# Trees



- In the single-linked and double-linked lists, the nodes are joined in a line.
- A *binary tree* is another kind of linked structure, but the nodes are not joined in a line; instead, they are connected in a series of branches like a tree.
- This structure allows the creation of more complex relationships between nodes.

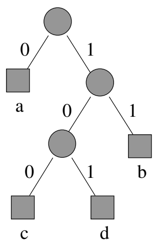


- The top of the tree, which consists of just one node, is called the *root*; the opposite of a real tree, just to make things confusing.
- In a binary tree, each node includes links to two other nodes (called the *left* and *right child nodes*).
- Each node in a binary tree (except the very bottom ones) is the root of some other nodes; that node and its descendants are called a *subtree*.
- A node at the very bottom of the tree has no children, and is called a *leaf* node.



## Uses of Trees

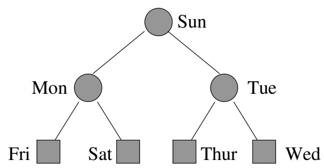
Code Tree






## Uses of Trees

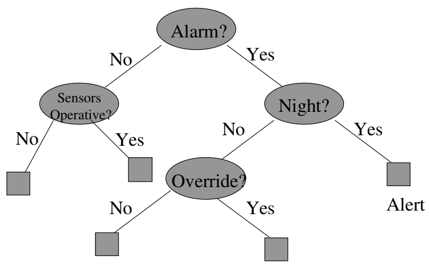
Binary Search Tree






## Uses of Trees

Decision Tree



## Binary Tree Terminology

- Let $T$ be a binary tree with root $u$
- Let $v$ be any node in $T$
- Let $u(1)$ and $u(2)$ be left & right subtrees of T
- If $v$ is the root of either $u(1)$ or $u(2)$, then we say $u$ is the parent of $v$, and that $v$ is the child of $u$
- If $w$ is also a child of $u$, and $w$ is distinct from $v$, we say that $v$ and $w$ are siblings.
- Also have grandparents and grandchildren



## Binary Tree Terminology

Given a binary tree T of n nodes, $n \geq 0$,

then $v$ is a descendent of $u$ if either

- $v$ is equal to $u$
  or
- $v$ is a child of some node $w$ and $w$ is a descendant of $u$.

We write $v\ desc_T\ u$.

v is a proper descendent of $u$ if $v$ is a descendant of $u$ and $v \neq u$.



## Binary Tree Terminology

Given a binary tree T of n nodes, $n \geq 0$,

then $v$ is a left descendent of $u$ if either

- $v$ is equal to $u$
  or
- $v$ is a left child of some node $w$ and $w$ is a left descendant of $u$.

We write $v\ ldesc_T\ u$.

Similarly we have $v\ rdesc_T\ u$.



## Binary Tree Terminology

- The height of $T$ is defined recursively as

  0 if $T$ is empty and

  $1 + max(height(T_1), height(T_2))$ otherwise, where $T_1$ and $T_2$ are the subtrees of the root.

- The height of a tree is the length of a longest chain of descendants



## Binary Tree Terminology

Levels of nodes

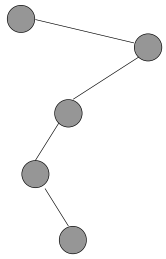
- The level of a node in a binary tree is computed as follows
- Number the root node 0
- Number every other node to be 1 more than its parent
- Then the number of a node $v$ is that node's level
- The level of $v$ is the number of branches on the path from to root to $v$



## Binary Tree Terminology

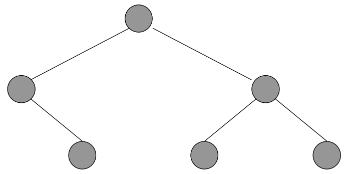
- Complete Binary Trees (Fat Trees): the leaf nodes appear on at most two adjacent levels
- Perfect Trees: complete trees having all their leaf nodes on one level
- Left-complete Trees: the leaf nodes on the lowest level is in the leftmost possible position.
- Skinny trees are the highest possible tree
- Complete trees are the lowest possible trees

