Discrete Mathematics 2

Chapter 10: Trees

Department of Mathematics
The FPT university

Chapter 10: Introduction

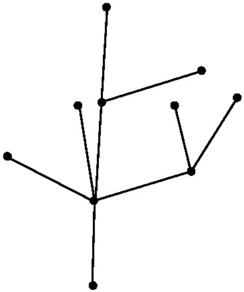
Topics covered:

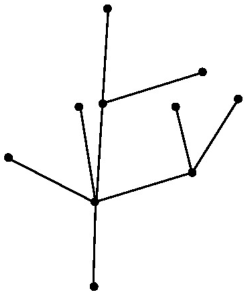
Topics covered:

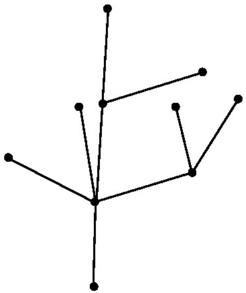
Topics covered:

Topics covered:

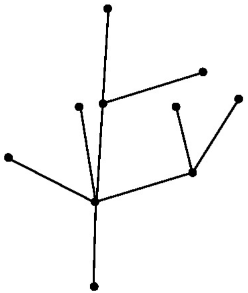
10.1 Introduction to Trees







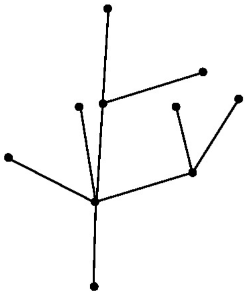
A tree is a connected undirected simple graph with no simple circuits.



A tree is a connected undirected simple graph with no simple circuits.

Theorem

10.1 Introduction to Trees

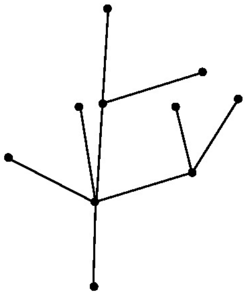


A tree is a connected undirected simple graph with no simple circuits.

Theorem

- An undirected graph is a tree if and only if between any two vertices there is a unique simple path.

10.1 Introduction to Trees



A tree is a connected undirected simple graph with no simple circuits.

Theorem

- An undirected graph is a tree if and only if between any two vertices there is a unique simple path.
- A tree with n vertices has $n - 1$ edges.

Rooted Trees

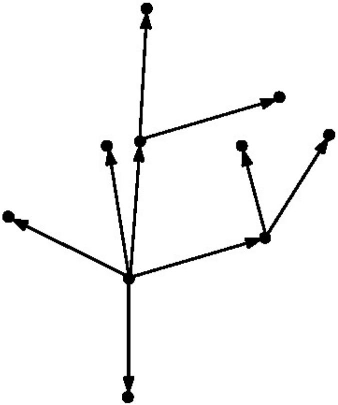
Rooted Trees

Rooted Trees

A rooted tree is a tree in which one vertex is designed as a root and every edge is directed away from the root.

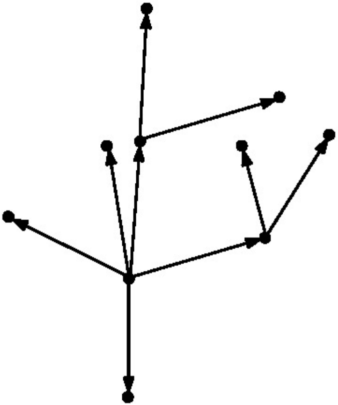
Rooted Trees

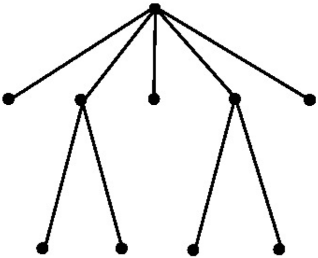
A rooted tree is a tree in which one vertex is designed as a root and every edge is directed away from the root.



Rooted Trees

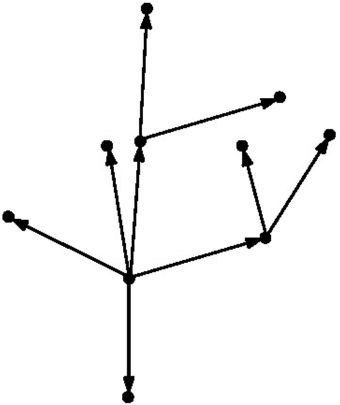
A rooted tree is a tree in which one vertex is designed as a root and every edge is directed away from the root.

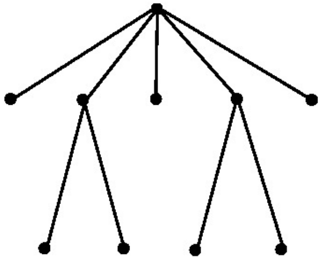




Rooted Trees

A rooted tree is a tree in which one vertex is designed as a root and every edge is directed away from the root.





Terminologies

Terminologies

- If there is an edge directed from u to v then u is called the parent of v and v is called a child of u.

