Linear Hashing

By Satyam Kumar (160123035)

Lecture on 14 March, 2019 by Dr. Amit Awekar

Linear Hashing

- Linear hashing is a dynamic hash table algorithm that allows the expansion of hash tables one slot at a time.
- This is another dynamic hashing scheme, an alternative to Extendible Hashing.
- Linear Hashing handles the problem of long overflow chains without using a directory, and handles duplicates.
- It exhibits near-optimal performance, both in terms of access cost and storage load.

How it works:

- Split pointer s decides which bucket to split
  - s is independent to overflowing bucket
  - At level *i*, *s* is between 0 and 2^i
  - s is incremented and if at end, is reset to 0.

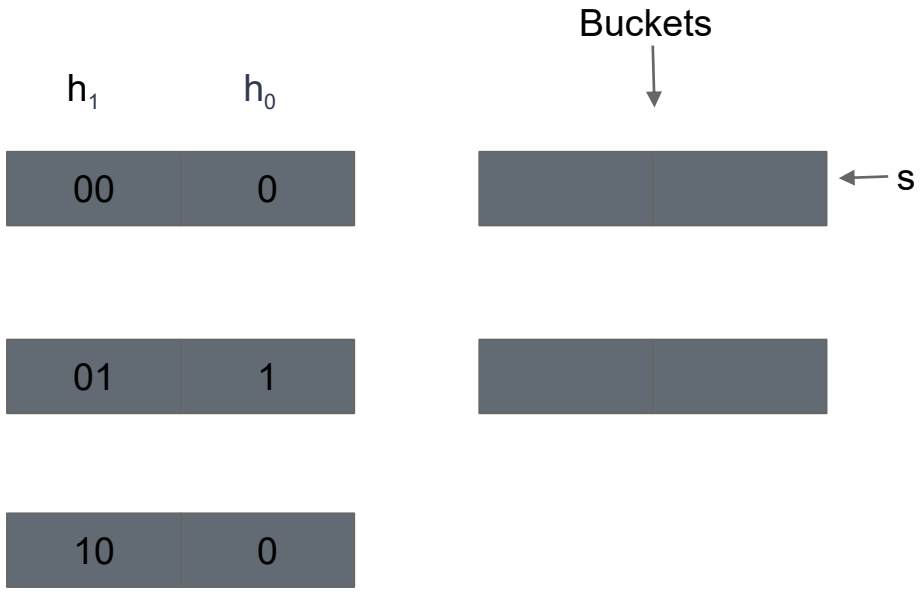
- $h_i(k) = h(k) \bmod (2^i n)$
- h_{i+1} doubles the range of h_i

- Algorithm for inserting 'k':

1. b = h0(k)
2. if b < split-pointer then
3. b = h1(k)

