

CSC 212: Data Structures and Abstractions

Balanced trees (part 1)

Prof. Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2025



Practice

- Assume a dictionary has n keys, and a book has m words
 - What is the time complexity of identifying which words from the book do NOT appear in the dictionary?
 - dictionary is represented as a BST and assume that $h = O(\log n)$
 - book is represented as an array (vector) of strings, where each string is a word

2

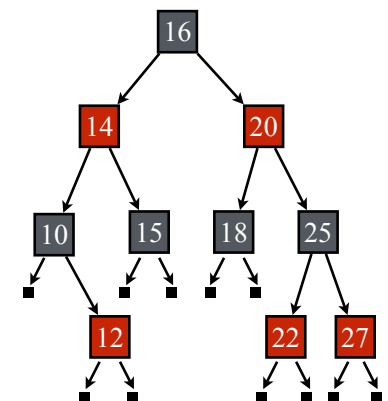
Balanced search trees

- Balanced search trees** are a type of search trees that maintain specific structural invariants to ensure their height h remains $O(\log n)$ for n nodes
 - among the most important data structures in computer science
 - widely implemented in standard libraries:
 - Java**: TreeSet and TreeMap,
 - C++**: `std::set` and `std::map`
 - Python**: no built-in implementation, but available through third-party libraries
- Examples of balanced trees:
 - AVL trees, **Red-Black trees**, B-trees, Treaps, etc.

3

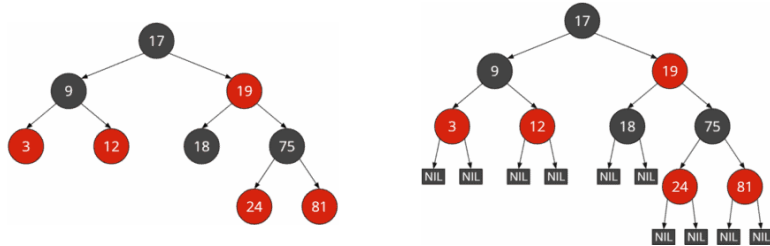
Red-black trees

- Red-black trees are BSTs that maintain near-perfect balance by enforcing the following properties on the nodes:
 - every node is colored **red** or **black**
 - the root is always **black**
 - red** nodes cannot have **red** children (no two consecutive red nodes)
 - null** nodes are considered **black**
 - every root-to-null path must have the same number of **black** nodes



4

Examples



Red-black tree with implicit NIL leaves

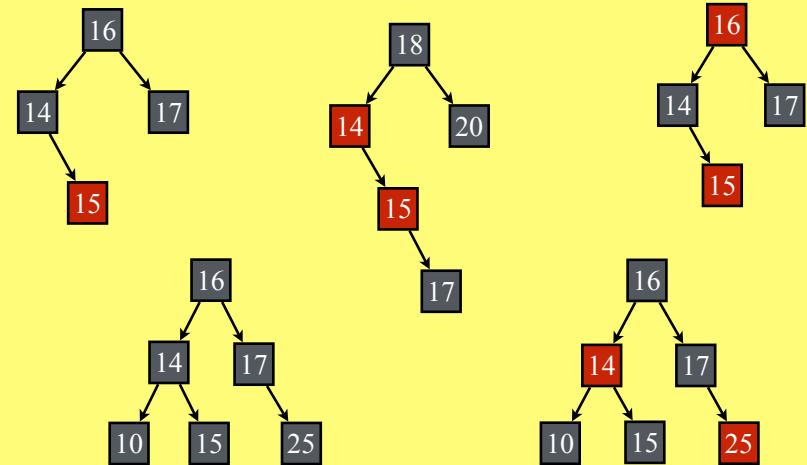
Red-black tree with explicit NIL leaves

<https://www.happycoders.eu/algorithms/red-black-tree-java/>

5

Practice

Are these valid red-black trees? — (null nodes not shown)



6

Height of red-black trees

- A red-black tree with n nodes has height $h = O(\log n)$
 - ✓ after an insertion or deletion, the tree may temporarily violate one or more red-black properties
 - ✓ these violations are efficiently corrected through:
 - rotations (left or right) and recoloring of nodes
- Equivalence to **2-3-4 Trees**
 - ✓ red-black trees are conceptually equivalent to 2-3-4 trees (B-trees of order 4)
 - ✓ this correspondence provides intuition for:
 - how rebalancing maintains logarithmic height
 - why red-black tree operations run in $O(\log n)$ time

7

2-3-4 Trees (interlude)

Multi-way search trees

• A **multi-way search tree** is a generalization of a BST in which:

- ✓ each node can store multiple keys (instead of just one)
- ✓ each node can have more than two children

• Properties

- ✓ the keys within each node are stored in **sorted** (increasing) order
- ✓ for a node containing keys $[k_1, k_2, \dots, k_m]$ and child subtrees $[T_0, T_1, \dots, T_m]$:
 - all keys in T_0 are less than k_1
 - all keys in T_i (for $0 < i < m$) are between k_i and k_{i+1}
 - all keys in T_m are greater than k_m

