Chapter 10
Trees

Topics

1. Introduction to Trees
2. Applications of Trees
3. Tree Traversal
4. Spanning Trees
5. Minimum Spanning Trees

INTRODUCTION TO TREES

Trees

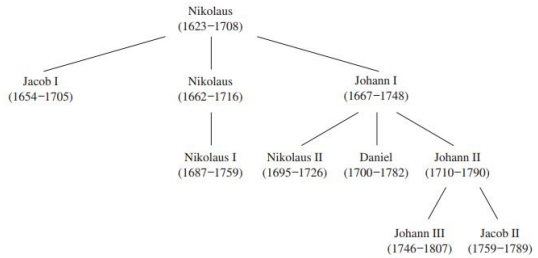


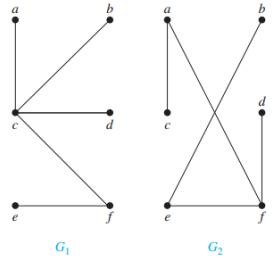
FIGURE 1 **The Bernoulli family of mathematicians.**

Trees

- **Definition.** A **tree** is a **connected undirected graph** with **no simple circuits**.
- **Properties.**
  - No loops.
  - No simple circuits (there is a unique simple path between any 2 of its vertices).
  - No multiple edges.
- **Theorem.** An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Trees

Example.

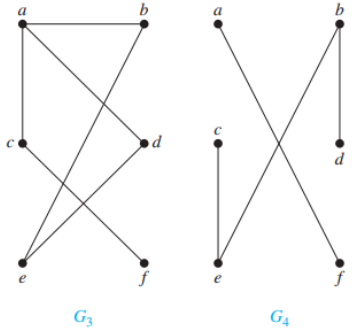


G_1 and G_2 are trees, because both are connected graphs with no simple circuits.

Trees

Example.

- G_3 is not a tree because e, b, a, d, e is a simple circuit in this graph.
- G_4 is not a tree because it is not connected.

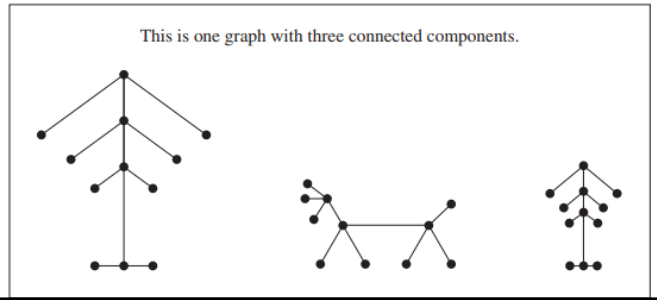


Forest

What about graphs containing no simple circuits that are not necessarily connected?
Each of their connected components is a tree.
These graphs are called **forests**.

Forest

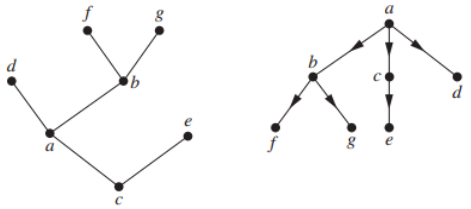
Example. Example of a forest



Rooted Trees

Definition. A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

Example.



Tree Terminologies

- Root: Vertex with in-degree 0

Example. Node a is the root

- Parent: Vertex u is a parent, such that there is directed edge from u to v

Example. b is parent of g and f

- Child: If u is parent of v, then v is child of u

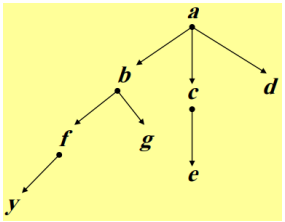
Example. g and f are children of b

- Siblings: Vertices with the same parents

Example. f and g

- Ancestors: Vertices in path from the root to vertex v, excluding v itself, including the root

Example. Ancestors of g are b and a



Tree Terminologies

- Descendents: All vertices that have v as ancestors

Example. Descendents of b are f, g, and y

- Leaf: Vertex with no children

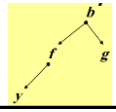
Example. y, g, e, d

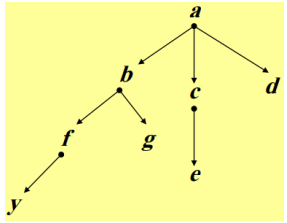
- Internal vertices: Vertices that have children

Example. a, b, c, f

- Subtree: Subgraphs consisting of v and its descendents and their incident edges

Example. Subtree rooted at b





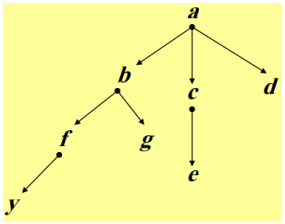
Tree Terminologies

- Level (of v): Length of unique path from root to v

Example. Level of root = 0, Level of b = 1, Level of g = 2

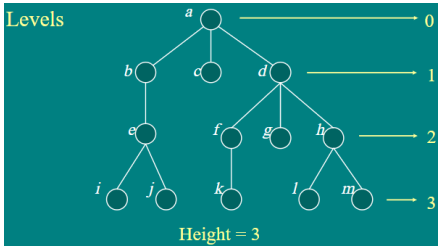
- Height: Maximum of vertices levels

Example. Height = 3



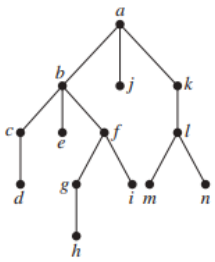
Tree Terminologies

Example.



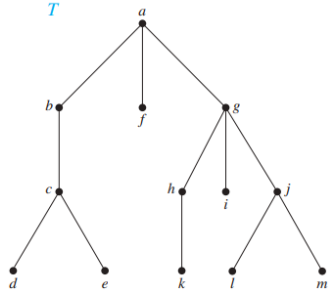
Tree Terminologies

Example. Find the level of each vertex in the rooted tree. What is the height of this tree?



Tree Terminologies

Example. In the rooted tree T (with root a), find the parent of c, the children of g, the siblings of h, all ancestors of e, all descendants of b, all internal vertices, and all leaves. What is the subtree rooted at g?



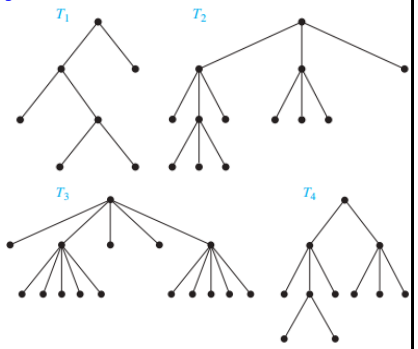
m-ary Tree

- **Definition.**
  - A rooted tree is called an m-ary tree if every internal vertex has no more than m children.
  - The tree is called a full m-ary tree if every internal vertex has exactly m children.
  - An m-ary tree with m = 2 is called a binary tree.

m-ary Trees

Example.

- T_1: a **full binary tree** because each of its internal vertices has 2 children
- T_2: a full 3-ary tree because each of its internal vertices has 3 children
- T_3: a full 5-ary tree because each internal vertex has 5 children
- T_4: not a full m-ary tree for any m because some of its internal vertices have 2 children and others have 3 children

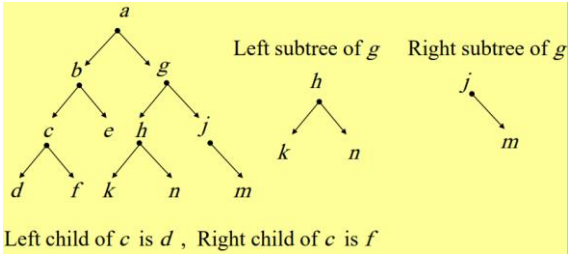


Ordered Rooted Trees

- An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.
- In an ordered binary tree (usually called just a binary tree):
  - If an internal vertex has two children, the first child is called the left child and the second child is called the right child.
  - The tree rooted at the left child of a vertex is called the left subtree of this vertex, and the tree rooted at the right child of a vertex is called the right subtree of the vertex.

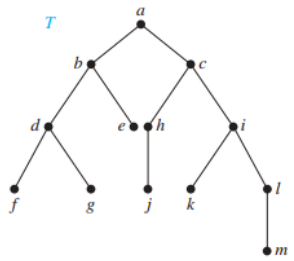
Ordered Rooted Trees

Example.



Ordered Rooted Trees

Example. What are the left and right children of d in the binary tree T (where the order is that implied by the drawing)? What are the left and right subtrees of c?

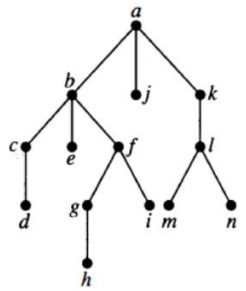


Properties of Trees

- A tree with n vertices has $n - 1$ edges.

Example.

