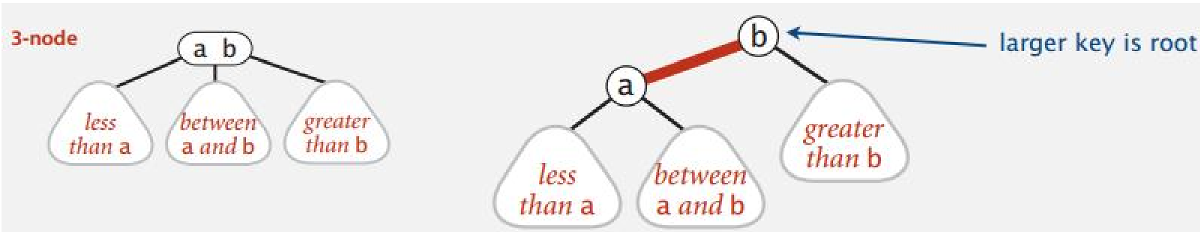
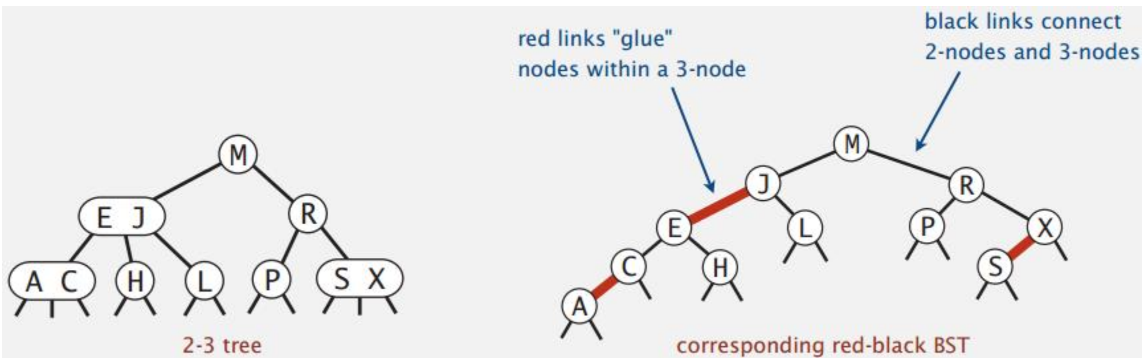
(Lecture xx) Red-Black Trees (Optional)

Left-leaning red-black BSTs (Guibas-Sedgewick 1979 and Sedgewick 2007): **LLRB**

1. Represent **2–3** tree as a **BST**.
2. Use "internal" left-leaning links as "glue" for **3–nodes**.



Example:



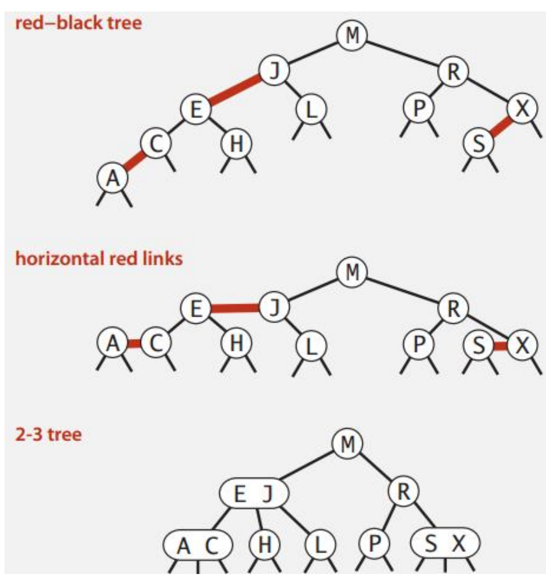
An equivalent definition:

A BST such that:

- No node has two red links connected to it.
- Every path from root to null link has the same number of black links.
- Red links lean left.

"perfect black balance"

Key property. 1–1 correspondence between **2–3** and **LLRB**.



To be continue.

(Lecture 21) B-Trees

An **M**-ary search tree allows **M**-way branching.

As branching increases, the depth decreases.

