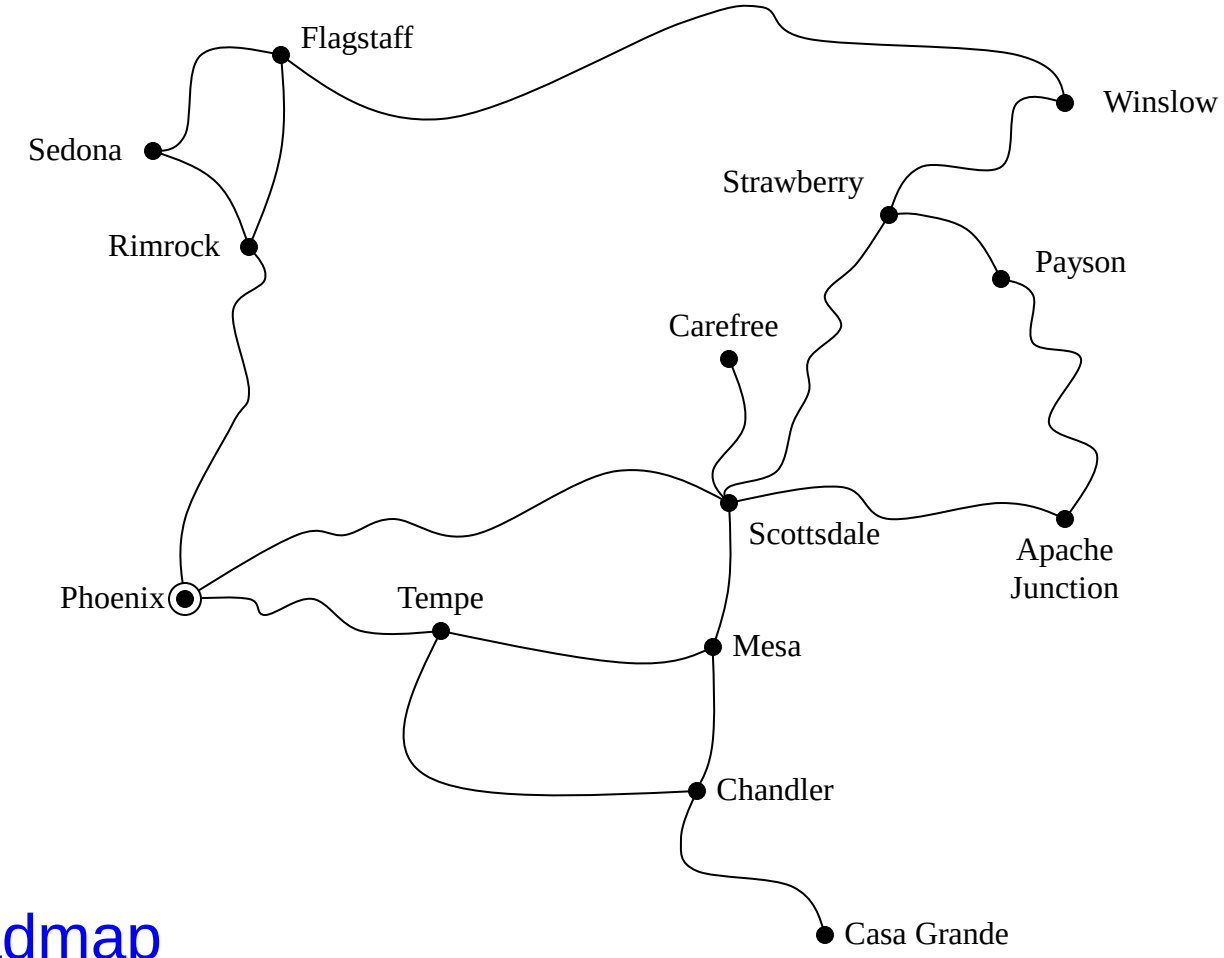
Graph variants

- Arcs
  - Unordered pairs: graph
  - Ordered pairs: directed graph
  - Unordered subsets of N: hypergraph
- Additional information
  - Labels attached to nodes or arcs: labeled graph
  - Numbers attached to nodes or arcs: weighted graph
- Many others …

Graph applications

- Graphs in discrete mathematics
  - Hasse diagram, § 5.1, graph
  - Function composition diagram, § 5.4, directed graph
  - PERT chart, § 5.2, directed graph
  - E-R diagram, § 5.3, graph
  - FSM state graph, § 9.2, directed graph
- Graph applications
  - Airline route map
  - Transportation network
  - Communications network
  - Distribution routes
- Many others …

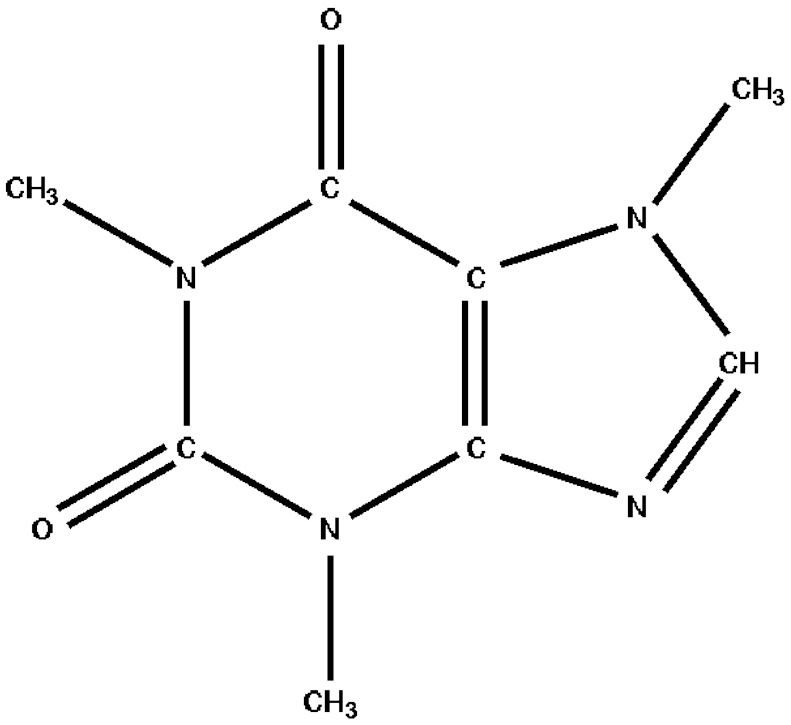
Graph application example



Practice 2, Figure 6.6

Roadmap

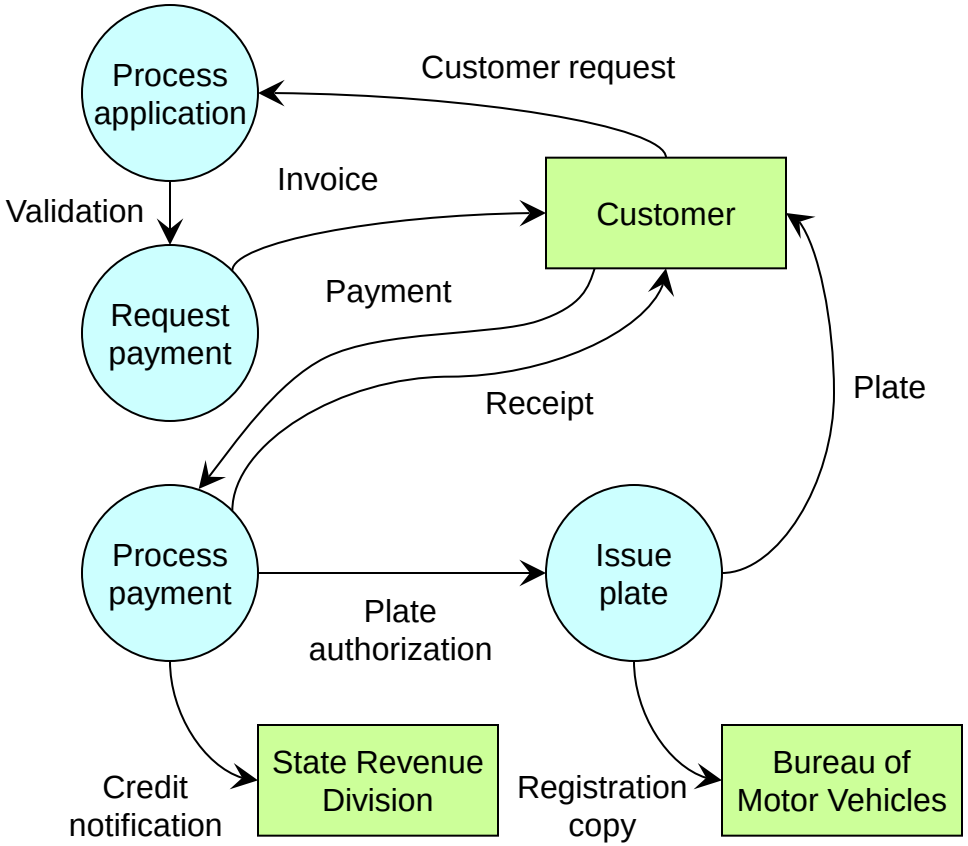
Graph application example



Molecule diagram

Practice 2, Figure 6.6

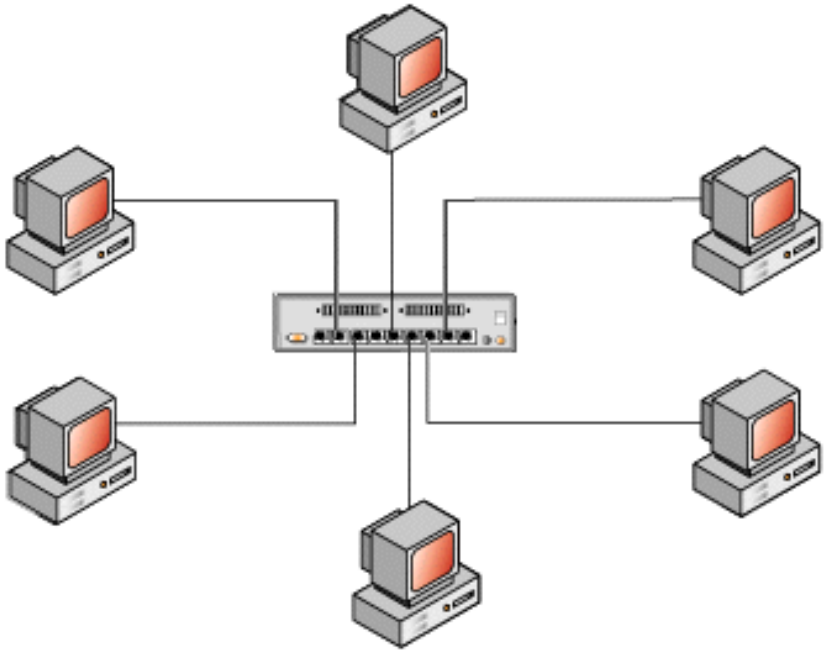
Graph application example



Data flow diagram

Example 5, Figure 6.7

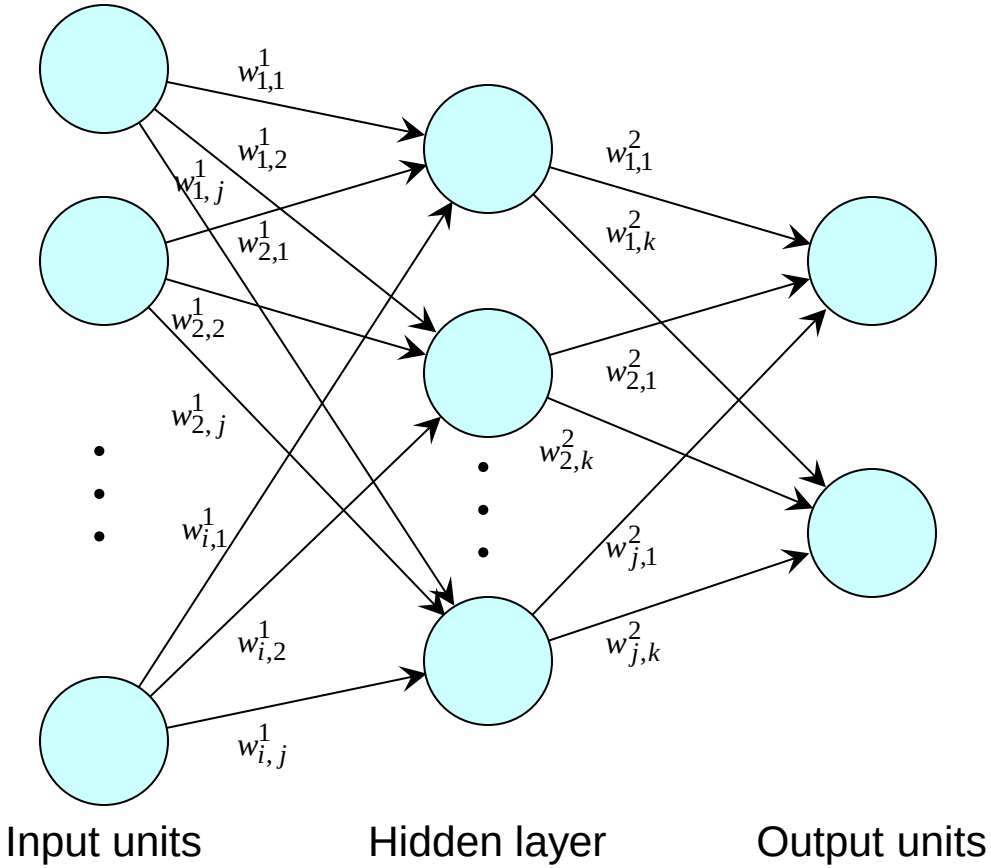
Graph application example



Computer network diagram

Example 6, Figure 6.8

Graph application example

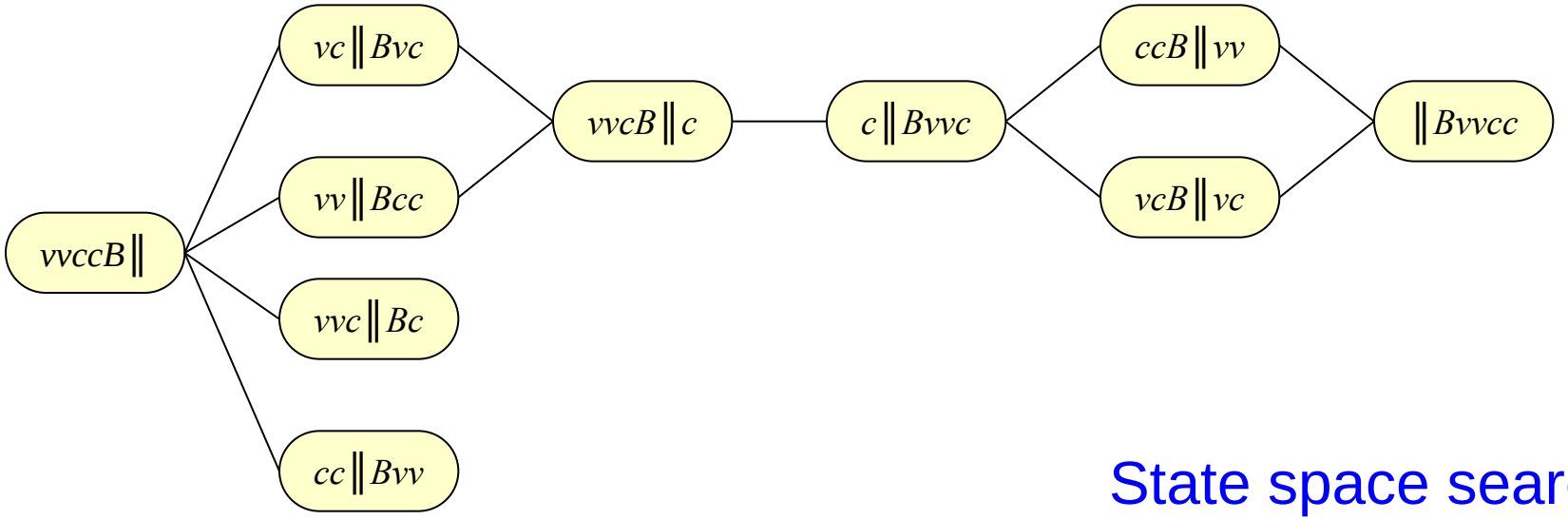


Neural net

Example 7, Figure 6.9

Graph example

- 2 vegetarians and 2 cannibals want to cross a river.
- They have a boat with a capacity of 2 people.
- Cannibals can not outnumber vegetarians on a bank.
- The boat cannot operate itself.



State space search

Example 10.1.7, p. 631, S. S. Epp, *Discrete Mathematics with Applications, Fourth Edition*, Brooks/Cole, Boston MA, 2011.

Graph applications summary

- Applications
  - Road map; graph
  - Molecule diagram; graph
  - Data flow diagram; directed graph
  - Computer network diagram; graph
  - Neural net; weighted directed graph
  - Many others…
- Common idea: graph abstractly represents structure of application
- Properties of graphs useful in application

Graph terms: nodes and arcs

- Adjacent; nodes that are the endpoints of an arc
- Isolated node; node not adjacent to other nodes
- Loop; arc with a single node as both endpoints
- Parallel; two arcs with the same endpoints
- Degree; for a node, number of arcs at a node
  - Loops count as 2, e.g., degree of node 2 is 5

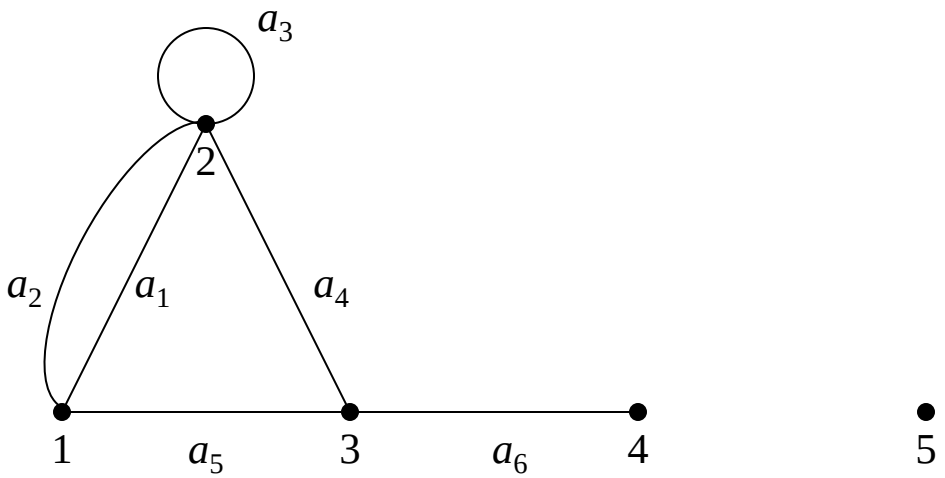


Figure 6.3

Graph terms: simple graph and subgraphs

- Loop-free; graph with no loops
- Simple graph; graph with no loops or parallel arcs
- Subgraph; given graph G, subgraph H consists of subsets of G's nodes and arcs, such that all arcs in H connect the same nodes as in G

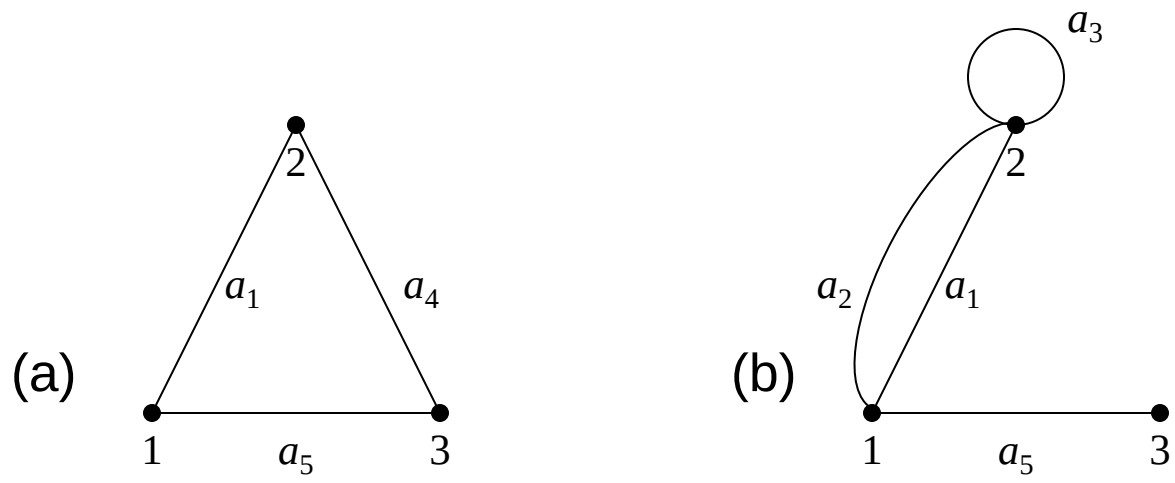
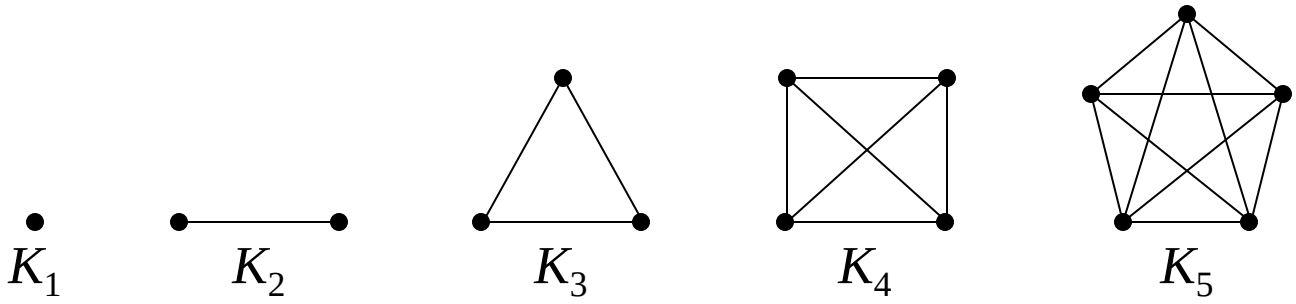