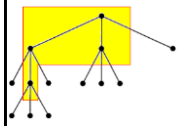
The tree in the figure has 14 vertices and 13 degrees



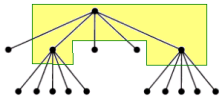
Properties of Trees

- For a full m-ary tree:
  1. Given n vertices: $i = \frac{n-1}{m}$ internal vertices and $l = \frac{(m-1)n+1}{m}$ leaves,
  2. Given i internal vertices: $n = mi + 1$ vertices and $l = (m-1)i + 1$ leaves,
  3. Given l leaves: $n = \frac{ml-1}{m-1}$ vertices and $i = \frac{l-1}{m-1}$ internal vertices

Example.



m=3
n=13
i=12/3= 4
l= [(3-1)13+1]/3=9



m=5
n=16
i= 15/5=3
l= (4.16+1)/5= 13

Properties of Trees

Example.

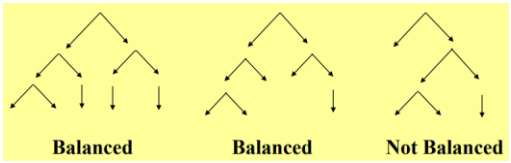
a. How many leaves are there in a full 5-ary tree with 56 nodes?

b. How many vertices are there in a full ternary (3-ary) tree with 27 leaves?

Balanced Trees

Definition. A rooted m-ary tree of height h is balanced if all leaves are at levels h or $h - 1$.

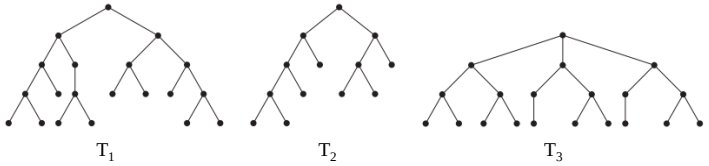
E.g.



h = 3 All leaves are at level 3	h = 3 All leaves are at levels 3 or 2	h = 3 All leaves are at levels 3 or 1

Balanced Trees

Example. Which of the rooted trees are balanced?



m-Ary Trees

Theorem.

- There are at most m^h leaves in an m-ary tree of height h.
- If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary tree is full and balanced, then $h = \lceil \log_m l \rceil$.

Example 1. Does there exist a full m-ary tree with height 4 and 100 leaves?

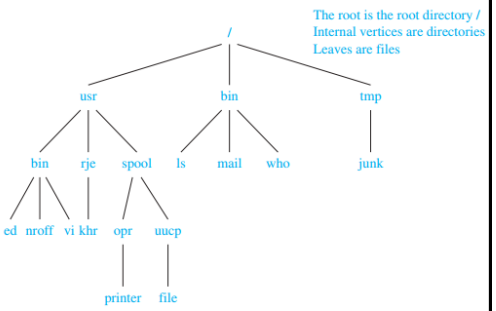
Example 2. What is the smallest height possible in a binary tree of 15 nodes? How many leaf nodes does it have?

Trees as Models

Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, and psychology.

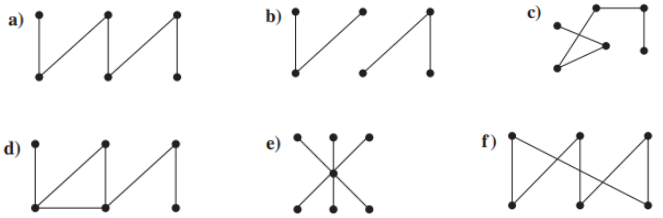
Trees as Models

Computer File Systems. Files in computer memory can be organized into directories. A directory can contain both files and subdirectories. The root directory contains the entire file system. Thus, a file system may be represented by a rooted tree, where the root represents the root directory, internal vertices represent subdirectories, and leaves represent ordinary files or empty directories.



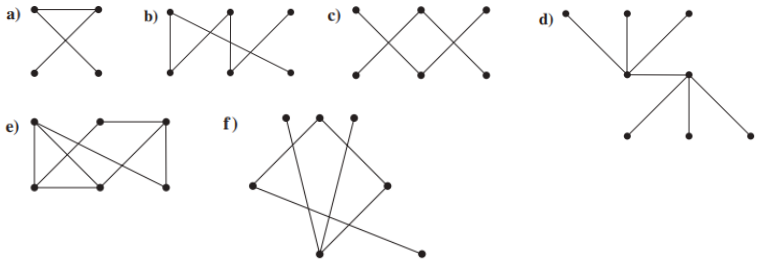
Exercises

1. Which of these graphs are trees?



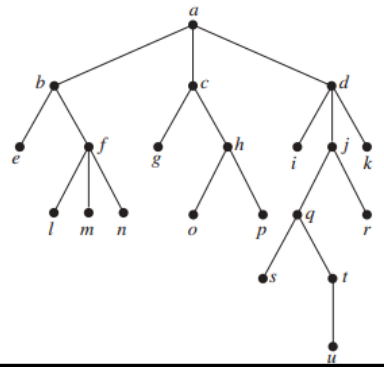
Exercises

2. Which of these graphs are trees?



Exercises

3. Given the rooted tree
a) Which vertex is the root?
b) Which vertices are internal?
c) Which vertices are leaves?
d) Which vertices are children of j?
e) Which vertex is the parent of h?
f) Which vertices are siblings of o?
g) Which vertices are ancestors of m?
h) Which vertices are descendants of b?



Exercises

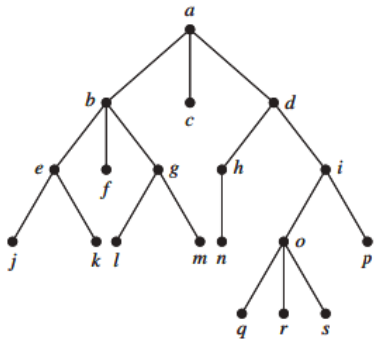
4. Given the rooted tree

a) Is the rooted tree a full m-ary tree for some positive integer m?

b) What is the level of each vertex of the rooted tree?

c) Draw the subtree of the tree that is rooted at

c_1) a c_2) c c_3) e



Exercises

5. How many edges does a tree with 10,000 vertices have?
6. How many vertices does a full 5-ary tree with 100 internal vertices have?
7. How many edges does a full binary tree with 1000 internal vertices have?
8. How many leaves does a full 3-ary tree with 100 vertices have?
9. A full m-ary tree T has 81 leaves and height 4.

a) Give the upper and lower bounds for m.

b) What is m if T is also balanced?

APPLICATIONS OF TREES

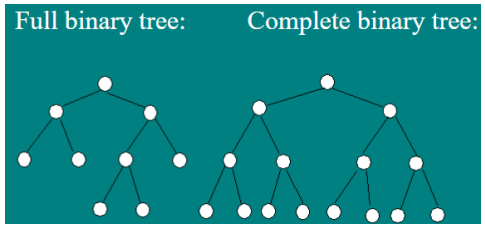
Applications of Trees

- Binary search trees
  - A simple data structure for sorted lists
- Decision trees
  - Minimum comparisons in sorting algorithms
- Prefix codes
  - Huffman coding
- Game trees (read book)

Review Full Binary Trees

- A **full binary tree** is a binary tree in which each node is either a leaf node or has degree 2 (i.e., has exactly 2 children).
- A complete binary tree is a full binary tree in which all leaves have the same depth.

Example.



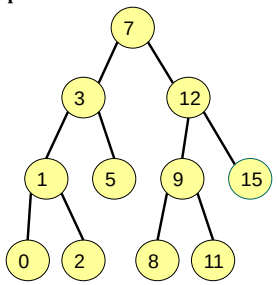
Binary Search Trees

- A representation for sorted sets of items.
  - Supports the following operations in $\Theta(\log n)$ average-case time:
    - Searching for an existing item.
    - Inserting a new item, if not already present.
  - Supports printing out all items in $\Theta(n)$ time.
- Note that inserting into a plain sequence a_i would instead take $\Theta(n)$ worst-case time.

Binary Search Trees

- Items are stored at individual tree nodes.
- We arrange for the tree to always obey this invariant:
  - For every item x,
    - Every node in x's left subtree is less than x.
    - Every node in x's right subtree is greater than x.

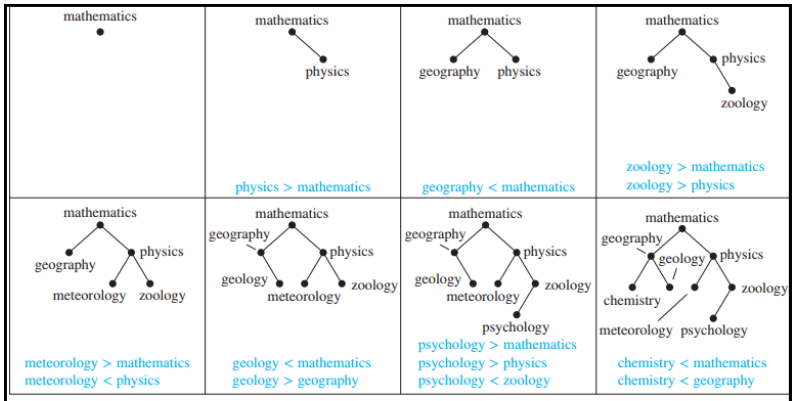
Example.