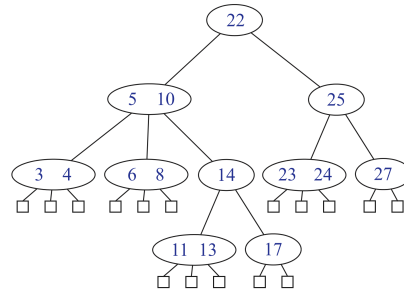


Image credit: Data Structures and Algorithms in C++ 2e

9

Example of a multi-way search tree

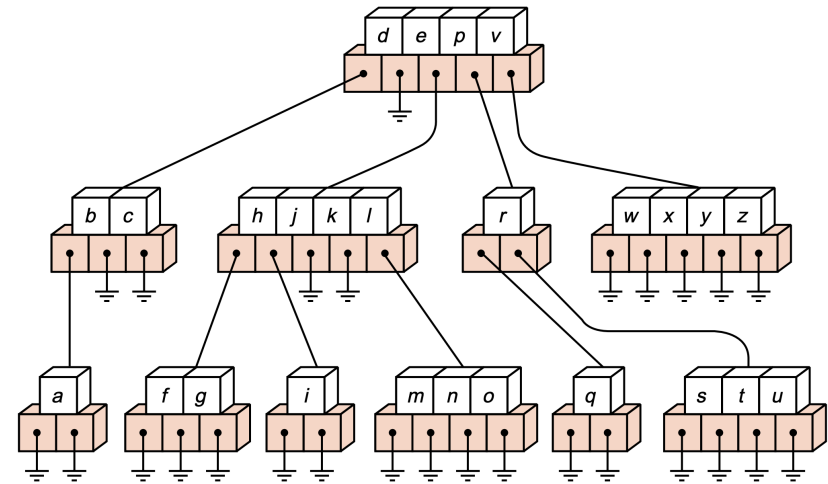
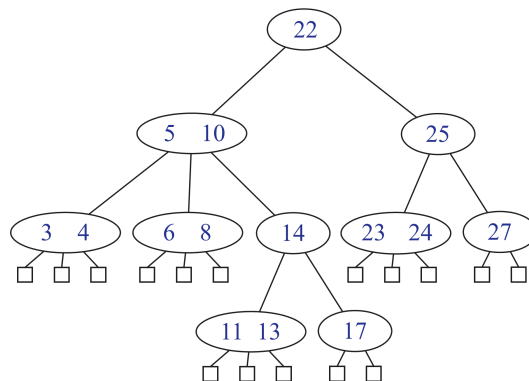


Image credit: Data Structures and Program Design In C++, Kruse and Ryba

10

Search on a multi-way search tree

• Perform **search** for 12, 17, 24, and 50 on the following tree



Assuming d denotes the maximum number of keys of any node of T , and h denotes the height of T . What is the cost of search?

Image credit: Data Structures and Algorithms in C++ 2e

11

Balanced multi-way search trees

• A **balanced multi-way search tree** is a multi-way search tree that:

- ✓ limits each node to a fixed maximum number of children
- ✓ keeps all leaf nodes at the same depth, ensuring the tree remains **perfectly balanced**

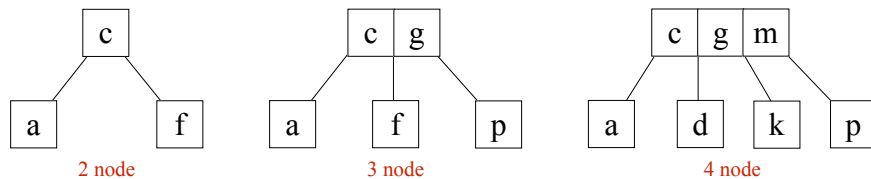
• A **B-tree** is a specific type of balanced multi-way search tree

- ✓ in a B-tree of **order m** (maximum number of children), every node, except the root, must have between $\lceil m/2 \rceil$ and m children
 - the term “order” can vary slightly between sources — some define it as the maximum number of keys, others as the maximum number of children
- ✓ B-trees are heavily used in databases, filesystems, and storage systems because they minimize disk I/O by storing many keys per node (typical orders: 1024, 2048, 4096, ...)

