SIR (1) 25

Hashing

The implementation of hashtable is called hashing. Hashing is a technique used to perform insertion, deletion and Search in Constant average time. Hash table is a datastructure used to store some data. It is basically a Searching technique.

The hashtable ~~data~~ structure is a Single dimensional array with Some fixed size which is specified by ~~H size~~ Tablesize. The index of the table starts from 0 to Tablesize-1.

Hashfunction is used to map the key into hashtable address. The identifier is directly Converted into hash address and in that hash address the item is placed.

Hash-function :- If the input Keys are integers, then we use a Simple hashfunction used mod function.

$H(x)$ = Key mod Tablesize. To distribute the items in the table in Uniform, then we have to select the size of the table as prime.

integer numbers -

(1) Division method :- $H(x)$ = Key mod table size

Suppose we have Some of number of employees, as each employee has a unique 4 digit employee number and the table size is 100 (0 TO 99). So the nearest prime - 97

H (3205) = 3205 mod 97 = 4   H (7148) = 7148 mod 97 = 67

H (2345) = 2345 mod 97 = 17.

(2) Midsquare method :- The key is Squared, then the hash function

H (Key) = l where l is obtained by deleting digits from both ends of Key^2.

Key :	3205	7148	2345
Key^2 :	10272025	51093904	5499025
	72	93	99

③ Folding method :- The Key is partitioned into a number of parts $k_1 \ldots k_r$ where each part, except the last, has the same number of digits as the required addr.. then the parts are added together, ignoring the last carry i.e.

$H(key) = k_1 + k_2 + \ldots + k_r$, where the carries are ignored.

$H(3205) = 32 + 05 = 37$

$H(7148) = 71 + 48 = 19$

$H(2345) = 23 + 45 = 68.$

Strings :- The given identifiers are strings (set of characters) But the table address is integer so we have to map the string idetifier into hashaddress. ie by ASCII values of characters in a string and add them. so it becomes an integer and then by using any integer hashfunction, the address is generated.

abc

$H(abc) =$ ~~Hash~~ ASCII [a] + ASCII [b] + ASCII [c] mod table size

A simple hashfunction is ~~take~~ as follows.

```
Algorithm Hash (~~Key~~ string type key, int ~~tablesize~~ keysize)
{
    ~~hash = 0~~; Hash-val = ord( Key [1])
    ~~While~~ for i := 2 TO key Size do
        hash-val = hash-val + ~~Key[i]~~; ord(Key[i])
    hash := hash-val mod tablesize
}
```

For this we will use four methods.

(1) Separate chaining (open hashing)

(2) Closed hashing (open addressing)

ⓐ Linear probing ⓑ Quadratic probing ⓒ double hashing

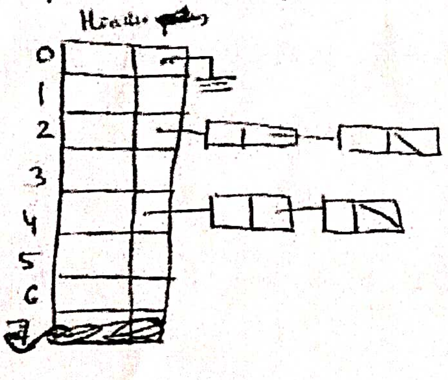
(3) Rehashing

(4) Extendible hashing.

(1) **Separate chaining (open hashing)** :- In this method, we maintain ~~head~~ list of header nodes which are useful if the two keys are mapped to same hashtable. In the header node, we will maintain the list of identifiers. We will u Simple linked list to store the identifiers. (Linkers or chaining). We can add any no nodes for the same hash address. ie array of headers. Initially all header are null. When a key is mapped by using a simple hashfunction

$$hash(x) = x \bmod tablesize.$$

Once the header node address is found then we will use ordinary traversing for appropriate position of the list. We can insert the node in the front or end of the list. We will prefer only at the front.



Take some data :- 20, 5, 21, ~~3~~5, 70, 15, 24

But if the hashtable size is too large 10,007 and suppose all strings eight or few characters and maximum ord value of key is 127.