Terminologies

- If there is an edge directed from u to v then u is called the parent of v and v is called a child of u.
- Vertices having the same parent are called siblings.

Terminologies

- If there is an edge directed from u to v then u is called the parent of v and v is called a child of u.
- Vertices having the same parent are called siblings.
- Ancestors of a vertex are the vertices on the path from the root to that vertex, excluding the vertex itself.

Terminologies

- If there is an edge directed from u to v then u is called the parent of v and v is called a child of u.
- Vertices having the same parent are called siblings.
- Ancestors of a vertex are the vertices on the path from the root to that vertex, excluding the vertex itself.
- Descendants of a vertex v are the vertices that have v as an ancestor.

Terminologies

- If there is an edge directed from u to v then u is called the parent of v and v is called a child of u.
- Vertices having the same parent are called siblings.
- Ancestors of a vertex are the vertices on the path from the root to that vertex, excluding the vertex itself.
- Descendants of a vertex v are the vertices that have v as an ancestor.
- Leaves of a tree are the vertices that do not have children.

Terminologies

- If there is an edge directed from u to v then u is called the parent of v and v is called a child of u.
- Vertices having the same parent are called siblings.
- Ancestors of a vertex are the vertices on the path from the root to that vertex, excluding the vertex itself.
- Descendants of a vertex v are the vertices that have v as an ancestor.
- Leaves of a tree are the vertices that do not have children.
- Internal vertices are the vertices that have children.

Terminologies

- If there is an edge directed from u to v then u is called the parent of v and v is called a child of u.
- Vertices having the same parent are called siblings.
- Ancestors of a vertex are the vertices on the path from the root to that vertex, excluding the vertex itself.
- Descendants of a vertex v are the vertices that have v as an ancestor.
- Leaves of a tree are the vertices that do not have children.
- Internal vertices are the vertices that have children.
- A subtree with root a is the subgraph consisting of a and all its descendants and all edges incident to these descendants.

Terminologies

- If there is an edge directed from u to v then u is called the parent of v and v is called a child of u.
- Vertices having the same parent are called siblings.
- Ancestors of a vertex are the vertices on the path from the root to that vertex, excluding the vertex itself.
- Descendants of a vertex v are the vertices that have v as an ancestor.
- Leaves of a tree are the vertices that do not have children.
- Internal vertices are the vertices that have children.
- A subtree with root a is the subgraph consisting of a and all its descendants and all edges incident to these descendants.

m-ary Trees

m-ary Trees

- A rooted tree is called an m-ary tree if each vertex has at most m children.

m-ary Trees

- A rooted tree is called an *m-ary tree* if each vertex has at most m children.
- A rooted tree is called a full m-ary tree if each internal vertex has exactly m children.

m-ary Trees

- A rooted tree is called an *m-ary tree* if each vertex has at most m children.
- A rooted tree is called a full m-ary tree if each internal vertex has exactly m children.
- A 2-ary tree is called a binary tree.

m-ary Trees

- A rooted tree is called an m-ary tree if each vertex has at most m children.
- A rooted tree is called a full m-ary tree if each internal vertex has exactly m children.
- A 2-ary tree is called a binary tree.

Theorem

Theorem

If n is the number of vertices and i is the number of internal vertices of a full m-ary tree then $n = mi + 1$.

Theorem

If n is the number of vertices and i is the number of internal vertices of a full m-ary tree then $n = mi + 1$.

Example 1. Does there exist a full 4-ary tree with 80 leaves?

Theorem

If n is the number of vertices and i is the number of internal vertices of a full m-ary tree then $n = mi + 1$.

Example 1. Does there exist a full 4-ary tree with 80 leaves?

Example 2.

Theorem

If n is the number of vertices and i is the number of internal vertices of a full m-ary tree then $n = mi + 1$.

Example 1. Does there exist a full 4-ary tree with 80 leaves?

Example 2. Some one starts a chain letter game. Each person receiving the letter either sends it to exactly four other persons, or to no one.

Theorem

If n is the number of vertices and i is the number of internal vertices of a full m-ary tree then $n = mi + 1$.

Example 1. Does there exist a full 4-ary tree with 80 leaves?

Example 2. Some one starts a chain letter game. Each person receiving the letter either sends it to exactly four other persons, or to no one. Assume that no one receives more than one letter.

Theorem

If n is the number of vertices and i is the number of internal vertices of a full m-ary tree then $n = mi + 1$.

Example 1. Does there exist a full 4-ary tree with 80 leaves?

Example 2. Some one starts a chain letter game. Each person receiving the letter either sends it to exactly four other persons, or to no one. Assume that no one receives more than one letter. The game ends when there have been exactly 100 persons receiving the letter and not sending it out.

Theorem

If n is the number of vertices and i is the number of internal vertices of a full m-ary tree then $n = mi + 1$.

Example 1. Does there exist a full 4-ary tree with 80 leaves?

Example 2. Some one starts a chain letter game. Each person receiving the letter either sends it to exactly four other persons, or to no one. Assume that no one receives more than one letter. The game ends when there have been exactly 100 persons receiving the letter and not sending it out.