12

2-3-4 tree

- A 2-3-4 tree (also called a 2-4 tree) is a B-tree of order 4
 - ✓ each node can have 2, 3, or 4 children
 - ✓ all nodes (except the root, which can be empty) must have at least 1 key and at most 3 keys



13

Insertion (2-3-4 tree)

- Steps
 - ✓ **start at the root** and traverse downward to find the correct **leaf** for insertion
 - ✓ if the leaf has fewer than 3 keys
 - insert the new key into the node in sorted order
 - ✓ if the leaf already has 3 keys
 - temporarily insert the new key (so it contains 4 keys)
 - split the node into two nodes by promoting the middle key to the parent node and forming two new child nodes with the remaining keys
 - if the parent now has more than 3 keys, repeat this splitting process upward until the root if necessary
- Tree remains balanced after each insertion
 - ✓ all leaf nodes are at the same level

14

Practice

- Insert the following sequence into a 2-3-4 tree
 - ✓ 15, 10, 25, 5, 1, 30, 45, 60, 100, 70, 80, 40, 35, 90

15

Practice

- What is the max h of a 2-3-4 tree with n nodes?
 - ✓ to maximize the height, we want to minimize the number of keys per node (**instance of a worst-case**)
 - ✓ draw an example tree and express h in terms of n
- What is the cost of search and insert on a 2-3-4 tree?
 - ✓ worst-case scenario

16

Analysis

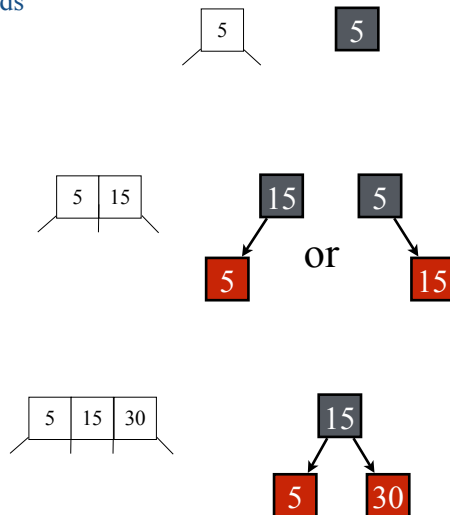
- The cost of operations in a B-tree of order b is $O(b \log_b n)$
 - ✓ insert, search, remove
 - ✓ small values of b make this cost optimal
- In practice ...
 - ✓ B-trees are widely used in databases and file systems to manage large amounts of data efficiently
 - ✓ useful for systems that read and write large blocks of data
 - B-trees can minimize the number of disk accesses required (much larger order values)

17

2-3-4 trees and red-black trees

Red-black trees \Leftrightarrow 2-3-4 trees

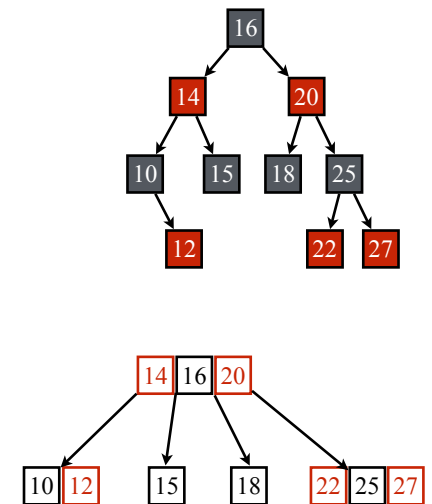
- A 2-node in a 2-3-4 tree corresponds to a **black** node in a red-black tree
- A 3-node corresponds to a **black** node with one **red** child
- A 4-node corresponds to a **black** node with two **red** children



19

Red-black trees \Leftrightarrow 2-3-4 trees

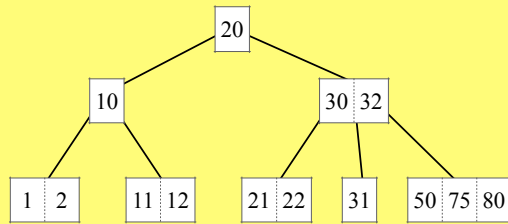
- Red-black trees are **isomorphic** to 2-3-4 trees
 - ✓ the number of black nodes on any *root-to-null* path corresponds to the number of levels of the 2-3-4 tree
 - ✓ every red-black tree can be transformed into an equivalent 2-3-4 tree and vice versa
 - ✓ the relationship is not bijective
 - a 3-node in a 2-3-4 tree can be represented in two ways in a red-black tree (leaning left or right)
 - each red-black tree corresponds to exactly one 2-3-4 tree (but not vice versa)



20

Practice

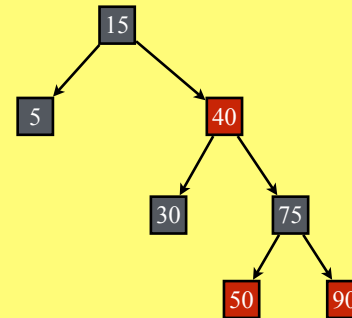
- Draw the red-black tree that corresponds to the following 2-3-4 tree



21

Practice

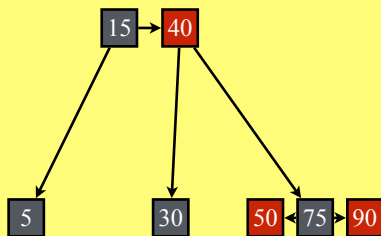
- Draw the 2-3-4 tree that corresponds to the following red-black tree



22

Practice

- Draw the 2-3-4 tree that corresponds to the following red-black tree



23