Binary Search Trees

Example. Form a binary search tree for the words mathematics, physics, geography, zoology, meteorology, geology, psychology, and chemistry (using alphabetical order).

$a < b < c \ldots; ab < ac < ae \ldots; abc < abd < abe \ldots$



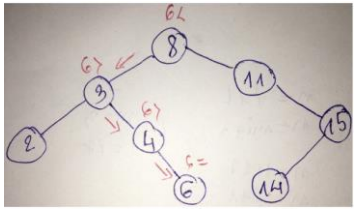
Constructing a binary search tree.

Binary Search Trees

Example.

a) Construct the Binary Search Trees for 8, 3, 11, 15, 2, 4, 14.

b) How many comparisons are required to locate number 6 in the binary search tree?



Binary Search Trees

Example. Construct the Binary Search Trees for

a) 23, 16, 43, 5, 9, 1, 6, 2, 33, 27 b) 60, 10, 70, 90, 5, 15, 2, 65, 3, 75, 40, 62.

Algorithm - Locating an Item in or Adding an Item to a Binary Search Tree

procedure *insertion*(*T*: binary search tree,
x: item)
v := root of *T*
{a vertex not present in *T* has the value *null* }
while *v* ≠ *null* and *label*(*v*) ≠ *x*
  if *x* < *label*(*v*) **then**
    if left child of *v* ≠ *null* **then** *v* := left child of *v*
    else add *new vertex* as a left child of *v* and set *v* := *null*
  else
    if right child of *v* ≠ *null* **then** *v* := right child of *v*
    else add *new vertex* as a right child of *v* and set *v* := *null*
if root of *T* = *null* **then** add a vertex *v* to the tree and label it with *x*
else if *v* is null or *label*(*v*) ≠ *x* **then** label *new vertex* with *x* and let *v* be this new vertex
return *v* {*v* = location of *x*}

Decision Trees

- A decision tree represents a *decision-making process*.
  - Each possible "decision point" or situation is represented by a node.
  - Each possible choice that could be made at that decision point is represented by an edge to a child node.
- In the extended decision trees used in *decision analysis*, we also include nodes that represent random events and their outcomes.

Decision Trees

Example. A decision tree that orders the elements of the list a, b, c.

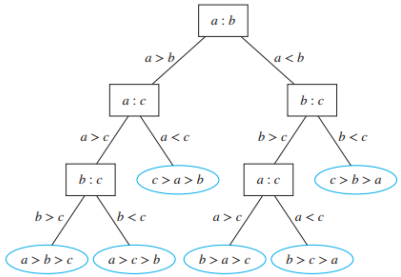


FIGURE 4 **A decision tree for sorting three distinct elements.**

Coin-Weighing Problem

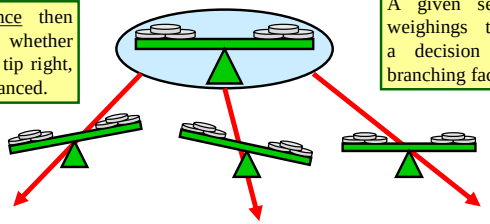
- Imagine you have 8 coins, one of which is a lighter counterfeit, and a free-beam balance.
  - No scale of weight markings is required for this problem!
- How many weighings are needed to guarantee that the counterfeit coin will be found?

Coin-Weighing Problem

- In each situation, we pick two disjoint and equal-size subsets of coins to put on the scale.

The balance then "decides" whether to tip left, tip right, or stay balanced.

A given sequence of weighings thus yields a decision tree with branching factor 3.



Coin-Weighing Problem

- The decision tree must have at least 8 leaf nodes, since there are 8 possible outcomes.
  - In terms of which coin is the counterfeit one.
- Recall the tree-height theorem, $h \geq \lceil \log_m l \rceil$.
  - Thus the decision tree must have height $h \geq \lceil \log_3 8 \rceil = \lceil 1.893\ldots \rceil = 2$.
- We can solve the problem with *at least* 2 weighings…

General Solution Strategy

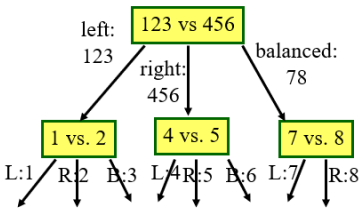
- The problem is an example of searching for 1 unique particular item, from among a list of n otherwise identical items.
  - Somewhat analogous to the adage of "searching for a needle in haystack."
- Armed with our balance, we can attack the problem using a divide-and-conquer strategy, like what's done in binary search.
  - We want to narrow down the set of possible locations where the desired item (coin) could be found down from n to just 1, in a logarithmic fashion.
- Each weighing has 3 possible outcomes.
  - Thus, we should use it to partition the search space into 3 pieces that are as close to equal-sized as possible.
- This strategy will lead to the minimum possible worst-case number of weighings required.

General Balance Strategy

- On each step, put $\lceil n/3 \rceil$ of the n coins to be searched on each side of the scale.
  - If the scale tips to the left, then:
    - The lightweight fake is in the right set of $\lceil n/3 \rceil \approx n/3$ coins.
  - If the scale tips to the right, then:
    - The lightweight fake is in the left set of $\lceil n/3 \rceil \approx n/3$ coins.
  - If the scale stays balanced, then:
    - The fake is in the remaining set of $n - 2\lceil n/3 \rceil \approx n/3$ coins that were not weighed!
- Except if $n \bmod 3 = 1$ then we can do a little better by weighing $\lfloor n/3 \rfloor$ of the coins on each side.

Coin Balancing Decision Tree

- Here's what the tree looks like in our case:



THE COMPLEXITY OF COMPARISON-BASED SORTING ALGORITHMS

Theorem. A sorting algorithm based on binary comparisons requires at least $\lceil \log_2 n! \rceil$ comparisons.

Corollary. The number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n \log n)$.

Theorem. The average number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n \log n)$.

Text Encoding

- Each English character is represented in the same number of bits (8 bits/1 byte) in ASCII.
  - ASCII uses fixed-length encoding
  - A text contains n characters, which take 8n bits in total to store the text in ASCII.
  - Is it possible to find a coding scheme of these letters such that fewer bits are used?

Text Encoding

Example. In ASCII

H	e	l	l	o	o	o	!
01001000	01100101	01101100	01101100	01101111	01101111	01101111	00100001

→ The total number of bits is 64 bits

The string Hellooo! Includes 5 characters (h, e, l, o, !) so in fact we can encode each character with 3 bits. Hence, The total number of bits used to encode data is $3 \times 8 = 24$ bits

H	000	0
e	001	1
l	010	2
o	011	3
!	100	4

H	e	l	l	o	o	o	!
000	001	010	010	011	011	011	100

Text Encoding

- In real-life English texts, characters do not appear with the same frequency.
- Using bit strings of different lengths to encode letters – variable-length encoding.
  - Frequent characters are encoded in fewer bits.
  - In-frequent characters are encoded in more bits.
- Then, we can reduce the total storage.

Text Encoding

Example.

H	110
e	1111
l	10
o	0
!	1110

H	e	l	l	o	o	o	!
110	1111	10	10	0	0	0	1110

The total number of bits used to encode is 18 bit.

Prefix Codes

- If the encoding code book becomes:
  - A → 0, B → 1, C → 00, D → 01, E → 101.
  - Suppose the encoded text is: 0101
  - The original texts can have several possibilities such as ABAB or ABD or DD …

→ Trouble when decoding the encoded message

→ Prefix codes

Prefix Codes

- Prefix code encoding scheme is used to resolve that each codeword is a prefix of another.
- For a text encoded by a prefix code, we can easily decode it in the following way:
  - Scan from left to right to extract the first code
  - Recursively decode the remaining part.
- A prefix code can be represented using a binary tree.

Example.

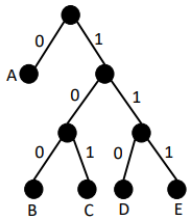
{a = 0, b = 110, c = 10, d = 111} is a prefix code

{a = 00, b = 01, c = 10, d = 11} is a prefix code

{a = 0, b = 01, c = 10, d = 101, e = 110} is not a prefix code

Prefix Code Tree

- A prefix code tree is a rooted tree such that:
  - each edge is labeled by a bit (the left edge at each internal vertex is labeled by 0 and the right edge by a 1)
  - each leaf denoted by a character.
  - The codeword for the character is based on the labels on root-to-leaf path.
- The tree representing a code can be used to decode a bit string.



Example. A → 0, B → 100, C → 101, D → 110, E → 111

The word encoded by 1000100111 using the prefix code is

BABE

Prefix Codes

Example. Draw the tree represents the encoding of e by 0, a by 10, t by 110, n by 1110, and s by 1111. Encode the string sant? Decode the string of bits 11111011100?

Huffman Coding Algorithm

We now introduce an algorithm that takes as input the frequencies (which are the probabilities of occurrences) of symbols in a string and produces as output a prefix code that encodes the string using the fewest possible bits, among all possible binary prefix codes for these symbols. This algorithm is known as **Huffman coding**.