Figure 6.10

Graph terms: complete graph

- Complete graph
  - Simple graph where every pair of nodes is adjacent
  - Function g almost onto
  - Loops at each node not required in complete graph
- Simple, complete graph with n nodes
  - Denoted K_n
  - Has $n(n-1)/2$ arcs



Example 8, Figure 6.11, Practice 4

Graph terms: paths and cycles

- Path
  - Sequence $n_0, a_0, n_1, a_1, \ldots, n_{k-1}, a_{k-1}, n_k$?
    of nodes and arcs from node n_0 to node n_k
  - For each arc a_i in path, endpoints are nodes n_i–n_{i+1}
  - Nodes and arcs may repeat in a path
  - Length of a path; number of arcs it contains
  - Connected; graph with a path between any two nodes
- Cycle
  - A path from node n_0 to n_0 where
    - No arc appears more than once
    - Only node n_0 appears more than once, at ends
  - Nodes and arcs may not repeat in a cycle
  - Acyclic; graph with no cycles

Example paths and cycles

Paths

$2, a_1, 1, a_2, 2, a_4, 3, a_6, 4$ length 4

$4, a_6, 3, a_5, 1, a_2, 2, a_3, 2$ length 4

Cycle

$1, a_1, 2, a_4, 3, a_5, 1$ length 3

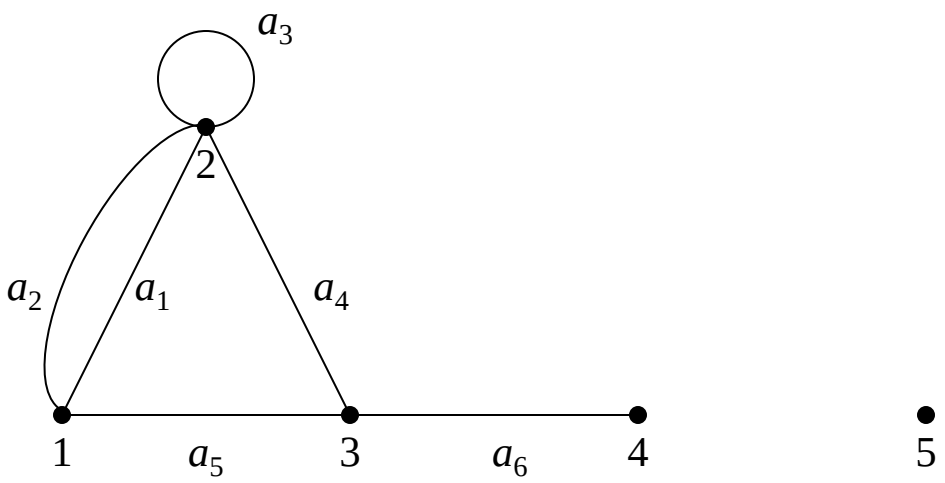


Figure 6.3

Graph terms: bipartite graph

- Bipartite graph (informal)
  - For graph (N, A, g), node set N partitioned into two subsets, N_1 and N_2; $N_1 \cup N_2 = N$, $N_1 \cap N_2 = \varnothing$
  - Every arc $a \in A$ has one endpoint in N_1, other in N_2
- Bipartite complete graph (informal)
  - Every node in N_1 is adjacent to every node in N_2
  - If $|N_1| = m$ and $|N_2| = n$, denoted $K_{m,n}$

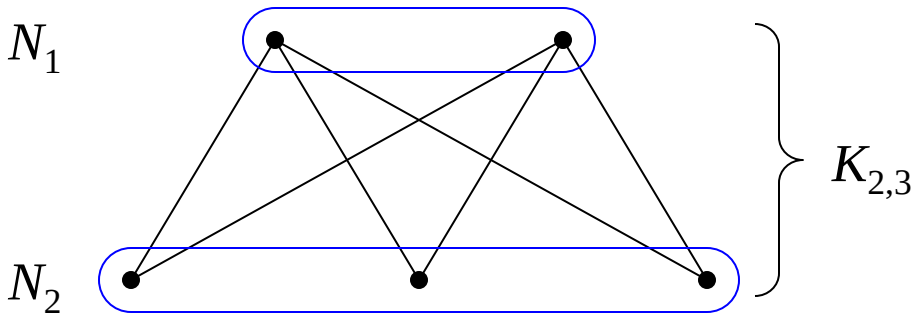


Figure 6.12

- Bipartite graph (formal)
  - A graph (N, A, g), is bipartite if its nodes can be partitioned into two disjoint nonempty sets N_1 and N_2 such that two nodes x and y are adjacent only if $x \in N_1$ and $y \in N_2$.
- Bipartite complete graph (formal)
  - A graph (N, A, g), is bipartite complete … x and y are adjacent if and only if $x \in N_1$ and $y \in N_2$.

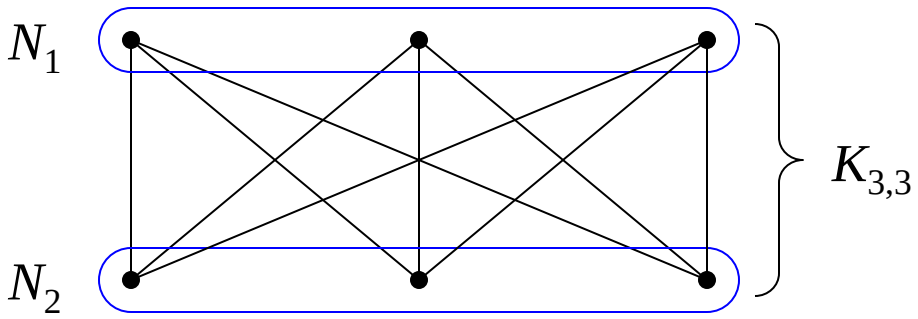


Figure 6.12

Graph terms: paths and cycles in digraphs

- Path in a directed graph
  - Sequence $n_0, a_0, n_1, a_1, \ldots, n_{k-1}, a_{k-1}, n_k$ of nodes and arcs from node n_0 to node n_k
  - For each arc a_i in path, n_i is initial point, n_{i+1} is terminal point
  - Reachable; n_k is reachable from n_0 if a path from n_0 to n_k exists
- Cycle
  - A path from node n_0 to n_0 where
    - No arc appears more than once
    - Only node n_0 appears more than once, at ends
  - Nodes and arcs may not repeat in a cycle
  - Acyclic; digraph with no cycles

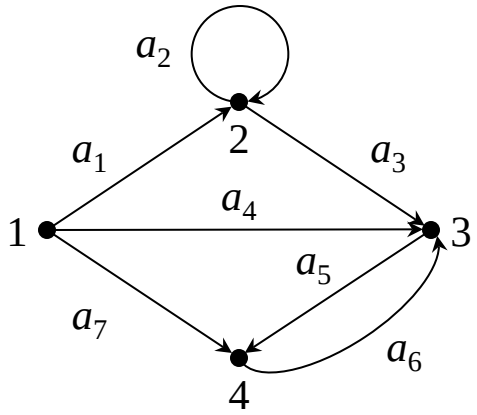
Example paths and cycles in digraphs

Paths

1, a_4, 3	length 1
1, a_1, 2, a_2, 2, a_2, 2, a_3, 3	length 4

Cycle

2, a_2, 2	length 1
3, a_5, 4, a_6, 3	length 2



Example 9, Figure 6.13

Graph proof example

Theorem
Any acyclic graph is simple.
If a graph is acyclic, then it is simple.

Proof (by contraposition; to prove $P \rightarrow Q$, show $Q' \rightarrow P'$)
Contrapositive: If a graph is not simple, then it has a cycle.

By definition, if a graph is not simple, it has parallel arcs or a loop. The parallel arcs and their endpoints form a cycle, or the loop and its endpoint forms a cycle.

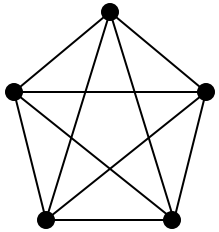
Thus, any acyclic graph is simple. ■

Example 10

Graph proof example

Theorem
Every complete graph is connected.

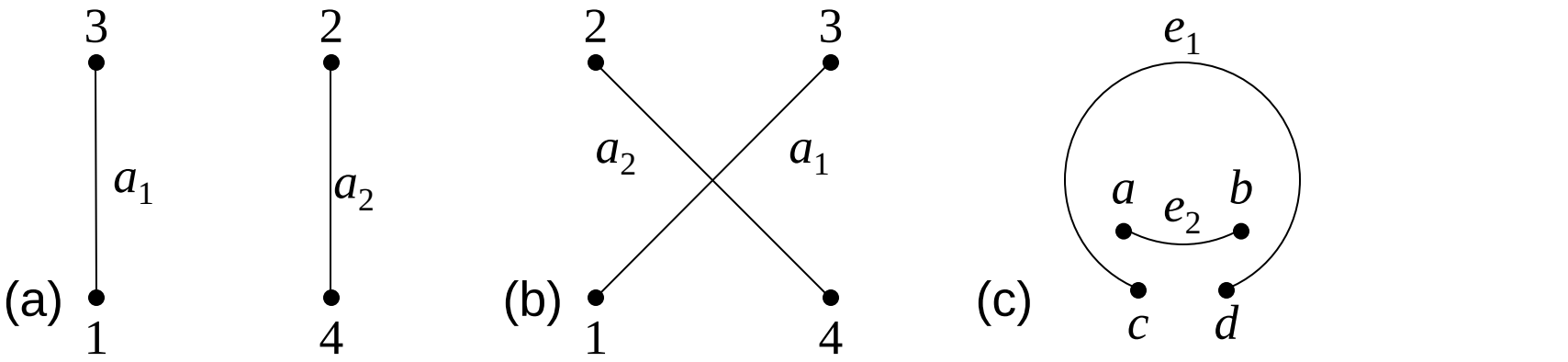
Proof
In a complete graph, any two distinct nodes are adjacent.
There is a path (length 1) from any node to any other node.
Therefore the graph is connected. ■



K_5

Isomorphic graphs concept

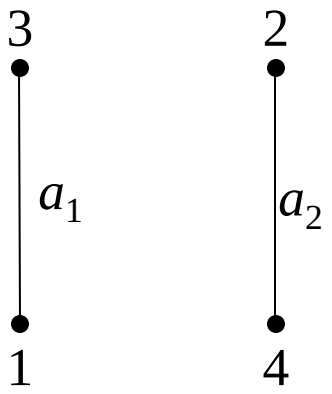
- Isomorphic; two graphs with same structure
  - Same number of nodes, same number of arcs
  - Same node and arc connections
- May appear different
  - Visual representation
  - Node and arc names



Figures 6.14, 6.15, 6.16

Isomorphic graphs definition

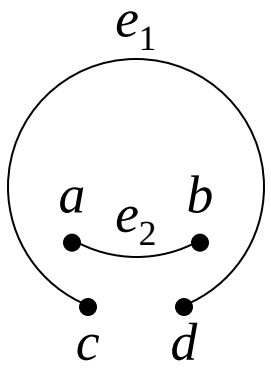
- Graphs (N_1, A_1, g_1) and (N_2, A_2, g_2) are isomorphic if there are bijections $f_1: N_1 \rightarrow N_2$ and $f_2: A_1 \rightarrow A_2$ such that for each arc $a \in A_1$, $g_1(a) = x–y$ if and only if $g_2[f_2(a)] = f_1(x)–f_1(y)$.
- The bijections need not be unique



$f_1: 1 \rightarrow a$
$2 \rightarrow c$
$3 \rightarrow b$
$4 \rightarrow d$

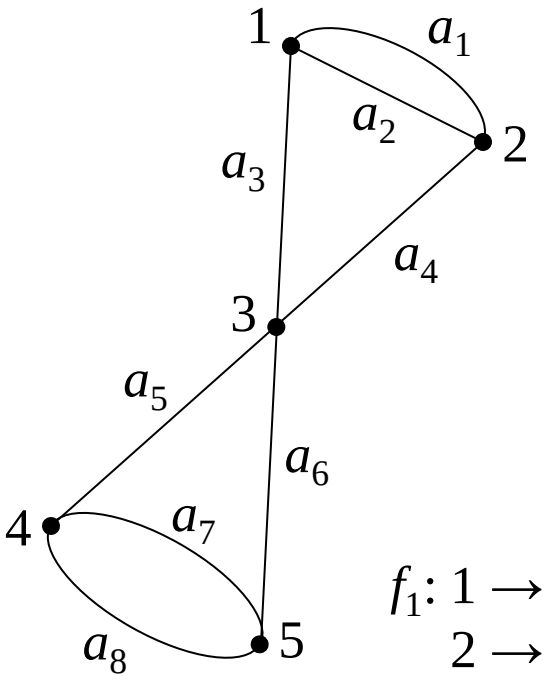
$f_2: a_1 \rightarrow e_2$
$a_2 \rightarrow e_1$

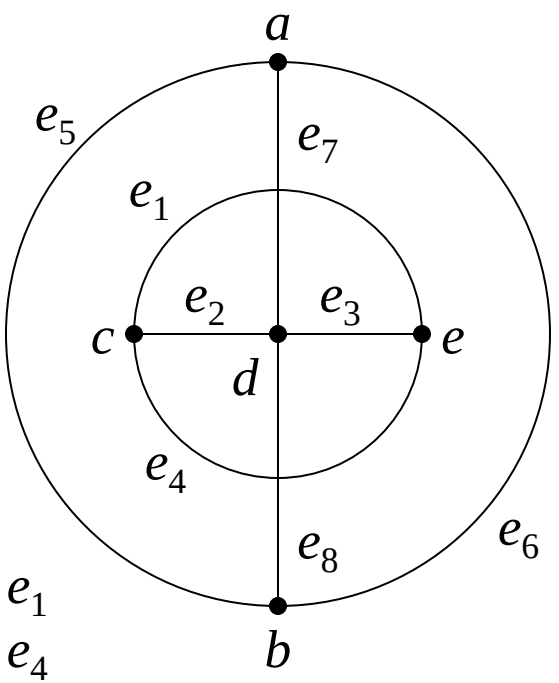
?



Figures 6.14, 6.16

Isomorphic graphs example





f_1: $1 \rightarrow c$, $2 \rightarrow e$, $3 \rightarrow d$, $4 \rightarrow b$, $5 \rightarrow a$

f_2: $a_1 \rightarrow e_1$, $a_2 \rightarrow e_4$, $a_3 \rightarrow e_2$, . . .

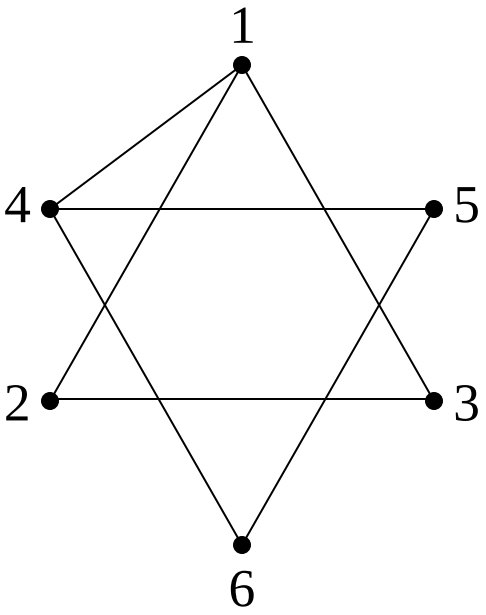
e.g., $g_1(a_3) = 1\text{–}3$ and $g_2[f_2(a_3)] = g_2(e_2) = c\text{–}d = f_1(1)\text{–}f_1(3)$

Figure 6.17

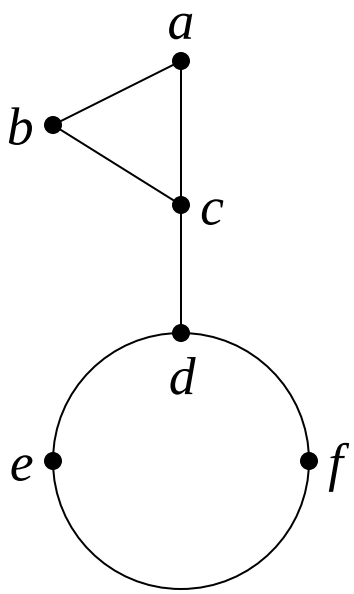
Simple graph isomorphism definition

- Simple graphs
  - No parallel arcs, no loops
  - Isomorphism easier to define and prove
- Formal definition
  - Two simple graphs (N_1, A_1, g_1) and (N_2, A_2, g_2) are isomorphic if there is a bijection $f_1: N_1 \rightarrow N_2$ such that for any nodes $n_i, n_j \in N_1$, n_i and n_j adjacent if and only if $f(n_i)$ and $f(n_j)$ adjacent
  - Bijection f is called an isomorphism

Isomorphic simple graphs example



f: $1 \rightarrow d$
$2 \rightarrow e$
$3 \rightarrow f$
$4 \rightarrow c$
$5 \rightarrow b$
$6 \rightarrow a$



Practice 8, Figure 6.18

Graph isomorphism proofs

- Proving graphs isomorphic
  - Graphs; give bijections that preserve structure
  - Simple graphs; give bijection that preserves structure
- Proving graphs not isomorphic
  - Prove that bijection(s) do not exist
    - Try all possible bijections; $n!$ for $|N| = n$
  - Prove that bijection(s) can not exist
    - One graph has more nodes than the other
    - One graph has more arcs than the other
    - One graph has parallel arcs and the other does not
    - One graph has a loop and the other does not
    - One graph has a node of degree k and the other does not
    - One graph is connected and the other is not
    - One graph has a cycle and the other does not

Graph isomorphism proof examples

Two graphs G_1 and G_2 are not isomorphic if ...

a. … one has more nodes than the other.
There cannot be a bijection between
the two node sets if they are not the same size. ■

e. … one has a node of degree k and the other does not.
A node n of degree k in G_1 is an endpoint to k arcs.
The image of n in G_2 (under isomorphism bijection)
must be an endpoint to the images of k arcs,
so it must have degree k as well. ■

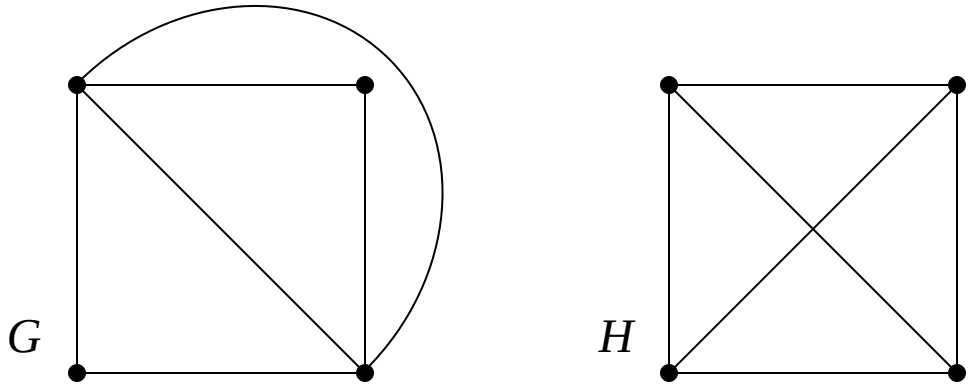
Exercise 21ae

Graph isomorphism proof example

Theorem
Graphs G and H (below) are not isomorphic.

Proof
Graph G has a node of degree 2; H does not. ■

Alternate proof
Graph G has parallel arcs; H does not. ■



Practice 9, Figure 6.19

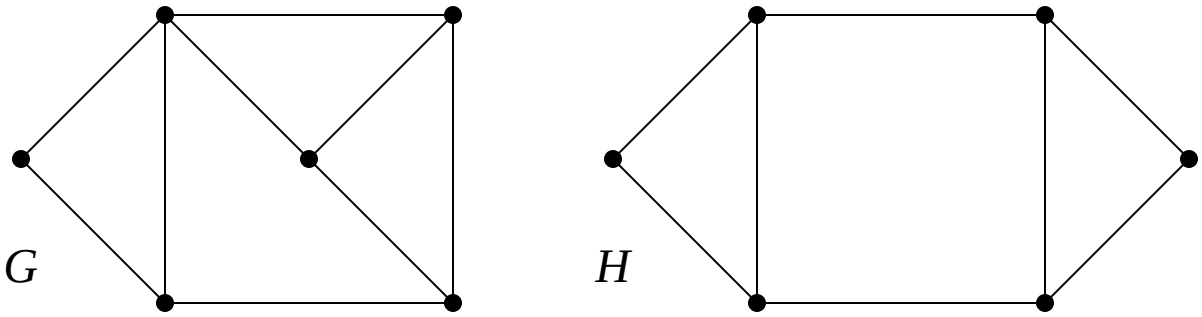
Graph isomorphism proof example

Theorem

Graphs G and H are not isomorphic.

Proof

Graph G has 9 edges; H has 8 edges. ■

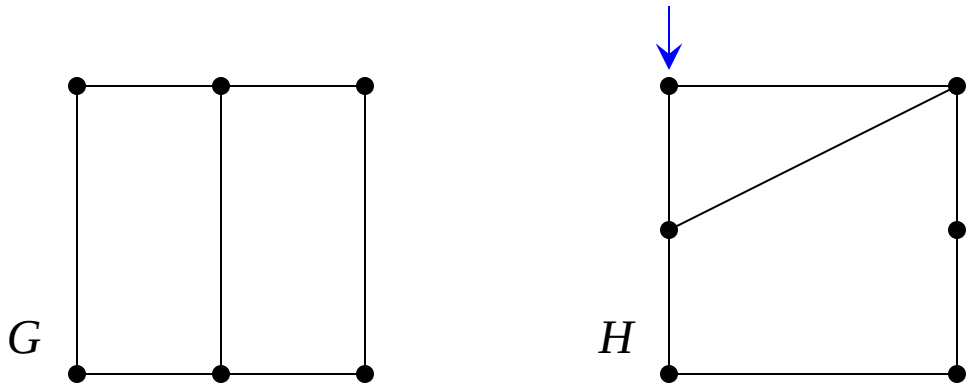


Example 10.4.3, p. 679, S. S. Epp, *Discrete Mathematics with Applications, Fourth Edition*, Brooks/Cole, Boston MA, 2011.