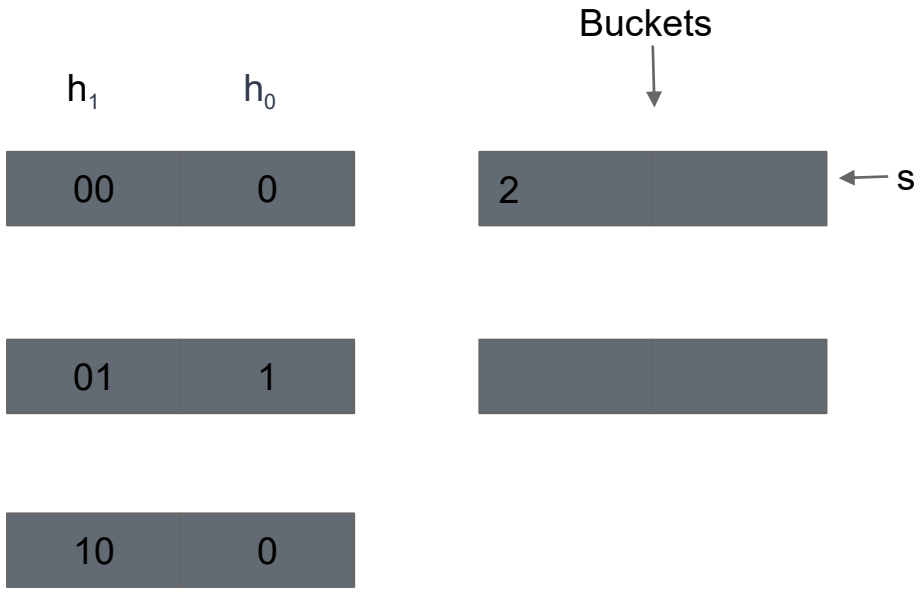
Order of Insertion : 2, 4, 17, 8, 6, 10

To be Inserted : 2



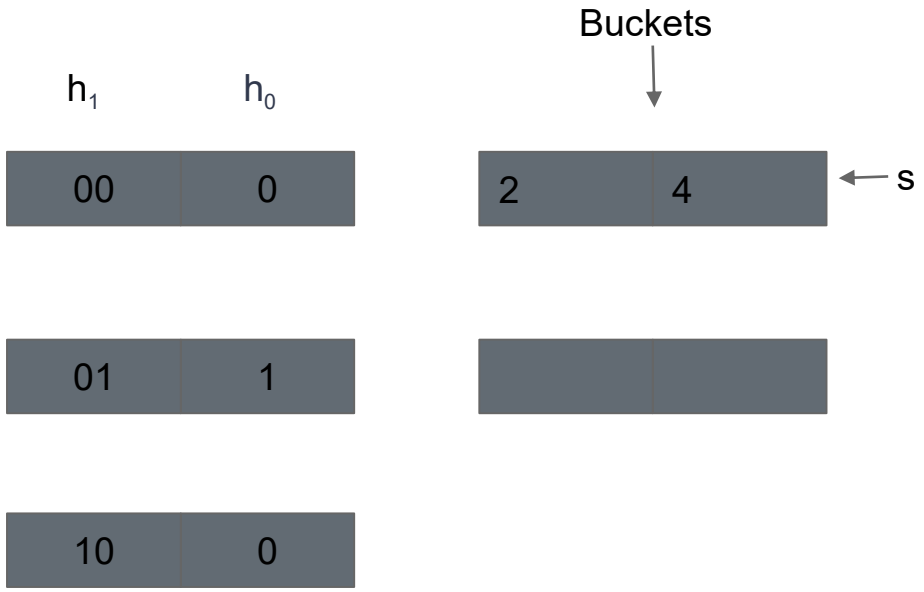
Order of Insertion : 2, 4, 17, 8, 6, 10

To be Inserted : 4



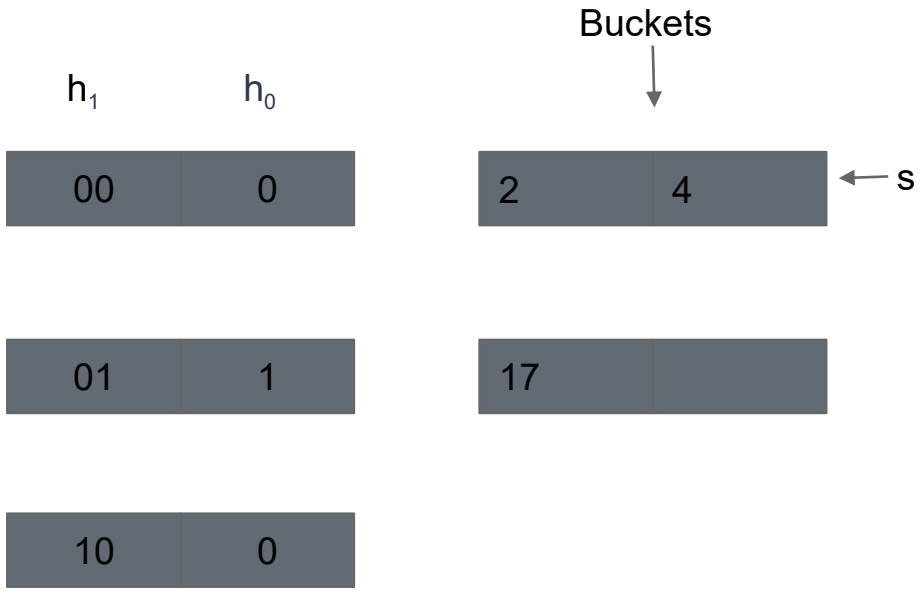
Order of Insertion : 2, 4, 17, 8, 6, 10

To be Inserted : 17



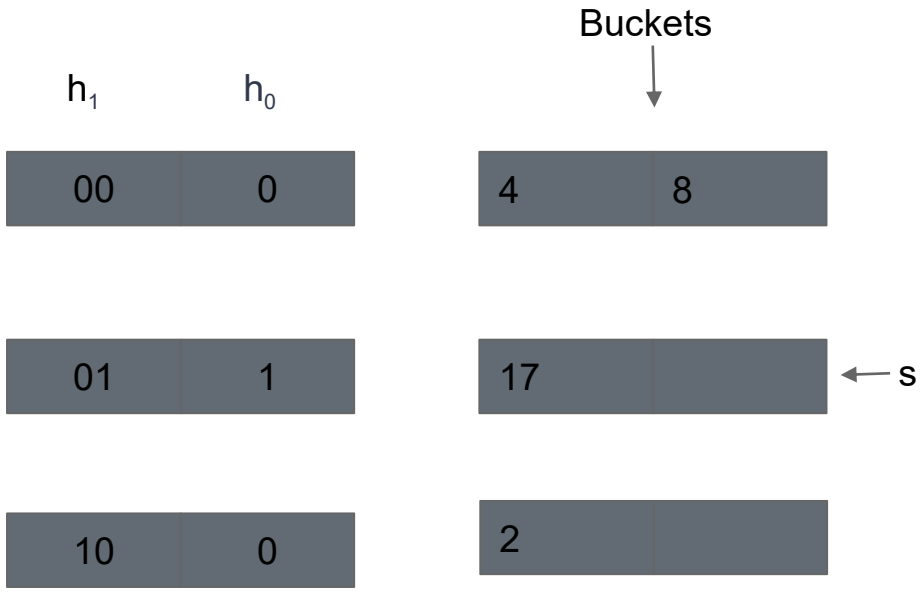
Order of Insertion : 2, 4, 17, 8, 6, 10

To be Inserted : 8

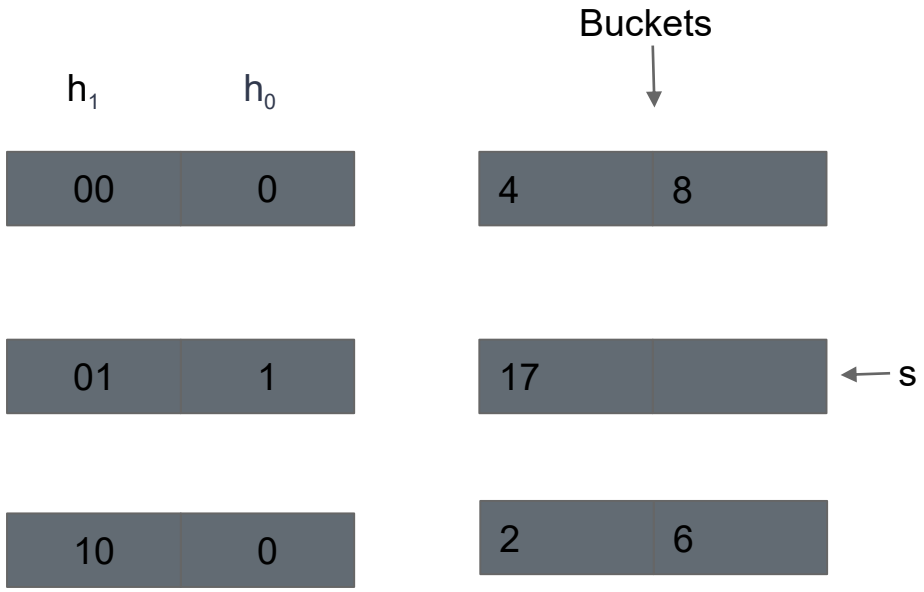


Order of Insertion : 2, 4, 17, 8, 6, 10

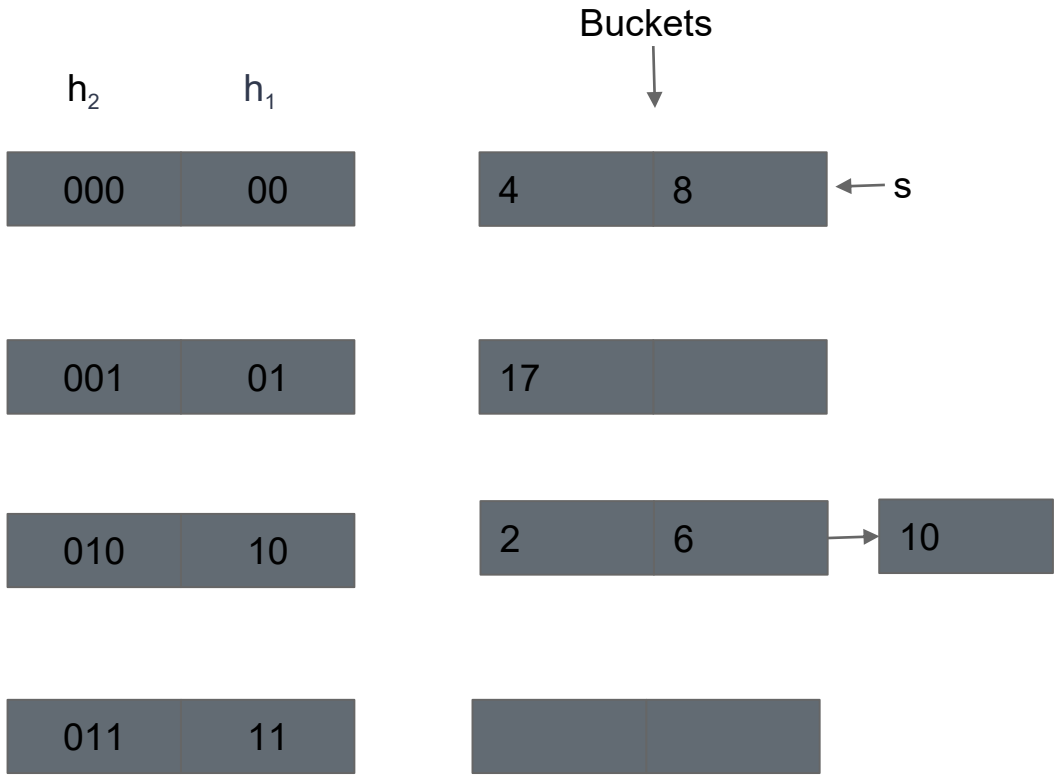
To be Inserted : 6



Order of Insertion : 2, 4, 17, 8, 6, 10



Order of Insertion : 2, 4, 17, 8, 6, 10



Pros:

- Performance does not come down as the data grows in the system. It simply increases the memory size to accommodate the data.
- Since it grows and shrinks with the data, memory is well utilized.
- Good for dynamic databases where data grows and shrinks frequently.

Cons:

- As the data size increases, the bucket size is also increased. These addresses will be maintained in bucket address tables. This is because, the address of the data will keep changing as buckets grow and shrink. When there is a huge increase in data, maintaining this bucket address table becomes tedious.
- Bucket overflow situation will occur in this case too. But it might take little time to reach this situation than static hashing.

CS345 Lecture 2 Notes

Rajat Paliwal
160123046

Linear Hashing :

- Linear hashing is a dynamic hash table algorithm
- Linear hashing allows for the expansion of the hash table one slot at a time.
- The frequent single slot expansion can very effectively control the length of the collision chain.
- The cost of hash table expansion is spread out across each hash table insertion operation, as opposed to being incurred all at once.
- Linear hashing is therefore well suited for interactive applications
- Linear Hashing handles the problem of long overflow chains without using a directory, and handles duplicates.

Algorithm :

First the initial hash table is set up with some arbitrary initial number of buckets. The following values need to be kept track of:

N : The initial number of buckets.: The initial number of buckets.
L : The current level which is an integer that indicates on a logarithmic scale approximately how many buckets the table has grown by.
S: The step pointer which points to a bucket. It initially points to the first bucket in the table.

Algorithm for inserting ‘k’ and checking overflow condition

- $b = h_0(k)$

- if b < split-pointer then
- b = $h_1(k)$

Searching in the hash table for ‘k’

- b = $h_0(k)$
- if b < split-pointer then
- b = $h_1(k)$
- read bucket b and search there
-

Split Pointer Decides which bucket is to be split.

At level L *S* is between 0 and (2^L)

Increment Split Pointer in Round Robin Format.

Splitting is triggered on Overflow in any bucket.

If Split Pointer reaches 0 increment the level value by 1.

Ex :

Order Of Insertion : 2, 4, 33, 8, 6, 10,17

N=2
L=0
S=0.

Initial Table :

h1	h0	
00	0	S
01	1	

After Inserting 2 : No Overflow -> No Splits

h1	h0			
00	0	2		S
01	1			

After Inserting 4 : No Overflow -> No Splits

h1	h0			
00	0	2	4	S
01	1			

After Inserting 33 : No Overflow -> No Splits

h1	h0			
00	0	2	4	S
01	1	33		

After Inserting 8 : Overflow in Row 1 Splitting takes place , add a new bucket And increment the Split Pointer.

h1	h0			
00	0	2	8	
01	1	33		S
10	1	4		

After Inserting 6 : No Overflow -> No Splits

h1	h0			
00	0	2	8	
01	1	33		S
10	1	4	6	

After Inserting 10 : Overflow in Row 3 Splitting takes place , add a new bucket And increment the Split Pointer and as Pointer reaches 0 Increment the level by 1.
Add a block at row 2 (0 based indexing) to include 10.

h1	h0				
000	00	2	8		S
001	01	33			
010	10	4	6	10	
011	11				

After Inserting 17 : No Overflow -> No Splits

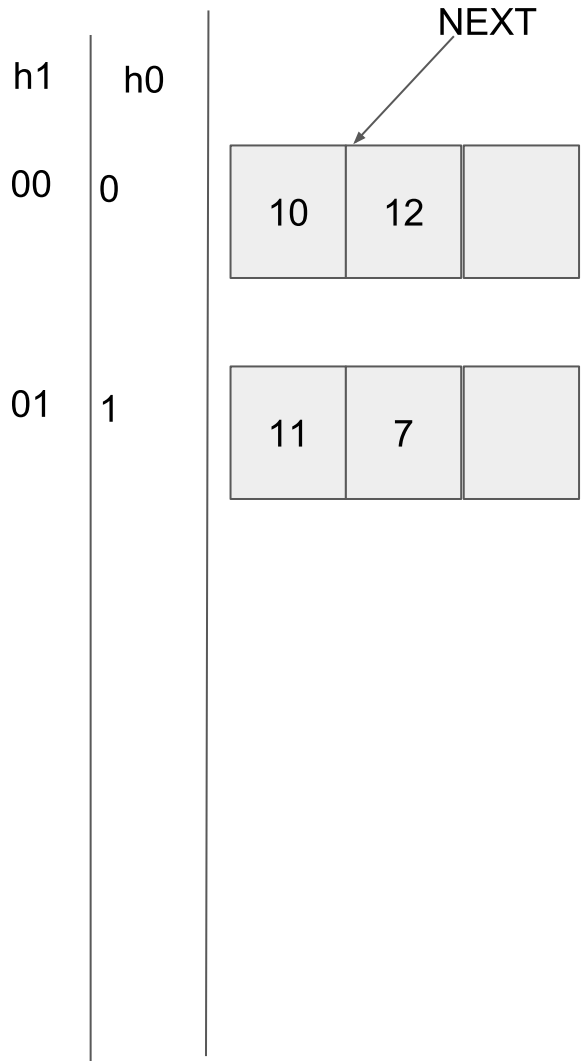
Linear Hashing

By Shubham Goel (160101083)

Lecture on 14 March, 2019 by Dr. Amit Awekar

Linear Hashing

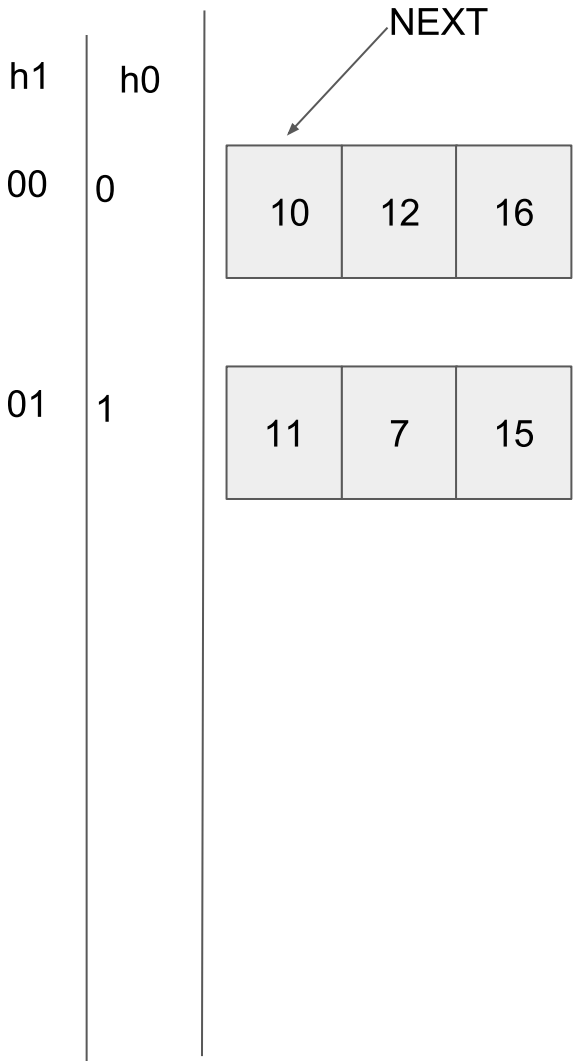
It is a dynamic hashing technique. In this technique there is no need to maintain the pointers. It work on rounds of splitting i.e it splits in a round robin fashion and doesn’t necessarily split the bucket in which overflow is there.



Implementation Details

We start with two buckets (size 3). Some data has been already inserted as shown (but splitting have never occurred). The next pointer points to the bucket to be split when overflow of bucket is encountered while inserting a data element. “h1” denotes the hashing used for the buckets numbered from 0 to next-1. While h0 is used to hash the remaining buckets.

