→ Efficient in terms of storage point of view
if v = Number of bit arrays (column)
r = Number of records. (row)

→ Total bits needed = v * r

* From computation point of view, this technique is efficient because no searching is involved.
* A record can be computed through logical operations like AND, OR, NOT & hence giving fast computations.
* One drawback is that it is not possible to store all kinds of information.
Eg: fields where all or nearly all the values are different, like name, regno, address, this technique is inefficient.

2/12/2020
Abstract datatype

HASHING

Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects.
Eg: In universities, each student is assigned

Page No :
Date :

a unique roll number that can be used to retrieve information about them.
- In libraries, each book is assigned a unique no. that can be used to determine info about the book, such as its exact position in the library or the users it has been issued to etc.

In both these examples the students & books were hashed to a unique number.

Assume that you have an object & you want to assign a key to it to make searching easy. To store the key/value pair, you can use a simple array like a D.S where keys can be used directly as an index to store values. However, in case where the keys are large & cannot be used directly as an index, we use hashing.

- In hashing, large keys are converted into small keys using hash functions. The values are then stored in a D.S called hash table.
- The idea of hashing is to distribute entries (key/value pairs) uniformly across an array.
- Each element is assigned a key (converted key)

By using that key you can access the element in O(1) time. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.

Hashing is a important D.S. which is designed to use a special function called hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends on the efficiency of the hash function used.

Hashing is implemented in two ~~ways~~ steps:

1) An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.

2) The element is stored in the hash table ~~where~~ where it can be quickly retrieved using hashed key.

```
hash = hashfunc(key)
index = hash % array_size
```

In this method, the hash is independent of the array size & it is then reduced to an index by using modulo Operator.

Hash Function :

* A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table.
* The value returned by a hash function is called hash values (hash codes / hash sums or simply hashes)

To achieve a good hashing Mechanism, it is important to have a good hash function with the following basic requirements:

1) Easy to compute : It should be easy to compute & must not become an algorithm in itself.

2) Uniform distribution : It should provide a uniform distribution across the hash table & should not result in clustering.

3) Less Collisions : Collision occur when pairs of elements are mapped to the same hash values These should be avoided

NOTE - Irrespective of how good a hash function is, Collisions are bound to occur. Therefore, to maintain the performance of a hash table, it is important to manage collisions through various collision resolution techniques.

Need for a good hash function

Let us understand the need for a good hash func. Assume that you have to ~~strong~~ store strings in a hash table by using the hashing techniques.

To compute the index ~~of~~ for storing the strings, use a hash function that states the following

"The index for a specific string will be equal to the sum of the ASCII values of the characters modulo 599"

As 599 is a prime no., it will reduce the possibility of indexing different strings. It is

Date :

recommended that you use prime no.s in case of modulo. The ASCII values of a, b, c, d, e & f are 97, 98, 99, 100, 101 & 102. Since all the strings contain the same characters with different permutations the sum will be 599.

The hash func. will compute the same index for all the strings & the strings will be stored in the hash table in the following format. As the index of all strings is the same, you can create a list on that index & insert all the strings in that list.

The two keys may hash to the same slot. We call this situation collision. Fortunately, we have effective techniques for resolving the conflict created by collisions.

HASHING METHODS

- Search time in array is O(n)
- Search time in tree is O(log n)
- Ideally, search time in array-based hash table is constant O(1) in the average case.

• Array-based, linked list based or tree-based hash table can be used.

application of hash tables

- Hashing is a technique often used to implement an ADT dictionary.
- Compilers use hash tables to keep track of declared variables in source code. The D.S is called symbol table.
- Spelling checker: entire dictionary can be prehashed & words can be checked in constant time.
- In this lecture, we examine a D.S (i.e, hash table) which is designed specifically with the objective of providing efficient insertion & searching operations.
- To achieve constant time performance objective, the implementation must be based in some way on an array rather than a linked list. (No specific ordering of items)
- This is because we can access the k-th element of an array in constant time, whereas the same operation in a linked

list takes longer time.

We are designing a container which will be used to hold some number of items of a given set K. We call the elements of the set K keys & K is called key space.

The general approach is to store the keys in an array. The position (location or index) i of a key k in the array is given by a function f(k), called a hash function, which determines the position of a given key directly from that key.
(i.e, i = f(k) is called the hash code of k)

In the general case, we expect the size of the set of keys, denoted as |K|, to be relatively large in comparison with the number of item stored in the container M (M << |K|)

or, the no. of items stored in the container is significantly less than |K|. We use an array of size M to contain items.

