## Chapter 3.3 Hash Table

*Definition:*

*Hash Table can be used as Index and Main Memory Data Structure. In such structure, there has Hash Function h, and it uses Data Key as Parameter and calculate the Integer which belongs to (0, B - 1), among which B is the number of bucket.*

Bucket Array, is an array with the Sequence from 0 to B - 1, among which there includes B Bucket Array, among which includes B linked list heads, each one corresponding to one bucket in the Array. If Query Key of Record equals to K, then we can link the record to bucket index h(K) to store Query Key.

### Chapter 3.3.1 Auxiliary Hash Table

### *Introduction:*

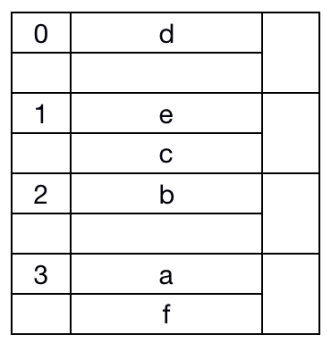
### Since Auxiliary Hash Table is too big to stay in Main Memory, therefore we put it into Auxiliary Storage, and there have tiny and important difference between Auxiliary Hash Table and Main Memory Hash Table.

*First of all, Bucket Array consists of the Storage Block but not Pointers that point to Linked List Head. Through Hash Table h, we can hash all records in this Hash Table. If there has too much records in the Bucket, then we can add extra Linked List into Bucket to save much more records.*

Assume that, given any i, then the Location of first Storage Block in Bucket i can be reached.

### *Example:*

Hash Table is just as shown below. In order to make it easy to manage, we need to assume that each Storage Block can store two pieces of records, and B = 4, and the return value in Hash Table h would between 0 and 3. We has listed some Hash Table records. In the Table below, then Key Value is Alphabet Number range from a to f. Assume that h(d) = 0, h(c) = h(e) = 1, h(b) = 2 and h(a) = h(f) = 3. The record range distribution is as below.



Attention that, the right side of each Storage Block has one tiny square, this tiny square represents additional information which is saved in the Storage Block.

### Chapter 3.3.2 Insertion into Hash Table

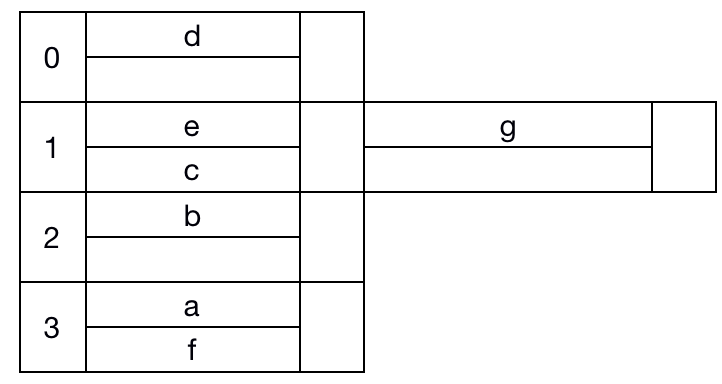
### *Principle:*

When one record with Query Key K needs to be inserted into Hash Table, then we need to calculate h(K).

* *If there has extra space for Bucket Number h(K), then this record would be stored into such bucket.*
* *Otherwise if there has no such space for Bucket with Number h(K), then we need to add one storage block of Bucket into List and store such block into the additional storage block of Bucket.*

### *Example:*

Assume that we add one Key Value which equals to g and h(g) = 1. We need to store such record into No.1 Bucket. However, there already have two Blocks, we need to add one extra new Block and linked the first Block. The Hash Table is just as image below.



### Chapter 3.3.3 Deletion from Hash Table

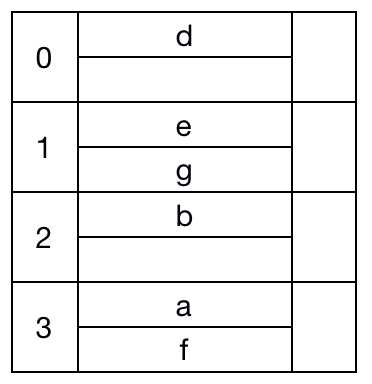
### *Principle:*

The method to delete the record with Query Key K is just the same as the method to insert.

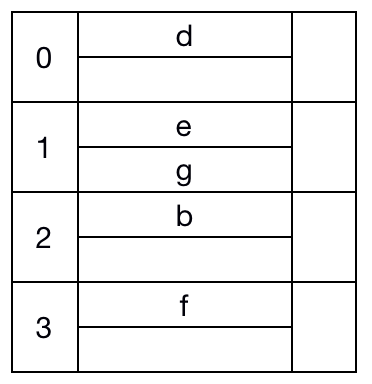
*We need to find the h(K) bucket and search the Query Key with Data Value K, and to find the record to be deleted. If the record can be moved among Blocks, after delete it, we can merge Storage Block in the same linked chain.*

### *Example:*

We need to delete the Data Record with Key Value c from Hash Table. Known from h(c) = 1, we can find the first Bucket and the record with Key Value c. After delete the record c from Hash Table, we need to remove the record with Key Value g from the Second Linked List to the first List and delete the second Storage Block.