Huffman Coding

We now introduce an algorithm that takes as input the frequencies (which are the probabilities of occurrences) of symbols in a string and produces as output a prefix code that encodes the string using the fewest possible bits, among all possible binary prefix codes for these symbols. This algorithm is known as **Huffman coding**.

Huffman Coding Algorithm

procedure *Huffman*(C: symbols a_i with frequencies w_i, $i = 1, \ldots, n$)
F := forest of n rooted trees, each consisting of the single vertex a_i and assigned weight w_i
while F is not a tree
  Replace the rooted trees T and T' of least weights from F with $w(T) \geq w(T')$ with a tree having a new root that has T as its left subtree and T' as its right subtree. Label the new edge to T with 0 and the new edge to T' with 1.
  Assign $w(T) + w(T')$ as the weight of the new tree.
{the Huffman coding for the symbol a_i is the concatenation of the labels of the edges in the unique path from the root to the vertex a_i}

Huffman Coding Tree

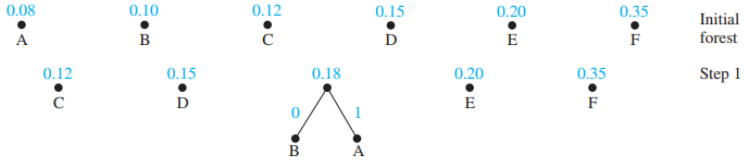
Steps of the algorithm (buildHuff):

1. Prepare a collection of n initial Huffman trees, each of which is a single leaf node. Put the n trees onto a priority queue organized by weight (frequency).
2. Remove the first two trees (the ones with lowest weight). Join these two trees to create a new tree whose root has the two trees as children, and whose weight is the sum of the weights of the two children trees.
3. Put this new tree into the priority queue.
4. Repeat steps 2-3 until all of the partial Huffman trees have been combined into one.

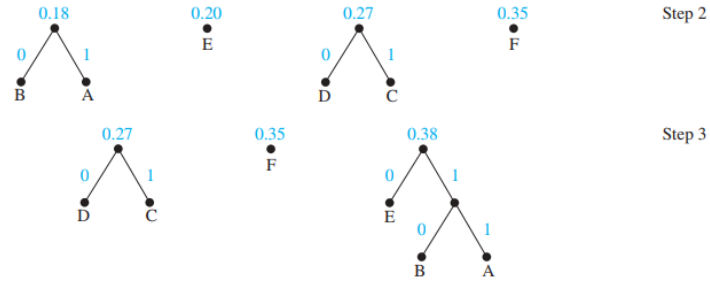
Huffman Coding

Example. Use Huffman coding to encode the following symbols with the frequencies listed: A: 0.08, B: 0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35. What is the average number of bits used to encode a character?

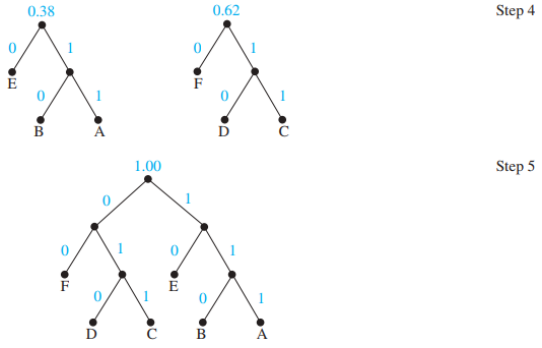
Solution.



Huffman Coding



Huffman Coding



Huffman Coding

The encoding produced encodes A by 111, B by 110, C by 011, D by 010, E by 10, and F by 00.

The average number of bits used to encode a symbol using this encoding is

$$3 \cdot 0.08 + 3 \cdot 0.1 + 3 \cdot 0.12 + 3 \cdot 0.15 + 2 \cdot 0.2 + 2 \cdot 0.35 = 2.45$$

- **Note.**
  - The average bits per letter of a prefix code is the sum over all symbols of its frequency times the number of bits of its encoding.

Huffman Coding

Example 1. Find prefix code for each symbol in following text by using Huffman Coding: ABCCDEBABFFBACBEBDFAAAABCDEEDCCBFEBFCAE

Example 2. Construct a Huffman coding to encode the message "good morning". How many bits are used, and what is the average number of bits used for each letter?

Exercises

1. Build a binary search tree for the words banana, peach, apple, pear, coconut, mango, and papaya using alphabetical order.

2. Build a binary search tree for the words oenology, phrenology, campanology, ornithology, ichthyology, limnology, alchemy, and astrology using alphabetical order.

3. How many comparisons are needed to locate or to add each of these words in the search tree for Exercise 1, starting fresh each time?

a) pear b) banana c) kumquat d) orange

4. How many comparisons are needed to locate or to add each of the words in the search tree for Exercise 2, starting fresh each time?

a) palmistry b) etymology c) paleontology d) glaciology

Exercises

5. Which of these codes are prefix codes?

a) a: 11, e: 00, t: 10, s: 01

b) a: 0, e: 1, t: 01, s: 001

c) a: 101, e: 11, t: 001, s: 011, n: 010

d) a: 010, e: 11, t: 011, s: 1011, n: 1001, i: 10101

6. Construct the binary tree with prefix codes representing these coding schemes.

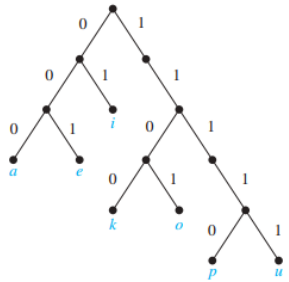
a) a: 11, e: 0, t: 101, s: 100

b) a: 1, e: 01, t: 001, s: 0001, n: 00001

c) a: 1010, e: 0, t: 11, s: 1011, n: 1001, i: 10001

Exercises

7. What are the codes for a, e, i, k, o, p, and u if the coding scheme is represented by this tree?



Exercises

8. Given the coding scheme a: 001, b: 0001, e: 1, r: 0000, s: 0100, t: 011, x: 01010, find the word represented by

a) 01110100011. b) 0001110000. c) 0100101010. d) 01100101010.

9. Use Huffman coding to encode these symbols with given frequencies: a: 0.20, b: 0.10, c: 0.15, d: 0.25, e: 0.30. What is the average number of bits required to encode a character?

10. Use Huffman coding to encode these symbols with given frequencies: A: 0.10, B: 0.25, C: 0.05, D: 0.15, E: 0.30, F: 0.07, G: 0.08. What is the average number of bits required to encode a symbol?

11. Construct two different Huffman codes for these symbols and frequencies: t: 0.2, u: 0.3, v: 0.2, w: 0.3

Exercises

12. Use Huffman coding algorithm to encode the word "football". What is the average number of bits required to encode a character?

A. 2.35 B. 2.5 C. 2.45 D. 2.25

13. Which codes are prefix codes?

i) a: 1010, b: 010, c: 1101, d: 100

ii) a: 10, b: 0101, c: 1110, d: 1001

A. I B. ii C. Both i and ii D. None of them

TREE TRAVERSAL

- Universal Address Systems
- Traversal Algorithms
  - Preorder
  - Inorder
  - Postorder
- Infix, prefix and postfix notation

Tree Traversals

- Ordered rooted trees are often used to store information. → Need procedures for visiting each vertex of an ordered rooted tree to access data.
- Tree traversal is a procedure for visiting all the vertices of a tree.
- Some common traversal algorithms:
  - Preorder
  - Inorder
  - Postorder

Universal Address Systems

To totally order the vertices of an ordered rooted tree:

1. Label the root with the integer 0.
2. Label its k children (at level 1) from left to right with 1, 2, 3, … , k.
3. For each vertex v at level n with label A, label its k_v children, as they are drawn from left to right, with A.1, A.2, … , A.k_v.

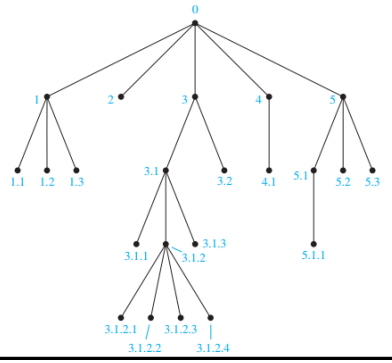
This labeling is called the **universal address system** of the ordered rooted tree.

Universal Address Systems

Example. The universal address system of an ordered rooted tree.

The lexicographic ordering of the labelings is

0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3

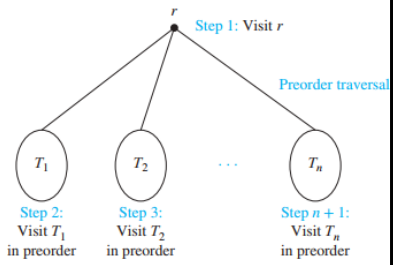


Traversal Algorithms

- A traversal algorithm is a procedure for systematically visiting every vertex of an ordered rooted tree
  - An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.
- The three common traversals are: preorder, inorder and postorder.