How many persons received the letter (including the one who starts the game)?

Theorem

If n is the number of vertices and i is the number of internal vertices of a full m-ary tree then $n = mi + 1$.

Example 1. Does there exist a full 4-ary tree with 80 leaves?

Example 2. Some one starts a chain letter game. Each person receiving the letter either sends it to exactly four other persons, or to no one. Assume that no one receives more than one letter. The game ends when there have been exactly 100 persons receiving the letter and not sending it out.

How many persons received the letter (including the one who starts the game)?

How many persons sent out the letter?

- The level of a vertex in a rooted tree is the length of the unique path from the root to that vertex.

- The level of a vertex in a rooted tree is the length of the unique path from the root to that vertex.
- The height of a rooted tree is the maximum of the levels of the vertices.

- The level of a vertex in a rooted tree is the length of the unique path from the root to that vertex.
- The height of a rooted tree is the maximum of the levels of the vertices.
- An m-ary tree of height h is called a balanced tree if each leave has level h or $h-1$.

- The level of a vertex in a rooted tree is the length of the unique path from the root to that vertex.
- The height of a rooted tree is the maximum of the levels of the vertices.
- An m-ary tree of height h is called a balanced tree if each leave has level h or $h-1$.

Theorem

- The level of a vertex in a rooted tree is the length of the unique path from the root to that vertex.
- The height of a rooted tree is the maximum of the levels of the vertices.
- An m-ary tree of height h is called a balanced tree if each leave has level h or $h-1$.

Theorem

- In an m-ary tree with height h and l leaves we have $l \leq m^h$.

- The level of a vertex in a rooted tree is the length of the unique path from the root to that vertex.
- The height of a rooted tree is the maximum of the levels of the vertices.
- An m-ary tree of height h is called a balanced tree if each leave has level h or $h-1$.

Theorem

- In an m-ary tree with height h and l leaves we have $l \leq m^h$. Therefore $h \geq \lceil \log_m l \rceil$.

- The level of a vertex in a rooted tree is the length of the unique path from the root to that vertex.
- The height of a rooted tree is the maximum of the levels of the vertices.
- An m-ary tree of height h is called a balanced tree if each leave has level h or $h-1$.

Theorem

- In an m-ary tree with height h and l leaves we have $l \leq m^h$. Therefore $h \geq \lceil \log_m l \rceil$.
- In a balanced m-ary tree we have $h = \lceil \log_m l \rceil$

- The level of a vertex in a rooted tree is the length of the unique path from the root to that vertex.
- The height of a rooted tree is the maximum of the levels of the vertices.
- An m-ary tree of height h is called a balanced tree if each leave has level h or $h-1$.

Theorem

- In an m-ary tree with height h and l leaves we have $l \leq m^h$. Therefore $h \geq \lceil \log_m l \rceil$.
- In a balanced m-ary tree we have $h = \lceil \log_m l \rceil$

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

10.2 Applications of Trees

Binary Search Trees.

Binary Search Trees.

Example.

Binary Search Trees.

Example.

- Form a binary search tree for the words: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, and *chemistry*

Binary Search Trees.

Example.

- Form a binary search tree for the words: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, and *chemistry*
- How many comparisons needed to determine whether the words *meteorology* or *oceanography* are already in the list?

Binary Search Trees.

Example.

- Form a binary search tree for the words: *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, and *chemistry*
- How many comparisons needed to determine whether the words *meteorology* or *oceanography* are already in the list?

Binary Search Trees.

Example.

- Form a binary search tree for the words: *mathematics, physics, geography, zoology, meteorology, geology, psychology,* and *chemistry*
- How many comparisons needed to determine whether the words *meteorology* or *oceanography* are already in the list?

Theorem

Binary Search Trees.

Example.

- Form a binary search tree for the words: *mathematics, physics, geography, zoology, meteorology, geology, psychology,* and *chemistry*
- How many comparisons needed to determine whether the words *meteorology* or *oceanography* are already in the list?

Theorem

The least number of comparisons to locate an element in a list of length n is $\lceil \log(n+1) \rceil$.

Binary Search Trees.

Example.

- Form a binary search tree for the words: *mathematics, physics, geography, zoology, meteorology, geology, psychology,* and *chemistry*
- How many comparisons needed to determine whether the words *meteorology* or *oceanography* are already in the list?

Theorem

The least number of comparisons to locate an element in a list of length n is $\lceil \log(n+1) \rceil$.

Decision Trees.

Decision Trees.

Example. There are 8 coins among which 7 coins are of equal weight, and a counterfeit coin that weighs less than the others. How many weighing are necessary using a balance scale to determine the counterfeit coin?

Decision Trees.

Example. There are 8 coins among which 7 coins are of equal weight, and a counterfeit coin that weighs less than the others. How many weighing are necessary using a balance scale to determine the counterfeit coin?

Complexity of Sorting Algorithms

Complexity of Sorting Algorithms

Example. How many comparisons are necessary to sort a list of 3 numbers using binary comparisons?

