1 MCQs

1. T/F: While B-Trees may have more than one copy of the same search key, B+Trees can only have one copy for each search key.
2. T/F: Cryptographic hashes are preferred for hash tables.
3. T/F: Nested loop joins are preferred when the number of desired tuples is small.
4. T/F: Under specific conditions, a bucket in an extendible hash table can have a local depth larger than the global depth.
5. T/F: $\pi_A(R_1 \cup R_2) = \pi_A(R_1) \cup \pi_A(R_2)$
6. T/F: $\sigma_A(\sigma_B(R)) = \sigma_B(\sigma_A(R))$

1.1 Solution

1. False. If a B+Tree is built on a non-unique key, it can still have duplicate keys.
2. False. Cryptographic hashes are too slow and may not be uniformly distributed over the output space.
3. True. There is no preprocessing step, so we can terminate early.
4. False. Local depth can never be larger than global depth.
5. True. This is an identity.
6. True. Selects are commutative (think of it like an “and”).

2 Short Answers

1. Consider an extendible hash table of depth d . Why might we want to use the d lowest bits in a hash function rather than the d highest bits?
2. Explain two scenarios where column stores are more effective (i.e. use less compute and/or storage).
3. When is $\sigma_{A \wedge B}(R_1 \bowtie R_2) = \sigma_A(R_1) \bowtie \sigma_B(R_2)$?

2.1 Solution

1. If we use the d lowest bits, when we split, we only have to split among the old bucket and the new bucket. If we use the larger bits, we may have to split among other buckets, which is messy.
2. Column stores are better when we care about compression, or when we only care about a few (one) of the columns.
3. When A only filters elements of R_1 and B only filters elements of R_2.

3 Question 1

Consider the following BTree with max degree = 4:

Next, answer the following questions. Note that the questions do not build on one another; the subject of all questions is the tree above, not the tree resulting from previous questions.

1. Explain what happens when 0 is inserted.
2. Explain what happens when 11 is inserted.
3. Explain what happens when 9 is removed.
4. Explain what happens when 1 is removed.

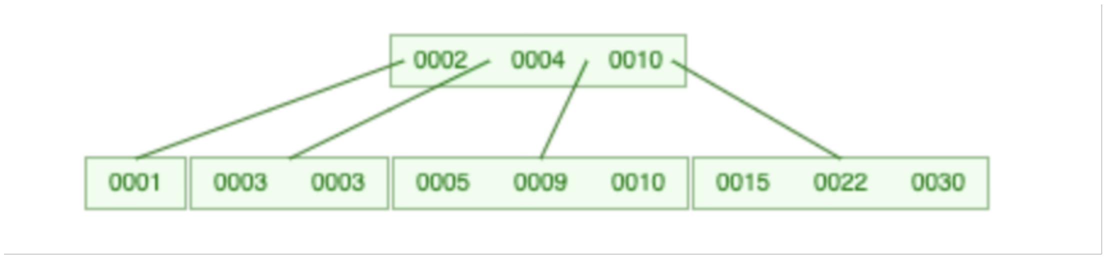

Figure 1: Q2

3.1 Solution

1. Explain what happens when 0 is inserted.

When 0 is inserted, it goes into the left-most leaf node. Since the leaf node is not full, it does not split.

2. Explain what happens when 11 is inserted.

When 11 is inserted, it goes into the right-most leaf node. Since the leaf is now full, it split in two, and pushes one of its middle values (say, 22) up. Now, the root is full, so it also split in two, and pushes its middle value (say, 10) up. Now, we have a tree of height 3.

3. Explain what happens when 9 is removed.

When 9 is removed from the second-from-the-right leaf, the leaf is not underfull, so nothing happens.

4. Explain what happens when 1 is removed.

When 1 is removed, a leaf is now empty, so it will attempt to borrow from a neighbor. 2 comes down and becomes a singleton leaf, and 3 is pushed up.

4 Question 2

Given the existing extendable hashing structure, answer the following questions:

1. What should be the value of a in the red box?
2. What should be the value of b in the green box?
3. Describe a potential indexing scheme and hashing function that might have been used in this case.
4. Describe the changes that happen to the current structure upon the insertion of the key 16. You may describe the changes in plain English or you may draw the resulting structure. Show your steps.

4.1 Solution

1. What should be the value of a in the red box?

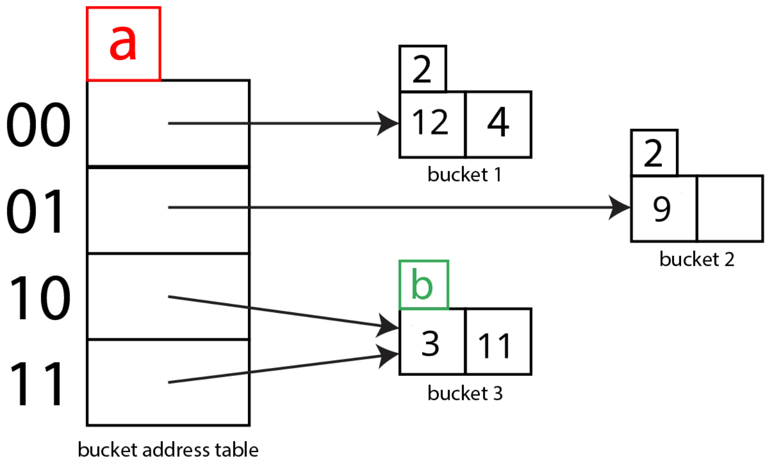

Figure 2: Q2

Global depth is 2.

2. What should be the value of b in the green box?

Local depth is 1.

3. Describe a potential indexing scheme and hashing function that might have been used in this case.

Extendible hashing, `d` least significant digits.

4. Describe the changes that happen to the current structure upon the insertion of the key 16. You may describe the changes in plain English or you may draw the resulting structure. Show your steps.

The new value would be added to bucket 1, which would then overflow. Since bucket 1 has the same depth as the global depth, the entire directory doubles in size, and a new bucket is created to accomodate values that end in 100. The rest of the new directory slots point to buckets that already exist, but with a lower local depth.

5 Question 3

Suppose that the relation R(a, b, c) and S(c, d, e) have the following properties: R has 12,000 tuples, S has 2,000 tuples. A block can fit either 100 tuples from relation R, or 20 tuples from relation S. Determine the optimal join method for processing the following query based on the number of block accesses:

```
select *
from R, S
where R.c = S.c;
```

Assume the following: - Main memory can fit 16 blocks. - There are no indices available. - S is the outer relation, and R is the inner relation. - The system only has the following join methods: nested loop join and sort-merge.

5.1 Solution

We'll analyze the cost for both nested loop join and sort-merge, then compare the costs to make a decision. Before anything else, we note that using the current block structure, R essentially has 12,000/100 = 120 blocks, and S has 2000/20 = 100 blocks.

Nested loop join: We notice that the buffer can only fit 16 blocks, which is a lot less than we need to fit the entirety of either relation in memory. So, we will end up having to do 120 block accesses (R is outer relation) plus 12,000 * 100 = 1,200,000 more accesses (number of blocks in S times number of tuples in R). This comes up to a cost of 1,200,120.

Sort-merge: The cost of sort-merge is the cost to sort R plus the cost to sort S plus the number of blocks in R plus the number of blocks in S. The cost to sort is kinda gross but it's logarithmic (2 * 120 * log_16(120 / 16) + 1 = about 175 for R, 2 * 100 * log_16(100 / 16) + 1 = about 133 for S). Thus, we add: 175 + 133 + 120 + 100 = 528, which seems obscenely small, so this value is almost certainly wrong, but I'm also pretty sure this will end up being cheaper than nested loop join anyways since sorting is not a quadratic procedure, unlike two nested loops.

So, we choose sort-merge since it's cheaper!

Feedback

As this is a new course, we appreciate any feedback you have for us! If you enjoyed this assignment, hated this assignment, or have other thoughts to share, please do so here!