##### Hashing

* + The main goal of Hashing/Hashed search is to find the data with only one test.
  + Hashing is a process of computing the address where an element is to be inserted or found in a hash table by applying a hash function to a given key.
  + Hashing is a Key -to-index transformation in which the keys map to index in an array called as Hash Table.
  + This method used a Hash function (Hashing algorithm) to map keys into

positions in a table called the Hash Table.

**K1**

**H(K1)**

**H(K2)**

**H(K3)**

**K2**

**K3**

**Keys**

##### Hash Table:

**0**

**1**

|  |
| --- |
|  |
| **Value3** |
|  |
| **Value1** |
|  |
| **Value2** |
|  |
|  |
|  |

**2**

**3**

**.**

**.**

**.**

**.**

**m-1**

#### Hash Table

* *Definition:* A Hash table is a data structure that stores elements and allows insertions, lookups, and deletions to be performed in O(1) time.
* A hash table consists of an array in which data is accessed via a special index called a Key.
* Insertion of data in the hash table is based on the key value. Hence every entry in the hash table is associated with some key.
* For example, for storing an employee record in the hash table the employee ID will work as a key.

**0**

**1**

**2**

**3**

**.**

**.**

**.**

**m-1**

The positions in the hash table are indexed 0 through m -1

**Hash table of Size m**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

o Load factor (α) = (no. of elements) / (no. of table slots)

##### Hash Function:

* + Hash functions are primarily used in hash tables, to quickly locate a data record given its search key
  + *Definition:* Hash function is used to map the search key to an index; the index gives the place in the hash table where the corresponding record should be stored.
  + A Key is mapped into one of the m slots of the Hash table using

Hash function.

**Hash Function**

#### Key

* + *Example Hash Function:*

#### H(K)= K%m

**Index/ position**

**0**

**1**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**2**

**3**

**.**

**.**

**.**

**m-1**

#### Hash Table

Here K is the Key, H is the Hash Function & m is the size of the Hash Table

* + A Hash function may map several keys into the same position (index) of the hash table.
  + E ach position of the hash table is often called as a **‘Bucket’.** Bucket consists of number of slots for holding elements.
  + Position h(K) is the **‘Home Bucket ’** for the element whose key is K.

##### Example:

Consider inserting the keys 80, 40, 65, 24, 44, 58 into a hash table of size m=11 using hash function h(k)=k mod m

|  |
| --- |
| **44** |
|  |
| **24** |
| **80 \*** |
|  |
|  |
|  |
| **40** |
|  |
|  |
| **65** |

1. K=80

h(80) = 80 mod 11 = 3

2. K=40

h(40) = 40 mod 11 = 7

3. K=65

h(65) = 65 mod 11 = 10

4. K=24

h(24) = 24 mod 11 = 2

5. K=44

h(44) = 44 mod 11 = 0

6. K= 58

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7 Hash Table**

**8**

**9**

**10**

h(58) = 58 mod 11 = 3 (already occupied by 80 -> Collision has occurred)

* A **Collision** occurs whenever two or more different keys have the same Home Bucket. (in our example keys 80 & 58 have same Home bucket 3)
* *“ The situation in which the hash function returns the same index for more than one key is called as collision ”* .
* An **Overflo w**occurs when there is no room in the home bucket for the new

element.

##### Applications of Hash Tables:

* + Used to implement Associative Arrays
  + Database Management Systems – Telephone book database, Library books Catalogue, Employee details
  + Compiler - Symbol Table
  + Operating System - Page Mapping Tables, Cache Memory
* ***Analysis of Hash tables:*** Time complexity

|  |  |  |
| --- | --- | --- |
| Operation | Average case | Worst case |
| Search | O(1) | O(n) |
| Insert | O(1) | O(n) |
| Delete | O(1) | O(n) |

## HASH FUNCTIONS/ HASHING METHODS :

* + Hash function is used to map the search key to an index; the index gives the place in the hash table where the corresponding record should be stored.
  + A Key is mapped into one of the m slots of the Hash table using Hash function.
  + Some Commonly used Hash functions are:

1. Division Method or Modulo Division
2. Multiplication Method

##### Division Method:

* + This method divides the key by the hash table size and uses the remainder as the index of the hash table.
  + In this method, we map a key K into one of the m slots of the hash table by taking remainder of K divided by m.
  + The hash function is

**H(K)= K % m**

Here K is the Key, H is the Hash Function & m is the size of the Hash Table

* + *Example:*

Consider inserting keys 26,33,76,86 into a hash table of size m=11 using

|  |
| --- |
| **33** |
|  |
|  |
|  |
| **26** |
|  |
|  |
|  |
|  |
| **86** |
| **76** |

division method **0**

h(26) = 26 % 11 = 4 **1**

**2**

**3**

h(33) = 33 % 11 = 0 **4**

**5**

**6**

h(76) = 76 % 11 = 10 **7**

**8**

**9**

h(86) = 86 % 11 = 9 **10**

* + Efficiency of this method depends on the value of ‘m’

**Hash Table**

* + When using a division method, we usually avoid certain values of m.

