We will consider hash table based data structures that are kept in secondary storage. These differ from the main-memory versions in some small, yet important ways.

To begin, assume that the hash table is a bucket array numbered 0..B-1 containing B buckets. As we are considering secondary storage, each bucket is a block. [For now, we will skip discussion of header bytes and information kept for bucket blocks.]

A hash function h maps the universe of keys to buckets 0..B-1. For any key K, h(K) is a bucket index in {0..B-1}

Example: We give an example of a hash table with 4 buckets.

Keys are letters.

h(a) = 3
h(b) = 2
h(c) = 1
h(d) = 0
h(e) = 1
h(f) =

Bucket #	
0	d
1	e, c
2	b
3	a, f

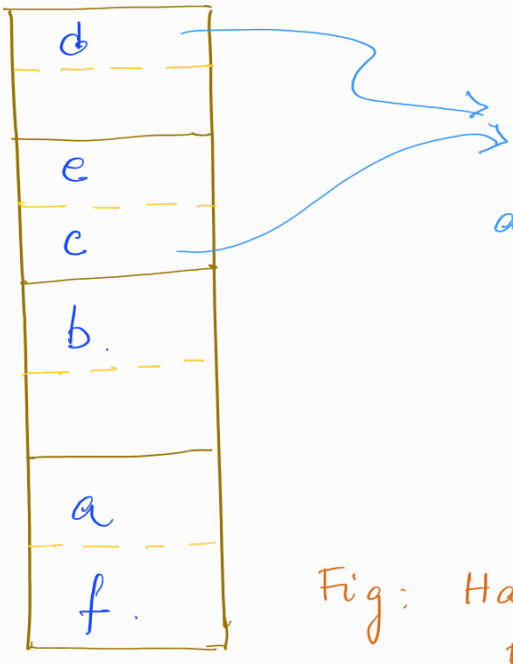


Fig: Hash Table File.

A bucket is a block. We assume that

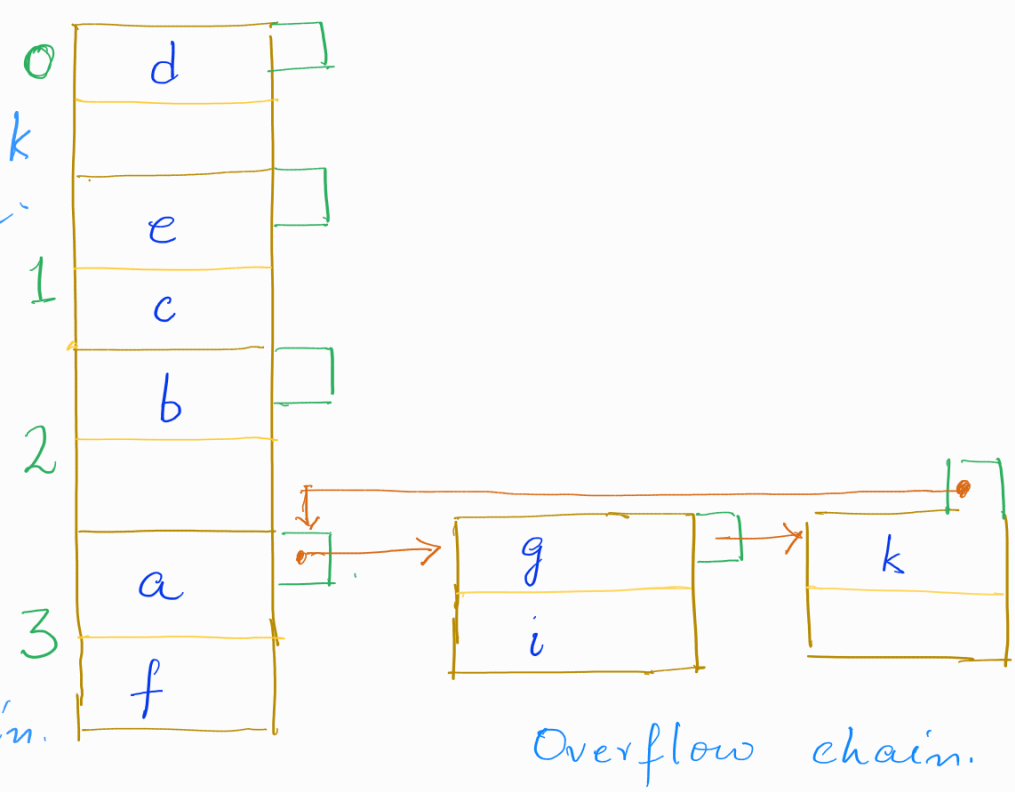
each bucket holds 2 records.

