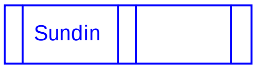
Indexing and Hashing – Practice Questions Solution

1. B$^+$-trees are often used as index structures for database files because they maintain their efficiency despite repeated insertion and deletion of data.

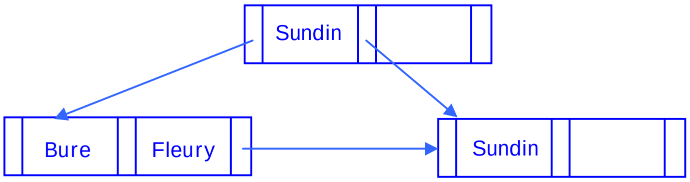
a) Show the structure of a B$^+$-tree for a file containing records with the following search key values, assuming that the tree is initially empty, that three pointers fit in one node, and that records are added in the order given:

Sundin, Fleury, Bure, Lindros, Federov

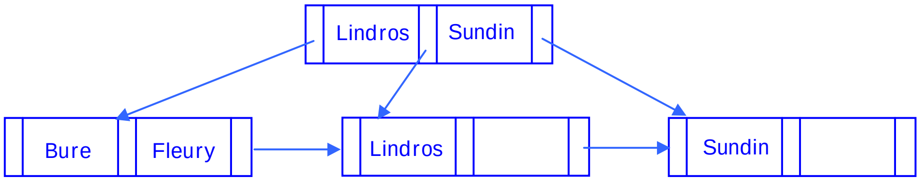
Add Sundin:



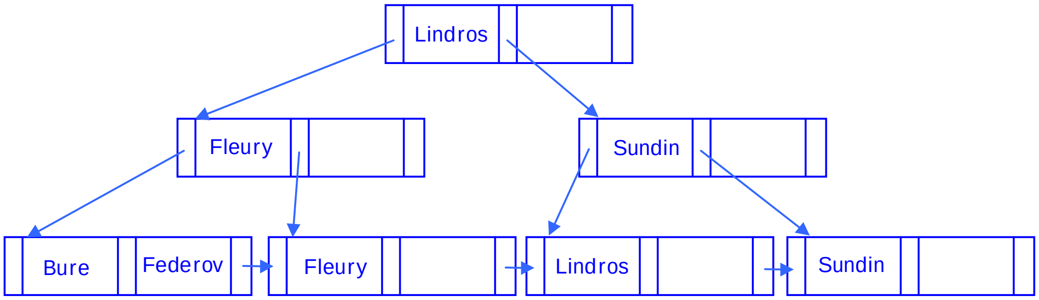
Add Fleury, Bure:



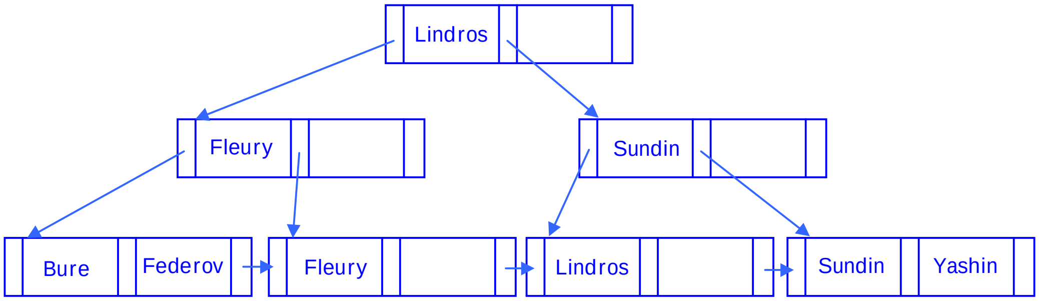
Add Lindros:



The final structure is shown below.



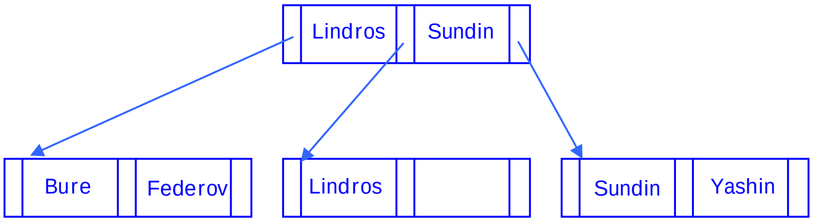
b) Now show the structure of the B$^+$-tree from part a) after the insertion of a record with the search key value 'Yashin'.



c) Now show the structure of the B$^+$-tree from part b) after the deletion of the record with the search key value 'Fleury'.

