Class Meeting

Topic 8: Accessing and Indexing Data

CS411 Database Systems

Kevin C.C. Chang

Concept and Questions

Clarifying hard concepts and answering questions.

Why is this topic important for database systems?

Your Answers

- (X) This topic is important because it allows us to think about how the data will actually be stored on a hard drive or some other form of memory, and how we can do this efficiently.
- (X) To create more efficient searches in databases.
- (X) We have a lot of data in database system, so we need constant time look up.
- Indexing is important for efficient access of data in our database and maintaining near constant time search even as the amount of data grows.
- Indexing is important for helping make databases work in constant speed so no matter how much data is in the database it still works as quickly.

What is the hardest concept to you?

Your votes…

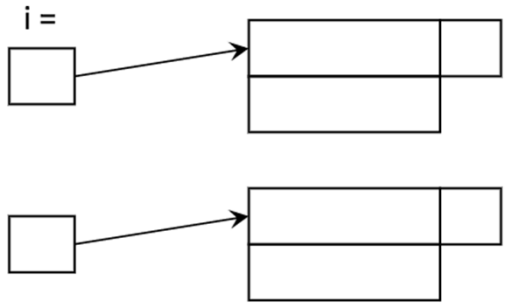

b-tree (29) bit (2) clustered (6) deletion (9)
dense (3) difference (4) duplicate (3) extensible (12)
file (2) flip (2) functions (3) growth (3) hash (62)
hashtable (4) index (21) insertion (9) keys (3) leaf (2)
linear (21) memory (2) overflow (2) sequential (2) sparse (5)
table (57) tree (39) types (2) unclustered (4) vs (4)

Problem 1

Use an extensible hash table to index the following search key values: 22, 69, 35. The hash function h(n) for integer valued search key n is h(n) = n mod 14, and each data block can hold 2 data items.

Part 1:

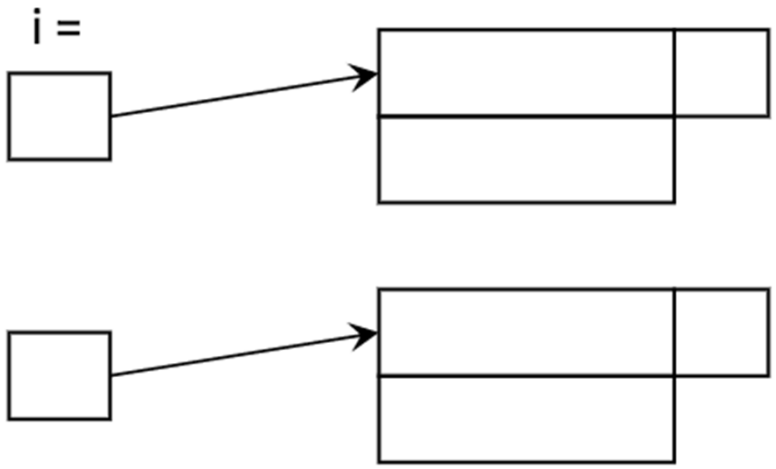
- Fill in the blanks of the empty hash table with both the buckets and the data blocks, after these three key values have been inserted. Show the keys along with their hash values in the data blocks, e.g., 22(1000). Indicate the number of bits in the hash value that are used in the buckets (the value ‘i' as discussed in the class). Also, indicate the "nub" value of each data block.



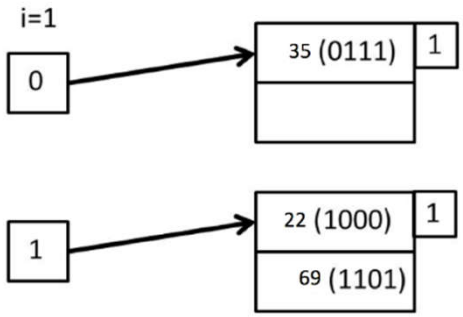
Part 2:

- We next insert key 81 into the hash table. Redraw the hash table in a similar format of Part 1. Indicate the number of bits in the hash value that are used in the buckets. Also, indicate the "nub" value of each data block.

Fill in the blanks of the empty hash table with both the buckets and the data blocks, after 22, 69, 35 have been inserted.



We next insert key 81 into the hash table.

