

Module – 5 Part A

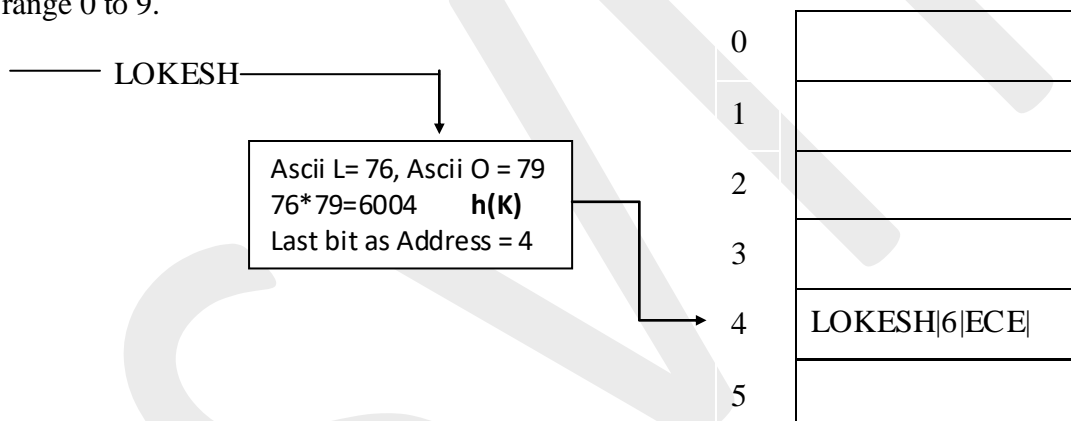
HASHING

Introduction:

Sequential access efficiency is $O(N)$. B trees/ B+ trees efficiency is $O(\log_K N)$. Both are dependent on 'N' where N is the number of records in the file i.e file size. So there was a need for a technique whose efficiency is independent of the size of the file. This lead to Hashing which provides an efficiency of $O(1)$.

A Hash function is like a black box that produces an address every time a key is dropped. The process of producing an address when a key is dropped to hash function is called Hashing. Formally hash function is given by $h(K)$ that transforms a key 'K' into an address. The resulting address is used to store and retrieve the record.

Example: Consider a key LOKESH which is dropped in to a hash function which does the following computation to transform the key to its address. Assume the address space is 10. i.e in the range 0 to 9.



In indexing we associate a key with relative record number (RRN). Hashing is also similar to indexing in this way. i.e associating a key to a address (relative record number). But hashing differs from indexing in two major ways. They are:

1. With hashing the address generated appear to be random. In indexing association of key with RRN is sequential whereas in hashing this association is random.
2. In hashing two different keys may be associated (Hashed) to same address. This is called collision. Whereas in indexing there is no collision. i.e there is no chance of two different keys being associated to same RRN.

The ideal solution is to find or devise a transformation algorithm that avoids collision altogether. Such an algorithm is called a perfect hashing algorithm. It is near to impossible to find a perfect

hashing algorithm. So a more practical solution is to reduce the number of collision to an acceptable level. Some of the ways to reduce the collusion to acceptable level are:

1. Spread out the records.
2. Use extra memory.
3. Put more than one record at a single address.

Spread out the records: Collision occur when two or more records compete for the same address. If we could find hashing algorithm that distributes the records fairly randomly among the available addresses, then we would not have large number of records clustering around certain address.

Use extra memory: It is easier to find a hash algorithm that avoids collisions if we have only a few records to distribute among many addresses than if we have about the same number of records as addresses. Example chances of collisions are less when 10 records needs to be distributed among 100 address space. Whereas chances of collision is more when 10 records needs to be distributed among 10 address space. The obvious disadvantage is wastage of space.

Put more than one record at a single address: Create a file in such a way that it can store several records at every address. Example, if each record is 80 bytes long and we create a file with 512 byte physical records, we can store up to six records at each address. This is called as bucket technique.

A Simple Hashing Algorithm

This algorithm has three steps:

1. Represent the key in numerical form.
2. Fold and add.
3. Divide by a prime number and use the remainder as the address.

Step 1: Put more than one record at a single address.

If the key is string of characters take ASCII code of each character and use it to form a number. Otherwise no need to do this step.