Consequently, we need is a function

$$f : K \longrightarrow \{0, 1 \ldots m-1\}$$

- This function maps (transforms) the set of key values to be stored in the container to subscripts (index) in an array of length m
- This function is called **hash function**.
- In general, since $|K| >> m$, the mapping defined by a hash function will be a many to one mapping.
- i.e, there will exist many pairs of distinct keys x & y, such that $x \neq y$, for which $f(x) = f(y)$
- This situation is called a collision & f is not a perfect hash function.

Characteristics of good hash function:

1) It avoids collision
2) It tends to spread keys evenly in the array.
3) It is easy to compute (i.e, computational time of a hash function should be $O(1)$)

Page No :
Date :

020

COLLISION RESOLUTION METHODS

OPEN ADDRESSING

1. Linear Probing Method
2. Quadratic Probing Method
3. Double Hashing Method
4. Coalesced chaining Method
5. Separate / direct chaining Method
6. Cuckoo Hashing Method.

* Three methods in **Open Addressing** are linear probing, Quadratic probing & double hashing.

↓

division hashing method because the hash function is $f(k) = K \% M$

* Some other hashing methods are middle-square hashing method, multiplication hashing method, & fibonacci hashing method.

(1) LINEAR PROBING METHOD

- hash table implemented using array

containing M nodes, each element (node) of the hash table has a field k used to contain the key of the node.

- M can be any +ve integer but M is often chosen to be a prime number.
- When the hash table is initialised, all fields k are assigned to -1 (i.e, empty or vacant)
- When a node with the key k needs to be added into the hash table, the hash function

$$f(k) = k \% M$$

will specify the address $i = f(k)$ within the range $[0, M-1]$.

- If there is no conflict (i.e, the cell is unoccupied), then this node is added ~~to~~ into the hash table at the address i.
- If a conflict takes place, then the hash function rehashes first time f_1 to consider the next address (i.e, i+1). If conflict occurs again, then the hash func. rehashes second time f_2 to examine the next address (i+2). This process repeats until the available address found then this node will be added at this address.
- The rehash function at the time t (i.e, the

collision num $t = 1, 2 \ldots$ is as

$$f_t(k) = (f(k) + t) \% m = (i + t) \% m$$

- when searching a node, the hash function $f(k)$ will identify the address i ($i = f(k)$) falling b/w 0 & $m-1$.

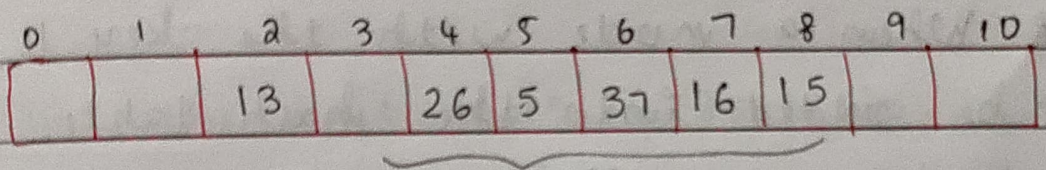
Eg: insert keys 32, 53, 22, 92, 17, 34, 24, 37 & 56 into a hash table of size $m = 10$.

(a) (b) (c) (d) (e)

0	-1	0	-1	0	-1	0	-1	0	56
1	-1	1	-1	1	-1	1	-1	1	-1
2	32	2	32	2	32	2	32	2	32
3	53	3	53	3	53	3	53	3	53
4	-1	4	22	4	22	4	22	4	22
5	-1	5	92	5	92	5	92	5	92
6	-1	6	-1	6	34	6	34	6	34
7	-1	7	-1	7	17	7	17	7	17
8	-1	8	-1	8	-1	8	24	8	24
9	-1	9	-1	9	-1	9	37	9	37

Page No :
Date :

Eg: Insertion the keys 13, 26, 5, 37, 16 & 15 into a hash table with integer keys using hash function $f(k) = k \bmod 11$. The value of $f(k)$ is called the hash code of the key k.



Primary clustering

Drawbacks:

→ It causes the primary clustering problem
→ It creates a long sequence of filled slots.
→ Primary clustering occurs when the keys hashed to different locations trace the same sequence in looking for an empty location.

2. QUADRATIC PROBING METHOD

* It is an attempt to keep clusters from forming.
* The idea is to probe more widely separated cells, instead of those adjacent to the primary hash site.
* The hash table in this case is implemented

using an array containing M nodes, each node of the hash table has a field k used to contain the key of the node.

* when the hash table is initialized, all fields k are assigned to -1.

* When a node with the key k needs to be added into the hash table, the hash function, $f(k) = k \% M$ will specify the address i within the range $[0, M-1)$.

* If there is no conflict, then this node is added into the hash table at the address i.

* If a conflict take place, then the hash function rehashes first time f_1 to consider the address $f(k) + 1^2$. If conflict occur again, then the hash function rehashes the second time f_2 to examine the address $f(k) + 2^2$. This process repeats until the available address found then this node will be added at this address.

