Data Structures

B-trees and multi-way trees

The notes for the lesson “B-trees and multi-way trees”: chapter 4.9, with their figures, worked examples, exam traps, quizzes and exam question bank, in one file.

1 chapter | Chapter 4.9

About these notes

These are the notes for the University free n easy video lesson **B-trees and multi-way trees** on Data Structures: chapter 4.9, with every figure, worked example, exam trap and quiz, and the long-answer exam questions on these chapters.

4.9 B-trees and multi-way search trees

When the data lives on a disk or SSD, what costs time is not a comparison but a block read: fetching 4 KB takes as long as thousands of comparisons in memory, and a binary tree with a million keys needs about 20 block reads per search. A multi-way search tree puts many keys and many children in one node, so one block read moves you down one level of a very flat tree; a B-tree is the balanced version, with every leaf at the same depth and every node at least half full, so a million keys need three or four reads. Bayer and McCreight introduced B-trees in 1970, and database indexes and file systems use a B-tree or its B+ variant. Some syllabi list "multi-way search trees"; others list the 2-4 tree, the B-tree of order 4.

You need: BST search (4.5) and the idea of balance from 4.6.

m-way search trees

An m-way search tree is a tree in which each node holds up to m − 1 keys in sorted order, $k_1 < k_2 < \ldots < k_j$, and up to m children $c_0 \ldots c_j$, with every key in child c_i lying between k_i and k_{i+1}. A BST is the case m = 2. Searching is a BST search with a scan inside each node: find the first key ≥ x, stop if equal, otherwise follow the child just before it. Without further rules an m-way tree can be as unbalanced as a BST, so search is O(h) with no bound on h.

B-tree of order m

A B-tree (Knuth's definition, order m = maximum number of children) is an m-way search tree in which:

1. every node has at most m children, so at most m − 1 keys;
2. every internal node except the root has at least ⌈m/2⌉ children, so at least ⌈m/2⌉ − 1 keys: nodes are at least half full;
3. the root has at least two children unless it is a leaf;
4. all leaves are at the same level;
5. a node with j children holds exactly j − 1 keys.

CLRS uses a **minimum degree** t instead: every non-root node has between t − 1 and 2t − 1 keys, a B-tree of order 2t. Either way the height is bounded: with n keys, h ≤ log_t((n + 1)/2). For t = 51 (50 to 101 keys per node, about one 4 KB block) and n = 1 000 000, $\log_{51}(500\,000.5) \approx 3.3$, so at most three levels below the root: four block reads per search. A 2-3 tree is a B-tree of order 3; a 2-3-4 tree is order 4 (the "2-4 trees" of some syllabi) and is the red-black tree of 4.8 in disguise, each 3- or 4-node spread into a black node with red children.

Insertion: fill, then split

1. Search for the key; it belongs in a leaf. Insert it there in sorted order.
2. If the leaf now has m keys (one too many), **split** it: the median key moves up into the parent, the keys below the median stay as the left node and the keys above become a new right node.
3. If the parent now overflows, split it the same way, and so on up. If the root splits, the median becomes a new root with two children: this is the only way a B-tree grows in height, and it grows at the top, which is why all leaves stay at one level.

Each insertion costs O(h) block reads plus O(m) work per node, O(log n) overall; splits are rare (a node splits only after m − ⌈m/2⌉ further insertions).

Deletion: borrow or merge

Delete from a leaf; a key in an internal node is first replaced by its inorder predecessor or successor, which is then deleted from its leaf. If the leaf then has fewer than ⌈m/2⌉ − 1 keys (underflow): if a neighbouring sibling has a spare key, **borrow** through the parent (the parent key comes down, the sibling's end key goes up); otherwise **merge** with the sibling, pulling the separating parent key down, and check the parent in turn. If the root loses its last key its single child becomes the new root and the height shrinks by one.

B+ trees

Databases (MySQL InnoDB, PostgreSQL, SQLite) use the B+ tree: all records sit in the leaves, internal nodes hold only copies of keys to guide the search, and the leaves are linked left to right. A range query such as "all bookings between 1 and 31 October" finds the first leaf and walks the linked leaves without climbing back up; internal nodes holding keys only fit more children per block, making the tree flatter still.

