14.3 B+ Trees

Ellie Thomas

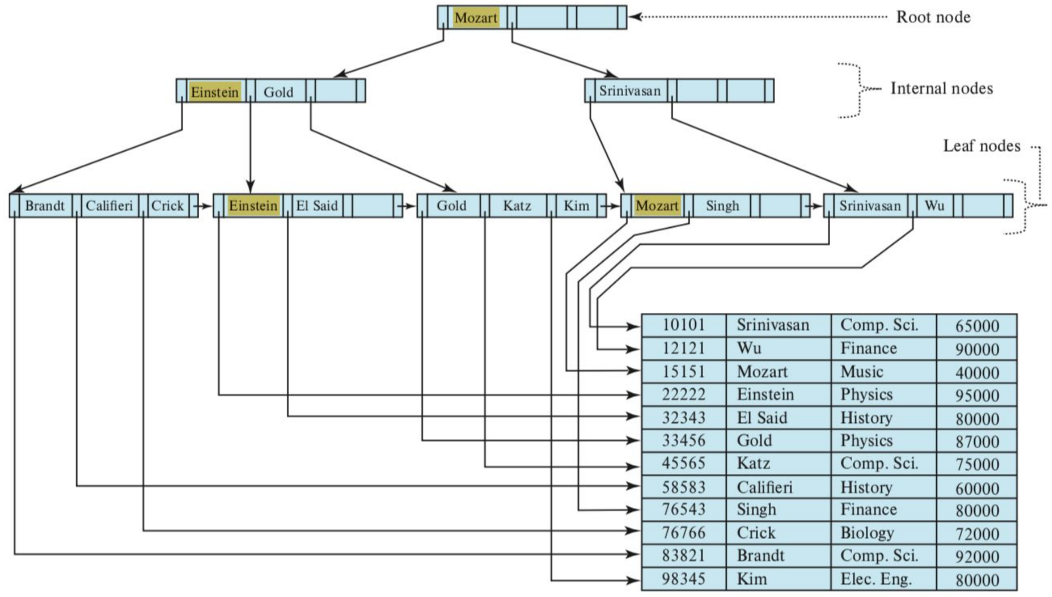
Overview

- B+ index tree structure is an alternative to the index-sequential file organization
- “B” stands for balanced
    - Balanced tree structure: equal length of every path from the root to the leaf
- Therefore it is a shallow tree
- Makes searches, insertions, deletions efficient

- Node Structure
    - Alternating pointer then search-key value, ending with another pointer
    - So there are n pointers and n-1 values
- Pick a column, tree based on that

Tree Structure

- n: the number of pointers/records corresponding to each node, constant for a given tree
  - Here n = 4
- Leaves of tree contain pointers to every record
- Higher level nodes form a sparse index on the leaf nodes - see highlight
- The data is imagined to be in a table like structure, but there is no need for the data to be stored together because we are using pointers, so we can store information anywhere and the pointer will indicate its location in memory
- Links across the bottom make range queries easy - linked across in sorted order

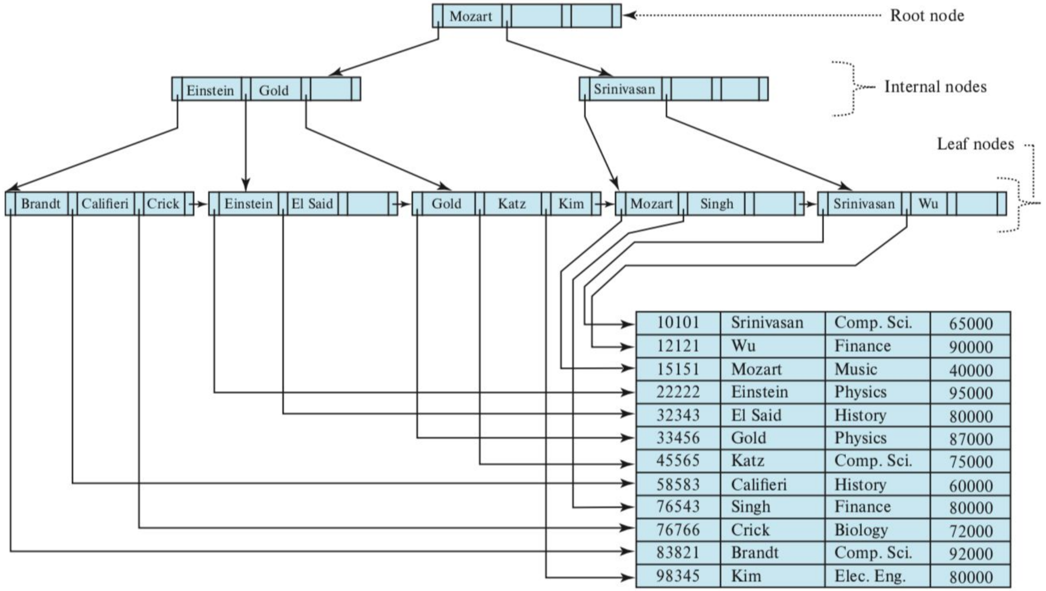


10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 14.9 B$^+$-tree for *instructor* file ($n = 4$).