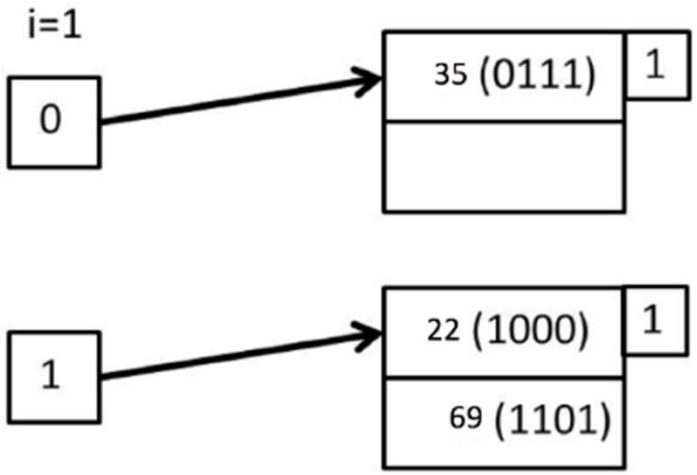


Solution

Use an extensible hash table to index the following search key values: 22, 69, 35. The hash function h(n) for integer valued search key n is h(n) = n mod 14, and each data block can hold 2 data items.

Part 1:

- Fill in the blanks of the empty hash table with both the buckets and the data blocks, after these three key values have been inserted. Show the keys along with their hash values in the data blocks, e.g., 22(1000). Indicate the number of bits in the hash value that are used in the buckets (the value ‘i' as discussed in the class). Also, indicate the "nub" value of each data block.

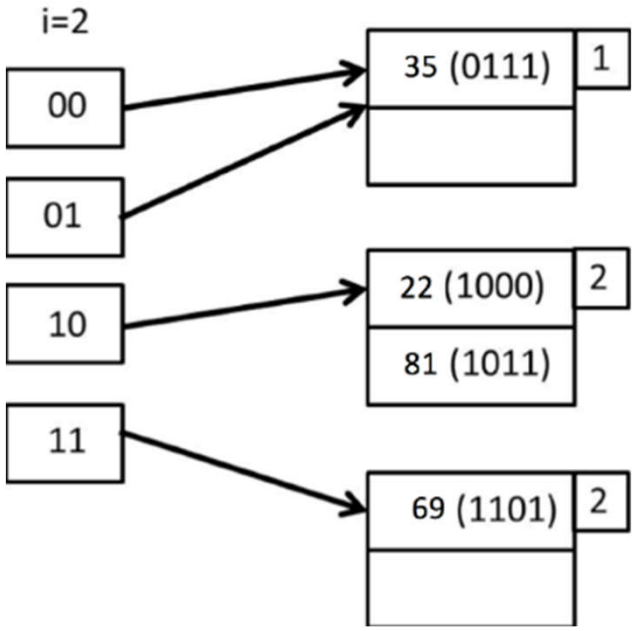


Solution

Use an extensible hash table to index the following search key values: 22, 69, 35. The hash function h(n) for integer valued search key n is h(n) = n mod 14, and each data block can hold 2 data items.

Part 2:

- We next insert key 81 into the hash table. Redraw the hash table in a similar format of Part 1. Indicate the number of bits in the hash value that are used in the buckets. Also, indicate the "nub" value of each data block.



What's the difference between this and what we learnt in CS225?
Why we would rather use extension hash table or linear hash table than simply add bits to let all data stored?

How do linear hash tables work? Why can you put entries that do not belong to that bucket into that bucket? Does adding one bucket require traversing the entire database to move entries back to the correct position? Doesn't that make it slow?