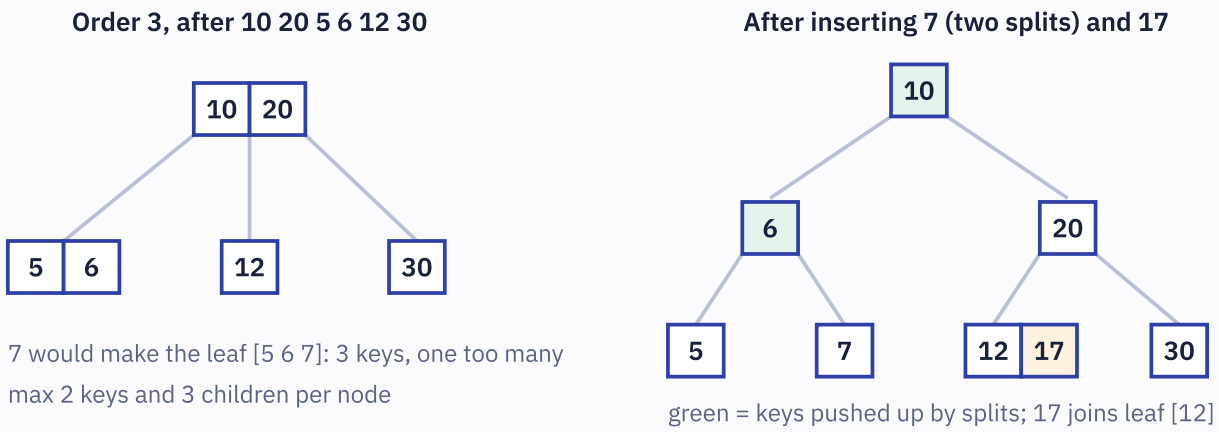


Figure 4.9.1 A B-tree of order 3 (at most 2 keys and 3 children per node, at least 1 key) during the worked example. Left: after six insertions the root holds 10 and 20 with three leaves. Right: inserting 7 overflows the leaf [5 6 7]; its median 6 moves up, the root becomes [6 10 20] and overflows in turn, so 10 moves up into a new root and the tree grows from two levels to three. 17 then slides into the leaf [12] with no split. All leaves stay at the same level.

WORKED EXAMPLE

Question. Insert 10, 20, 5, 6, 12, 30, 7, 17 in that order into an empty B-tree of order 3 (each node holds at most 2 keys and has at most 3 children). Draw the tree after every split and state the final height.

1. **10, 20**: the root is a leaf, [10 20]. Full but legal.
2. **5**: the root becomes [5 10 20], 3 keys, overflow. Split at the median 10: new root [10] with children [5] and [20]. Height 1.
3. **6**: 6 < 10, go left: leaf [5 6]. **12**: 12 > 10, go right: leaf [12 20].
4. **30**: right leaf becomes [12 20 30], overflow. Median 20 goes up: root [10 20], children [5 6], [12], [30]. This is the left picture of Figure 4.9.1.
5. **7**: 7 < 10, left leaf becomes [5 6 7], overflow. Median 6 goes up: root [6 10 20] with children [5], [7], [12], [30]. The root now has 3 keys, overflow again. Split the root at its median 10: new root [10]; left child [6] with children [5], [7]; right child [20] with children [12], [30]. Height 2.
6. **17**: 17 > 10, go right to [20]; 17 < 20, go left to leaf [12]; insert: [12 17]. No overflow. Final tree (right picture of Figure 4.9.1): root [10]; internal nodes [6] and [20]; leaves [5], [7], [12 17], [30]. **Height 2**, all four leaves at level 2, every non-root node with at least 1 key: a valid B-tree.

Follow-up. Delete 7 from the final tree: leaf [7] becomes empty, underflow. Its sibling [5] has only 1 key, nothing to spare, so merge: parent key 6 comes down, giving leaf [5 6] under the parent node, which is now empty. That parent underflows in turn; its sibling [20] has no spare key, so merge with the root key 10 coming down: node [10 20] with children [5 6], [12 17], [30], and the root is gone. Height drops to 1, the left tree of Figure 4.9.1 with 17 added to the leaf [12].