1. m should not be a power of 2
2. A prime number not too close to an exact power of 2 is often a good choice for m. Because it results in fewer collisions than other values for m.

##### Multiplication Met hod:

* + The multiplication method for generating hash functions operates in two steps:

1. Multiply the key **k** by a constant **A** in the range 0 < A < 1 and extract the fractional part of **kA**
2. Multiply this fractional part by **m** and take the floor.
   * The hash function is

#### h(k) = m(kA mod 1)

Where **k A mod 1 =** e the fractional part of KA

**A = 0.61804** (Golden value suggested by Knuth)

* + *Example:*

Consider m=32 and key k=100 (i). kA = 100 \* 0.61804

= 61.804

Taking the fractional part of the result kA mod 1 = 61.804 mod 1

= 0.804

(ii). m\* (kA mod 1) = 32 \* 0.804

= 25.728

By taking the floor of the result the index we got is 25 h(32) = 25

* + The advantage of this method is that the value of m is not critical. Typically it is chosen to be a power of 2.

## COLLISION RESOLUTION TECHNIQUES :

* + ***Definition:*** The situation in which a hash function maps two or more keys to same index (Home bucket) in hash table is called as Collision.
  + When two keys hash to the same position in a hash table they collide.
  + If x1 and x2 are two different keys, it is possible that **h(x1) = h(x2)** , then collision is said to occur between keys x1 and x2.
  + Whatever may be the hash function used in hashing, co mplete removal of collisions is almost impossible. So collisions in hashing cannot be ignored.
  + Two techniques used to resolve collisions are

1. Open hashing or Separate chaining
2. Closed hashing or Open addressing

**Open Hashing**

**Collision Resolution Techniques**

**Quadratic Probing**

**Closed Hashing**

**Double Hashing**

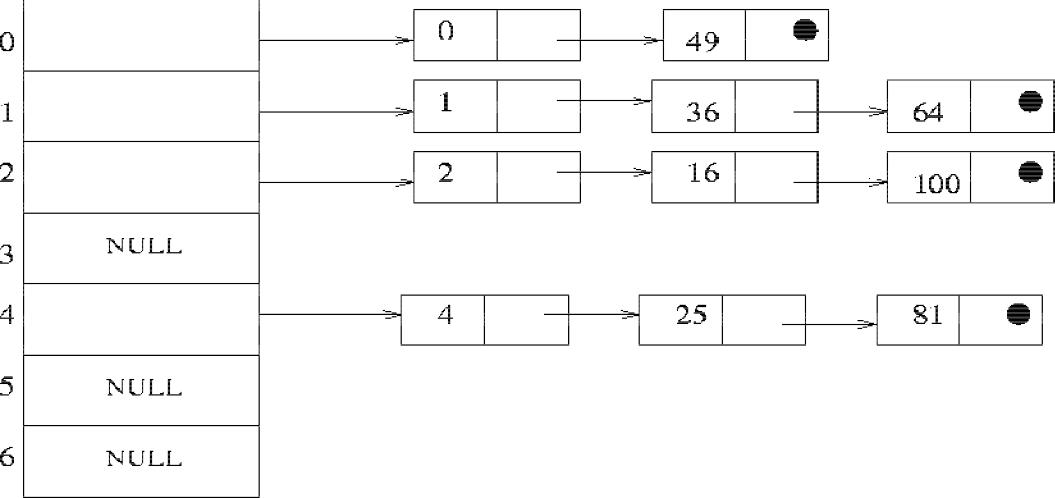
**Linear Probing**

#### OPEN HASHING:

* + It is also known as Separate chaining
  + In this technique, hash table is implemented as an array of Linked lists.
  + In this technique *, “ hash table is an array of pointers and each pointer will point to one linked list ” .*
  + Whenever a collision occurs then a linked list (chain) is maintained at that home bucket.
  + All the elements that hash to the same value are placed in the same home bucket in a Linked list.
  + Hash table that used open hashing is called as Open Hash table(or chained hash table).