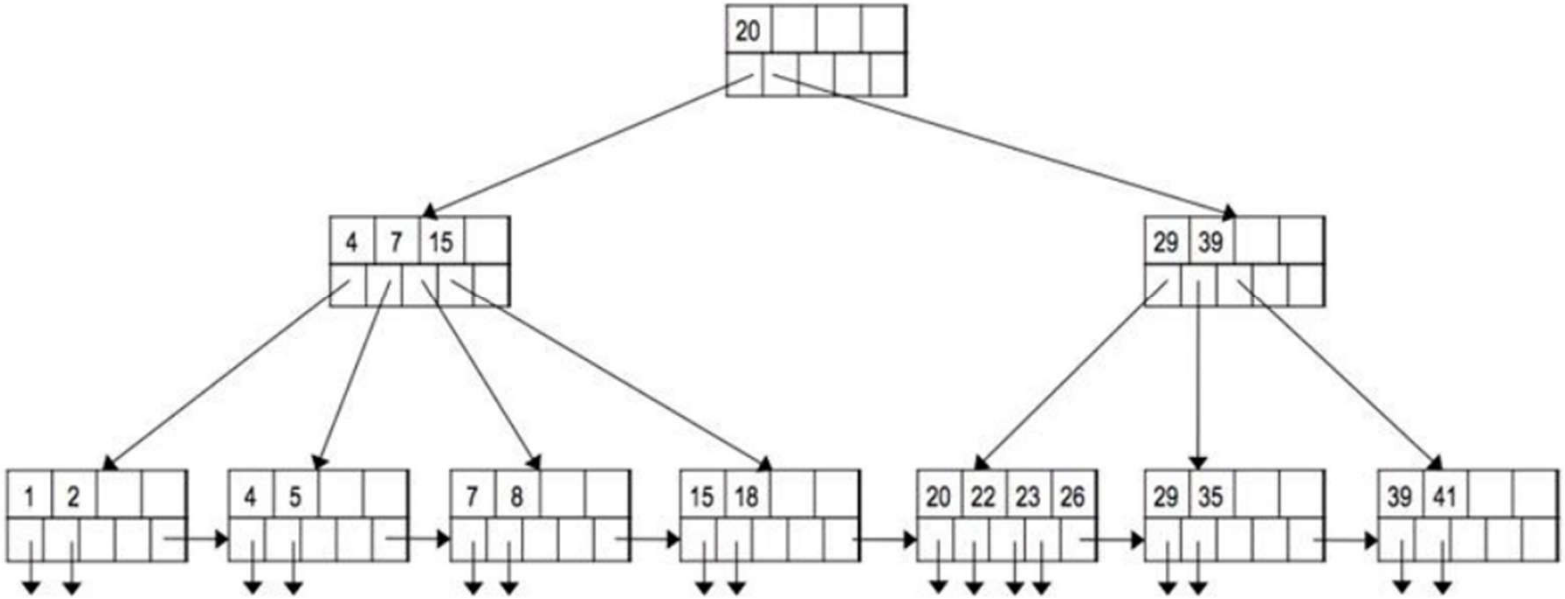
	Overflow?	When to Expand?	n: Expansion of Number Buckets	k: Key Mapping Reorganization
Extensible Hashing		When _____ Is too full.	If n = 4 (i=2) now, what can n expand to?	if you add key 100, from what old keys can the data values come from?
Linear Hashing		When _____ Is too full.	If n = 4 (i=2) now, what can n expand to? Why?	if you add key 100, from what old keys can the data values come from?

For the food of thought, can we make B+ tree, ISAM and extensive table index clustered or unclustered? Dense or sparse? Can we discuss those situations during the class?

Are there any more data structures/methods of maintaining data that we should look into in our off time? Since we have previously covered B trees and hashing in 225, we were already partially exposed to these topics which made them easier to understand. I don't really have any confusion, I was just wondering what else we might have already covered that could potentially expand into more useful data management techniques?

Problem 2

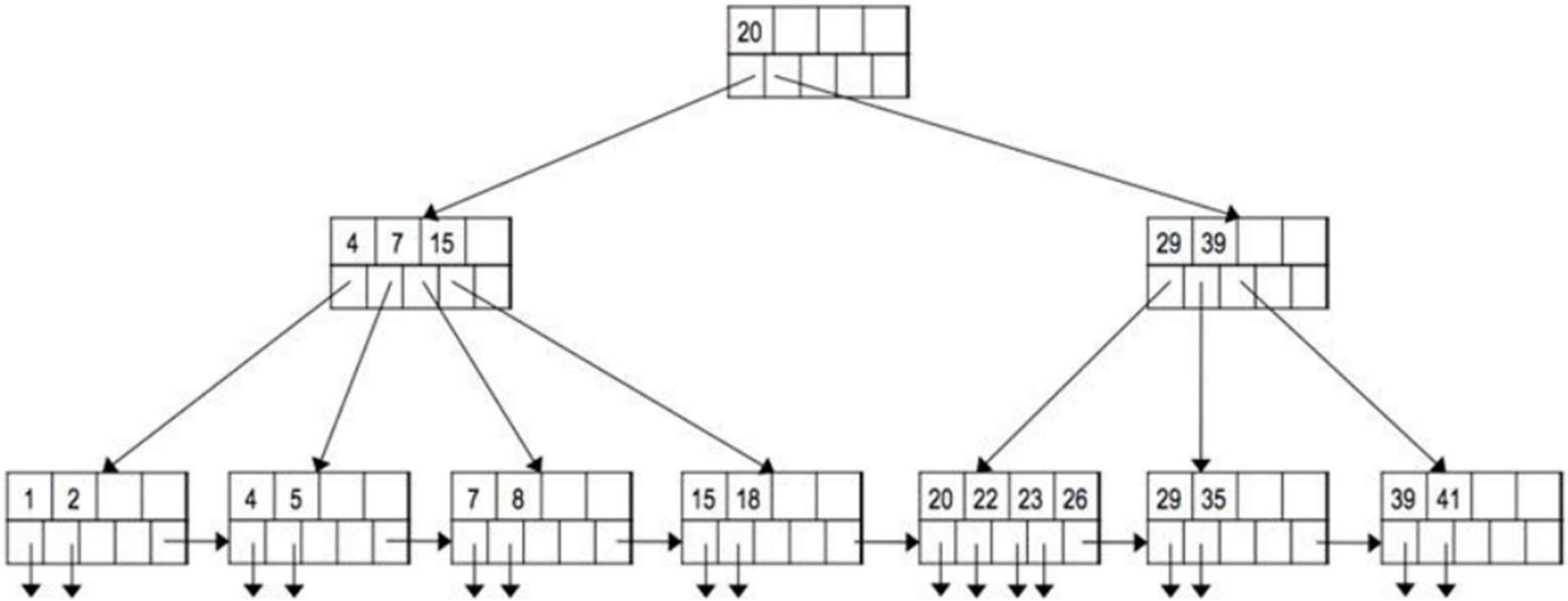
Consider the following B+ tree. Each index node can hold at most 4 keys:



1. Show the steps in looking up all records in the range 16 to 30.
2. Show the resulting tree after deleting key 39 from the original tree.
3. Show the resulting tree after inserting key 24 into the original tree.

Solution

1. Show the steps in looking up all records in the range 16 to 30.

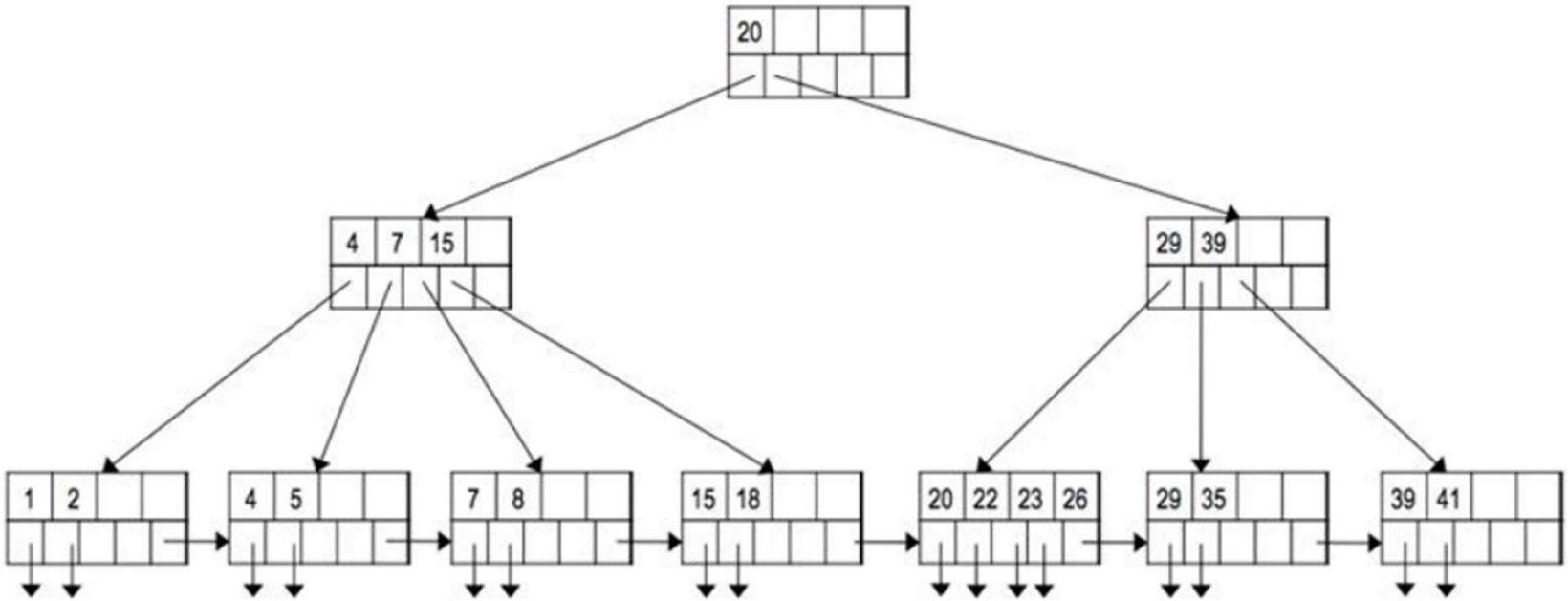


Solution:

- Step 1: At level 1, since 16 < 20, go to the 1st pointer;
- Step 2: At level 2, since 16 > 15, go to the 4th pointer;
- Step 3: Get the record with key 18, follow the chain from there to 5th leaf and get records 20, 22, 23, 26, 29;
- Step 4: Stop at 35 as 30 < 35.