Example if the key is LOWELL we will have to take ASCII code for each charaters to form a number. If the key is 550 then no need to do this step. We will consider LOWELL as the key.

LOWELL – 76 79 87 69 76 76 32 32 32 32 32 32

32 is ASCII code of blank space. We are padding the key with blank space to make it 12 characters long in size.

Step 2: Fold and add.

In this step Folding and adding means chopping off pieces of the number and add them together. Here we chop off pieces with two ASCII numbers each as shown below:

7679 | 8769 | 7676 | 3232 | 3232 | 3232 ----- Folding is done

On some microcomputers, for example, integer values that exceed 32767 causes overflow errors or become negative. So addition of the above integer values results in 33820 causing an overflow error. Consequently we need to make sure that each successive sum is less than 32767. This is done by first identifying the largest single value we will ever add in summation and making sure after each addition the intermediate result differs from 32767 by that amount.

Assuming that keys contain only blanks and upper case alphabets, so the largest addend is 9090 corresponding to ZZ. Suppose we choose 19937 as our largest allowable intermediate result. This differs from 32767 by much more than 9090, so no new addition will cause overflow.

Ensuring of intermediate result not exceeding 19937 is done by using mod operator, which returns the remainder when one integer is divided by another.

$$7679 + 8769 = 16448 \Rightarrow 16448 \bmod 19937 \Rightarrow 16448$$

$$16448 + 7676 = 24124 \Rightarrow 24124 \bmod 19937 \Rightarrow 4187$$

$$4187 + 3232 = 7419 \Rightarrow 7419 \bmod 19937 \Rightarrow 7419$$

$$7419 + 3232 = 10651 \Rightarrow 10651 \bmod 19937 \Rightarrow 10651$$

$$10651 + 3232 = 13883 \Rightarrow 13883 \bmod 19937 \Rightarrow 13883$$

The number 13883 is the result of the fold and add operation.

Step 3: Divide by the Size of the Address Space.

In this step the number produced by the previous step is cut down so that it falls in the within the range of addresses of records in the file. This is done by dividing the number by address size of the file. The remainder will be home address of record. It is given as shown below

$$a = s \bmod n$$

If we decide to have 100 addresses i.e 0-99 than $a = 13883 \bmod 100$, which is equal to 83. So hash address of LOWELL will be 83.

Hashing Functions & Record Distribution:

The figure below shows three different distributions of seven records among nine addresses.

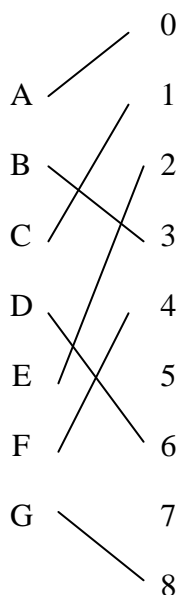
If the hash function distributes seven records in ten addresses without any collision then it is called as **uniform distribution** as shown in figure 'a'.

If all the seven records are hashed to the same home address then it is called as **worst distribution** as shown in the figure 'b'

Figure 'c' illustrates a distribution in which the records are somewhat spread out, but with a few collisions. This is called as **Acceptable distribution**.

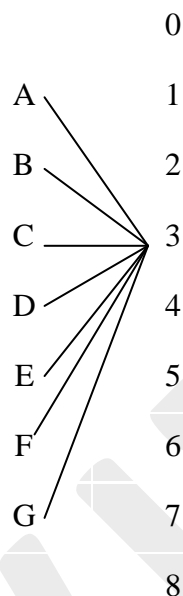
Uniform Distribution

Records Addresses



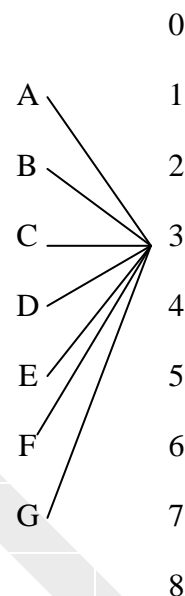
Worst Distribut ion

Records Addresses



Acceptable Distribution

Records Addresses



Some Other Hashing Methods

Some methods those are potentially better than random. The simple hashing algorithm explained in section 7.2 which has three steps are designed to take advantage of natural ordering among the keys. The next two methods can be tried when for some reasons, the better than random methods do not work.