#### Example:

Consider the keys 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100. Let the hash function be:h(x) = x % 7



##### Advantages:

1. Simple and effective approach to collision resolution.
2. Hash table size need not be a prime number

##### Disadvantages:

1. Additional Data structure (linked list) needs to be used to accommodate collision data.
2. Wastage of memory space for storing pointers (linked lists).
3. More number of collisions leads to un evenly distributed keys resulting in
   1. Long length lists
   2. Increased search time
   3. Many empty spaces in the hash table

#### CLOSED HASHING:

* + It is also known as Open Addressing.
  + In Open Addressing, all the elements are stored in the hash table itself. That is, each table entry contains either an element or NULL.
  + It avoids pointers
  + Instead of following pointers as in case of Open hashing, we compute the sequence of slots to be examined.
  + The process of examining the locations in the hash table is called ***Probing*** .
  + To perform insertion using open addressing, we successively examine, or probe, the hash table until we find an empty slot in which to put the key.
  + Collisions are handled by generating a sequence of rehash values

h : U **×** {0,1,2,……. } **-->**{0,1,2,…….,m-1}

Universe of Keys

Probe number Hash Table

* + Given a key x, it has a hash value h(x,0) and a set of rehash values h(x,1), h(x,2), h(x,3),………,h(x,m-1)
  + Closed Hashing Techniques:

1. Linear Probing
2. Quadratic Probing
3. Double Hashing

##### Linear Probing :

* + Linear probing is a scheme for resolving collisions of values by *sequentially searching the hash table for a free location* .
  + Suppose the hash table size is m, and key value is mapped to index i, with a hash function . If collision has occurred at i th index. then follow the following sequence of locations in hash table and do sequential / linear search.

i+1, i+2,……..,m,0,1,2, i

* + The search will continue until any one of the following cases occur

1. The key value is already present in the table
2. The unoccupied locatio n(empty slot) is encountered.
3. It reaches to the location where search was started.
   * Here hash table is considered as circular, so that when the last location is reached, the search proceeds to the starting location of the table.
   * Hash function in linear pr obing is defined as

**h1 (k,i) = ( h(k) + i ) mod m** fori=0,1,2,3,….,m-1 and h(k) is the primary hash function

* + **Example :**Insert the following keys 15,11,25,16,9,8,12,8,23,47into hash table of size m=11 using linear probing, consider the primary hash function as h(k)=k mod m

0

1

2

3

4

5

6

7

8

9

10

**Initially**

**hashtable**

0

1

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |
| 15 |
|  |
|  |
|  |
|  |
|  |
|  |

2

3

4

5

6

7

8

9

10

**Inserting 15**

0

1

|  |
| --- |
| 11 |
|  |
|  |
|  |
| 15 |
|  |
|  |
|  |
|  |
|  |
|  |

2

3

4

5

6

7

8

9

10

**Inserting 11**

0

1

|  |
| --- |
| 11 |
|  |
|  |
| 25 |
| 15 |
|  |
|  |
|  |
|  |
|  |
|  |

2

3

4

5

6

7

8

9

10

**Inserting 25**

**is empty**

0

|  |
| --- |
| 11 |
|  |
|  |
| 25 |
| 15 |
| 16 |
|  |
|  |
|  |
|  |
|  |

1

2

3

4

5

6

7

8

9

10

**Inserting 16**

0

|  |
| --- |
| 11 |
| 12 |
|  |
| 25 |
| 15 |
| 16 |
|  |
|  |
| 8 |
| 9 |
|  |

1

2

3

4

5

6

7

8

9

10

0

1

|  |
| --- |
| 11 |
|  |
|  |
| 25 |
| 15 |
| 16 |
|  |
|  |
|  |
| 9 |
|  |

2

3

4

5

6

7

8

9

10

**Inserting 9**

0

1

|  |
| --- |
| 11 |
|  |
|  |
| 25 |
| 15 |
| 16 |
|  |
|  |
| 8 |
| 9 |
|  |

2

3

4

5

6

7

8

9

10

**Inserting 8**

0

|  |
| --- |
| 11 |
| 12 |
| 23 |
| 25 |
| 15 |
| 16 |
|  |
|  |
| 8 |
| 9 |
|  |

1

2

3

4

5

6

7

8

9

10

0

1

|  |
| --- |
| 11 |
| 12 |
|  |
| 25 |
| 15 |
| 16 |
|  |
|  |
| 8 |
| 9 |
|  |

2

3

4

5

6

7

8

9

10

**Inserting 12**

0

|  |
| --- |
| 11 |
| 12 |
| 23 |
| 25 |
| 15 |
| 16 |
| 47 |
|  |
| 8 |
| 9 |
|  |

1

2

3

4

5

6

7

8

9

10

#### Inserting 8

Here already key 8 is presented in hash table. So does not insert once again.

