SENIOR
45/23/26/3/21

CHAPTER 4

Hash Table

Objectives ...

- To study Basic Concepts of Hashing
- To learn Hash Table and Hash Function
- To understand Terminologies in Hashing
- To study Collision Resolution Techniques

4.0 INTRODUCTION

- Hashing is a data structure which is designed to use a special function called the hash function which is used to map a given value with a particular key for faster access of elements.
- Hashing is the process of mapping large amount of data item to smaller table with the help of hashing function. IMP
- The mapping between an item and the slot where that item belongs in the hash table is called the hash function.
- The hash function will take any item in the collection and return an integer in the range of slot names, between 0 and m–1.
- A hash table is a collection of items which are stored in such a way as to make it easy to find them later.

Need for Hash Table Data Structure:

- In the linear search and binary search, the location of the item is determined by a sequence/series of comparisons. The data item to be searched is compared with items at certain locations in the list.
- If any item/element matches with the item to be searched, the search is successful. The number of comparisons required to locate an item depends on the data structure like array, linked list, sorted array, binary search tree, etc. and the search algorithm used.
- For example, if the items are stored in sorted order in an array, binary search can be applied which locates an item in O(log n) comparisons.
- On the other hand, if an item is to be searched in a linked list or an unsorted array, linear search has to be applied which locates an item in O(n) comparisons.

- However, for some applications, searching is very critical operation and they need a search algorithm which performs search in constant time, i.e. O(1).
- Although, ideally it is almost impossible to achieve a performance of O(1), but still a search algorithm can be derived which is independent of n and can give a performance very close to O(1).
- That search algorithm is called hashing. Hashing uses a data structure called hash table which is merely an array of fixed size and items in it are inserted using a function called hash function.
- Best case timing behavior of searching using hashing = O(1) and Worst case timing Behavior of searching using hashing = O(n).
- A hash table is a data structure in which the location of a data item is determined directly as a function of the data item itself rather than by a sequence of comparisons.
- Under ideal condition, the time required to locate a data item in a hash table is O(1) i.e. it is constant and does nor depend on the number of data items stored.

4.1 CONCEPT OF HASHING

- Hashing is the process of indexing and retrieving element (data) in a data structure to provide a faster way of finding the element using a hash key.
- Here, the hash key is a value which provides the index value where the actual data is likely to be stored in the data structure.
- In this data structure, we use a concept called hash table to store data. All the data values are inserted into the hash table based on the hash key value.
- The hash key value is used to map the data with an index in the hash table. And the hash key is generated for every data using a hash function.
- That means every entry in the hash table is based on the hash key value generated using the hash function.
- Hash table is just an array which maps a key (data) into the data structure with the help of hash function such that insertion, deletion and search operations are performed.
- Generally, every hash table makes use of a function called hash function to map the data into the hash table.
- Hash function is a function which takes a piece of data (i.e. key) as input and produces an integer (i.e. hash value) as output which maps the data to a particular index in the hash table.
- Basic concept of hashing and hash table is shown in the Fig. 4.1.

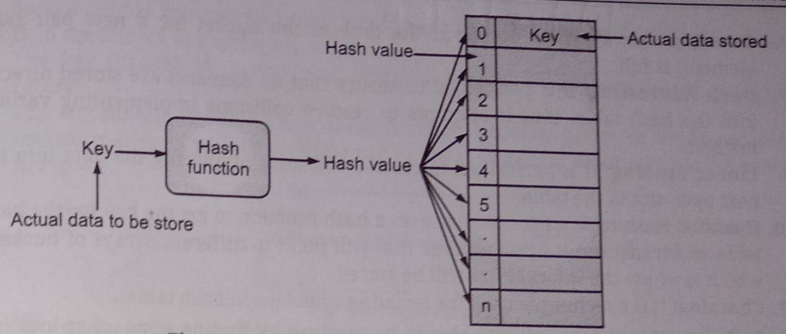

Fig. 4.1: Pictorial representation of Simple Hashing

4.2 TERMINOLOGY

- The basic terms used in hashing are explained below:

1. **Hash Table:** A hash table is a data structure that is used to store keys/value pairs. Hashing uses a data structure called hash table which is merely an array of fixed size and items in it are inserted using a hash function. It uses a hash function to compute an index into an array in which an element will be inserted or searched. A hash table is a data structure that maps keys to values. a hash table (hash map) is a data structure that implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found.
2. **Hash Function:** A hash function is a function that maps the key to some slot in the hash table. A hash function, is a mapping function which maps all the set of search keys to the address where actual records are placed.
3. **Bucket:** A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.
4. Hash Address: A hash function is a function which when given a key, generates an address in the table. Hash index is an address of the data block.
5. **Collision:** The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision. A collision occurs when two data elements are hashed to the same value and try to occupy the same space in the hash table. In simple words, the situation in which a key hashes to an index which is already occupied by another key is called as collision.
6. **Synonym:** It is possible for different keys to hash to the same array location. This situation is called collision and the colliding keys are called synonyms.