## Skinny Tree



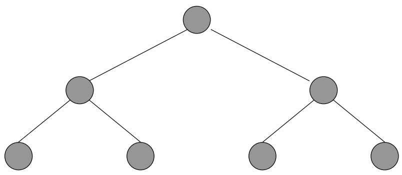


## Complete Tree



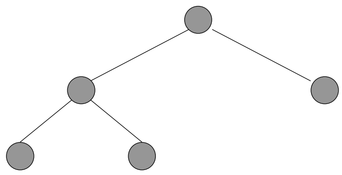


## Perfect Tree





## Left-Complete Tree





## Binary Tree Terminology

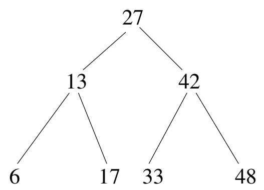
- A binary tree of height $h \geq 0$ has **size** at least $h$
- A binary tree of height at most $h \geq 0$ has **size** at most $2^h - 1$
- A binary tree of size $n \geq 0$ has **height** at most $n$
- A binary tree of size $n \geq 0$ has **height** at least $\lceil \log (n + 1) \rceil$



## Binary Search Trees

- A binary tree becomes a powerful search tool if its members are sorted properly.
- A *binary search tree* is a particular kind of binary tree in which the data item in each left child node precedes the item in its parent, and the data item in the right child node follows the item in its parent.
- You can create a binary search tree using dynamically allocated nodes linked by pointers.

## A Binary Search Tree






## Balanced Binary Trees

- The binary search tree is efficient only when it is fully populated, or *balanced*.
- When you add records to, or delete records from, the binary search tree, you must reorganise the tree so that each node has an equal number of nodes to each side.
- As long as you maintain this structure, you can search through the tree for a particular value just like you earlier conducted a binary search on a simple array.



- *A binary tree is a useful data structure when 2 way decisions must be made at each point in a process.*
- Constructing a binary search tree is also an efficient way to check for duplicate values.
- Traversing a binary tree is more complicated than traversing a linked list, because each node has two branches; this branching poses a problem for searching the tree. Recursion, however, can overcome this.



## Tree Traversals

- To perform a traversal of a data structure, we use a method of visiting every node in some predetermined order
- Traversals can be used
  - to test data structures for equality
  - to display a data structure
  - to construct a data structure of a give size
  - to copy a data structure



## Depth-First Traversals

- There are 3 depth-first traversals
  - inorder
  - postorder
  - preorder



## In Order Traversal

- traverse the left subtree
- process the value in the node
- traverse the right subtree

- Always examines the values in ascending order
- If a search program printed each value in step 2, it would produce the following list for the tree example used earlier :

  6, 13, 17, 27, 33, 42, 48

# Pre Order Traversal

- process the value in the node
- traverse the left subtree
- traverse the right subtree

- If the program printed the values as it encountered them, it would produce a list beginning with the tree's middle value. It would then print all of the lower values, and then all the higher values:

27, 13, 6, 17, 42, 33, 48



# Post Order Traversal

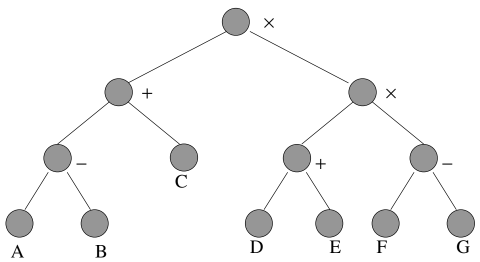
- traverse the left subtree
- traverse the right subtree
- process the value in the node

Printing these results would produce this list :

6, 17, 13, 33, 48, 42, 27



# Example: Expression Tree






# Depth-First Traversals

- Inorder traversal
  $A - B + C \times D + E \times F - G$

- Postorder traversal
  $A\ B - C + D\ E + F\ G - \times \times$

- Preorder traversal
  $\times + - A\ B\ C \times + D\ E - F\ G$



# Depth-First Traversals

- The parenthesised Inorder traversal

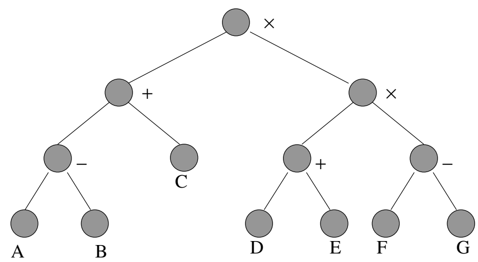
  $((A - B) + C) \times ((D + E) \times (F - G))$