Reference: pages 350-351 of text.

- delete 'Fleury' from the leaf node. Now the leaf node is empty and should be eliminated. Point the previous leaf node to the next one. (*i.e.* Bure/Federov now points to Lindros).
- delete pointer to the now non-existent leaf from its parent. Note that the parent node now becomes too small (*i.e.* 0 entries).
- examine the parent's sibling (*i.e.* Sundin). It has only one entry, so examine its parent (*i.e.* Lindros, the root node). It too has one entry, these two can be coalesced and the depth of the tree decreases by one.



2. Suppose that extendable hashing is being used on a database file that contains records with the following search key values:

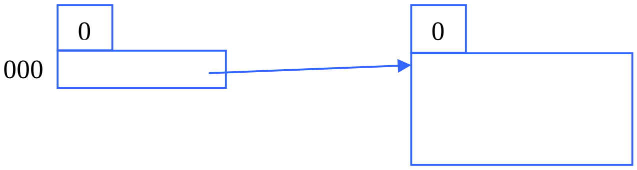
2, 3, 5, 7, 11, 17, 19, 23, 29, 31

a) Construct the extendable hash structure for this file if the hash function is $h(x) = x \bmod 7$ and each bucket can hold three records.

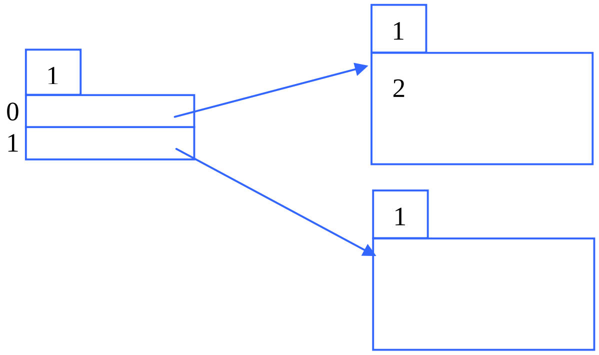
Decimal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110

Use the extendable hashing algorithm described in the text. The key is to realize that the hash key uses three bits and that the bits start from the left side.

The initial hash structure looks like this:

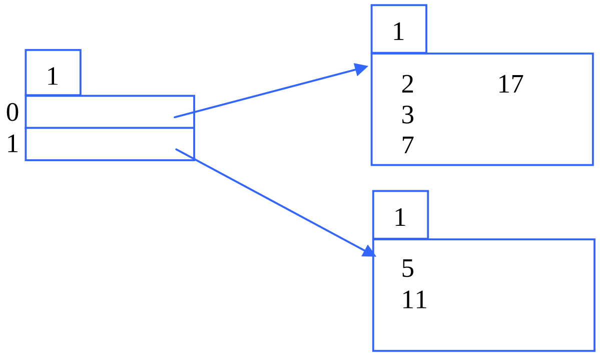


When the record 2 is added, the hash structure is extended to look like this:



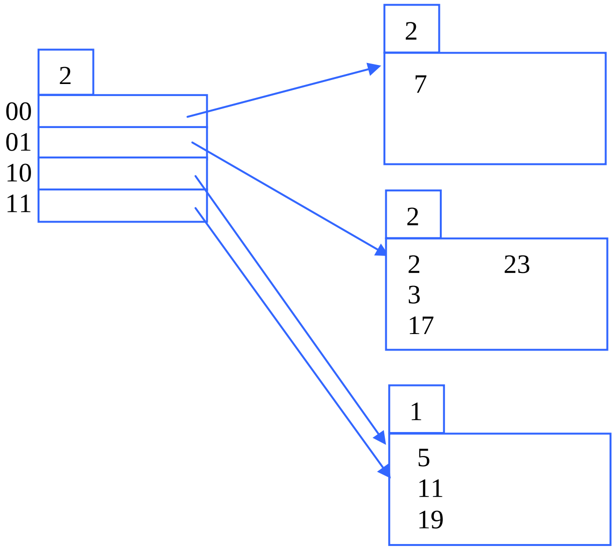
Note that 2 mod 7 = 2 = 010 – the first bit is 0, so it goes in the bucket pointed to by the 0 slot of the bucket address table. Likewise, the following records can be added to the hash structure as it stands:

Add 3 → 3 mod 7 = 3 = 011, goes to bucket 0
Add 5 → 5 mod 7 = 5 = 101, goes to bucket 1
Add 7 → 7 mod 7 = 0 = 000, goes to bucket 0
Add 11 → 11 mod 7 = 4 = 100, goes to bucket 1
Add 17 → 17 mod 7 = 3 = 011, goes to bucket 0