7. **Overflow:** An overflow occurs at the time of the bucket for a new pair (key, element) is full.
8. **Open Addressing:** It is performed to ensure that all elements are stored directly into the hash table, thus it attempts to resolve collisions implementing various methods.
9. **Linear Probing:** It is performed to resolve collisions by placing the data into the next open slot in the table.
10. **Hashing:** Hashing is a process that uses a hash function to get the key for the hash table and transform it into an index that will point to different arrays of buckets, which is where the information will be stored.
11. **Chaining:** It is a technique used for avoiding collisions in hash tables.
12. **Collision Resolution:** Collision should be resolved by finding some other location to insert the new key, this process of finding another location is called as collision resolution.

4.3 PROPERTIES OF GOOD HASH FUNCTION

- The properties of a good hash function are given below:

1. The hash function is easy to understand and simple to compute.
2. A number of collisions should be less while placing the data in the hash table.
3. The hash functipn should generate different hash values for the similar string.
4. The hash function "uniformly" distributes the data across the entire set of possible hash values.
5. The hash function is a perfect hash function when it uses all the input data. A hash function that maps each item into a unique slot is referred to as a perfect hash function.

4.4 HASH FUNCTIONS

- A hash function h is simply a mathematical formula that manipulates the key in some form to compute the index for the key in the hash table.
- The process of mapping keys to appropriate slots in a hash table is known as hashing. Hash function is a function which is used to put the data in the hash table.
- Hence, one can use the same hash function to retrieve the data from the hash table. Thus, hash function is used to implement the hash table.
- A hash function h is simply a mathematical formula that maps the key to some slot in the hash table T.
- Thus, we can say that the key k hashes to slot h(k), or h(k) is the hash value of key k. If the size of the hash table is N, then the index of the hash table ranges from 0 to N-1. A hash table with N slots is denoted by T[N].
- Hashing (also known as hash addressing) is generally applied to a file F containing R records. Each record contains many fields, out of these one particular field may uniquely identify the records in the file.

- Such a field is known as primary key (denoted by k). The values k_1, k_2,... k_n in the key field are known as keys or key values.
- The key through an algorithmic function determines the location of a particular record.

The algorithmic function i.e. hashing function basically performs the key-to-address transformation in which key is mapped to the addresses of records in the file as shown in Fig. 4.2.

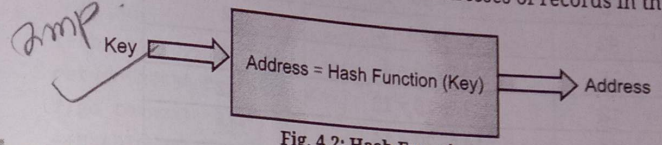

Fig. 4.2: Hash Function

4.4.1 Division Method

- In division method, the key k is divided by the number of slots N in the hash table, and the remainder obtained after division is used as an index in the hash table. That is, the hash function is,

$$h(k) = k \bmod N$$

where, mod is the modulus operator. Different languages have different operators for calculating the modulus. In C/C++, '%' operator is used for computing the modulus.
- For example, consider a hash table with N=101. The hash value of the key value 132437 can be calculated as follows:

$$h(132437) = 132437 \bmod 101 = 26$$

- Note that above hash function works well if the index ranges from 0 to N-1 (like in C/C++). However, if the index ranges from 1 to N, the function will be,

$$h(k) = k \bmod N + 1$$

- This technique works very well if N is either a prime number not too close to a power of two. Moreover, since this technique requires only a single division operation, it is quite fast.
- For examole, suppose, k = 23, N = 10 then
h(23) = 23 mod 10 + 1 = 3 + 1 = 4. The key whose value is 23 is placed in 4th location.
- Take another example, consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key, value) format.hash function is, h(k) = k mod 20.

(1, 20)
(2, 70)
(42, 80)
(4, 25)
(12, 44)
(14, 32)
(17, 11)
(13, 78)
(37, 98)

- Hashing is a technique to convert a range of key values into a range of indexes of an array.

Sr. No.	Key	Hash	Array Index
1.	1	1 % 20 = 1	1
2.	2	2 % 20 = 2	2
3.	42	42 % 20 = 2	2
4.	4	4 % 20 = 4	4
5.	12	12 % 20 = 12	12
6.	14	14 % 20 = 14	14
7.	17	17 % 20 = 17	17
8.	13	13 % 20 = 13	13
9.	37	37 % 20 = 17	17

Basic Operations:

- Following are the basic primary operations of a hash table:
  1. **Search:** Searches an element in a hash table.
  2. **Insert:** inserts an element in a hash table.
  3. **Delete:** Deletes an element from a hash table.

Program 4.1: Program for operations on hashing.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#define SIZE 20
struct DataItem
{
   int data;
   int key;
};
struct DataItem* hashArray[SIZE];
struct DataItem* dummyItem;
struct DataItem* item;
int hashCode(int key)
{
   return key % SIZE;
}
```



```
struct DataItem *search(int key)
{
   //get the hash
   int hashIndex = hashCode(key);
   //move in array until an empty
   while(hashArray[hashIndex] != NULL)
   {
      if(hashArray[hashIndex]->key == key)
      return hashArray[hashIndex];
      //go to next cell
      ++hashIndex;
      //wrap around the table
      hashIndex %= SIZE;
   }
   return NULL;
}
void insert(int key,int data)
{
   struct DataItem *item = (struct DataItem*) malloc(sizeof(struct DataItem));
   item->data = data;
   item->key = key;
   //get the hash
   int hashIndex = hashCode(key);
   //move in array until an empty or deleted cell
   while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1)
   {
      //go to next cell
      ++hashIndex;
      //wrap around the table
      hashIndex %= SIZE;
   }
   hashArray[hashIndex] = item;
}
struct DataItem* delete(struct DataItem* item)
{
   int key = item->key;
   //get the hash
   int hashIndex = hashCode(key);
   //move in array until an empty
```

```
      while(hashArray[hashIndex] != NULL)
      {
         if(hashArray[hashIndex]->key == key)
         {
            struct DataItem* temp = hashArray[hashIndex];
            //assign a dummy item at deleted position
            hashArray[hashIndex] = dummyItem;
            return temp;
         }
         //go to next cell
         ++hashIndex;
         //wrap around the table
         hashIndex %= SIZE;
      }
      return NULL;
   }
   void display()
   {
      int i = 0;
      for(i = 0; i<SIZE; i++)
      {
         if(hashArray[i] != NULL)
            printf(" (%d,%d)",hashArray[i]->key,hashArray[i]->data);
         else
            printf(" ~~ ");
      }
      printf("\n");
}
int main()
{
   dummyItem = (struct DataItem*) malloc(sizeof(struct DataItem));
   dummyItem->data = -1;
   dummyItem->key = -1;
   insert(1, 20);
   insert(2, 70);
   insert(42, 80);
   insert(4, 25);
   insert(12, 44);
```



```
   insert(14, 32);
   insert(17, 11);
   insert(13, 78);
   insert(37, 97);
   display();
   item = search(37);
   if(item != NULL)
   {
      printf("Element found: %d\n", item->data);
   } else
   {
      printf("Element not found\n");
   }
   delete(item);
   item = search(37);
   if(item != NULL)
   {
      printf("Element found: %d\n", item->data);
   } else
   {
      printf("Element not found\n");
   }
}
```

Output:

```
$gcc -o hashpgm *.c
$hashpgm
~~  (1,20) (2,70) (42,80) (4,25) ~~  ~~  ~~  ~~  ~~  ~~  ~~  (12,44)
(13,78) (14,32) ~~  ~~  (17,11) (37,97) ~~
Element found: 97
Element not found
```

4.4.2 Mid Square Method

- In this method, we square the value of a key and take the number of digits required to form an address, from the middle position of squared value.
- Suppose a key value is 16, then its square is 256. Now if we want address of two digits, then we select the address as 56 (i.e., two digits starting from middle of 256).
- Mid square method operates in following two steps:
  1. First, the square of the key k (that is, k^2) is calculated and
  2. Then some of the digits from left and right ends of k^2 are removed.

- The number obtained after removing the digits is used as the hash value. Note that the digits at the same position of k^2 must be used for all keys.
- Thus, the hash function is given below,

  $h(k) = s$

  where, s is obtained by deleting digits from both sides of k^2.
- For example, consider a hash table with N = 1000. The hash value of the key value 132437 can be calculated as follows:
  1. The square of the key value is calculated, which is, 17539558969.
  2. The hash value is obtained by taking 5^{th}, 6^{th} and 7^{th} digits counting from right, which is, 955.

4.4.3 Folding Method

- The folding method also operates in two steps. In the first step, the key value k is divided into number of parts, k^1, k^2, k^3, ... , k^r, where each part has the same number of digits except the last part, which can have lesser digits.
- In the second step, these parts are added together and the hash value is obtained by ignoring the last carry, if any. For example, if the hash table has 1000 slots, each part will have three digits, and the sum of these parts after ignoring the last carry will also be three-digit number in the range of 0 to 999.
- For example, if the hash table has 100 slots, then each group will have two digits, and the sum of the groups after ignoring the last carry will also be a 2-digit number between 0 and 99. The hash value for the key value 132437 is computed as follows:
  1. The key value is first broken down into a group of 2-digit numbers from the left most digits. Therefore, the groups are 13, 24 and 37.
  2. These groups are then added like 13 + 24 + 37 = 74. The sum 74 is now used the hash value for the key value 132437.
- Similarly, the hash value of another key value, say 6217569, can be calculated as follows:
  1. The key value is first broken down into a group of 2-digit numbers from the left most digits. Therefore, the groups are 62, 17, 56 and 9.
  2. These groups are then added like 62 + 17 + 56 + 9 = 144. The sum 44 after ignoring the last carry 1 is now used as the hash value for the key value 6217569.

4.5 COLLISION RESOLUTION TECHNIQUES

- Collision resolution is the main problem in hashing. The situation in which a key hashes to an index which is already occupied by another key is called collision.
- Collision should be resolved by finding some other location to insert the new key. This process of finding another location is called collision resolution.

- If the element to be inserted is mapped to the same location, where an element is already inserted then we have a collision and it must be resolved.
- There are several strategies for collision resolution. The most commonly used are:
  1. **Separate Chaining:** Used with open hashing.
  2. **Open Addressing:** Used with closed hashing.

4.5.1 Open Addressing

- Open addressing or closed hashing is a method of collision resolution in hash tables. With this method a hash collision is resolved by probing, or searching through alternate locations in the array (the probe sequence) until either the target record is found.
- In open addressing separate data structure is used because all the key values are stored in the hash table itself. Since, each slot in the hash table contains the key value rather than the address value, a bigger hash table is required in this case as compared to separate chaining.
- Some value is used to indicate an empty slot. For example, if it is known that all the keys are positive values, then –1 can be used to represent a free or empty slot.
- To **insert a key value**, first the slot in the hash table to which the key value hashes is determined using any hash function. If the slot is free, the key value is inserted into that slot.
- In case the slot is already occupied, then the subsequent slots, starting from the occupied slot, are examined systematically in the forward direction, until an empty slot is found. If no empty slot is found, then overflow condition occurs.
- In case of **searching of a key value** also, first the slot in the hash table to which the key value hashes is determined using any hash function. Then the key value stored in that slot is compared with the key value to be searched.
- If they match, the search operation is successful; otherwise alternative slots are examined systematically in the forward direction to find the slot containing the desired key value. If no such slot is found, then the search is unsuccessful.
- The process of examining the slots in the hash table to find the location of a key value is known as probing. The linear probing, quadratic probing and double hashing that are used in open addressing method.

4.5.1.1 Linear Probing

- Linear probing is a technique for resolving collisions in hash tables, data structures for maintaining a collection of key–value pairs and looking up the value associated with a given key.
- In linear probing whenever there is a collision, cells are searched sequentially (with wraparound) for searching the hash-table free location.

- Let us suppose, if k is the index retrieved from the hash function. If the kth index is already filled then we will look for (k+1) %M, then (k+2) %M and so on. When we get a free slot, we will insert the object into that free slot.
- Below table shows the result of Inserting keys (5,18,55,78,35,15) using the hash function h(k) = k % 10 and linear probing strategy.

	Empty Table	After 5	After 18	After 55	After 78	After 35	After 15
0							15
1							
2							
3							
4							
5		5	5	5	5	5	5
6				55	55	55	55
7						35	35
8			18	18	18	18	18
9					78	78	78

- Linear probing is easy to implement but it suffers from "primary clustering".
- When many keys are mapped to the same location (clustering), linear probing will not distribute these keys evenly in the hash table. These keys will be stored in neighborhood of the location where they are mapped. This will lead to clustering of keys around the point of collision.

Example 1: Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key, value) format. Here, we can search the next empty location in the array by looking into the next cell until we find an empty cell. This technique is called linear probing.

(1, 20)
(2, 70)
(42, 80)
(4, 25)
(12, 44)
(14, 32)
(17, 11)
(13, 78)
(37, 98)

Solution:

Sr. No.	Key	Hash	Array Index	After Linear Probing, Array Index
1.	1	1 % 20 = 1	1	1
2.	2	2 % 20 = 2	2	2
3.	42	42 % 20 = 2	2	3
4.	4	4 % 20 = 4	4	4
5.	12	12 % 20 = 12	12	12
6.	14	14 % 20 = 14	14	14
7.	17	17 % 20 = 17	17	17
8.	13	13 % 20 = 13	13	13
9.	37	37 % 20 = 17	17	18

Example 2: A hash table of length 10 uses open addressing with hash function h(k)=k mod 10, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below:

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

What is a possible order in which the key values could have been inserted in the table?

Solution: 46, 34, 42, 23, 52, 33 is the sequence in which the key values could have been inserted in the table.

How many different insertion sequences of the key values using the same hash function and linear probing will result in the hash table shown above?

Solution: 30

In a valid insertion sequence, the elements 42, 23 and 34 must appear before 52 and 33, and 46 must appear before 33.

Total number of different sequences = $3! \times 5 = 30$

In the above expression, 3! is for elements 42, 23 and 34 as they can appear in any order, and 5 is for element 46 as it can appear at 5 different places.

4.5.1.2 Quadratic Probing

- Quadratic probing is a collision resolving technique in open addressing hash table. It operates by taking the original hash index and adding successive values of an arbitrary quadratic polynomial until an open slot is found.
- One way of reducing "primary clustering" is to use quadratic probing to resolve collision. In quadratic probing, we try to resolve the collision of the index of a hash table by quadratically increasing the search index free location.
- Let us suppose, if k is the index retrieved from the hash function. If the kth index is already filled then we will look for $(k+1^2)$ %M, then $(k+2^2)$ %M and so on. When we get a free slot, we will insert the object into that free slot.
- It does not ensure that all cells in the table will be examined to find an empty cell. Thus, it may be possible that key will not be inserted even if there is an empty cell in the table.

Double Hashing:

- Double hashing is a collision resolving technique in open addressing hash table. Double hashing uses the idea of using a second hash function to key when a collision occurs.
- This method requires two hashing functions for f1(key) and f2 (key). Problem of clustering can easily be handled through double hashing.
- Function f1 (key) is known as primary hash function. In case the address obtained by f1 (key) is already occupied by a key, the function f2 (key) is evaluated.
- The second function f2 (key) is used to compute the increment to be added to the address obtained by the first hash function f1 (key) in case of collision.
- The search for an empty location is made successively at the addresses f1 (key) + f2 (key), f1 (key) + 2f2 (key), f1 (key) + 3f2(key), ...

4.5.1.3 Rehashing

- As the name suggests, rehashing means hashing again. Rehashing is a technique in which the table is resized, i.e. the size of table is doubled by creating a new table.
- Several deletion operations are intermixed with insertion operations while performing operations on hash table. Eventually, a situation arises when the hash table becomes almost full.
- At this time, it might happen that the insert, delete, and search operations on the hash table take too much time. The insert operation even fails in spite of performing open addressing with quadratic probing for collision resolution. This condition/situation indicates that the current space allocated to the hash table is not sufficient to accommodate all the keys.

- A simple solution to this problem is rehashing, in which all the keys in the original hash table are rehashed to a new hash table of larger size.
- The default size of the new hash table is twice as that of the original hash table. Once the new hash table is created, a new hash value is computed for each key in the original hash table and the keys are inserted into the new hash table.
- After this, the memory allocated to the original hash table is freed. The performance of the hash table improves significantly after rehashing.

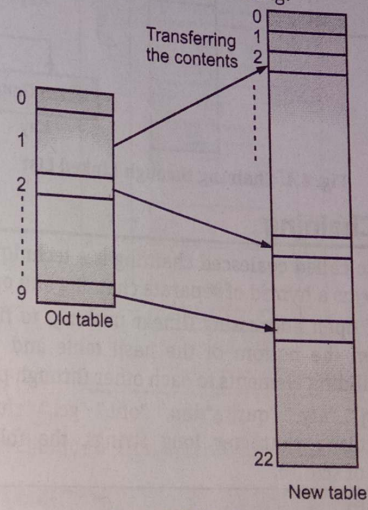

Fig. 4.3: Rehashing

4.5.2 Chaining

- In hashing the chaining is one collision resolution technique. Chaining is a possible way to resolve collisions. Each slot of the array contains a link to a singly-linked list containing key-value pairs with the same hash.
- Chaining allows storing the key elements with the same hash value into linked list as shown in Fig. 4.4.
- Thus, cach slots in the hash table contains a pointer to the head of the linked list of all the elements that hashes to the value h.
- All collisions are chained in the lists attached to the appropriate slot. This allows an unlimited number of collisions to be handled and does not require a prior knowledge of how many elements are contained in the collection.
- The tradeoff is the same as with linked lists versus array implementations of collections: linked list overhead in space and to a lesser extent, in time.

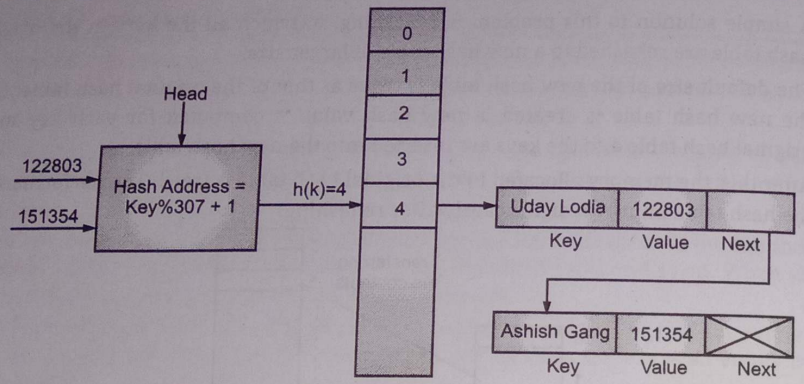

Fig. 4.4: Chaining through Linked List

4.5.2.1 Coalesced Chaining

- Coalesced hashing also called coalesced chaining is a technique of collision resolution in a hash table that forms a hybrid of separate chaining and open addressing.
- It uses the concept of open addressing (linear probing) to find first empty place for colliding element from the bottom of the hash table and the concept of separate chaining to link the colliding elements to each other through pointers.
- Given a sequence "qrj," "aty," "qur," "dim," "ofu," "gcl," "rhv," "clq," "ecd," "qsu" of randomly generated three character long strings, the following table would be generated with a table of size 10.

(null)			
"clq"			
"qur"			
(null)			
(null)			
"dim"			
"aty"	"qsu"		
"rhv"			
"qrj"	"ofu"	"gel"	"ecd"
(null)			
(null)			

- Fig. 4.5 shows an example of coalesced hashing example (for purpose of this example, collision buckets are allocated increasing order, starting with bucket 0).

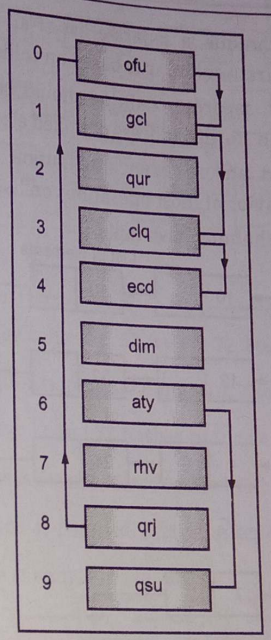

Fig. 4.5

- Coalesced hashing technique is effective, efficient, and very easy to implement.

4.5.2.2 Separate Chaining

- In separate chaining, hash table is an array of pointers and each element of array points to the first element of the linked list of all the records that hashes to that location.
- Open hashing is a collision avoidance method which uses array of linked list to resolve the collision. It is also known as the separate chaining method (each linked list is considered as a chain).
- In separate chaining collision resolution technique, a linked list of all the key values that hash to the same hash value is maintained. Each node of the linked list contains a key value and the pointer to the next node.
- Each index i (0 <= I < N) in the hash table contains the address of the first node of the linked list containing all the keys that hash to the index i.
- If there is no key value that hashes to the index i, the slot contains NULL value. Therefore, in this method, a slot in the hash table does not contain the actual key values; rather it contains the address of the first node of the linked list containing the elements that hash to this slot.

- In separate chaining technique, a separate list of all elements mapped to the same value is maintained. Separate chaining is based on collision avoidance.
- If memory space is tight, separate chaining should be avoided. Additional memory space for links is wasted in storing address of linked elements.
- Hashing function should ensure even distribution of elements among buckets; otherwise the timing behavior of most operations on hash table will deteriorate.
- Fig. 4.6 shows a separate chaining hash table.

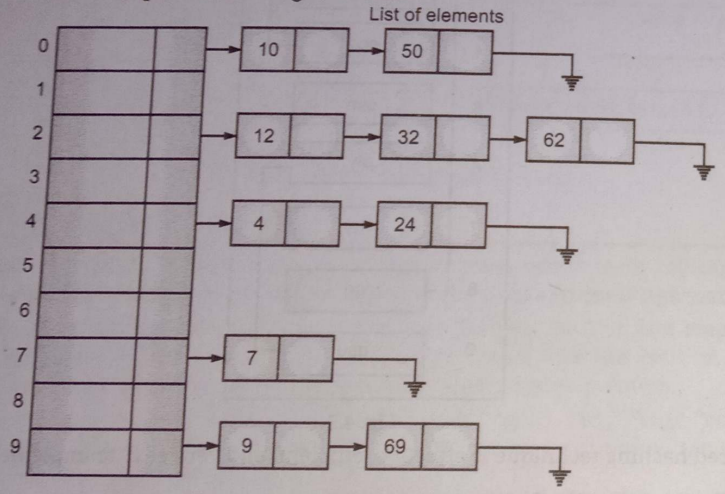

Fig. 4.6: A separate Chaining Hash Table

Example 1: The integers given below are to be inserted in a hash table with 5 locations using chaining to resolve collisions. Construct hash table and use simplest hash function.
1, 2, 3, 4, 5, 10, 21, 22, 33, 34, 15, 32, 31, 48, 49, 50,

Solution: An element can be mapped to a location in the hash table using the mapping function h(k) = k % 10.

Hash Table Location	Mapped Elements
0	5, 10, 15, 50
1	1, 21, 31
2	2, 22, 32
3	3, 33, 48
4	4, 34, 49

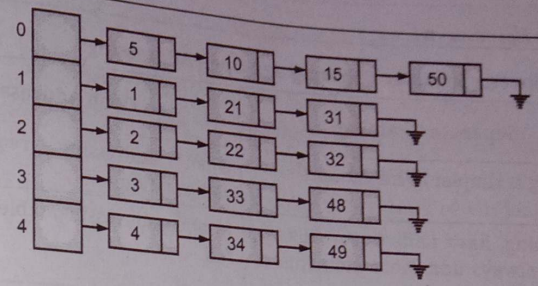

Fig. 4.7

Example 2: Consider the key values 20, 32, 41, 66, 72, 80, 105, 77, 56, 53 that need to be hashed using the simple hash function h(k) = k mod 10. The keys 20 and 80 hash to index 0, key 41 hashes to index 1, keys 32 and 72 hashes to index 2, key 53 is hashed to index 3, key 105 is hashed to index 5, keys 66 and 56 are hashed to index 6 and finally the key 77 is hashed to index 7. The collision is handled using the separate chaining (also known as synonyms chaining) technique as shown in Fig. 4.8.

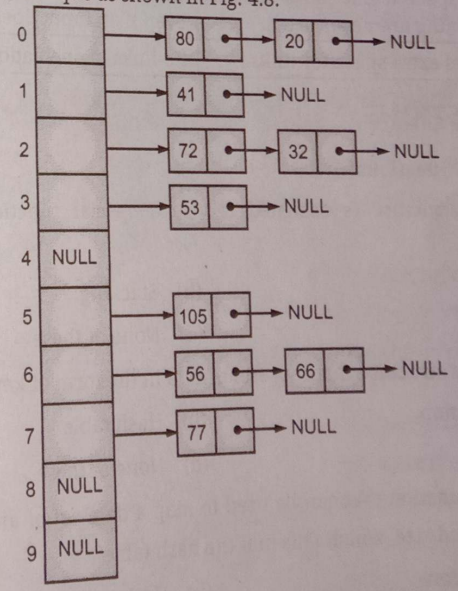

Fig. 4.8: Collision Resoltuion by Separate Chaining

Comparison between Separate Chaining and Open Addressing:

Sr. No.	Separate Chaining	Open Addressing
1.	Chaining is simpler to implement.	Open addressing requires more computation.
2.	In chaining, hash table never fills up, we can always add more elements to chain.	In open addressing, table may become full.
3.	Chaining is less sensitive to the hash function or load factors.	Open addressing requires extra care for to avoid clustering and load factor.
4.	Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.	Open addressing is used when the frequency and number of keys is known.
5.	Cache performance of chaining is not good as keys are stored using linked list.	Open addressing provides better cache performance as everything is stored in the same table.
6.	Wastage of space (some parts of hash table in chaining are never used).	In open addressing, a slot can be used even if an input does not map to it.
7.	Chaining uses extra space for links.	No links in open addressing.

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which data structure is designed to use a special function called the hash function?
   (a) Hashing (b) Stacking
   (c) Queueing (d) None of these
2. Which is a data structure that represents data in the form of key-value pairs?
   (a) Hash function (b) Hash table
   (c) Hashing (d) None of these
3. Which 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?
   (a) Hash function (b) Hash table
   (c) Hash values (d) None of these

4. A good hash function has the following properties:
   (a) **Easy to compute:** It should be easy to compute and must not become an algorithm in itself.
   (b) **Uniform distribution:** It should provide a uniform distribution across the hash table and should not result in clustering.
   (c) **Less collisions:** Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.
   (d) All of these
5. Which tables are used to perform insertion, deletion and search operations very quickly in a data structure?
   (a) Root (b) Hash
   (c) Routing (d) Root
6. Which is in a hash file is unit of storage that can hold one or more records?
   (a) Bucket (b) Hash values
   (c) Token (d) None of these
7. A _____ occurs when two data elements are hashed to the same value and try to occupy the same space in the hash table.
   (a) Bucket (b) Underflow
   (c) Collision (d) Token
8. In which method all the key values are stored in the hash table itself?
   (a) Open addressing (b) Chaining
   (c) Separate chaining (d) None of these
9. Which is a technique in all the keys in the original hash table are rehashed to a new hash table of larger size?
   (a) Chaining (b) Hashing
   (c) Rehashing (d) None of these
10. In which hashing method, successive slots are searched using another hash function?
    (a) Linear probing (b) Quadratic probing
    (c) Double (d) All of these
11. Name which methods are used by open addressing for hashing?
    (a) Linear probing (b) Quadratic probing
    (c) Double hashing (d) All of these

12. Hashing is implementing using,
    (a) Hash table (b) Hash function
    (c) Both (a) and (b) (d) None of these
13. Hash table is a data structure that represents data in the form of,
    (a) value-key pairs (b) key-value pairs
    (c) key-key pairs (d) None of these

Answers

1. (d)	2. (c)	3. (b)	4. (a)	5. (c)	6. (d)	7. (c)
8. (a)	9. (c)	10. (b)	11. (d)	12. (a)	13. (b)	

Q. II Fill in the Blanks:

1. _____ is the process of mapping large amount of data item to a hash table with the help of hash function.
2. The situation where a newly inserted key maps to an already occupied slot in the hash table is called _____.
3. In rehashing the default size of the new table is _____ as that of the original hash table.
4. The process of examining the slots in the hash table to find the location of a key value is known as _____.
5. A _____ function is simply a mathematical formula that manipulates the key in some form to compute the index for this kay in the hash table.
6. Hash _____ is an array of fixed size and items in it are inserted using a hash function.
7. In _____ method the key is divided by number of slots in the hash table and the remainder obtained after division is used as an index in the hash table.
8. In _____ hash table is an array of pointers and each element/item of array points to the first element of the linked list of all the records that hashes to that location.
9. _____ allows storing the key elements with the same hash value into linked list.
10. _____ hashing is a combination of both Separate chaining and Open addressing.

Answers

1. Hashing	2. collision	3. twice	4. probing	5. hash
6. table	7. division	8. Separate chaining	9. Chaining	10. Coalesced

Q. III State True or False:

1. The process of mapping kays to appropriate slots in a hash table is known as hashing.
2. In mid-square method we first square the item, and then extract some portion of the resulting digits.
3. The hash table is depending upon the remainder of division.
4. In open addressing, all elements are stored in the hash table itself.
5. The idea of separate chaining is to make each cell of hash table point to a linked list of records that have same hash function value.
6. open addressing is a method for handling collisions.
7. In linear probing, we linearly probe for next slot.
8. Double hashing uses the idea of applying a second hash function to key when a collision occurs.
9. A hash function is a data structure that maps keys to values.
10. Quadratic probing operates by taking the original hash index and adding successive values of an arbitrary quadratic polynomial until an open slot is found.
11. Coalesced hashing is a collision avoidance technique when there is a fixed sized data.

Answers

1. (T)	2. (T)	3. (F)	4. (T)	5. (T)	6. (T)	7. (T)	8. (T)	9. (F)	10. (T)	11. (T)

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. What is hashing? Hashing is the process of transforming any given key or string of characters into another value
2. What is hash function?
3. Define bucket.
4. List steps for mid square method.
5. What is collision?
6. List techniques for collision resolution.
7. Define rehashing?
8. What is linear probing?
9. Define quadratic probing.
10. Define separate chaining?

(B) Long Answer Questions:

1. Define hashing. State need for hashing. Also list advantages of hashing.
2. What is chaining? Explain with diagram.
3. With the help of example describe separate chaining?

SENIOR 45/23 | 26/3/21

4. Describe quadratic probing with example.
5. With the help of diagram describe concept of hashing.
6. What is hash table? Explain in detail.
7. Write a short note on: Linear probing.
8. Compare separate chaining and open addressing?
9. What is rehashing? Describe with diagrammatically.
10. Differentiate between hashing and rehashing?
11. With the help of diagram describe hash function.
12. Describe coalesced chaining with example.

Syllabus ...

1. Tree **(10 Hrs.)**

1.1 Concept and Terminologies

1.2 Types of Binary Trees - Binary Tree, Skewed Tree, Strictly Binary Tree, Full Binary Tree, Complete Binary Tree, Expression Tree, Binary Search Tree, Heap

1.3 Representation - Static and Dynamic

1.4 Implementation and Operations on Binary Search Tree - Create, Insert, Delete, Search, Tree Traversals - Preorder, Inorder, Postorder (Recursive Implementation), Level-Order Traversal using Queue, Counting Leaf, Non-Leaf and Total Nodes, Copy, Mirror

1.5 Applications of Trees

1.5.1 Heap Sort, Implementation

1.5.2 Introduction to Greedy Strategy, Huffman Encoding (Implementation using Priority Queue)

2. Efficient Search Trees **(8 Hrs.)**

2.1 Terminology: Balanced Trees - AVL Trees, Red Black Tree, Splay Tree, Lexical Search Tree -Trie

2.2 AVL Tree - Concept and Rotations

2.3 Red Black Trees - Concept, Insertion and Deletion

2.4 Multi-Way Search Tree - B and B+ Tree - Insertion, Deletion

3. Graph **(12 Hrs.)**

3.1 Concept and Terminologies

3.2 Graph Representation - Adjacency Matrix, Adjacency List, Inverse Adjacency List, Adjacency Multi-list

3.3 Graph Traversals - Breadth First Search and Depth First Search (With Implementation)

3.4 Applications of Graph

3.4.1 Topological Sorting

3.4.2 Use of Greedy Strategy in Minimal Spanning Trees (Prim's and Kruskal's Algorithm)

3.4.3 Single Source Shortest Path - Dijkstra's Algorithm

3.4.4 Dynamic Programming Strategy, All Pairs Shortest Path - Floyd Warshall Algorithm

3.4.5 Use of Graphs in Social Networks

4. Hash Table **(6 Hrs.)**

4.1 Concept of Hashing

4.2 Terminologies - Hash Table, Hash Function, Bucket, Hash Address, Collision, Synonym, Overflow etc.

4.3 Properties of Good Hash Function

4.4 Hash Functions: Division Function, Mid Square, Folding Methods

4.5 Collision Resolution Techniques

4.5.1 Open Addressing - Linear Probing, Quadratic Probing, Rehashing

4.5.2 Chaining - Coalesced, Separate Chaining

■■■