  This is the infix expression corresponding to the expression tree

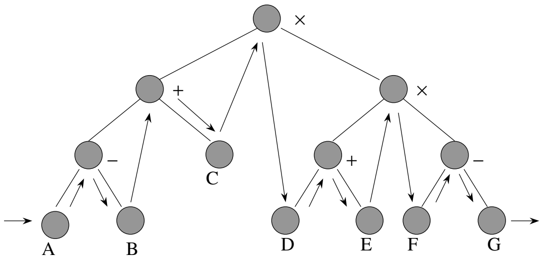
- Postorder traversal gives a postfix expression
- Preorder traversal gives a prefix expression



# Example: Inorder Traversal

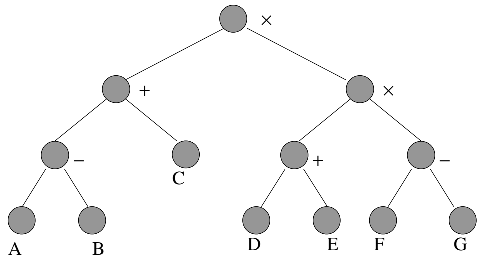


## Example: Inorder Traversal



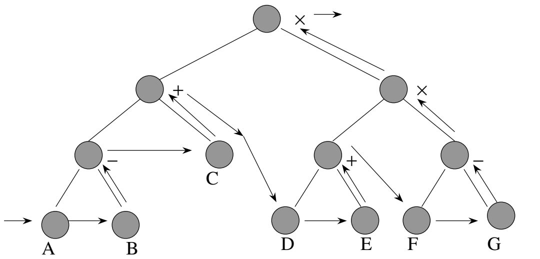



## Example: Postorder Traversal



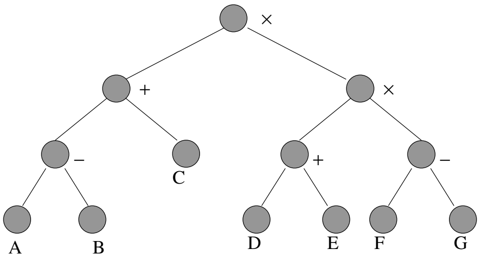



## Example: Postorder Traversal



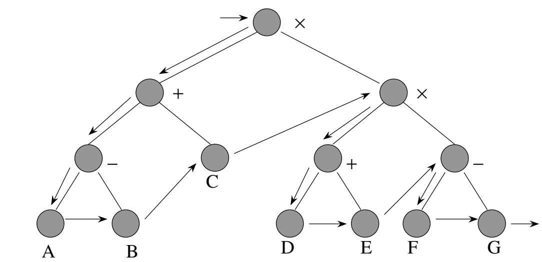



## Example: Preorder Traversal






## Example: Preorder Traversal






## Multiway Trees

- > 2 children per node
- Useful when constructing large scale search trees in which insertions and deletions are necessary , but in which the primary store of the computer is not large enough or is to costly to be used for long time storage
- Assume that the nodes of the tree will be stored in secondary storage. Call these nodes pages.

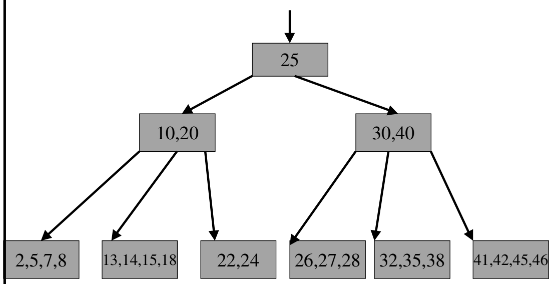
# B trees

B Trees are Multiway trees with the following characteristics: n is the *Order* of the B tree.