B-trees (Bayer-McCreight, 1972)

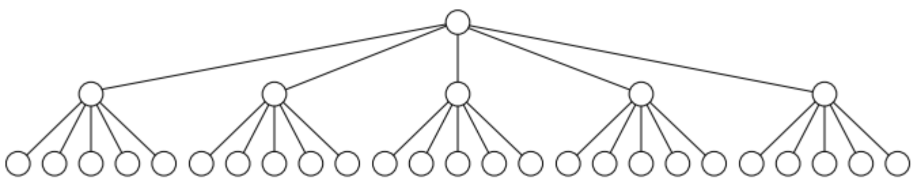
B-tree. Generalize 2-3 trees by allowing up to $M - 1$ key-link pairs per node.

- At least 2 key-link pairs at root.
- At least $M / 2$ key-link pairs in other nodes.
- External nodes contain client keys.
- Internal nodes contain copies of keys to guide search.

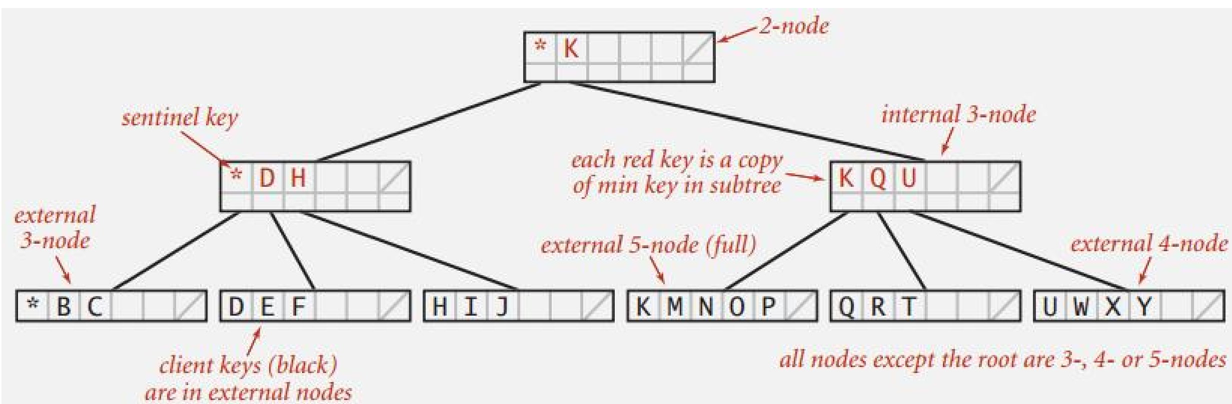
(choose M as large as possible so that M links fit in a page, e.g., M = 1024)

Nodes **must** be half full to guarantee that the tree does not degenerate into a simple binary tree.

Example: A 5-ary tree of 31 nodes has only three levels:



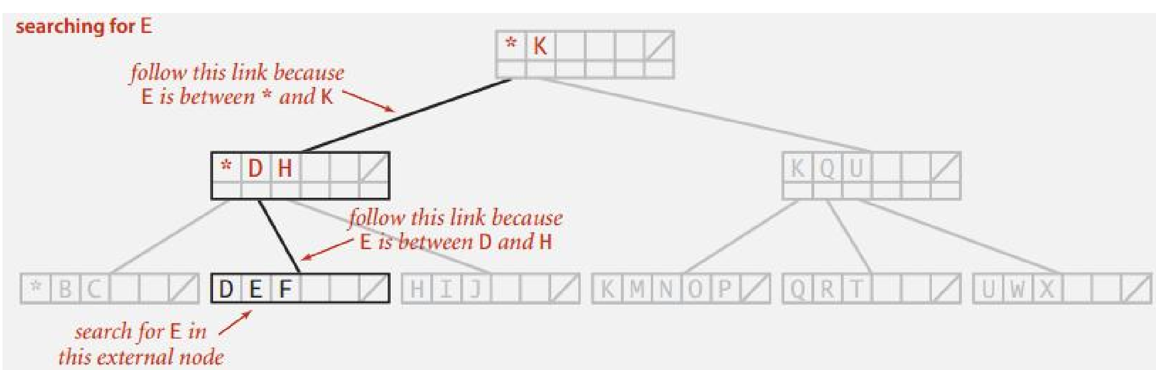
Example:



Anatomy of a B-tree set (M = 6)

Searching in a B-tree

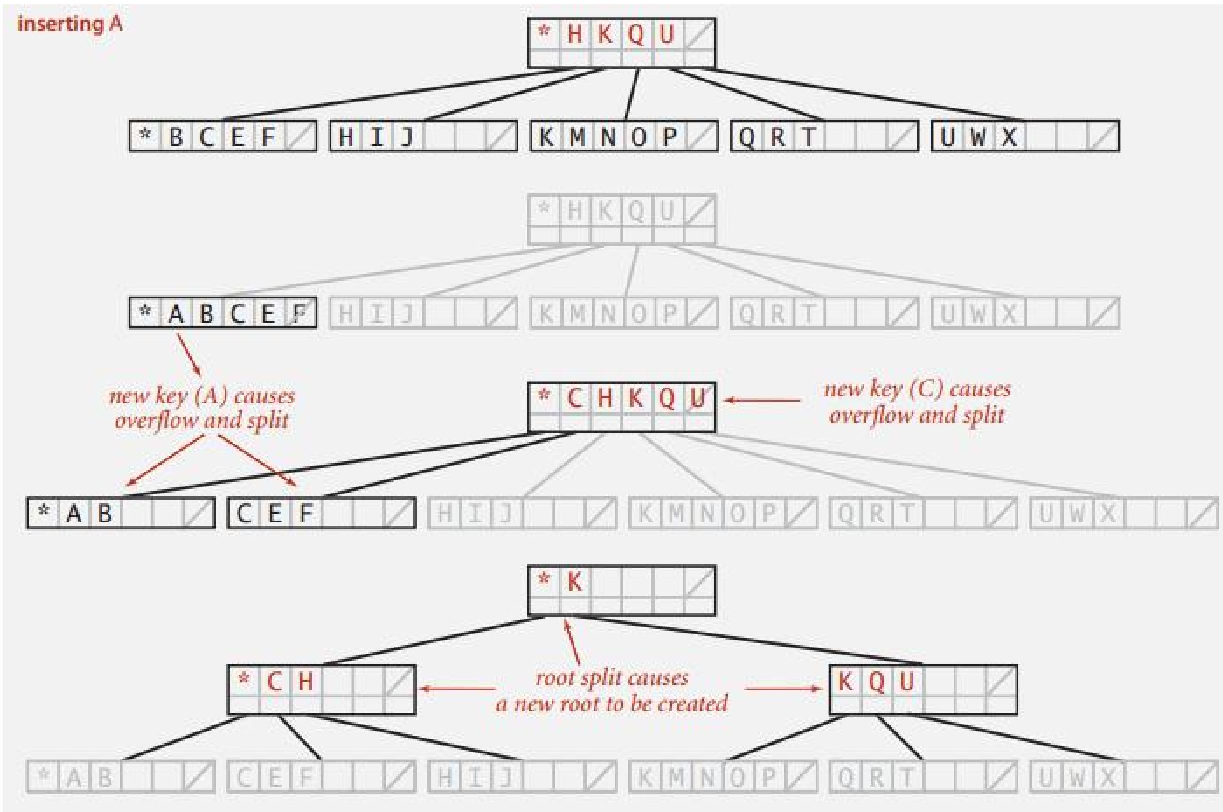
- Start at root.
- Find interval for search key and take corresponding link.
- Search terminates in external node.



Searching in a B-tree set (M = 6)

Insertion in a B-tree

- Search for new key.
- Insert at bottom.
- Split nodes with M key-link pairs on the way up the tree.



Inserting a new key into a B-tree set

Balance in B-tree

Proposition. A search or an insertion in a B-tree of order M with N keys requires between $\log_{M-1} N$ and $\log_{M/2} N$ probes.