Max eight character $127 * 8 = 1016 \rightarrow 0$ to 1016.

It is not equal distribution, only some part of the table is filled. So we have to for most hash function. i.e by considering only first three characters and some multiples.

$$ord(Key[1]) + 27 * ord(Key[2]) + 729 * ord(Key[3])$$

Algorithm hash (String type : Key , Integer Keysize)
{ hash := (ord (Key [1]) + 27 * ord (Key[2]) + 729 * ord(key[3]))
mod table size
}

The draw back of this is, if two strings will have first three characters are common.

So a good hash function is

Algorithm hash (String type. Key ; Integer : Keysize)
{ hashval := ord (Key[1])
for j := 2 To Keysize do
hashval := hashval * 32 + ord (Key[J]) mod table size.
Hash := hashval;
}

This may not be the best but it is good compare to other methods.

Collision :- If two or more identifiers are mapped to the same hash index or address, then collision will be occured. So we have to resolve this collision. Collision occurs, but have to resolve it.

only drawback is additional effort required to perform a search ~~in the~~ which is the time required to evaluate the hash function plus the time to traverse the list

In a closed hashing, a collision occurs, alternative cells are tried until empty cell is found. Cells $h_0(x), h_1(x), h_2(x) \ldots$ are tried until $h_i(x)$ = empty cell or an initial collision. For this we are using three techniques one is Linear probing, quadratic probing and double hashing.

Linear probing:- R is a new identifier and suppose in a table H already H(R) is filled. Then due to this new insertion, collision occurs. We have to resolve the collision by placing 'R' to some other location. Assume that the memory locations are sequential and circular. ie after last location, again we can goto first location. We search the table from initial collision point to some cell until an empty cell is found with condition that the table contains atleast some empty cells.

For example given keys are {89, 18, 49, 58, 69} and the table of size 10

	Empty table	After 89	After 18	After 49	After 58	After 69
0				49	49	49
1					58	58
2						69
3						
4						
5						
6						
7						
8			18	18	18	18
9		89	89	89	89	89

89 mod 10 = 9.

18 mod 10 = 8

49 mod 10 = 9 *

(9+1) mod 10 = 0 ✓

58 mod 10 = 8 *

(8+1) mod 10 = 9 *

(8+2) mod 10 = 0 *

(8+3) mod 10 = 1

69 mod 10 = 9 *

(9+1) mod 10 = 0 *

(9+2) mod 10 = 1 *

(9+3) mod 10 = 2

Quadratic probing :- The collision function is quadratic. The initial hash is same when collision occurs.

$$F(i) = i^2 \qquad i \rightarrow \text{collision number}$$

After collision, it must be placed into 1st position of the element 89

$$F(1) = 1^2 = 1$$

89, 18, 49, 58, 69

89 mod 10 = 9 ✓

18 mod 10 = 8 ✓

0	49
1	
2	58
3	69
4	
5	
6	
7	
8	18
9	89

49 mod 10 = 9 * (first collision)

$F(1) = 1^2 = 1$ place at 1 position after 89 element cell

ie 0th position

58 mod 10 = 8 * (first coll.)

$F(1) = 1^2 = 1$ but again the collision

$F(2) = 2^2 = 4$ place item in the 4th position of the initial collision point ie at 2

69 mod 10 = 9 × $F(1) = 1^2 = 1$ × $F(2) = 2^2 = 4$

Ex - 10, 25, 20, 40, 12, 62, 65 (Table size is 10)

0	10
1	20
2	12
3	62
4	40
5	25
6	65
7	
8	
9	

28

Double hashing:- For double hashing, we will use $F(i) = i \cdot hash_2(x)$.
This formula says that we apply a second hash function to x and probe at a distance of $hash_2(x)$, $2hash_2(x)$ ... and ~~so~~ so on. A $hash_2(x)$ function is $hash_2(x) = R - (x \bmod R)$ where R is a prime smaller than tablesize.