Complexity of Sorting Algorithms

Example. How many comparisons are necessary to sort a list of 3 numbers using binary comparisons?

Theorem

Complexity of Sorting Algorithms

Example. How many comparisons are necessary to sort a list of 3 numbers using binary comparisons?

Theorem

A sorting algorithm based on binary comparisons requires at least $\lceil \log(n!) \rceil$ comparisons.

Complexity of Sorting Algorithms

Example. How many comparisons are necessary to sort a list of 3 numbers using binary comparisons?

Theorem

A sorting algorithm based on binary comparisons requires at least $\lceil \log(n!) \rceil$ comparisons.

Note.

Complexity of Sorting Algorithms

Example. How many comparisons are necessary to sort a list of 3 numbers using binary comparisons?

Theorem

A sorting algorithm based on binary comparisons requires at least $\lceil \log(n!) \rceil$ comparisons.

Note.

$$\log(n!) = \log 1 + \log 2 + \cdots + \log n = \Theta(n \log n)$$

Prefix Codes

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Problem: Is this possible to encode this message using fewer bits?

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Problem: Is this possible to encode this message using fewer bits?

We cannot use bit strings of length 2 to represent the letters of this message. We need to use bit strings of different lengths.

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Problem: Is this possible to encode this message using fewer bits?

We cannot use bit strings of length 2 to represent the letters of this message. We need to use bit strings of different lengths.

However, there might be troubles when decoding the encoded message.

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Problem: Is this possible to encode this message using fewer bits?

We cannot use bit strings of length 2 to represent the letters of this message. We need to use bit strings of different lengths.

However, there might be troubles when decoding the encoded message.

Example.

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Problem: Is this possible to encode this message using fewer bits?

We cannot use bit strings of length 2 to represent the letters of this message. We need to use bit strings of different lengths.

However, there might be troubles when decoding the encoded message.

Example. If encode the message "tea" using representations

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Problem: Is this possible to encode this message using fewer bits?

We cannot use bit strings of length 2 to represent the letters of this message. We need to use bit strings of different lengths.

However, there might be troubles when decoding the encoded message.

Example. If encode the message "tea" using representations

e: 0
a: 1
t: 01

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Problem: Is this possible to encode this message using fewer bits?

We cannot use bit strings of length 2 to represent the letters of this message. We need to use bit strings of different lengths.

However, there might be troubles when decoding the encoded message.

Example. If encode the message "tea" using representations

e: 0
a: 1
t: 01

the bit string "0101" can be decoded as "eat", "eaea" or "tt".

Prefix Codes

To encode the message "good morning" by a bit string we can represent each letter by a bit string of length 3. In this case the total number of bits used is 33.

Problem: Is this possible to encode this message using fewer bits?

We cannot use bit strings of length 2 to represent the letters of this message. We need to use bit strings of different lengths.

However, there might be troubles when decoding the encoded message.

Example. If encode the message "tea" using representations

e: 0
a: 1
t: 01

the bit string "0101" can be decoded as "eat", "eaea" or "tt".

Prefix codes

Prefix codes

- Prefix codes are bit strings used to represent letters in such a way that no string is the first part of another string.

Prefix codes

- Prefix codes are bit strings used to represent letters in such a way that no string is the first part of another string.
- Binary trees can be used to create prefix codes.

Prefix codes

- Prefix codes are bit strings used to represent letters in such a way that no string is the first part of another string.
- Binary trees can be used to create prefix codes.

Example.

Prefix codes

- Prefix codes are bit strings used to represent letters in such a way that no string is the first part of another string.
- Binary trees can be used to create prefix codes.

Example. Use binary trees to create prefix codes to encode the message "good morning".

Prefix codes

- Prefix codes are bit strings used to represent letters in such a way that no string is the first part of another string.
- Binary trees can be used to create prefix codes.

Example. Use binary trees to create prefix codes to encode the message "good morning".

Question. How many bits are used?

Prefix codes

- Prefix codes are bit strings used to represent letters in such a way that no string is the first part of another string.
- Binary trees can be used to create prefix codes.

Example. Use binary trees to create prefix codes to encode the message "good morning".

Question. How many bits are used? Is this possible to construct a binary tree for which fewer bits are used?

Prefix codes

- Prefix codes are bit strings used to represent letters in such a way that no string is the first part of another string.
- Binary trees can be used to create prefix codes.

Example. Use binary trees to create prefix codes to encode the message "good morning".

Question. How many bits are used? Is this possible to construct a binary tree for which fewer bits are used?

Huffman Coding

Huffman Coding

Huffman Coding

- Write out all distinct letters of the message together with their frequencies.

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex.

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex. Call the frequency corresponding to each tree its weight.

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex. Call the frequency corresponding to each tree its weight.
- At each step, combine two trees whose sum of weights is a minimum into a single tree by adding a new root, and assign this sum of weights as the weight of the new tree.

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex. Call the frequency corresponding to each tree its weight.
- At each step, combine two trees whose sum of weights is a minimum into a single tree by adding a new root, and assign this sum of weights as the weight of the new tree.
- The algorithm is finished if a single tree is constructed.