Searching for a key.

1. Given a key K, we apply the hash fn h to K to get the bucket index h(K). This block is retrieved to obtain the record(s) with key K, if it exists.

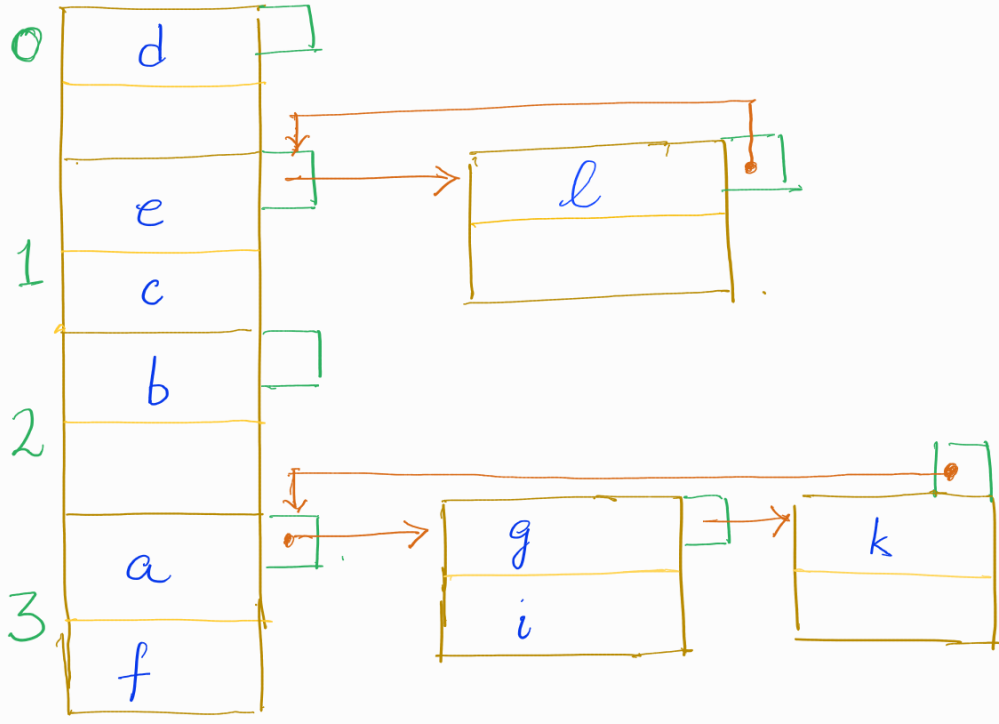
Note: It is possible that there are multiple records with same or different keys that map to the same bucket. This bucket, if unable to store all these records, normally have all these records stored in a linked list of blocks starting from this bucket. This linked list is called an "overflow chain."

Records with keys g, i and k also have hash value 3. They are successively inserted into the hash table by adding to the overflow chain.



at bucket 3. Suppose we now insert a record with key n such that $h(n) = 3$ (again). The overflow chain bucket at the end has space and this record is inserted.

Suppose we insert a record with key l and say $h(l) = 1$. The main bucket is full and so an overflow chain bucket is created and a linked list is attached to this bucket. See below.



