1 Question 1

Please provide short answers to the following questions:

1. What is the difference between static and dynamic hashing?
2. What problem does dynamic hashing address that static hashing does not?
3. Why is it important to keep track of the local depth in extendible hashing?
4. In extendible hashing, what happens when try to insert into a bucket that is full? Please consider all cases.
5. Why do we want a hash function to uniformly distribute our input space?
6. Would a cryptographic hash like SHA-256 be good for a hash table? Why or why not?

2 Question 2

Given bucket sizes of 2, a hashing function that uses the lowest d bits (d = depth), and an initial hash of 0 bits, hash 2, 3, 5, 8, 12, 17, 19, 21, 27, 31 in the given order and draw a diagram of the final result, showing all relevant information, (bucket lookup table, corresponding hash keys, pointers, buckets, entries, overflow buckets, local and global depth values, next pointers, level, and anything else relevant), using each of the following hashing schemes: 1. Extendible hashing. 2. Linear hashing.

3 Question 3

Consider relations r1(A, B, C) and r2(C, D, E) with the following properties: r1 has 20,000 tuples, r2 has 45,000 tuples, 25 tuples of r1 fit on one block, and 30 tuples of r2 fit on one block. Assume we have M pages of memory; i.e. we can hold M blocks in memory at a time. Assume that an in memory hash table can be computed for r2 and that neither r1 nor r2 are sorted on their join key. Assuming we do not need to leave enough space for disk output, estimate the number of block transfers required, using each of the following join strategies for r1 and r2: 1. Block nested-loop join. 2. Merge join. 4. Hash join.

4 Question 4

Consider a Bloom Filter, which is a very fancy hash table. We can use a Bloom Filter to quickly find if an entry may exist or definitely does not exist.

We’ll make a small and simple Bloom Filter of only 8 bits. Since we’re starting empty, it will look like this:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Now, to show that an entry exists, we can hash the key for that entry and get a value. We assign a 1 at that corresponding slot in the Bloom Filter. For example, if we have an entry “Buzzwell”, and h(“Buzzwell”) = 3, then the we can note that “Buzzwell” exists by updating the Bloom Filter to look like this:
1. Add the rest of the bees to our Bloom Filter. What does the final Bloom Filter look like?

<table>
<thead>
<tr>
<th>name</th>
<th>h(name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barry Benson</td>
<td>6</td>
</tr>
<tr>
<td>Adam Flayman</td>
<td>2</td>
</tr>
<tr>
<td>Janet Benson</td>
<td>7</td>
</tr>
<tr>
<td>Lou Lo Duca</td>
<td>6</td>
</tr>
<tr>
<td>Bob Bumble</td>
<td>1</td>
</tr>
</tbody>
</table>

2. How do we know that “Vanessa Bloome”, where h(“Vanessa Bloome”) = 5, definitely does not exist in our data?

3. Why do we know that “Barry Benson” might exist in our data?

4. Suppose we utilized a Bloom Filter for our database of bees. Thus, for an EXISTS query, we consult the Bloom Filter. Then, if necessary, we lookup in the database itself.
   a. What is the minimum number of database lookups that will occur? In what situation will this happen? Why will this happen?
   b. What is the maximum number of database lookups that will occur? In what situation will this happen? Why will this happen?

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!