DBMS Tutorial 23.01.2019

Topics

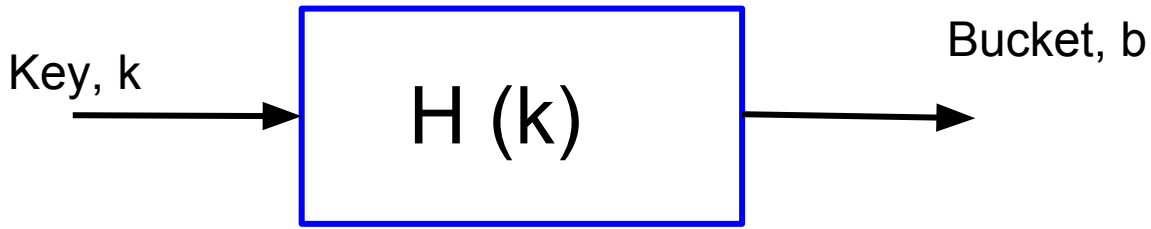
Hashing

Extensible and Linear

Hashing

Main Idea :

A Hash Function h() which takes in key, k and gives bucket number. We store the record in this bucket.



To insert/retrieve record with key K, compute h(K), look up bucket number h(K), insert/retrieve.
On insert if bucket is full, create a overflow block.

Hashing

Not good for range queries

Good for query such : select * from table where k=”xzy”

Simple Hashing : Fixed number of buckets right from the start. As the file grows overflow blocks might be required, but the number of buckets remain fixed, which means a bucket is made of one primary block and 0 or more overflow blocks.

Dynamic Hashing : Linear and Extensible, the number of buckets increases as the file grows.

Extensible Hashing

At any given point size of the directory depends on ***i*** bits from ***h(k)***.

(i≤ total bits of h(k)), i are the leftmost bits.

In the Ullman book:
directory = buckets; and the buckets = blocks holding records.
Different terminology from what we are using here, can be confusing.

Directory :

- A level of indirection. Holds the pointers to buckets containing the records.
- Multiple entries in the directory can point to the same bucket.
- Its size is always a power of 2 (2^i), therefore a growing step means **doubling the directory**.

Bucket Array :

- Holds the records.
- k bits (k≤i) decide membership in bucket, ie which bucket a record will go to.
- If bucket is full, it is split (now k= k+1 bits will decide membership). If new k>i, double directory first and then split the bucket. Recursive splitting possible!
- Overflows not possible, as bucket splits if its full when a new record arrives.

Example1 Extensible

Extensible Hashing.

Starting with two directory entries, 0 and 1. Insert into this records whose keys hash to (in this order):

a) 0001,1001 and 1100

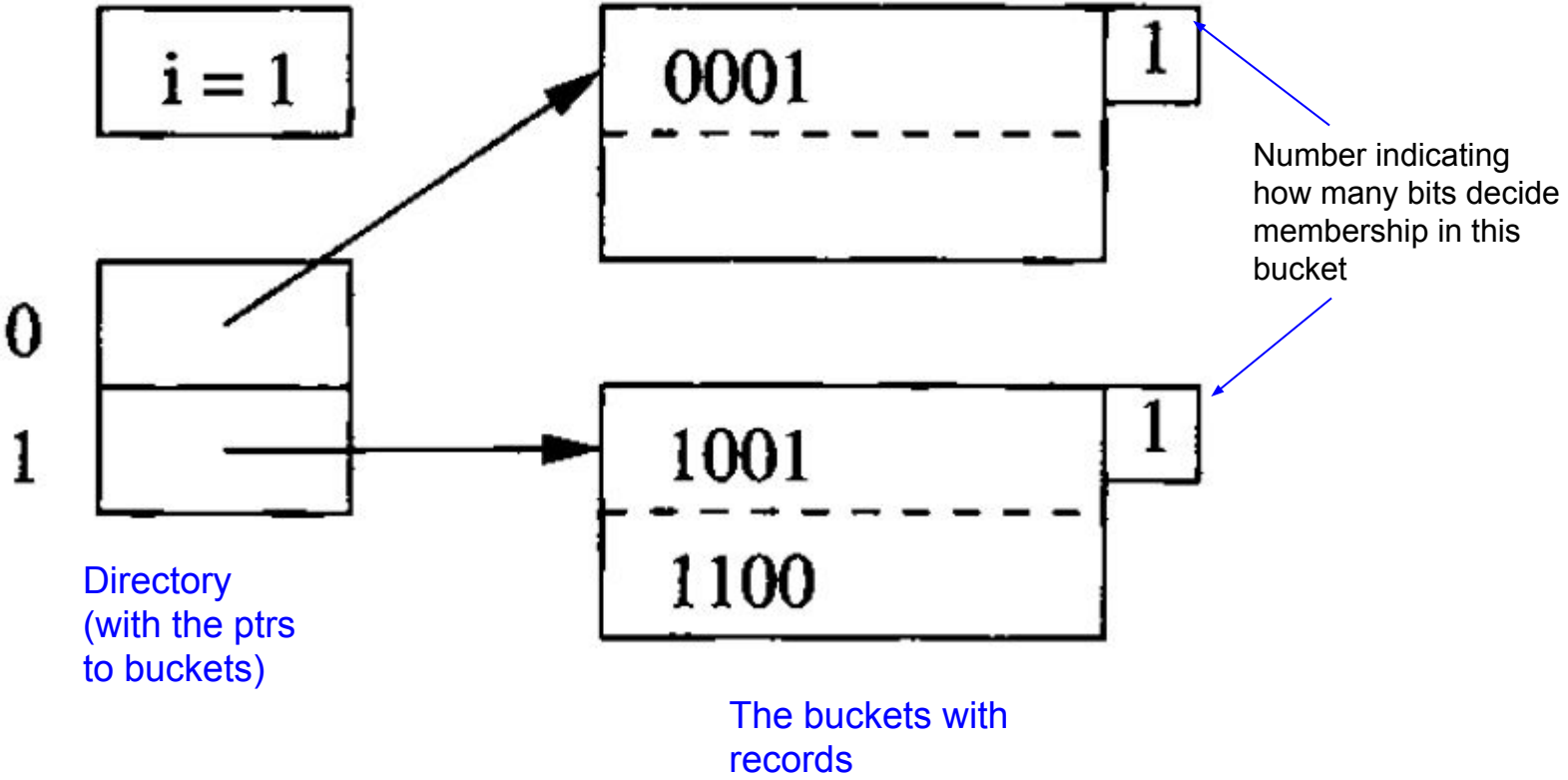
b) 1010

c) 0000, 0111 and 1000

Each bucket can hold 2 records.