Graph isomorphism proof example

Theorem
Graphs G and H (below) are not isomorphic.

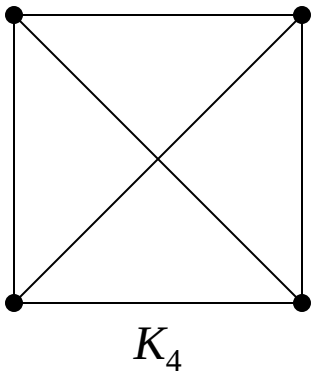
Proof
Graph H has a node of degree 2 adjacent to two nodes of degree 3; G does not. ■



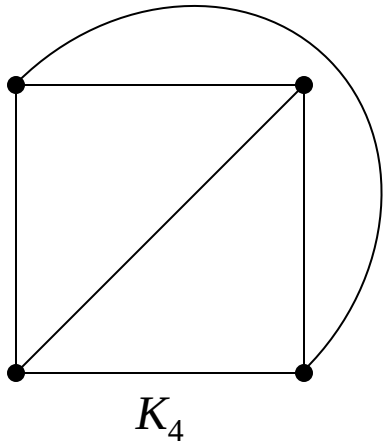
Example 12, Figure 6.20

Planar graphs

- Graph visual representation vs graph structure
  - Graph visual representation usually not important
  - Planarity, a characterisitic of visual representation, reveals important aspect of structure
- Definition
  - Planar; graph can be drawn in the plane so that arcs intersect only at nodes



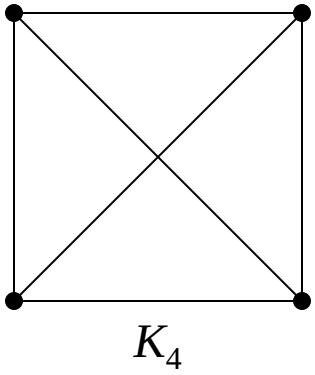
K_4



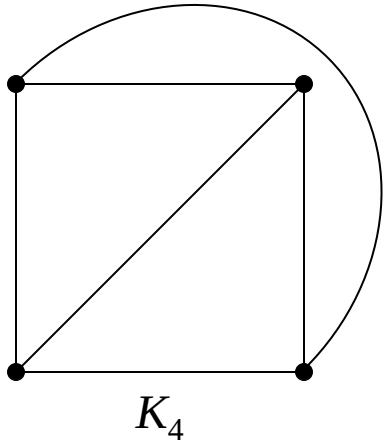
K_4

Proving planarity

- Proofs of planarity
  - Drawing graph planar
- Invalid proofs of non-planarity
  - Drawing graph non-planar
  - Failure to find a planar drawing
- Valid proofs of non-planarity
  - Kuratowski's Theorem (later)



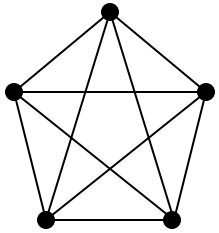
K_4



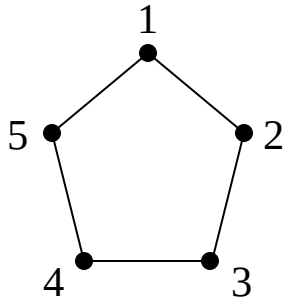
K_4

Non-planar graph example

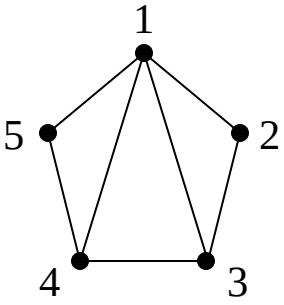
K_5 is simple complete graph with 5 nodes.
All attempts to draw K_5 planar fail;
this suggests, but does not prove, that K_5 is non-planar.



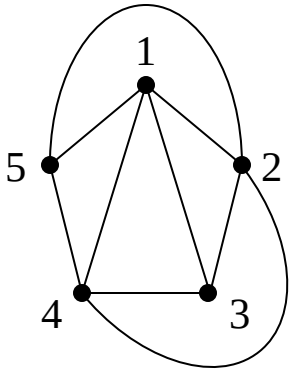
K_5



(a)



(b)



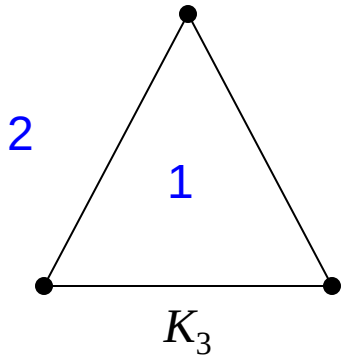
(c)

Where to put arc 3–5?

Example 13, Figure 6.21

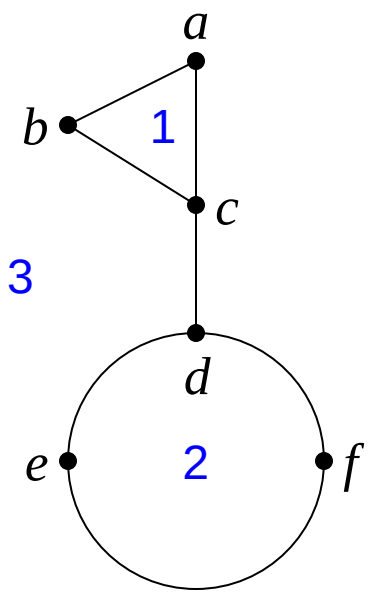
Euler's formula

- Important property of planar graphs
  - Simple, connected, planar graph drawn planar
- Euler's formula: $n - a + r = 2$
  - n = number of nodes
  - a = number of arcs
  - r = number of regions

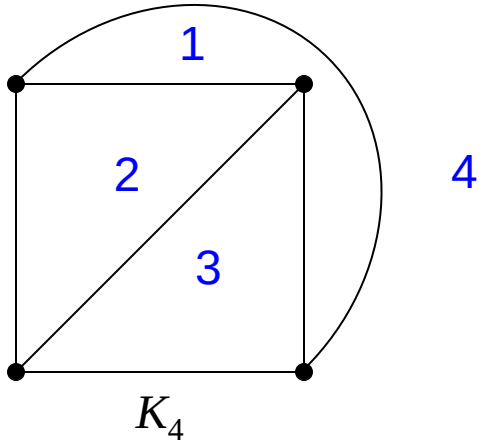


$n = 3$
$a = 3$
$r = 2$
$3 - 3 + 2 = 2$

Euler's formula examples



$n = 6$
$a = 7$
$r = 3$
$6 - 7 + 3 = 2$



$n = 4$
$a = 6$
$r = 4$
$4 - 6 + 4 = 2$

Proving Euler's formula

Theorem
Given a simple connected planar graph drawn in the plane, $n - a + r = 2$.

Proof (by induction on a, number of arcs)

Basis. Let $a = 0$; the graph consists of a single node. Then $1 - 0 + 1 = 2$ and the formula holds.

Induction hypothesis. Assume $n - a + r = 2$ for any simple connected planar graph with k arcs.

Figure 6.22(a)

Inductive step. Consider a simple connected planar graph with $k + 1$ arcs. There are two cases:

Case 1. The graph has a node of degree 1.
Temporarily erase this node and is connecting arc.
This leaves a simple connected planar graph with k arcs, some number n nodes, and some number r regions.
By the inductive hypothesis $n - k + r = 2$ in this graph.

The original graph had one more node and one more arc, giving $(n + 1) - (k + 1) + r = 2$, true by IH and algebra.

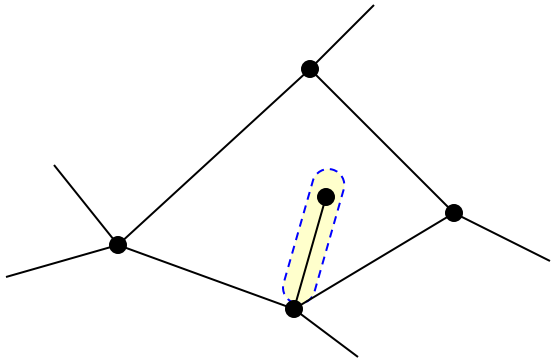


Figure 6.22(b)

Case 2. The graph does not have a node of degree 1. Temporarily erase one arc that defines an enclosed region. This leaves a simple connected planar graph with k arcs, some number n of nodes, and some number r of regions. By the inductive hypothesis $n - k + r = 2$ in this graph.

The original graph had one more arc and one more region, giving $n - (k + 1) + (r + 1) = 2$, true by IH and algebra. ■

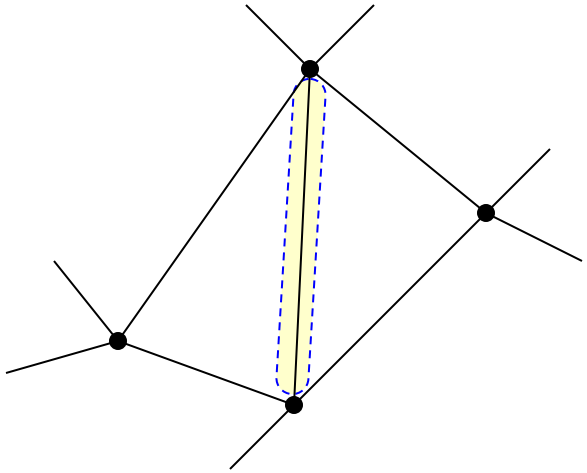


Figure 6.22(c)

Euler's formula special case 1

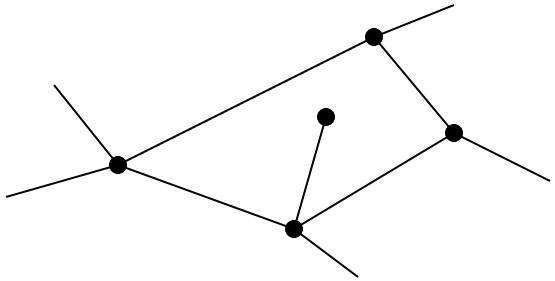
Theorem
Given a simple connected planar graph with $n \geq 3$ drawn in the plane, $a \leq 3n - 6$.

Proof
Define "region edge" as an arc on the boundary of a region.
Each arc forms 2 region edges, giving $2a$ total.
No loops, thus no regions with 1 region edge.
No parallel arcs, thus no regions with 2 region edges.
Thus each region has ≥ 3 region edges, giving $\geq 3r$ total.

$2a \geq 3r$

$2a \geq 3(2 - n + a) = 6 - 3n + 3a$

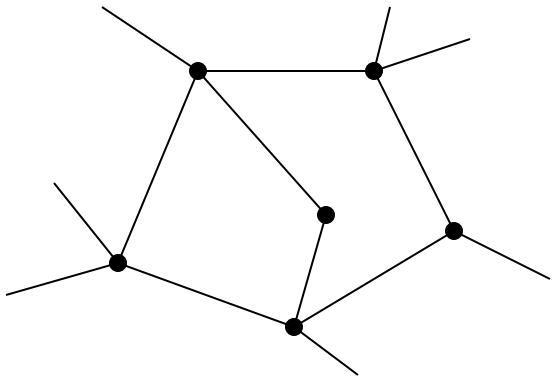
$a \leq 3n - 6$ ■



Euler's formula special case 2

Theorem
Given a simple connected planar graph with $n \geq 3$
and no cycles of length 3 drawn in the plane, $a \leq 2n - 4$.

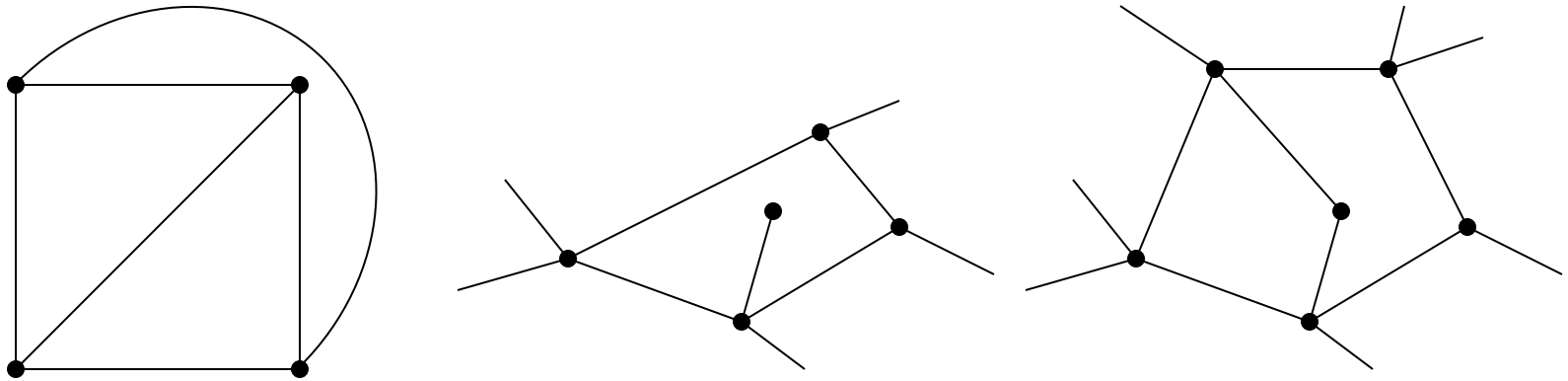
Proof
Because there are no cycles of length 3,
each region has ≥ 4 region edges, giving $\geq 4r$ total.
$2a \geq 4r$
$2a \geq 4(2 - n + a) = 8 - 4n + 4a$
$a \leq 2n - 4$ ■



Theorem on the number of nodes and arcs

For a simple connected planar graph
with n nodes and a arcs:

1. If the planar representation divides the plane into r regions, then $n - a + r = 2$. (Euler's formula)
2. If $n \geq 3$, then $a \leq 3n - 6$. (Special case 1)
3. If $n \geq 3$ and there are no cycles of length 3, then $a \leq 2n - 4$. (Special case 2)

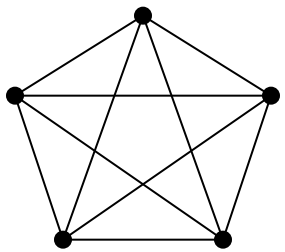


Proving non-planarity with the formulas

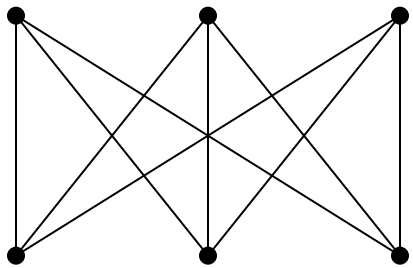
Euler's formula and its special cases can be used to prove graphs are non-planar.

For K_5, inequality 2 does not hold.
$a \leq 3n - 6$, but for K_5 $10 > 3 \cdot 5 - 6$; thus K_5 is non-planar.

For $K_{3,3}$, inequality 3 does not hold.
$a \leq 2n - 4$, but for $K_{3,3}$ $9 > 2 \cdot 6 - 4$; thus $K_{3,3}$ is non-planar.



K_5



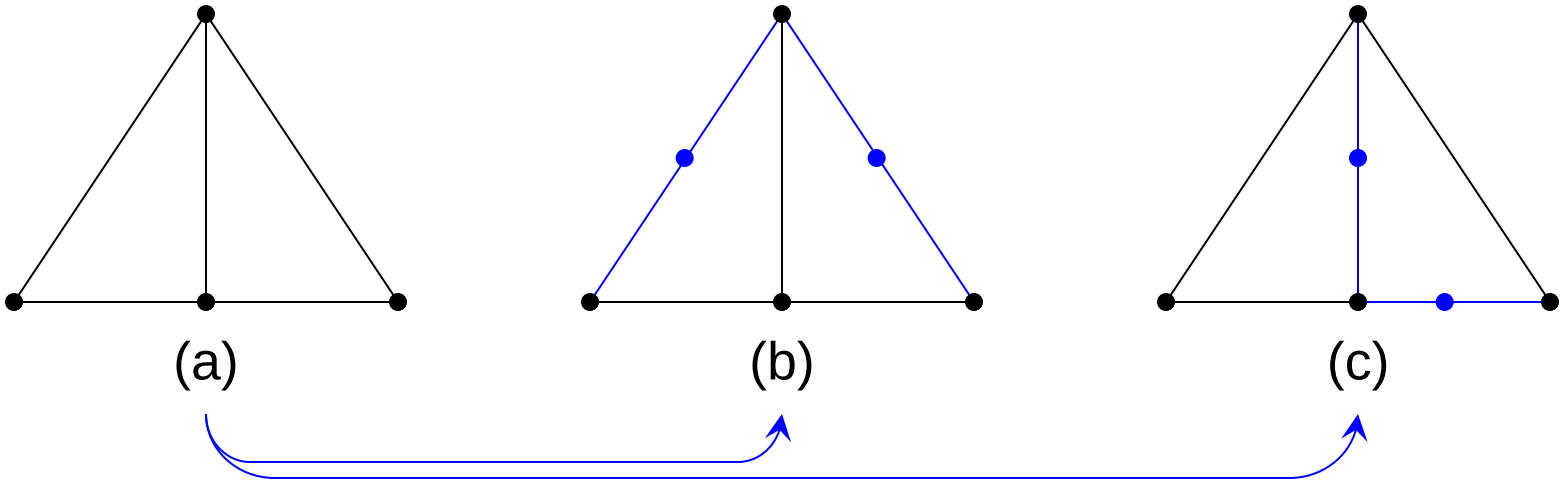
$K_{3,3}$

Example 14

Homeomorphic graphs

?

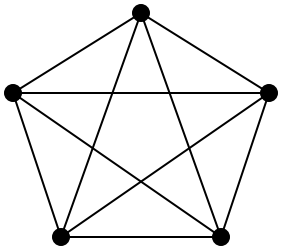
- Elementary subdivision; replacing single arc x–y with two new arcs x–v and v–y and new node v
- Homeomorphic; two graphs that can be obtained from the same graph by elementary subdivisions

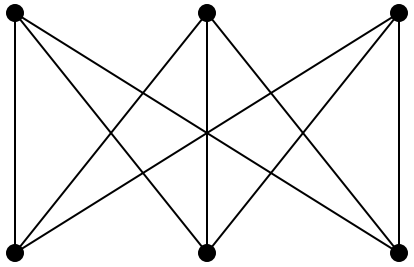


Example 15, Figure 6.23

Kuratowski's theorem

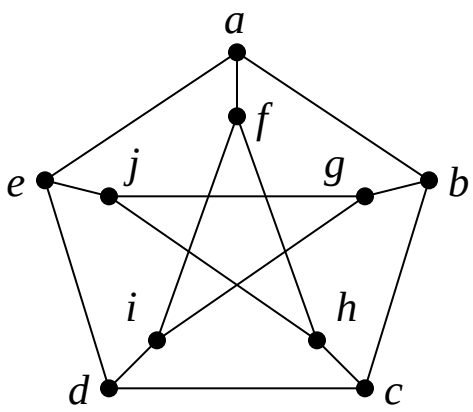
- Homeomorphism and planarity
  - Elem subs can not make planar graph non-planar
  - Elem subs can not make non-planar graph planar
- K_5 and $K_{3,3}$ non-planar by Euler's formula
- Kuratowski's theorem
  - A graph is non-planar if and only if it contains a subgraph homeomorphic to K_5 or $K_{3,3}$
  - Note only if; every non-planar graph contains K_5 or $K_{3,3}$ (or their homeomorphs)

K_5 

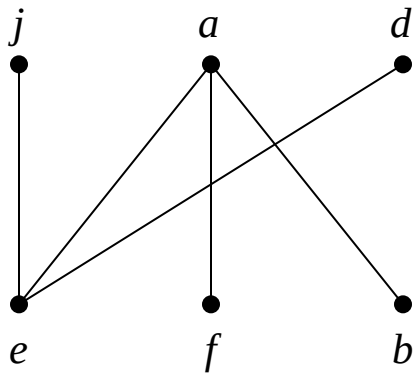
$K_{3,3}$ 

Kuratowski's theorem example

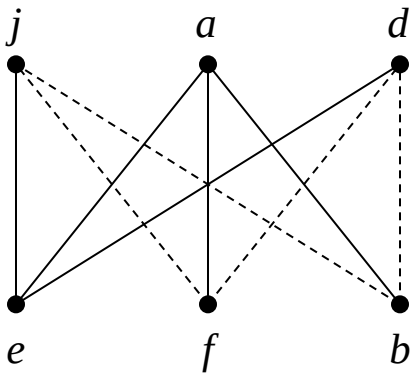
Petersen graph non-planar;
it contains subgraph homeomorphic to $K_{3,3}$.