Ex:- Table size is 10 and nearest small prime is $R=7$

$\{89, 18, 49, 58, ~~69~~\ 10, 69\}$

	Empty	89	18	49	58	10	69
0							10
1							69
2							
3					58		58
4							
5							
6				49	49		49
7							
8		~~18~~	18	18	18		18
9		89	89	89	89		89

$hash(89) = 89 \bmod 10 = 9$

$hash(18) = 18 \bmod 10 = 8$

$hash(49) = 49 \bmod 10 = 9$ *

$hash_2(49) = 7 - (49 \bmod 7)$
$= 7 - 0 = 7$

$hash(58) = 58 \bmod 10 = 8$ *

$has_2(58) = 7 - (58 \bmod 7)$
$= 7 - 2 = 5$

$hash(10) = 10 \bmod 10 = 0$

$hash(69) = 69 \bmod 10 = 9$ *

$hash_2(69) - 7 - (69 \bmod 7)$
$= 1$ *

$2 hash_2(69) = 2 \times 1 = \underline{\underline{2}}$

3] Rehashing :- If the table gets too full, almost 70% is full, then we have to ~~use~~ rehashing. ie we have to create another new hashtable whose size is ~~more than~~ double of the previous one. Then scan the entire old table ~~to~~ with old hashfunction and then find the new address for that identifier using new hash function. ie Rehashing.

Ex:- Table size is 7 and keys are $\{13, 15, 24, 6\}$

$h(13) = 13 \bmod 7 = 6$

$h(15) = 15 \bmod 7 = 1$

$h(24) = 24 \bmod 7 = 3$

$h(6) = 6 \bmod 7 = \underline{6}$ * collision

(use linear probing method)

6	13
5	
4	
3	24
2	
1	15
0	6

Now the table is 70% full.

So we have to build another table whose size is more than double.

$h(x) = x \bmod 17$

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

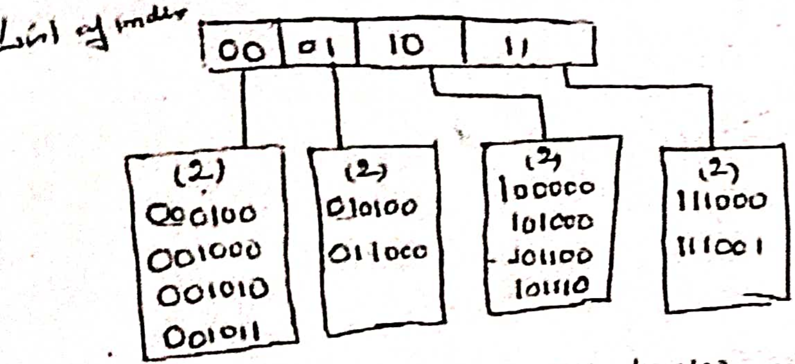
7

29

Extendible hash function :- (no of disks)

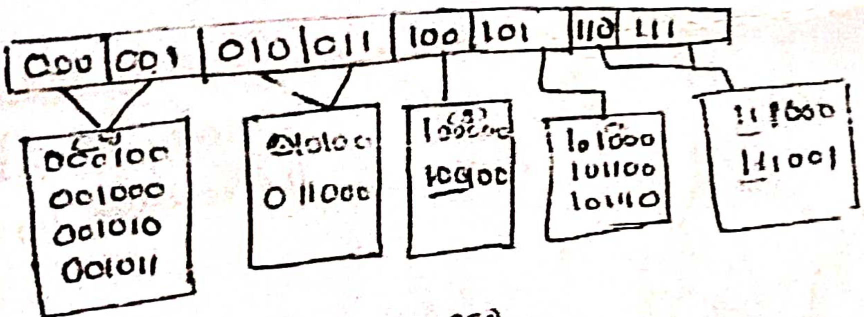
List of Indexes : Each index will indicate one memory block and each block will have fixed no of records.

Ex:- 4 indexes and each can accomodate 4 records (Binary Data)



Original data.

Now we want to insert 100100 and it must goto 3rd block. but already it is full. So we have to change the directory structure Now the length of the index is 3



TO insert 000000