Data Structure Lab

Hashing

Instructor: Muhammad Saood Sarwar

Email: saood.sarwar@nu.edu.pk

Course Code: CL2001

Department of Computer Science,

National University of Computer and Emerging Sciences FAST Peshawar Campus

Hash Table

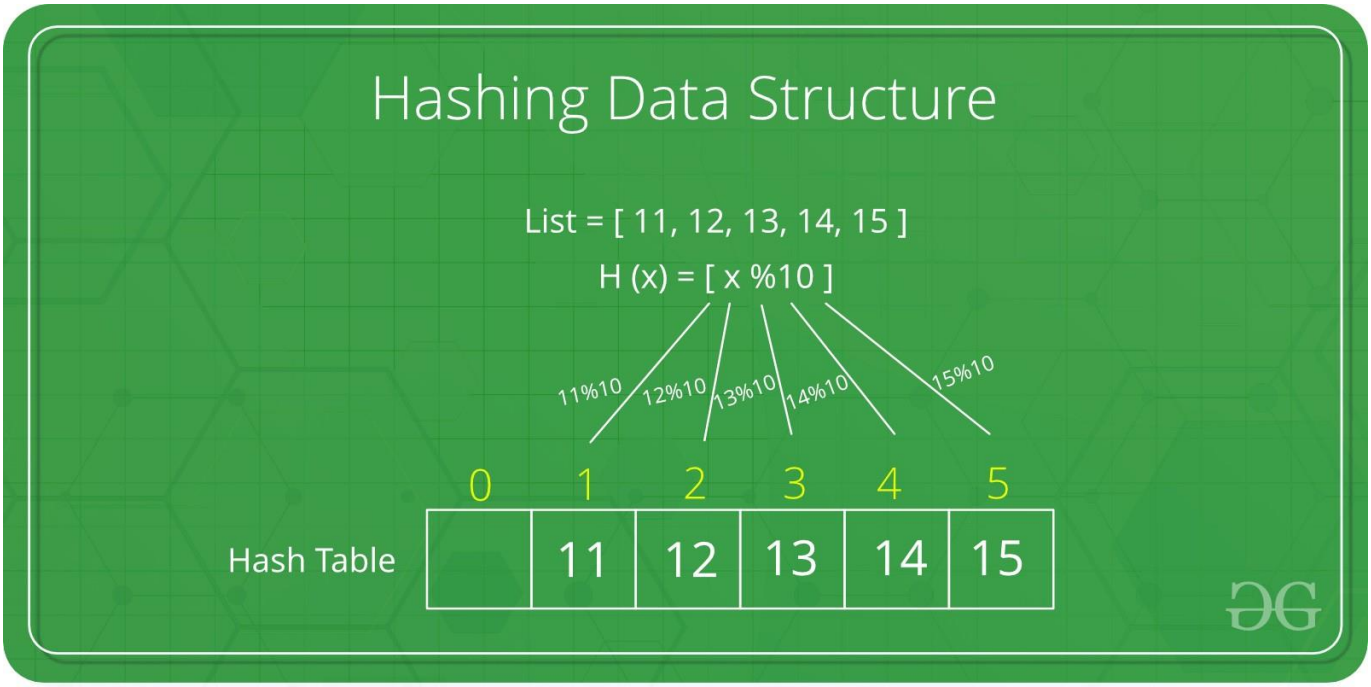
The Hash table data structure stores elements in key-value pairs where

- Key- unique integer that is used for indexing the values.
- Value - data that is associated with keys.

Hashing (Hash Function)

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

Let a hash function H(x) maps the value **x** at the index **x%10** in an Array. For example, if the list of valuesis [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



Hash Collision

When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is called a hash collision.

