PRINCIPLES OF DATA ORGANISATION

Hierarchical Indexing

MOTIVATION

- Key, pointer pairs ~ index
- Hashing drawbacks
  - No easy range queries
- Alternative: search trees (binary tree, a-b tree)

DRAWBACKS OF INDEX(-SEQUENTIAL) ORGANIZATION

- We have a primary file and an index built on top of it
  - OK for static data (OLAP – online analytical processing)
- When inserting a record at the beginning of the file, the whole index needs to be rebuilt
  - Typical for OLTP – online transaction processing
  - Overflow handling slows down efficiency
- Reorganisation can take a lot of time, especially for large tables

TREE INDEXES

- Most common dynamic indexing structure for external memory
- When inserting/deleting into/from the primary file, the indexing structure(s) residing in the secondary file is modified to accommodate the new key
- The modification of a tree is implemented by splitting/merging nodes
- Used in: DBMSs, NTFS, …

TREE BASICS

- An undirected graph without cycles
- Rooted trees ~ one node designated as the root ~ orientation
  - Nodes are in a parent-child relation
- Every parent has a finite set of descendants (children nodes)
  - There is an upper boundary (we need to implement the tree)
- Every child has exactly one parent
  - Root is the only node without parent ~ root of the hierarchy
- Leaves are nodes without children
  - Bottom level
- Inner nodes are nodes with children

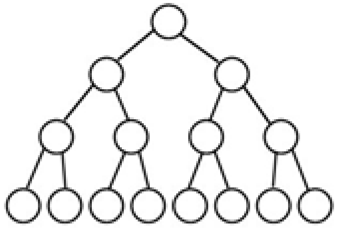
TREE BASICS

- Tree arity/degree – maximum number of children of any node
  - Binary search tree: 2
- Node depth – the length of the path (number of edges) from the root node
- Tree depth – maximum of node depths
- Tree level – set of nodes with the same depth (distance from root)
  - Level 0 … root
- Level width – number of nodes at a given level
- Node height – number of edges on the longest downward path from a node to a leaf
- Tree height – the height of the root node

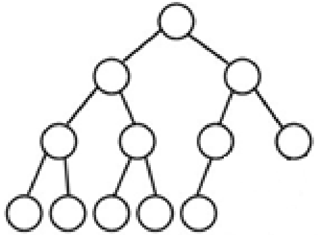
TREE BASICS

- **Balanced tree**
  A tree whose subtrees differ in height by no more than one and the subtrees are balanced as well
- **Unsorted tree**
  A tree where the descendants of a node are not sorted at all
- **Sorted tree**
  Children of a node are sorted based on a given key

BINARY SEARCH TREE



- Sorted tree
- Each inner node contains at most two child nodes
- Perfect binary tree ~ every non-leaf node has two child nodes
- Complete binary tree ~ full tree except for the last level where nodes are as far left as possible



- Perfect tree
  - #nodes $= \sum 2^i = 2^{h+1} - 1$
  - #leaf nodes $= 2^h$
    - How big the tree is going to be

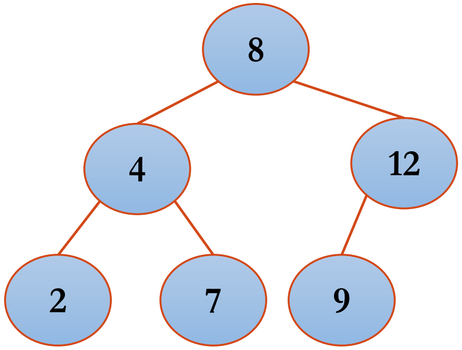
BINARY SEARCH TREE (BST)

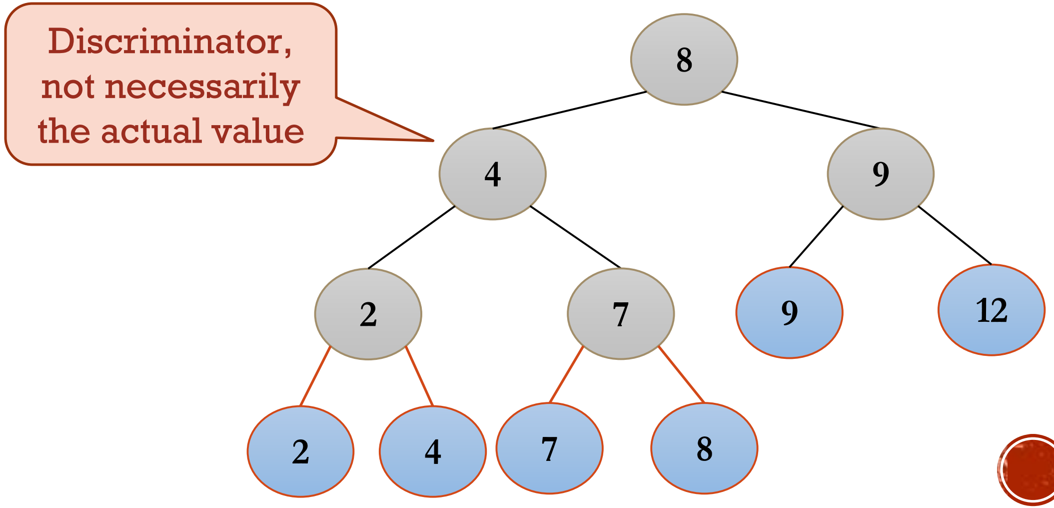
Non-Redundant

- Records stored in (addressed from) both inner and leaf nodes.