Example1 Sol.a

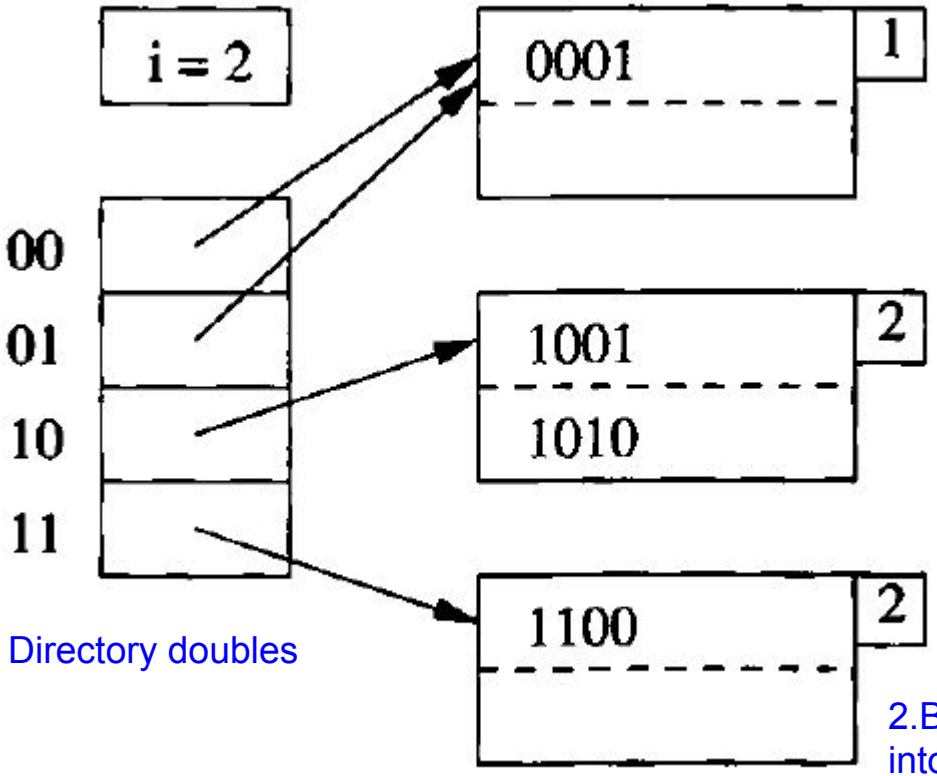
a) 0001,1001 and 1100



Example1 Sol.b

b) 1010

Before we can insert 1010, we need to split the Bucket 1 as it is full into 10 and 11. Now we will need two bits to decide (k>i) therefore we double directory first.



1. Directory doubles

2.Bucket 1 splits into 10 and 11

Since this bucket still needs only one bit to decide, all directory entries starting with 0 point here.

These buckets have different number of bits deciding membership!

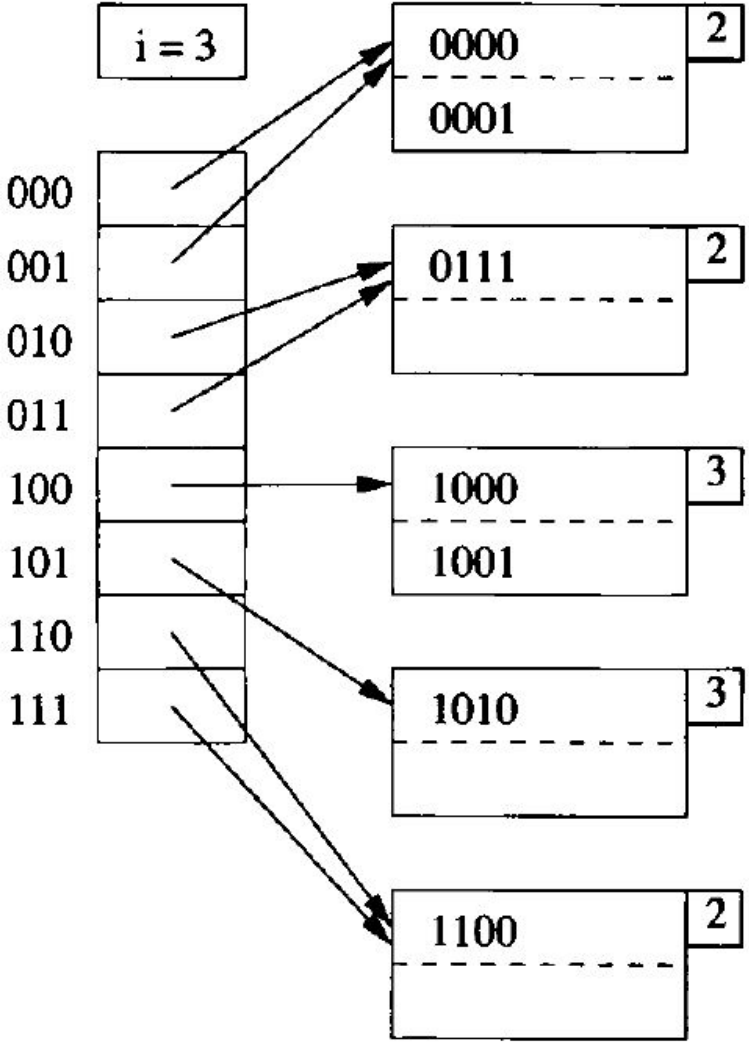
Example 1 Sol.c

c) 0000, 0111 and 1000

000: is not a problem as there is space in bucket starting with 0.

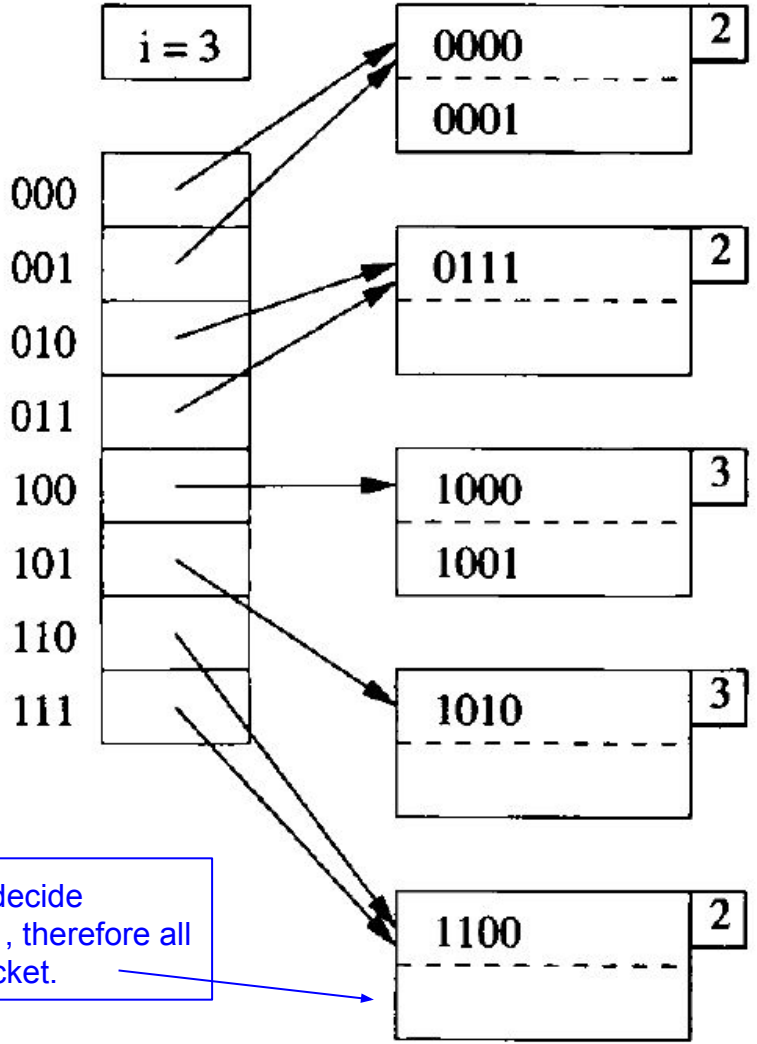
0111: causes the bucket pointed to by 0 to split into 00 and 01.

1000: needs the bucket pointed to by 10 to split into 100 and 101, but first we need to double the directory to get the 3bit pointers.



