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 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?

1.1 Solution

1. What is the difference between static and dynamic hashing?

In static hashing, the hash function is fixed, meaning that a particular element will always be mapped to the same bucket. By corrolary, this means the hash table never grows. In dynamic hashing, the hash function changes as more elements are added, making the buckets grow and shrink according to the number of records in them.

2. What problem does does dynamic hashing address that static hashing does not?

Dynamic hashing scales. As a static hash gets full, eventually each bucket holds a very large number of items, meaning large overflow chains and basically linear search. As a dynamic hash fills up, we continuously split buckets to ensure no long overflow chains, therefore never doing an inefficient linear search.

3. Why is it important to keep track of the local depth in extendible hashing?

We need to know the local depth in each bucket so we know what to do when it overflows. If the local depth is less than the global depth, we can split this bucket in two without needing to consider other buckets. Otherwise, if it is equal to the global depth, we need to double the entire directory.

4. In extendible hashing, what happens when try to insert into a bucket that is full? Please consider all cases.

The bucket will overflow, and we will split the bucket, incrementing its local depth counter by one and distributing the keys from the old bucket into the two new buckets by search key. If the local depth counter now exceeds the global depth counter, the global table will extend by doubling in size, copying all other pointers to the other buckets.

5. Why do we want a hash function to uniformly distribute our input space?

If a hash function has a bias towards certain values, then certain buckets will fill up and overflow more often than others. Depending on the hashing scheme, this can trigger unnecessary and unwanted splits of the hash directory, leading to wasted space and a loss of efficiency. For example, consider an adversarial workload that always inserts hashed values that have the same last 10 bits. Then, an extendible hash table might split many times even though only one bucket is ever triggering these splits.

6. Would a cryptographic hash like SHA-256 be good for a hash table? Why or why not?

No, for two reasons. Cryptographic hashes are not uniformly distributed, and they are slow. Also, there is no need to compute a reversible hash in a hash table since we're storing the data anyways.

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.

2.1 Solution

Link

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.

3.1 Solution

*Note: The point of this problem is **estimation**, it's okay if your answer is slightly off.*

1. Block nested-loop join.

In block-nested loop join, each block in r1 will be read in once, and each block in r2 will be read in `n` times, where `n` is the number of blocks in r1.

r1 can be broken up into 20,000 / 25 = 800 blocks. r2 can be broken up into 45,000 / 30 = 1500 blocks. Moreover, we can hold `M-1` blocks in r1 at a time. Therefore, the number of block transfers necessary, assuming r1 is the outer relation, is 800 + 1500 * 800 / (M-2). If we take r2 to be the outer relation, then we have 1500 + 800 * 1500 / (M-2).

2. Merge join.

Assuming that r1 and r2 are not initially sorted on the join key, the total sorting cost inclusive of the output is Bs = 1500 (2 ceil(log_{M-1}(1500/M)) + 1) + 800 (2 ceil(log_{M-1}(800/M)) + 1) disk accesses.

If we are sorted, we don't have to sort, and never have to double back, since we're exploiting the fact that the relations are sorted and crawling through each one, matching as we go. Thus, we only need 800 + 1500 = 2300 block transfers.

3. Hash join.

Since r1 is smaller, we use it as the build relation and r2 as the probe relation. If M > 800/M, i.e. no need for recursive partitioning, then the cost is 3 (1500 + 800) = 6900 disk accesses, else the cost is 2 (1500 + 800) ceil(log_{M-1}(800) - 1) + 1500 + 800 disk accesses.

4 Question 4

Consider a Bloom Filter, which is a very fancy `has` 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:

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

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:

0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0

1. Add the rest of the bees to our Bloom Filter. What does the final Bloom Filter look like?

name	h(name)
Barry Benson	6
Adam Flayman	2
Janet Benson	7
Lou Lo Duca	6
Bob Bumble	1

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?

4.1 Solution

1. Adding Bees

0	1	2	3	4	5	6	7
0	1	1	1	0	0	1	1

2. Nonexistence

We know that "Vanessa Bloome" is not in our data because the hash of the entry's value is marked as a 0. No other entry can be mapped to the same slot in the Bloom Filter.

3. Possible Existence

"Barry Benson" might exist in our data because the hash of the value has been set to 1. We know something with the hashed value exists, but we don't know what the value actually was. It's possible there was a collision.

4. Lookups

a. The minimum number of database lookups will be 0 when the entry is **not** in the database, according to the Bloom filter. We know that we don't even have to waste time looking in the database if we know for sure it's not there.

b. The maximum number of database lookups will be 1 when the entry **might be** in the database, according to the Bloom filter. We will have to confirm that the value is actually in the database.

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!