|  |
| --- |
| **NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**  **CS 201–DATA STRUCTURES LAB**  **Lab Session 15** |
| **Instructors:** Mr. Faizan Yousuf, Ms. Safia, Ms. Maham Mobin |

**Introduction to Hashing**

Hash is a search in which the key (index) by an algorithm determine the location of data. Here we basically transform the key into the index that contains the data we are looking for. If more than one key locates to same index then we calls it synonyms. Therefore if actual data that we insert into list of data contains two or more synonyms then we will have collisions. A collision take place when a hashing algorithm produces an address for an insertion key and that address is already occupied. We use different algorithm to resolve collision and finds the next location. Each calculation of an address and test for success is known as a prob. Collision resolution have two techniques, one, Open addressing which includes double hashing (linear and quadrate probing), second by linked list.

### Objectives:

* What is Hash Function
* Collision
* Open Addressing
* Separate Chaining

**Searching in array**

Searching in array takes O(1) if index of the element is known, but if data is unsorted with 10000 elements and if index is not known then it becomes complex to search a particular element. To overcome this a data structure is used which provide very fast insertion and searching.

# Hash Function

It hashes (converts) a number in a large range into a number in a smaller range. This smaller range corresponds to the index numbers in an array.

smallerRange = largerRange % arraySize;

# Hash Table

A hash table (hash map) is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. It is an array into which data is inserted using a hash function.

We have several hashing methods:

**Direct Hashing:** In direct hashing, the key is the address without any algorithm implementation.

**Subtraction Hashing:** In subtraction, the key is transformed to an address by subtracting a fixed number from it.

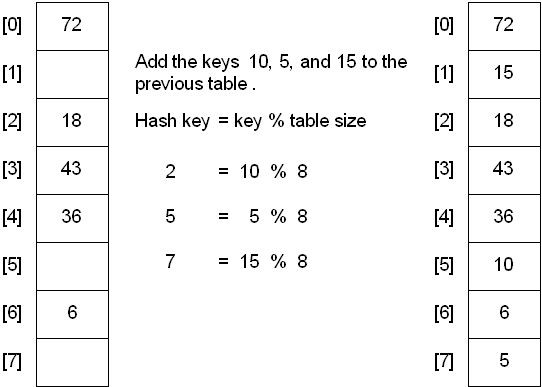
**Modulo-Division Hashing:** In modulo, the key is divided by the array size, recommended to be a prime number, and the number plus 1 is used as the address.

**Digit-Extraction Hashing:** In digit, selected digits are extracted from the key and used as an address.

**Midsqure Hashing:** In Midsquare, the key is squared and the address is selected from the middle of the result.

**Fold Shift Hashing:** In fold, the key is divided into parts whose sizes match the size of the required address. Then the parts are added to obtain the address.

**Rotation Hashing:** In rotation, the right most digit of the key is rotated to the left to determine an address. However, this method is usually used in combination with other methods.



Example: Given an array of size 10, we want to insert “ab” in index. Following steps will be done:

#### Apply hash function to “ab”

1. Since a stands at position 1 in English alphabets, so

a = 1 X 271

1. And b stands at position 2 in English alphabets, so

b = 2 X 270

1. Now we’ll add the result of a and b:

Value\_ab = a + b value\_ab = 29

1. Apply hash function

Index = value\_ab % size;

Index = 29 % 10;

Index = 2

#### Placement in hash table

The value ab will be placed at index 2.

# Collisions

Perhaps we want to insert the word melioration into the array. You hash the word to obtain its index number, but find that the cell at that number is already occupied by the word demystify, which happens to hash to the exact same number (for a certain size array). This situation is called a collision. Collision can be resolved using two techniques:

### 1. Open Addressing

Search the array in some systematic way for an empty cell, and insert the new item there, instead of at the index specified by the hash function.

There are three methods of open addressing

#### Linear Probing

The linear probing hash table is a fairly simple structure where data items are stored directly inside the hash element array.

If faced with a collision situation, the linear probing table will look onto to subsequent hash elements until the first free space is found.

This traversal is known as *probing* the table; and as it goes by one element at a time, it is *linear probing*.

#### Steps:

#### Calculate a hash code from the key

#### Access that hash element

#### If the hash element is empty, add straight away

#### If not, probe through subsequent elements (looping back if necessary), trying to find a free place

#### If a free places is found, add the data at that position

#### If no free place is found, add will fail.

#### Quadratic Probing

In the linear probing approach, once a cluster forms it tends to grow larger. Items that hash to any value in the range of the cluster will step along and insert themselves at the end of the cluster, thus making it even bigger. The bigger the cluster gets, the faster it grows.

Quadratic probing is an attempt to keep clusters from forming. The idea is to probe more widely separated cells, instead of those adjacent to the primary hash site.

#### Double Hashing

To eliminate secondary clustering, synonyms must have different probe sequences. Double hashing achieves this by having two hash functions that both depend on the hash key. here hp (or h2) is another hash function. The probing sequence is:

hi(key) = [h(key) + i\*hp(key)]% tableSize **for i = 1, . . . , tableSize – 1**

Common definitions for **hp** are :

hp(key) = 1 + key % (tableSize - 1)

hp(key) = q - (key % q) where **q** is a prime less than **tableSize**

hp(key) = q\*(key % q) where **q** is a prime less than **tableSize**

Performance of double hashing: Much better than linear or quadratic probing because it eliminates both primary and secondary clustering, but requires a computation of a second hash function **hp**.

# Separate Chaining

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. New key-value pairs are added to the end of the list. Lookup algorithm searches through the list to find matching key. Initially table slots contain nulls. List is being created, when value with the certain hash is added for the first time.

Steps:

* Calculate a hash code from the key
* Perform the operation on the list at that hash element

# ACTIVITY

* Implement all functions of separate chaining, double hashing and linear probing.