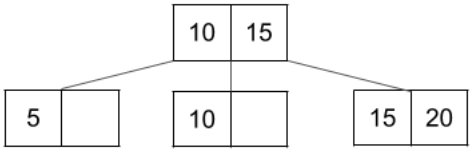
CS W186 - Spring 2020
Guerrilla Section 2
B+ Trees, Sort/Hash, Buffer Management, and Relational Algebra

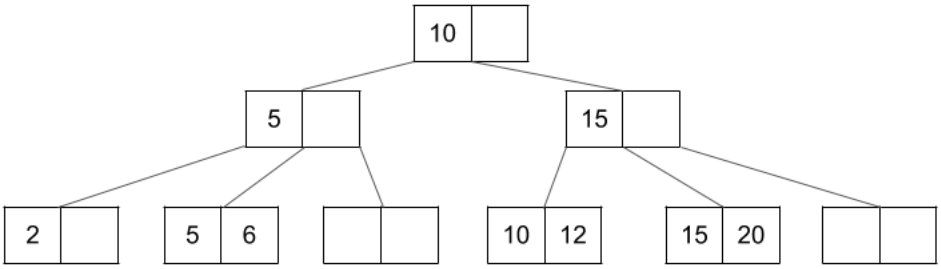
Sunday, February 23, 2020

1 B+ Trees

Given the following (degree $d = 1$) B+ tree:



(a) Draw what the tree looks like after adding 2, 6, and 12 in that order.



For the following questions, consider the tree directly after 2, 6, and 12 have been added.

(b) What is the maximum number of inserts we can do without changing the height of the tree?

Answer: **11 inserts**. The maximum number of keys for a fixed height h is given by $2d \cdot (2d+1)^h$. We have $d = 1$ from the question, and $h = 2$ (we must remember h is the number of non-root layers). Plugging these numbers in gives us $2 * 3^2 = 18$.
Now, we already have 7 keys in the tree, so we can insert $18 - 7 = 11$ more.

(c) What is the minimum number of inserts we can do that will change the height of the tree?

Answer: **4 inserts** (e.g. 16, 17, 18, 19).
The idea is to repeatedly insert into the fullest nodes; this is hopefully apparent from understanding how and when B-trees split.

2 More B+ Trees

Consider the following schema:

```
CREATE TABLE FineWines (
    name CHAR(20) NOT NULL,
    years_aged INTEGER NOT NULL,
    price INTEGER NOT NULL
);
```

Suppose that we have built an index over `<years_aged, price>`, and suppose this index is two levels deep (in addition to the root node), and data is stored by *list of references* in separate data pages, each of which can hold hundreds of records.

(a)
```
SELECT * FROM FineWines
WHERE years_aged = 5
  AND price = 100
```

What is the worst case number of page I/Os to execute this query on an unclustered index if there is 1 matching record? 2 matching records? 3 matching records?

Answer: **4, 5, 6 page I/Os**.
As the B-tree is two levels deep, we must do 3 page reads to traverse the index tree: root node, layer 1 node, layer 2 (leaf) node. As the query is an *exact* index key match, and we are storing by *list of references*, all pointers to records for this key will be on the same leaf node.
However, since the index is unclustered, each record might be stored on its own page. Thus, we would need 1 additional page read for each matching record.

(b) What is the worst case number of page accesses to execute this query on a clustered index if there is 1 matching record? 2 matching records? 3 matching records?

Answer: **4, 5, 5 page I/Os**.
As before, the tree index traversal is 3 page I/Os. Since the index is now clustered, records are stored next to each other on pages.
However, there is no guarantee that they won't cross a page boundary. So, two matching records might be on two neighbouring pages: at the end of one, and at the beginning of another. Thus we need up to 2 additional page reads for ≥ 2 matching records.

(c)
```
SELECT * FROM FineWines
WHERE price = 100
```

What is the worst case number of page I/Os to execute this query if there are 150 data pages?

Answer: **150 page I/Os**.
Since this query *does not match* the available index, we just have to read every page.

(d)
```
DELETE FROM FineWines
WHERE years_aged = 1
  AND price = 5
```

Suppose this query deletes exactly one record. How many page I/Os are needed to execute this query? (Assume no tree rebalancing is required.)

Answer: **6 page I/Os**.
(3) Read index pages (traversing index tree). (1) Read data page. (1) Write data page. (1) Write leaf page in index (to delete the key from the index).

3 Sorting and Hashing

Suppose the size of a page is 4KB, and the size of the memory buffer is 1 MB (1024 KB).

1. We have a relation of size 800 KB. How many page IOs are required to sort this relation?

   Answer: **400**
   200 to read in, 200 to write out. Since the relation is small enough to completely fit into the buffer we only need to read it in, sort it (no I/Os required for sorting), then write the sorted pages back to disc.

2. We have a relation of size 5000 KB. How many page IOs are required to sort this relation?

   Answer: **5000.** 2 passes. 5000 KB with 4KB per page means 1250 pages are needed to store the relation. We have 1024 / 4 = 256 pages in our buffer. Number of Passes = $1 + \lceil \log_{255} \lceil 1250/256 \rceil \rceil = 2$

3. What is the size of the largest relation that would need two passes to sort?

   Answer: **261,120 KB.** (255 * 256 pages).

4. What is the size of the largest relation we can possibly hash in two passes (i.e. with just one partitioning phase)?

   Answer: **261,120 KB.**

5. Suppose we have a relation of size 3000 KB. We are executing a `DISTINCT` query on a column `age`, which has only two distinct values, evenly distributed. Would sorting or hashing be better here, and why?

   Answer: **Hashing**, which allows us to remove duplicates early on and potentially improve performance (in this case, we might be able to finish in 1 pass, instead of 2 for sorting).