* The rehash function at the function time t (collision number $t = 1, 2, \ldots$) is presented as follows:

Page No :
Date :

$$f_t(k) = (f(k) + t^2) \% m = (i + t^2) \% m$$

* whem searching a mode, the hash function f(k) will identify the address i (i.e, i = f(k)) falling between 0 & m-1.

Eg: insert the keys 10, 15, 16, 20, 30, 25, 26 & 36 into a hash table of size m=10

(a)

0	10
1	-1
2	-1
3	-1
4	-1
5	15
6	16
7	-1
8	-1
9	-1

(b)

0	10
1	20
2	-1
3	-1
4	-1
5	15
6	16
7	-1
8	-1
9	-1

(c)

0	10
1	20
2	-1
3	-1
4	30
5	15
6	16
7	26
8	-1
9	25

(d)

0	10
1	20
2	-1
3	-1
4	30
5	15
6	16
7	26
8	-1
9	25

(e)

0	10
1	20
2	36
3	-1
4	30
5	15
6	16
7	26
8	-1
9	25

* The problem with Quadratic probing Method is that it causes Secondary clustering problem.
* Secondary clustering occurs when keys which hash to the same location trace the

Page No :
Date :

same sequence in looking for an empty location.

3) DOUBLE HASHING METHOD

> To eliminate secondary clustering, we can use another approach: double hashing

> Double Hashing is one of the effective methods of probing an Open addressed hash table.

> It works by adding the hash codes of two hash functions. This method uses two functions f & g as the hash functions.

> When a node with the key k needs to be added into the hash table, the first hash function

$$f(k) = K \% M = i$$

will specify the address i within a range [0, M-1], where M is a prime No.

> A prime Num M produces fewer collisions.

> If there is no conflict, this node is

Date :

added into the hash table at the address i.

> If a conflict takes place, the second hash func. $g(k) = c - (k \% c) = j$, where the constant c is a prime no. less than hash table size m & j is called step size is used.

> Then the first hash func. rehashes the first time to consider the address $f_1(k) = (f(k) + j) \% m = (i + j) \% m = i_1$

If conflict happens again, the first hash func. rehashes the second time to consider the address $f_2(k) = (f_1(k) + j) \% m = i_2$. This process repeats until the address found then this node will be added at this address.

> The rehash func. at the time t (i.e, the collision number t 1, 2 .. ?) is

$$f_t(k) = (f_{t-1}(k) + j) \% m = (i_{t-1} + j) \% m$$

where

$f_0(k) = i_0 = i = f(k)$, $j = g(k) = c - (k \% c)$, c & m are primes, $c < m$

> Double Hashing requires that the size of the

table is a prime number (eg: m=11)

Eg: insert the keys 14, 17, 25, 37, 34, 16 & 26 into hash table of size m = 11, c = 5 (c should be less than m)

0	-1
1	-1
2	-1
3	14
4	-1
5	-1
6	17
7	-1
8	-1
9	-1
10	-1

0	-1
1	34
2	-1
3	14
4	37
5	16
6	17
7	-1
8	25
9	-1
10	-1

0	-1
1	34
2	-1
3	14
4	37
5	16
6	17
7	-1
8	25
9	26
10	-1

ize = prime number

$c = c - (I \% c) = 5 - (25 \% 5) = 5$ //

$i_1 = (i_0 + j) \% 11$

$= (i + j) \% 11$

$= (3 + 5) \% 11 = 8$ //

4) COALESCED CHAINING METHOD

- The hash table in this case implemented using an array containing m nodes.
- Each node of the hash table is a class consisting of two fields as follows.

Field k : contains the key of the node.

Field next : contains a reference to next node, if conflict occurs.

- When the hash table is initialized, all fields k and next are assigned to -1.

- When a node with the key k needs to be added into the hash table, the hash function f(k) = k % m = i will identify the address i within the range [0, m-1]

- If there is no conflict, then this node will be added into the hash table at address i.
- If a conflict happens, then this node will be added into the hash table at the first available address, say j, from the bottom of the hashing. The field next of the last node at index i in the chain will be updated to j

Eg: Insert the keys 10, 15, 26, 30, 25 & 35 into a hash table of size 10.

(a) (b)

0	10	-1		0	10	9
1	-1	-1		1	-1	(9)
2	-1	-1		2	-1	-1
3	-1	-1		3	-1	-1
4	-1	-1		4	-1	-1
5	15	-1		5	15	~~→~~ 8
6	26	-1		6	26	-1
7	-1	-1		7	~~→~~ 35	-1
8	-1	-1		8	~~→~~ 25	~~→~~ 7
9	-1	-1		9	30	-1

5) SEPARATE (DIRECT) CHAINING METHOD

→ The hash table is implemented by using singly linked list.

