Hashing

searching techniques where search time is hash dependent on the no. of elements.

For 0 to n-1 array size each and every elem are different key.

For storing record

key
↓
Generate array index
↓
store the record on that array i

For accessing record

key
↓
Generate array index
↓
Get the record from the array index

The generation of array index uses hash function, which converts the keys into array index and the array which supports hashing for storing reco or searching record called hash table.

2 important things are there

1. choosing a good hash function which ensures minimum collision.
2. Resolving the collision.

h(K) → a → hash table.
↓ hash fn. ↘ key

Choosing a hash function

2 criteria are there for a good hash fn.

1. It should be easy to compute
2. It should ~~be~~ generate unique address but it is an ideal situation so it should generate address with minimum collision.

1. Truncation Method —

Easiest method for computing key.

e.g. Let us take some 8 digit keys

8 2 3 9 4 5 6 1, 8 7 1 3 9 4 6 5, 8 3 5 6 7 2 7 1, 8 5 9 4 3 2 2 8

We can take the no. at rightmost digits or leftmost digits based on the size of hash table.

Suppose table size is 100, then take the 2 rightmost digits for getting the hash table address.

∴ The keys are 61, 65, 71, 28.

So, the no. of collision will be increased.

2. Mid square Method —

We square the key, after getting the no. we take some digits from the middle of that no. as an address.

1 3 3 7, 1 2 7 3, 1 3 9 1, 1 0 2 6

∴ The squares are

1 7 8 7 5 6 9 1 6 2 0 5 2 9 1 9 3 4 8 8 1 1 0 5 2 6 7 6

Suppose table size is 1000

∴ The hash addresses for the keys are

75, 05, 48, 26.

3. Folding Method

Break the keys into pieces, add them and get the hash addresses,

8 2 3 9 4 5 6 1, 8 7 1 3 9 4 6 5, 8 3 5 6 7 2 7 1, 8 5 9 4 3 2 2 8

$8\,2\,3\,9\,4\,5\,6\,1 = \overset{3}{8\,2\,3} + \overset{2}{9\,4} + \overset{3}{5\,6\,1} = 1478$

$8\,7\,1\,3\,9\,4\,6\,5 = 8\,7\,1 + 3\,9 + 4\,6\,5 = 1375$

$8\,3\,5\,6\,7\,2\,7\,1 = 8\,3\,5 + 6\,7 + 2\,7\,1 = 1173$

$8\,5\,9\,4\,3\,2\,2\,8 = 8\,5\,9 + 4\,3 + 2\,2\,8 = 1130$

```
 823
  94
+ 561
------
1478
```

Now truncate them up to the digit based on the size of hash table. suppose the table size is 1000. so the hash address can be from 0 to 999. So, we will truncate here the higher digit at number. Hash add. keys are as.

$H(8\,2\,3\,9\,4\,5\,6\,1) = 478$

$H = 375$

$= 173$

$= 130$

} keys addr.

4. Modular Method. (Best case table size is prime no. to reduce collision)

8 2 3 9 4 6 1,

Table size = 97

$8\,2\,3\,9\,4\,5\,6\,1 \;\%\; 97 = 45$

$8\,7\,1\,3\,9\,4\,6\,5 \;\%\; 97 = 0$

$8\,3\,5\,6\,7\,2\,7\,1 \;\%\; 97 = 25$

$8\,5\,9\,4\,3\,2\,2\,8 \;\%\; 97 = 64$

} keys address.

This method can be applicable with any other method.

5. Hash function for floating point nos.

1. Take the fractional part at key.
2. Multiply the fractional part with size of the hash tables
3. Take the integer part of the multiplication result as a hash add. at key.

table size = 97

123.4321, 19.463, 2.0298, 8.9956

$0.4321 \times 97 = 41.9137$
$0.463 \times 97 = 44.911$
$0.0298 \times 97 = 2.8906$
$0.9956 \times 97 = 96.5732$

H (123.4321) = 41
= 44
= 2
= 96

suppose keys are already in the range of 0 to 1. Then skip the step 1 and do the rest of the operation. Let us take some key.

$0.5932 \times 97 = 57$

Integer part / table size = Floating point value

Do the following operation for collision.

Hash fn. for strings

Table size = 97

suresh = s + u + r + e + s + h (Add ASCII value)
= 115 + 117 + 114 + 101 + 115 + 104
= 666

H (suresh) = 666 % 97 = 84

2nd method.

Suresh = s x 127 + u x 127 + r x 127 + e x 127 + s x 127 + h x 127

H (suresh) = 845821, 97

= 95

97) 84582 (871
776
698
579
192
97
95

ASCII character has maximum value 127

Collision Resolution (open Hashing)

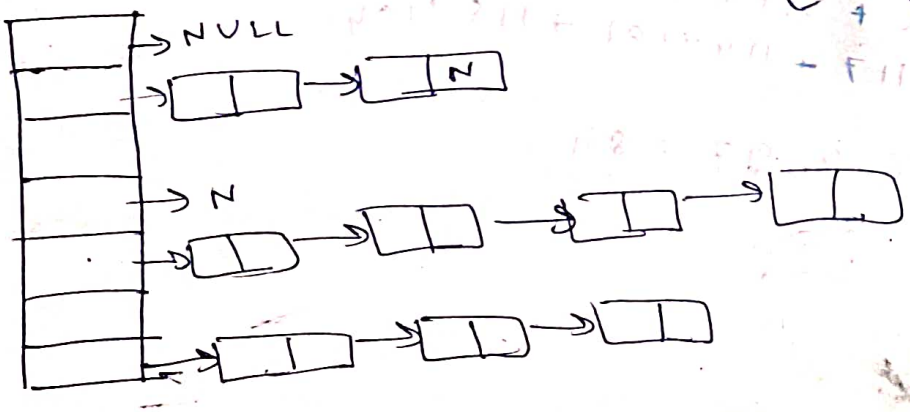
1. separate chaining

we can take the hash table as an array of pointers. Each pointer will point to one linked list and the elements which have same hash address will be maintained in the linked list. Basically it involves two operations.

1. Creation of good hash function for getting hash key value in the hash table
2. Maintain the elements in the linked list which is pointed by pointer available in hash table.

Here we can maintain the linked list in sorted order.

[advantage of memory spaces]



2. Closed Hashing (Open addressing)

i) Linear probing —

29, 18, 43, 10, 36, 25, 46, 49, Table size 11

0	10
1	
2	46
3	36
4	25
5	
6	
7	29
8	18
9	49
10	43

H (29) = 29 % 11 = 7
H (18) = 18 % 11 = 7
H (43) = 43 % 11 = 10
H (49) = 49 % 11 = 9

disadvantage of clustering.

Hash table as circular array. If any no. found the position is occupied then it goes to the next place.

ii) Quadratic Probing

29, 18, 43, 10, 46, 54

Linear probing search the location h, h+1, h+2 ... (%, size).

In Quadratic probing it search the location $h + i^2$ % size (for i = 1, 2 ...)

so, location h+1, h+4, h+9, ...

0	10
1	
2	46
3	54
4	
5	
6	40
7	29
8	18
9	
A	43

H(29) = 29 % 11 = 7 → h h = 7
H (18) = 18 % 11 = 7
H (43) = 43 % 11 = 10
H (10) = 10 % 11 = 10
H (46) = 46 % 11 = 2
H (54) = 54 % 11 = 10
H (40) = 40 % 11 = 7
7 + 1 = 8
7 + 4 = 11

10 + 4
14 % 11 = 3
40 + 1 = 0

3. Double Hashing

h is a hash key, the in case of collision we will again do the hashing at this hash key.

Hash (h) = h.

search h, h+h', h+2h', h+3h' . . . .

It requires 2 times calculation of hashing.

h = key % 13

h' = 11 − (key % 11) ⇒

so the time at collision hash address for the next probe will be as (h + h') mod 13

= ((key % 13) + (11 − (key % 11))) mod 13

8, 55, 48, 68

8 % 13 = 8

55 % 13 = 3

68 % 13 = 3

For 68 new address will be

((68 % 13) + (11 − (68 % 11))) % 13

= (3 + (11 − 2)) % 13

= (3 + 9) % 13

= 12 % 13

= 12

0	
1	
2	
3	55
4	
5	
6	
7	
8	8
9	48
10	
11	
12	68