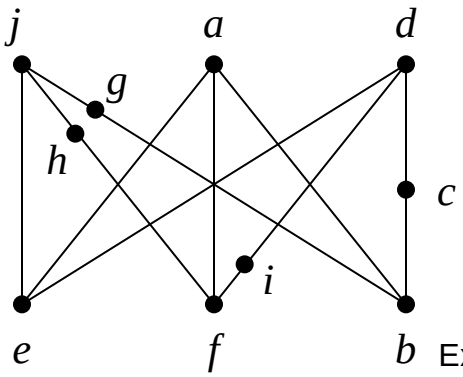
(a)
Find $K_{3,3}$ in
Petersen graph



(b)
Arcs of $K_{3,3}$ in
Petersen graph



(c)
Arcs needed to
complete $K_{3,3}$

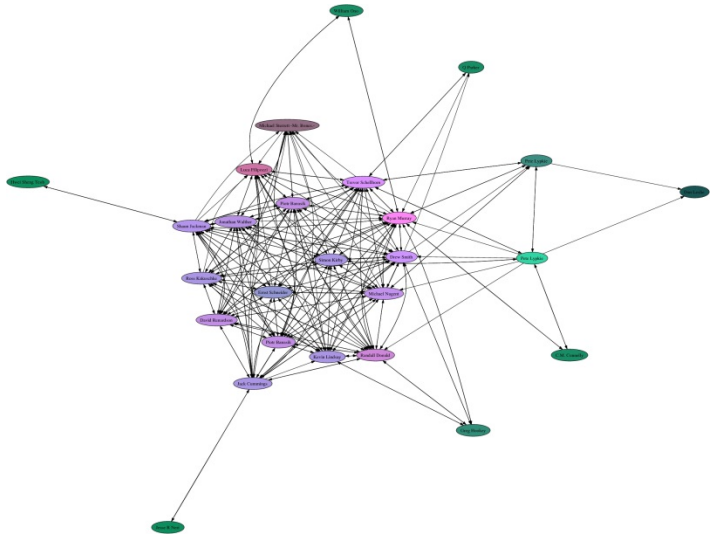


(d)
Remaining arcs
are elem subs

Example 16, Figure 6.24

Computer representation of graphs

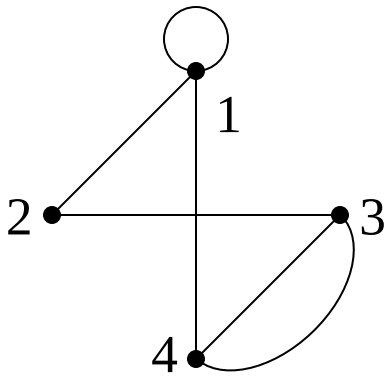
- Storing graphs in digital form
  - Image (e.g., .jpg) of visual representation
  - Data structure of graph nodes and arc connections
- Graph data structures
  - Adjacency matrix
  - Adjacency list



Adjacency matrix

- Assign numbers to nodes
  - $N = \{n_1, n_2, \ldots, n_n\}$ where $n = |N|$
  - Numbering arbitrary, node set unordered
- Store graph in $n \times n$ matrix $\mathbf{A}$
  - $a_{ij} = p$, where there are p arcs between n_i and n_j
  - $p = 0$ means no arcs, $p > 1$ means parallel arcs

?



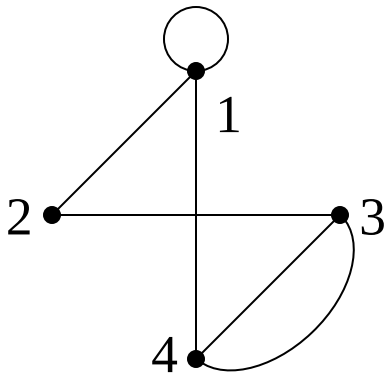
$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \\ 1 & 0 & 2 & 0 \end{bmatrix}$$

Example 17, Figure 6.25

Practice 16

Adjacency matrix symmetry

- Adjacency matrix for undirected graph symmetric along diagonal
  - Always true for undirected graph
  - p arcs between n_i and $n_j \leftrightarrow p$ arcs between n_j and n_i
- Only "lower triangular" portion need be stored
  - Space saving important for large graphs

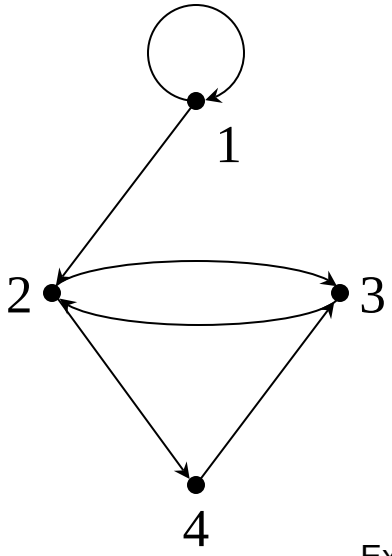


$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \\ 1 & 0 & 2 & 0 \end{bmatrix}$$

Figure 6.25

Adjacency matrix for directed graphs

- Adjacency matrix **A** stores arc direction
  - $a_{ij} = p$, where there are p arcs from n_i to n_j
  - $p = 0$ means no arcs, $p > 1$ means parallel arcs
- Adjacency matrix for digraph not necessarily symmetric

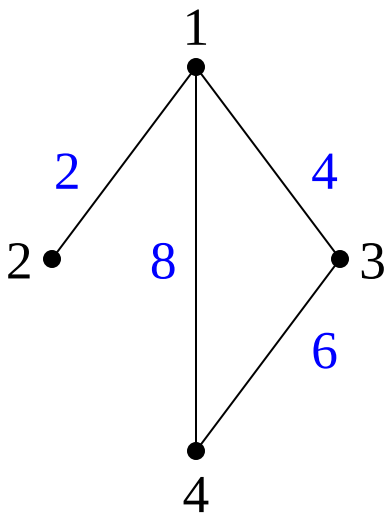


$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Example 18, Figure 6.26

Adjacency matrix for simple weighted graph

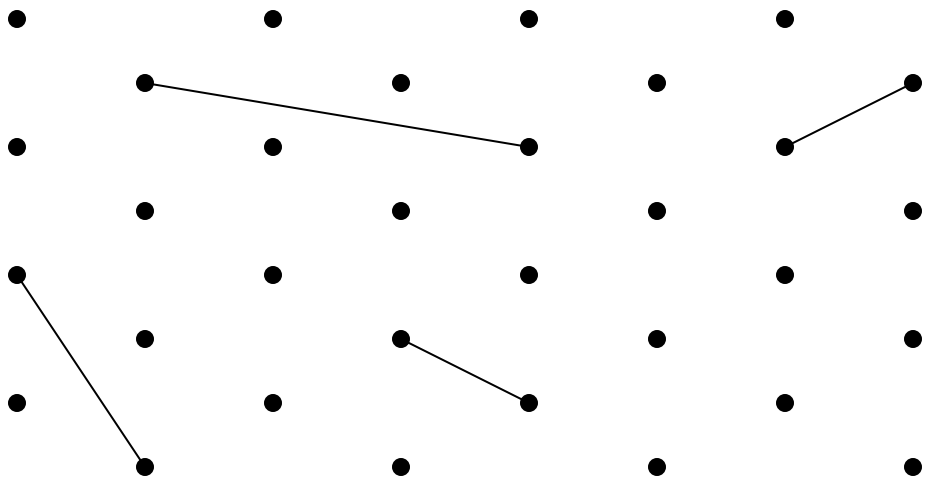
- Simple graph: no loops or parallel arcs
  - i.e., 0 or 1 arcs between any two nodes
- Representing weighted graph in matrix **A**
  - $a_{ij} = w$, where w is weight of arc n_i–n_j
  - $a_{ij} = 0$ means no arc n_i–n_j



$$\mathbf{A} = \begin{bmatrix} 0 & 2 & 4 & 8 \\ 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 6 \\ 8 & 0 & 6 & 0 \end{bmatrix}$$

Representing sparse graphs

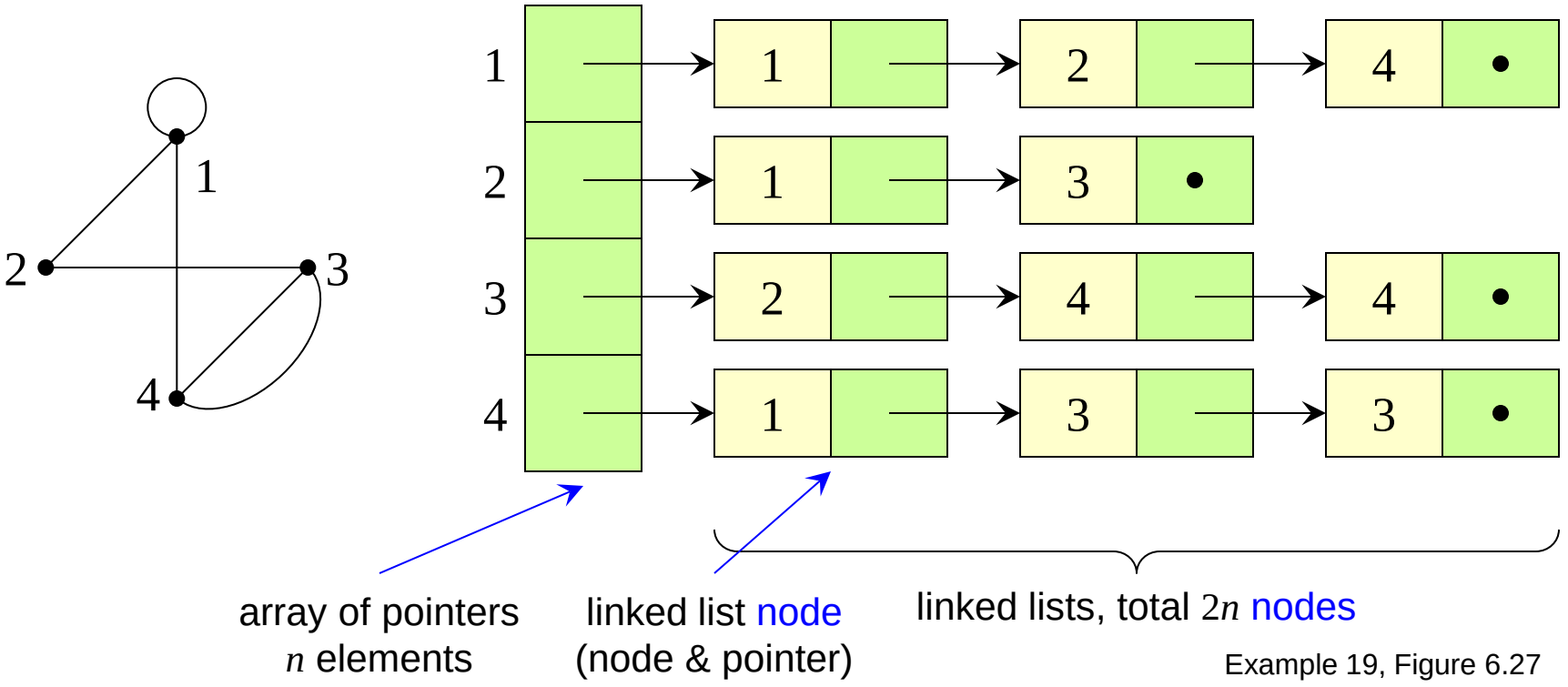
- Definition and representation
  - Sparse graph; large graph (many nodes) with few arcs
  - Adjacency matrix mostly 0s, yet uses n^2 space
- Processing sparse graph adjacency matrices
  - Accessing all arcs $O(n^2)$
  - Accessing all arcs connecting to a node $O(n)$



Adjacency list

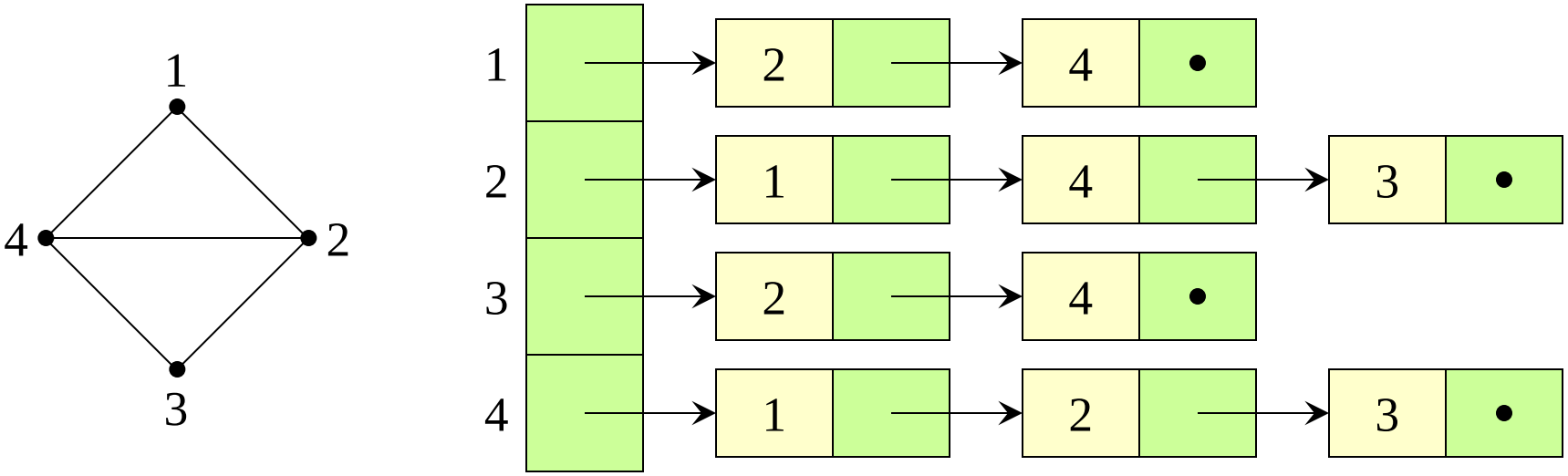
- Represent graph as linked list
  - For each node, list holds nodes adjacent to it
  - Pointers connect each node to next in list

?



Example 19, Figure 6.27

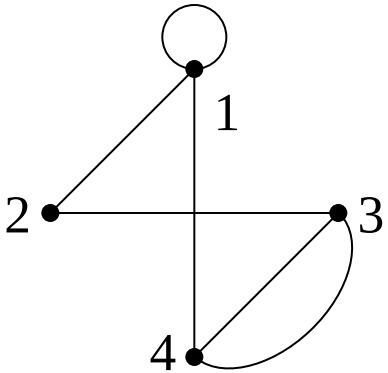
Adjacency list example

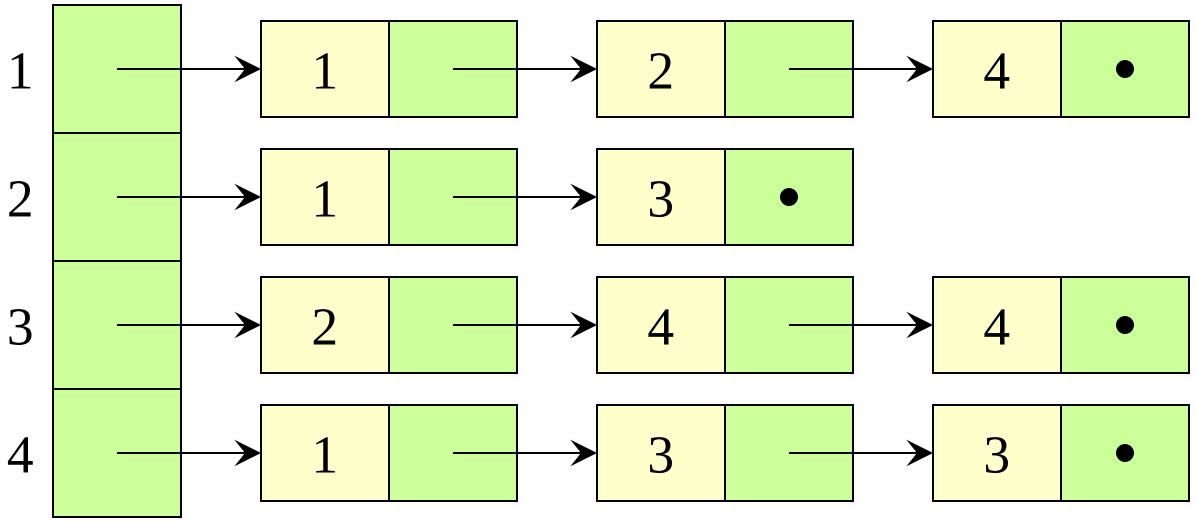


Practice 17, Figure 6.28

Adjacency list algorithms

- Accessing specific arc n_i–n_j
  - Search list i for j (or vice versa)
- Processing all …
  - … nodes adjacent to given node n_i; traverse list i
  - … arcs in graph; traverse all lists 1 to n (each arc 2x)

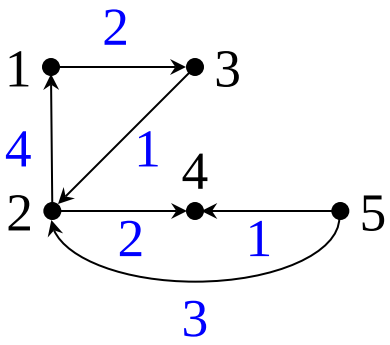


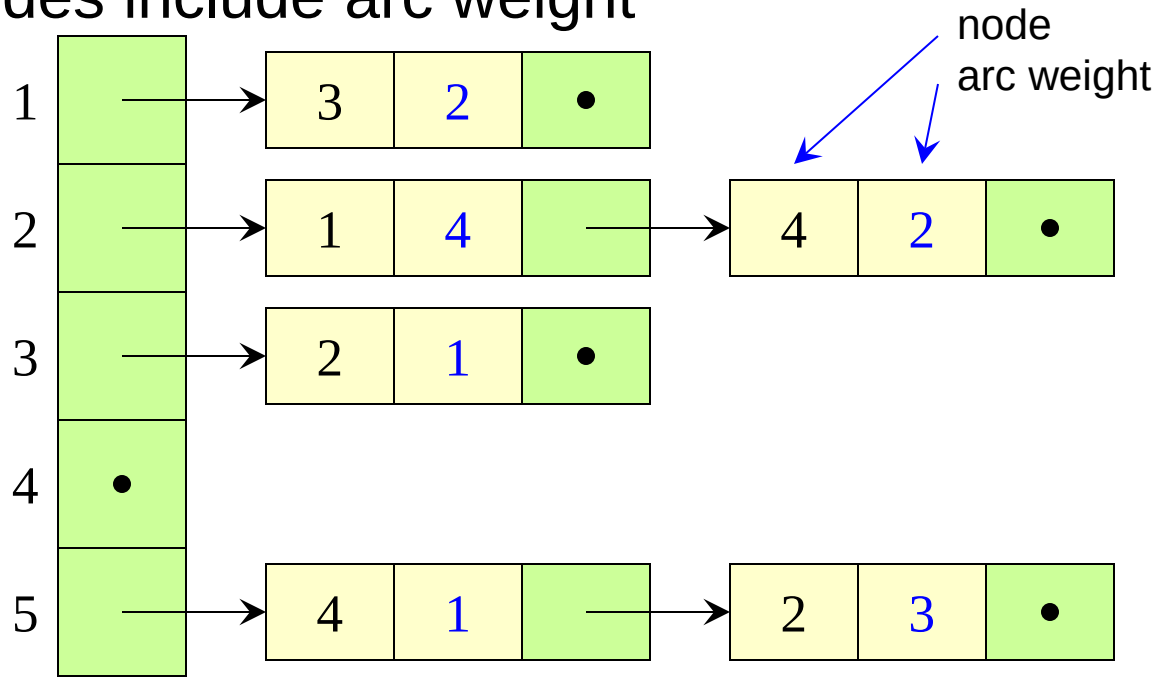


Example 19, Figure 6.27

Adjacency lists for directed & weighted graphs

- Directed graphs
  - Arc from n_i to n_j on list for n_i, not n_j
  - Each arc appears once, not twice as for undirected
- Weighted graph
  - Linked list nodes include arc weight

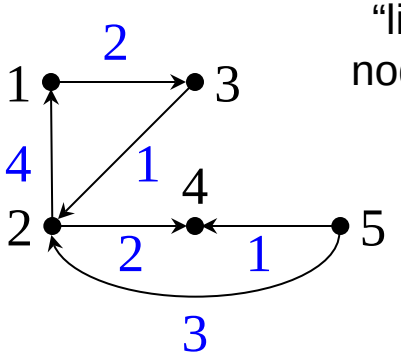




Example 20, Figure 6.29

Adjacency lists without pointers

- Concept
  - In programming language without pointers and dynamic memory, store adjacency list in array
  - Each node is row in array



"list" headers
node n_i at row i

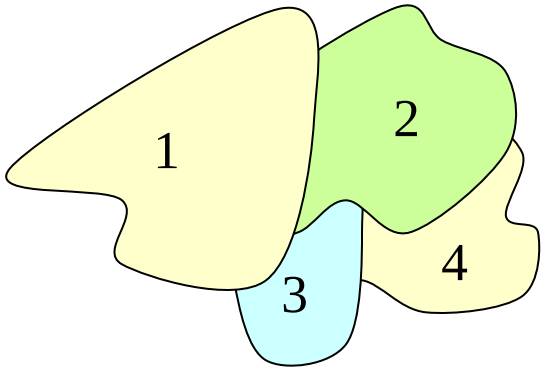
	Node	Weight	Pointer
1			6
2			7
3			9
4			0
5			10
6	3	2	0
7	1	4	8
8	4	2	0
9	2	1	0
10	4	1	11
11	2	3	0

0 indicates end of list

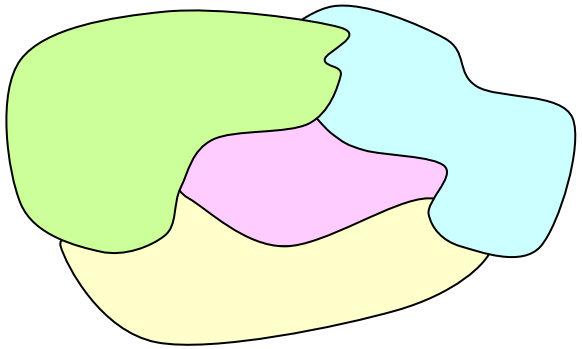
Example 21, Figure 6.30

Graph coloring

- Problem statement
  - Given a map of multiple regions (countries, states, …) on a sheet of paper, assign each a color
  - Regions with common border must be different colors
  - Regions that meet only a point may have same color
- Question
  - What is the minimum number of colors required?