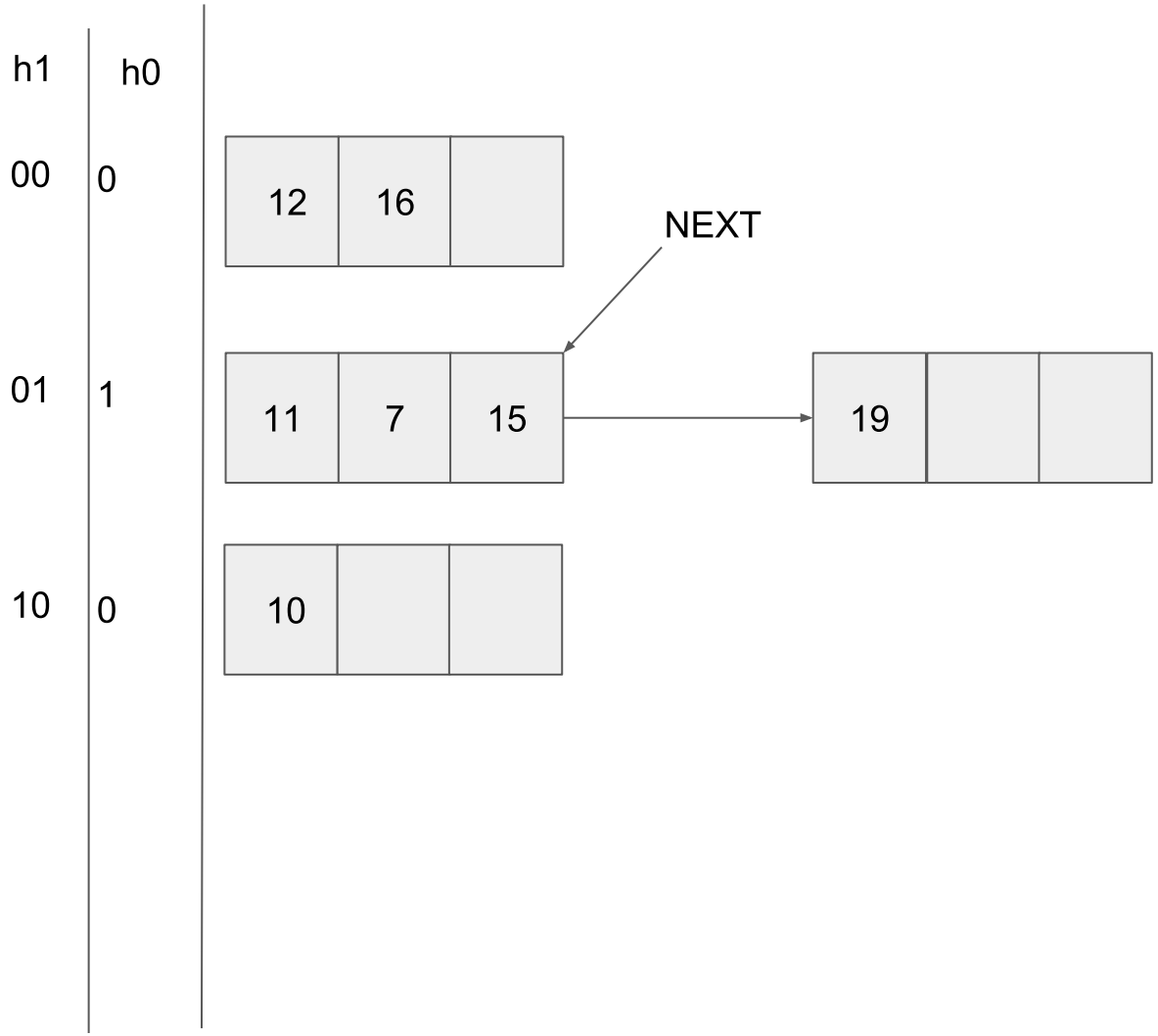
Let us assume we have to insert an element ‘x’. Let us assume it goes in bucket b with hash function h0. If b is greater than equal to next then data is entered into bucket b, otherwise new bucket number is calculated using “h1” and the data inserted according to this new bucket number.
“h0” denotes the current level while h1 denotes the next level.
Level is incremented when the last bucket in the beginning of the current level is splitted and next is set at the beginning.



INSERT 16 and 15

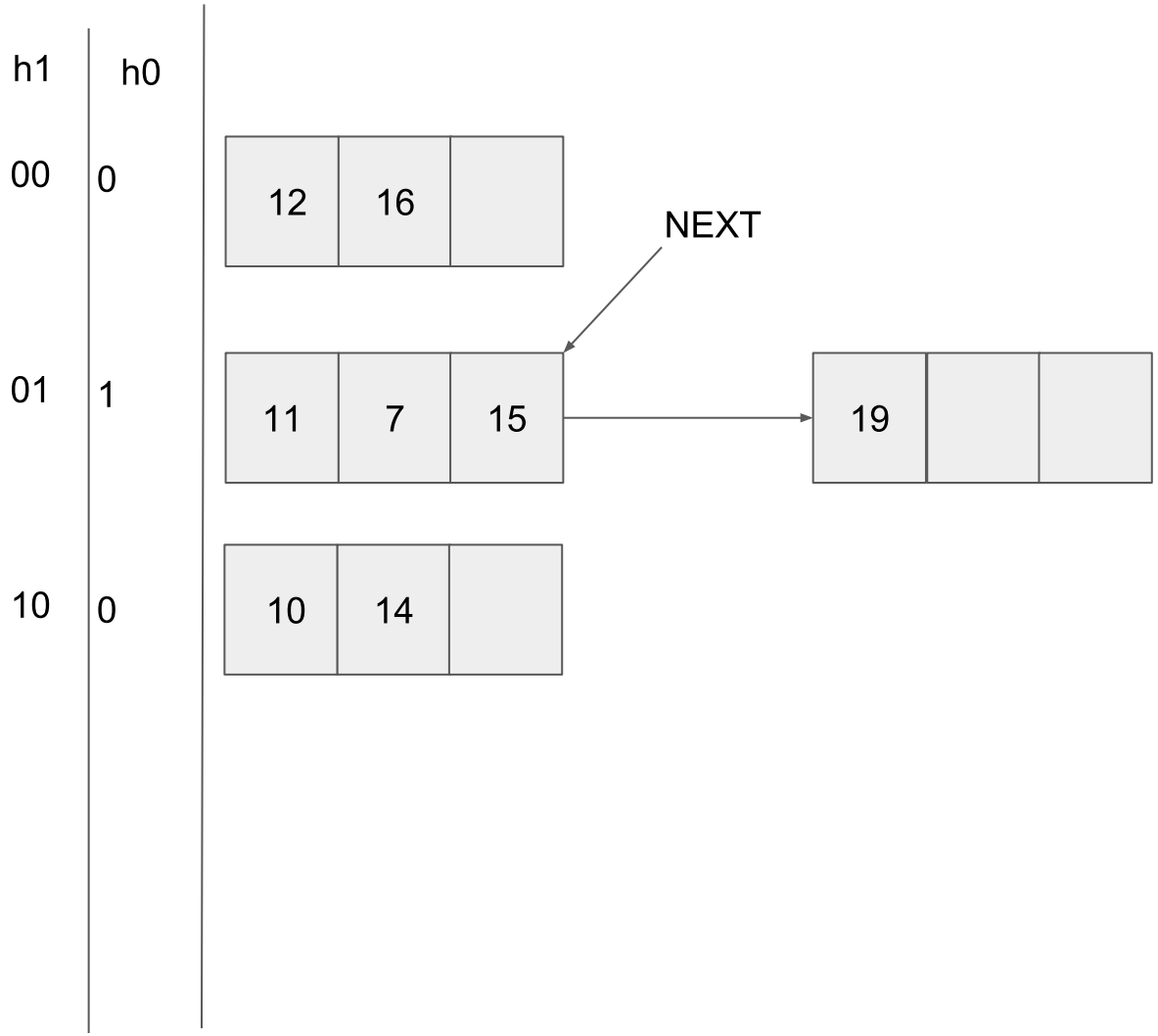
When 16 is inserted the bucket number with h0 is 0 which is greater than equal to next(0) hence 16 is inserted into bucket 0.

When 15 is inserted the bucket number with h0 is 1 which is greater than equal to next(0) hence 15 is inserted into bucket 1.



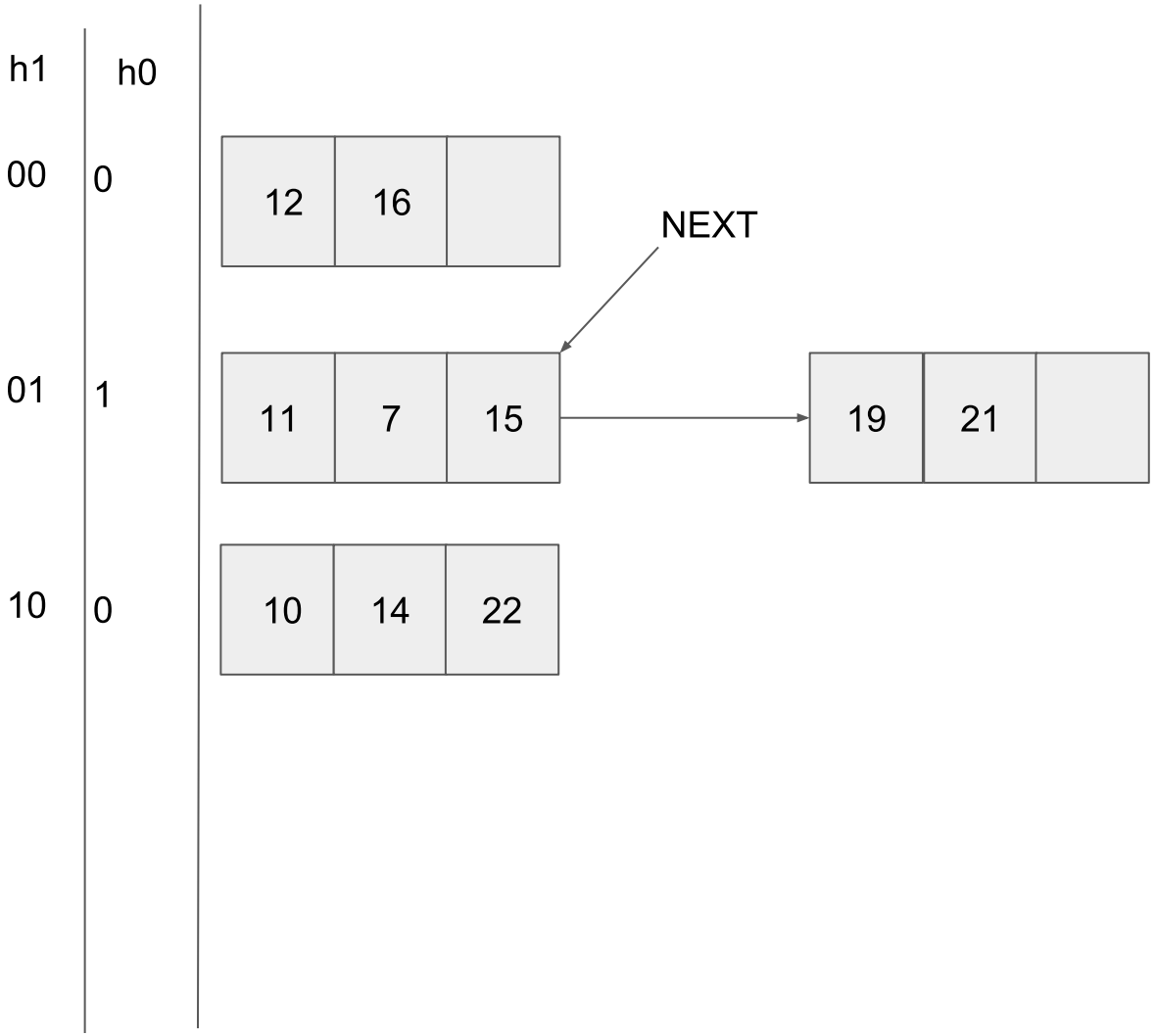
INSERT 19

When 19 is inserted the bucket number with h0 is 1 which is greater than equal to next(0) hence 19 is inserted into bucket 1. But overflow is there and hence spit will be take place on bucket next(0). All the values in bucket next will be reinserted according to h1 and then next will be incremented.