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex. Call the frequency corresponding to each tree its weight.
- At each step, combine two trees whose sum of weights is a minimum into a single tree by adding a new root, and assign this sum of weights as the weight of the new tree.
- The algorithm is finished if a single tree is constructed.

Example 1. Construct a Huffman coding to encode the message "good morning".

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex. Call the frequency corresponding to each tree its weight.
- At each step, combine two trees whose sum of weights is a minimum into a single tree by adding a new root, and assign this sum of weights as the weight of the new tree.
- The algorithm is finished if a single tree is constructed.

Example 1. 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?

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex. Call the frequency corresponding to each tree its weight.
- At each step, combine two trees whose sum of weights is a minimum into a single tree by adding a new root, and assign this sum of weights as the weight of the new tree.
- The algorithm is finished if a single tree is constructed.

Example 1. 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?

Example 2. Construct a Huffman coding to encode the letters with their frequencies, and find the average number of bits used to encode a letter:

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex. Call the frequency corresponding to each tree its weight.
- At each step, combine two trees whose sum of weights is a minimum into a single tree by adding a new root, and assign this sum of weights as the weight of the new tree.
- The algorithm is finished if a single tree is constructed.

Example 1. 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?

Example 2. Construct a Huffman coding to encode the letters with their frequencies, and find the average number of bits used to encode a letter:

A: 0.08, B: 0.1, C: 0.12, D: 0.15, E: 0.2, F: 0.35.

Huffman Coding

- Write out all distinct letters of the message together with their frequencies. Consider each letter as a binary tree with only one vertex. Call the frequency corresponding to each tree its weight.
- At each step, combine two trees whose sum of weights is a minimum into a single tree by adding a new root, and assign this sum of weights as the weight of the new tree.
- The algorithm is finished if a single tree is constructed.

Example 1. 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?

Example 2. Construct a Huffman coding to encode the letters with their frequencies, and find the average number of bits used to encode a letter:

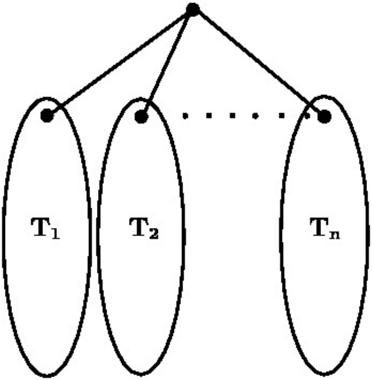
A: 0.08, B: 0.1, C: 0.12, D: 0.15, E: 0.2, F: 0.35.

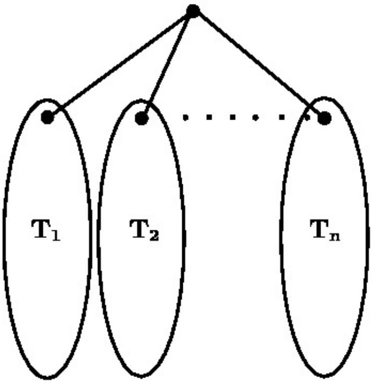
Game trees.

Game trees.

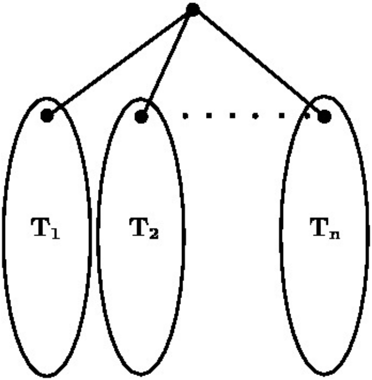
Read textbook!