Example 1 Sol.c Notes

c) 0000, 0111 and 1000



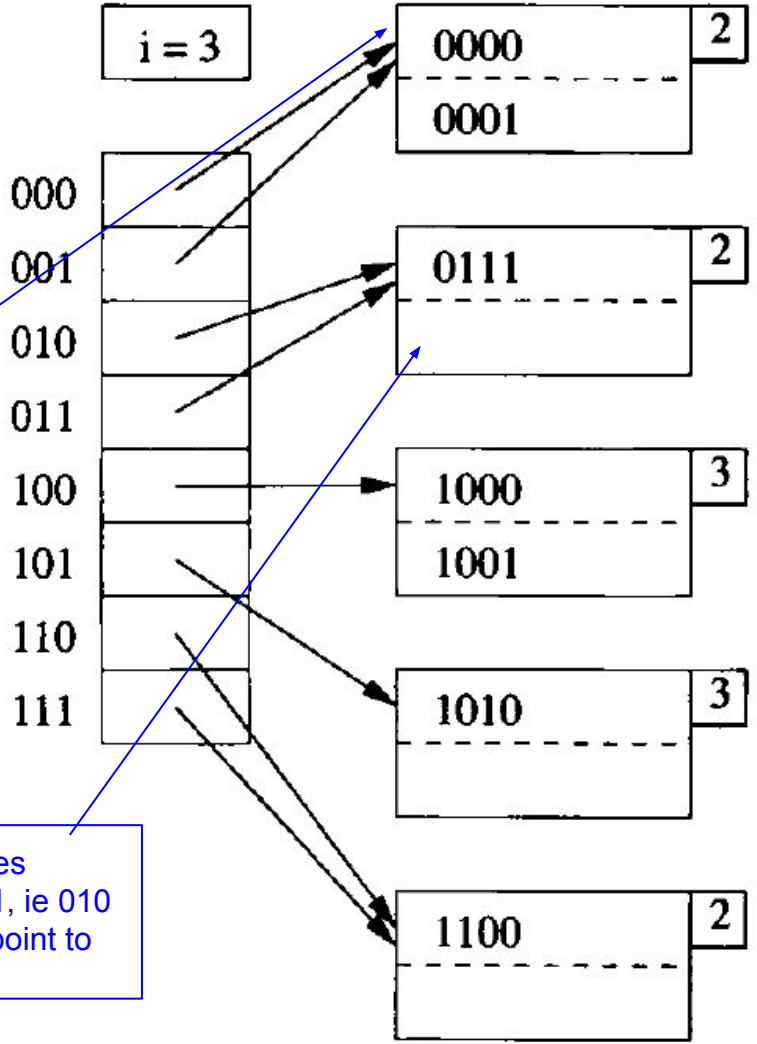
Even though some buckets are using three bits to decide membership, this bucket still needs only two bits, 11, therefore all directory entries starting with 11 will point to this bucket.

Example 1 Sol.c Notes

c) 0000, 0111 and 1000

This bucket also needs only two bits, 00, to decide, therefore both the directory entries starting with 00 i.e. 000 and 001, point to this bucket. If a new record comes with 00x this bucket will split and use three bits to decide.

Similarly, entries starting with 01, ie 010 and 011 both point to this bucket.



Linear Hashing

Number of buckets depends on Utilization factor. Buckets are numbered using ***i*** bits (rightmost) from ***h(k)***. (i≤ total bit of h(k))

Bucket Array :

- Holds the records and are numbered according to the bits in use.
- No indirection, Size grows linearly.
- Might require re-arrangement on adding a bucket.

On Insert :

- If this new insertion does not violate utilization factor, insert in the right bucket, (if bucket is full add an overflow block).
- If new insertion violates utilization factor, create a new bucket, (rearrange existing records if required), then insert in the correct bucket.
- If m bits decide which bucket, and there are n buckets, m<n is not a problem, but if m>n, let change the MSB of m to 0, and insert in that bucket.

Linear Hashing : Insertion

Suppose ***i*** bits of the hash function are being used to number buckets and a record with key K is intended for bucket $\mathbf{a_1 a_2 .. a_i}$, let $\mathbf{a_1 a_2 .. a_i}$ = m, and n = current number of buckets

- If m < n, then the bucket numbered m exists, and we place the record in that bucket.
- If n < m < 2^i, then the bucket m does not yet exist, so we place the record in bucket $\mathbf{0 a_2 .. a_i}$ (Please note : $\mathbf{a_1}$ must 1 if this condition is true!)

Linear Hashing: Utilization Factor

Decides when a new bucket will be added.

The bucket being added bears no relationship to the bucket into which the insertion occurs!

Total keys i.e. in primary + overflow

$$\text{Utilization Factor} = \frac{\text{Total number of keys inserted so far}}{\text{Capacity of the buckets (primary blocks only)}}$$

It represents how full the buckets are on average. If a new insertion will violate the utilization factor, add new bucket first and then insert the record.

Linear Hashing: Rearrangment

If the binary representation of the number of the bucket we add is **$1a_2..a_i$**, then we split the bucket numbered **$0a_2..a_i$**, putting records into one or the other bucket, depending on their last *i* bits.

Note that all these records will have hash values that end in **$a_2..a_i$** and only the ***i***th bit from the right end will vary.

For eg. if we are adding bucket numbered 111, it is like splitting existing bucket numbered 011 into 011 and 111, therefore any existing records in bucket 011 which were ending with the bits 111 can now move to newly created bucket 111.

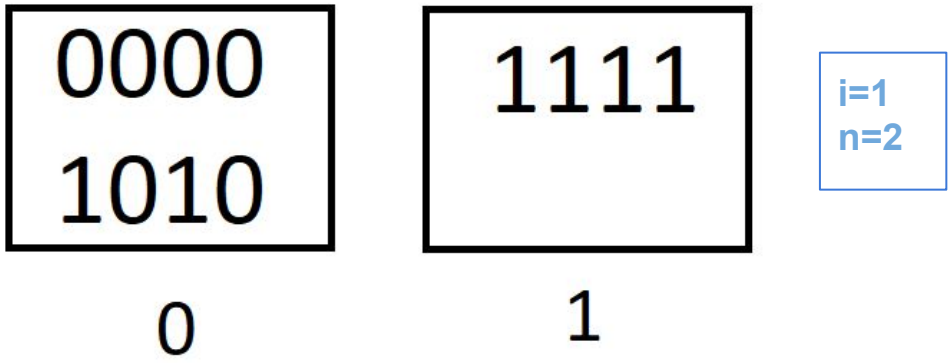
Example2 Linear

Insert the following records using Linear Hashing technique. Start with n (ie number of buckets) = 2. Utilization Factor 85%. Capacity of block 2 records.

a) 1111, 0000, 1010
b) 0101
c) 0001
d) 1101
e) How many buckets will there be if the number of records = 100.

