Today’s Outline

• Announcements
  › Assignment #2 due Fri, April 19 (TODAY) at the BEGINNING of lecture
  › Assignment #3 is a programming project.

• Today’s Topics:
  › B-Trees (Weiss 4.7 and my own details, which are NOT in the text)

B-Trees

B-Trees are multi-way search trees commonly used in database systems or other applications where data is stored externally on disks and keeping the tree shallow is important.

A B-Tree of order M has the following properties:

1. The root is either a leaf or has between 2 and M children.
2. All nonleaf nodes (except the root) have between \( \lceil M/2 \rceil \) and M children.
3. All leaves are at the same depth.

All data records are stored at the leaves. Internal nodes have "keys" guiding to the leaves. Leaves store between \( \lceil L/2 \rceil \) and L data records, where L can be equal to M (default) or can be different.

B-Tree Details

Each (non-leaf) internal node of a B-tree has:

› Between \( \lceil M/2 \rceil \) and M children.
› Up to \( M-1 \) keys \( k_1 < k_2 < \ldots < k_{M-1} \)

Keys are ordered so that:

\( k_1 < k_2 < \ldots < k_{M-1} \)

Properties of B-Trees

Children of each internal node are "between" the items in that node.

- All keys in \( T_i \) must be between keys \( k_{i-1} \) and \( k_i \)
- \( k_{i-1} \) is the smallest key in \( T_i \)
- All keys in first subtree \( T_1 \) < \( k_{i-1} \)
- All keys in last subtree \( T_M \) = \( k_{M-1} \)

B-Tree Nonleaf Node

- The Ks are keys
- The Ps are pointers to subtrees.
Detailed Leaf Node Structure (B+ Tree)

\[ K[1] \rightarrow R[1] \rightarrow \ldots \rightarrow K[q-1] \rightarrow R[q-1] \rightarrow \text{Next} \]

- The Ks are keys (assume unique).
- The Rs are pointers to records with those keys.
- The Next link points to the next leaf in key order (B+-tree).

4/19/2013 B-Trees 7

Searching in B-trees

- B-tree of order 3: also known as 2-3 tree (2 to 3 children)
- Examples: Search for 9, 14, 12
- Note: If leaf nodes are connected as a Linked List, B-tree is called a B+ tree – Allows sorted list to be accessed easily

4/19/2013 B-Trees 8

Searching a B-Tree T for a Key Value K

Find(ElementType K, Btree T) {
  B = T;
  while (B is not a leaf) {
    find the Pi in node B that points to
    the proper subtree that K will be in;
    B = Pi;
  }
  /* Now we’re at a leaf */
  if key K is the jth key in leaf B, use the jth record pointer to find
  the associated record;
  else /* K is not in leaf B */ report failure;
}

4/19/2013 B-Trees 9

Inserting into B-Trees

- Insert X: Do a Find on X and find appropriate leaf node
  - If leaf node is not full, fill in empty slot with X
  - If leaf node is full, split leaf node and adjust parents up to root node
    - E.g. Insert 5
    - E.g. Insert 9

4/19/2013 B-Trees 10

Inserting a New Key in a B-Tree of Order M (and \( L = M \))

Insert(ElementType K, Btree B) {
  find the leaf node LB of B in which K belongs;
  if notfull(LB) insert K into LB;
  else {
    split LB into two nodes LB and LB2 with
    \( j = \lceil (M+1)/2 \rceil \) keys in LB and the rest in LB2;
    LB
    \[ K[1] \rightarrow \ldots \rightarrow K[j] \rightarrow \ldots \rightarrow K|M+1| \rightarrow \text{Next} \]
    LB2
    \[ K[1] \rightarrow \ldots \rightarrow K[j] \rightarrow \ldots \rightarrow K|M+1| \rightarrow \text{Next} \]
    \( \text{if IsNull(Parent(LB)) } \)
    CreateNewRoot(LB, K[j+1], LB2);
    else
    InsertInternal(Parent(LB), K[j+1], LB2);
  }
}

4/19/2013 B-Trees 11

Inserting a (Key,Ptr) Pair into an Internal Node

If the node is not full, insert them in the proper place and return.
If the node is already full (M pointers, M-1 keys), find the place for the new pair and split the adjusted (Key,Ptr) sequence into two internal nodes with
\( j = \lceil (M+1)/2 \rceil \) pointers and \( j-1 \) keys in the first, the next key is inserted in the node’s parent, and the rest in the second of the new pair.

4/19/2013 B-Trees 12
Example of Insertions into a B+ tree with M=3, L=2

Insertion Sequence: 9, 5, 1, 7, 3, 12

Deleting From B-Trees

- Delete X: Do a find and remove from leaf
  - Leaf underflows – borrow from a neighbor
    - E.g. 11
  - Leaf underflows and can’t borrow – merge nodes, delete parent
    - E.g. 17

Run Time Analysis of B-Tree Operations

- For a B-Tree of order M
  - Each internal node has up to M-1 keys to search
  - Each internal node has between \( \lceil M/2 \rceil \) and M children
  - Depth of B-Tree storing N items is \( O(\log_{M/2} N) \)
- Find: Run time is:
  - \( O(\log M) \) to binary search which branch to take at each node. But M is small compared to N.
  - Total time to find an item is \( O(\text{depth} \times \log M) = O(\log \text{N}) \)

What are B+ Trees heavily used for? Databases

- A relational database is conceptually a set of 2D tables.
- The columns of a table are called attributes; they are the keys.
- Each table has at least one primary key by which it can be accessed rapidly.
- The rows are the different data records, each having a unique primary key.
- B+ trees are one very common implementation for these tables.

Creating a table in SQL

```sql
create table Company
(cname varchar(20) primary key,
country varchar(20),
oe_employees int,
for_profit char(1));
```

insert into Company values ('GizmoWorks', 'USA', 20000,'y');
insert into Company values ('Canon', 'Japan', 50000,'y');
insert into Company values('Hitachi', 'Japan', 30000,'y');
insert into Company values('Charity', 'Canada', 500,'n');
create table Company
  (cname varchar(20) primary key,
   country varchar(20),
   no_employees int,
   for_profit char(1));

Queries

• select * from Company;
• select cname from Company
  where no_employees > 50;
• select cname, country from Company
  where no_employees < 50 AND
  for_profit = "y";

create table Product
  (pname varchar(20) primary key,
   price float,
   category varchar(20),
   manufacturer varchar(20) references
   Company);

Another Table

A JOIN query uses both tables

SELECT DISTINCT cname FROM Product P1, Product P2, Company
WHERE country = 'Japan'
AND P1.category = 'gadget'
AND P2.category = 'photography'
AND P1.manufacturer = cname
AND P2.manufacturer = cname;

Requires retrievals according to country attribute and restricted to
category attribute and then further constrained.....
Needs a database course.

Summary of Search Trees

• Problem with Binary Search Trees: Must keep tree balanced to
  allow fast access to stored items
• AVL trees: Insert/Delete operations keep tree balanced
• Splay trees: Repeated Find operations produce balanced trees
  Multi-way search trees (e.g. B-Trees):
    › More than two children per node allows shallow trees; all
      leaves are at the same depth.
    › Keeping tree balanced at all times.
    › Excellent for indexes in database systems.