INSERT 14

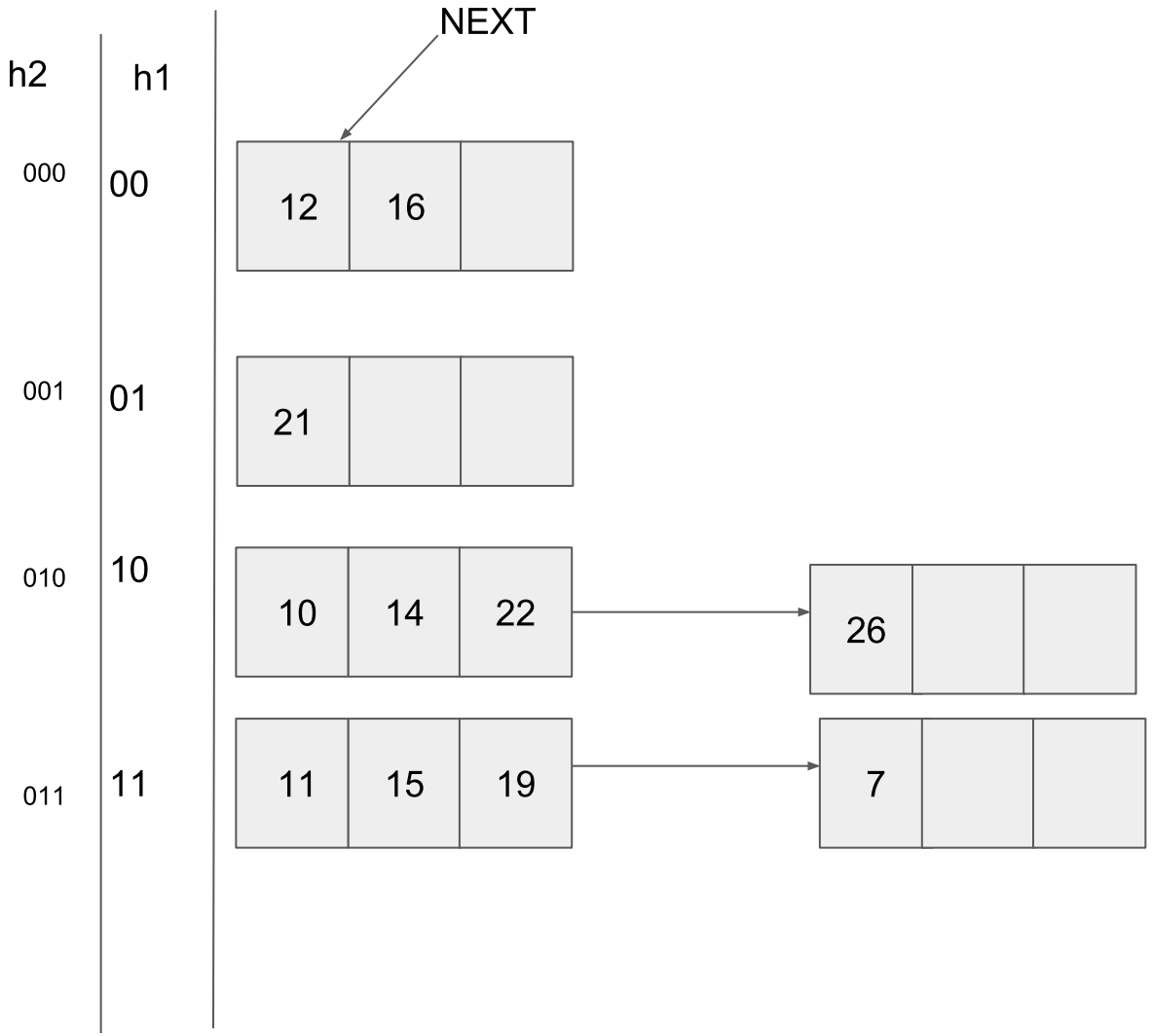
When 14 is inserted the bucket number with h0 is 0 which is less than next(1) hence 14 is reevaluated using hash function h1 which gives value “2”. 14 is inserted into bucket 2.



INSERT 22 and 21

When 22 is inserted the bucket number with h0 is 0 which is less than next(1) hence 22 is revaluated using hash function h1 which gives value “2”. 22 is inserted into bucket 2.

When 21 is inserted the number with hash function h0 is 1 which is greater than equal to next(1) hence 21 is inserted into bucket “1” and overflow is not there.



INSERT 26

When 22 is inserted the bucket number with h0 is 0 which is less than next(1) hence 26 is reevaluated using hash function h1 which gives value “2”. 26 is inserted into bucket 2.

But overflow is there and hence spit will be take place on bucket next(1). All the values in bucket next will be reinserted according to h1.

Since all buckets initially presented on the beginning of level 0 are splitted so level is incremented and next is set to 0.

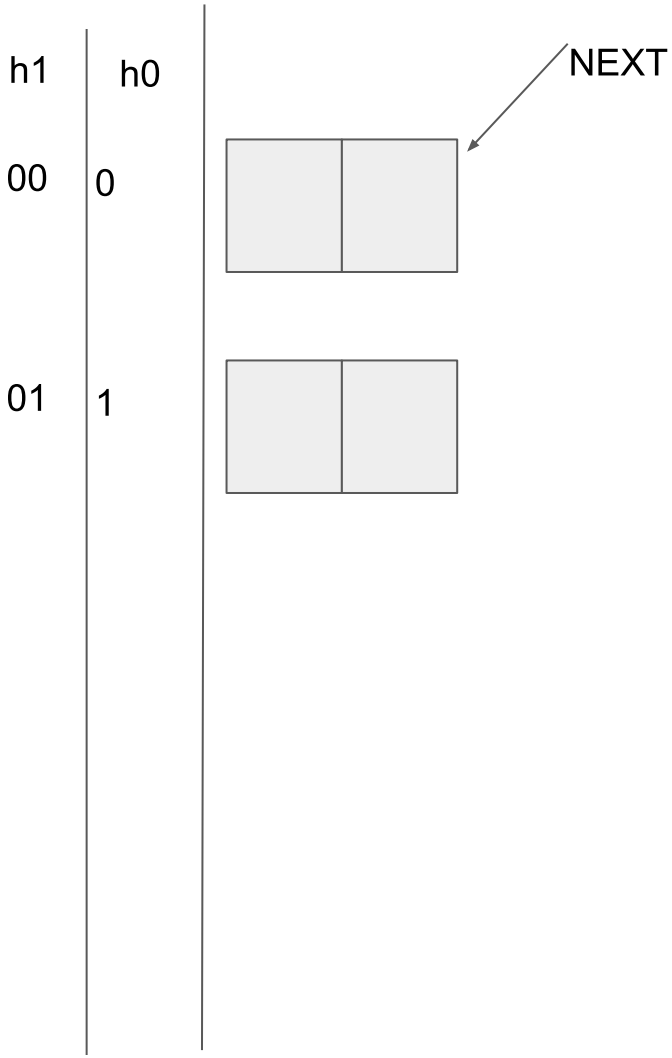
Linear Hashing

By Shaurya Gomber (160101086)

Lecture on 14 March, 2019 by Dr. Amit Awekar

Linear hashing

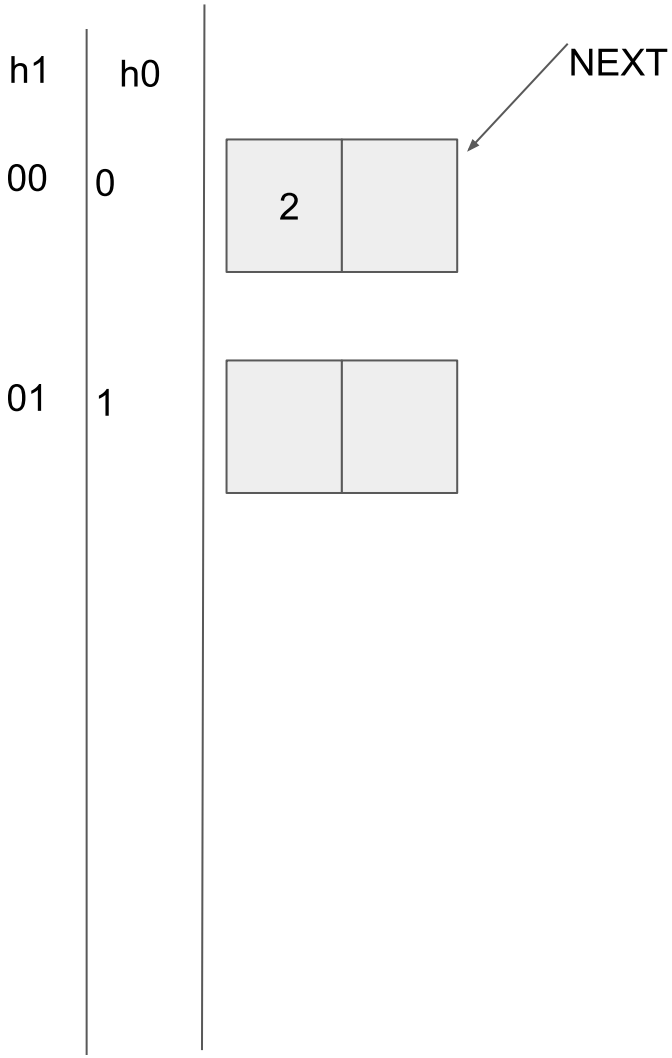
Linear hashing is a dynamic hash table algorithm.Linear hashing allows for the expansion of the hash table one slot at a time. The frequent single slot expansion can very effectively control the length of the collision chain.