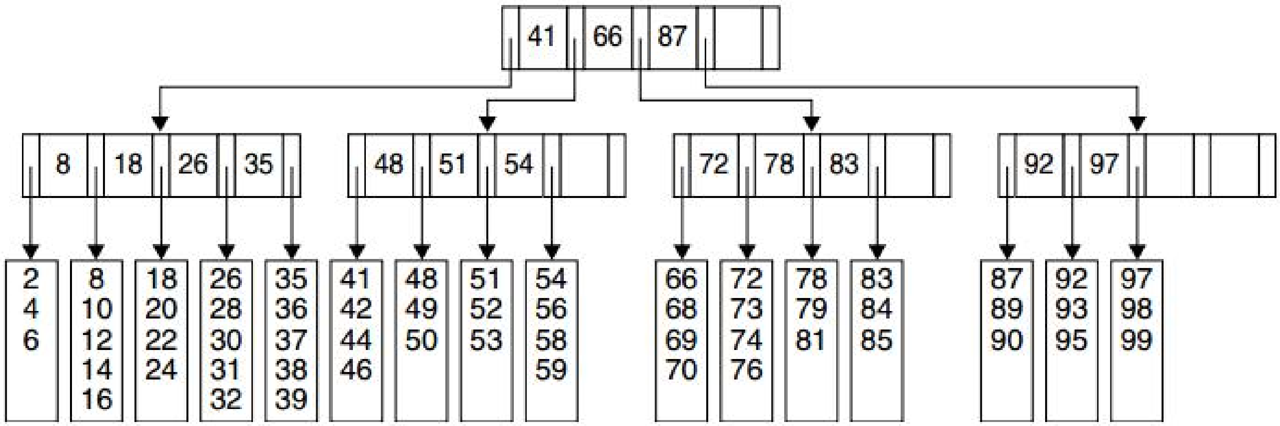
Pf. All internal nodes (besides root) have between $M / 2$ and $M - 1$ links.

In practice. Number of probes is at most 4. ← $M = 1024$; $N = 62$ billion, $\log_{M/2} N \leq 4$

Optimization. Always keep root page in memory.

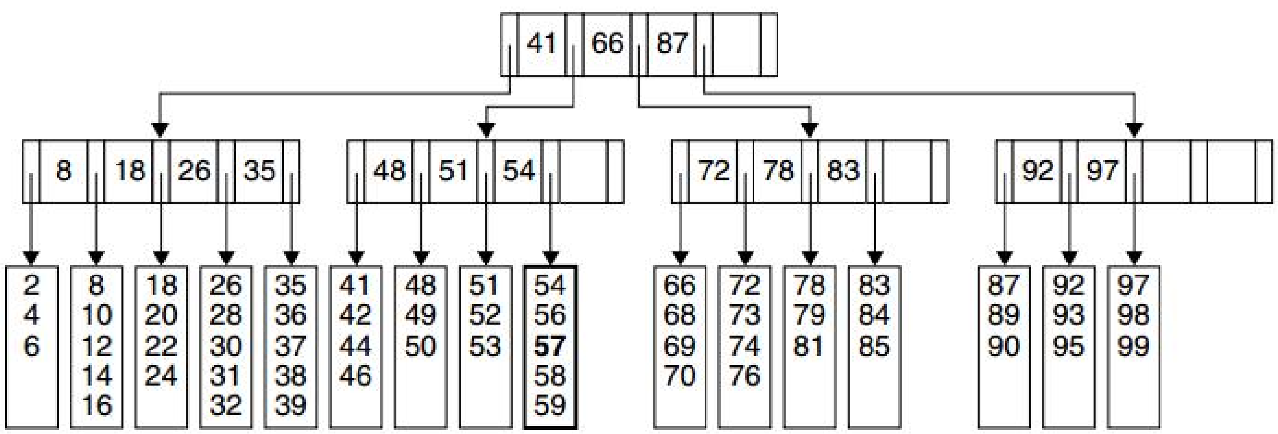
The B-tree is the most popular data structure for disk bound searching.