Solution

2. Show the resulting tree after deleting key 39 from the original tree.

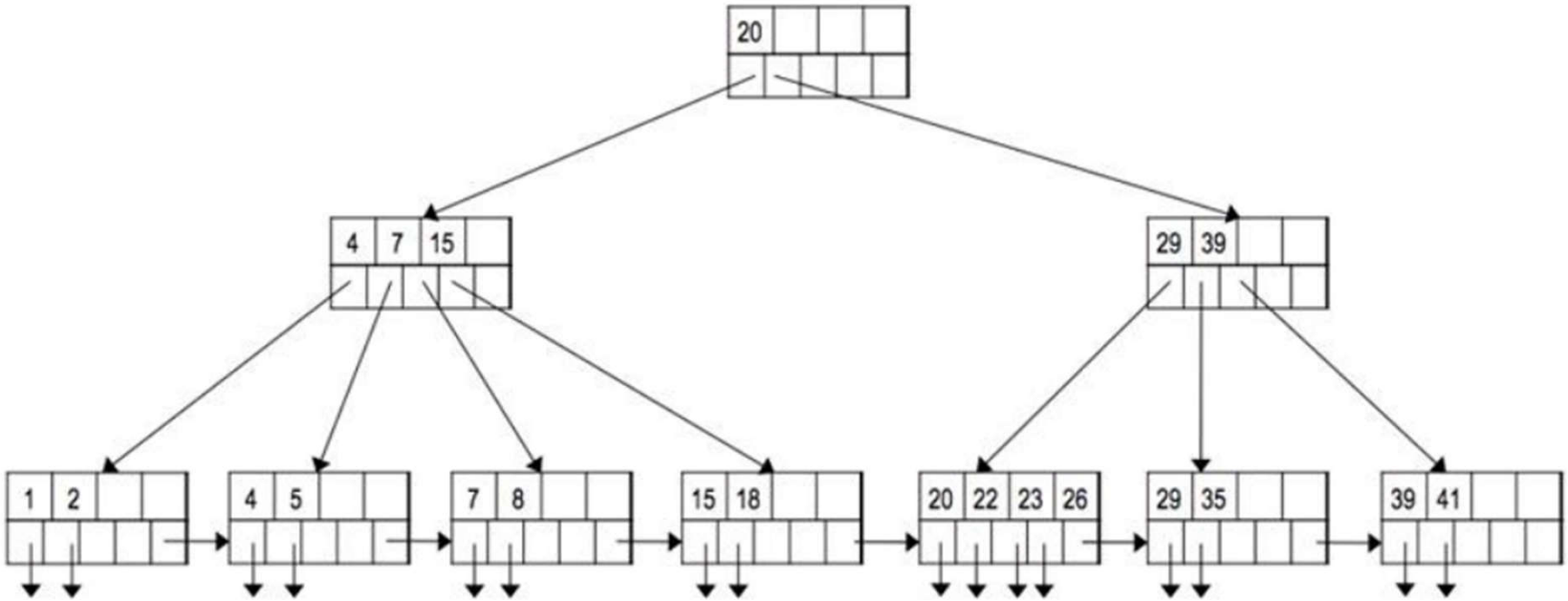


Solution:

- Step 1: Move 41 to merge with leaf [29,35]. The rightmost node at level 2 now only points to 2 leaves, which is problematic;
- Step 2: Move leaf [15,18] to be a child of [29]. The leftmost node at level 2 now has 2 keys [4,7] and 3 pointers. The rightmost node at level 2 now has 2 keys [20,29] and 3 pointers;
- Step 3: Finally, fix the root node to be not 20 but 15.

Solution

3. Show the resulting tree after inserting key 24 into the original tree.



Solution 1: Split the 5th leaf into [20, 22, 23] and [24, 26]. Add a key and a pointer to the rightmost node at level 2, which then has 3 keys [24, 29, 39] and 4 pointers.

Solution 2: Split the 5th leaf into [20, 22] and [23, 24, 26]. Add a key and a pointer to the rightmost node at level 2, which then has 3 keys [23, 29, 39] and 4 pointers.

Or, other solutions?

Any questions?