Example2 Sol.a

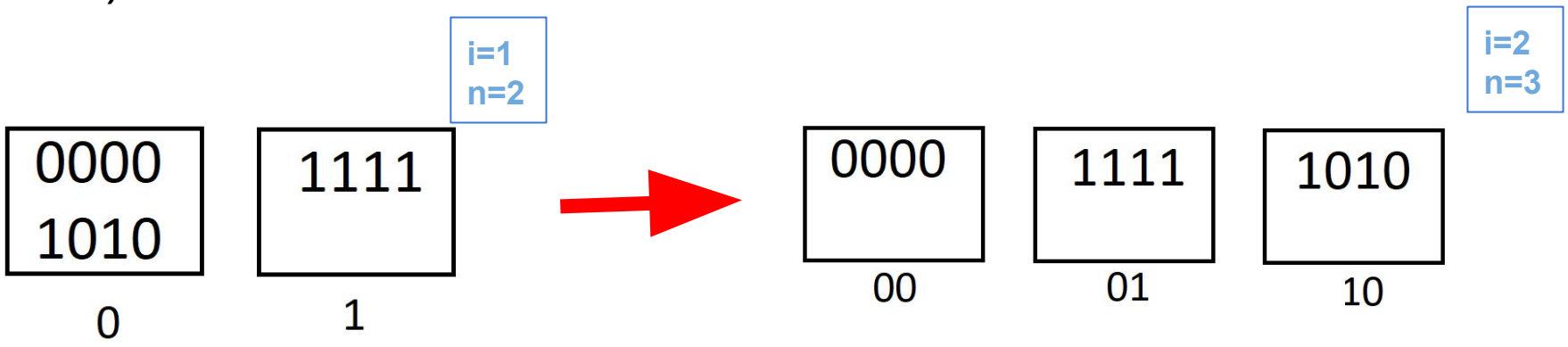
a) 1111, 0000, 1010



Utilization = ¾ = 75% < 85%

Example2 Sol.b1

b) 0101

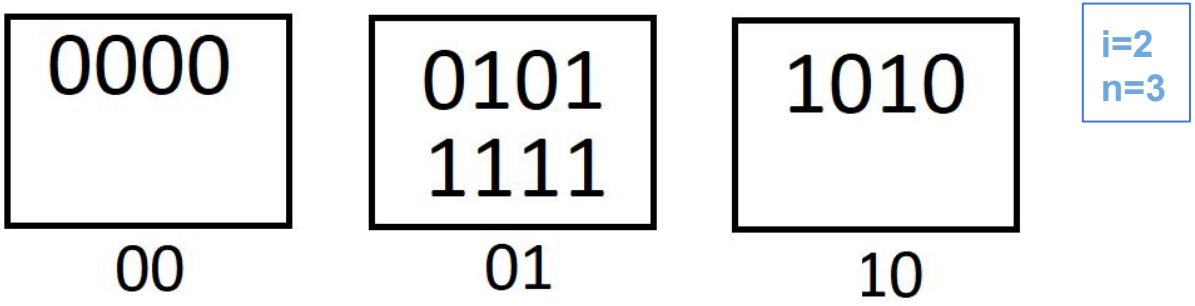


Adding 0101 will make the Utilization = 4/4 = 100% > 85%

Therefore we need to add a new bucket first, this will make n=3, ie $>2^i$, therefore we increase i as well.

Example2 Sol.b2

b) 0101

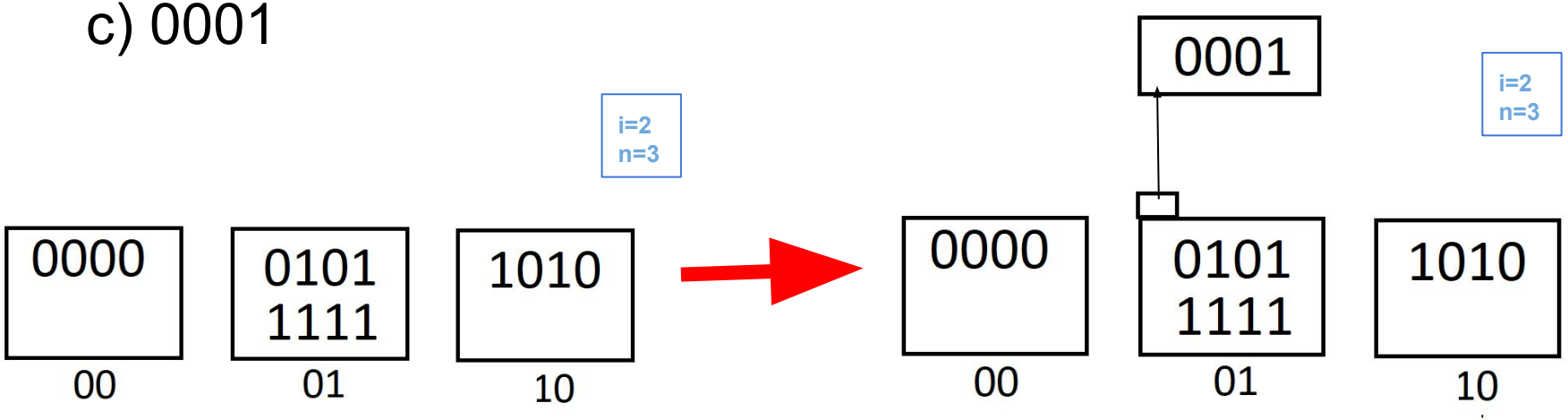


We can add 0101 now.

Utilization = 4/6 = 66.7% < 85%

Example2 Sol.c1

c) 0001

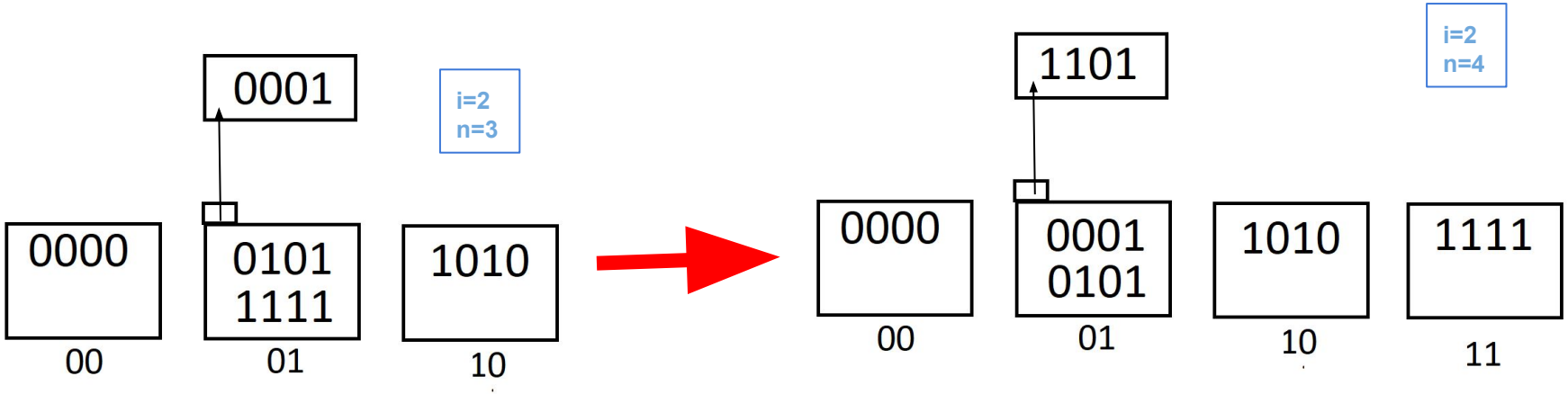


We can add 0001 without violating UF, but need an overflow block.

Utilization after adding 0001 = 5/6 = 83.3% < 85%

Example2 Sol.d

d) 1101



If we add 1101 Utilization = 6/6 = 100%, therefore need to split first, next bucket will be 11 (which is like splitting the bucket 01, therefore any record in bucket 01 ending in the bits 11 will now move to newly created bucket 11).

After adding 1101 Utilization = 7/8 = 75%

Example2 Sol.e

e) How many buckets will there be if the number of records = 100.

Even though the capacity of a block is 2 records, buckets are allowed, on average to be 85% full, therefore,

No. of buckets = ceil(100/(2*0.85)) = 59

Extensible and Linear

Which bucket :

Extensible : Most Significant Bits, ie, Leftmost bits decide

Linear : Least Significant Bits, ie, Rightmost bits decide

Overflow Blocks

Extensible : No overflow blocks as buckets split as and when required.

Linear : Can have overflow blocks, as new buckets added only when adding a new record will exceed utilization limit.

Look Up:

Extensible : Never need to search more than one data block but directory adds indirection, therefore 2 IOs if directory in on disk.

Linear: One or more depending on how many overflow blocks there are.

Exercise1 : Recursive Splitting in Extensible

Starting with two directory entries, 0 and 1. Insert into this records whose keys hash to (in this order):

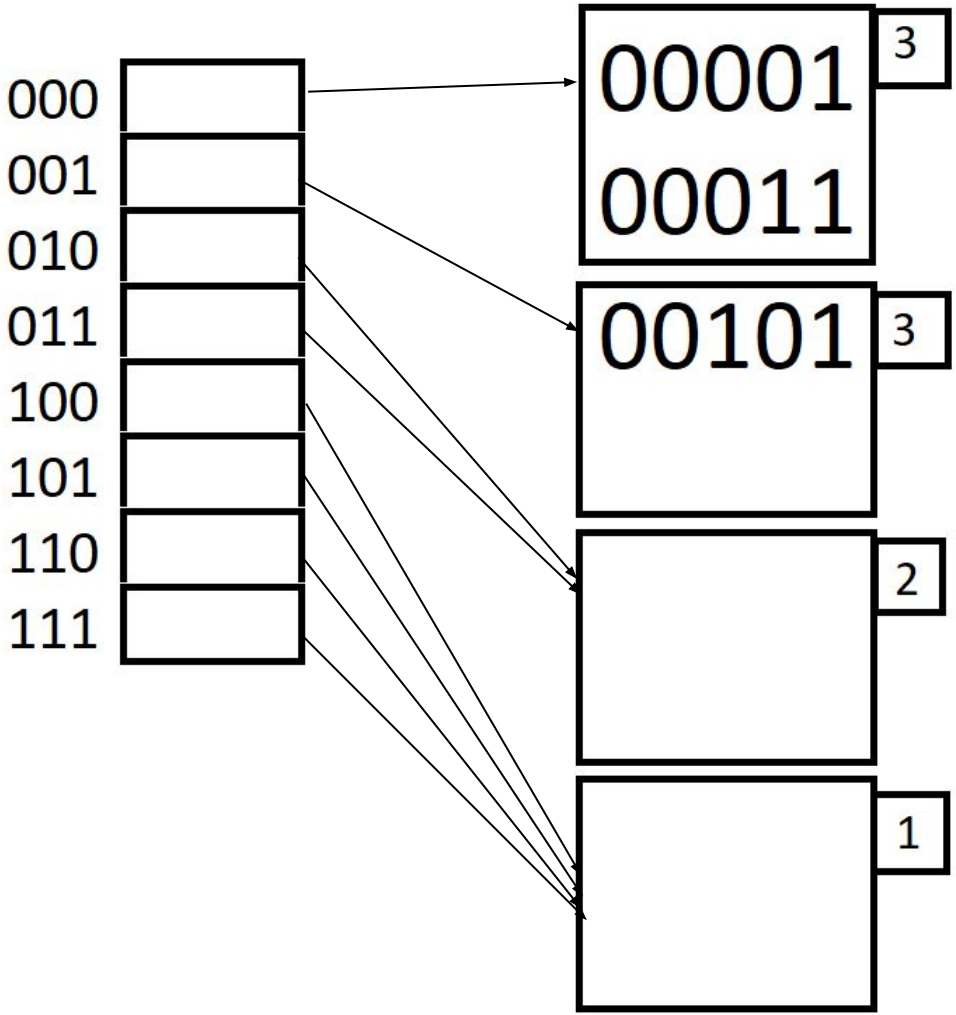
00001, 00101, 00011.

Each bucket can hold 2 records.

Exercise1 : Re

Starting with two directory entries, 0 and 1.
Insert into this records whose keys hash to (in this order):
00001, 00101, 00011.

Each bucket can hold 2 records.



13.4.6 Exercise

Exercise 13.4.6: Suppose keys are hashed to four-bit sequences, also suppose that blocks can hold three records. If we start with a hash table with two empty blocks (corresponding to 0 and I), show the organization after we insert records with keys:

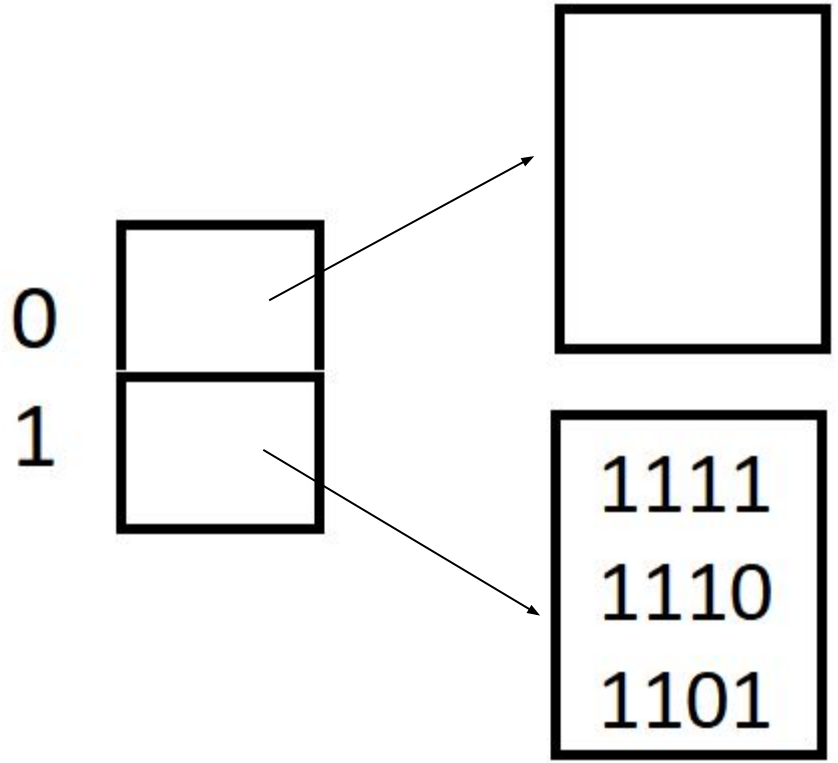
c) 1111,1110,. . . ,0000, and the method of hashing is extensible hashing.

d) 1111,1110,. . . : 0000, and the method of hashing is linear hashing with a capacity threshold of 75%.

Only the first few steps are shown in the next few slides.

13.4.6 Exercise Sol.c1

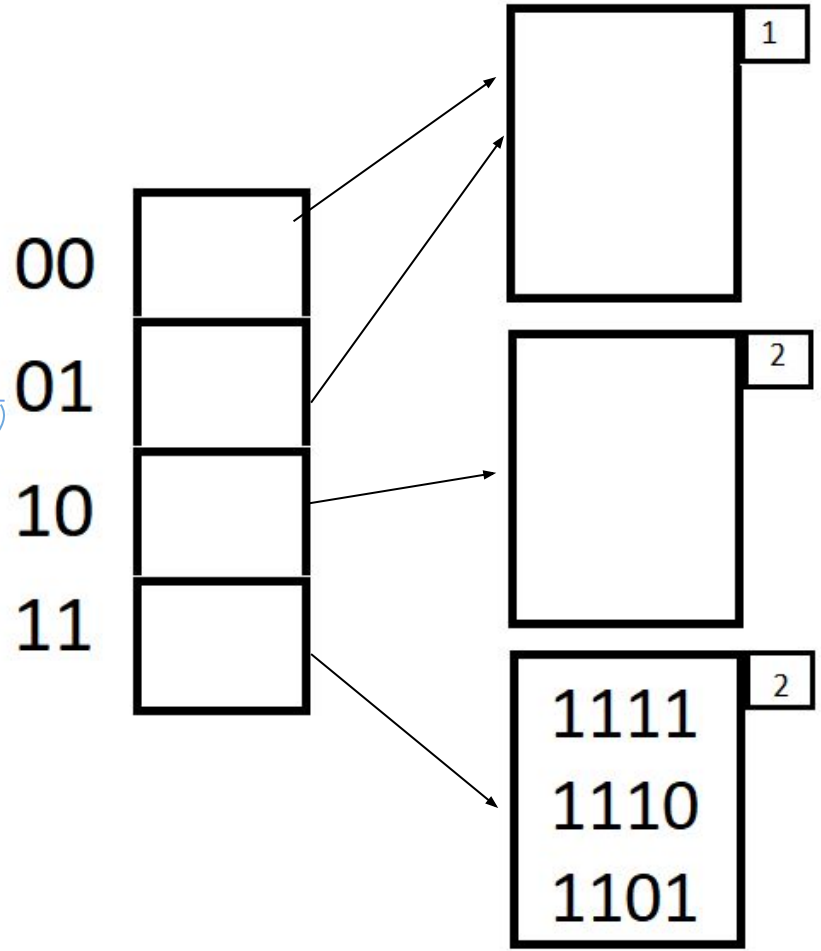
First three additions : 1111, 1110, 1101 are straightforward and they all go to the same bucket.