Example: A B-tree of order **5**



Insertion: insert 57

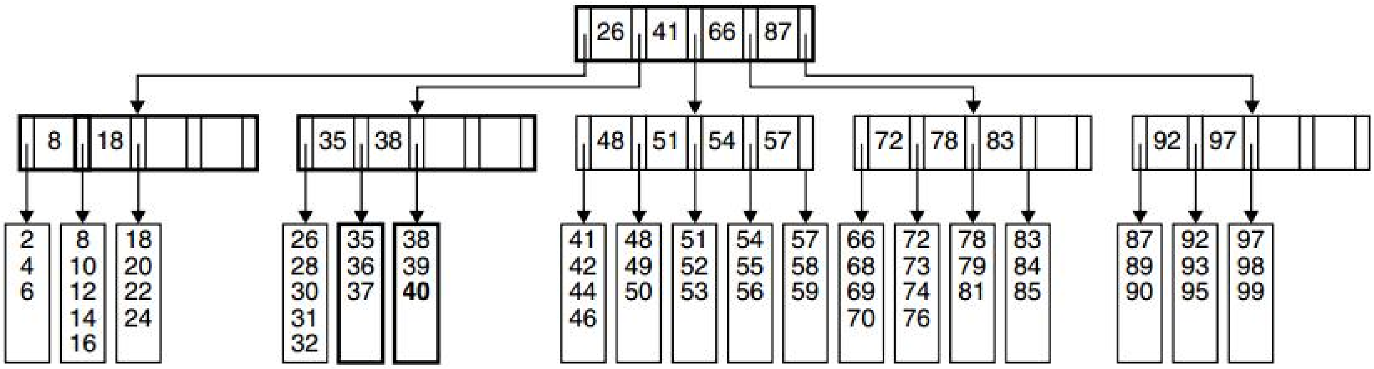
- If the leaf contains room for a new item, we insert it and are done.
- If the leaf is full, we can insert a new item by splitting the leaf and forming two half-empty nodes.



The B-tree after insertion of 57

Insertion: insert 40

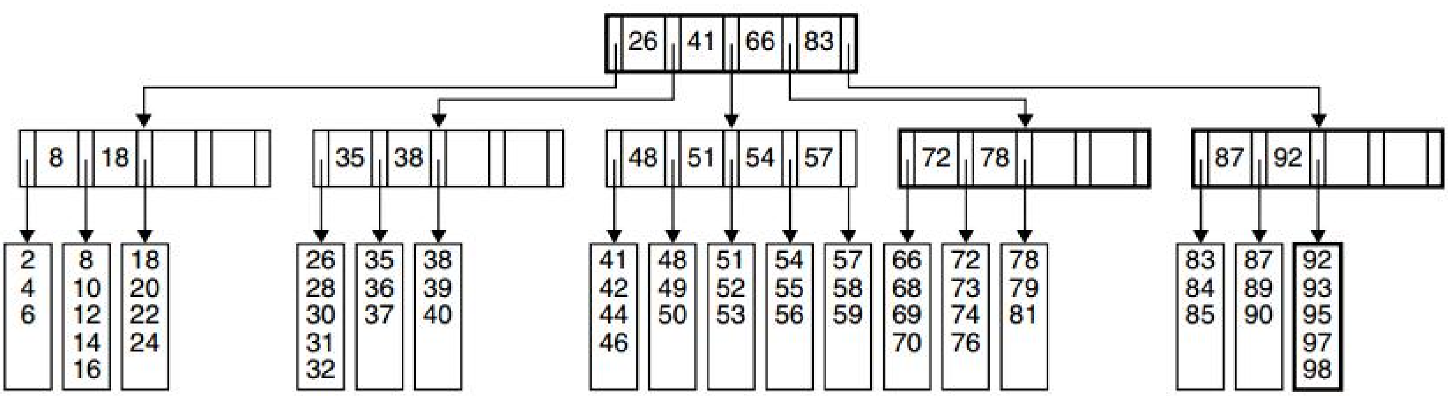
- Node splitting creates an extra child for the leaf's parent.
- If the parent already has a full number of children, we split the parent.
- We may have to continue splitting all the way up the tree (though this possibility is unlikely).
- In the worst case, we split the root, creating a new root with two children.



Insertion of **40** causes a split into two leaves and then a split of the parent node.

Deletion works in reverse: **remove 99:**

- If a leaf loses a child, it may need to combine with another leaf.
- Combining of nodes may continue all the way up the tree, though this possibility is unlikely.
- In the worst case, the root loses one of its two children. Then we delete the root and use the other child as the new root.



The B-tree after deletion of **99** from the tree