- **Square the key and take the mid (Mid Square method):** This method involves treating the key as single large number, squaring the number and extracting whatever number of digits are required from the middle of the result.
For example: consider the key 453, its square is $(453)^2 = 205209$. Extracting the middle two digits yields a number 52 which is between 0 – 99.
- **Radix Transformation:** This method involves converting the key from one base system to another base system, then dividing the result with maximum address and taking the reminder.

For example: If the hash address range is 0 – 99 and key is $(453)_{10}$. Converting this number to base 11 system results in $(382)_{11}$. Then $382 \bmod 99 = 85$. So 85 is the hash address.

7.3.2 Predicting the Distribution of Records.

7.3.3 Predicting Collision for a Full File.

Refer class notes & problems solved in the class

7.4 How Much Extra Memory should be Used.

Collision Resolution by Progressive Overflow

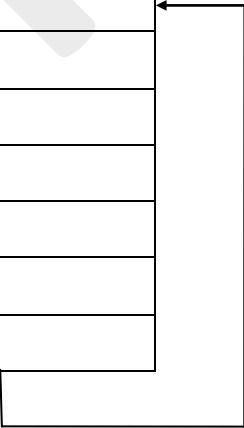
- **Progressive overflow** is a collision resolution technique which places overflow records at the first empty address after the home address; if the file is full the search comes back to where it began. Only then it is clear that the file is full and no empty address is available.
- With progressive overflow, a sequential search is performed beginning at the home address and if end of the address is reached, then wrap around is done and the search continuous from the beginning address in the file.
- The search is continued until the desired key or a blank record is found.
- If the file is full the search comes back to where it began. Only then it is clear that the record is not in the file.
- Progressive overflow is also referred to as *linear probing*.

Consider the table given below:

Key	Home address
Adams	20
Bates	21
Cole	21
Dean	22
Evans	20

Table 1.

19	
20	Adams
21	Bates
22	Cole
23	Dean
24	Evans
25	



Key	Home address	Actual Address	Search Length
Adams	20	20	1
Bates	21	21	1
Cole	21	22	2
Dean	22	23	2
Evans	20	24	5

Average search length is $\frac{1+1+2+2+5}{5} = 2.2$

Making Deletions

Now suppose record Dean is deleted which makes the address 23 empty. When the key Evans is searched it starts from home address of the i.e 20 and when it reaches address 23, which is empty the search stops and displays that key not present. But actually the key is present in the file. So

deletion of records leads to this type of problems. Using the concept of Tombstone technique this limitation can be overcome. The figure below depicts the above mentioned scenario.

19	
20	Adams
21	Bates
22	Cole
23	#####
24	Evans
25	

In tombstone technique whenever a record is deleted that address space is marked by a special character. So whenever a special character is encountered the search continues to next address without stopping at that point only. Further these marked addresses can be reused or reclaimed to store the new record.

Implications of Tombstone for Insertions.

Now suppose a new record with key Evans needs to be stored. When the key Evans is dropped to hash function it produces an address of 20. Since address 20 is already occupied by Adams according to progressive overflow technique the search for empty address slot starts and when it reaches address 23 the New Evans is inserted. But the problem is already there is a record for Evans i.e duplicate records are inserted which is certainly not acceptable. So to avoid we have to search the entire address space if already a key with same name is present or not. Only then the new record can be inserted. This in turn slows down the insertion operation.

Storing more than One Record per Address: Buckets

- **Bucket:** An area of a hash table with a single hash address which has room for more than one record.
- When using buckets, an entire bucket is read or written as a unit. (Records are not read individually).
- The use of buckets will reduce the average number of probes required to find a record.

Address	Counter	Records		
19	0			
20	2	Adams	Evans	
21	2	Bates	Cole	

22	1	Dean		
23	0			

- There should be a counter to keep track of how many records are already stored in any given address. The figure above represents storage of records with bucket size of 3.

Other Collision Resolution Techniques: Double Hashing

- Double hashing is similar to progressive overflow.
- The second hash value is used as a stepping distance for the probing.
- The second hash value should never be one. (Add one.)
- The second hash value should be relatively prime to the size of the table. (This happens if the table size is prime.)
- If there is collision at the hash address produced by $h_1(k)$ = Say 'X', then the key is dropped to second hash function to produce address 'C'.
- The new record is stored at the address 'X+C'.
- A collision resolution scheme which applies a second hash function to keys which collide, to determine a probing distance 'C'.
- The use of double hashing will reduce the average number of probes required to find a record.