Insertion of “Adams”

- Assume unique keys
- When inserting:
- If necessary, split nodes, redistribute search key values



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tor file ($n = 4$).

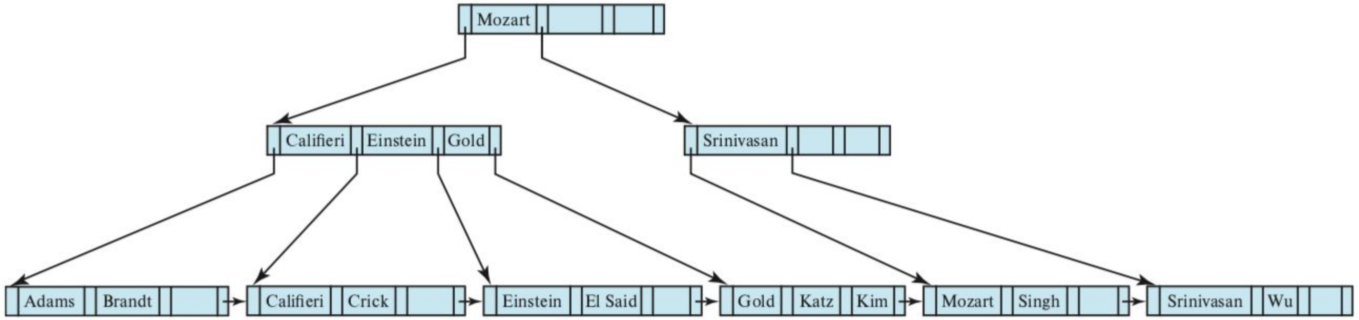
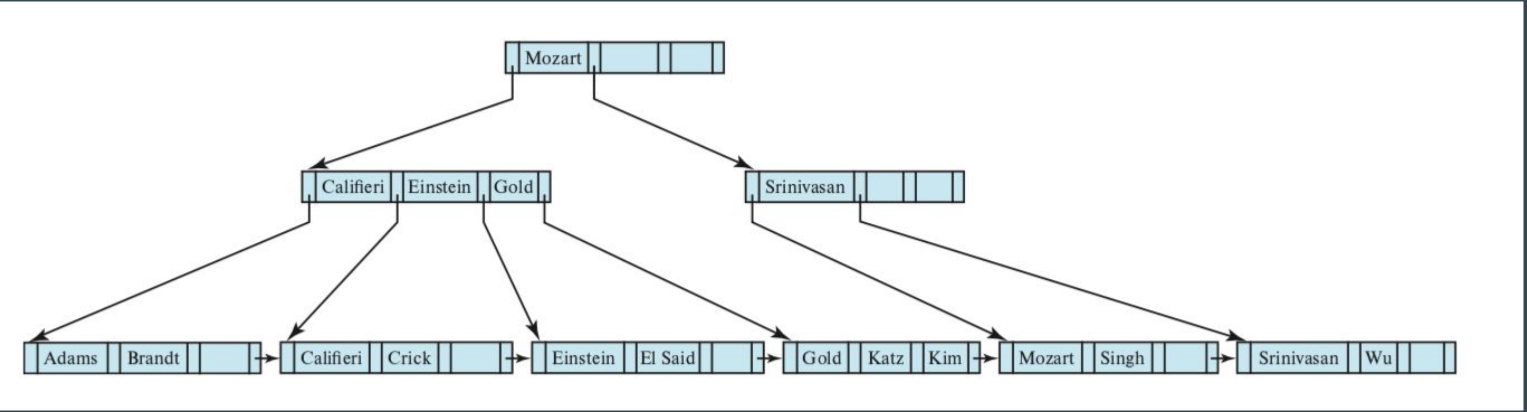


Figure 14.14 Insertion of “Adams” into the B$^+$-tree of Figure 14.9.

Insertion of “Lamport”



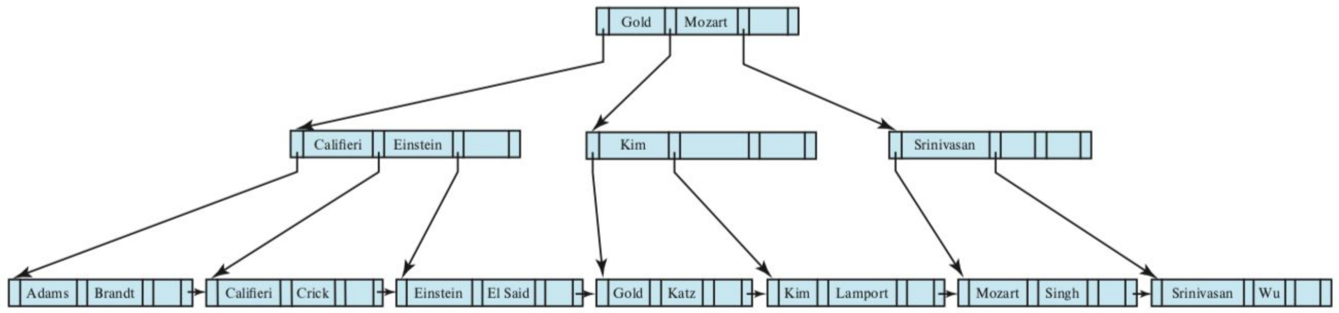
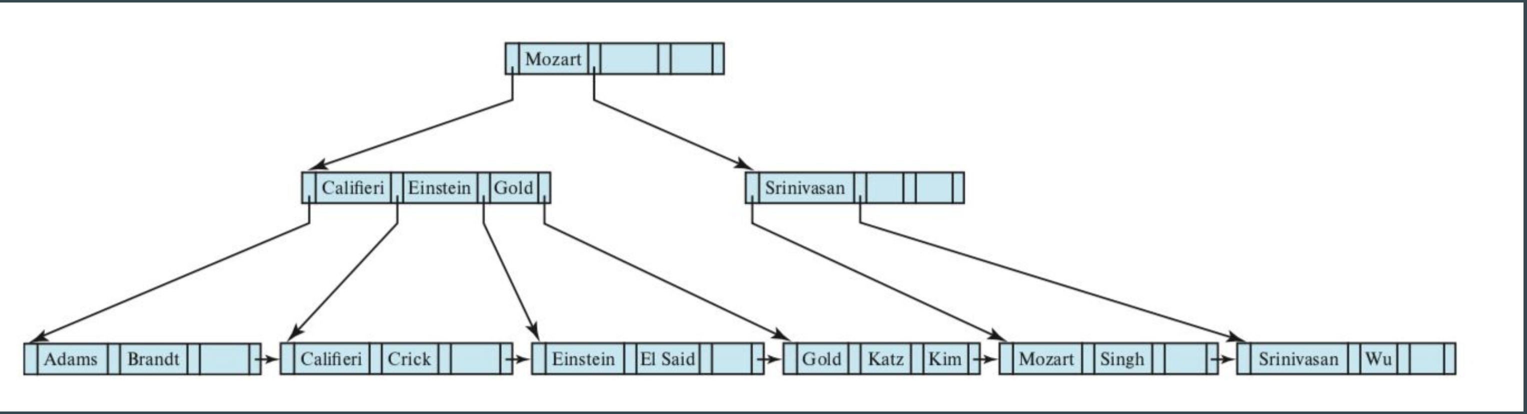


Figure 14.15 Insertion of “Lamport” into the B$^+$-tree of Figure 14.14.

Deletion of “Srinivasan”



- If underfilled, first try to merge nodes
- If that’s impossible, redistribute search key values
- Underfilled means less than [n/2] pointers in the node

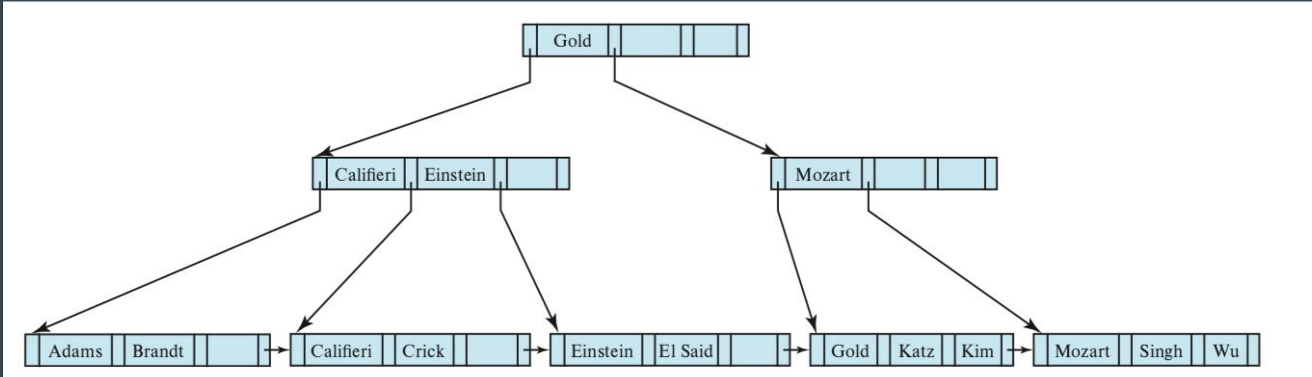
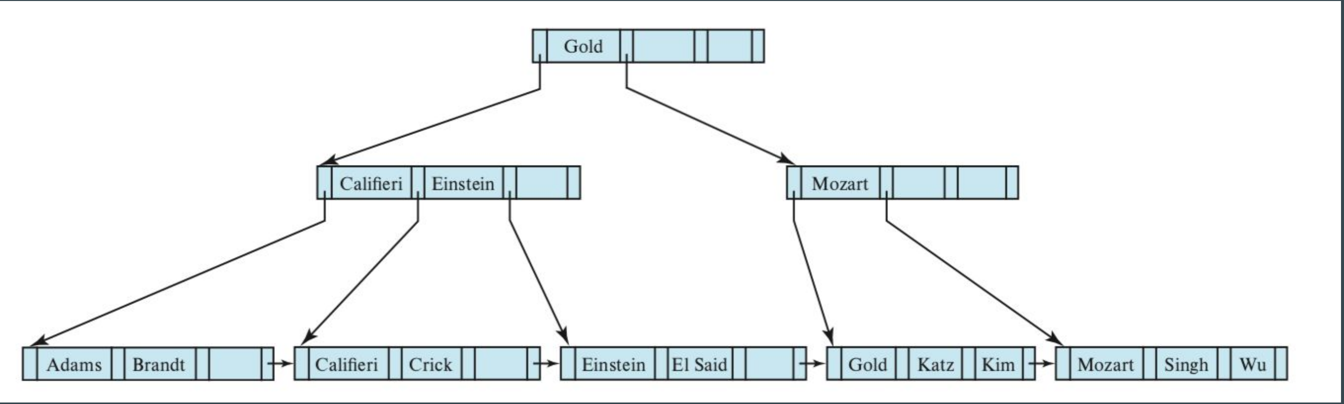


Figure 14.18 Deletion of “Srinivasan” from the B$^+$-tree of Figure 14.14.

Deletion of “Singh” and “Wu”



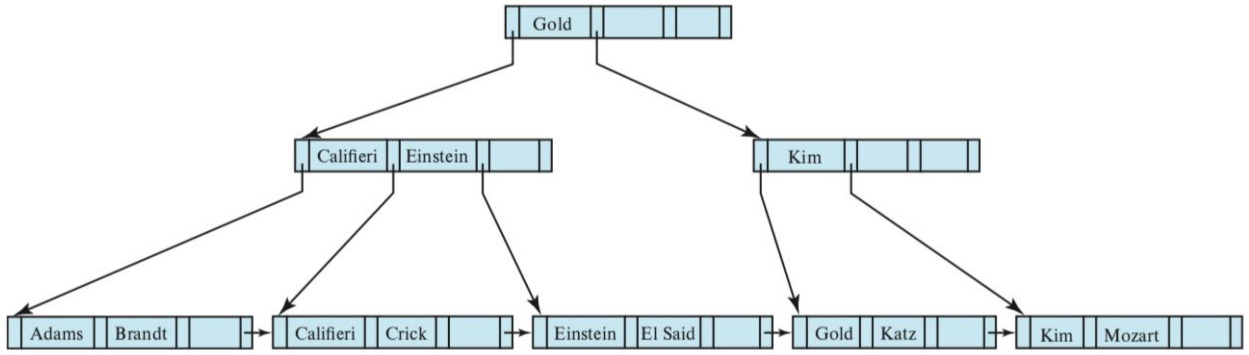


Figure 14.19 Deletion of “Singh” and “Wu” from the B$^+$-tree of Figure 14.18.

Order for B+ Tree Operations

- For any operations, (search, insert, delete) time complexity is $O(\log_{[n/2]}(N))$
  - N: total # of records
  - n: max # of pointers in a nods (4 for our example)
    - n/2 because each node must have 2 pointers
- So time complexity is proportional to the height of a tree
  - Balanced tree -> shallow tree -> efficient operations

Source

Silberschatz, Abraham, et al. *Database System Concepts*. Seventh edition, McGraw-Hill, 2020.