Searching for keys would require making a pass over the overflow chain linked list if there is one. For efficiency's sake, it is important to restructure the hash table to limit the sizes of the overflow chains

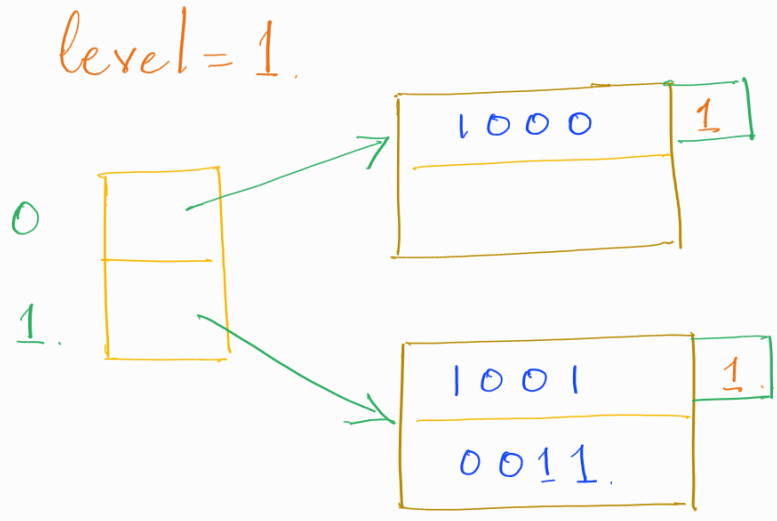
Note: The above is a description where the records are kept in a hash table. The master data file for a table is organized as a hash table. An alternative is to keep the master file organized as a sequence of blocks, in some order. (or no order). A hash table index can now be created just as a hash table file, except, that

- each record in the hash table index file stores the key value and a pointer to the record in the data file (just like an index entry (key, pointer) pair.

So far, we have only looked at static hash tables since the number, B of buckets in the hash table does not change. We now look at dynamic tables.

EXTENSIBLE HASHING

We start with an example of extensible hash table.



Buckets. Data Blocks.

1. It is assumed that the hash fn h applied to any key value returns a k-bit binary value. Here $k = 4$, for simplicity.

2. At any time, the hash file uses l lower order bits of any key value.

The value l, $1 \leq l \leq k$, is called the level of the hash table. In the example, the level is $l = 1$. For level l, the hash table has 2^l buckets numbered $0 \ldots 2^l - 1$, and corresponds to lower order l bits of the hash value of the keys.

3. The buckets only store pointers to blocks, the blocks actually store records

In the example, level $l = 1$, there are 2 buckets numbered 0 and 1. Each bucket b points to a block which stores records of all key values K, such that the low order l bits of $h(K)$ gives index b. If the number of records with keys whose low order l bits equals b exceeds the capacity of a block, then an overflow chain is created and maintained starting at the first block pointed to by the bucket.

3. At the moment, the "nub" in the block (which is the block header) contains the level of this block, among other things. The level of a block is always $\leq l$, where l is the level of the file. The significance of this will become clear shortly.

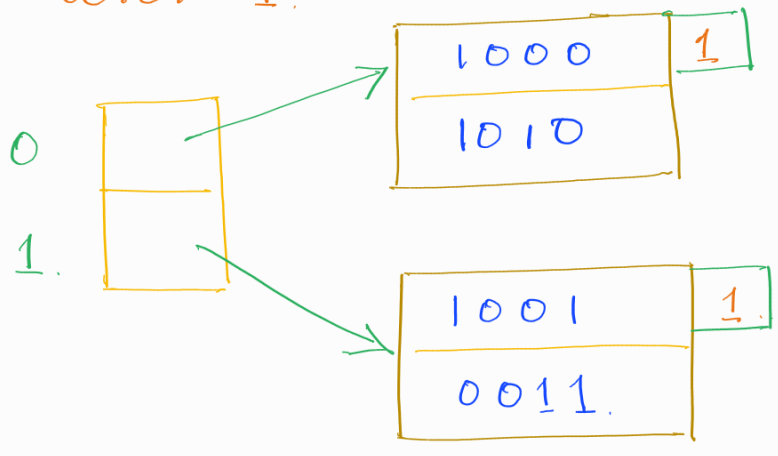
Insertion into Extensible Files.