EXAM TRAPS

- **"Order" means different things.** Knuth's order m is the maximum number of children; CLRS's minimum degree t gives maximum 2t children. State the maximum keys per node in your first line and stick to it.
- **Split on m keys, not m – 1.** A node of order m may hold m – 1 keys; it splits only when a further insertion makes m. Splitting too early is the commonest mistake.
- **The median goes up, not a copy.** In a B-tree the median leaves the node; in a B+ tree a copy stays in the leaf.
- **Leaves at the same level is a rule, not a result.** Leaves at different depths mean the insertion was done wrongly.
- **Minimum keys for the root is 1**; the half-full rule applies to non-root nodes.

CHECK YOURSELF

– REMEMBER **State the properties of a B-tree of order m.** 3 marks

At most m children (m – 1 keys) per node; every non-root internal node has at least ⌈m/2⌉ children; the root has at least 2 children unless it is a leaf; all leaves at the same level; keys in a node sorted, with child i holding keys between key i and key i + 1.

– REMEMBER **What is the difference between a B-tree and a B+ tree?** 2 marks

In a B+ tree all records are in the leaves, internal nodes hold only guide keys, and the leaves are linked in sequence; a B-tree stores records in every node.

– UNDERSTAND **Why do B-trees grow at the root rather than at the leaves?** 3 marks

Insertion always happens in a leaf; overflow sends a median upward, and only a root split creates a new level. Growing at the top keeps every leaf at the same depth, which a tree growing at the leaves could not guarantee.

– APPLY **Insert 1, 2, 3, 4, 5, 6, 7 into an empty 2-3 tree (order 3). Draw the result.** 4 marks

[1 2] → insert 3 splits: 2(1, 3). Insert 4: 2(1, [3 4]). 5 splits [3 4 5]: root [2 4] with leaves 1, 3, 5. 6: [2 4](1, 3, [5 6]). 7 splits [5 6 7], 6 goes up: root [2 4 6] overflows, 4 goes up: root 4 with children [2] (leaves 1, 3) and [6] (leaves 5, 7). Height 2.

– APPLY **A B-tree has minimum degree t = 2 (order 4). What are the minimum and maximum number of keys in a tree of height 2?** 4 marks

Minimum: root 1 key, 2 children each with 1 key, 4 grandchildren each with 1 key: 1 + 2 + 4 = **7**. Maximum: 3 keys per node, 4 children per node: 3 + 4×3 + 16×3 = 3 + 12 + 48 = **63**.

– ANALYZE **Explain how a 2-3-4 tree corresponds to a red-black tree.** 4 marks

A 2-node is a black node; a 3-node (2 keys) is a black node with one red child; a 4-node (3 keys) is a black node with two red children. Every 2-3-4 leaf at the same level becomes equal black-height, and "no node with more than 3 keys" becomes "no red node with a red child". Splitting a 4-node is the red-black recolouring step.

BEYOND THE SYLLABUS

Not for the exam. Nearly every index you will ever query is a B+ tree: InnoDB clusters each table by its primary key in one, PostgreSQL's default index is a B-tree, and ext4, NTFS, Btrfs and APFS keep directories and extents in B-tree variants. Write-heavy systems (LevelDB, RocksDB, Cassandra) use the LSM tree instead, which batches writes and merges sorted runs. Search "B+ tree InnoDB", "LSM tree vs B-tree" and "learned index".

Key terms of this lesson

TERM	MEANING OR FORMULA	CHAPTER
B-tree of order m	≤ m children, ≥ ⌈m/2⌉ for non-root; leaves at one level; split sends the median up	4.9

Exam question bank

Long-answer exam questions on these chapters, each with an outline of the points an examiner looks for. Answer one in full on paper before you read its outline.

Remember and understand

- **State the properties of a red-black tree and of a B-tree of order m. Give the height bound of each and say where each is used. (8 marks)**

  Outline: the five red-black rules, height ≤ $2 \log_2(n + 1)$, used in C++ `std::map` and the Linux kernel; B-tree rules (≤ m children, ≥ ⌈m/2⌉ for non-root, leaves at one level, j – 1 keys for j children), height ≤ log_t((n + 1)/2), used by database indexes and file systems because one node is one disk block.