13.4.6 Exercise Sol.c2

Next element 1100 needs to go to the bucket pointed to by 1 but there is no space there so we need to split the bucket, but since the directory has only two pointers (i=1), we first need to double the directory.

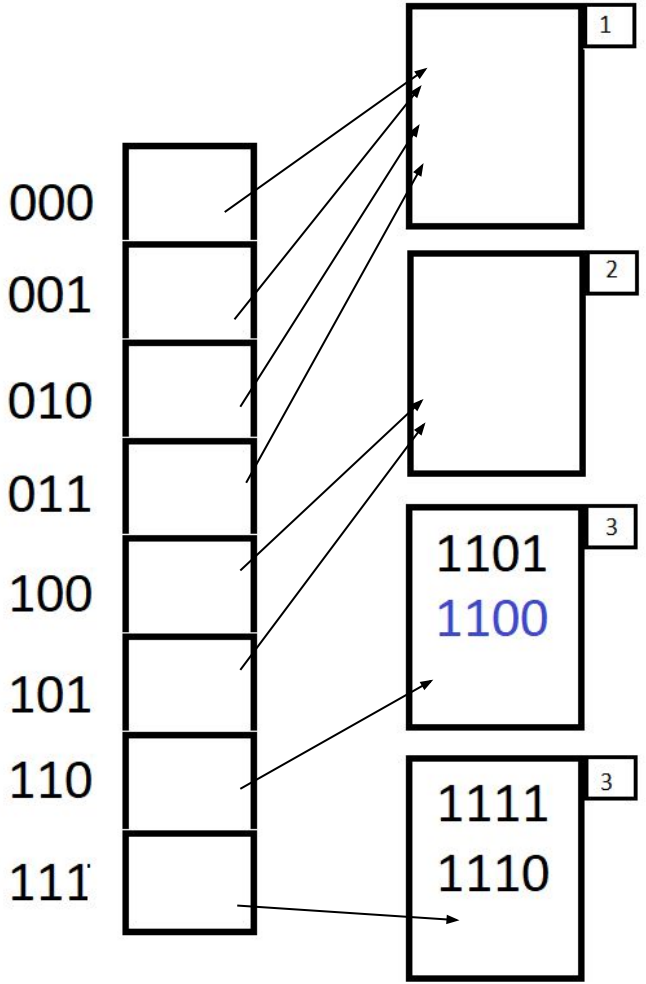
After doubling (i=2) we have enough directory entries and we split bucket 1 into 11 and 10, but all the records still go to bucket 11 and we still don't have enough space, so we need to split bucket 11 further into 111 and 110, for this we need 3bits pointers and hence we need to double the directory again.



13.4.6 Exercise Sol.c3

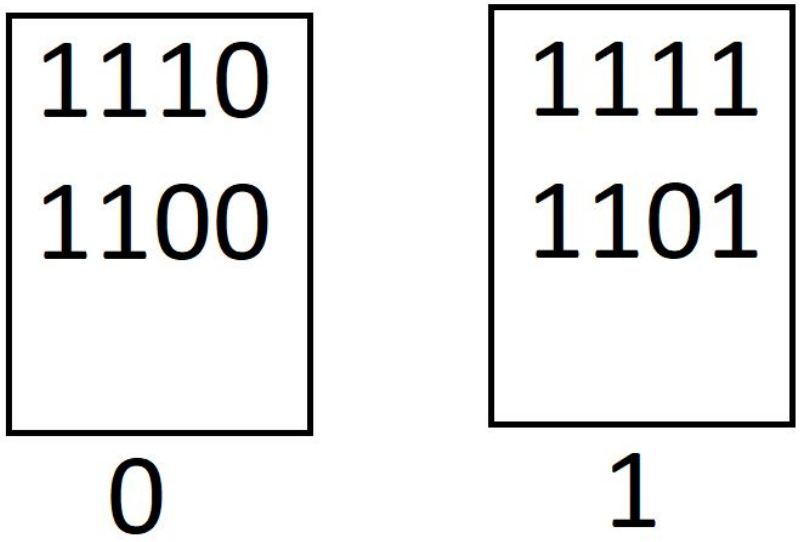
Now we can insert 1100.

The next 12 entries will not require doubling the directory, we will simply split the buckets as and when needed.



13.4.6 Exercise d1

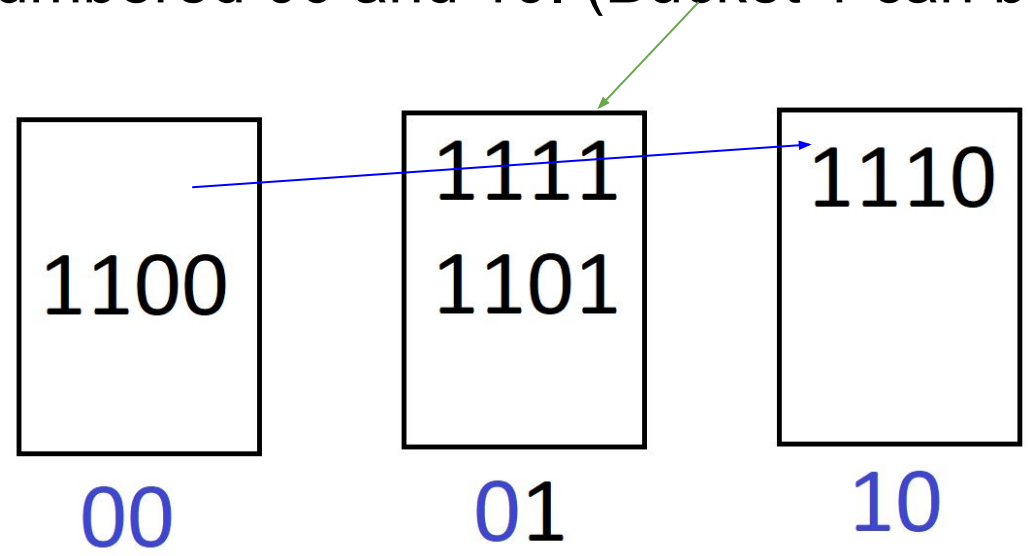
d) We can add the first 4 records without violating the utilization factor.



Current Utilization factor = 4/6, ie 66.67%

13.4.6 Exercise d2

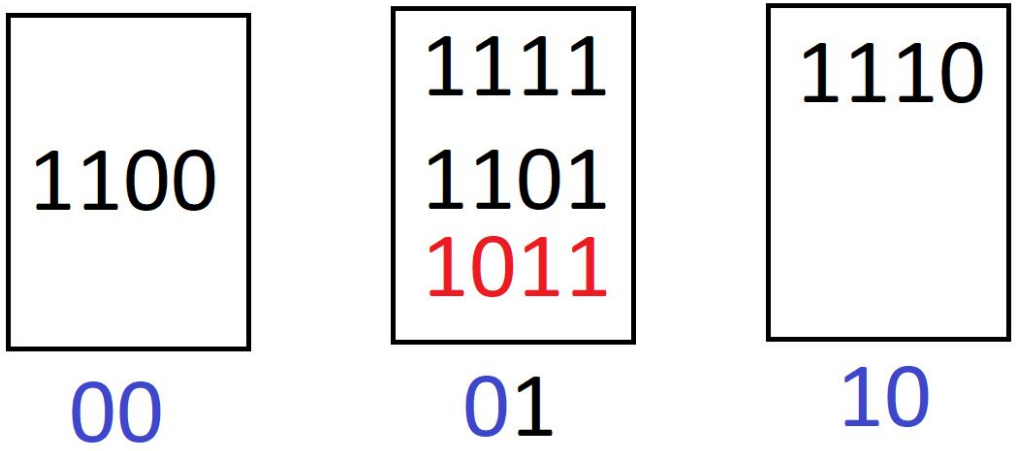
d) Addition of 1011 will make the utilization ⅝ >75% therefore we need to split first. Bucket 0 will split and we will get buckets numbered 00 and 10. (Bucket 1 can be written as 01 now.)



Rearrangement is usually required on addition of a bucket, here Entry 1110 moves from bucket 00 to bucket 10.

13.4.6 Exercise d3

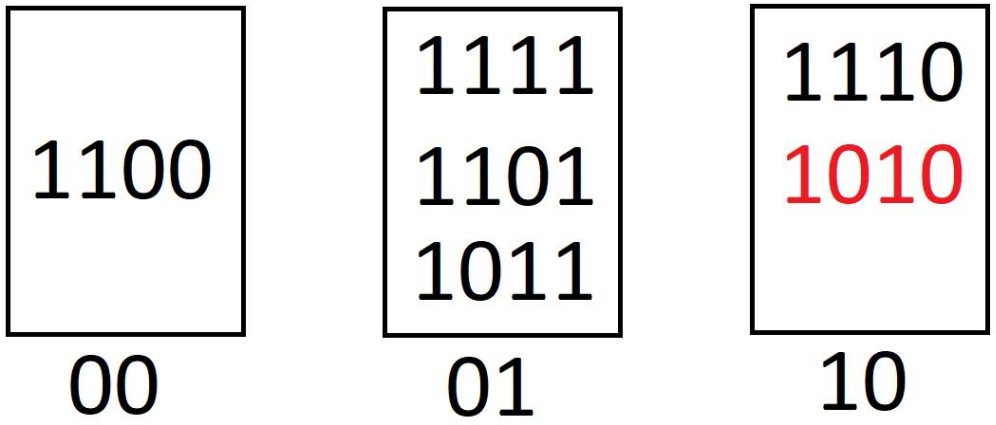
d) Now we can add 1011. Right now 2 bits are in use, here the deciding bits were 11, but since bucket 11 doesn't exist, we put the record into bucket 01 (This record will eventually move to bucket 11 when it is created in the future).



Buckets are only created if an insertion will violate the utilization factor, otherwise we use the strategy we just did!

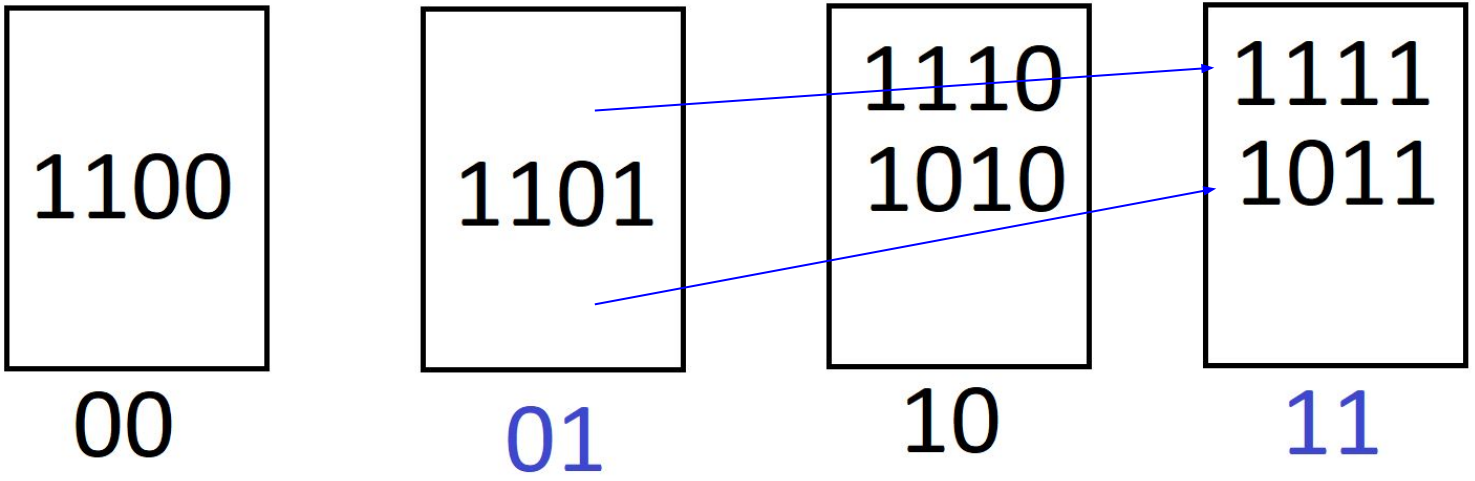
13.4.6 Exercise d4

d) Next is 1010. This will make the utilization 6/9 = 66.67%
(which is fine as its <= 75%)



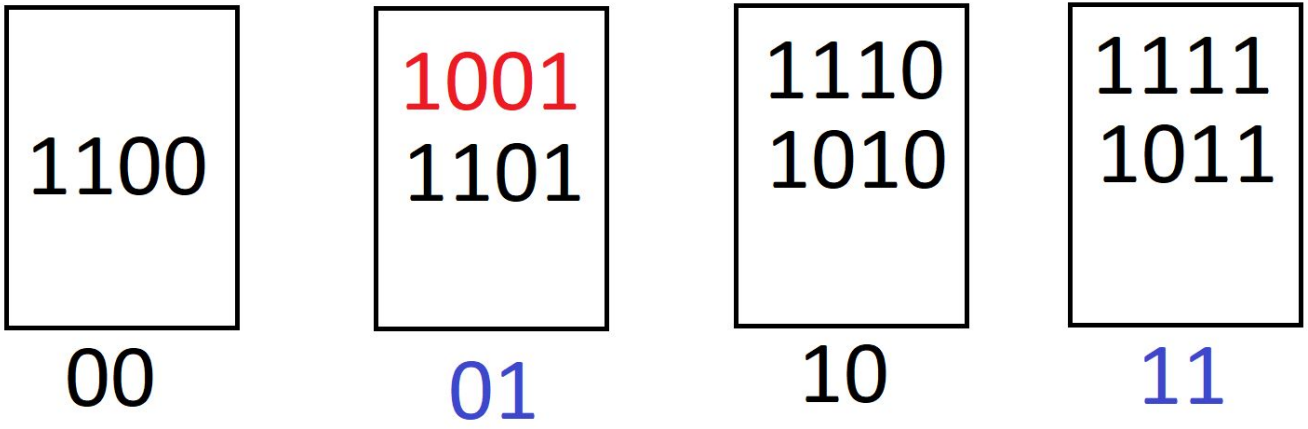
13.4.6 Exercise d5

d) Now we want to add 1001. This will make the utilization factor 7/9 = 77.78%, which will violate 75%, therefore we need to split first. This requires rearrangement since records ending with bits 11 from bucket 01 move to this new bucket.



13.4.6 Exercise d5

d) Now we can add 1001. This will make the utilization 7/12 = 58.33%



Continue adding the remaining entries!

Right now we can add two more entries (9/12 = 75%) before we need the next bucket.

13.4.7 Exercise

Exercise 13.4.7: Suppose we use a linear or extensible hashing scheme, but there are pointers to records from outside. These pointers prevent us from moving records between blocks, as is sometimes required by these hashing methods. Suggest several ways that we could modify the structure to allow pointers from outside.

13.4.7 Exercise (DS:CB) Ans

Exercise 13.4.7: Suppose we use a linear or extensible hashing scheme, but there are pointers to records from outside. These pointers prevent us from moving records between blocks, as is sometimes required by these hashing methods. Suggest several ways that we could modify the structure to allow pointers from outside.

1. Leaving forwarding addresses in buckets on split.
2. Use the full sequence of bits produced by the hash function as the pointer, and use hash table to look up a record whenever an external pointer is followed.

Sheet 9 Q3

3) In an extensible hash table with n records per block, what is the probability that an overflow will have to be handled recursively, i.e., all members of the block will go into the same one of the two blocks created in the split?

Sheet 9 Q3 Ans 1/4

3) In an extensible hash table with n records per block, what is the probability that an overflow will have to be handled recursively, i.e., all members of the block will go into the same one of the two blocks created in the split?

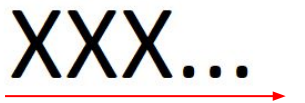
A bucket needs to be split if its full and a new record arrives which qualifies for membership in that bucket. That is, if the bucket is using i bits to decide membership, all the records in the bucket (say n) plus the new record have the same first i bits.

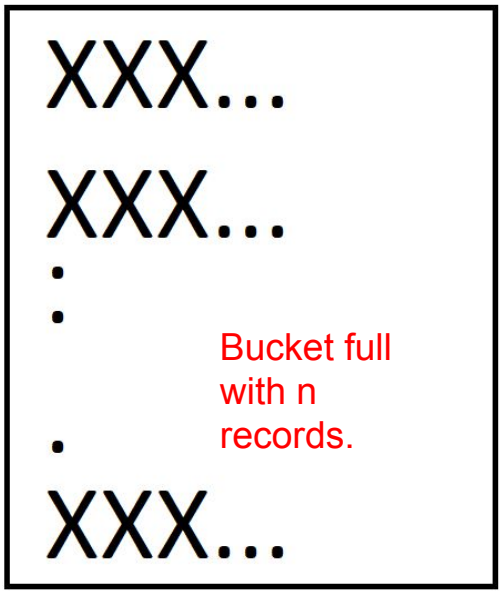
It will be a recursive split, however, if the first split doesn't solve the problem and all the records still qualify for membership in one bucket, ie all n+1 records have the same first i+1 bits. It is only possible if all n+1 records either have 1 as their i+1th bit or have 0 as their i+1th bit.

Sheet 9 Q3 Ans 2/4

Consider a bucket which is currently full with n records. The bucket needs XXX as the first bits for membership. Suppose a new n+1 record arrives.

Now this bucket needs to split into XXX1 and XXX0.

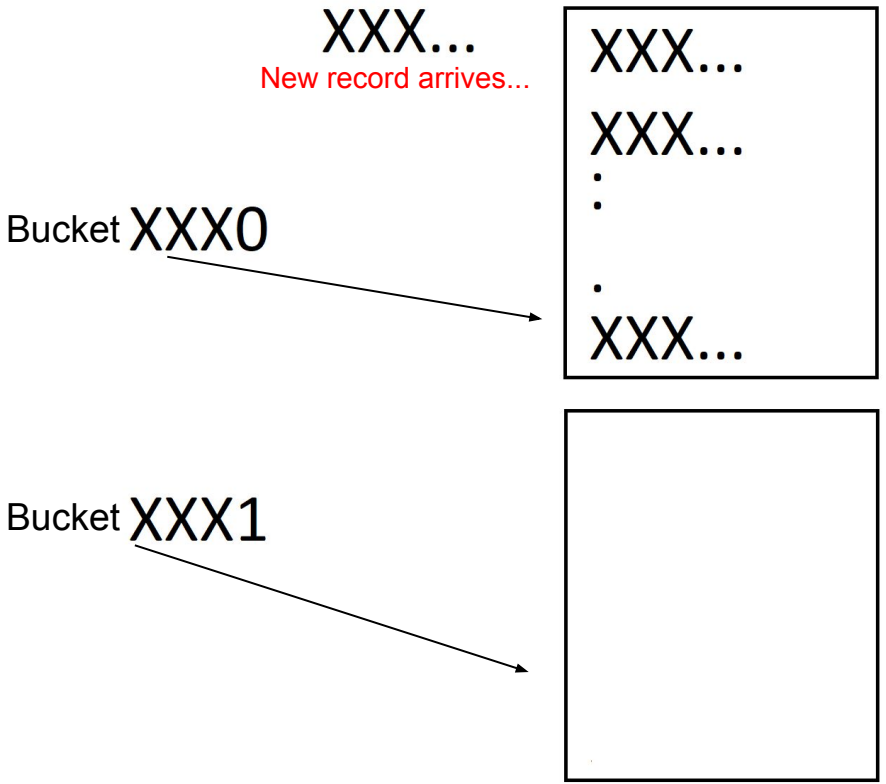




Sheet 9 Q3 Ans 3/4

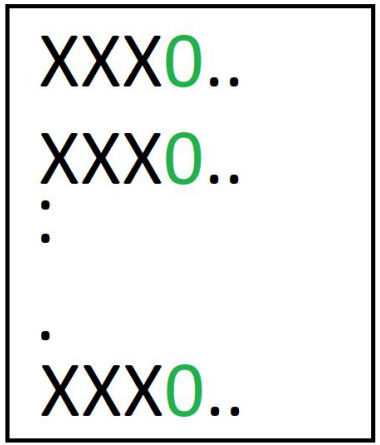
Normally after this split all records starting with XXX0 with stay in bucket XXX0 and all records starting with XXX1 will move to bucket XXX1.

But what if all n+1 records have the same next bit, either all start as XXX1 or all start as XXX0 , these two possibilities will result in a recursive split.



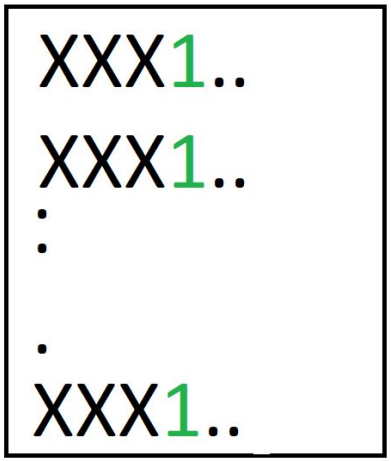
Sheet 9 Q3 Ans 4/4

XXX0..
New record arrives...



OR

XXX1..
New record arrives...



Probability all n+1 records have the next bit as 0
= (1/2)^(n+1)

Probability all n+1 records have the next bit as 1
= (1/2)^(n+1)

Therefore, total probability of a recursive split = 2 x (1/2)^(n+1)