4 regions, 3 colors

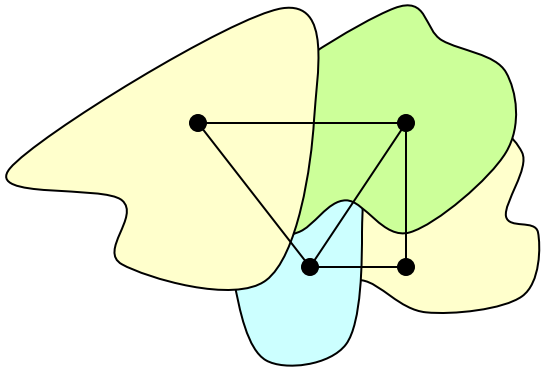


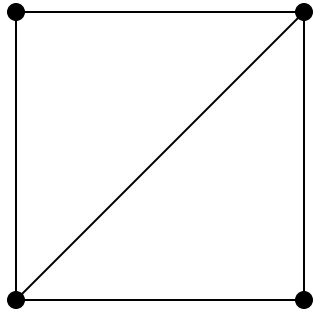
4 regions, 4 colors

Exercises 77, 78

Maps and dual graphs

- Dual graph; graph associated with a map
  - Define one node per region
  - Connect two nodes with an arc
    iff their corresponding regions share a common border
- Maps “on a sheet of paper” produce
  dual graphs that are simple, connected, planar

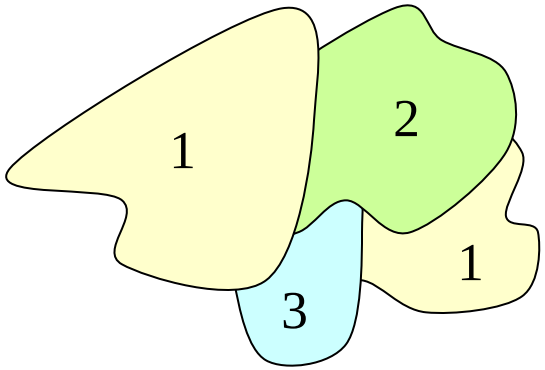


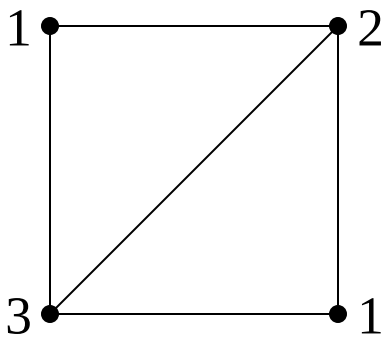


Exercise 79a

Map coloring as a graph problem

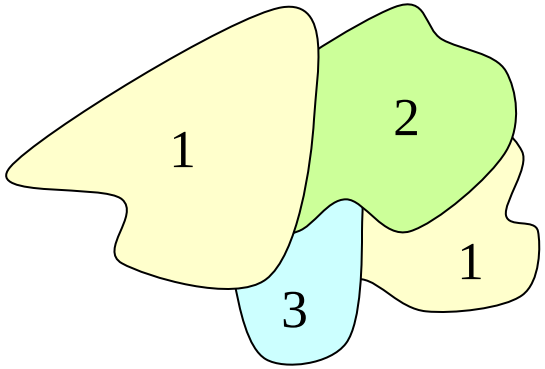
- Dual problems
  - Assign colors to map regions so that no two regions with common border get same color
  - Assign “colors” (numbers) to dual graph nodes so that no two adjacent nodes get the same color
- Question
  - What is minimum number of colors required?

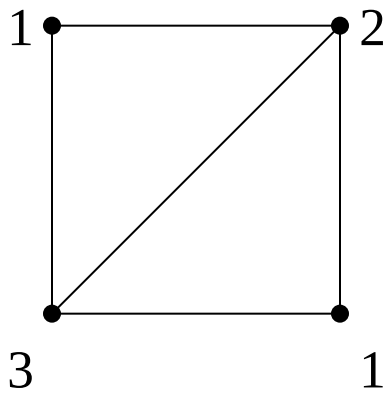




Graph coloring terminology

- Definitions
  - Chromatic number; for a graph,
    the minimum number of colors required to color it
  - n-colorable; graph which can be colored with n colors



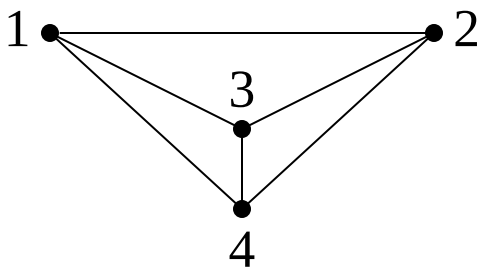


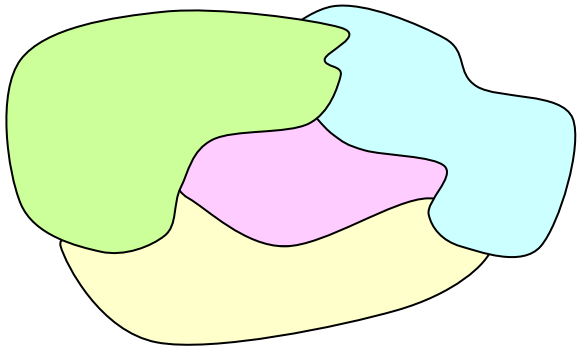
3-colorable

Exercise 80

Four color problem

- Four color problem
  - At least 4 colors are required for some graphs
  - Is more than 4 colors ever needed?
- Problem statements
  - Informal: What is the maximum number of colors needed for any graph?
  - Formal: the chromatic number of any simple, connected, planar graph is at most 4





Exercise 81

Graph coloring application example

Problem

Five political lobbyists $\{1, 2, 3, 4, 5\}$ are visiting seven members of Congress $\{A, B, C, D, E, F, G\}$.

The members of Congress the lobbyists must see are:

1. A, B, D
2. B, C, F
3. A, B, D, G
4. E, G
5. D, E, F

More than one lobbyist may meet with a member of Congress during a meeting, but not vice versa.

How many meeting time slots are needed?

Exercise 85

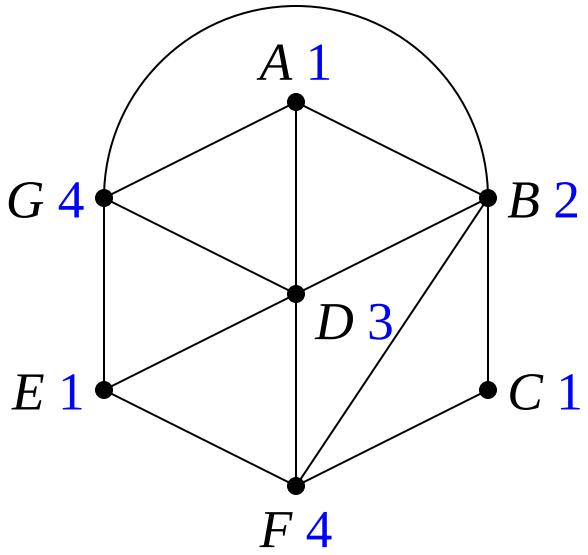
Solution

Let the members of Congress be the nodes of a graph and let node colors be time slots.

Include arc x–y iff any lobbyist must see both x and y, thus requiring different time slots for the two meetings.

Adjacent nodes must be assigned different colors.

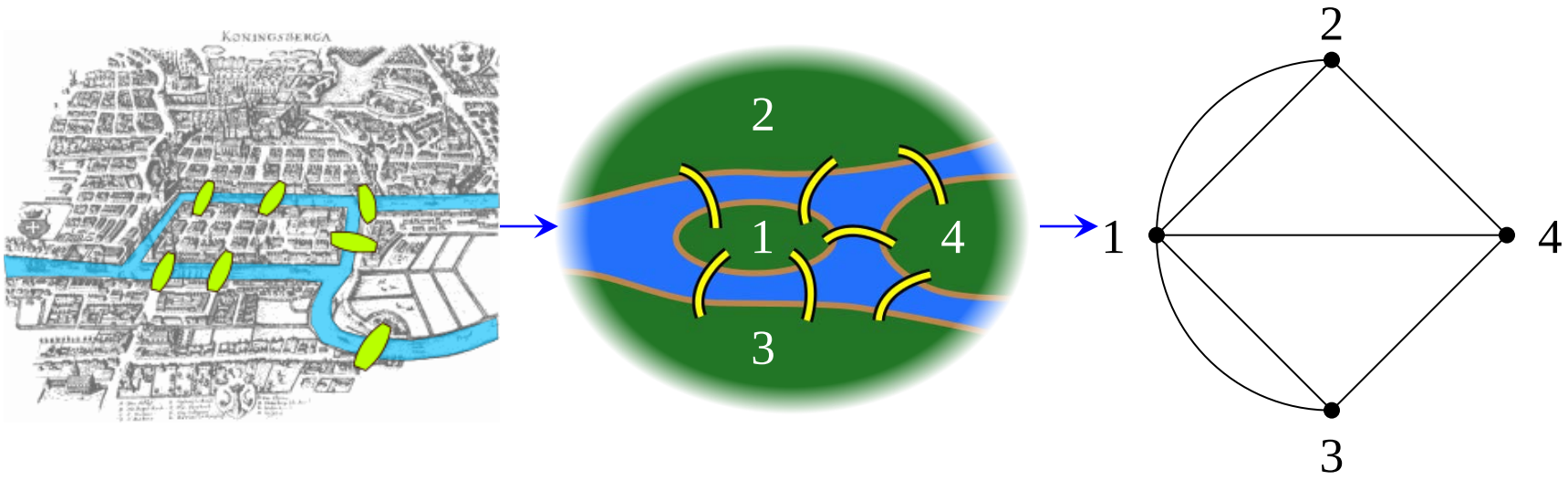
The resulting graph is simple, connected, and planar, and requires 4 colors, hence 4 time slots are required.



Exercise 85

Solution to sample problem 2

Seven Bridges of Königsberg
Is it possible to find a walk that crosses each of the seven bridges exactly once?



Convert problem to graph.

Euler walk. A path in a graph that traverses each edge exactly once.

Theorem. A connected graph has an Euler walk iff it has exactly 0 or 2 nodes of odd degree. (Euler, 1735)

Proof (informal). A walk must enter and leave any node other than the start and end nodes 1 or more times; thus any non-start and non-end node must have even degree. The start and end nodes may have odd degree (for start, leave but do not enter, or for end, enter but do not leave), or if they are the same node, it must have even degree. ■

Thus the answer to the Seven Bridges of Königsberg problem is “no”; the graph does not meet the requirement of the theorem (it has 4 nodes with odd degree).

Euler
1707-1783

Section 6.1 homework assignment

See homework list for specific exercises.

6.2 Trees

Trees

- Definitions
  - Tree; acyclic connected graph
  - Root; designated node of the tree
  - Free tree or nonrooted tree; tree with no root

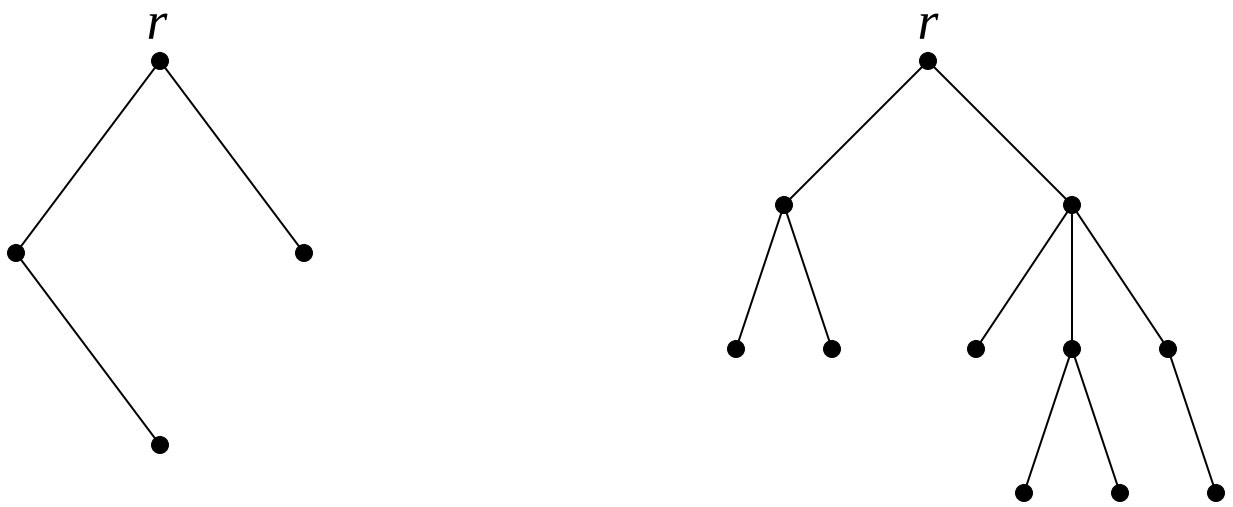
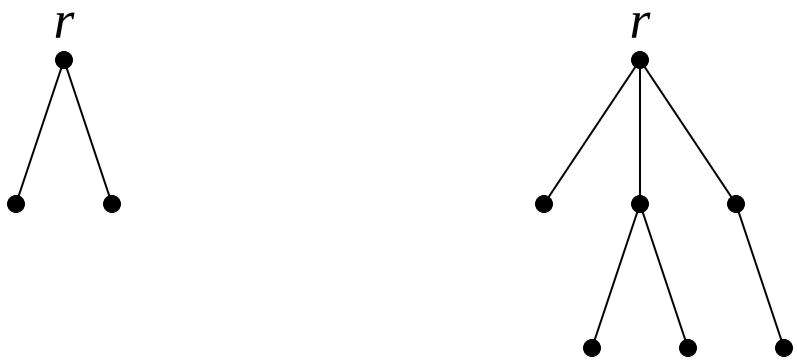


Figure 6.31

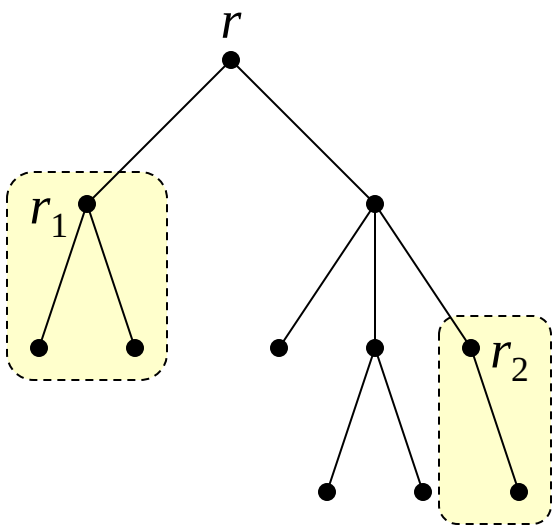
Tree terminology

- Basic terms
  - Depth; for a node, length of path from root to node
  - Depth; for a tree, max depth of any node, aka height
  - Parent; for a node, adjacent node closer to root
  - Child; for a node, adjacent node farther from root
  - Leaf; node with no children
  - Internal node; node with children, aka nonleaf node
  - Forest; disjoint collection of trees



Subtrees

- Definition
  - Subtree; given a tree, a portion of that tree obtained by treating one of its nodes as a root
  - Subtrees are always trees



Recursive definition of a tree

- Definition
  - Basis: a single node is a tree (the node is the root)
  - Recursive step: if $T_1, T_2, \ldots, T_t$ are disjoint trees with roots $r_1, r_2, \ldots, r_t$ respectively, the graph formed by connecting a new node r with a single arc to $r_1, r_2, \ldots, r_t$ is a tree with root r

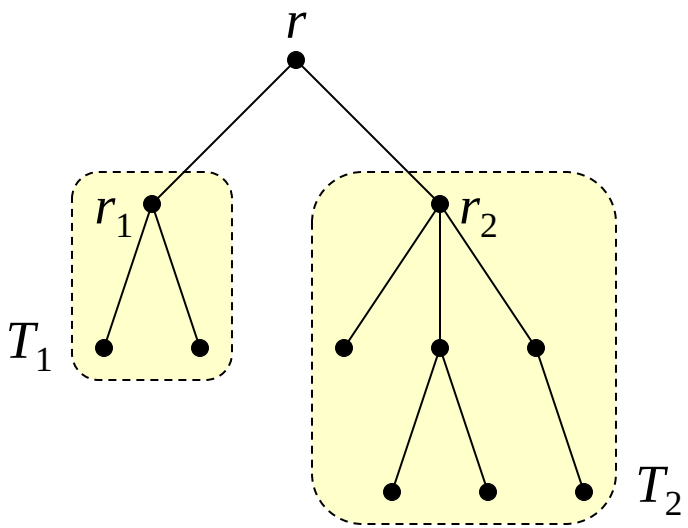
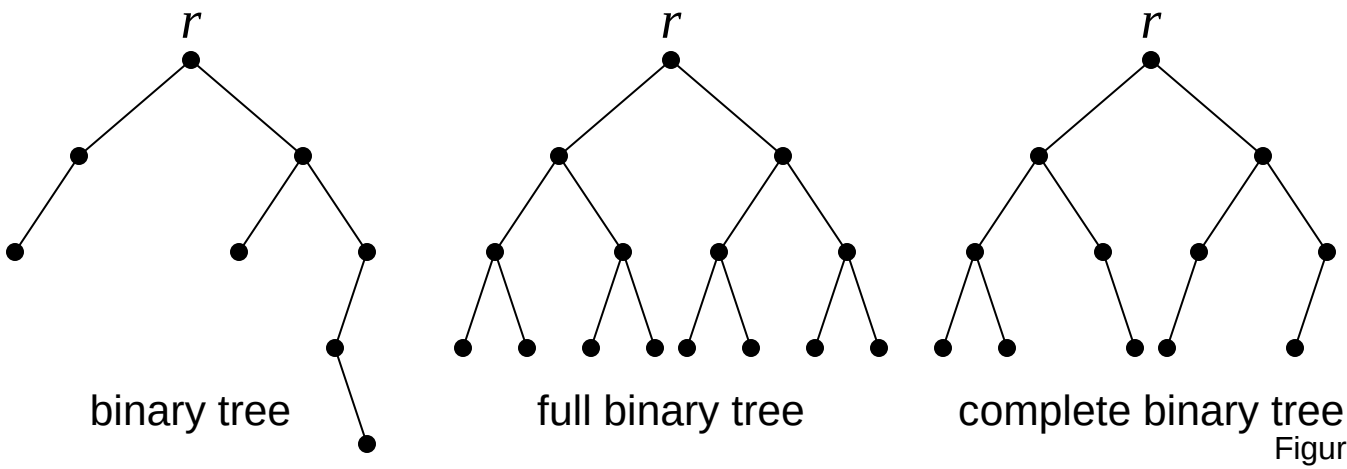


Figure 6.32

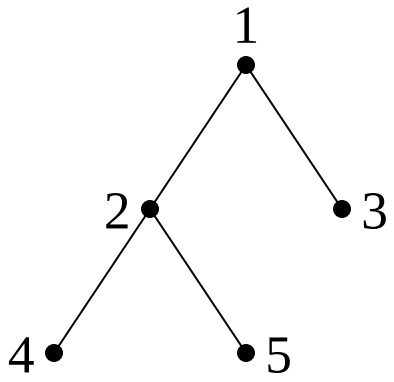
Binary trees

- Definitions
  - Binary tree; tree where each node has at most two children, designated left child and right child
  - Full binary tree; all internal nodes have two children and all leaves are at same depth
  - Complete binary tree; "almost full", bottom level may not have all leaves

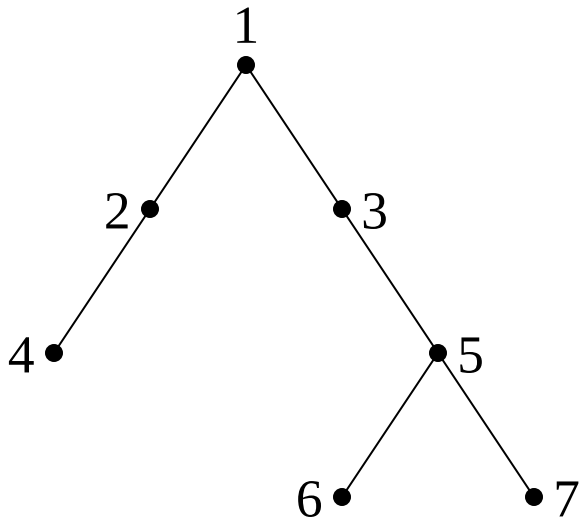


Figures 6.33, 6.34, 6.35

Example binary trees



Depth (height) of tree: 2
Left child of node 2: node 4
Depth of node 5: 2
Depth of node 3: 1



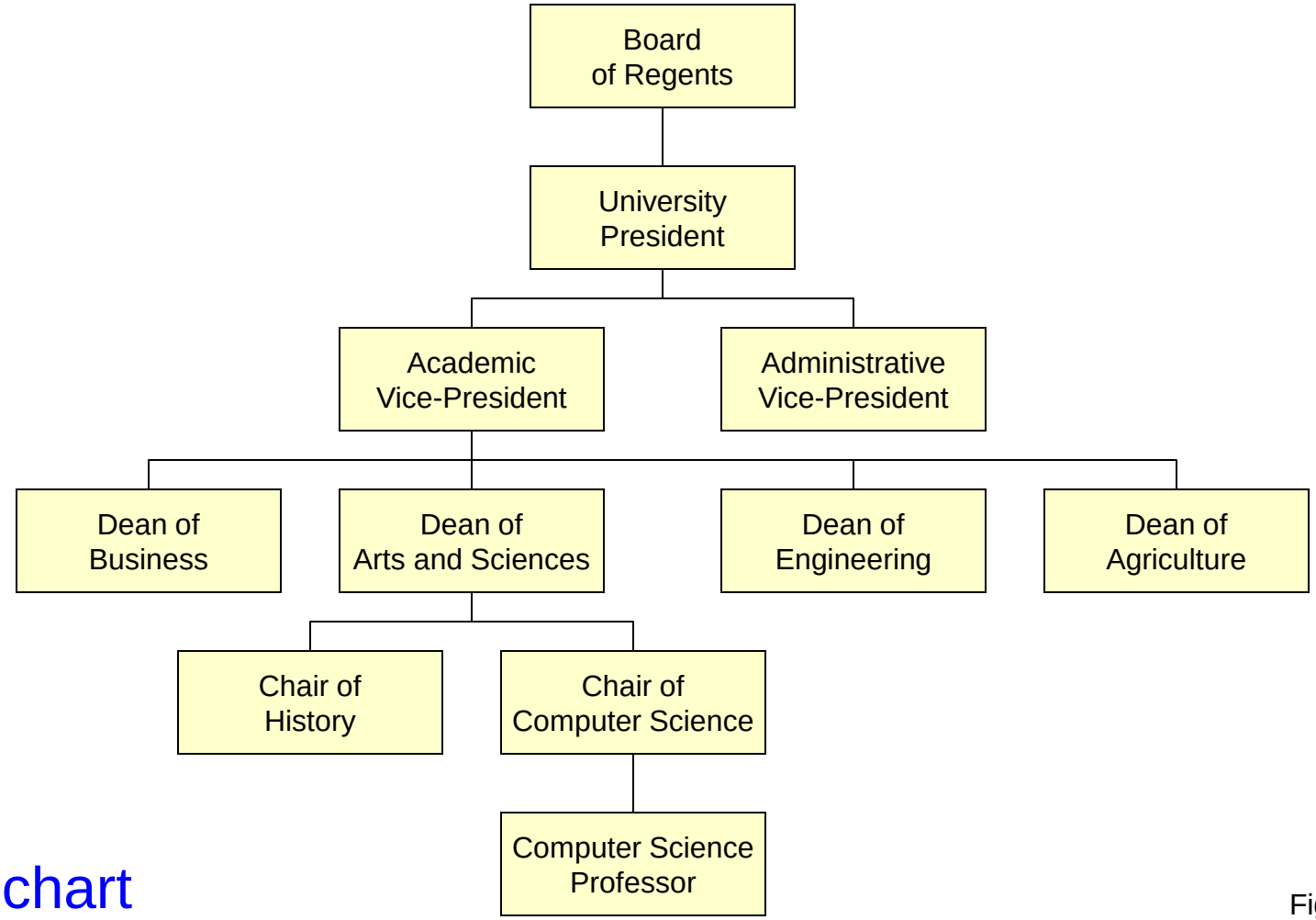
Depth (height) of tree: 3
Left child of node 3: none
Right child of node 3: node 5
Depth of node 5: 2
Depth of node 2: 1

Practice 18, Figure 6.36

Tree applications

- Trees in discrete mathematics
  - Decision trees (in counting), § 4.2
  - Decision trees (in algorithms), § 6.3
  - Binary tree data structure, § 6.1
  - Parse trees for formal languages, § 9.5
- Tree applications
  - Organizational chart
  - Computer file system
  - Help topics
  - Computer virus spread analysis
  - Algebraic expressions
- Many others …
- Data stored at tree nodes supports application

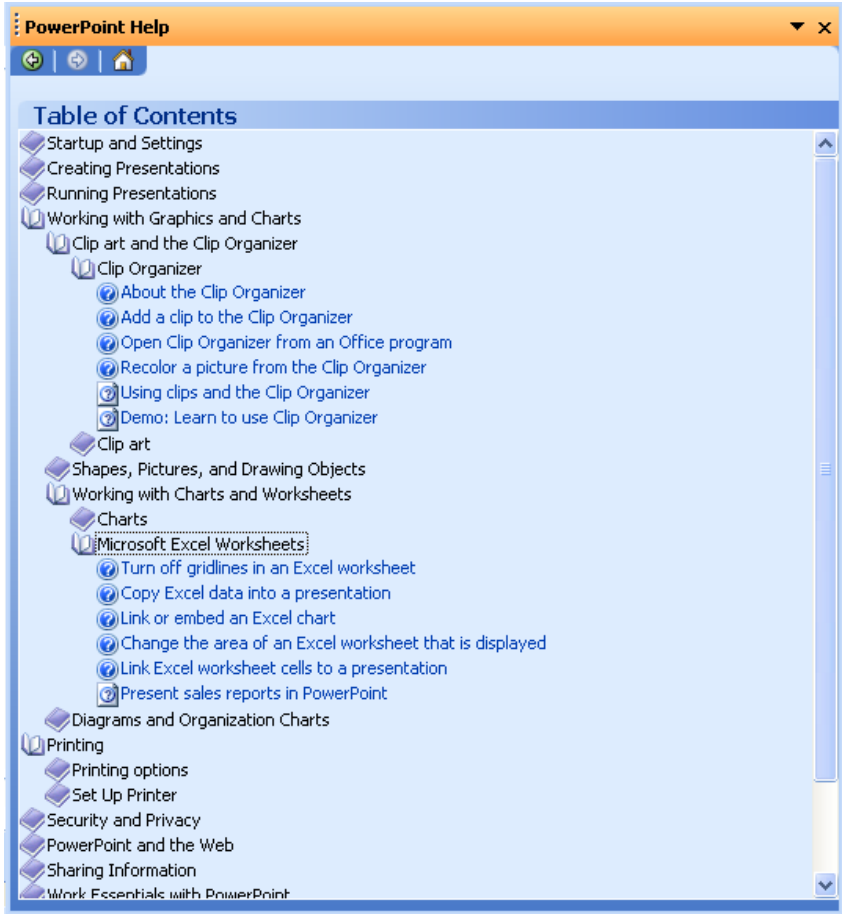
Tree application example



Org chart

Figure 6.37

Tree application example

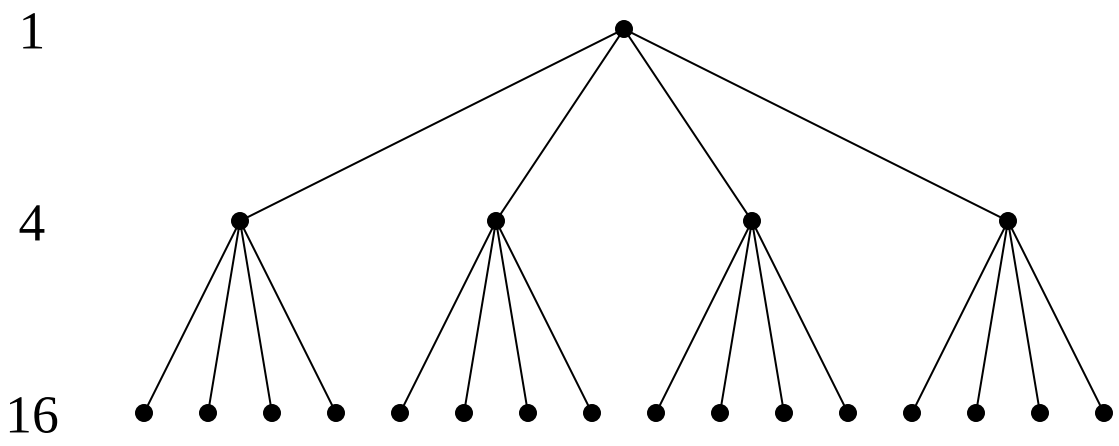


Help topics

Figure 6.38

Tree application example

A computer virus is spread via email.
Each second, 4 new machines are infected.
A 4-ary tree represents the spread of the virus.
By the multiplication principle,
4^n machines infected after n seconds.

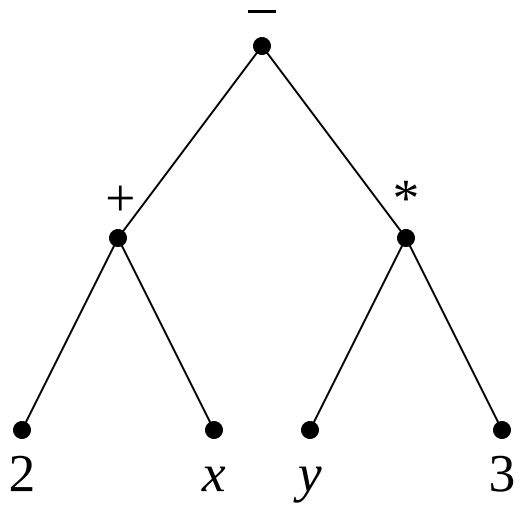


Computer virus spread analysis

Example 22, Figure 6.39

Tree application example

Algebraic expressions involving binary operations
can be represented with binary trees;
leaves labeled with operands, nonleaves with operators.
Each operation performed on expression
associated with its node's left and right subtrees.



$(2 + x) - (y * 3)$

Algebraic expressions

Example 23, Figure 6.40

Computer representation of binary trees

- Binary tree data structures
  - Similar to graph data structures
  - Want to represent left and right children
- Alternatives
  - Array; children stored in each row
  - Linked structure; children stored in each node

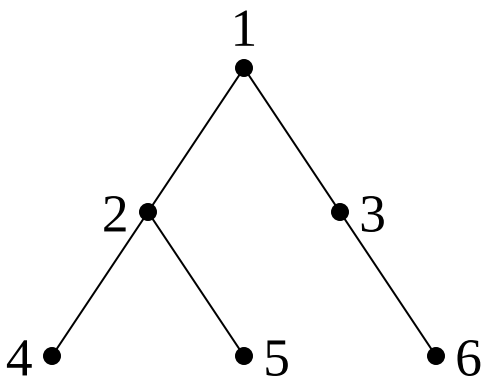
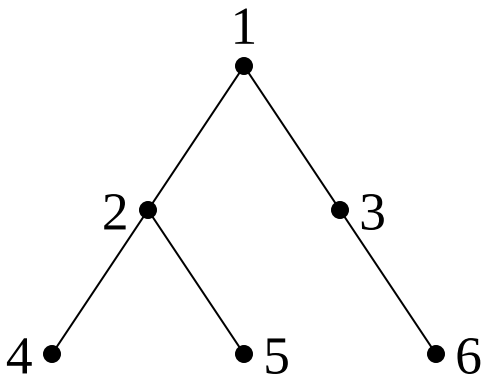


Figure 6.41

Binary tree represented as an array

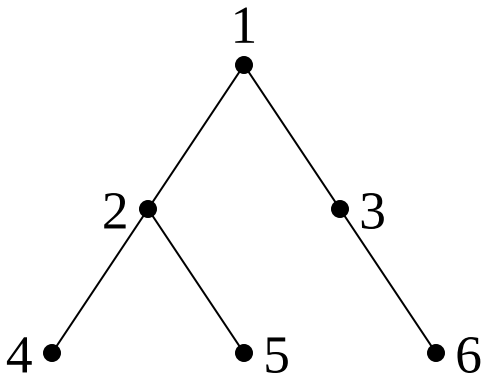


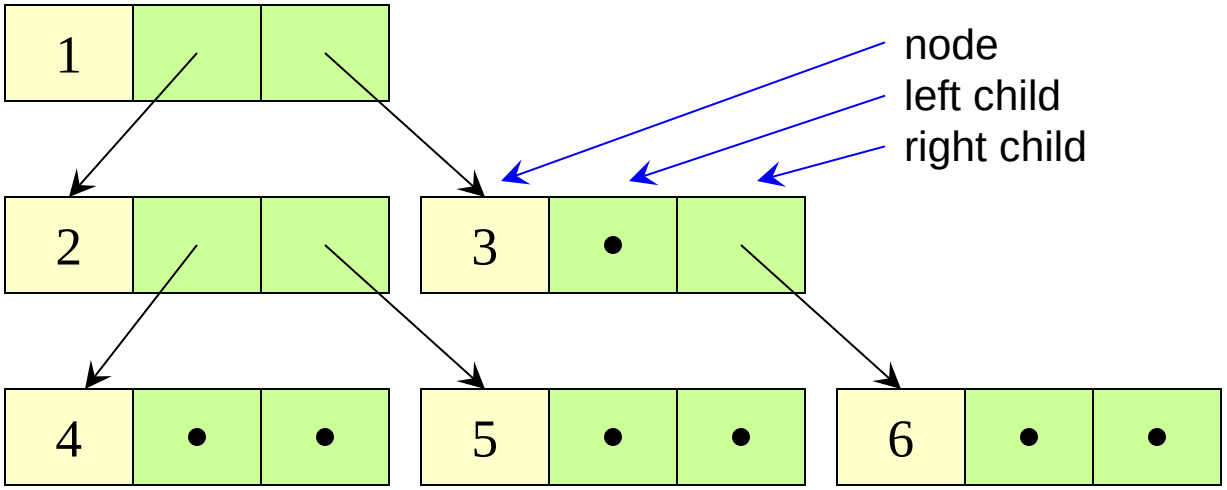
	Left child	Right child
1	2	3
2	4	5
3	0	6
4	0	0
5	0	0
6	0	0

0 indicates no child

Example 24, Figures 6.41, 6.42(a)

Binary tree represented as a linked structure





Example 24, Figures 6.41, 6.42(b)

Traversing trees

- Traverse; visit each node of a tree in some order
- Applications
  - List contents of tree
  - Back up tree to file
  - Calculate cumulative statistics (e.g., mean) on contents

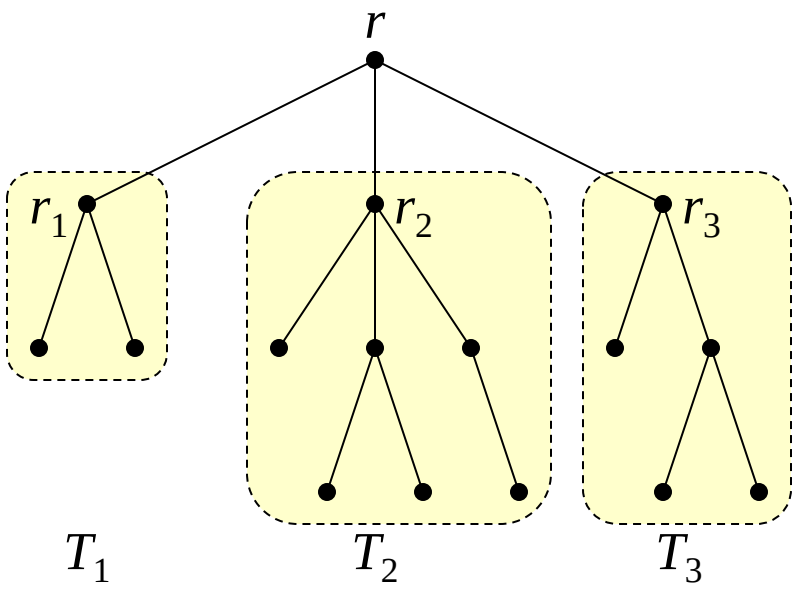


Figure 6.44

Binary tree traversal algorithms

Algorithm Preorder
Preorder(tree *T*)
// Writes the nodes of a binary tree with root *r* in preorder

```
    write(r)
    Preorder(left(r))    // left(r) is left subtree of the current node
    Preorder(right(r))   // right(r) is right subtree of the current node
```

end Preorder

Algorithm Inorder
Inorder(tree *T*)
// Writes the nodes of a binary tree with root *r* in inorder

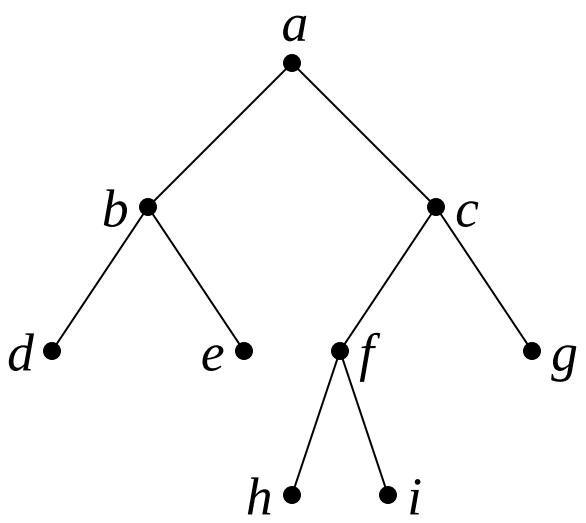
```
    Inorder(left(r))     // left(r) is left subtree of the current node
    write(r)
    Inorder(right(r))    // right(r) is right subtree of the current node
```

end Inorder

Algorithm Postorder
Postorder(tree T)
// Writes the nodes of a binary tree with root r in postorder

```
    Postorder(left(r))   // left(r) is left subtree of the current node
    Postorder(right(r)) // right(r) is right subtree of the current node
    write(r)
```

end Postorder



Preorder: $a, b, d, e, c, f, h, i, g$

Inorder: $d, b, e, a, h, f, i, c, g$

Postorder: $d, e, b, h, i, f, g, c, a$

Examples 25, 26, Figure 6.45

Non-binary tree traversal algorithms

Algorithm Preorder
Preorder(tree T)
// Writes the nodes of a tree with root r in preorder

```
    write(r)
    for i = 1 to t do
        Preorder(T_i)  // T_i is subtree i of the current tree
    end for

end Preorder
```

Algorithm Inorder
Inorder(tree T)
// Writes the nodes of a tree with root r in Inorder

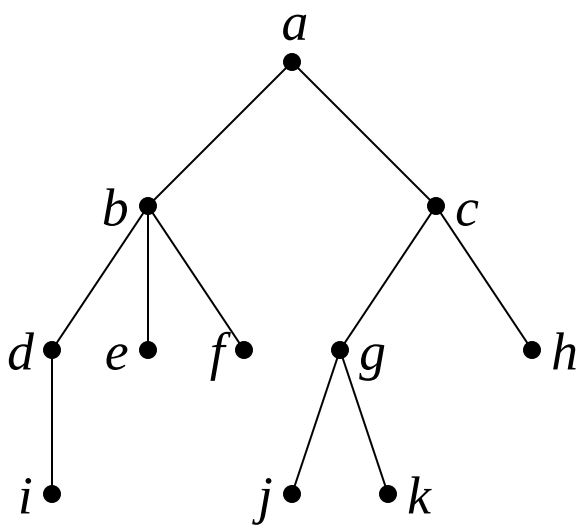
```
    Inorder(T_1)
    write(r)
    for i = 2 to t do
        Inorder(T_i)  // T_i is subtree i of the current tree
    end for

end Inorder
```

Algorithm Postorder
Postorder(tree T)
// Writes the nodes of a tree with root r in Postorder

```
    for i = 1 to t do
        Postorder(T_i)  // T_i is subtree i of the current tree
    end for
    write(r)

end Preorder
```



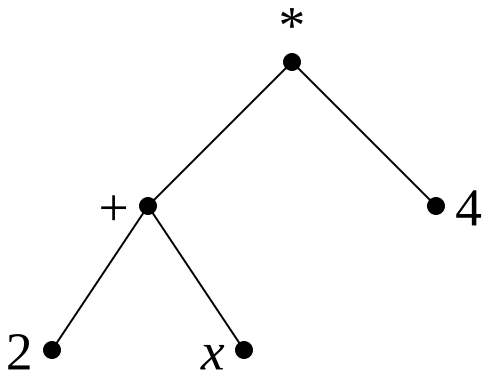
Preorder: $a, b, d, i, e, f, c, g, j, k, h$

Inorder: $i, d, b, e, f, a, j, g, k, c, h$

Postorder: $i, d, e, f, b, j, k, g, h, c, a$

Example 27, Figure 6.46

Algebraic expression tree example



Inorder:	$(2 + x) * 4$	aka infix (parentheses added for subtree)
Preorder:	$* + 2\ x\ 4$	aka prefix, Polish notation
Postorder:	$2\ x + 4\ *$	aka postfix, reverse Polish notation

Example 28, Figure 6.48

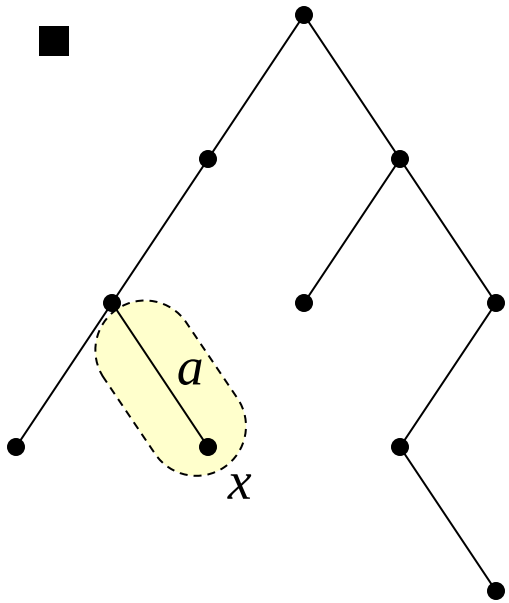
Tree proof example

Theorem
A tree with n nodes has $n - 1$ arcs.

Proof (by induction on number of nodes n)
Basis. For $n = 1$, the tree consists of 1 node and 0 arcs.

Inductive hypothesis. Assume tree with k nodes has $k - 1$ arcs.

Inductive step. Consider tree T with $k + 1$ nodes.
We seek to show T has k arcs.
Let x be a leaf of T; x has a unique parent.
Remove x and arc a connecting x to its parent.
The remaining graph T' is still a tree and has k nodes.
By the inductive hypothesis, T' has $k - 1$ arcs.
T has one more arc than T' (arc a),
so T has $(k - 1) + 1 = k$ arcs. ■



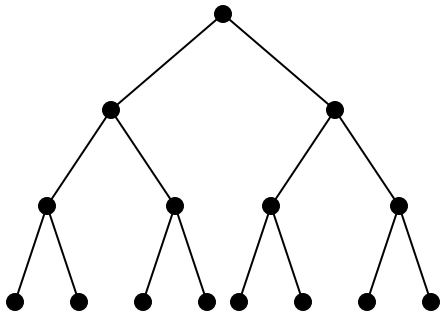
Example 29, Figure 6.50

Tree proof example

Theorem
A binary tree has at most 2^d nodes at depth d.

Proof (by induction on depth d)
Basis. For $d = 0$, there is 1 node (the root) and $2^0 = 1$.

Inductive hypothesis. Assume that
at depth k there are at most 2^k nodes.