Chained Progressive Overflow

- Chained progressive overflow forms a linked list, or chain, of synonyms.
- Each home address contains a number indicating the location of the next record with the same home address.
- The next record in turn contains a pointer to the other record with the same home address.
- This is shown in the figure below: In the figure below Adams contain the pointer to Cole which is synonym. Then Bates contain pointer to Dean which are again synonym. (Consider the below given Table)

Key	Home Address
Adams	20
Bates	21
Cole	20
Dean	21
Evans	24
Flint	20

The figure below represents the chained progressive overflow technique.

Home address	Actual Address	Records	Address of next Synonym	Search Length
	19	.	.	.
20	20	Adams	22	1
21	21	Bates	23	1
20	22	Cole	25	2
21	23	Dean	-1	2
24	24	Evans	-1	1
20	25	Flint	-1	3
	26		.	.
			.	.

- Now suppose if the Dean's home address was 22 instead of 21. By the time Dean is loaded, address 22 is already occupied by Cole, so Dean ends up at address 23. Does this mean that Cole's pointer should point to 23 (Dean's actual address) or to 25 (the address of Cole's synonym Flint)? If the pointer is 25, the linked list joining Adams, Cole, Flint is kept intact, but Dean is lost. If the pointer is 23 Flint is lost.

The problem here is that a certain address (22) that should be occupied by a home record (Dean is occupied by a different record. Solution to this type of problem are:

1. Two pass loading.
2. Chaining with a separate overflow area.

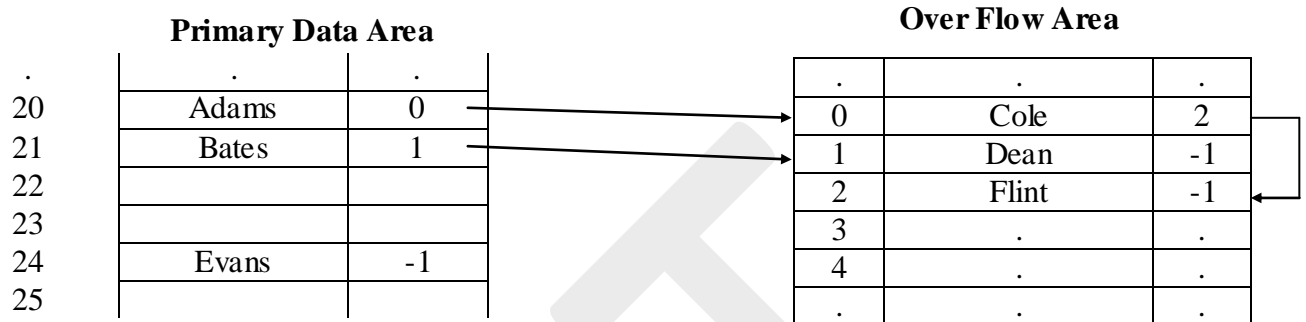
Two Pass Loading

- As the name implies, involves loading a hash file in two passes. On the first pass, only home records are loaded.
- All records that are not home records are kept in separate file.
- On the second pass, each overflow record is loaded and stored in one of the free addresses according to whatever collision resolution technique is being used.

Chaining with a separate overflow area.

- One way to keep overflow records from occupying home addresses where they should not be is to move them all to a separate overflow area.
- The set of home addresses is called prime data area, and the set of overflow addresses is called the overflow area.
- With respect to the previous example the records for Cole, Dean and Flint are stored in a separate overflow area as shown in the figure below.

- Whenever a new record is added if its home address is empty, it is stored in primary storage area. Otherwise it is moved to overflow area, where it is added to a linked list that starts at home address.



Scatter Tables

- If all records are moved into a separate "overflow" area, with only links being left in the hash table, the result is a *scatter table*.
- In scatter table records are not stored at the hash address.
- Only pointer to the record is stored at hash address.
- Scatter tables are similar to index but they are searched by hashing rather than some other method which is used in case of indexing.
- Main advantage of scatter tables is they support use of variable length records.

Key	Home Address
Adams	20
Bates	21
Cole	20
Dean	21
Evans	24
Flint	20