10.3 Tree Traversal


T_1
T_2
T_n

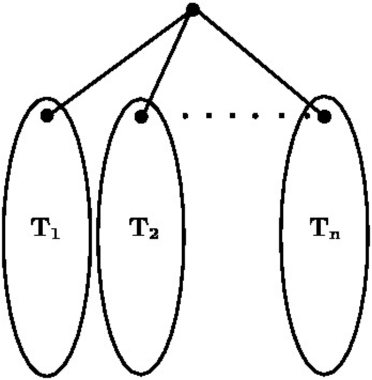


Preorder traversal



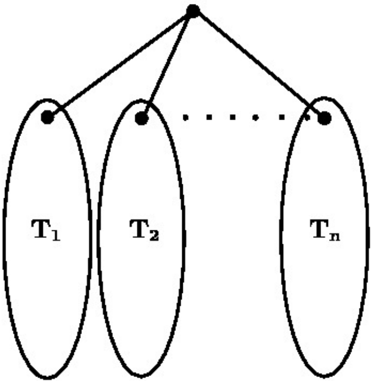
Preorder traversal

- Visit the root



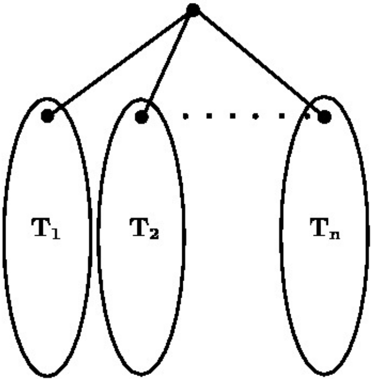
Preorder traversal

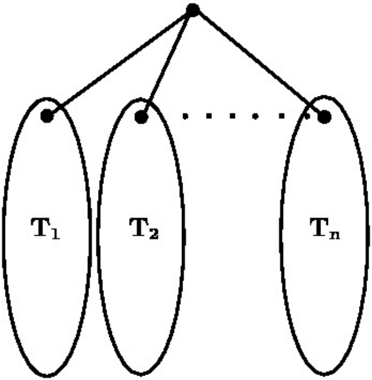
- Visit the root
- Traverse each of $T_1, T_2, \ldots, T_n$ in preorder.



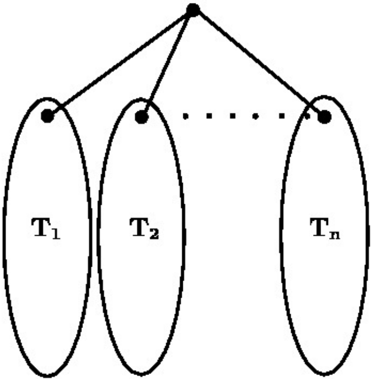
Preorder traversal

- Visit the root
- Traverse each of $T_1, T_2, \ldots, T_n$ in preorder.


T1
T2
Tn

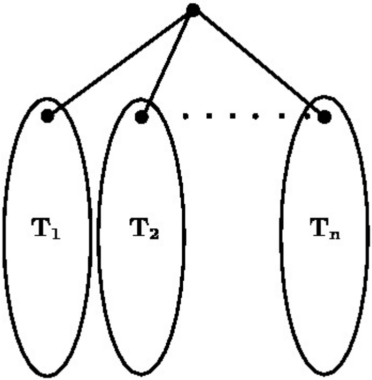


Inorder traversal



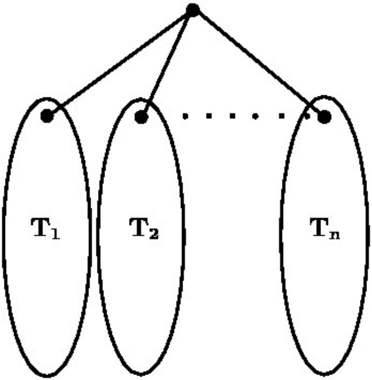
Inorder traversal

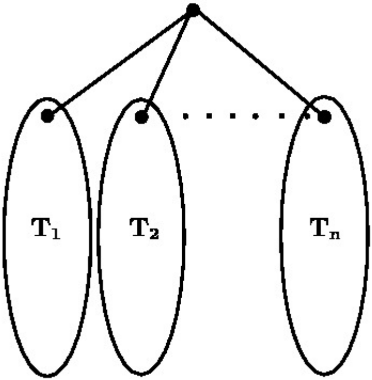
- Traverse T_1 in inorder.



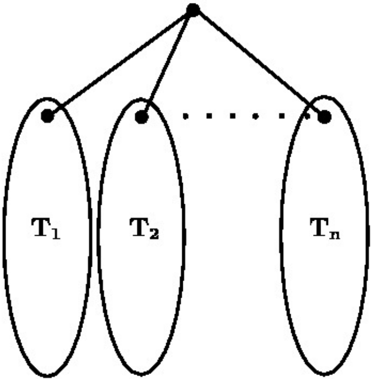
Inorder traversal

- Traverse T_1 in inorder.
- Visit the root.
- Traverse each of $T_2, \ldots, T_n$ in inorder.


T1
T2
Tn

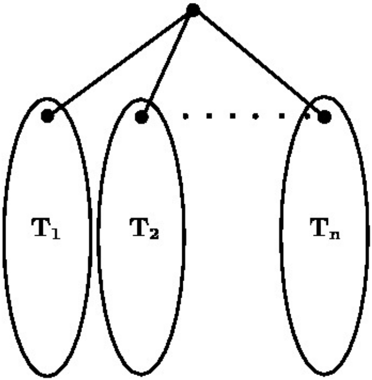


Postorder traversal



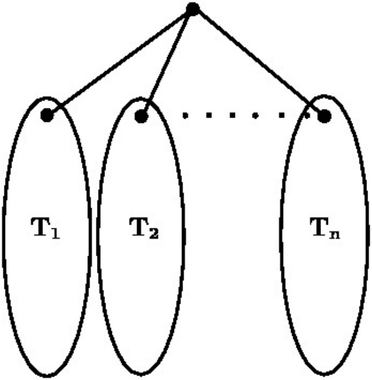
Postorder traversal

- Traverse each of $T_1, T_2, \ldots, T_n$ in postorder.
- Visit the root.



Postorder traversal

- Traverse each of $T_1, T_2, \ldots, T_n$ in postorder.
- Visit the root.



Postorder traversal

- Traverse each of $T_1, T_2, \ldots, T_n$ in postorder.
- Visit the root.