Preorder Traversal

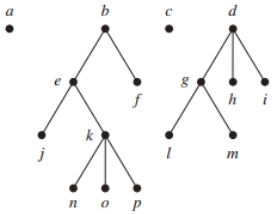
- Pre-order: **Ro**ot, **L**eft, **Ri**ght
  - Step 1. Visit r
  - Step 2. Visit T_1
  - Step 3. Visit T_2
  - …
  - Step $n + 1$. Visit T_n

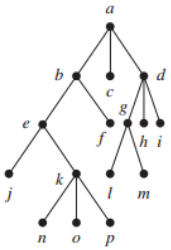


Preorder Traversal

Example. Preorder traversal: Visit root, visit subtrees left to right

Step 1.

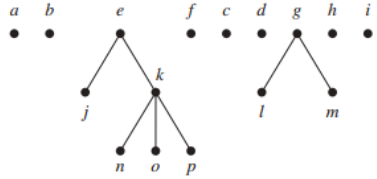


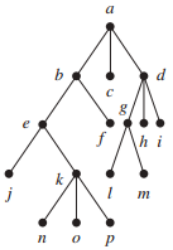


Preorder Traversal

Example. Preorder traversal: Visit root, visit subtrees left to right

Step 2.

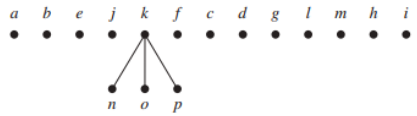




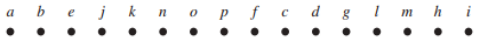
Preorder Traversal

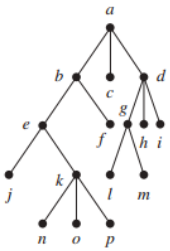
Example. Preorder traversal: Visit root, visit subtrees left to right

Step 3.



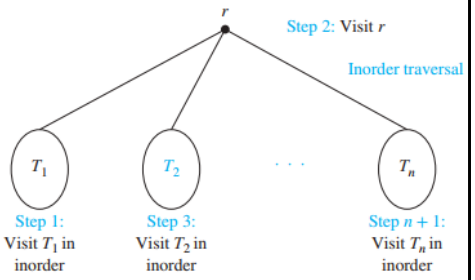
Step 4.





Inorder Traversal

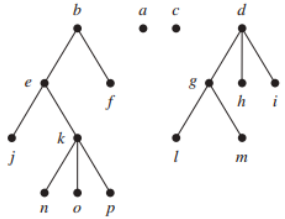
- Inorder: **L**eft, **Ro**ot, **R**ight.
  - Step 1. Visit T_1
  - Step 2. Visit r
  - Step 3. Visit T_2
  - …
  - Step $n + 1$. Visit T_n

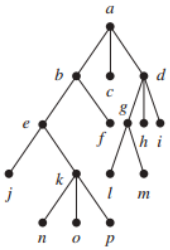


Inorder Traversal

Example. Inorder traversal: Visit leftmost subtree, visit root, visit other subtrees left to right

Step 1.

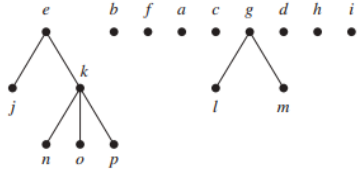


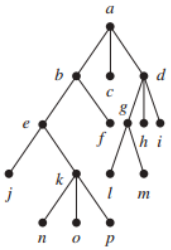


Inorder Traversal

Example. Inorder traversal: Visit leftmost subtree, visit root, visit other subtrees left to right

Step 2.

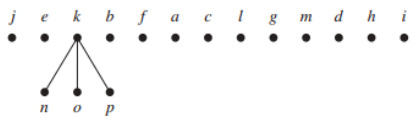




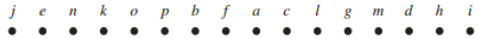
Inorder Traversal

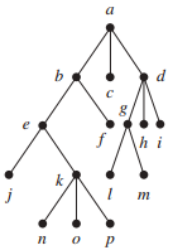
Example. Inorder traversal: Visit leftmost subtree, visit root, visit other subtrees left to right

Step 3.



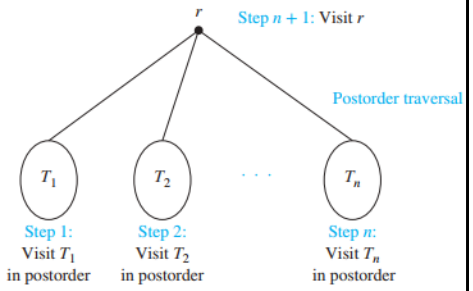
Step 4.





Postorder Traversal

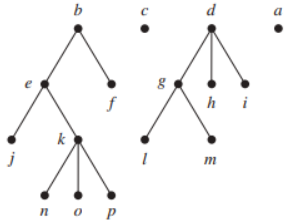
- Postorder: **L**eft, **R**ight, **Ro**ot.
  - Step 1. Visit T_1
  - Step 2. Visit T_2
  - …
  - Step n. Visit T_n
  - Step $n + 1$. Visit r

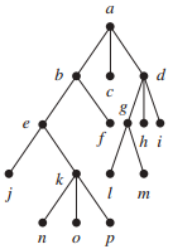


Postorder Traversal

Example. Postorder traversal: Visit subtrees left to right; visit root

Step 1.

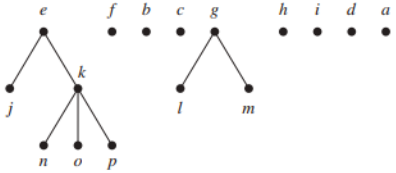


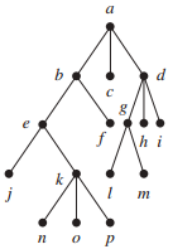


Postorder Traversal

Example. Postorder traversal: Visit subtrees left to right; visit root

Step 2.

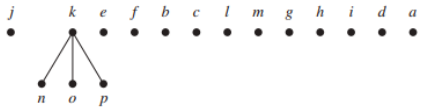




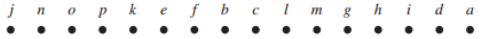
Postorder Traversal

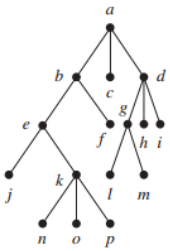
Example. Postorder traversal: Visit subtrees left to right; visit root

Step 3.



Step 4.





Summary

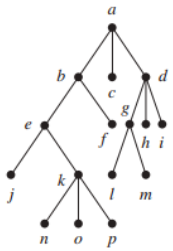
- Preorder: Root, Left, Right.
- Inorder: Left, Root, Right.
- Postorder: Left, Right, Root.

Example.

Preorder: a b e j k n o p f c d g l m h i

Inorder: j e n k o p b f a c l g m d h i

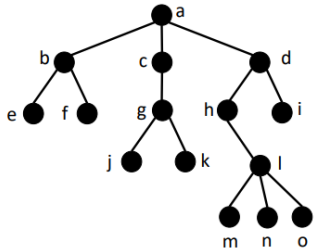
Postorder: j n o p k e f b c l m g h i d a



Tree Traversals

Example. Consider the following tree T.

What are the order of characters after applying preorder/inorder/postorder?



Representing Arithmetic Expressions

- Complicated arithmetic expressions can be represented by an ordered rooted tree
  - Internal vertices represent operators (+ (addition), − (subtraction), ∗ (multiplication), / (division), and ↑ (exponentiation))
  - Leaves represent operands
- Build the tree bottom-up
  - Construct smaller subtrees
  - Incorporate the smaller subtrees as part of larger subtrees

Representing Arithmetic Expressions

Example. What is the ordered rooted tree that represents the expression $((x + y)\ \ 2) + ((x - 4)/3)$

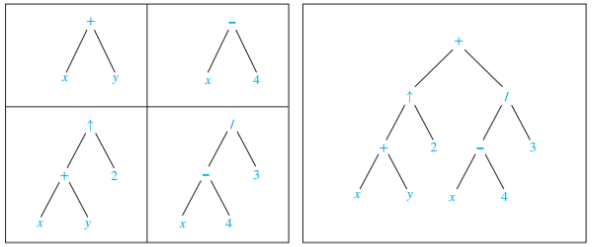


FIGURE 10 **A binary tree representing $((x + y) \uparrow 2) + ((x - 4)/3)$.**

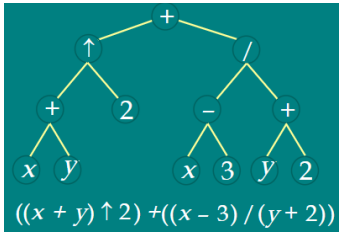
Representing Arithmetic Expressions

Example. What is the ordered rooted tree that represents the expression $(x + y) \uparrow 2 + (x - 3)/(y + 2)$

Infix Notation

- Traverse in **inorder** adding parentheses for each operation

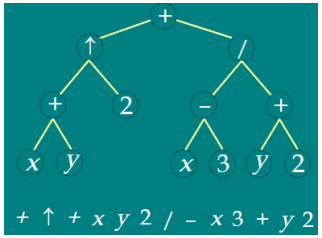
Example.