#### Inserting 23

Here collision occurs at 1st index. So next index is 2

#### Inserting 47

Here collision occurs at 3, 4, 5 indexes. So next

index is 6

#### Drawbacks of linear probing :

* 1. The major drawback is that, as half of the hash table is filled, there is a tendency towards clustering. The key values are clustered in large groups and as a result sequential search becomes slower. This kind of clustering known as primary clustering.
  2. Performance degrades as the hash table gets nearly full.

##### Quadratic Probing :

* + Quadratic probing is a collision resolution method that eliminates the primary clustering problem in linear probing.
  + If there is a collision at index i

In linear probing , the next indexes to be probe are i+1, i+2, i+3, -------------

But in quadratic probe, the next indexes to be probe are i+1 2, i+22, i+32,------

* + If m is the size of the hash table and h(k) is the hash function , then quadratic probing search the indexes as

**h1 (k,i) = (h(k)+ i2 ) mod m**

for i=0,1,2,3,----------

* + ***Example:*** Insert the following keys 89, 18, 49, 58, 69 into hash table of size m=10 using quadratic probing , consider the Hash function h(k) = k % m Hash function used in quadratic probing is **h1 (k,i) = (h(k)+ i 2 ) mod m**

**1.** h1(89,0)= (h(89) + 0)% 10

h(89) = 89% 10 = 9

h1(89,0) = 9 (empty slot)

**2.** h1(18,0) = (h(18) + 0)% 10

h(18) = 18% 10 = 8

h1(18,0) = 8 (empty slot)

**3. a)**h1(49,0) = (h(49) + 0)% 10

h(49) = 49% 10 =9

h1(49,0) = 9 (Collision)

**b)**h1(49,1) = (h(49) + 12)% 10

= (9+1) % 10

h1(49,1) = 0 (empty slot)

**4. a)**h1(58,0) = (h(58) + 0)% 10

h(58) = 58% 10 = 8

h1(58,0) = 8 (Collision)

**b)** h1(58,1) = (h(58) + 12)% 10

=(8+1) % 10

h1(58,1) = 9 (Collision)

**c)**h1(58,2) = (h(58) + 22)% 10

= (8+4) % 10

= 2 (empty slot)

**5. a)**h1(69,0) = (h(69) + 0)% 10

h(69) = 69% 10 = 9

h1(69,0) = 9 (Collision)

**b)**h1(69,1) = (h(69) + 12)% 10

= (9+1)% 10

h1(69,1) = 0 (Collision)

**c)**h1(69,2) = (h(69) + 22)% 103

=(9+4)% 10

h1(69,2) = 3 (empty slot)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  | 0 |  | 0 |  | 0 | 49 | 0 | 49 | 0 | 49 |
| 1 |  | 1 |  | 1 |  | 1 |  | 1 |  | 1 |  |

2

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

3

4

5

6

7

8

9

10

Empty hash table

2

3

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 89 |
|  |

4

5

6

7

8

9

10

Insert 89

2

3

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
| 18 |
| 89 |
|  |

4

5

6

7

8

9

10

Insert 18

2

3

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
| 18 |
| 89 |
|  |

4

5

6

7

8

9

10

Insert 49

2

3

|  |
| --- |
| 58 |
|  |
|  |
|  |
|  |
|  |
| 18 |
| 89 |
|  |

4

5

6

7

8

9

10

Insert 58

2

3

|  |
| --- |
| 58 |
| 69 |
|  |
|  |
|  |
|  |
| 18 |
| 89 |
|  |

4

5

6

7

8

9

10

Insert 69

#### Drawbacks of Quadratic probing :

1. There is no guarantee of finding an empty cell once more than half of the table gets full or even before that, if the table size is not prime.
2. It suffers from Secondary Clustering. If two keys have the same initial probe position, then their probe sequences are the same.

If h1(k1,0) = h1(k2,0) then h 1(k1,i) = h1(k2,i) for i=1,2,3…….

##### Double Hashing :

* + Double Hashing overcomes the problem of Secondary Clustering .
  + This approach uses 2 hash functions. This second hash function results the value of m for a key which is different from the first hash function value.
  + Double hashing uses hash function of the form

**h(k,i) = (h1(k) + i h2(k)) mod m** for i=0,1,2,3…….

* + Here h1 and h2 are the auxiliary hash functions
  + There are two important rules to be followed for the second function:

1. It must never evaluate to Zero
2. Must make sure that all the slots can be examined
   * Generally used hash functions are First hash function=> **h1(k)=k mod m**

Second hash function => **h2(k)=R -(k mod R)**

* + In second hash function, R is a prime number, which is smaller than m.
  + ***Disadvantage*** of double hashing is that Comple x computation is required. We have to use the second hash function when there is a collision.
  + ***Example :***Insert the following key s89, 18, 49, 58, 69, 59into a hash table of size 10 using Double hashing

h(k,i) = (h1(k) + i h2(k)) mod m for i=0,1,2,3……. h1(k) = k% m

h2(k) = R – k mod r

m= 10 and consider R = 7 which is prime and less than m

**1.** h(89,0) = (h1(89)+ 0\*h2(89)) mod 10 h1(89) = 89% 10 =9

h(89,0) = 9 (empty slot)

**2.** h(18,0) = (h1(18)+ 0\*h2(18)) mod 10 h1(18) = 18% 10 =8

h(18,0) = 8 (empty slot)

**3. a)**h(49,0) = (h1(49)+ 0\*h2(49)) mod 10 h1(49) = 49% 10 = 9

h(49,0) = 9 (Collision)

**b)**h(49,1) = (h1(49)+ 1\*h2(49)) mod 10 h1(49) = 49% 10 = 9

h2(49) = 7-(49% 7) = 7

h1(49,1) = (9+7) mod 10

= 6 (empty slot)

**4. a)** h(58,0) = (h1(58)+ 0\*h2(58)) mod 10 h1(58) = 58% 10 = 8

h(58,0) = 8 (Collision)

**b)**h(58,1) = (h1(58)+ 1\*h2(58)) mod 10 h1(58) = 58% 10 = 8

h2(58) = 7-(58% 7) = 5

h(58,1) = (8+5) mod 10

= 3 (empty slot)

**5. a)** h(69,0) = (h1(69)+ 0\*h2(69)) mod 10 h1(69) = 69% 10 = 9

h(69,0) = 9 (Collision)

**b)**h(69,1) = (h1(69)+ 1\*h2(69)) mod 10 h1(69) = 69% 10 = 9

h2(69) = 7-(69% 7) = 1

h(69,1) = (9+1) mod 10

= 0 (empty slot)

**6. a)** h(59,0) = (h1(59)+ 0\*h2(59)) mod 10 h1(59) = 59% 10 = 9

h(59,0) = 9 (Collision)

**b)**h(59,1) = (h1(59)+ 1\*h2(59)) mod 10 h1(59) = 59% 10 = 9

h2(59) = 7-(59% 7) = 4

h(59,1) = (9+4) mod 10

h(59,1) = 3 (Collision)

**c)**h(59,2) = (h1(59)+ 2\*h2(59)) mod 10 h1(59) = 59% 10 = 9

h2(59) = 7-(59% 7) = 4 h(59,2) = (9+2\*4) mod 10

= (9+8) mod 10

h(59,2) = 7 (empty slot)

**Empty Hash table**

**Insert 89**

**Insert 18**

**Insert 49**

**Insert 58**

**Insert 69**

**Insert 59**

0 0 0 0

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 89 |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 18 |
| 89 |

1 1 1 1

2 2 2 2

3 3 3 3

4 4 4 4

5 5 5 5

6 6 6 6

7 7 7 7

8 8 8 8

9 9 9 9

0 0 0

1 1 1

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
| 49 |
|  |
| 18 |
| 89 |

|  |
| --- |
|  |
|  |
|  |
| 58 |
|  |
|  |
| 49 |
|  |
| 18 |
| 89 |

|  |
| --- |
| 69 |
|  |
|  |
| 58 |
|  |
|  |
| 49 |
|  |
| 18 |
| 89 |

|  |
| --- |
| 69 |
|  |
|  |
| 58 |
|  |
|  |
| 49 |
| 59 |
| 18 |
| 89 |

2 2 2

3 3 3

4 4 4

5 5 5

6 6 6

7 7 7

8 8 8

9 9 9