Infix, Prefix, and Postfix Notation

Infix, Prefix, and Postfix Notation

Consider the expression $(x + y)^2 + (x - 4)/3$.

Infix, Prefix, and Postfix Notation

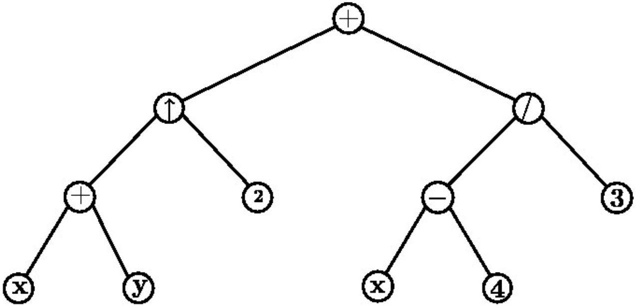
Consider the expression $(x + y)^2 + (x - 4)/3$.

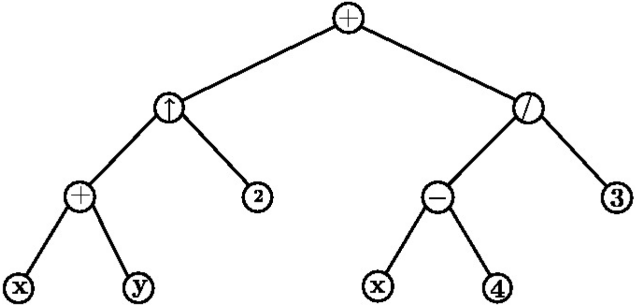
We can use a binary tree to represent this expression, where the internal vertices represent operations, and the leaves represent numbers or variables.

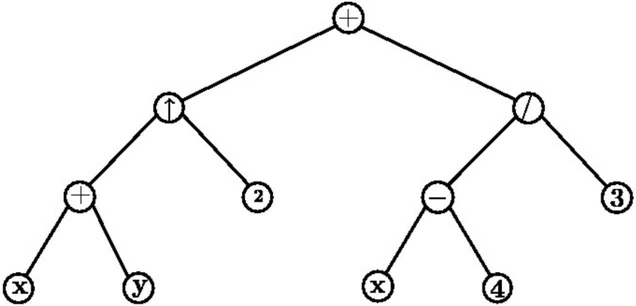
Infix, Prefix, and Postfix Notation

Consider the expression $(x+y)^2 + (x-4)/3$.

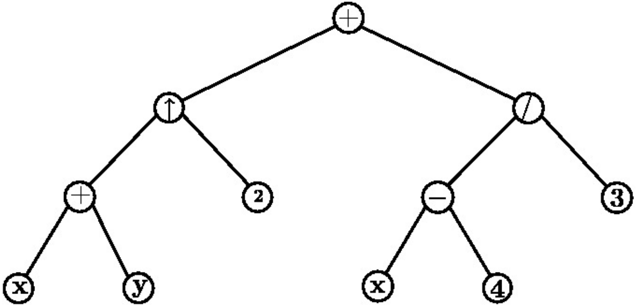
We can use a binary tree to represent this expression, where the internal vertices represent operations, and the leaves represent numbers or variables.



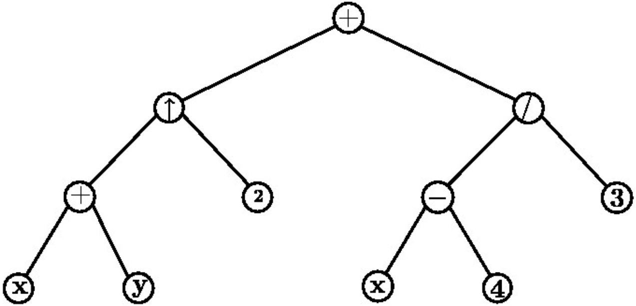

+
↑
/
+
2
−
3
x
y
x
4



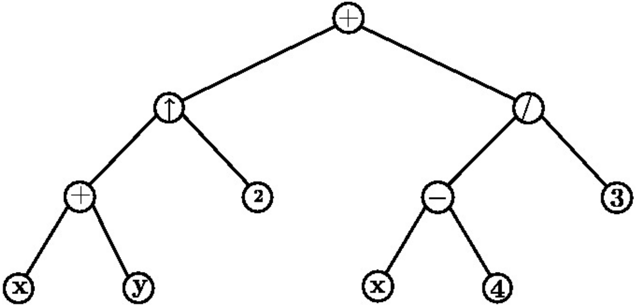
- Prefix notation (Polish notation):



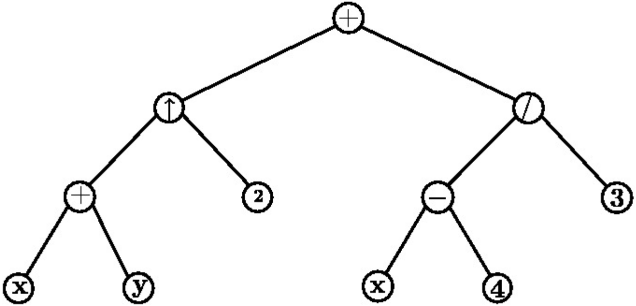
- Prefix notation (Polish notation):
$+ \uparrow + x\, y\, 2\, /\, -\, x\, 4\, 3$