1. Insertion begins like insertion into a hash table. Given a record with key value K, compute $h(K)$ and

consider the low order l bits of $h(K)$. For e.g. let $h(K) = 1010$. In the given file, level $= 1$, so there are $2^1 = 2$ buckets. The lowest bit is 0, hence this key with $h(K) = 1010$ is to be inserted into bucket 0. This bucket has space for a record, and so it gets inserted there.

insert key with $h(K) = 1010$

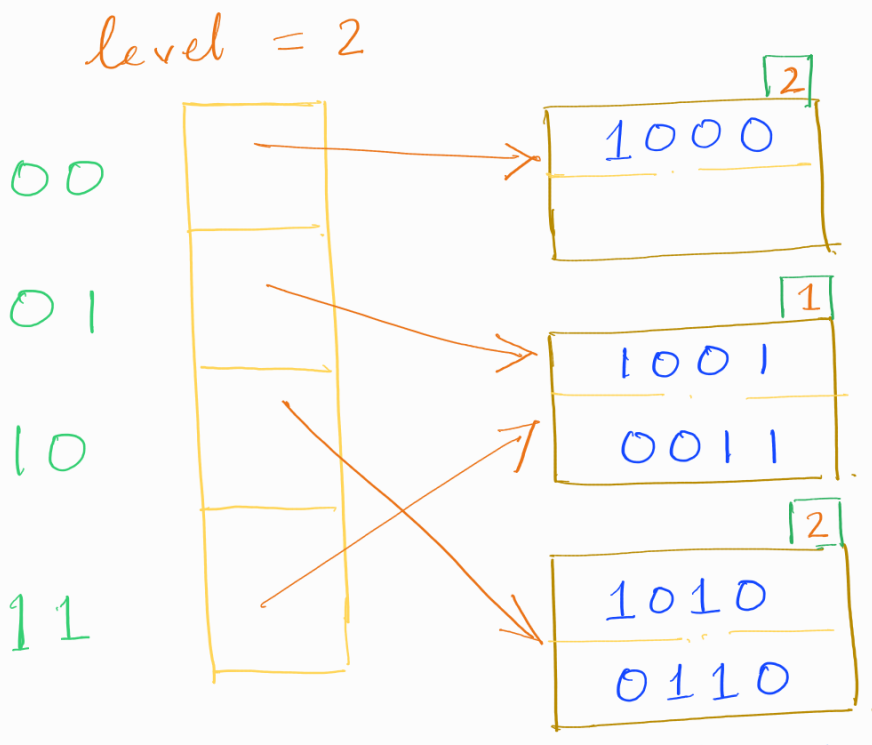
level = 1.



Buckets. Data Blocks.

Now suppose we wish to insert a key value with $h(K) = 0110$. Once again, with level $l = 1$, the lower order bit is 0, and the key maps to bucket 0. However this bucket is full. The extensible hash table now increases the level of the file to 2

and splits the data block 0 into 2 buckets corresponding to lower order hash values 00 and 10, both having low order bit 1. The index buckets containing pointers to data blocks now are numbered 00, 01, 10, 11. - total of 4 buckets.



Index Buckets

Data buckets/ blocks.

The original data bucket for low order 1 bit =0 after the insertion would have 3 entries, 1000, 1010, 0110. Since it doesn't fit in the block, this is split to consider last 2 bits: 00 or 10. The level increases by 1:

the index buckets are now indexed as 00, 01, 10, 11. No of buckets $= 2^l$. $= 2^2 = 4$. The low order 0 bucket is split into 00 and 10 and the keys are distributed appropriately. So the key 1000 stays in the original 0 bucket, which now is the 00 bucket; the keys 1010 and 0110 are shifted to low order 10 bucket. The index 00 bucket points to data bucket 00 — analogously for 10 bucket. The old index 1 bucket is not split. It continues as before and its level remains 1. Both index pointers 01 and 11 point to the same data bucket - the 1 bucket.

