CHAPTER 24 Advanced Indexing Techniques

Solutions for the Practice Exercises of Chapter 24

Practice Exercises

24.1

Answer:
The biggest difference is that buffer trees require more random I/O for insert operations as compared to LSM trees, whereas LSM trees perform more sequential I/O operations. For Big Data settings where data is resident on magnetic disks, random I/O is significantly more expensive than sequential I/O, and thus LSM trees are preferred.

24.2

Answer:
A larger array requires fewer add operations, but if you increase the size of the array beyond the cache size, there will be an increased overhead for array element access. Thus, the array should be sized to fit in cache. With L1 cache sizes of a few megabytes, an array indexed by 16 to 24 bytes would work well.

24.3

Answer:
FILL

24.4

Answer:
Idea: Priortize search of children nodes based on the distance of the bounding box fom the query point (the distance is 0 if the bounding box contains the query point). Maintain a priority queue of nodes based on the distance. When you find an answer at distance d and also all nodes whose bounding boxes

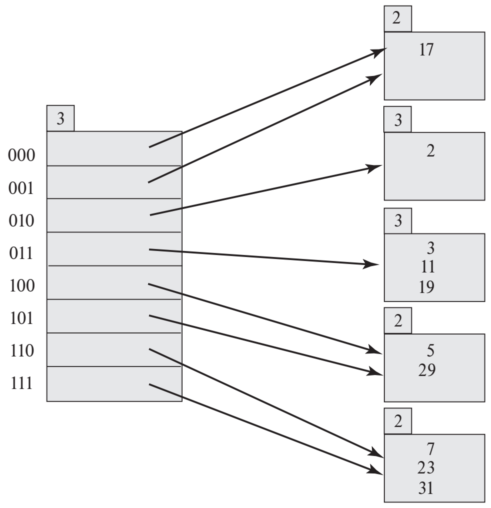


Figure 24.101 The extendable hash structure for Exercise 24.6.

are at distance *d* or closer have been explored, the search can stop since any unexplored items are at distance greater than *d*.

24.5

Answer:
Every pair of bounding boxes at each level that intersect need to be explored further; when exploring a pair, all intersecting child pairs are generated and explored further, unless they are leaves in which case the answer can be generated.

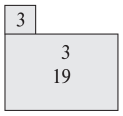
24.6

Answer:
The extendable hash structure is shown in Figure 24.101

24.7

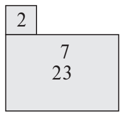
Answer:

a. Delete 11: From the answer to Exercise 24.1, change the third bucket to:

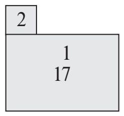


At this stage, it is possible to coalesce the second and third buckets. Then it is enough if the bucket address table has just four entries instead of eight. For the purpose of this answer, we do not do the coalescing.

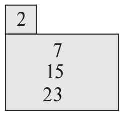
b. Delete 31: From the answer to Exercise 24.1, change the last bucket to:



c. Insert 1: From the answer to Exercise 24.1, change the first bucket to:



d. Insert 15: From the answer to Exercise 24.1, change the last bucket to:



24.8

Answer:

Let i denote the number of bits of the hash value used in the hash table. Let **bsize** denote the maximum capacity of each bucket. The pseudocode is shown in Figure 24.102.

Note that we can only merge two buckets at a time. The common hash prefix of the resultant bucket will have length one less than the two buckets merged. Hence we look at the buddy bucket of bucket j differing from it only at the last bit. If the common hash prefix of this bucket is not i_j, then this implies that the buddy bucket has been further split and merge is not possible.

When merge is successful, further merging may be possible, which is handled by a recursive call to *coalesce* at the end of the function.

```
delete(value $K_l$)
begin
    $j$ = first $i$ high-order bits of $h(K_l)$;
    delete value $K_l$ from bucket $j$;
    coalesce(bucket $j$);
end

coalesce(bucket $j$)
begin
    $i_j$ = bits used in bucket $j$;
    $k$ = any bucket with first $(i_j - 1)$ bits same as that
        of bucket $j$ while the bit $i_j$ is reversed;
    $i_k$ = bits used in bucket $k$;
    if($i_j \neq i_k$)
        return; /* buckets cannot be merged */
    if(entries in $j$ + entries in $k$ > bsize)
        return; /* buckets cannot be merged */
    move entries of bucket $k$ into bucket $j$;

    decrease the value of $i_j$ by 1;
    make all the bucket-address-table entries,
    which pointed to bucket $k$, point to $j$;

    coalesce(bucket $j$);
end
```

Figure 24.102 Pseudocode for delition of Exercise 24.8.

24.9

Answer:

If the hash table is currently using i bits of the hash value, then maintain a count of buckets for which the length of common hash prefix is exactly i.

Consider a bucket j with length of common hash prefix i_j. If the bucket is being split, and i_j is equal to i, then reset the count to 1. If the bucket is being split and i_j is one less than i, then increase the count by 1. It the bucket is being coalesced, and i_j is equal to i, then decrease the count by 1. If the count becomes 0, then the bucket address table can be reduced in size at that point.

However, note that if the bucket address table is not reduced at that point, then the count has no significance afterwards. If we want to postpone the reduction, we have to keep an array of counts, i.e., a count for each value of common hash prefix. The array has to be updated in a similar fashion. The bucket address table can be reduced if the i^{th} entry of the array is 0, where i is the number of bits the table is using. Since bucket table reduction is an expensive operation, it is not always advisable to reduce the table. It should be reduced only when a sufficient number of entries at the end of the count array become 0.