We also delete the record with Key Value a. For this Key - Value, we find the third Storage Block and delete this record, and move the left block forward to make it much more compact.



### Chapter 3.3.4 Efficient of Hash Index

### *Principle:*

### The optimal situation is that there would exist enough buckets and most of which consist of single block. If that, then the general Query would only need one Disk I/O and File Insertion and Deletion would only need two times Disk I/O. Such result would be much better than Sparse Index, Dense Index or B - Tree.

However, as the number of File is growing, then the situation that there would have a lot of Blocks linked after the Linked List. Under this kind of situation, we need to Query in the Long Linked List, and each block only needs one time Disk I/O. So we need try to decrease the block number of each Bucket.

*So far, we need to know that the Hash Table is called Static Hash Table, since the number of Bucket B would never change.* And, there also have some other kinds of *Dynamic Hash Table, they allow the number of Bucket B to change, and B equals to the total number of record number divide by the contained record numbers, which means that, each bucket would contain only one Storage Block.* We will discuss two methods:

1. *Extensible Hash Table:*

*The first method would double the number of B when it is considered to small.*

1. *Linear Hash Table*

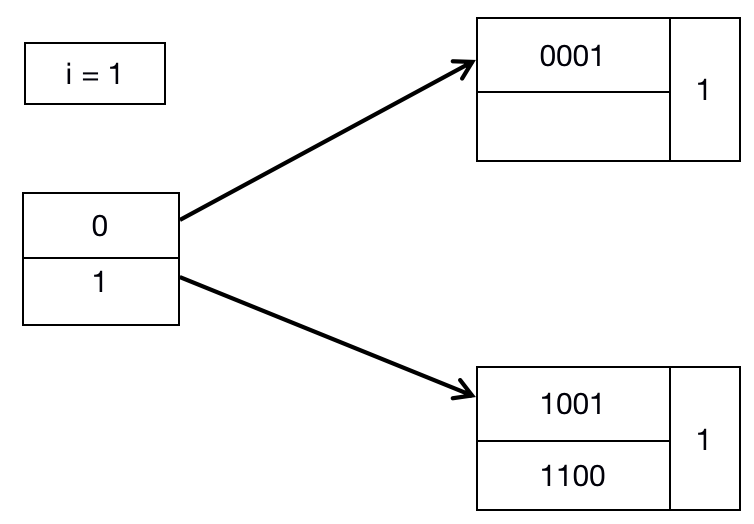
*The second method would add 1 to B when the Statistical Number of File indicates.*

### Chapter 3.3.5 Extensible Hash Table

### *Definition:*

The first kind of Dynamic Hash Method is called Extensible Hash Table. It can only add below items to the Simple Static Hash Table:

* Introduce the Intermediate Level for the Bucket, which means to use the Pointer Array which points to the Bucket to represent Bucket, but not the Data Block Array itself to represent Bucket.
* The Pointer Array can increase, it’s length is always the value of 2 power, because each time the length of Array increases, then the number of Bucket would be doubled.
* However, not each Bucket has it’s Data Block; If the records in some Buckets can be put into one Bucket, then these buckets can share one Block.
* Hash Function h would calculate one K bytes Binary Sequence for each Key, if the value of K is big enough, such as 32. But, the number of Bucket would always be several bytes start from the first byte of Binary Sequence or from the last byte of Binary Sequence, the number of these bytes is less than K, we can assign it the number of i. Which is to say that when i equals to the used bytes, then the Bucket Array would have 2^i items.



### *Example:*

The table above has shown one small extensible Hash Table. For simpler, we assume that K = 4, which means by using Hash Function, we can generate 4 bytes Binary Sequence. When we need to use one of byte of the Sequence, then just like i = 1, there will have two items which corresponds to 0 and 1.

We can check from Hash Table that the first Block save the Query Record starts with 0; while the Second Block save the Query Record starts with 1. For convenience, here the data stored in the Hash Table is the Binary Sequence Number. Therefore, the first block saves one record with Data Value equals to 0001 and the second block saves two records with Data Value equals to 1001 and 1100.

Here, we need to pay attention that, the number after the Data Value in the Bucket which are all as 1. This number is used to signify that how much bytes are used to decide the membership of the record in the Bucket.

In the example above, there has only one byte of the Data Value Sequence to decide where to locate among all Bucket Blocks. However, as the size of Bucket increases, then the different bytes in different Bucket Block may need to be considered, which is to say that, the size of Bucket Block is decided by the biggest binary byte, but some Bucket Block may need less bytes.

### Chapter 3.3.6 Insertion into Extensible Hash Table

### Chapter 3.3.7 Linear Hash Table

### Chapter 3.3.8 Insertion into Linear Hash Table