6. Now suppose we were executing a `GROUP BY` on `age` instead. Would sorting or hashing be better here, and why?

   Answer: **Sorting** because hashing won't work; each partition is larger than memory, so no amount of hash partitioning will suffice.

4 External Sorting

Assume our buffer pool has 8 frames. In this question, we'll externally sort a 500 page file.

1. How many passes will it take to sort this file?

   Answer: **4 passes.** Number of Passes = $1 + \lceil \log_7 \lceil 500/8 \rceil \rceil = 4$

2. Given the number of passes you calculated in 4.1, how many I/Os are necessary to externally sort the file?

   Answer: **4000 I/Os.** 2 * Number of Pages * Passes = 2 * 500 * 4 = 4000 I/Os.

3. What is the minimum number of additional frames needed to reduce the number of passes found in 4.1 by 1?

   Answer: **1 additional frame.** Given that we had 4 passes in 2.1, we need to calculate how many pages it will take to sort the relation in 3 passes.
   $B(B-1)^2 >= 500$
   B = 9 frames. 9 - 8 = 1 additional frame. I/Os.

4. What is the minimum number of additional frames needed to sort the file in one pass?

   Answer: **492 pages.** If we can fit the entire table into the buffer, our initial sorting pass will sort the table. Therefore 500 - 8 = 492 pages.

5 Hashing

1. Suppose the size of each page is 4KB, and the size of our memory buffer is 64KB. What would be the I/O cost of hashing a file of 128 pages, assuming that the first hash function creates 2 partitions of 32 pages, and all other partitions are uniformly partitioned?

   Answer: **721 pages.**
   There are B = 64KB/4KB = 16 pages of RAM.
   We read the 128 pages of the file into B-1=15 partitions.

   - Partition 1: 32 pages
   - Partition 2: 32 pages
   - Partitions 3-15: $\lceil (128-64)/13 \rceil = 5$ pages
   - We write the 32 + 32 + 13*5 = 129 pages to memory.

   To recursively partition the partitions of 32 pages:

   - We read in 32 pages for each into B-1=15 partitions.
   - Partition size: $\lceil 32/15 \rceil = 3$ pages
   - We write out 15 partitions of 3 page each: 3*15 = 45.

   In the conquer phase:

   - From the first divide phase, we have 13 partitions that fit into memory, of 5 pages each.
   - From each of the recursive partitions, we have 15 partitions that fit into memory, of 3 pages each.
   - From the conquer phase, this is a total of (65+45+45) = 155 I/O's for reading, and (65+45+45) = 155 I/O's for writing.

   This gives us a total of (128 + 129) + 2*(32 + 45) + (155+155) = 721 pages.

6 Buffer Management

We're given a buffer pool with 4 pages, which is empty to begin with. Answer the following questions given this access pattern:

A B C D E B A D C A E C

1. What is the hit rate for MRU?

   Answer: **4/12.**

A						*	D				
	B				*						
		C						*	A		
			D	E						*	C

2. In order of when they were first placed into the buffer pool, what pages remain in the buffer pool after this sequence of accesses?

   Answer: **B, D, A, C**

3. What is the hit rate for clock?

   Answer: **5/12. You can see a step-by-step walkthrough by clicking on this sentence.**

4. Which pages are in the buffer pool after this sequences of accesses?

   Answer: **A, C, D, E.**

5. Which pages have their reference bits set?

   Answer: **A, C, E.**

6. Which page is the arm of the clock pointing to?

   Answer: **A. On a page hit, the arm doesn't move.**

7 Relational Algebra

1. Consider two relations with the same schema: R(A,B) and S(A,B). Which one of the following relational algebra expressions is not equivalent to the others?

   (A) $\pi_{\text{R.A}}((R \cup S) - S)$

   (B) $\pi_{\text{R.A}}(R) - \pi_{\text{R.A}}(R \cap S)$

   (C) $\pi_{\text{R.A}}(R - S) \cap \pi_{\text{R.A}}(R)$

   (D) They are all equivalent.

   Answer: **b.** Consider the counter example where R(A, B) = (1,2), (1,3) and S(A, B) = (1,2). a and c will result in a size 1 set, while b will result in a size 0 set.

2. The CS 186 TAs have decided to go into real estate! All of the information we need is described by the following schema, where the primary key of each table is underlined:

```
Homes(home_id int, city text, bedrooms int, bathrooms int, area int)
Transactions(home_id int, buyer_id int, seller_id int,
             transaction_date date, sale_price int)
Buyers(buyer_id int, name text)
Sellers(seller_id int, name text)
```

   Write the relational algebra plan to find the duplicate-free set of id's of all homes in Berkeley with at least 6 bedrooms and at least 2 bathrooms that were bought by "Bobby Tables".

   Answer: $\pi_{\textbf{home\_id}}($
   $\sigma_{\textbf{name='Bobby Tables'}}($
   $((\sigma_{\textbf{city='Berkeley' AND bedrooms}\geq\textbf{6 AND bathrooms}\geq\textbf{2}}\ \textbf{Homes})$
   $\bowtie \textbf{Transactions}) \bowtie \textbf{Buyers}))$

8 Hints for Question 3: Sorting and Hashing

1. Convert all numbers to pages, and then apply the equation you learned in class.
2. Same.
3. Again, convert all numbers to pages.
   The second pass merges how many runs?
   For the second pass to complete the sort, this must be enough to merge all the runs created by the first sort. How big does the relation have to be for the first sort to create that many runs?
4. Similar to the previous question, but think about the hashing protocol instead.
5. Which protocol benefits from removing duplicates?
6. `GROUP BY` does not allow us to remove duplicates. Are both protocols capable of handling such large numbers of duplicates?