Implementation Details

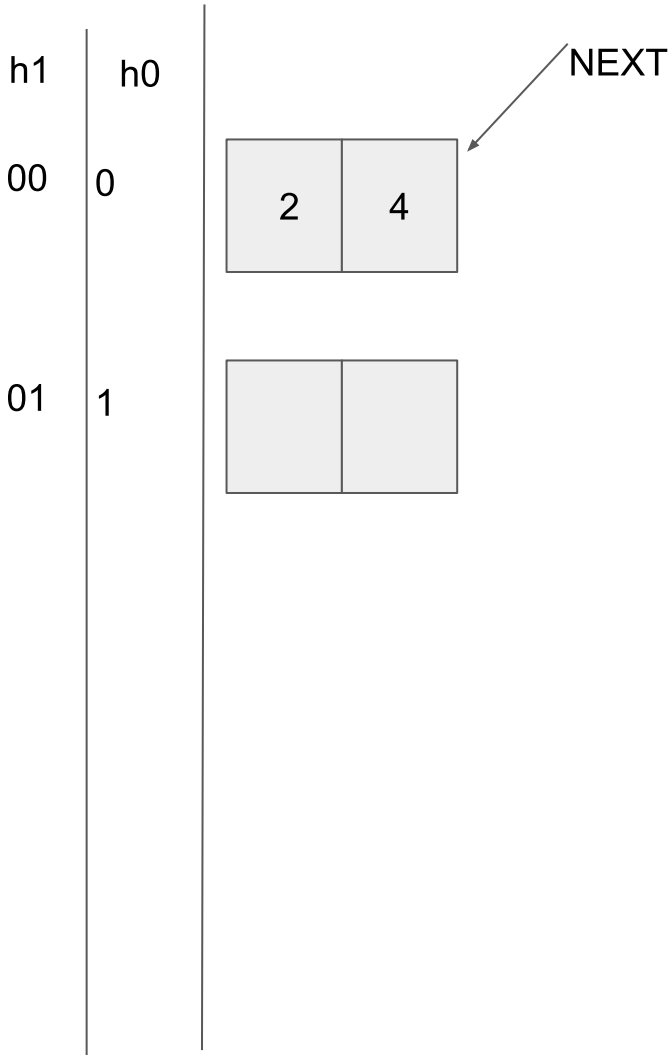
We start with 2 buckets with space of 2 elements each, i.e N=2 at start and level=0 which means that we would be using the first bit (h0) and first two bits (h1) for this iteration. h0 is hash function corresponding to level 0 i.e. 2 buckets . h1 is hash function corresponding to level 1 i.e. 4 buckets. We assume the identity hash function and the hash value hi will be the i least significant bits of the key.
Next points to the bucket that is to be splitted no matter where overflow occurs. After splitting it will point to the next bucket. As soon as it splts the 2^(level+1)th bucket, level is increased by 1, next is set to 0 and we use h{level} and h{level+1} as hash functions.

At any step, compute h{level} of the key. If it is greater than equal to next, just go to the corresponding bucket, put the key there if there is place or if no place is there, use overflow page to accommodate it and split the bucket pointed by next and increment the next.
If h{level} of the key is less than next,use h{level+1} as the key and repeat the above steps.



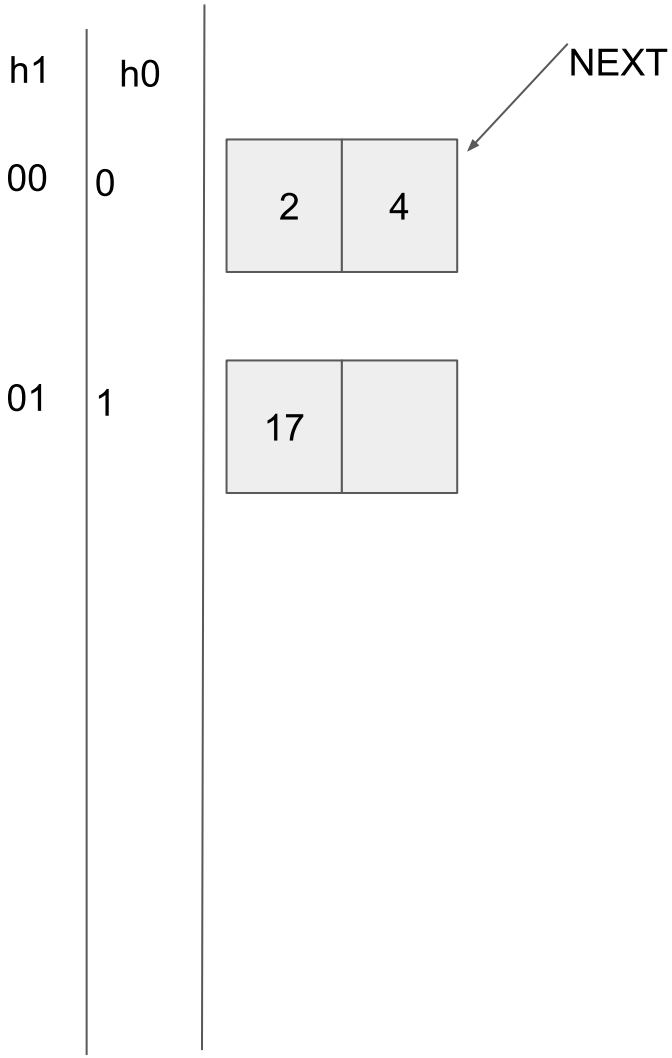
INSERT 2

2 is represented as 10 (h0 = 0) in binary so goes to 1st bucket (0>=next) and as no overflow, so no split operation to be done



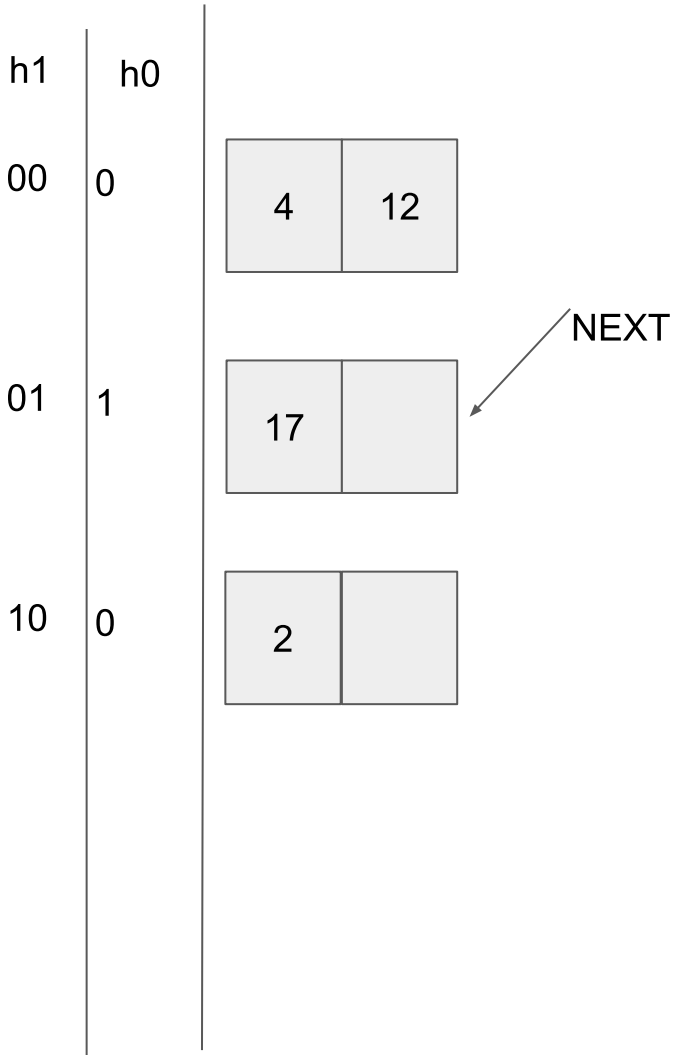
INSERT 4

4 is represented as 100 (h0 = 0) in binary so goes to 1st bucket (0>=next) and as no overflow, so no split operation to be done



INSERT 17

17 is written as 10001 in binary. h0 = 1 so goes in 2nd bucket as 1>=next and it is not full so no overflow and thus no splitting.

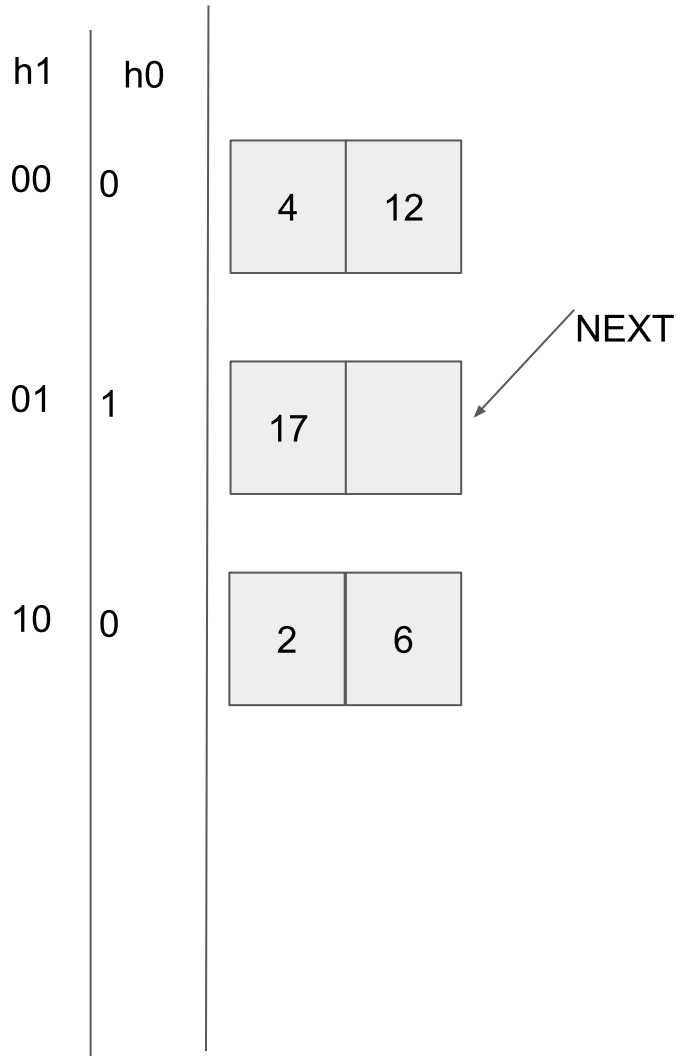


INSERT 12

12 is written as 1100 so h0 = 0 and 0>=next so it should go to the first bucket but it is full, so we would add it in a overflow page to store (4,2)->(12,NULL) in the first bucket but next is also at one and overflow condition so it will split.

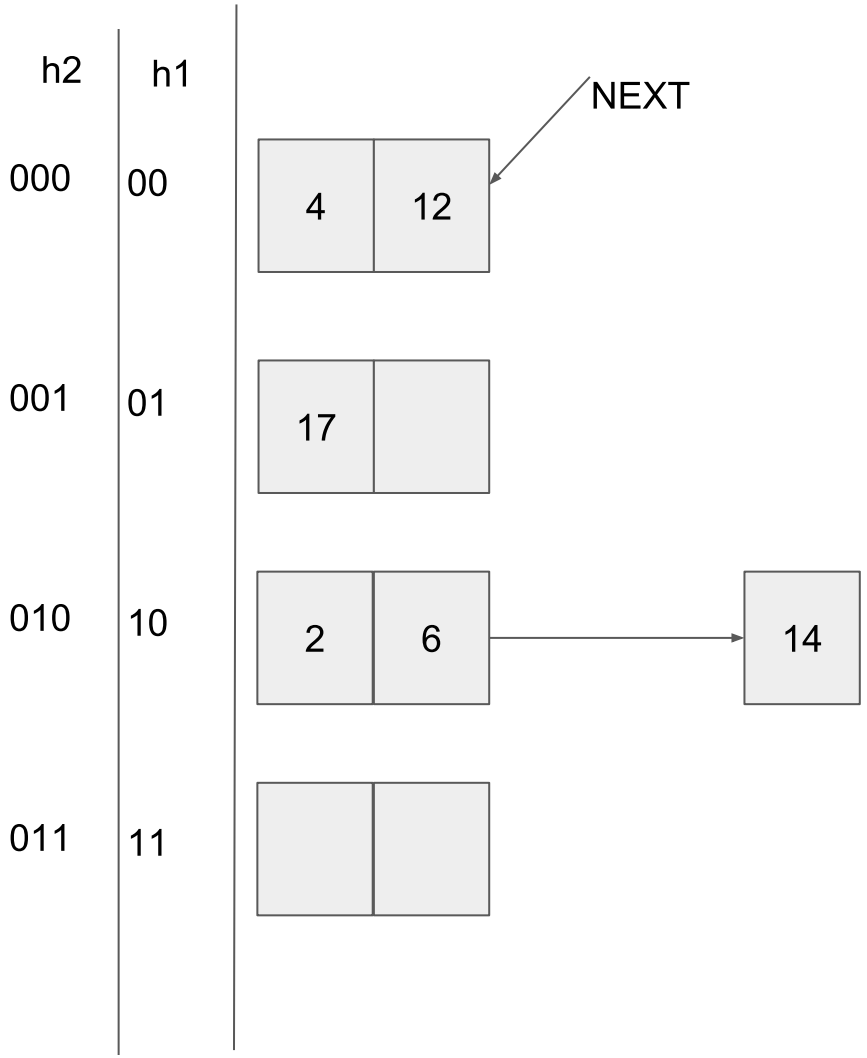
New bucket formed will be mirror of the one splitted with 1 at the most signifacnt bit and rest same as h0. The values 4,2,12 are redistributed to get 4(100) and 12(1100) in the first bucket and 2(10) in the third based on h1.

Next is incremented.



INSERT 6

6 is 110 in binary so h0 is 0 but as 0<next we use h1 i.e 10 to put 6 in the third bucket which has one place left so no overflow and no split



INSERT 14

14 is 1110 in binary so h0=0 is <next so we see h1 which is 10 to put it in third bucket which is full so we use an overflow page to accommodate it and as it is overflow condition we split the second bucket (as pointed by next)

Now next =(1+1)%2=0 i.e we have completed all the 2^(level+1) buckets.

Now since one cycle has finished, we need to update our hash functions to h1 and h2 Also N=N*2=4, level=1 and next is set to 0.

Linear Hashing

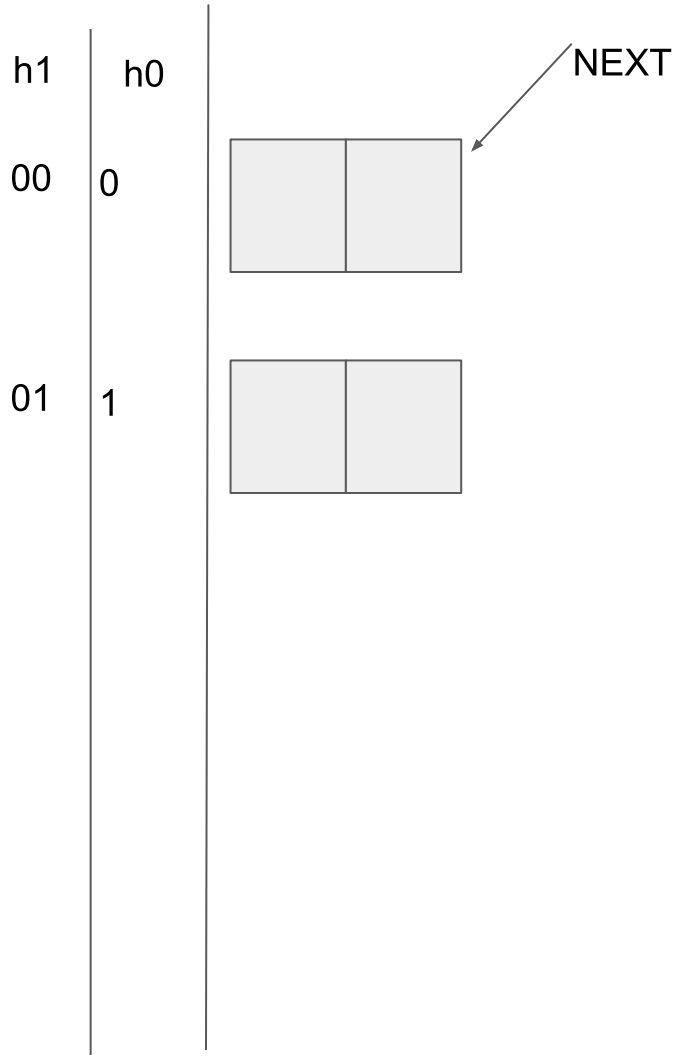
By Archit Jugran (160101087)

Lecture on 14 March, 2019 by Dr. Amit Awekar

Linear Hashing

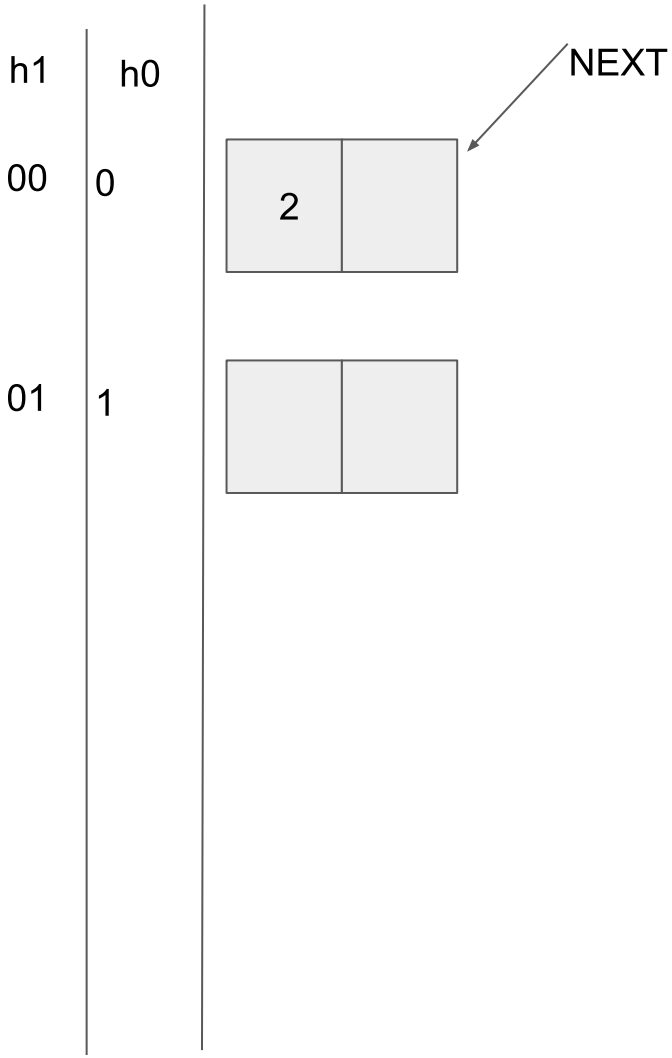
Linear Hashing is a dynamically updateable disk-based index structure which implements a hashing scheme and which grows one bucket at a time.

Implementation : Assumption - Hash function is the identity function i.e f(x)=x. We start with 2 buckets with space of 2 elements each(N = number of buckets). h0 is hash function corresponding to level 0 i.e. 2 buckets . h1 is hash function corresponding to level 1 i.e. 4 buckets (to resolve clash if mirror image is present)
hi(x) = least significant i bits of number x in binary
Next points to the bucket that is to be splitted no matter where overflow occurs.
Now as soon as there is an overflow anywhere, i.e. no space left in bucket in which we want to insert, we will trigger a split operation on the bucket pointed to by “Next” pointer and then increment next by formula
Next= (Next+1)%N
As soon as it splits the 2^(level+1)th bucket, next is set to 0 and we use h{level+1} and h{level+2} as hash functions and level is increased by 1
At any step, compute h{level} of the key. If it is greater than equal to next, just go to the corresponding bucket, put the key there if there is place or if no place is there, use overflow page to accommodate it and split the bucket pointed by next and increment the next.
If h{level} of the key is less than next,use h{level+1} as the key and repeat the above steps.



Initially ,
Next =0
N=2
h0 is hash function corresponding to level 0 i.e. 2 buckets
h1 is hash function corresponding to level 1 i.e. 4 buckets (to resolve clash if mirror is present)
Now as soon as there is an overflow anywhere, i.e. no space left in bucket in which we want to insert, we will trigger a split operation on the bucket pointed to by “Next” pointer and then increment next by formula
Next= (Next+1)%N