Prefix Notation (Polish Notation)

- Traverse in **preorder**. (no parenthesis needed)

Example.



Evaluating Prefix Notation

- In an prefix expression, a binary operator precedes its two operands
- The expression is evaluated **right – left**
- Look for the **first operator** from the right
- Evaluate the operator with the two operands immediately to its right

Evaluating Prefix Notation

Example. What is the value of the prefix expression + – * 2 3 5 / ↑ 2 3 4?

+ – * 2 3 5 / ↑ 2 3 4

2 ↑ 3 = 8

+ – * 2 3 5 / 8 4

8 / 4 = 2

+ – * 2 3 5 2

2 * 3 = 6

+ – 6 5 2

6 – 5 = 1

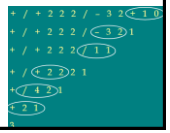
+ 1 2

1 + 2 = 3

Value of expression: 3

Evaluating Prefix Notation

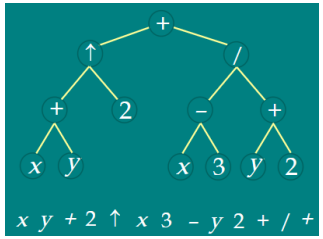
Example. What is the value of the prefix expression + / + 2 2 2 / – 3 2 + 1 0?



Postfix Notation (Reverse Polish)

Traverse in **postorder**. (no parenthesis needed)

Example.



x y + 2 ↑ x 3 – y 2 + / +

Evaluating Postfix Notation

- In an postfix expression, a binary operator follows its two operands
- The expression is evaluated **left – right**
- Look for the **first operator** from the left
- Evaluate the operator with the two operands immediately to its left

Evaluating Postfix Notation

Example. What is the value of the prefix expression 7 2 3 ∗ − 4 ↑ 9 3 / +?

7 2 3 ∗ − 4 ↑ 9 3 / +

$2 * 3 = 6$

7 6 − 4 ↑ 9 3 / +

$7 - 6 = 1$

1 4 ↑ 9 3 / +

$1^4 = 1$

1 9 3 / +

$9 / 3 = 3$

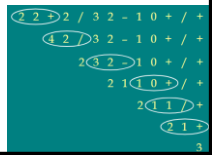
1 3 +

$1 + 3 = 4$

Value of expression: 4

Evaluating Postfix Notation

Example. What is the value of the postfix expression 2 2 + 2 / 3 2 − 1 0 + / +?



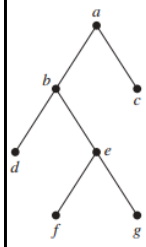
Example

Find the ordered rooted tree representing the compound proposition (¬(p ∧ q)) ↔ (¬p ∨ ¬q). Then use this rooted tree to find the prefix, postfix, and infix forms of this expression.

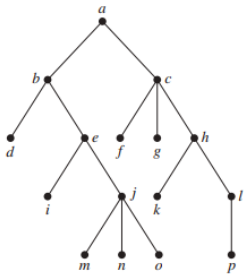
Exercises

1. Determine the order in which a preorder traversal visits the vertices of the given ordered rooted tree.

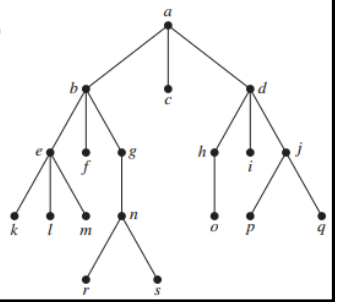
a)



b)



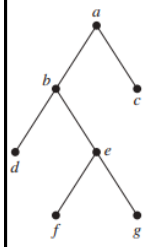
c)



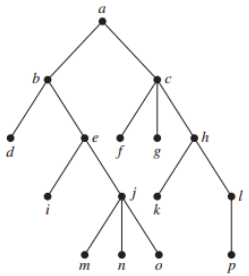
Exercises

2. Determine the order in which a inorder traversal visits the vertices of the given ordered rooted tree.

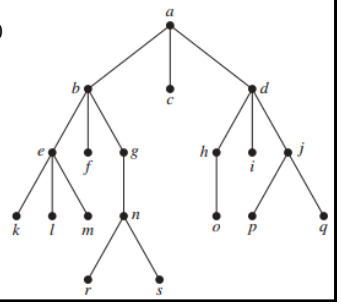
a)



b)



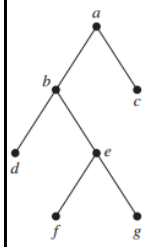
c)



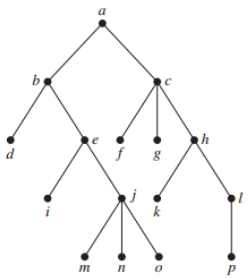
Exercises

3. Determine the order in which a postorder traversal visits the vertices of the given ordered rooted tree.

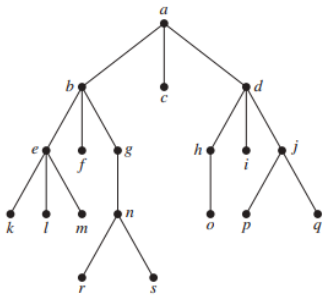
a)



b)



c)



Exercises

4. a) Represent the expression $((x + 2) \uparrow 3) * (y - (3 + x)) - 5$ using a binary tree.
Write this expression in
b) prefix notation. c) postfix notation. d) infix notation.

5. a) Represent the expressions $(x + xy) + (x/y)$ and $x + ((xy + x)/y)$ using binary trees.
Write these expressions in
b) prefix notation. c) postfix notation. d) infix notation.

6. a) Represent the compound propositions $\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q)$ and $(\neg p \wedge (q \leftrightarrow \neg p)) \vee \neg q$ using ordered rooted trees.
Write these expressions in
b) prefix notation. c) postfix notation. d) infix notation.

Exercises

7. Draw the ordered rooted tree corresponding to each of these arithmetic expressions written in prefix notation. Then write each expression using infix notation.

a) + ∗ + − 5 3 2 1 4 b) ↑ + 2 3 − 5 1 c) ∗ / 9 3 + ∗ 2 4 − 7 6

8. What is the value of each of these prefix expressions?

a) − ∗ 2 / 8 4 3 c) + − ↑ 3 2 ↑ 2 3 / 6 − 4 2

b) ↑ − ∗ 3 3 ∗ 4 2 5 d) ∗ +3 + 3 ↑ 3 + 3 3 3

9. What is the value of each of these postfix expressions?

a) 5 2 1 − − 3 1 4 ++ ∗

b) 9 3 / 5 + 7 2 − ∗

c) 3 2 ∗ 2 ↑ 5 3 − 8 4 / ∗ −

Exercises

10. Set up a binary search tree for the following list, in the given order, using alphabetical ordering 5, 3, 6, 2, 4, 7. Write the preorder traversal of the tree

A. 5, 2, 3, 4, 6, 7 C. 5, 3, 2, 4, 6, 7

B. 2, 4, 3, 7, 6, 5 D. 2, 3, 4, 5, 6, 7

11. The following prefix expression will be evaluated to + − ∗ 2 3 4 / + 1 2 3

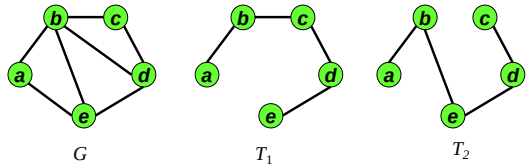
A. 4 B. 2 C. 1 D. 3

SPANNING TREES

Spanning Tree

- **Definition.** Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.
- Spanning trees play an important role in multicasting over Internet Protocol (IP) networks.

Example. A simple graph G and 2 spanning trees of G

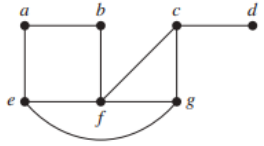


G T_1 T_2

Spanning Tree

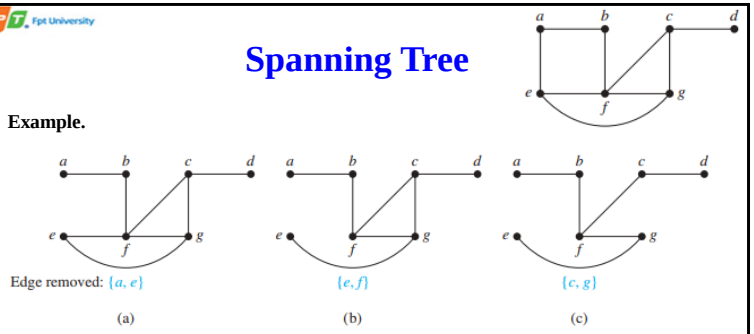
- **Theorem.** A simple graph is connected if and only if it has a spanning tree.
- To obtain a spanning tree from a connected undirected graph with cycles, we can remove edges until there are no cycles.

Example. Find a spanning tree of the simple graph G



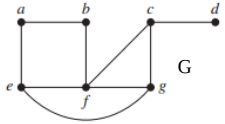
Spanning Tree

Example.