- Every page contains at most 2n items (keys)
- Every page except the root contains at least n items
- Every page is either
  - a leaf page i.e. has no descendants
  - has m+1 descendants where m is the number of keys
- All leaf pages appear at the same level



# A B-Tree of order 2



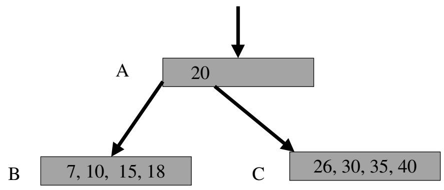



# Insertion

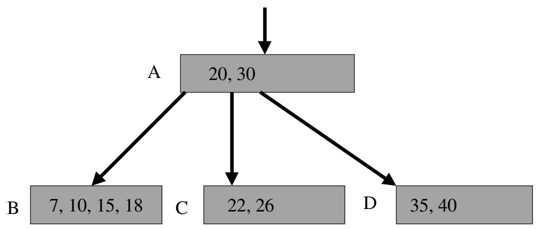
- If an item is to be inserted into a page which has m <2n items, the insertion process remains constrained to that page.
- It is only insertion into an already full page that has consequences upon the tree structure and may cause the allocation of new pages
  - the full page C must be split into two pages (C & D)
  - the keys from the full page are equally distributed between C and D
  - The middle key is moved up a level to C's ancestor



# Insert 22








- Insertion of an item into an ancestor page may cause that page to overflow, thereby causing the split to propagate.
- In extreme cases it may propagate up to the root
- This is the only way a B tree may increase in height
- B trees grow from the leaf nodes to the root

## Exercise.

Develop a B tree of order 2 to represent the following sequence of values:

20;

40, 10, 30, 15;

35, 7, 26, 18, 22;

5;

42, 13, 46, 27, 8, 32;

38, 24, 45, 25;



## Deletion

- If the item to be deleted is on a leaf page the removal algorithm is straightforward
- If the item to be deleted is not on a leaf page, it must be replaced by one of the two lexicographically adjacent items, which happen to be leaf pages and can easily be deleted
  - to find the adjacent items we descend along the rightmost pointers down to the leaf page P and replace the item to be deleted by the rightmost element of that leaf page



## Removal ctd.

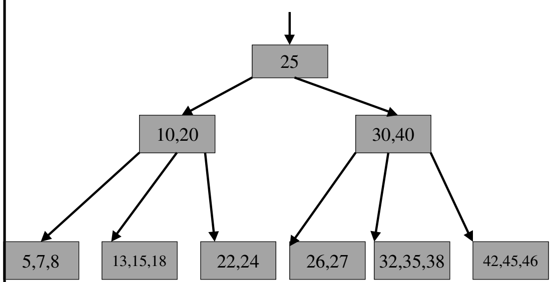
- When a key is removed from a leaf node we must check to see if the number of items in the page is less than n (which would violate the B-tree rules)
- This condition is called underflow
- To solve we borrow an item from one of the neighbouring pages Q
- The usual strategy is to evenly distribute the items on P and Q.
- This is called balancing.



- If there is no page to be moved from Q (as Q has reached its minimal size) then the values in P and Q may be merged, adding the middle item from the ancestor page of P and Q and tehn entirely dispose of page Q
- This is the inverse to page splitting.
- Once again removal of the middle value in the ancestor page may cause the size to drop below n and balancing or merging will be required at the next level.
- This is the only way a B tree may shrink in height.



## A B-Tree of order 2






## Deletion

- Delete Keys in the following order:

25, 45, 24;

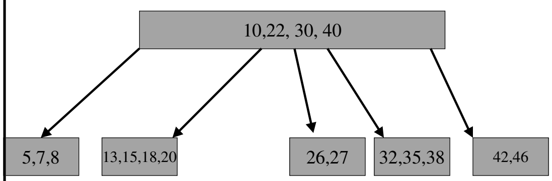
38, 32;

8, 27, 46, 13, 42;

5, 22, 18, 26;

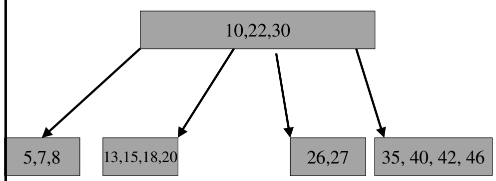
7, 35, 15;

## A B-Tree of order 2



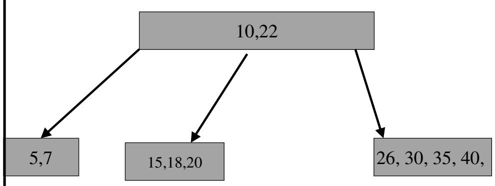



## A B-Tree of order 2






## A B-Tree of order 2






## A B-Tree of order 2