COL362/632 Introduction to Database Management Systems

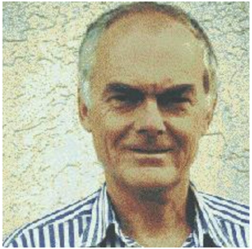
Indexing – B+Trees

Kaustubh Beedkar

Department of Computer Science and Engineering
Indian Institute of Technology Delhi

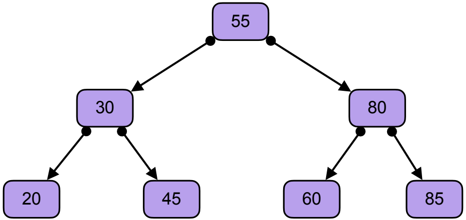
B/B+Tree Trivia

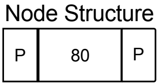
- Rudolf Bayer (invented B-trees in 1972; with McCreight)
- Gold standard for indexing,
  a.k.a. the greatest data structure of all times!
- Impact
  - Databases
  - FileSystems
  - Key Value stores
  - Search Engines
- No official explanation for what **B** stands for in B-Trees



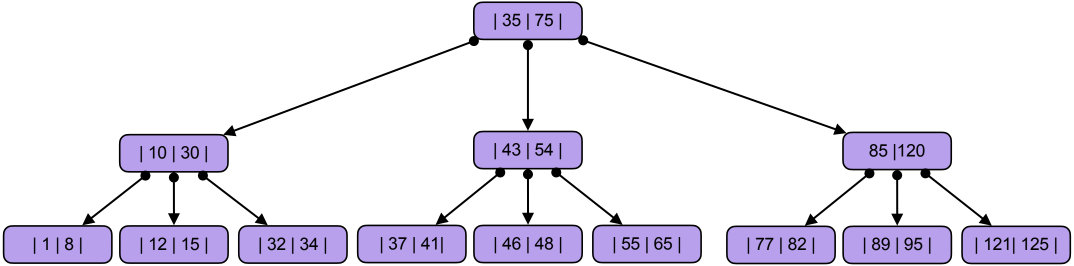
"the more you think about what the B in B-Tree means, the better you understand B-Trees!"

Binary Search Trees





m-Way Search Trees

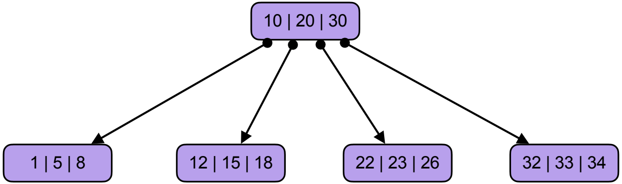


Node Structure

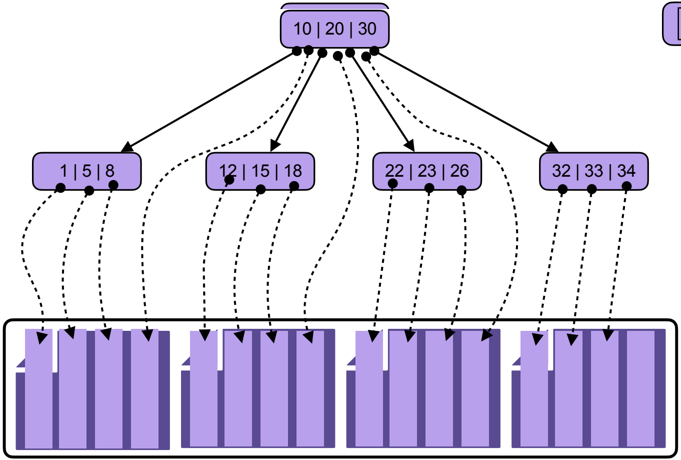
P1	K1	P2	K2	P3	K3	...	P_{M-1}	K_{M-1}	P_M

B-Trees

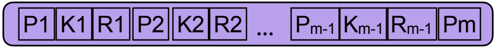
- *m*-way search trees
- Root node has at least two children (unless it a leaf)
- Every inner node has at least $\lceil \frac{m}{2} \rceil$ children
- All leaf nodes have the same number of ancestors



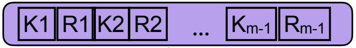
B-Trees as a Database Index



Database File



Non-leaf Node



Leaf Node

B-Tree Family

Generally referred to as class of balanced tree data structures

- B-Tree
- B+Tree
- B*Tree
- B^{link}Tree
- $B\epsilon$-Trees
- ...

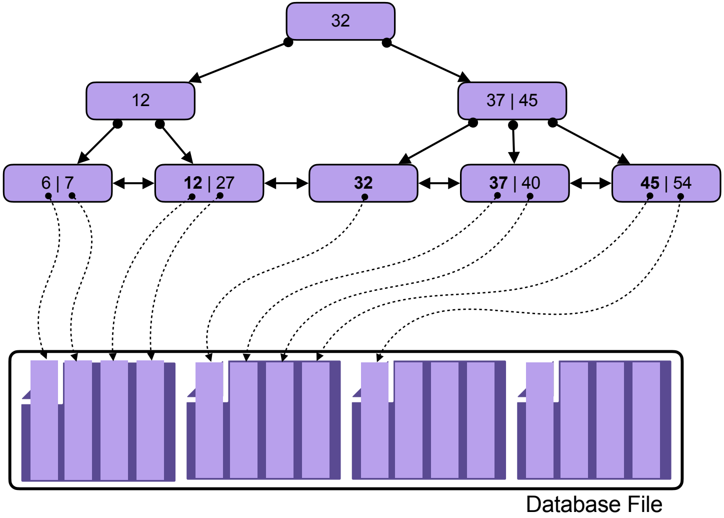
B+Tree

- Self balancing m-way search tree
- Allows searches, insertions, deletion, and sequential access in $O(\log n)$
- Optimized for reading and writing large blocks of data

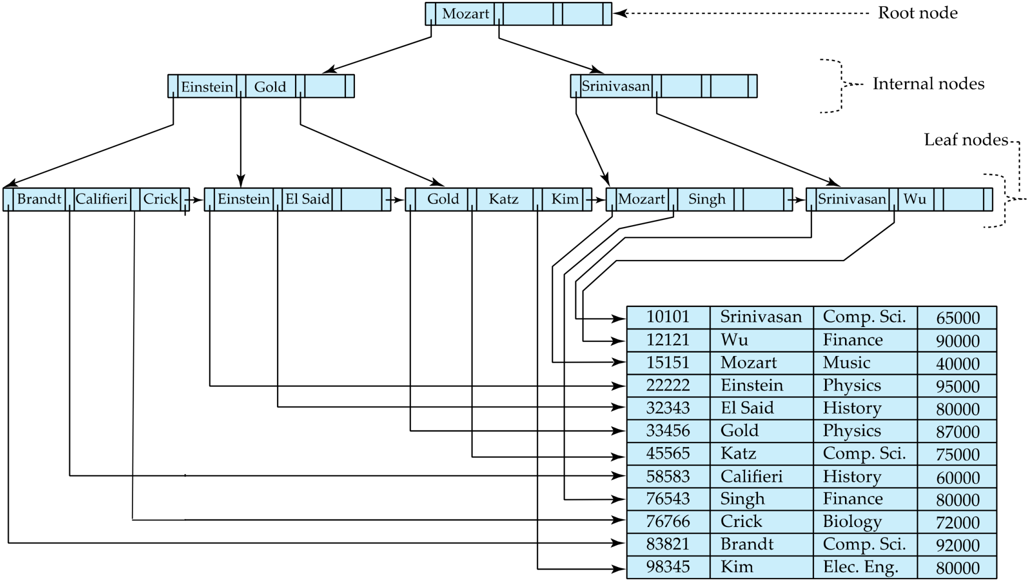
Properties

- All leaf nodes have the same number of ancestors
- Root node has at least 2 and at most m children
- Every leaf node has $\left\lceil \frac{m-1}{2} \right\rceil \leq \#keys \leq m-1$
- Every internal node has $\left\lfloor \frac{m-1}{2} \right\rfloor \leq \#keys \leq m-1$

B+Tree Example



B+ Tree (Example from Book B1)



Node Structure

Non-leaf Nodes

- Node with K keys has $K + 1$ children

P1	K1	P2	K2	P3	K3	...	P_{M-1}	K_{M-1}	P_M

Leaf Nodes

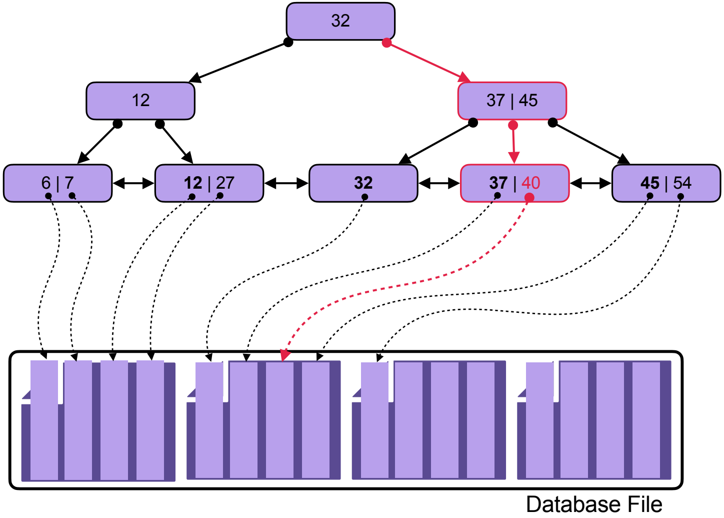
- Node with K keys has K **record pointers**
- Has pointers to previous and next leaf notes

P1	K1	P2	K2	P3	K3	...	P_{M-1}	K_{M-1}	P_M

B+Tree Operations

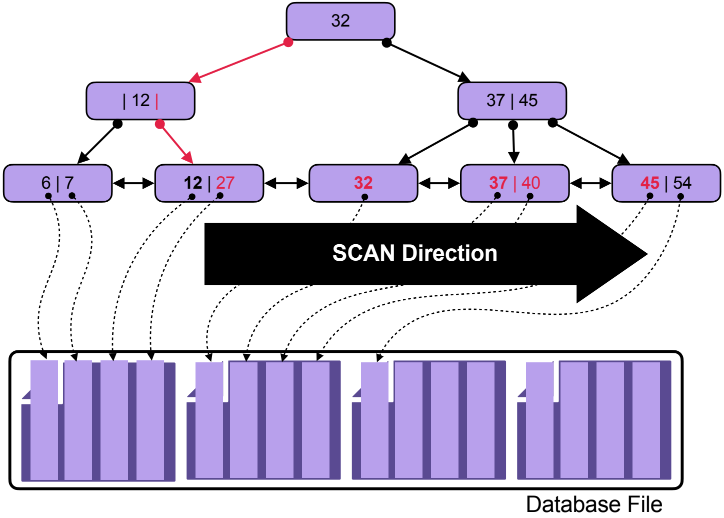
- Search: point and range queries
- Insertions
- Deletions
- Bulk loading

key=40



Range Queries

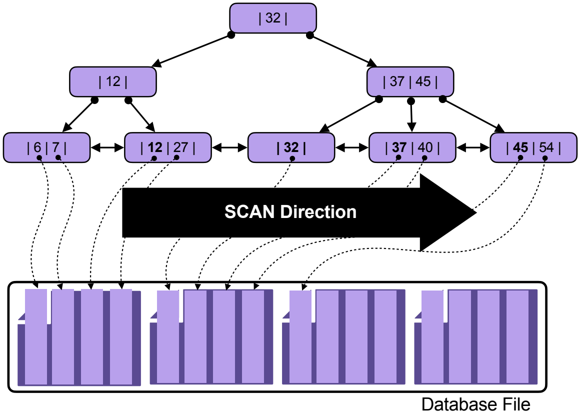
$27 \leq \textbf{key} \leq 45$



Clustered and non-clustered B+Tree

Clustered B+Tree

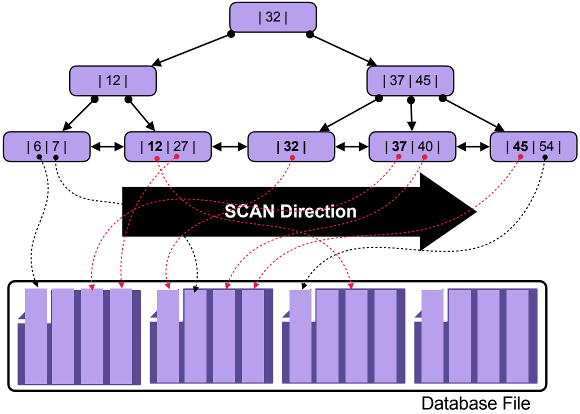
- Find the left most entry and sequentially scan right



Clustered and non-clustered B+Tree

Non-clustered B+Tree

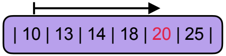
- First find all pages
- then, sort by page id



Searching within a node

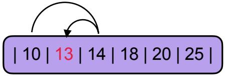
1. Linear Search

   e.g., key=20



2. Binary Search

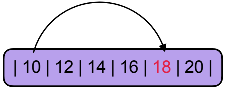
   e.g., key=13



3. Interpolation Search

   Improvement over binary search where values are uniformly distributed

   e.g., key=18; offset: $\frac{(18-10)\times 6}{20-10}$



B+Tree Insertion

1. Perform search to find the leaf node ℓ
2. Check if ℓ is full or not

– If ℓ is not full, then add the record

– Otherwise split ℓ into
  - ℓ with $\left\lfloor \frac{K+1}{2} \right\rfloor$
  - ℓ' with $\left\lceil \frac{K+1}{2} \right\rceil$

3. Insert $\left\lceil \frac{K+1}{2} \right\rceil$-th key to parent pointing to ℓ'
4. Propagate recursively

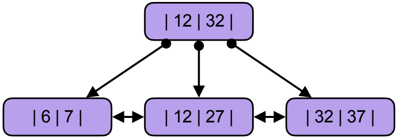
Examples

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

Bulk Insert

Given a collection of records, create a B+tree index on some key

- Point inserts are expensive!
- Instead perform bulk loading
  1. Sort entries according to key
  2. Construct tree bottom-up
- **Example** $m = 4$
  - Bulk insert: 27, 12, 7, 6, 32, 37
  - Sorted keys: 6, 7, 12, 27, 32, 37



B+Tree Deletions

1. Perform search to find the leaf node ℓ
2. If ℓ is at least half full, delete the entry, done!
3. Otherwise, try redistributing by borrowing from sibling
   - If successful, update the parent node
   - If not, merge[1] ℓ and sibling
     - update parent by deleting the index key pointing to deleted entry

[1] Some DBMS may delay merging

Examples

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

Misc. Aspects

Handling duplicates

1. Append Record ids to keys
2. Overflow pages

Handling nulls

- Store all null keys at either first or last leaf nodes

Handling variable length keys

1. Use pointers (useful for in-memory DBMS)
2. Variable length nodes (requires intricate memory management)
3. Padding (make all keys of the same length)