Data Structures
Lab Manual

Topic: Hash Tables

Session: Fall 2022

Faculty of Information Technology

UCP Lahore Pakistan

Hash Table:

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Thus, it becomes a data structure in which insertion, updation and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

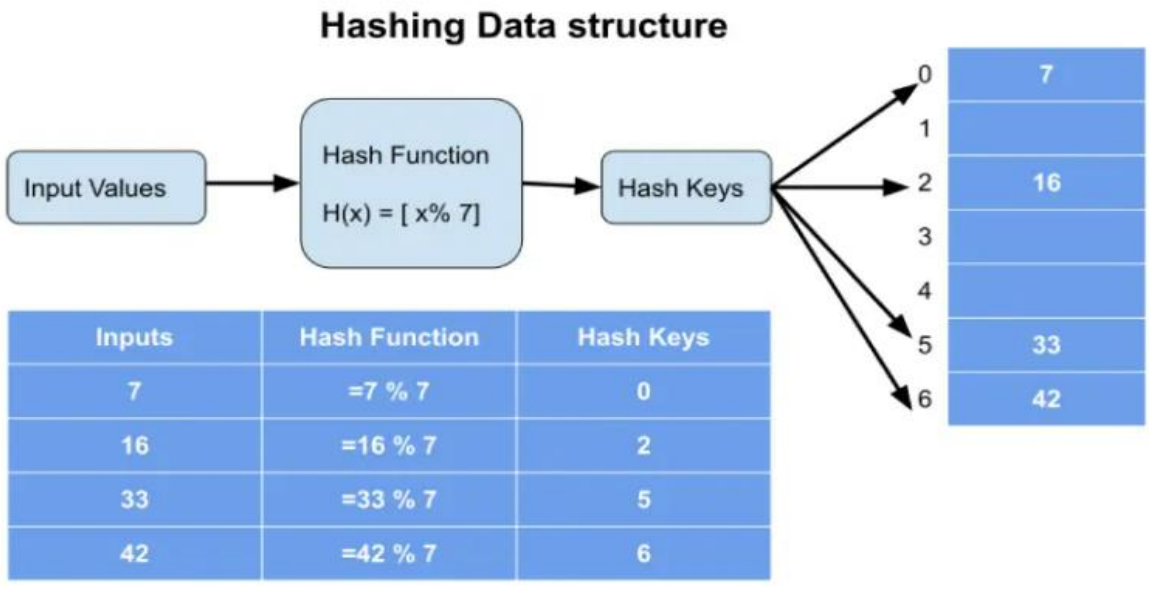
Hashing:

A technique of mapping a large chunk of data into small tables using a hashing function. It converts a range of key values into a range of indexes of an array.

Hash Function:

The hash function in a data structure maps the arbitrary size of data to fixed-sized data. It returns the following values: a small integer value (also known as hash value), hash codes, and hash sums. We're going to use modulo operator to get a range of key values. The hashing techniques in the data structure are very interesting, such as:

hash = hashfunc(key)
index = hash % array_size



Inputs	Hash Function	Hash Keys
7	=7 % 7	0
16	=16 % 7	2
33	=33 % 7	5
42	=42 % 7	6

Linear Probing

As we can see, it may happen that the hashing technique is used to create an already used index of the array. In such a case, 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.

Basic Operations

Following are the basic primary operations of a hash table.

- **Search** − Searches an element in a hash table.
- **Insert** − inserts an element in a hash table.
- **delete** − Deletes an element from a hash table.

DataItem

Define a data item having some data and key, based on which the search is to be conducted in a hash table.

```
struct DataItem {
   int data;
   int key;
};
```

Hash Method

Define a hashing method to compute the hash code of the key of the data item.

```
Int hashCode(int key){
   return key % SIZE;
}
```

Search Operation

Whenever an element is to be searched, compute the hash code of the key passed and locate the element using that hash code as index in the array. Use linear probing to get the element ahead if the element is not found at the computed hash code.

```
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;
}
```

Insert Operation

Whenever an element is to be inserted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing for empty location, if an element is found at the computed hash code.

```
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;
}
```

Delete Operation

Whenever an element is to be deleted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing to get the element ahead if an element is not found at the computed hash code. When found, store a dummy item there to keep the performance of the hash table intact.

```
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;
}
```

TASKS

Task 1

Consider an empty hash table of size 10 and insert following keys 12, 18, 13, 2, 3, 23, 5 and 15 using open addressing with hash function h(k) = k mod 10 and linear probing.

Task 2

Perform the following operations
Search: search a key inside the hash table and return the value that is associated with that key

Insert: insert a new key-value pair inside the hash table

Delete: delete a particular key from the hash table

Delete Hash table: delete the hash table