Inductive step. Consider depth $k + 1$.
There are at most 2 children
at depth $k + 1$ for each node at depth k.
The maximum number of nodes at depth $k + 1$
is thus $2 \cdot 2^k = 2^{k+1}$. ■

Exercise 43

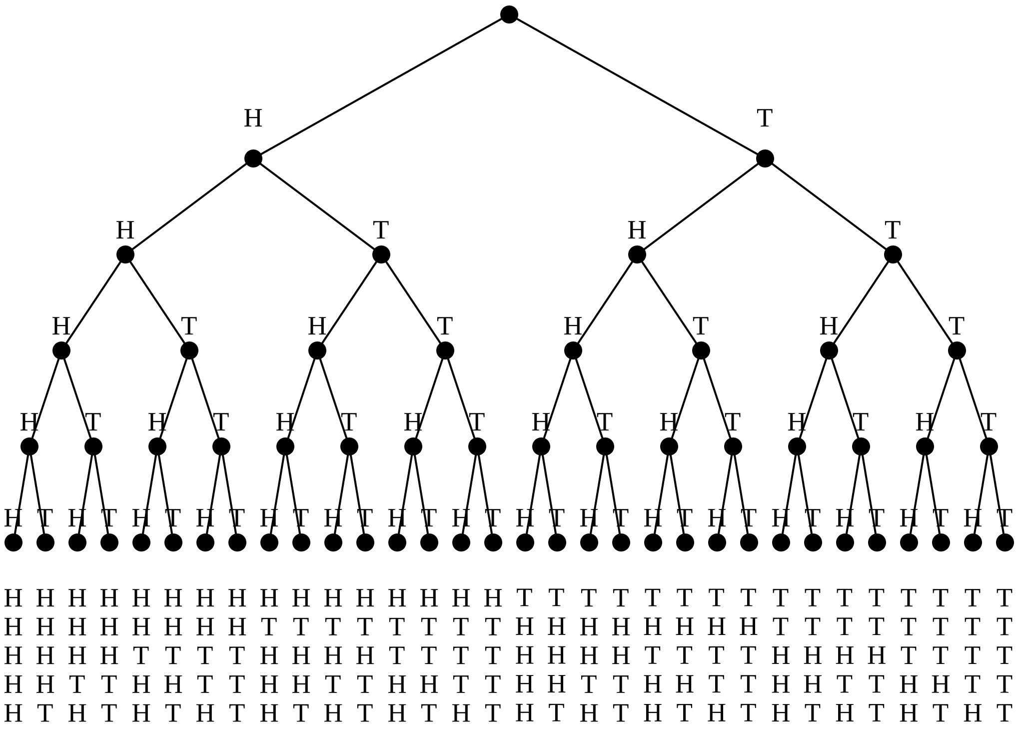
Section 6.2 homework assignment

See homework list for specific exercises.

6.3 Decision Trees

Example decision tree

- Internal nodes: coin toss (action or decision)
- Arcs: H or T (outcome of action)
- Leaves: 5 toss sequence (final outcome)



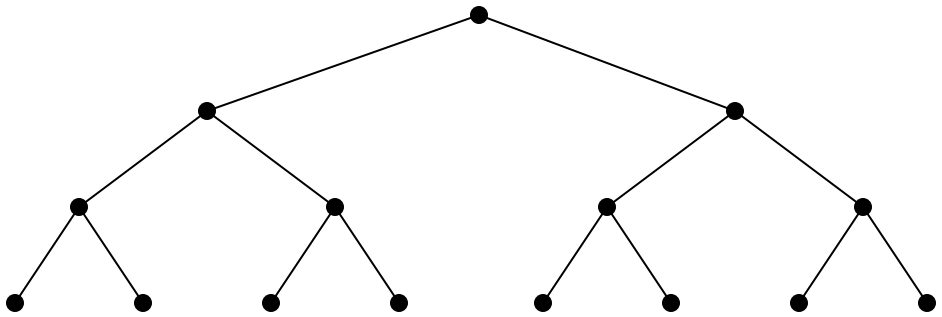
Chapter 3: Example 39, Practice 24

Decision tree definition

- A decision tree is a tree …
  - Internal nodes represent actions
  - Arcs represent the outcomes of actions
  - Leaves represent final outcomes
- Using decision trees to represent an algorithm
  - Internal nodes: actions/decisions algorithm takes
  - Arcs: decision outcomes or program paths
  - Leaves: state or condition on program termination
- A decision tree is not …
  - Data structure
  - Algorithm operating only on tree structures

Facts about binary trees

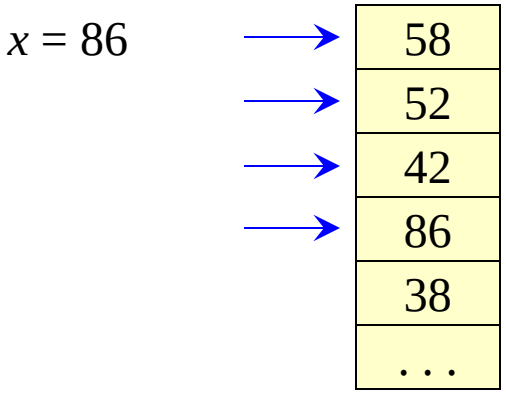
- Any binary tree
  - … of depth d has at most $2^{d+1} - 1$ nodes
  - … with m nodes has depth $\geq \lfloor \log m \rfloor$



Depth	Total nodes
$\lfloor \log 1 \rfloor = 0$	$2^{0+1} - 1 = 1$
$\lfloor \log 2 \rfloor = 1$	$2^{1+1} - 1 = 3$
$\lfloor \log 7 \rfloor = 2$	$2^{2+1} - 1 = 7$
$\lfloor \log 15 \rfloor = 3$	$2^{3+1} - 1 = 15$

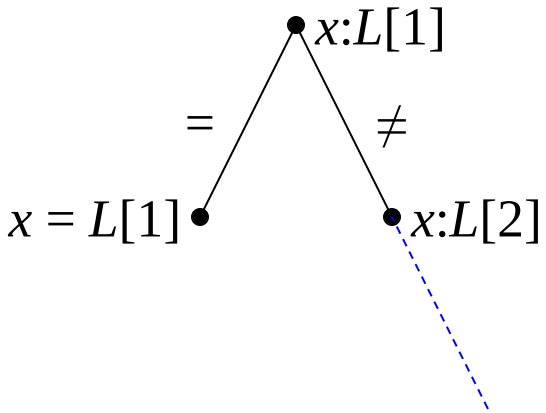
Search

- Find target x in list, or determine not present
- Different search algorithms available
  - Sequential search
  - Binary search
- Basic operation: comparison
  - Comparison notation: $x{:}L[i]$

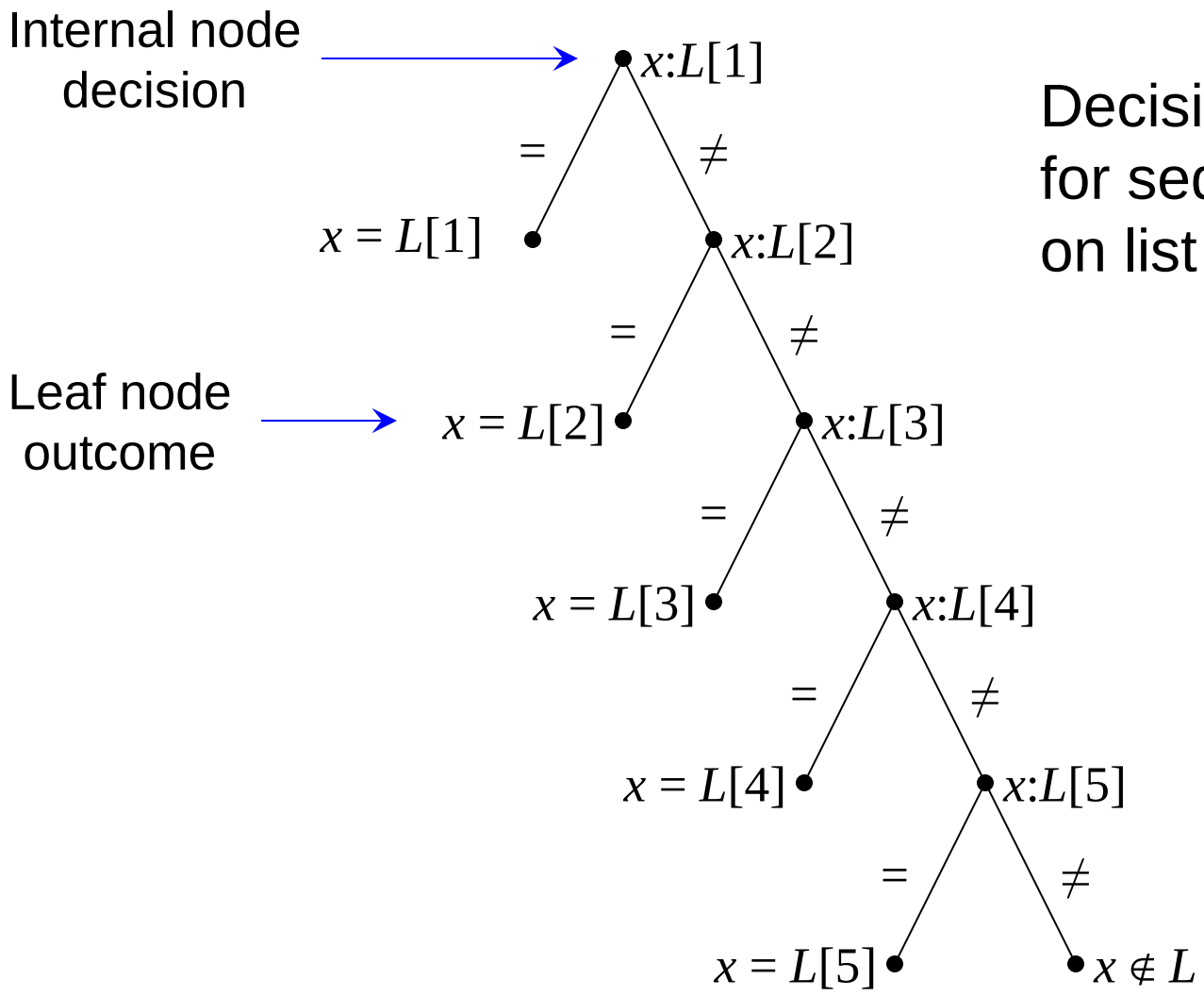


Decision tree for sequential search

- Compare x to each item in list
- Comparison outcomes
  - If $x = L[i]$ then x found, terminate
  - If $x \neq L[i]$ then x not found, continue at $L[i + 1]$



Partial decision tree
for sequential search

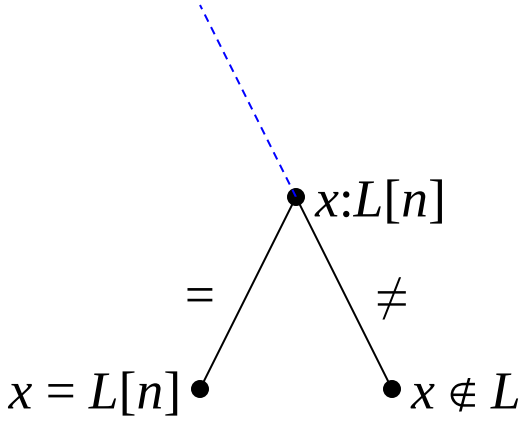


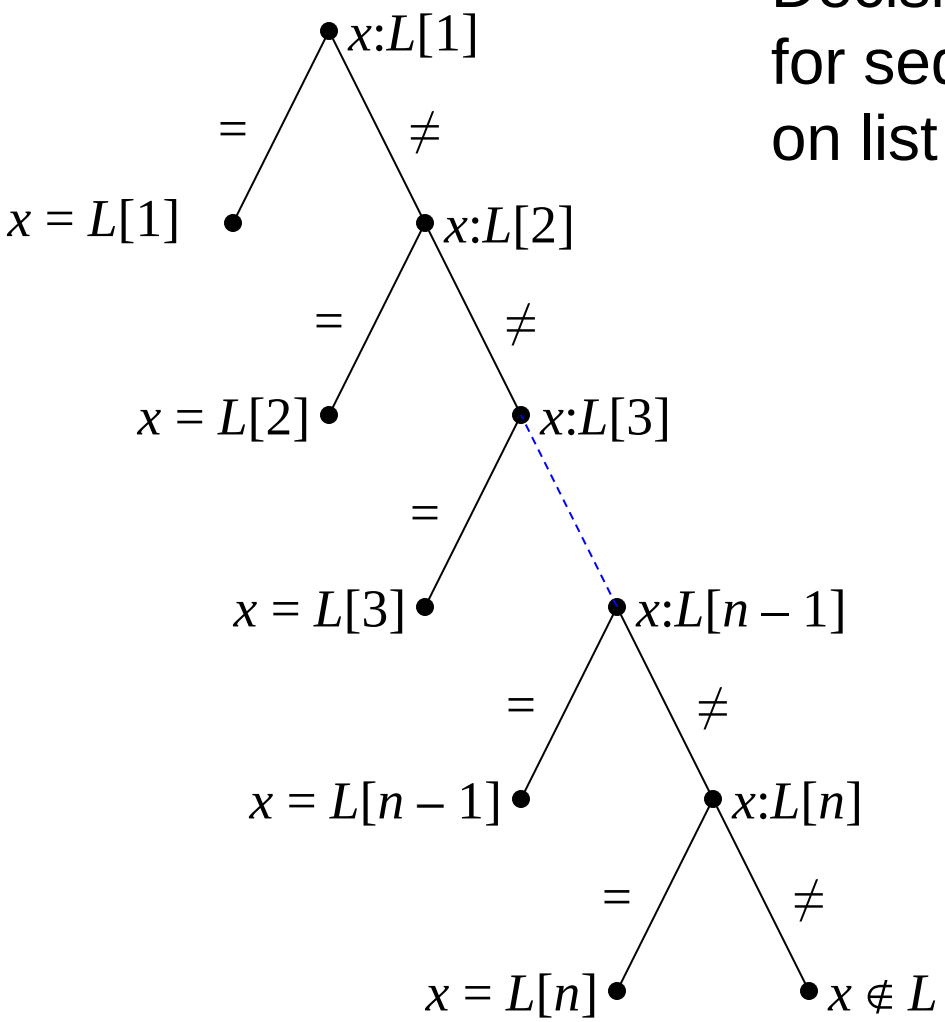
Decision tree for sequential search on list of 5 items.

Example 31, Figure 5.52

Algorithm analysis for sequential search

- Basis for analysis
  - Number of comparisons to reach final outcome
    = number of internal nodes from root to leaf
- Worst case analysis
  - Worst case (max comparisons)
    = max length path root to leaf, i.e., depth of tree
  - For sequential search, worst case is n comparisons

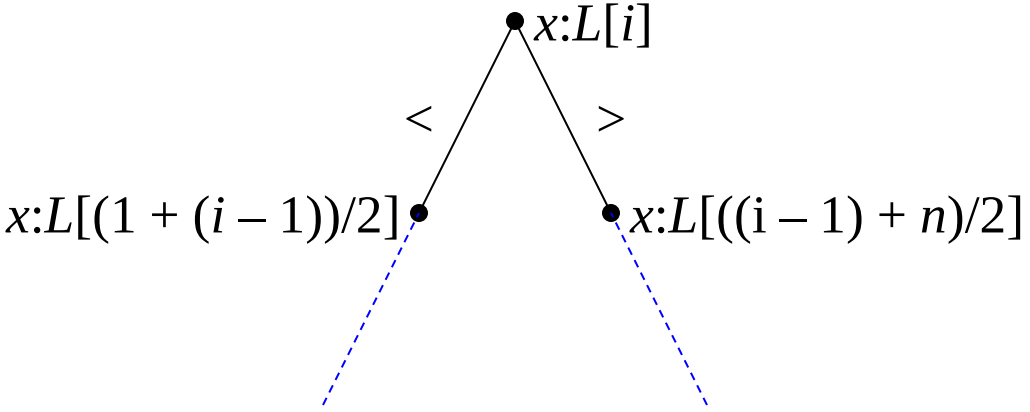




Decision tree
for sequential search
on list of n items.

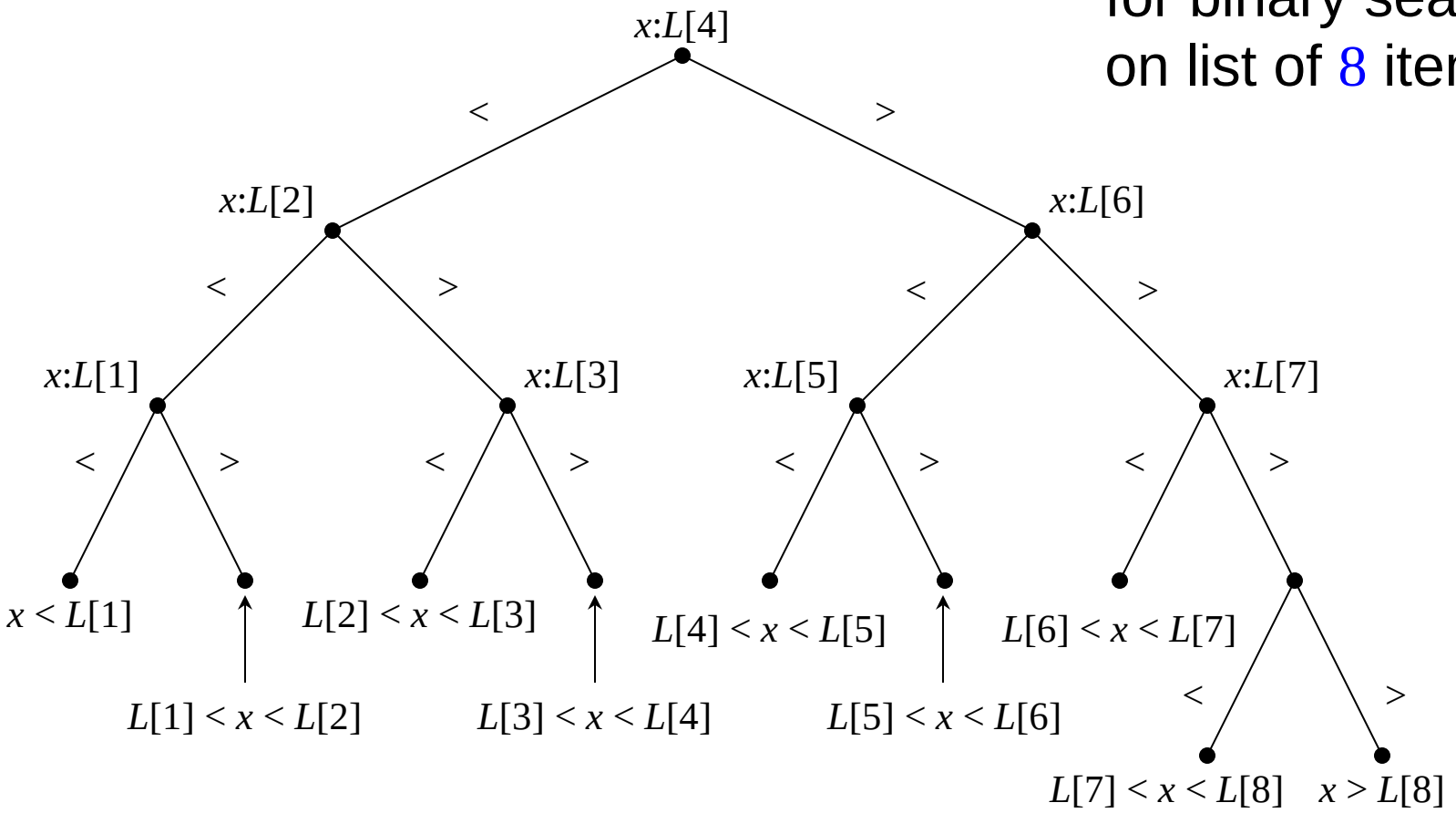
Decision tree for binary search

- Compare x to middle item, recurse on half of list
- Comparison outcomes
  - If $x = L[i]$ then x found, terminate
  - If $x < L[i]$ recurse on lower half
  - If $x > L[i]$ recurse on upper half



Partial decision tree
for binary search

Decision tree for binary search on list of 8 items

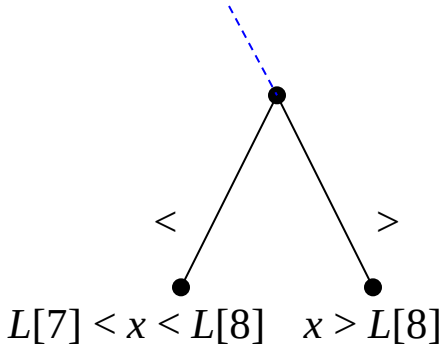


- Arcs for $x = L[i]$ "found" not shown
- Leaves are "not found" outcomes

Example 32

Algorithm analysis for binary search

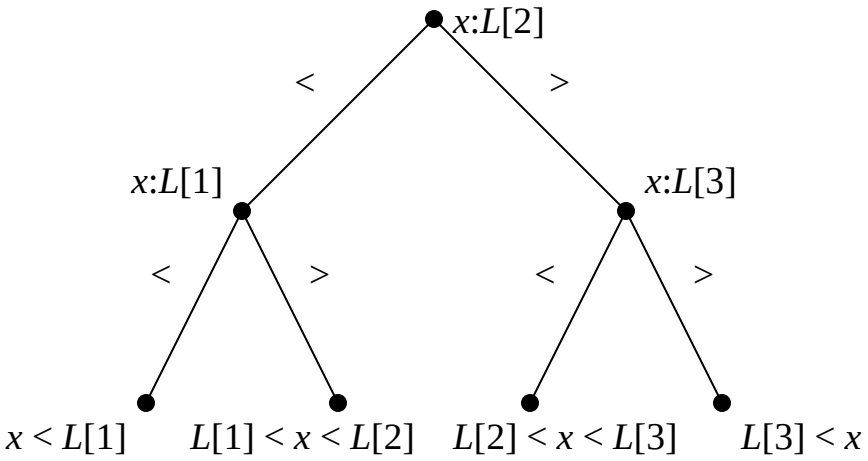
- Basis for analysis
  - Number of comparisons to reach final outcome
    = number of internal nodes from root to leaf
- Worst case analysis
  - Worst case (max comparisons)
    = max length path root to leaf, i.e., depth of tree
  - For binary search, worst case is 1 + log n comparisons
  - Formula and tree consistent, 1 + log 8 = 4



Lower bound for searching

- Lower bound; worst case for best algorithm
- Search algorithms
  - Specific algorithm unknown; basic op is comparison
  - Representable by decision tree
- Decision tree for search algorithm
  - Nodes are comparisons
  - For n item list, $\geq n$ comparisons, else possible miss
  - Number internal nodes $m \geq n$
  - Comparisons two children ($<$, $>$), hence binary tree

- Lower bound for searching n items
  - Decision tree T_1 for searching n items has m internal nodes, $m \geq n$
  - Create decision tree T_2 with m nodes by removing leaves from T_1
  - T_2 has depth $d \geq \lfloor \log m \rfloor \geq \lfloor \log n \rfloor$
  - T_1 has depth $d \geq \lfloor \log n \rfloor + 1$ because leaf removed
  - Conclusion: lower bound for search $\Theta(\log n)$