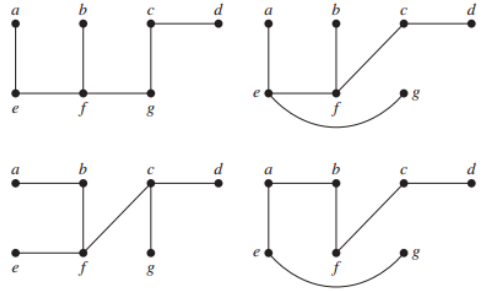
Producing a spanning tree for G by removing edges that form simple circuits.

Spanning Tree



G

Example.



Spanning trees of G

Spanning Tree

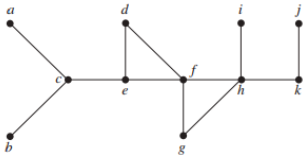
- Finding spanning trees by removing edges from simple circuits can be inefficient because it requires that simple circuits be identified.
- Instead of constructing spanning trees by removing edges, spanning trees can be built up by successively **adding edges**. Two algorithms based on this principle are
  - Depth-first search
  - Breadth-first search

Depth-First Search (Backtracking)

- Arbitrarily choose a vertex of the graph as the root.
- Form a path starting at this vertex by successively adding vertices and edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path.
- Continue adding vertices and edges to this path as long as possible.
  1. If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree.
  2. If the path does NOT go through all vertices, more vertices and edges must be added. Move back to the next to last vertex in the path, and, if possible, form a new path starting at this vertex passing through vertices that were not already visited. If this cannot be done, move back another vertex in the path, that is, two vertices back in the path, and try again.
- Repeat.

Depth-First Search

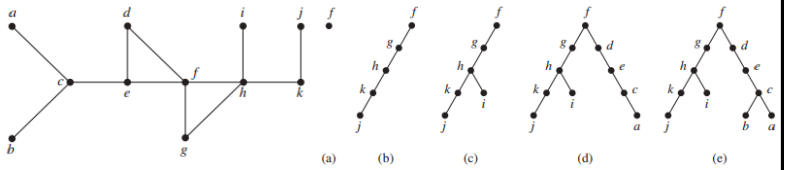
Example. Use depth-first search to find a spanning tree for the graph G



The graph G

Depth-First Search

Example. Use depth-first search to find a spanning tree for the graph G



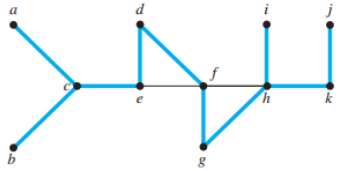
The graph G

Deep-first search of G

Depth-First Search

- The edges selected by depth-first search of a graph are called **tree edges**.
- All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree. These edges are called **back edges**

Example. The tree edges (highlight) and back edges (thinner black lines) of the depth-first search in the previous example



Deep-First Search Algorithm

procedure *DFS*(*G*: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of the vertex v_1
visit(v_1)

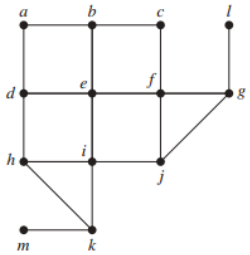
procedure *visit*(*v*: vertex of *G*)
for each vertex *w* adjacent to *v* and not yet in *T*
  add vertex *w* and edge {*v*, *w*} to *T*
  visit(*w*)

Breadth-First Search

- Arbitrarily choose a root from the vertices of the graph. Then add all edges incident to this vertex. The new vertices added at this stage become the vertices at level 1 in the spanning tree. Arbitrarily order them.
- Next, for each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit.
- Arbitrarily order the children of each vertex at level 1. This produces the vertices at level 2 in the tree.
- Follow the same procedure until all the vertices in the tree have been added. The procedure ends because there are only a finite number of edges in the graph.

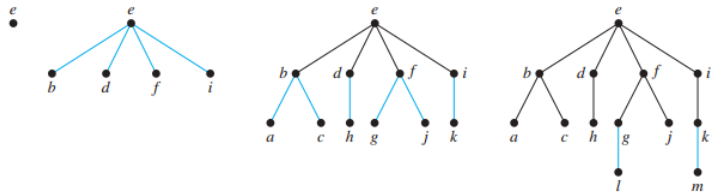
Breadth-First Search

Example. Use breadth-first search to find a spanning tree for the graph G



Breadth-First Search

Example. Breadth-first search of G

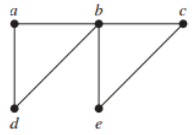


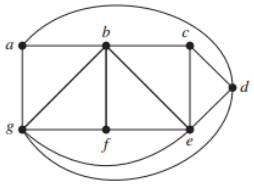
Breadth-First Search Algorithm

procedure *BFS*(*G*: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of vertex v_1
L := empty list
put v_1 in the list L of unprocessed vertices
while L is not empty
  remove the first vertex, v, from L
  for each neighbor w of v
    if w is not in L and not in T **then**
      add w to the end of the list L
      add w and edge $\{v, w\}$ to T

Exercises

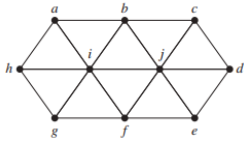
1. How many edges must be removed from a connected graph with n vertices and m edges to produce a spanning tree?
2. Find a spanning tree for the graph shown by removing edges in simple circuits.

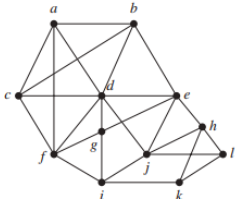
a) 

b) 

Exercises

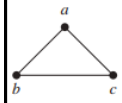
2. Find a spanning tree for the graph shown by removing edges in simple circuits (cont.)

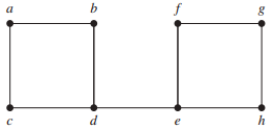
c) 

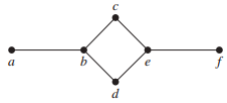
d) 

Exercises

3. Find a spanning tree for each of these graphs.
a) K_5 b) $K_{4,4}$ c) $K_{1,6}$ d) Q_3 e) C_5 f) W_5
4. Draw all the spanning trees of the given simple graphs

a) 

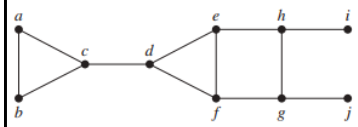
b) 

c) 

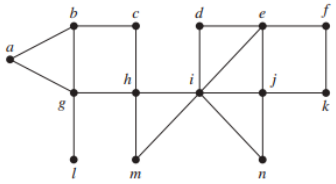
Exercises

5. Use depth-first search to produce a spanning tree for the given simple graph. Choose a as the root of this spanning tree and assume that the vertices are ordered alphabetically.

a)



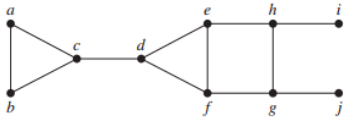
b)



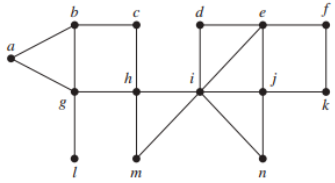
Exercises

6. Use breadth-first search to produce a spanning tree for the given simple graph. Choose a as the root of each spanning tree..

a)

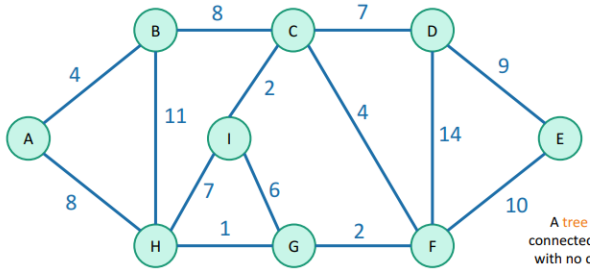


b)



MINIMUM SPANNING TREES

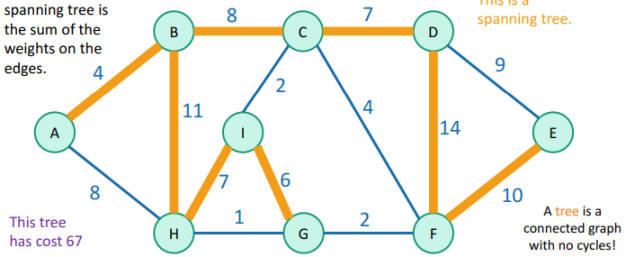
Minimum Spanning Tree



A spanning tree is a tree that connects all of the vertices.

Minimum Spanning Tree

The cost of a spanning tree is the sum of the weights on the edges.

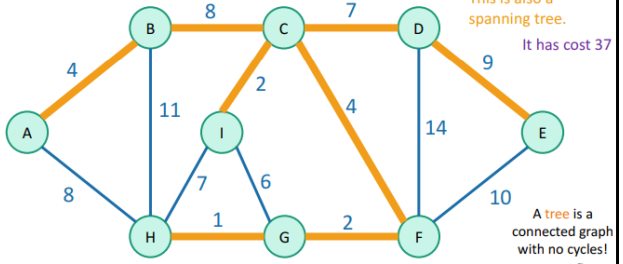


A spanning tree is a tree that connects all of the vertices.