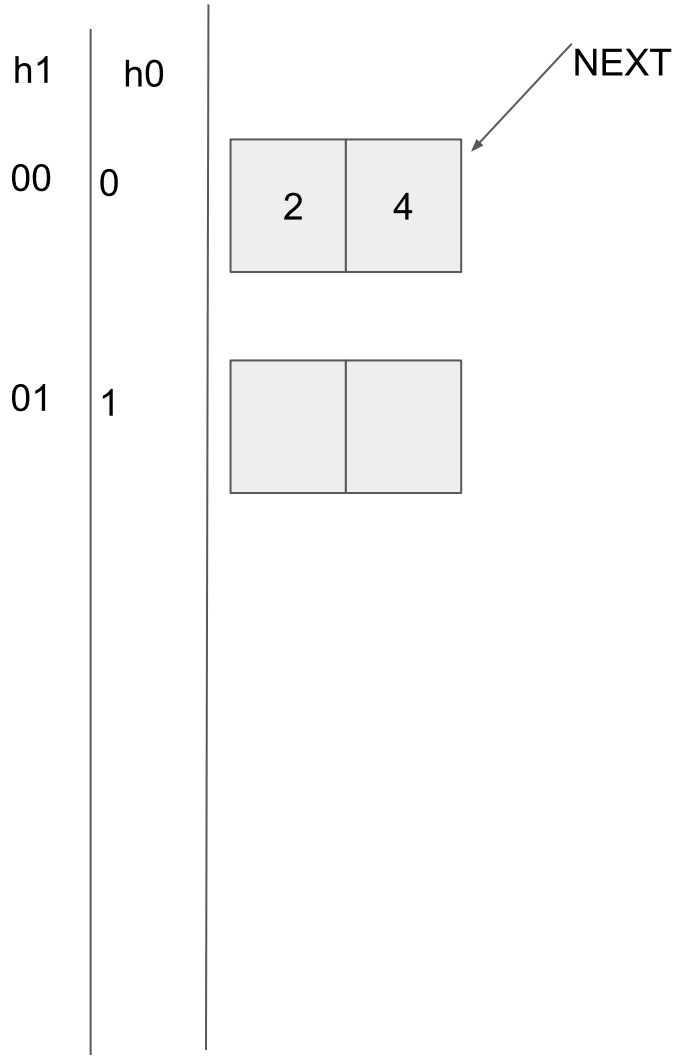
Now we are about to INSERT 2



INSERT 2 DONE

No overflow anywhere , so no split operation to be done

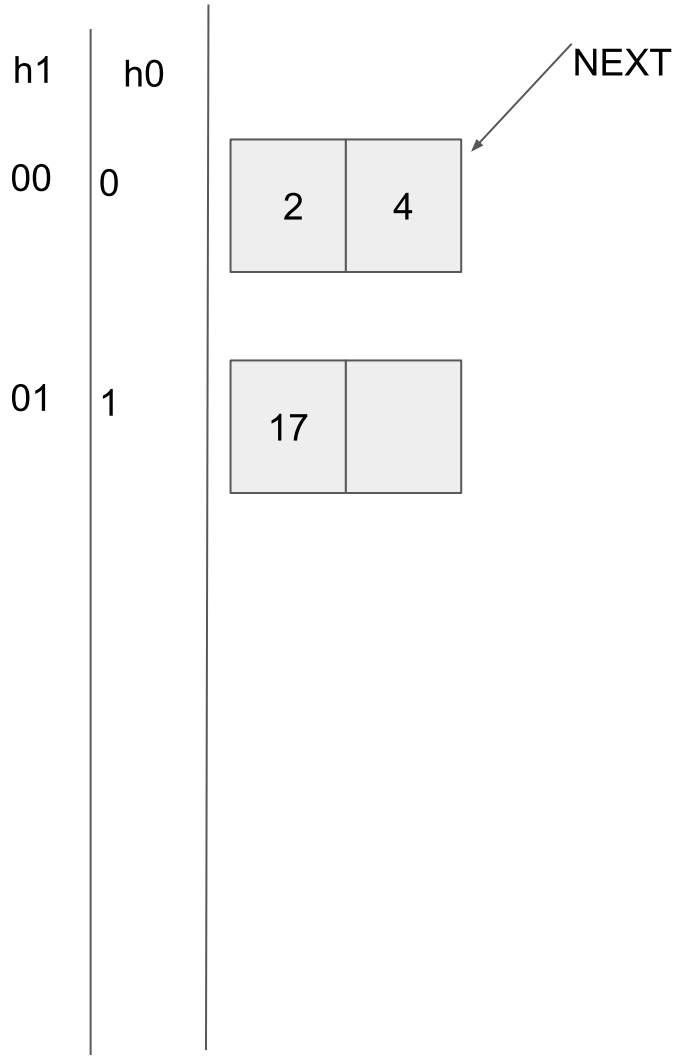
Now we are about to INSERT 4



INSERT 4 DONE

No overflow anywhere , so no split operation to be done

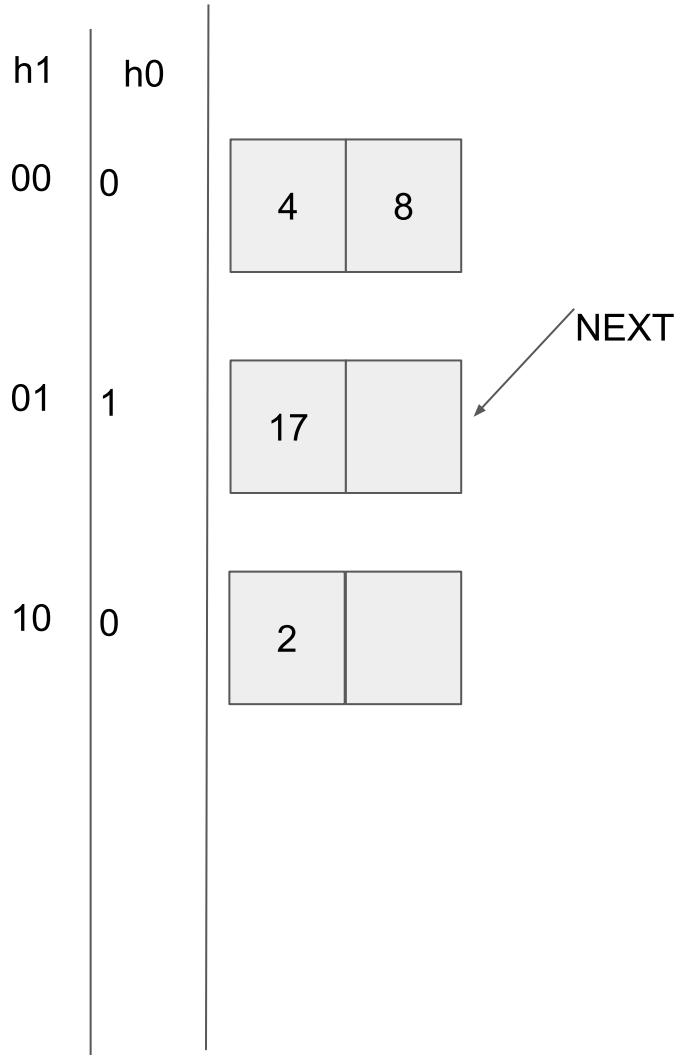
Now we are about to INSERT 17



INSERT 17 DONE

No overflow again so no split

Now we are about to INSERT 8



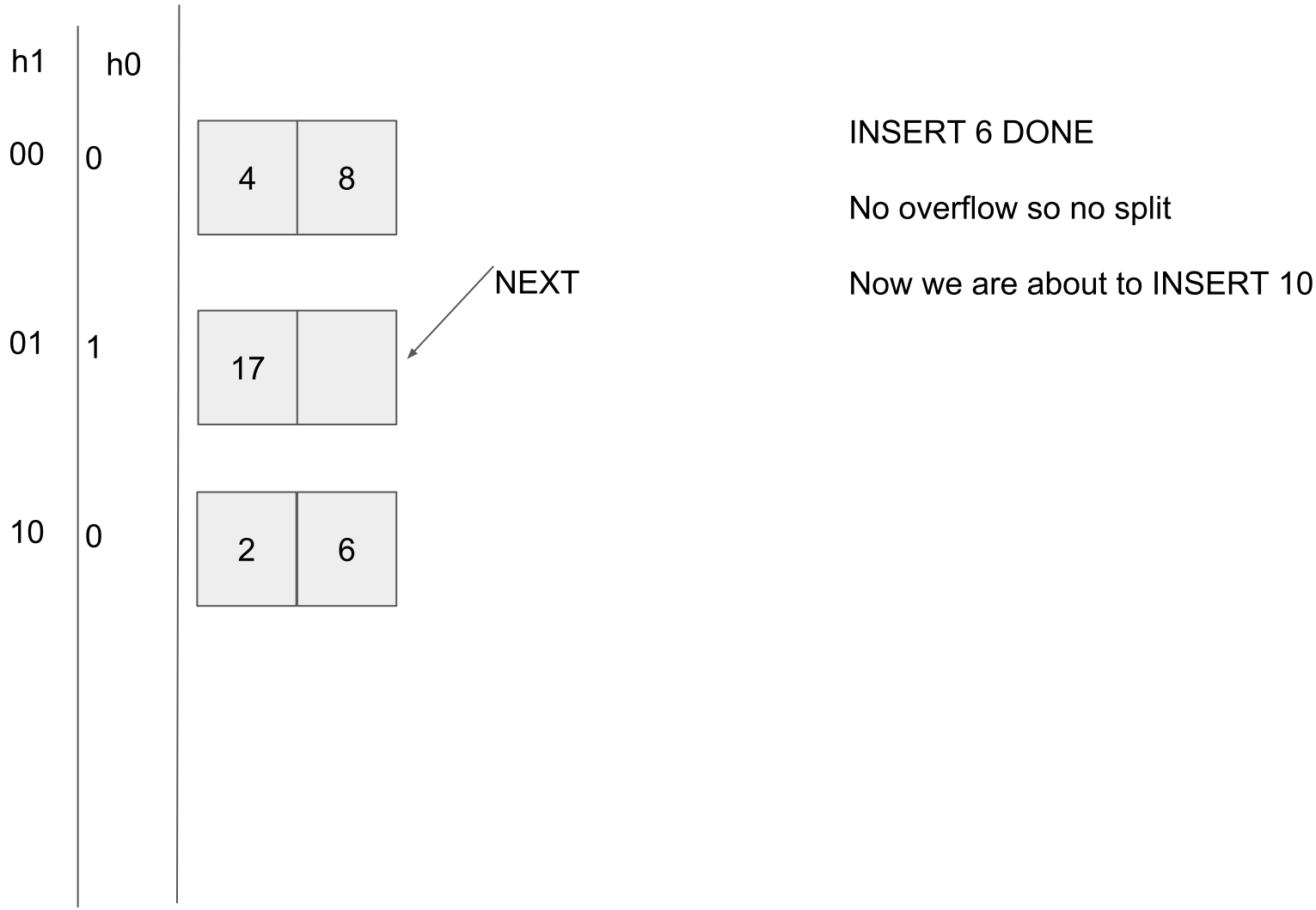
INSERT 8 DONE

Inserting 8 caused an overflow in bucket 0 , hence split operation triggered on bucket pointed by next pointer .

Now we redistributed values 2,4,8 using hash function h1 between the original and new bucket

Now next =1

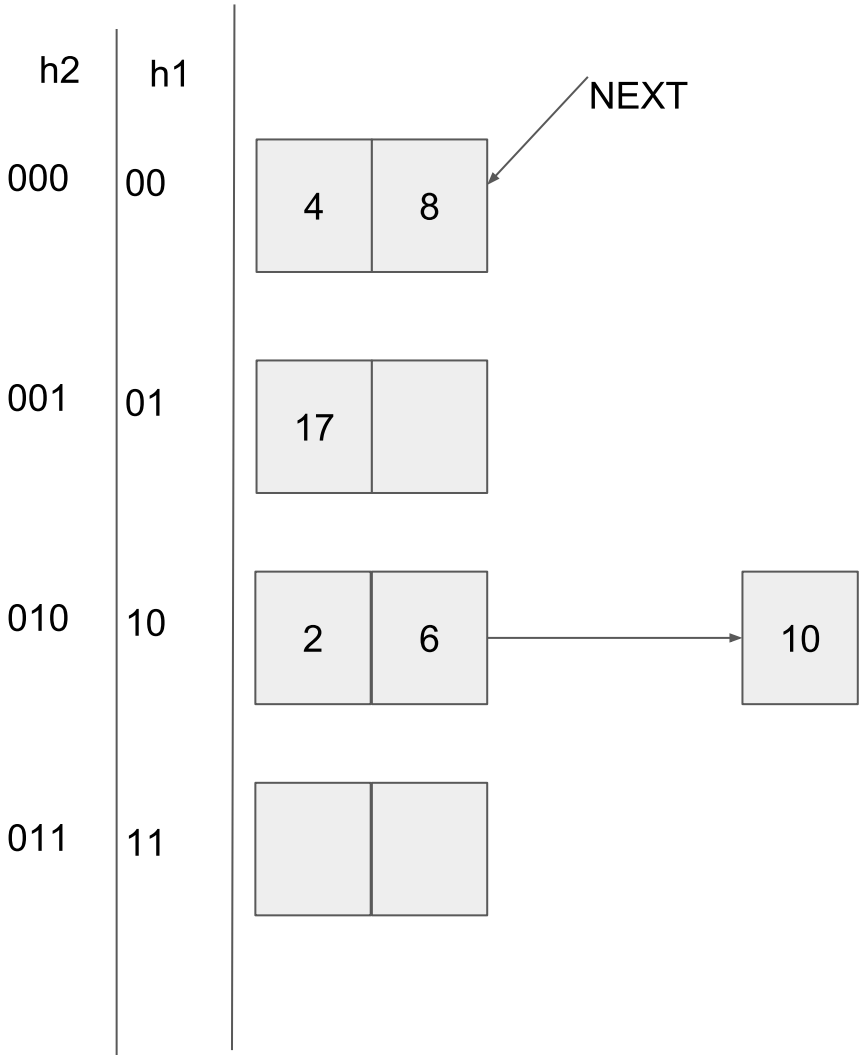
Next we will INSERT 6



INSERT 6 DONE

No overflow so no split

Now we are about to INSERT 10



INSERT 10 DONE

Inserting 10 caused an overflow in bucket 2 , hence split operation triggered on bucket pointed by next pointer (1).

Now we redistributed values 17 using hash function h1 between the original and new bucket

Now next =(1+1)%2=0
Now since one cycle has finished, we need to update our hash functions to h1 and h2
Also N=N*2=4

Linear Hashing

By Rishabh Jain (160101088)

LINEAR HASHING

Linear hashing is a dynamic hash table algorithm invented by Witold Litwin (1980), and later popularized by Paul Larson.

Linear hashing allows for the expansion of the hash table one slot at a time. The frequent single slot expansion can very effectively control the length of the collision chain, thus ensuring that the lookup operation remains O(1).

The cost of hash table expansion is spread out across each hash table insertion operation, as opposed to being incurred all at once. Linear hashing is therefore well suited for interactive applications.

Implementation

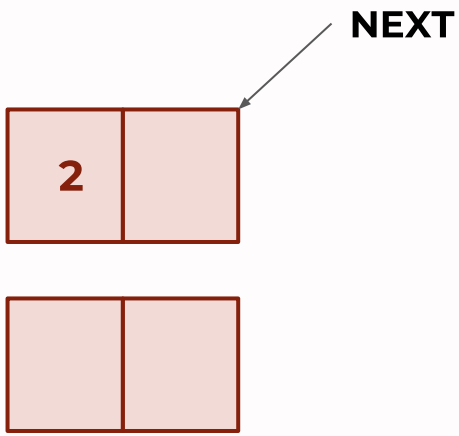
Initially we have 2 buckets with space of 2 elements each, i.e **N=2** at start and **level=0** which means that we would be using the first bit ($\mathbf{h_1}$) and first two bits ($\mathbf{h_2}$) for this iteration.
Here for simplicity sakes, we will assume that our hash function is -

$h_i(x)$: i least significant bits of the key X

NEXT pointer is used to point to the bucket that will be splitted irrespective of where the overflow occurs. After splitting it points to the next bucket.
As soon as the split occurs the bucket no. $\mathbf{2^{(level+1)}}$, the level is increased by 1, **NEXT** is set to **0** and we use $\mathbf{h_{level}}$ and $\mathbf{h_{level+1}}$ as the new hash functions.

If the $\mathbf{h_{level}}$**(key) >= NEXT**, we simply go to the corresponding bucket, and insert the key. If the bucket is full, we use overflow page to accommodate it and split the bucket pointed by next and increment the next.
On the other hand, if $\mathbf{h_{level}}$**(key) < NEXT**, we use $\mathbf{h_{level+1}}$**(key)** as the new hash value and repeat the above steps.

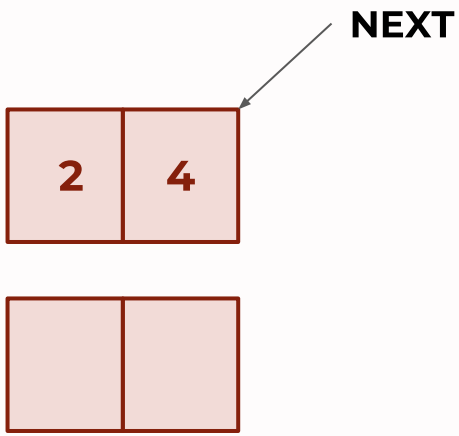
h_2	h_1
00	0
01	1



INSERT 2 :

$(2)_{10} = (1\underline{0})_2$, so it is simply inserted into the first bucket.

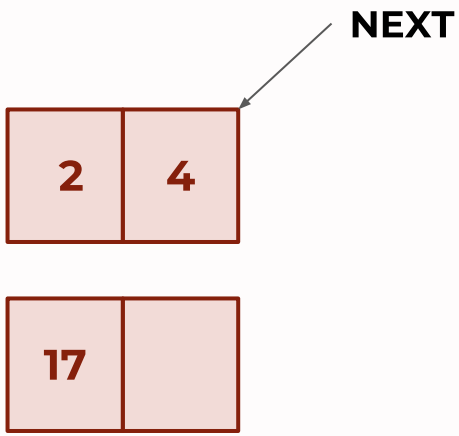
h_2	h_1
00	0
01	1



INSERT 4 :

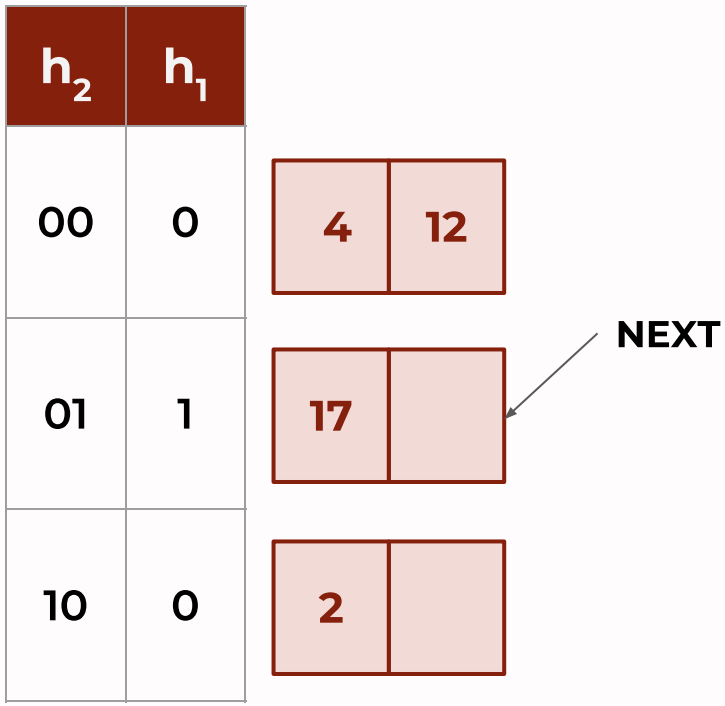
$(4)_{10} = (100)_2$, so it is simply inserted into the first bucket.

h_2	h_1
00	0
01	1



INSERT 17 :

$(17)_{10} = (10001)_2$, so it is simply inserted into the first bucket.



INSERT 12 :

$(12)_{10} = (1100)_2$, but the first bucket is full.
So we put it in an overflow page. We see that NEXT is also at the same bucket and overflow condition is valid, so we split the node.

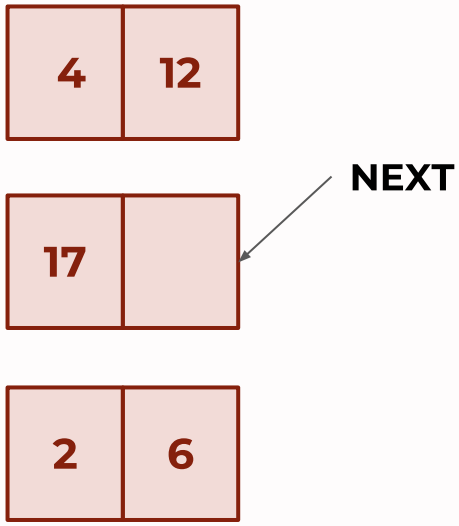
New bucket formed is the mirror of the original bucket with 1 at the most significant bit.

The values 4, 2, 12 are now rehashed using h_2 to get their new buckets.

$(4)_{10} = (100)_2$
$(2)_{10} = (10)_2$
$(12)_{10} = (1100)_2$

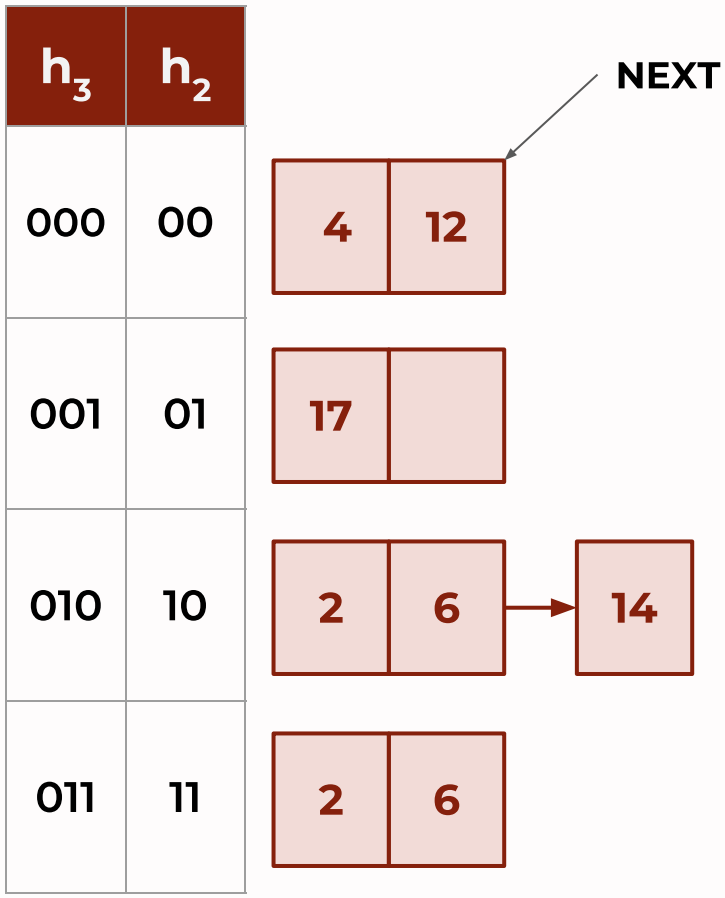
Next is updated.

h_2	h_1
00	0
01	1
10	0



INSERT 6 :

$(6)_{10} = (\underline{110})_2$, so it is simply inserted into the third bucket.



INSERT 14 :

$(14)_{10} = (1110)_2$, but the third bucket is full.
So we put it in an overflow page. We see that NEXT is also at the same bucket and overflow condition is valid, so we split the second bucket.

We observe that we have created all buckets for this level and thus we update our hash functions

NEXT is set to the first bucket.