Database Management System – 48 (Multilevel indexes, B Tree and B+ tree)

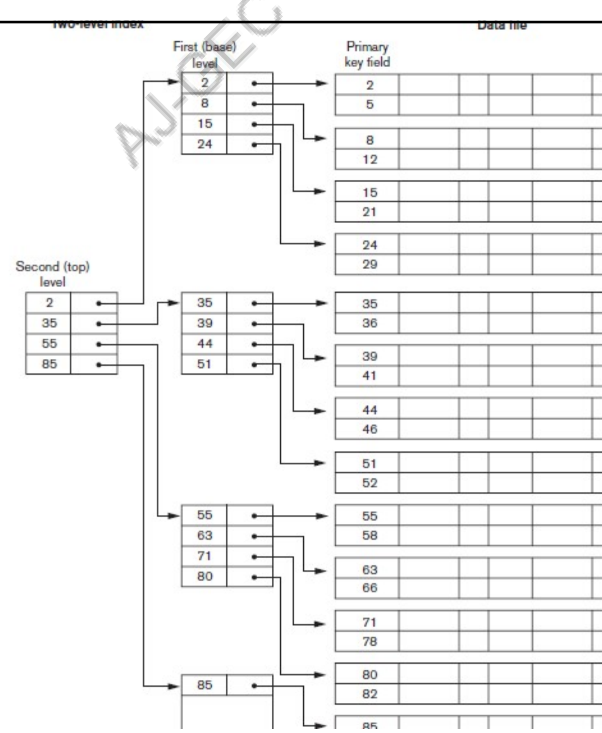
Ajay James
Asst. Prof in CSE
Government Engineering College Thrissur

Outline

- Multilevel indexes
- Search Tree
- B Tree
- B+ Tree

Multilevel Indexes

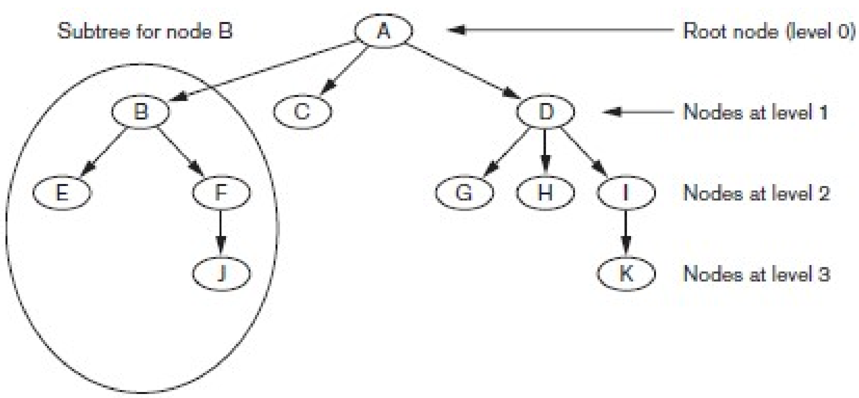
- Designed to greatly reduce remaining search space as search is conducted
- Index file
  - Considered first (or base level) of a multilevel index
- Second level
  - Primary index to the first level
- Third level
  - Primary index to the second level



Dynamic Multilevel Indexes Using B Trees and B+ Trees

- Tree data structure terminology
  - Tree is formed of nodes
  - Each node (except root) has one parent and zero or more child nodes
  - Leaf node has no child nodes
    - Unbalanced if leaf nodes occur at different levels
  - Nonleaf node called internal node
  - Subtree of node consists of node and all descendant nodes

Search tree



(Nodes E, J, C, G, H, and K are leaf nodes of the tree)

Search Trees

- Search tree used to guide search for a record
  - Search tree of ***order p*** is a tree such that each node contains ***at most p − 1 search values***
  - and ***p pointers*** in the order $<P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q>$, where $q \le p$
  - Each P_i is a pointer to a child node (or a NULL pointer),
  - and each K_i is a search value from some ordered set of values.

Constraints on Search Tree

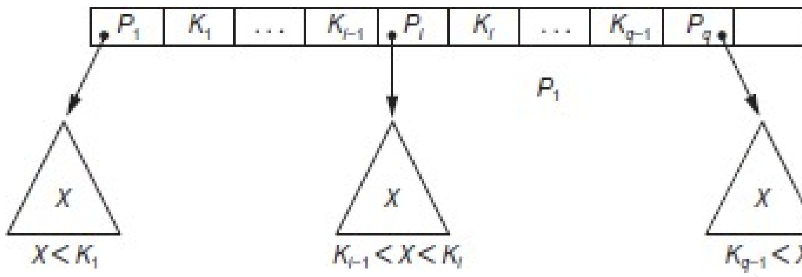
1. Within each node, $K_1 < K_2 < \dots < K_{q-1}$.
2. For all values X in the subtree pointed at by P_i, we have

   $K_{i-1} < X < K_i$ for $1 < i < q$;

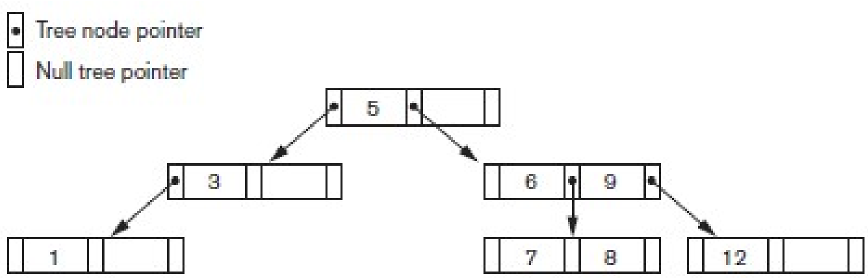
   $X < K_i$ for $i = 1$;

   and $K_{i-1} < X$ for $i = q$

Node in search tree



Example



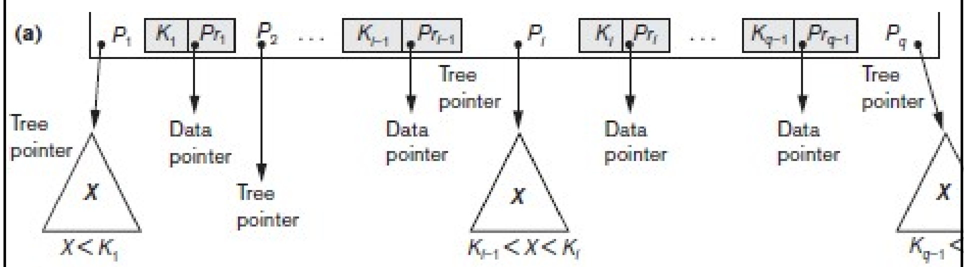
B Trees

- Provide multi-level access structure
- Tree is always balanced
- Space wasted by deletion never becomes excessive
- Each node is at least half-full
- Each node in a B-tree of order *p can have at* most *p-1 search values*

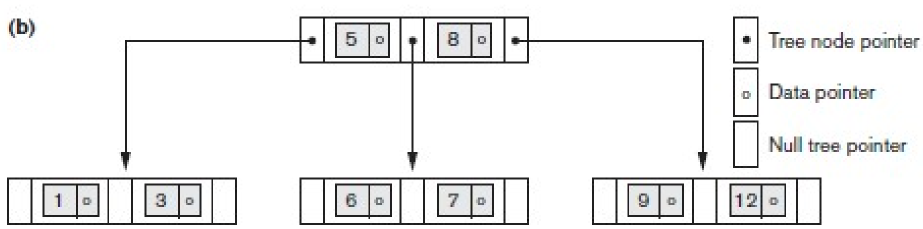
B Tree

1. Each internal node in the B-tree is of the form $\boldsymbol{<P_1, <K_1, P_{r1}>, P_2, <K_2, Pr_2>, \ldots, <K_{q-1}, P_{rq-1}>, P_q>}$ where $q \le p$.
2. Within each node, $\boldsymbol{K_1 < K_2 < \ldots < K_{q-1}}$
3. For all search key field values X in the subtree pointed at by P_i (the i^{th} subtree), we have: $\boldsymbol{K_{i-1} < X < K_i}$ for $1 < i < q$; $X < K_i$ for $i = 1$; and $K_{i-1} < X$ for $i = q$
4. Each node has ***at most p*** tree pointers
5. Each node, except the root and leaf nodes, has at least $\lceil \mathbf{(p/2)} \rceil$ tree pointers
6. A node with q tree pointers, $q \le p$, has ***q − 1 search*** key field values (and hence has q − 1 data pointers)
7. All leaf nodes are at the ***same level***

Node in a B Tree



Example



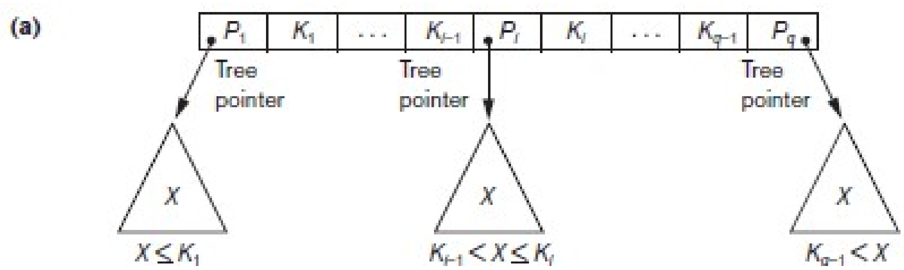
B+ Trees

- Data pointers stored only at the leaf nodes
  - Leaf nodes have an entry for every value of the search field, and a data pointer to the record if search field is a key field
  - For a nonkey search field, the pointer points to a block containing pointers to the data file records
  - Leaf nodes of the B+ tree are usually linked
- Internal nodes
  - Some search field values from the leaf nodes are repeated to guide search

Internal node of B+ tree

1. Each internal node is of the form ***<P_1, K_1, P_2, K_2, … , P_{q-1}, K_{q-1}, P_q>*** where $q \le p$ and each P_i is a tree pointer.
2. Within each internal node, ***$K1 < K2 < \ldots < K_{q-1}$***
3. For all search field values X in the subtree pointed at by P_i, we have ***$K_{i-1} < X \le K_i$*** for $1 < i < q$; $X \le K_i$ for $i = 1$; and $K_{i-1} < X$ for $i = q$
4. Each internal node has at most ***p tree pointers***
5. Each internal node, except the root, has at least $\lceil$**(p/2)**$\rceil$ tree pointers
6. An internal node with q pointers, $q \le p$, has ***q − 1 search*** field values

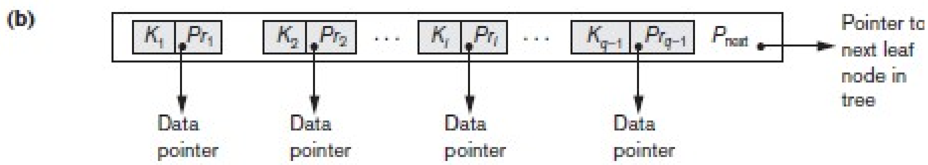
Internal node



Leaf Node of B + tree

1. Each leaf node is of the form ***<<K_1, P_{r1}>, <K_2, P_{r2}>, ... , <K_{q-1}, P_{rq-1}>, P_{next}>*** where $q \leq p$, each P_{ri} is a data pointer, and P_{next} points to the next leaf node of the B+ tree
2. Within each leaf node, ***$K_1 \leq K_2 \ldots , K_{q-1}$***, $q \leq p$.
3. Each P_{ri} is a ***data pointer*** that points to the record whose search field value is K_i or to a file block containing the record
4. Each leaf node has at least $\lceil \mathbf{(p/2)} \rceil$ values.
5. All leaf nodes are at the ***same level***

Leaf node



Exercise

Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $P_R = 7$ bytes long. A file has $r = 30,000$ EMPLOYEE records of *fixed length*. Each record has the following fields: Name (30 bytes),Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

a. Calculate the record size R in bytes.

b. Calculate the blocking factor *bfr* and the number of file blocks *b*, assuming an unspanned organization.

c. Suppose that the file is *ordered* by the key field Ssn and we want to construct a *primary index* on Ssn. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out *fo*); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value—using the primary index.

Exercise contd...

d. Suppose that the file is *not ordered* by the key field Ssn and we want to construct a *secondary index* on Ssn. Repeat the previous exercise (part c) for the secondary index and compare with the primary index.

e. Suppose that the file is *not ordered* by the nonkey field Department_code and we want to construct a *secondary index* on Department_code, using option 3 of Section 17.1.3, with an extra level of indirection that stores record pointers. Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of blocks needed by the level of indirection that stores record pointers; (iii) the number of first-level index entries and the number of first-level index blocks; (iv) the number of levels needed if we make it into a multilevel index; (v) the total number of blocks required by the multilevel index and the blocks used in the extra level of indirection; and (vi) the approximate number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the index.

Exercise contd...

f. Suppose that the file is *ordered* by the nonkey field Department_code and we want to construct a *clustering index* on Department_code that uses block anchors (every new value of Department_code starts at the beginning of a new block). Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the clustering index (assume that multiple blocks in a cluster are contiguous).

Reference

- Elmasri R. and S. Navathe, Database Systems: Models, Languages, Design and Application Programming, Pearson Education 6th edition and 7th edition

Thank you