Apply

- **Insert 20, 40, 10, 30, 15, 35, 7, 26 into an empty B-tree of order 4 (maximum 3 keys per node), showing each split. Then delete 40. (10 marks)**

  Outline (taking the second of four keys as the median; the third is also accepted if used consistently): [20 40] → [10 20 40] → 30 overflows: median 20 up, root [20] with [10] and [30 40]; 15 → [10 15]; 35 → [30 35 40]; 7 → [7 10 15]; 26 → [26 30 35 40] overflows, median 30 up: root [20 30], children [7 10 15], [26], [35 40]. Delete 40 from leaf [35 40]: [35], no underflow (minimum 1 key for order 4).

Analyze and evaluate

- **Keys arrive in sorted order. Compare the resulting BST, AVL tree and red-black tree after n insertions in height, number of rotations and search cost, and recommend one for a phone-book application that is read far more often than written. (10 marks)**

  Outline: BST becomes a chain of height n – 1, search O(n); AVL height ≤ $1.44 \log_2 n$ with about one rotation per insertion; red-black height ≤ $2 \log_2(n + 1)$ with at most two rotations per insertion but many recolourings. Read-heavy favours the shallower AVL; write-heavy favours red-black. Mention that a B-tree would be the answer if the book lives on disk.

Glossary (terms used in this lesson)

B-tree	A balanced multi-way search tree with all leaves at one level and every non-root node at least half full.
B+ tree	A B-tree variant with all records in linked leaves and only guide keys in internal nodes.

Black-height	The number of black nodes on any path from a node down to a NULL leaf in a red-black tree.
Degree	Of a node, its number of children; of a tree, the largest node degree.
Depth	The number of edges from the root to a node; the root has depth 0.
Height	The longest downward path from a node to a leaf, in edges; a leaf has height 0 and an empty tree −1.
Leaf	A node with no children.
m-way search tree	A tree whose nodes hold up to m − 1 sorted keys and up to m children.
Red-black tree	A BST with red and black nodes, no red-red parent-child pair and equal black counts on all paths.
Split	In a B-tree, dividing an overfull node into two and sending its median key to the parent.

Sources and further reading

- Mark Allen Weiss, *Data Structures and Algorithm Analysis in C*, 2nd edition, Pearson. Chapter 4 (Trees: terminology, traversals, BST, AVL, splay, B-trees) and Chapter 6 (Priority Queues: binary heaps). A common prescribed text.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*, 3rd edition. Chapters on heapsort (binary heaps and build-heap analysis), binary search trees, red-black trees and B-trees (minimum-degree definition and height bound).
- Ellis Horowitz, Sartaj Sahni and Susan Anderson-Freed, *Fundamentals of Data Structures in C*, 2nd edition. Chapter 5 (Trees) for representations, threaded trees and heaps; note that it counts height in levels.
- Aaron M. Tenenbaum, Yedidyah Langsam and Moshe J. Augenstein, *Data Structures Using C*, Pearson. Trees and their applications, expression trees.
- Michael T. Goodrich, Roberto Tamassia and Michael H. Goldwasser, *Data Structures and Algorithms in Java*, 6th edition, Wiley. (2,4) trees and their relation to red-black trees.
- Wikipedia, "AVL tree": en.wikipedia.org/wiki/AVL_tree (opened for the inventors, 1962, and the balance-factor definition).
- Wikipedia, "Red–black tree": en.wikipedia.org/wiki/Red–black_tree (opened for the property list and the Guibas–Sedgewick 1978 origin).
- Wikipedia, "Splay tree": en.wikipedia.org/wiki/Splay_tree (opened for the zig, zig-zig and zig-zag definitions and the Sleator–Tarjan 1985 amortised bound).
- Wikipedia, "B-tree": en.wikipedia.org/wiki/B-tree (opened for Knuth's order-m definition and the Bayer–McCreight 1970 origin).

- Syllabi: the universal syllabus and a range of university course documents.

Built for the University free n easy YouTube channel from the universal syllabus of Data Structures (Oct 2026). Written with top universities' syllabi and standard textbooks; check a formula against your prescribed book before an exam, and report mistakes on the channel.