Minimum Spanning Tree

Which is a minimum spanning tree?

This is a minimum spanning tree



A spanning tree is a tree that connects all of the vertices.

Minimum Spanning Tree

- Some applications
  - We want to connect phone lines to houses, but laying down cable is expensive. How can we minimize the amount of wire we must install?
  - We have items on a circuit we want to be "electrically equivalent". How can we connect them together using a minimum amount of wire?
  - Implement efficient multiple constant multiplication
  - Minimizing number of packets transmitted across a network
  - Machine learning (e.g. real-time face verification)
  - Graphics (e.g. image segmentation)

Algorithms for Minimum Spanning Trees

- **Minimum spanning tree problem.**
  - **Given:** An undirected, weighted graph G
  - **Find:** A minimum weight set of edges such that you can get from any vertex of G to any other on only those edges.
- **Definition.** A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.
- Two algorithms for constructing minimum spanning trees that proceed by successively adding edges of smallest weight from those edges with a specified property that have not already been used are
  - Prim's algorithm (Prim-Jarn´ık algorithm)
  - Kruskal's Algorithm

Prim's Algorithm

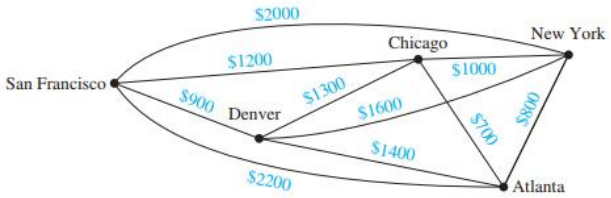
- Begin by choosing any edge with smallest weight, putting it into the spanning tree.
- Successively add to the tree edges of minimum weight that are incident to a vertex already in the tree, never forming a simple circuit with those edges already in the tree.
- Stop when n − 1 edges have been added.

Prim's Algorithm

procedure *Prim*(*G*: weighted connected undirected graph with *n* vertices)
T := a minimum-weight edge
for *i* := 1 **to** *n* − 2
  e := an edge of minimum weight incident to a vertex in *T* and not forming a simple circuit in *T* if added to *T*
  T := *T* with *e* added
return *T* {*T* is a minimum spanning tree of *G*}

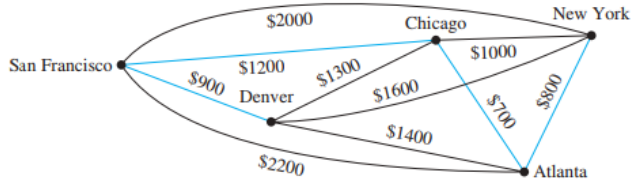
Prim's Algorithm

Example. Use Prim's algorithm to design a minimum-cost communications network connecting all the computers represented by the graph.



Prim's Algorithm

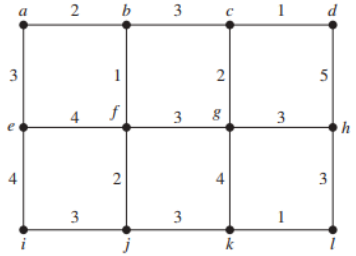
Example.



Choice	Edge	Cost
1	{Chicago, Atlanta}	$ 700
2	{Atlanta, New York}	$ 800
3	{Chicago, San Francisco}	$1200
4	{San Francisco, Denver}	$ 900
	Total:	$3600

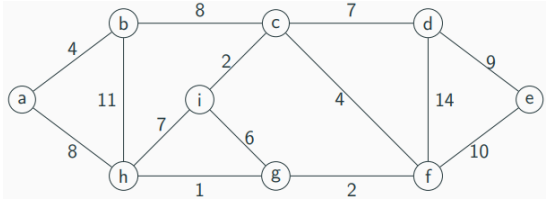
Prim's Algorithm

Example. Use Prim's algorithm to find a minimum spanning tree in the graph



Prim's Algorithm

Example. Use Prim's algorithm to find a minimum spanning tree in the graph



Kruskal's Algorithm

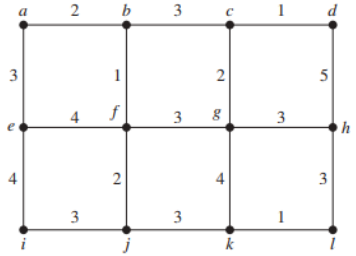
- Choose an edge in the graph with minimum weight.
- Successively add edges with minimum weight that do not form a simple circuit with those edges already chosen.
- Stop after n − 1 edges have been selected.

Kruskal's Algorithm

procedure *Kruskal*(G: weighted connected undirected graph with n vertices)
T := empty graph
for $i := 1$ **to** $n - 1$
  e := any edge in G with smallest weight that does not form a simple circuit when added to T
  $T := T$ with e added
return T {T is a minimum spanning tree of G}

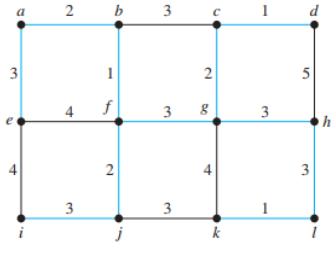
Kruskal's Algorithm

Example. Use Kruskal's algorithm to find a minimum spanning tree in the weighted graph



Kruskal's Algorithm

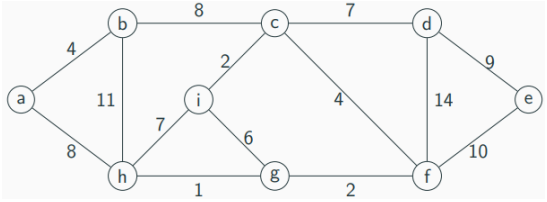
Example.



Choice	Edge	Weight
1	{b, f}	1
2	{a, b}	2
3	{f, j}	2
4	{a, e}	3
5	{i, j}	3
6	{f, g}	3
7	{c, g}	2
8	{c, d}	1
9	{g, h}	3
10	{h, l}	3
11	{k, l}	1
	Total:	24

Kruskal's Algorithm

Example. Use Kruskal's algorithm to find a minimum spanning tree in the weighted graph



Minimum Spanning Tree Algorithms

Prim's algorithm

- Edges of minimum weight that are incident to a vertex already in the tree, and not forming a circuit, are chosen.
- If the edges are not ordered, there may be more than one choice for the edge to add at a stage of this procedure.

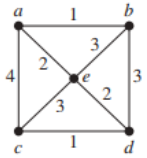
Kruskal's algorithm

- Edges of minimum weight that are not necessarily incident to a vertex already in the tree, and that do not form a circuit, are chosen.

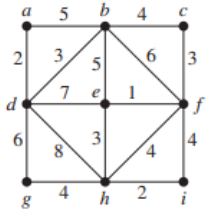
Exercises

1. Use Prim's algorithm to find a minimum spanning tree for the given weighted graph.

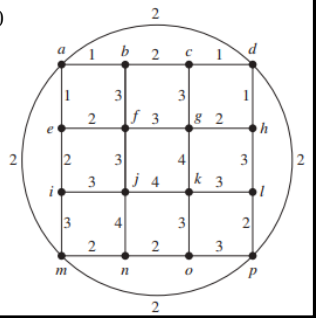
a)



b)



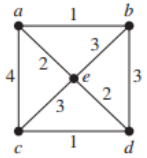
c)



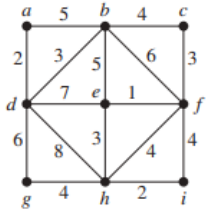
Exercises

2. Use Kruskal's algorithm to find a minimum spanning tree for the given weighted graph.

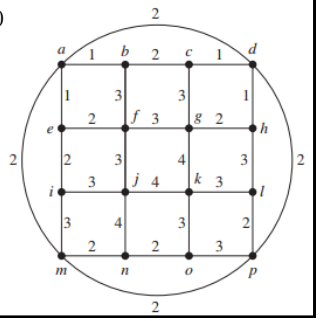
a)



b)

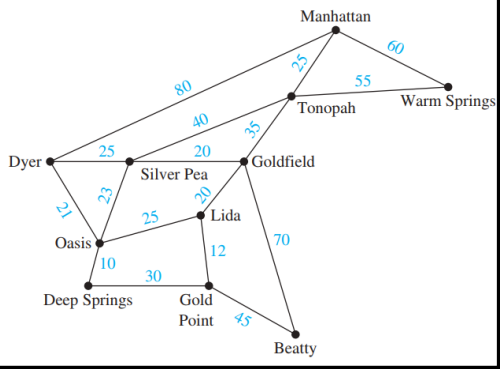


c)



Exercises

3. The roads represented by this graph are all unpaved. The lengths of the roads between pairs of towns are represented by edge weights. Which roads should be paved so that there is a path of paved roads between each pair of towns so that a minimum road length is paved? (Note: These towns are in Nevada.)



Thanks