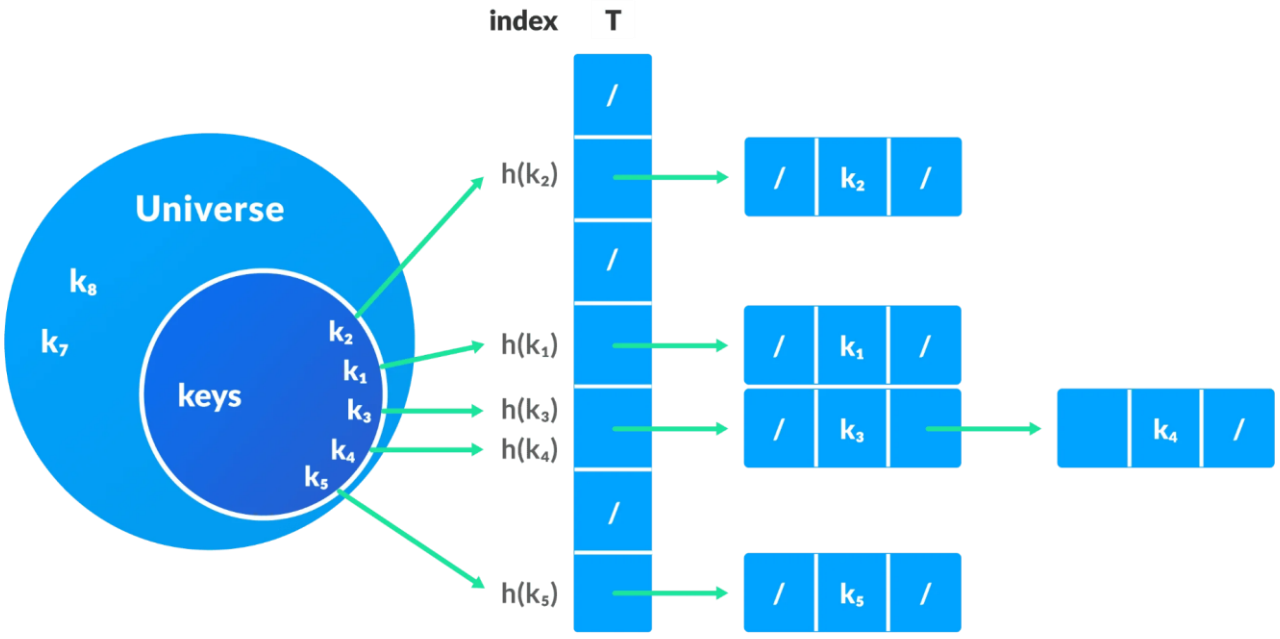
We can resolve the hash collision using one of the following techniques.

- Collision resolution by chaining
- Open Addressing: Linear/Quadratic Probing and Double Hashing

1. Collision resolution by chaining

In chaining, if a hash function produces the same index for multiple elements, these elements are stored in the same index by using a doubly linked list.

If j is the slot for multiple elements, it contains a pointer to the head of the list of elements. If no element is present, j contains NIL.



2. Open Addressing

Unlike chaining, open addressing doesn't store multiple elements into the same slot. Here, each slot is either filled with a single key or left NIL.

Different techniques used in open addressing are:

Linear Probing

In linear probing, collision is resolved by checking the next slot.

$h(k, i) = (h'(k) + i) \bmod m$

where

- i = {0, 1, ….}
- h'(k) is a new hash function.

If a collision occurs at h(k, 0), then h(k, 1) is checked. In this way, the value of i is incremented linearly.
The problem with linear probing is that a cluster of adjacent slots is filled. When inserting a new element,the entire cluster must be traversed. This adds to the time required to perform operations on the hash table.

Hash Table Implementation using Separate Chaining

```
// Implementation of hash table (Separate Chaining) in C++

#include <iostream>

#include <list>

using namespace std;

class HashTable

{

 int capacity;

 list<int> *table;
```

```
public:
 HashTable(int V);
 void insertItem(int data);
 void deleteItem(int key);
 void displayHash();
 void searchItem(int key);
 int hashFunction(int key)
 {
 return (key % capacity);
 }
 };
HashTable::HashTable(int c)
{
 this->capacity = c;
 table = new list<int>[capacity];
}
void HashTable::insertItem(int data)
{
 int index = hashFunction(data);
 table[index].push_back(data);
}
void HashTable::searchItem(int key){
    int index = hashFunction(key);
    list<int>::iterator i;
    int y=0;
    bool flag=false;
    for(i=table[index].begin(); i!=table[index].end(); i++){
```

```
        if(*i==key){
cout << "Record Found at["<< index << "]["<<y<<"]--->" << key<< endl;
            flag = true;
            break;
        }
        y++;
    }
    if(!flag)
    cout<<"Record Not Found"<<endl;
}
void HashTable::deleteItem(int key)
{
 int index = hashFunction(key);
 list<int>::iterator i;
 for (i = table[index].begin(); i != table[index].end(); i++)
 {
 if (*i == key)
   break;
 }
 if (i != table[index].end())
 table[index].erase(i);
}
void HashTable::displayHash()
{
 for (int i = 0; i < capacity; i++)
 {
 cout << "table[" << i << "]";
```

```
 for (int x : table[i])        // auto
   cout << " --> " << x;
 cout << endl;
 }
}
int main()
{
 int data[] = {1, 9, 23, 4, 5, 6,7};
 int size = sizeof(data) / sizeof(data[0]);
 HashTable h(size);
 for (int i = 0; i < size; i++)
 h.insertItem(data[i]);

 h.deleteItem(7);
 h.displayHash();
 h.searchItem(9);
}
```

Reference

https://www.programiz.com/dsa/hash-table

https://www.geeksforgeeks.org/hashing-data-structure/