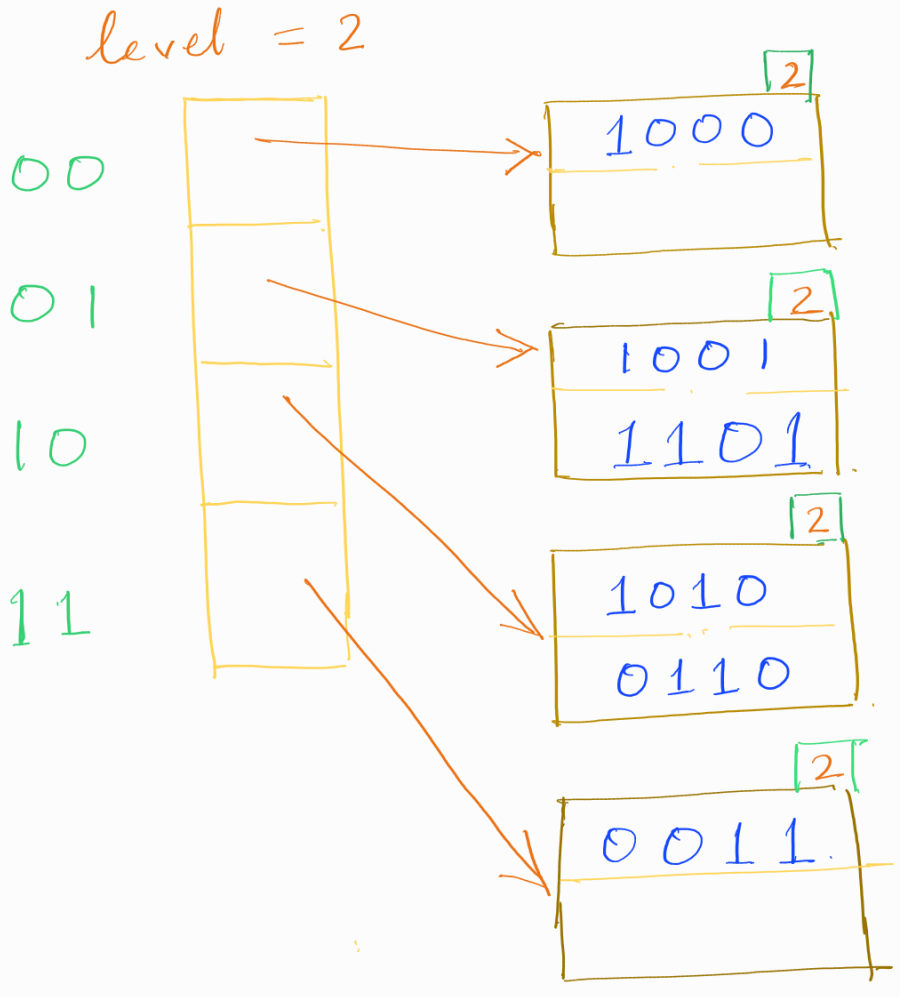
Continuing with this example suppose we insert records whose keys have hash values

1101

1110

in sequence. The hash value 1101 causes.

causes data bucket 1 (level 1) to split and its records reavranged.