Redundant BST

- Records stored in (addressed from) the leaves
- Structure of inner and leaf nodes differ





BINARY SEARCH TREE – USAGE

- Expression trees
  - leaves = variables
  - inner nodes = operands
- Huffman coding
  - leaves = data
  - coding along a branch leading to give leaf = leaf's binary representation
- Query optimizer in DataBase Management Systems
  - Query can be represented by an algebraic expression which can be in turn represented by a binary tree
- Binary trees are not suitable for secondary memory because of their height

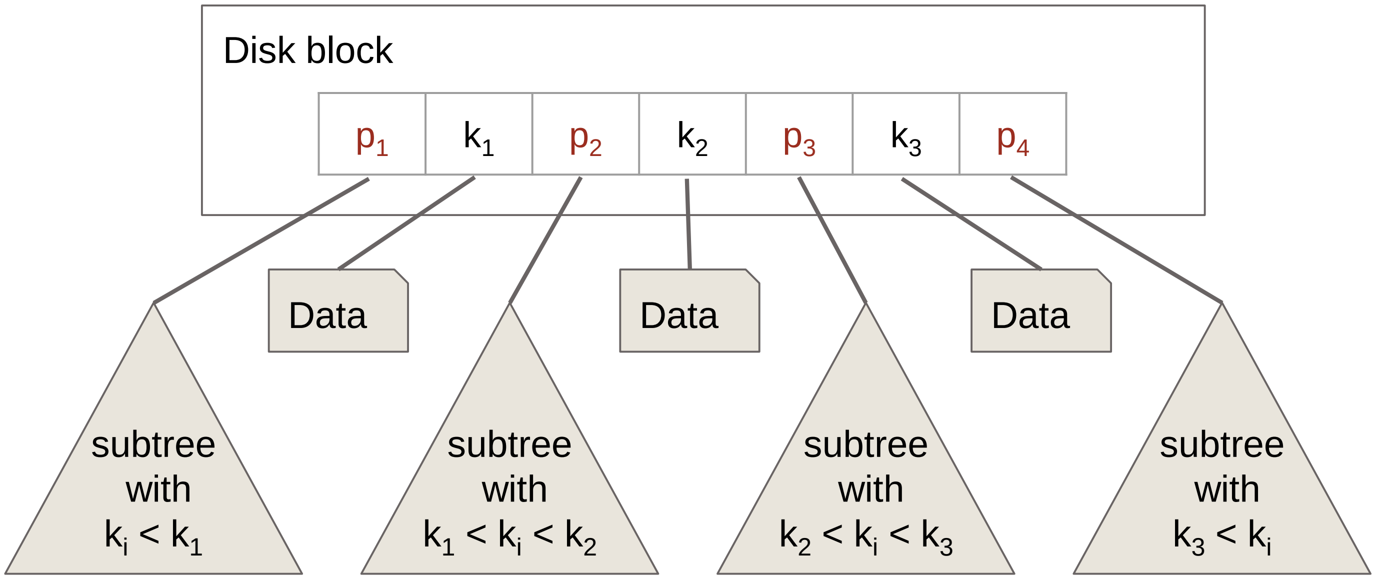
  $\log_2 1.000 \sim 10$ $\log_2 1.000.000 \sim 20$

M-ARY TREES

- Binary trees: $m = 2$
- Increasing arity leads to decreasing the tree height
- Use $\boldsymbol{m - 1}$ discriminators in each node
- Every subtree contains records with keys restricted by a pair of discriminators between which the subtree is rooted
- The left/right most subtree of a node contains values lower/higher then every discriminator in the node

M-ARY TREES

- M-ary trees $m = 4$
  - In reality much higher
- Data could be stored directly in the node as well. But it is not usual in real-world database environments.



CHARACTERISTICS OF M-ARY TREES

Maximum number of nodes

- $m^0 + m^1 + m^2 + \dots + m^h$
  - h - height
- $\sum_0^h m^i$
- $(m^{h+1} - 1) / (m - 1)$

Maximum number of records

- Every node contains up *to m – 1* records
- *# nodes * m – 1 = $m^{h+1} - 1$*
- *Examples:*
  - For m = 3, h = 3 we get 80
  - For m = 100, h = 3 we get ~ 100.000.000 -1

Minimum height

- $h = \lceil \log_m n \rceil$

Maximum height

- h ~ n / m

CHARACTERISTICS OF M-ARY TREES

- Height of the tree corresponds to the minimum/maximum number of disk operations needed to fetch a record.
- $O(\log_m n) < \ \ldots < \ O(n/m)$
- The challenge is to keep the complexity logarithmic, that is to keep the tree more or less balanced