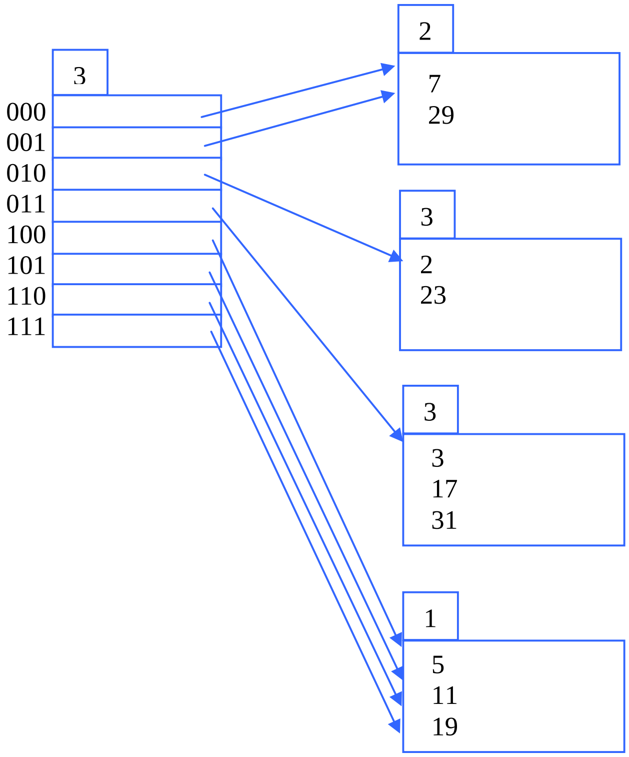
At this point, bucket 0 has overflowed, so we must split the bucket and rehash its contents using 2 bits. After doing that, we add the following records:

Add 19 → 19 mod 7 = 5 = 101, goes to bucket 10
Add 23 → 23 mod 7 = 2 = 010, goes to bucket 01



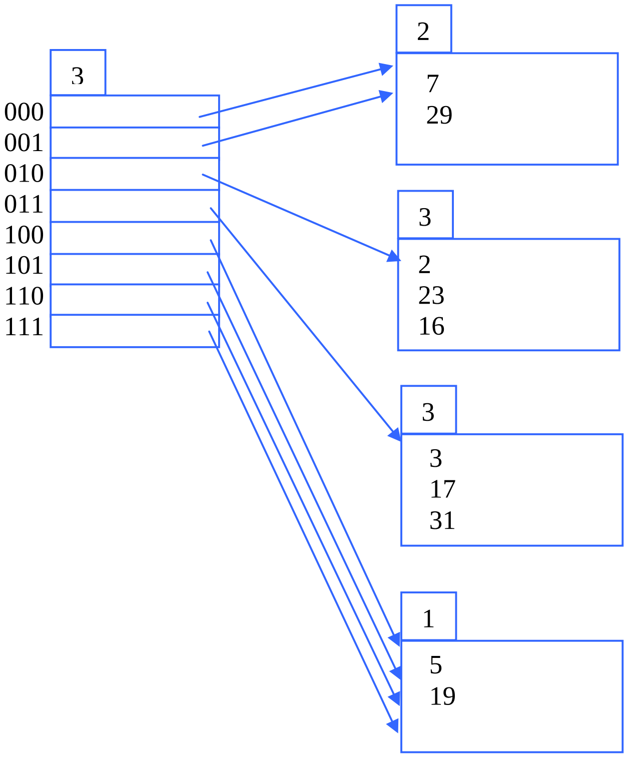
At this point, bucket 01 has overflowed, so we must split the bucket and rehash its contents using 3 bits. This allows us to add the remaining records, giving us the following hash structure:

Add 29 → 29 mod 7 = 1 = 001, goes to bucket 001
Add 31 → 31 mod 7 = 3 = 011, goes to bucket 011



b) Show how the structure from part a) changes after inserting a record with the search key value of 16 and then deleting the record with the search key value of 11.

In both the addition and the deletion, we do not have to split or coalesce any buckets, so the hash structure appears as follows:



c) Why is a hash structure not the best choice for a search key on which range queries (*i.e.* select * from relation where key > a and key <=b) are likely?

A range query cannot be answered efficiently using a hash index, as all the hash buckets would have to be read because key values in the range do not occupy consecutive locations in the buckets. Remember that keys should be distributed uniformly and randomly throughout all the buckets.