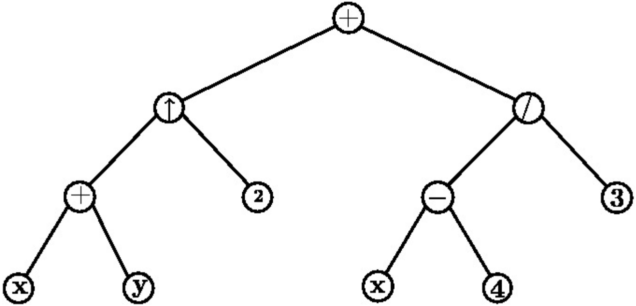
- Prefix notation (Polish notation):
  $+ \uparrow + x\, y\, 2 \,/\, - \; x\, 4\, 3$
- Postfix notation (reverse Polish notation):



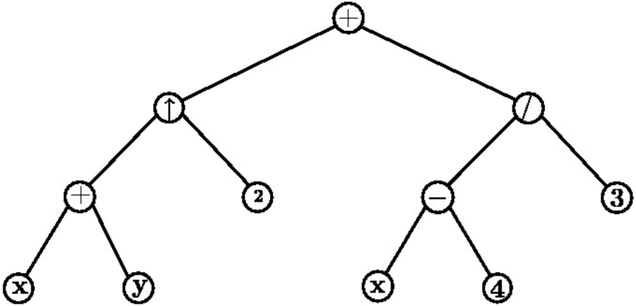
- Prefix notation (Polish notation):
  $+ \uparrow + x\, y\, 2 \,/\, - \, x\, 4\, 3$
- Postfix notation (reverse Polish notation):
  $x\, y\, +\, 2 \uparrow x\, 4\, -\, 3 \,/\, +$



- Prefix notation (Polish notation):
  $+ \uparrow + x\, y\, 2 \,/\, - \ x\, 4\, 3$
- Postfix notation (reverse Polish notation):
  $x\, y\ +\ 2 \uparrow x\, 4\ -\ 3 \,/+$
- Infix notation:



- Prefix notation (Polish notation):
  $+ \uparrow + x\, y\, 2 \,/ - x\, 4\, 3$
- Postfix notation (reverse Polish notation):
  $x\, y + 2 \uparrow x\, 4 - 3 \,/ +$
- Infix notation:
  $x + y \uparrow 2 + x - 4 \,/\, 3$



- Prefix notation (Polish notation):
  $+ \uparrow + x\,y\,2\,/\,-\,x\,4\,3$
- Postfix notation (reverse Polish notation):
  $x\,y\,+\,2\uparrow x\,4\,-\,3\,/\,+$
- Infix notation:
  $x + y \uparrow 2 + x - 4\,/\,3$

To evaluate the value of a form:

To evaluate the value of a form:

- For postfix form, work from left to right, carrying out operations whenever an operator follows two operands. After each operation is carried out, the result of this operation is a new operand.

To evaluate the value of a form:

- For postfix form, work from left to right, carrying out operations whenever an operator follows two operands. After each operation is carried out, the result of this operation is a new operand.
- For prefix form, follows the same procedure, but work from right to left instead.

To evaluate the value of a form:

- For postfix form, work from left to right, carrying out operations whenever an operator follows two operands. After each operation is carried out, the result of this operation is a new operand.
- For prefix form, follows the same procedure, but work from right to left instead.

Example.

To evaluate the value of a form:

- For postfix form, work from left to right, carrying out operations whenever an operator follows two operands. After each operation is carried out, the result of this operation is a new operand.
- For prefix form, follows the same procedure, but work from right to left instead.

Example.

(a) Find the value of the postfix expression:

$$7\,2\,3 * - 4 \uparrow 9\,3 / +$$

To evaluate the value of a form:

- For postfix form, work from left to right, carrying out operations whenever an operator follows two operands. After each operation is carried out, the result of this operation is a new operand.
- For prefix form, follows the same procedure, but work from right to left instead.

Example.

(a) Find the value of the postfix expression:

$$7\,2\,3\,*\,-\,4\,\uparrow\,9\,3\,/\,+$$

(b) Find the value of the prefix expression:

$$+\,-\,*\,2\,3\,5\,/\,\uparrow\,2\,3\,4$$

To evaluate the value of a form:

- For postfix form, work from left to right, carrying out operations whenever an operator follows two operands. After each operation is carried out, the result of this operation is a new operand.
- For prefix form, follows the same procedure, but work from right to left instead.

Example.

(a) Find the value of the postfix expression:

$$7\,2\,3 * - 4 \uparrow 9\,3 / +$$

(b) Find the value of the prefix expression:

$$+ - * 2\,3\,5 / \uparrow 2\,3\,4$$

and find its postfix expression.