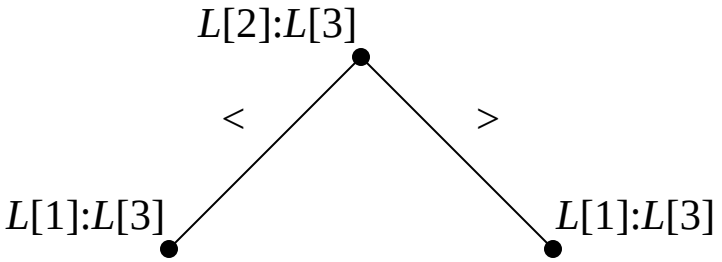


Sorting

- Put list of n items in order
- Different search algorithms available
  - Bubble sort
  - Heap sort
  - Quick sort
  - Insertion sort
  - …
- Basic operation: comparison
  - Comparison notation: $L[i]:L[j]$

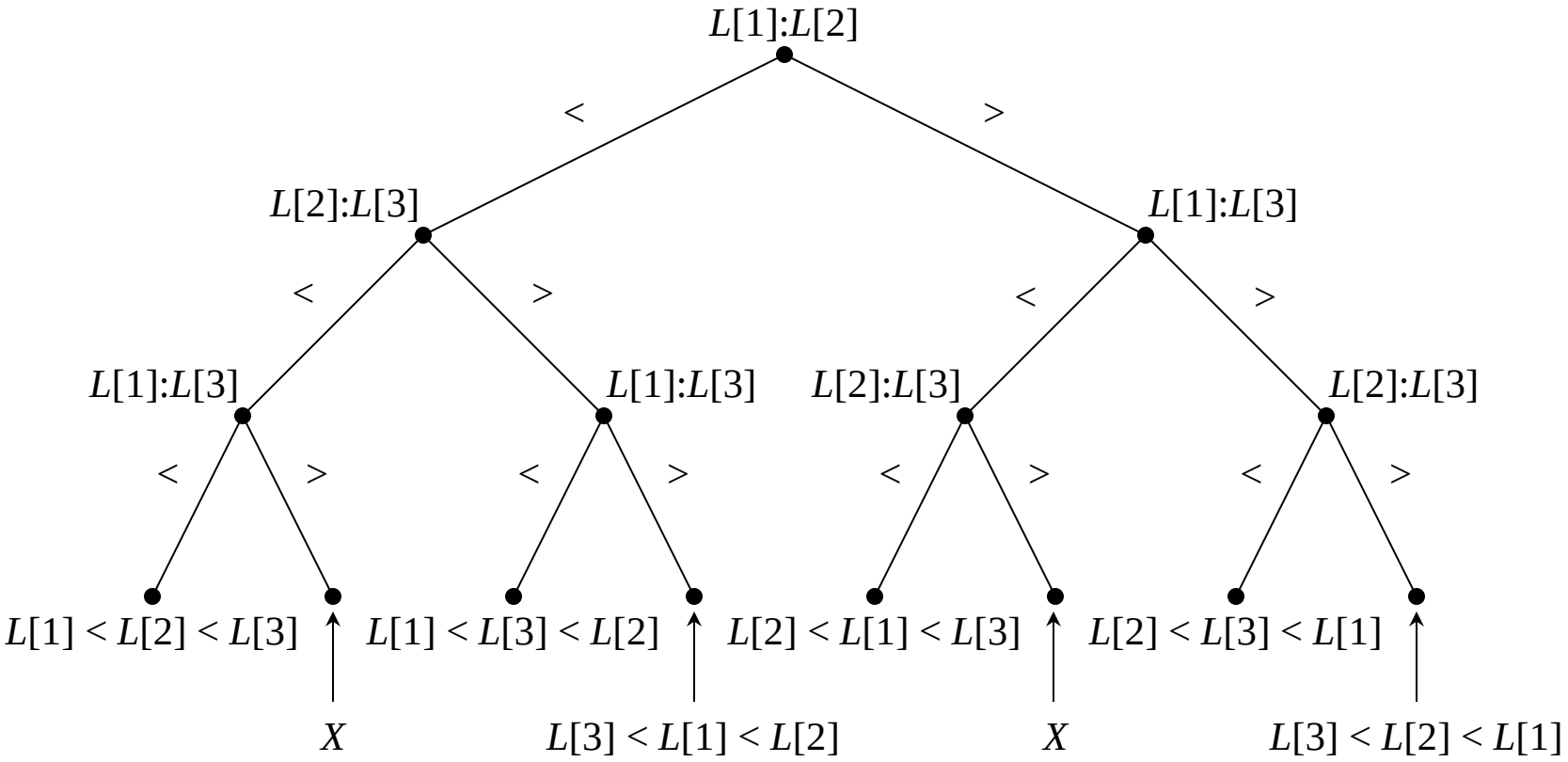
Decision tree for sorting

- Decision tree applies to all sort algorithms
  - Compare two items; next action depends on order
  - Simplifying assumption: no duplicate items
- Comparison outcomes
  - If $L[i] < L[j]$ proceed to left child in decision tree
  - If $L[i] > L[j]$ proceed to right child in decision tree



Partial decision tree
for sorting

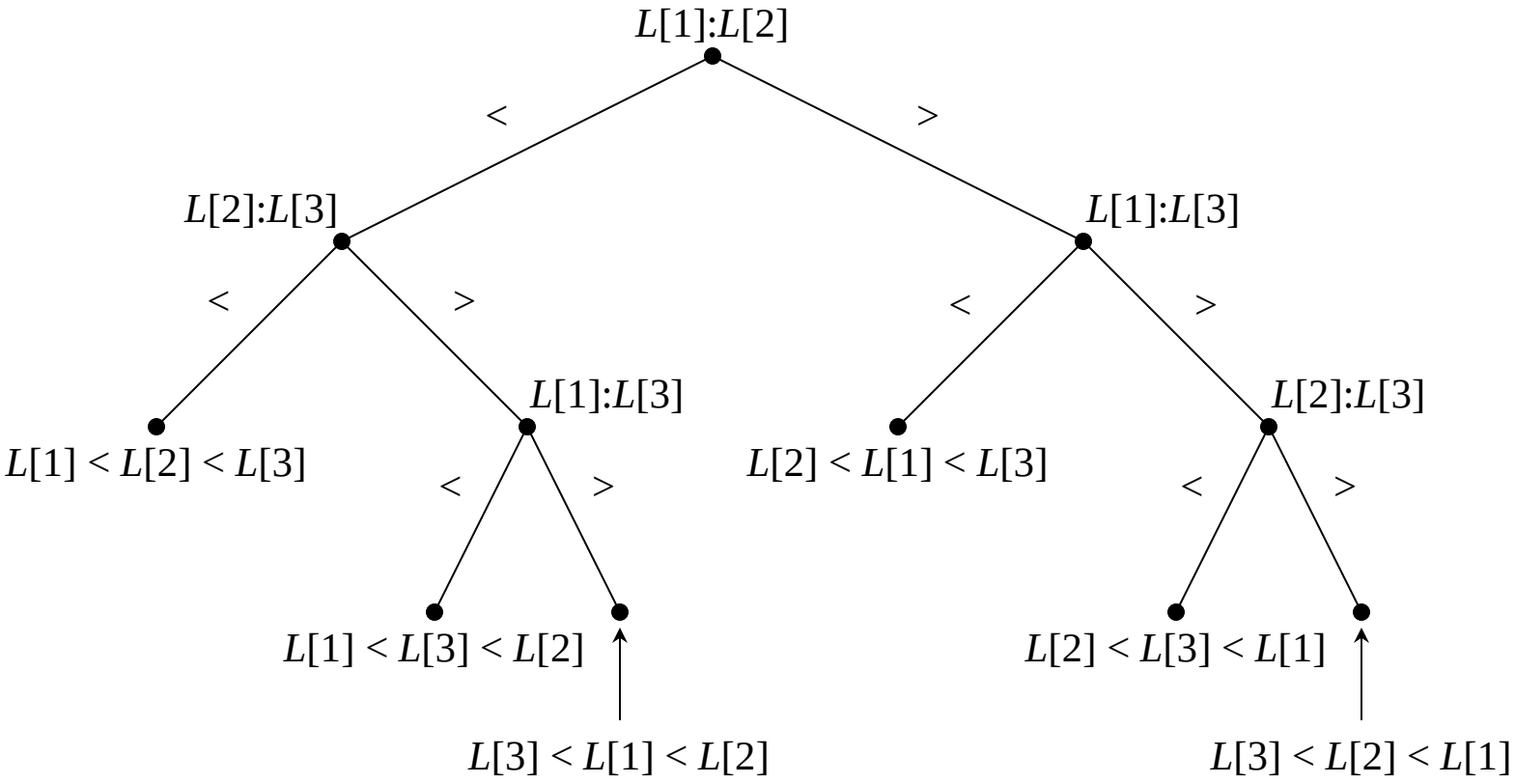
Decision tree for sorting algorithm on list of 3 items



- Internal nodes are comparisons
- Leaves are order outcomes; X outcomes not possible

Example 35

Decision tree for sorting algorithm on list of 3 items, revised



- Internal nodes are comparisons
- Leaves are order outcomes

Lower bound for sorting

- Lower bound; worst case for best algorithm
- Sort algorithms
  - Specific algorithm unknown; basic op is comparison
  - Representable by decision tree
- Decision tree for sort algorithm
  - Leaves are outcomes (sort orders)
  - For n item list, $n!$ different sort orders
  - Number of leaves $p \geq n$

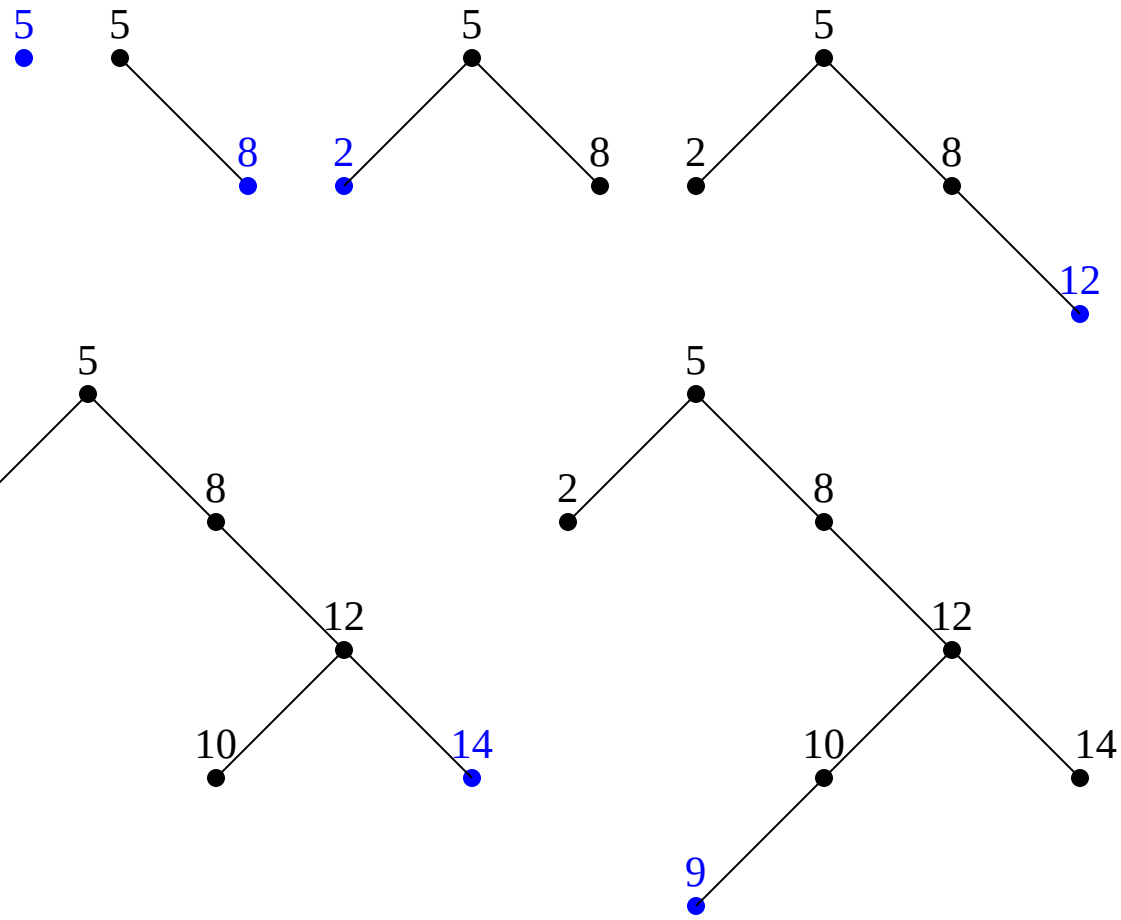
- Lower bound for sorting n items
  - Worst case is depth of decision tree
  - Binary tree with p leaves has depth d, $p \leq 2^d$
  - $p \leq 2^d$, $\log p \leq d$, $d = \lceil \log p \rceil$, $d \geq \lceil \log n! \rceil$
  - $\log n! = \Theta(n \log n)$
  - Conclusion: lower bound for sort $\Theta(n \log n)$

Binary search tree

- Data structure
- Constructing a binary search tree (BST)
  - Insert data items into BST in order given
  - First item is root
  - Subsequent items: starting at root, compare to nodes
    - If less, continue at left subtree
    - If more, continue at right subtree
    - If no subtree, insert item there
- BST can match decision tree to search itself

Example binary search tree

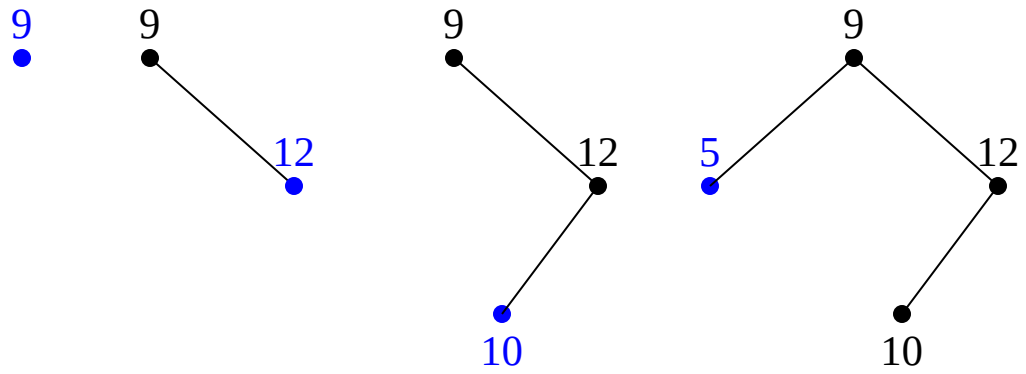
Data items to insert:
5, 8, 2, 12, 10, 14, 9

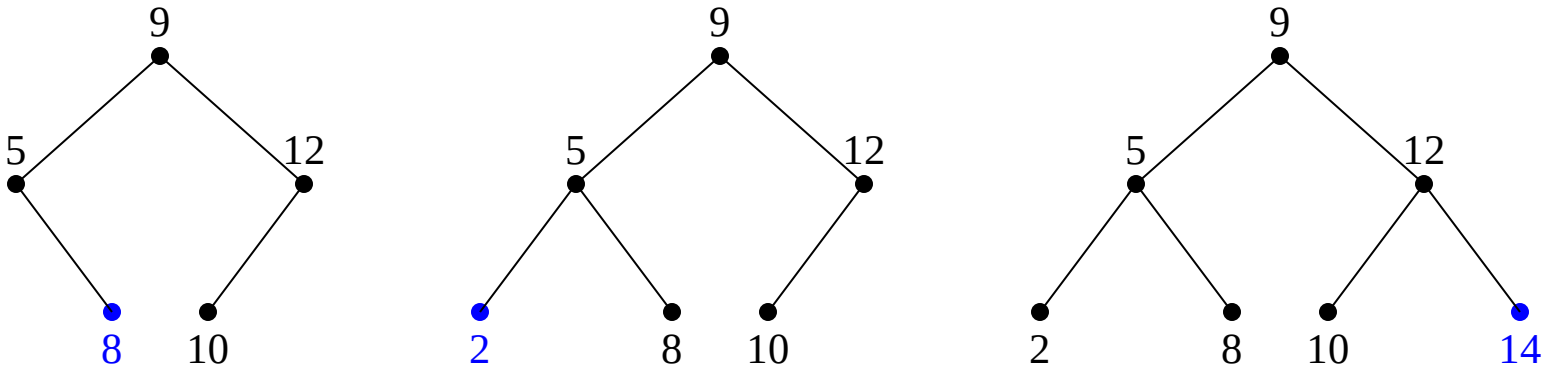


Example 33

Example binary search tree

Data items to insert:
9, 12, 10, 5, 8, 2, 14



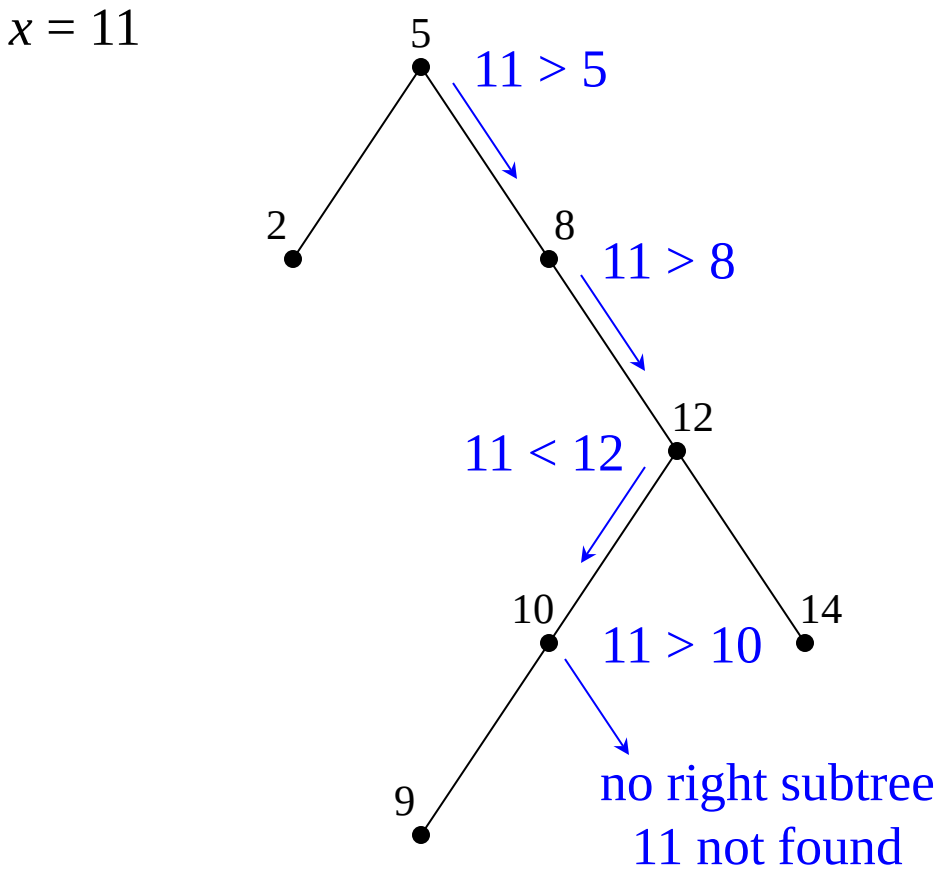


Example 34

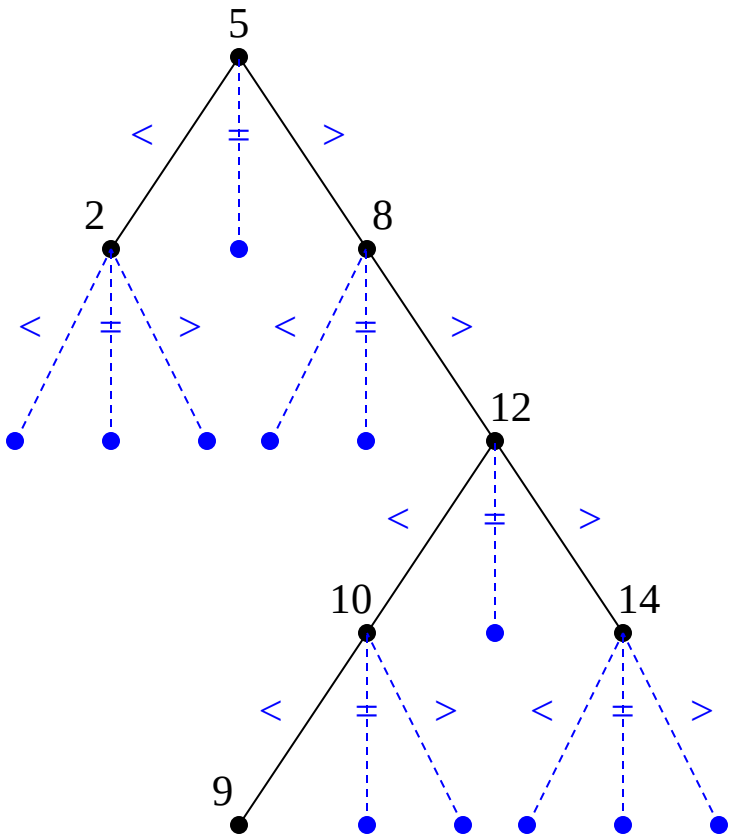
Binary tree search

- BST properties: value at each node …
  - … greater than all values in left subtree
  - … less than all values in right subtree
- Binary tree search
  - Compare target x to root
  - Comparison outcomes
    - x = root: x found, terminate
    - x < root: recurse on left subtree
    - x > root: recurse on right subtree
  - No subtree: x not in tree, terminate

Example binary tree search



Decision tree for binary tree search



Analysis of binary tree search

- Worst case is depth of tree (including leaves)
- Structure, hence depth, of BST varies with input
- For m nodes, minimum depth $d \geq \lfloor \log m \rfloor$
- Consistent with lower bound for search

Section 6.3 homework assignment

See homework list for specific exercises.

6.4 Huffman Codes

End

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE