CHAPTER 12

Indexing and Hashing

This chapter covers indexing techniques ranging from the most basic one to highly specialized ones. Due to the extensive use of indices in database systems, this chapter constitutes an important part of a database course.

A class that has already had a course on data-structures would likely be familiar with hashing and perhaps even B^+-trees. However, this chapter is necessary reading even for those students since data structures courses typically cover indexing in main memory. Although the concepts carry over to database access methods, the details (e.g., block-sized nodes), will be new to such students.

The sections on B-trees (Sections 12.4), grid files (Section 12.9.3) and bitmap indexing (Section 12.9.4) may be omitted if desired.

Changes from 3rd edition:
The description of querying on B^+-trees has been augmented with pseudo-code. The pseudo-code for insertion on B^+-trees has been simplified. The section on index definition in SQL (Section 12.8) is new to this edition, as is the coverage of bitmap indices (Section 12.9.4).

Exercises

12.1 When is it preferable to use a dense index rather than a sparse index? Explain your answer.
Answer: It is preferable to use a dense index instead of a sparse index when the file is not sorted on the indexed field (such as when the index is a secondary index) or when the index file is small compared to the size of memory.

12.2 Since indices speed query processing, why might they not be kept on several search keys? List as many reasons as possible.
Answer: Reasons for not keeping several search indices include:

a. Every index requires additional CPU time and disk I/O overhead during inserts and deletions.
b. Indices on non-primary keys might have to be changed on updates, although an index on the primary key might not (this is because updates typically do not modify the primary key attributes).
c. Each extra index requires additional storage space.
d. For queries which involve conditions on several search keys, efficiency might not be bad even if only some of the keys have indices on them. Therefore database performance is improved less by adding indices when many indices already exist.

12.3 What is the difference between a primary index and a secondary index?
Answer: The primary index is on the field which specifies the sequential order of the file. There can be only one primary index while there can be many secondary indices.

12.4 Is it possible in general to have two primary indices on the same relation for different search keys? Explain your answer.
Answer: In general, it is not possible to have two primary indices on the same relation for different keys because the tuples in a relation would have to be stored in different order to have same values stored together. We could accomplish this by storing the relation twice and duplicating all values, but for a centralized system, this is not efficient.

12.5 Construct a B^+-tree for the following set of key values:

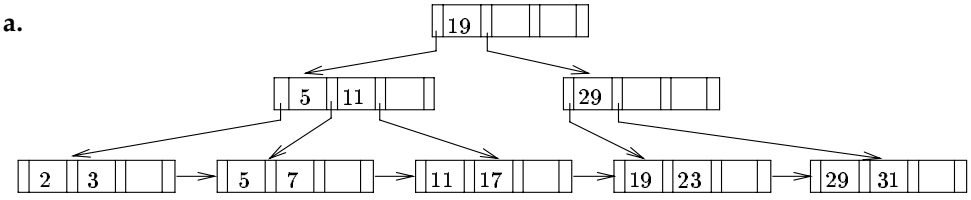
$$(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)$$

Assume that the tree is initially empty and values are added in ascending order. Construct B^+-trees for the cases where the number of pointers that will fit in one node is as follows:

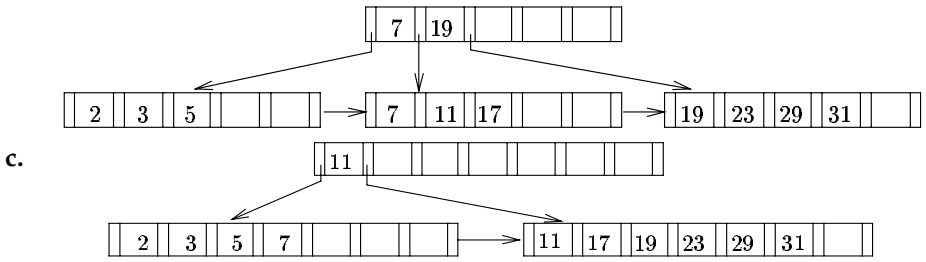
a. Four
b. Six
c. Eight

Answer: The following were generated by inserting values into the B^+-tree in ascending order. A node (other than the root) was never allowed to have fewer than $\lceil n/2 \rceil$ values/pointers.

a.



b.



12.6 For each B^+-tree of Exercise 12.5, show the steps involved in the following queries:

a. Find records with a search-key value of 11.
b. Find records with a search-key value between 7 and 17, inclusive.

Answer:

With structure 0.a:

a. Find records with a value of 11
   i. Search the first level index; follow the first pointer.
   ii. Search next level; follow the third pointer.
   iii. Search leaf node; follow first pointer to records with key value 11.
b. Find records with value between 7 and 17 (inclusive)
   i. Search top index; follow first pointer.
   ii. Search next level; follow second pointer.
   iii. Search third level; follow second pointer to records with key value 7, and after accessing them, return to leaf node.
   iv. Follow fourth pointer to next leaf block in the chain.
   v. Follow first pointer to records with key value 11, then return.
   vi. Follow second pointer to records with with key value 17.

With structure 0.b:

a. Find records with a value of 11
   i. Search top level; follow second pointer.
   ii. Search next level; follow second pointer to records with key value 11.
b. Find records with value between 7 and 17 (inclusive)
   i. Search top level; follow second pointer.
   ii. Search next level; follow first pointer to records with key value 7, then return.
   iii. Follow second pointer to records with key value 11, then return.
   iv. Follow third pointer to records with key value 17.

With structure 0.c:

a. Find records with a value of 11
   i. Search top level; follow second pointer.
   ii. Search next level; follow first pointer to records with key value 11.
b. Find records with value between 7 and 17 (inclusive)

i. Search top level; follow first pointer.
ii. Search next level; follow fourth pointer to records with key value 7, then return.
iii. Follow eighth pointer to next leaf block in chain.
iv. Follow first pointer to records with key value 11, then return.
v. Follow second pointer to records with key value 17.

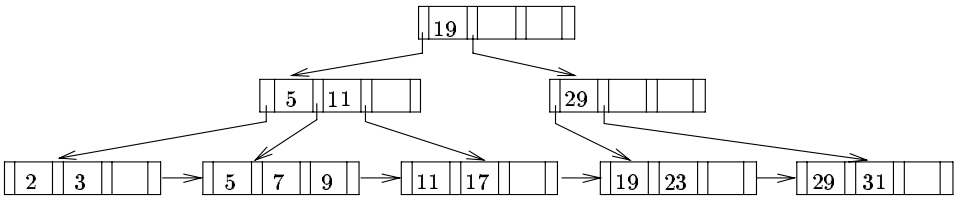
12.7 For each B^+-tree of Exercise 12.5, show the form of the tree after each of the following series of operations:

a. Insert 9.
b. Insert 10.
c. Insert 8.
d. Delete 23.
e. Delete 19.

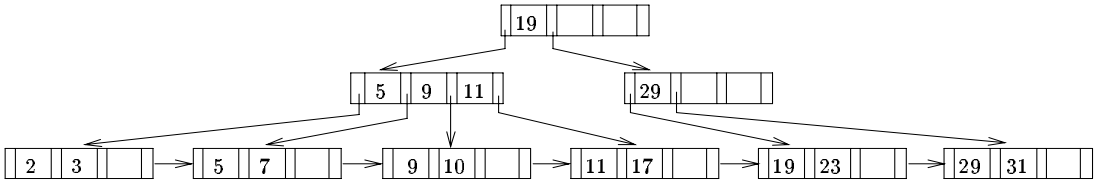
Answer:

- With structure 0.a:

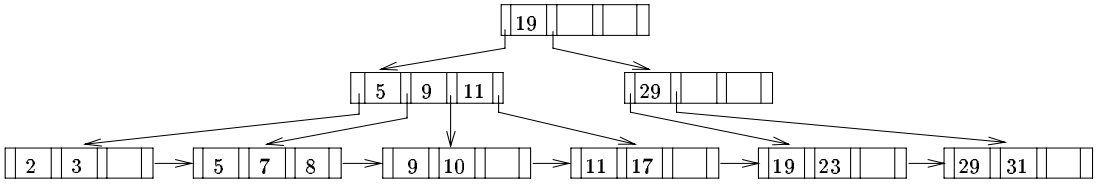
Insert 9:



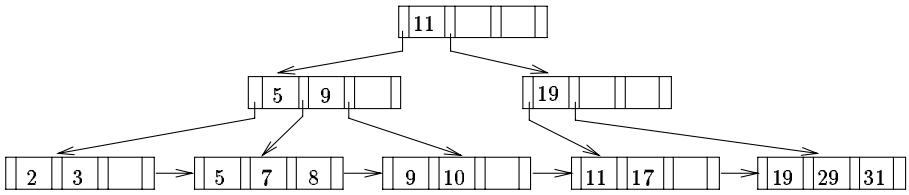
Insert 10:



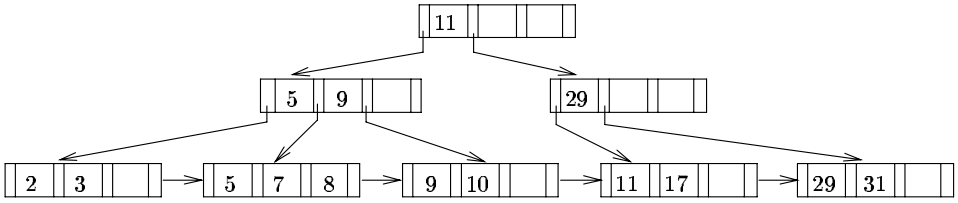
Insert 8:



Delete 23:

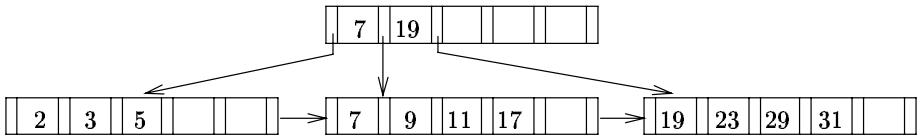


Delete 19:

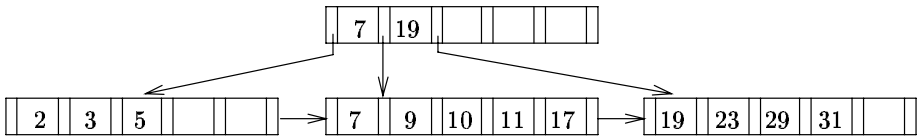


- With structure 0.b:

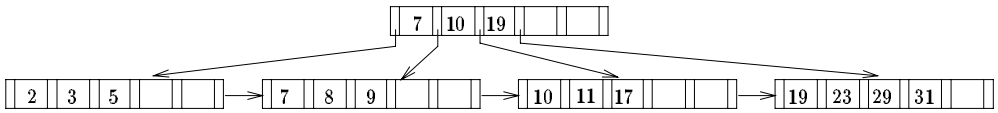
  Insert 9:



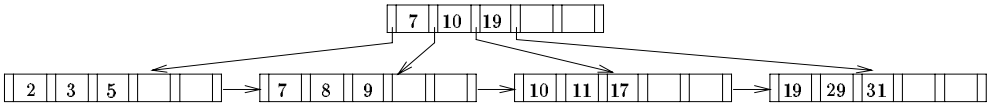
  Insert 10:



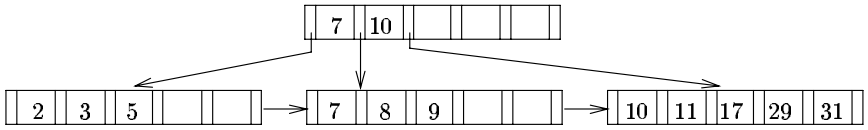
  Insert 8:



  Delete 23:



  Delete 19:



- With structure 0.c: