HASHING

The time required to access any element in the array irrespective of its position is same. The physical location of i^{th} item in the array can be calculated by multiplying the size of each element of the array with i and adding to the base address as shown below:

$$Loc\ (Ai) = Base\text{-}address + i*c$$

Here, i is the index, c is the size of each element of the array. Using this formula, the address of any item at any specified position i can be obtained and from the address obtained, we can access the item. The similar idea is used in *hashing* to store and retrieve the data.

Definition: The data can be stored in the form of a table using arrays and from this table the data has to be retrieved. The table on which insertion, deletion and retrieve operations takes place is called hash table*.* Thus, a hash table can be implemented usually using an ordinary array. The process of mapping large amounts of data into a smaller table is called hashing*.*

The hash function provides the mapping between the original data and the smaller table by transforming a key into an index to the hash table. This function that transforms a key into an index to the hash table is called hash function*.* The various keys are distributed among the locations 0 to m-1 and it is required to compute the keys using hash function. The hash function assigns an integer value from 0 to m-1 to keys and these values which act as index to the hash table are called hash addresses or hash values. A good hash function should satisfy two criteria:

1. A hash function should generate the hash addresses such that all the keys are distributed as evenly as possible among the various cells of the hash table
2. Computation of a key by hash function should be simple.

Collision and its detection: If the size of the hash table is fixed and the size of the table is smaller than the total number of keys generated, then two or more keys will have the same hash address. This phenomenon of two or more keys being hashed to the same location of hash table is called collision.

Note that we can expect collision even if the number of keys generated is less than the size of the hash table. Most of the time collision cannot be avoided and in the worst case all keys may have the same value. In such situation, all the keys may be stored in only one cell in the form of a list and so, it is required to search all n keys in the worst-case requiring time complexity of Θ (n). But, with appropriately chosen size of the hash table and using a good hash function, this phenomenon will not occur and under reasonable assumptions, the expected time to search for an element in a hash table will be Θ (1). So, in practical situation, hashing is extremely effective and a practical technique to implement dictionaries where insertion, deletion and searching takes place frequently. The various hashing techniques using which collision can be avoided are:

1. Open hashing (Separate chaining)
2. closed hashing (Open addressing)

<u>Open Hashing / Chaining (Separate Chaining):</u> The open hashing technique is a very simple method using which collisions can be avoided. This is achieved using an array of linked lists. If an item is to be inserted, using a hash function, the hash address is obtained. This hash address is used to find the list to which the item is to be added and using the appropriate function an item can be inserted at the end of the list. If more than one key has same hashing address, all the keys hashed into same hashing address will be inserted at the end of the list and thus collision is avoided.

The searching using this technique takes less time. When an item has to be searched, we find the hash address using hash function. This hash address corresponds to an index which will be used to obtain the address of the list. If the address of the list is NULL, the search item is not found. If the address of the list is not NULL, that list is traversed sequentially to search for the item is found in the list report successful search and if end of list is encountered report unsuccessful search.

Deleting a node from the hash table is also simple. The hash function is used to determine the hash address which can act as an index of the item to be deleted. The linked list corresponding to the index is searched and if the item is present, the corresponding node in that list can be deleted. During searching if end of list is encountered, the item not found.

The only disadvantage of this technique is that linked list requires extra storage space for the pointers.

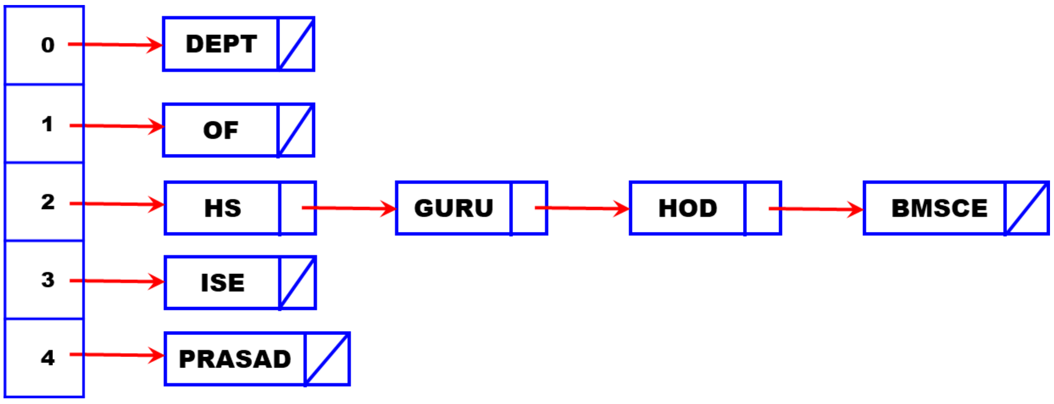
HS	**8 + 19 = 27**	**2**
GURU	**7 + 21 + 18 + 21 = 67**	**2**
PRASAD	**16 + 18 + 1 + 19 + 1 + 4 = 59**	**4**
HOD	**8 + 15 + 4 = 27**	**2**
DEPT	**4 + 5 + 16 + 20 = 45**	**0**
OF	**15 + 6 = 21**	**1**
ISE	**9 + 19 + 5 = 33**	**3**
BMSCE	**2 + 13 + 19 + 3 + 5 = 42**	**2**

Let the take the size of hash table as 5. Insert the following words HS, GURU, PRASAD, HOD, DEPT, OF, ISE, BMSCE into a hash table using appropriate hash function and show how each item is inserted into the table, how an item can be searched and deleted.

Solution: A hash table of size 5 indicates that it is required to construct a hash table by using an array of 5 linked lists. Let us discuss the way to create a hash function, to insert an element into the table, to search for an element and to delete an element from the table.

Design of Hashing function: Let us design a simple hashing function which will accept a string consisting of a key and add the positions of word's letters in the alphabet and compute the remainder by dividing the sum by the size of the table. The words to be inserted into the hash table, the sum of positions of each letter in the word and the hash address are shown in the table. The size of the hash table is assumed to be 5, so that all the hash address of all the words lie within 0 and 4.

Construction of Hash table: Now, let us see how to construct a hash table. Since the size of the hash table is 5, we will have an array of 5 arrows with each row having a linked list. Obtain the words one by one, find the hash address and insert at the end of the list in the appropriate location in the array. The hash table obtained by computing the hash addresses is shown below:



Search for the given patterns: Searching is very efficient if the hash table size is appropriately selected and the items are distributed evenly in the table. If it is required to search for the string STR, then compute the hash address (hash value) of the string STR. If there is no entry in the location identified by hash address, string STR is not present in the table and search is unsuccessful. If there is any entry in that location, search for the string in the string in the corresponding list and if found it can be retrieved. For example, if the word to be searched is "GURU", find the hash value which is given by (7+21+18+21) % 5 = 2. So, it is required to search for "GURU" in h [2]. In the list identified by h [2] has the words "HS", "GURU", "HOD" and "BMSCE". The string "GURU" is compared with all the words in the list. If there is no matching word, unsuccessful message has to be displayed and if found, display the appropriate message. Thus, string search can be achieved very efficiently using hashing technique.

Deletion of specified word: Like search operation, deletion of specified item is also very efficient if the hash table size is appropriately selected and the items are distributed evenly in the table. If it is required to delete the string STR, then compute the hash address (hash value) of the string STR. If there is no entry in the location identified by hash address, string STR is not present in the table and search is unsuccessful and so deletion is not possible. If there is any entry in that location, search for the string in the corresponding list and if found the corresponding node can be deleted from that list.

Closed Hashing (Open Addressing)

In closed hashing or open addressing hashing, the amount of space available for storing various data is fixed at compile time by declaring a fixed array for the hash table. So, all the keys are stored in this fixed hash table itself without the use of linked lists. In such a table, collision can be avoided by finding another, unoccupied location in the array. The various strategies using which collision can be avoided are:

- Linear probing
- Double hashing
-

Linear Probing: In this technique, if an item is to be inserted, as usual, using the hash function find the hash address which gives an index into the array. If the position identified by the index is empty, an item is inserted at that location. In case, at that position if an item already exists, then a collision occurs and a next empty location has to be searched and the item has to be inserted at the empty location. For example, if collision occurs at i^{th} position, the locations from i+1 onwards are verified for empty locations and whenever we get an empty location, item is inserted at that position.

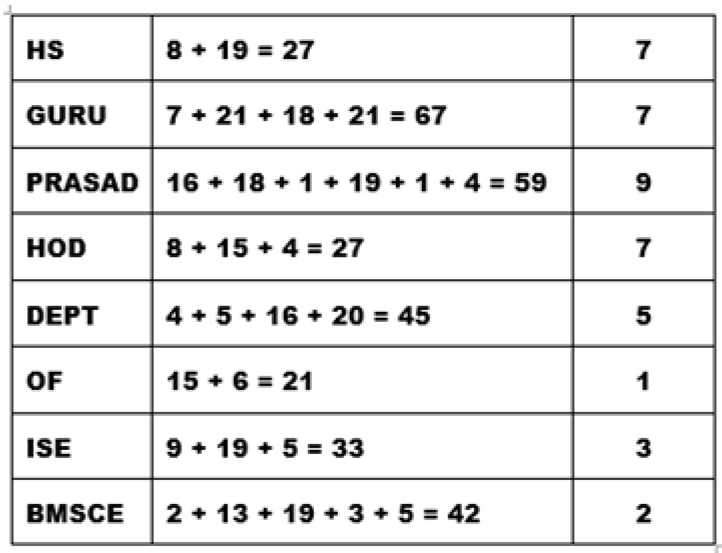
To search for an item, say key, it is required to find the hash address using hash function. Let h () be the hash function which generates the hash address h (key). If the location identified by h (key) is empty, the item is not found in the table and search is unsuccessful. If the location is not empty, it is required to compare the key with item in that location. If they are equal, search is successful. If they are not equal, check the item with the items in the following locations until the item found (and search is successful) or an empty location is encountered. If empty location is encountered, the search is unsuccessful.

Deleting an element using this technique can be quite complex and is explained with a typical example.

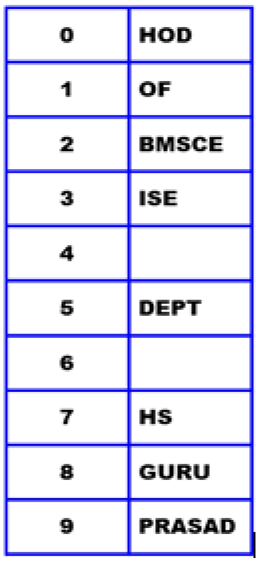
Example: Let the size of hash table is 15. insert the following words HS, GURU, PRASAD, HOD, DEPT, OF, ISE, BMSCE into a hash table using appropriate hash function and show how each item is inserted into the table, how an item can be searched and deleted using linear probing.

Note: In closed hashing, the size of the hash table must be at least equal to the number of unique words in a given text. For example, in this example the size of the hash table should be at least 8, since we have 8 words. Let us assume the size of the hash table as 10.

Insert an item: The words to be inserted into the hash table are: HS, GURU, PRASAD, HOD, DEPT, OF, ISE, BMSCE. As usual find the hash address by taking the sum of positions of alphabets in the word and taking the remainder by dividing it by 10. The various hash addresses are shown below:



HS	8 + 19 = 27	7
GURU	7 + 21 + 18 + 21 = 67	7
PRASAD	16 + 18 + 1 + 19 + 1 + 4 = 59	9
HOD	8 + 15 + 4 = 27	7
DEPT	4 + 5 + 16 + 20 = 45	5
OF	15 + 6 = 21	1
ISE	9 + 19 + 5 = 33	3
BMSCE	2 + 13 + 19 + 3 + 5 = 42	2



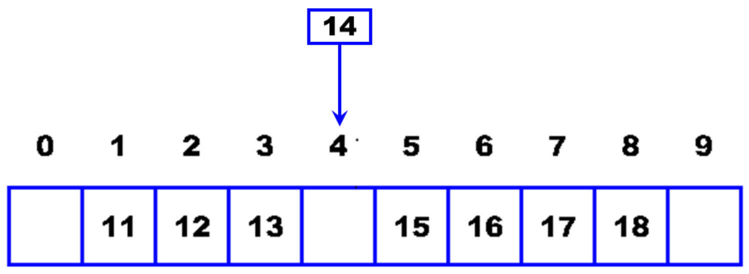
0	HOD
1	OF
2	BMSCE
3	ISE
4	
5	DEPT
6	
7	HS
8	GURU
9	PRASAD

Search for an item: Consider the word “GURU” to be searched. The hash valued k (GURU) = 7 + 21 + 18 + 21 = 67. So, compare the word “GURU” with the items as position 7. Since it is not found, search the next location 8. It is found and hence it is successful search. It is required to search for a word from its hash position onwards, till the word is found or till an empty location is found. If an empty location is found, the word is not found.

Delete an item: Deleting an element from the hash table using this technique is quite complex. Suppose, it is required to delete the word “HS”. Its hash value is 7. So, the location 7 is searched for the word “HS” and since there is a match, we can delete that word from that location. But, we cannot delete the word “GURU”, because, the hash value of the word “GURU” starts from position 7 onwards. Since the word “HS” at location is already deleted, the algorithm encounters an empty location and reports an unsuccessful search result. This problem can be avoided by a technique called “lazy deletion”, that is, mark the previously occupied locations by a special symbol to distinguish them from the locations that have not been occupied. So, the complexity of the algorithm increases.

Disadvantages: The disadvantage of linear probing is that its performance deteriorates because of the phenomenon called clustering. *Clustering* is a phenomenon where the data tend to cluster (gather) at certain points in the table, with other part of the table not being used.

Primary Clustering: Whenever an insertion is made between two clusters that are separated by an empty location, the two clusters become one, thereby potentially increasing the cluster length. This phenomenon is called primary clustering. For example, there are two clusters: the first cluster has three elements 11, 12 and 13 and the second cluster has four elements 15, 16, 17 and 18. If an item, say 14 is inserted at position 4, the insertion results in a single clustering consisting of elements from 11 to 18.



Double hashing
Using this approach, the primary clustering can be avoided. This technique requires two distinct hash function h (key) and S (key) both functions accept the same key as the parameter. Whenever the item has to be inserted or retrieved, the primary hashing function

I = h (key)

is used to determine the position at which the item has to be inserted. The table size m is used in primary hashing function. If that position is occupied by some other element the secondary hash function s (key) is called to get second hash address which can be computed as shown below:

I = (I +s (key)) % m-1

where m is the size of the hash table. Once this new hash address (or index) is obtained, determine if the location identified by I is empty. If empty, insert the item at that position; otherwise, check the subsequent locations obtained by computing the value of I repeatedly using the relation:

I = (I + s (key)) % m-1

Applications: Hashing technique is widely used in generating a symbol table during compilation process. With some modifications to hashing function (called extensible hashing), it is also proved that this technique can be used to store very large dictionaries on disks.