→ Nodes on the hash table are hashed into m (m = 10) singly linked list (from list 0 to list m-1)

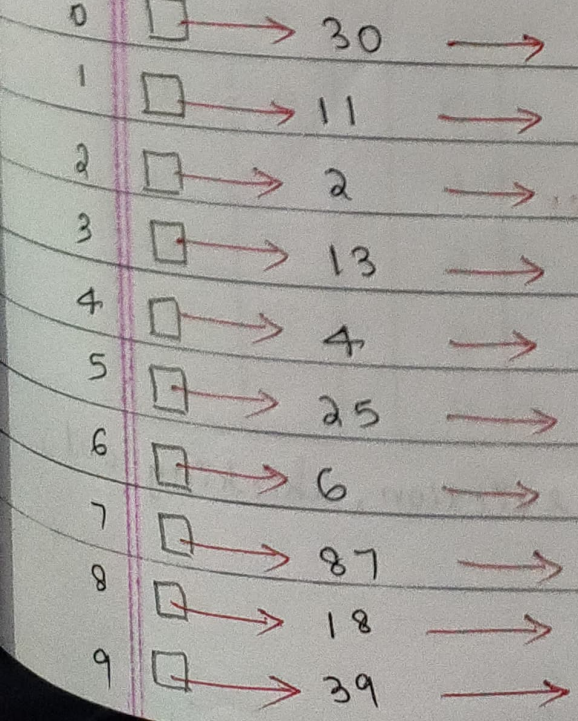
→ Nodes conflicted at the address i are directly connected in the list i, where $0 \leq i \leq M-1$

→ When a node with the key k is added into the hash table, the hash func. $f(k) = k \% M$ will identify the address i b/w 0 & M-1 corresponding the singly linked list i where this node will be added.

→ When a node needs to be searched on the hash table, the hash function $f(k) = k \% M$ will specify the address i within the range [0, M-1] corresponding the singly linked list i that may contain the node.

→ Searching a node on the hash table turns out the problem of searching an element on the singly linked list.

Eg: insert 13, 4, 16, 87, 11, 30, 2, 18, 25.



Page No:
Date:

6) CUCKOO HASHING METHOD

a) In the Cuckoo hashing ~~Metho~~ Scheme, we use two hash tables T_0 & T_1, each of M size

b) In addition, we use a hash function h_0 for T_0 & a different hash func - h_1 for T_1

c) For any given key $k \in K$, the are 2 possible places where we can store the key k, namely, either in $T_0[h_0(k)]$ or $T_1[h_1(k)]$

Eg: the key $K = A(0,2)$ will be stored at index 0 in T_0 & ~~th~~ at index 2 in T_1 (i.e, $h_0(k)=0$, $h_1(k)=2$)

```
procedure insert(k)
if (K = T0[h0(k)]) or (K = T1[h1(k)]) then return
  i ← 0

loop t times
if Ti[hi(k)] is empty then
  Ti[hi(k)] ← k
return
temp ← Ti[hi(k)]
Ti[hi(k)] ← k // cuckoo eviction, kicking out, displace
```

Date :

K ← temp

i ← (i+1) mod 2 // i = 0, 1

Rehash all the items including k using new hash func. ho & h1.

function search(K)

return (K = T0[h0(K)]) or (K = T1[h1(K)]

End;

Eg: Insert the keys A(0,2), B(0,0), C(1,4), D(1,0) E(3,2) F(3,4) into T0 & T1

a) A : 0, 2

Table 0	
0	A
1	
2	
3	
4	

Table 1	
0	
1	
2	
3	
4	

b) A : 0, 2 B : 0, 0

Table 0	
0	B
1	
2	
3	
4	

Table	
0	
1	
2	A
3	
4	

A: 0, 2 B: 0, 0 C: 1, 4

A: 0, 2 B: 0, 0 C: 1, 4 D: 1, 0

E: 3, 2

Table 0		Table 1			Table 0		Table 1		
0	B	0			0	B	0		
1	C	1			1	D	1		
2		2	A		2		2	A	
3		3			3	E	3		
4		4			4		4	C	

- ~~We cannot do~~
- A family H of hash func is 2-universal if for any two distinct keys x & y in K, where $x \neq y$ & for a hash func. h chosen uniformly at random from H, we have

$$\text{Probability}(h(x) = h(y)) \leq 1/M$$

AMORTISED ANALYSIS

Asymptotic notation

$O(n)$ → worst Case

$\Omega(n)$ → Best Case

$\Theta(n)$ → Average

↳ No change in space