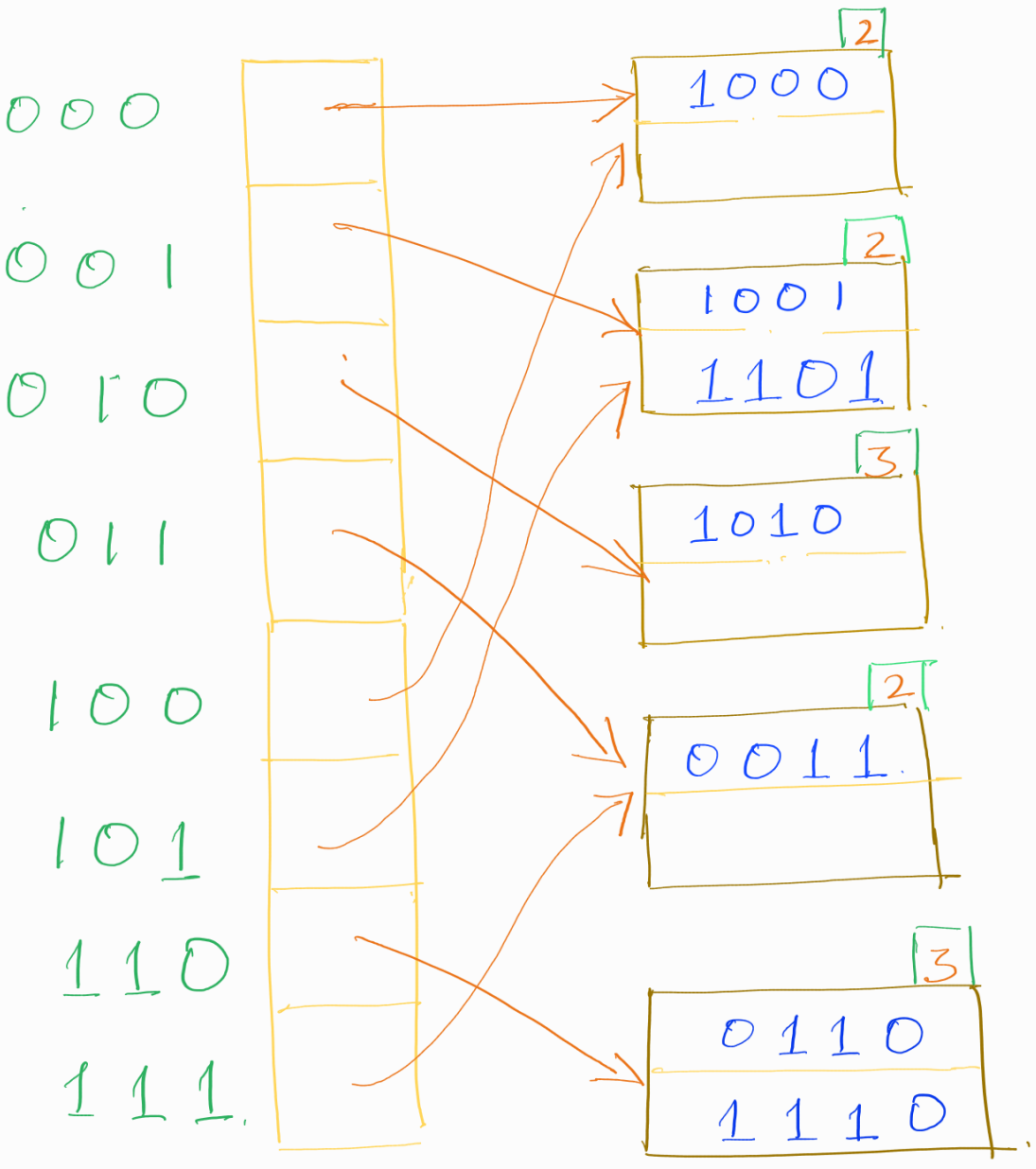


Index Buckets

Data buckets/ blocks.

We now insert 1110. The 3rd. data bucket corresponding to. 10 splits. Level of file increases by 1. The levels of data buckets 010 and 110 are each 3, rest are all 2.

level = 3



Index buckets.

Data Buckets.

Remark on Deletion: Often a data occupancy rate is used, for e.g. if the desired occupancy rate $0.3 \leq P \leq 0.9$

it is intended to mean that in a data bucket, the used space by records is between 30% to 90% of the bucket capacity. The upper bd., say 90% allows some free space for possible expansion of records in that bucket. The lower bd. attempts to enforce a certain min. capacity.

Deletion operation is simply the reverse of the possible split operation: if a record is deleted from a bucket and this bucket falls below the minimum threshold, then the algorithm considers the possibility of reducing the level l by 1. This would require merging of buckets whose local levels are at the original level l. The deletion operation can be somewhat expensive as multiple buckets may need to be merged.