Study on Hashing Techniques

Md. Nazmul Khan

Student ID: 011163064

Suraiya Tasnim Ema

Student ID: 011132079

Samia Jahan

Student ID: 011132027

A thesis in the Department of Computer Science and Engineering presented

in partial fulfillment of the requirements for the Degree of

Bachelor of Science in Computer Science and Engineering

United International University

Dhaka, Bangladesh

May 2018

Declaration

We, Md. Nazmul Khan, Suraiya Tasnim Ema and Samia Jahan declare that this thesis "**Study on Hashing Techniques**", Thesis Title and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a BSc degree at United International University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at United International University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by ourselves jointly with others, we have made clear exactly what was done by others and what we have contributed ourselves.

_____

Md. Nazmul Khan

ID: 011163064

Department of CSE

_____

Suraiya Tasnim Ema

ID: 011132079

Department of CSE

_____

Samia Jahan

ID: 011132027

Department of CSE

Certificate

I do hereby declare that the research works embodied in this thesis entitled "**Study on Hashing Techniques**" is the outcome of an original work carried out by Md. Nazmul Khan, Suraiya Tasnim Ema and Samia Jahan under our supervision.

I further certify that the dissertation meets the requirements and the standard for the degree of BSc in Computer Science and Engineering.

_____

Dr. Mohammad Nurul Huda

Professor, Dept. of CSE
United International University

Abstract

The main aims of this thesis “Study on hashing techniques” are to get familiar with hash functions and hash background, knowing different hashing techniques and collision resolution techniques.

It will help you to find out the most efficient way to make hash function for a specific types of data.

Collision resolution techniques are also discussed in this paper. Load factor, advantage and disadvantage of different collision resolution techniques can be understood by reading this paper.

Time complexity of different searching techniques and compared those techniques with hashing techniques.

Proving why hashing is better than other searching techniques and be familiar with the objectives of hashing techniques.

Acknowledgement

We would first like to thank our thesis advisor Dr. Mohammad Nurul Huda, Professor at United International University. Without his help it was impossible for us to complete the research and writing paper. We are really grateful to our honorable teacher.

We would also like to acknowledge Ayesha Akter, Junior Officer of MSCSE Program at United International University as the second reader of this thesis and we are grateful to her for her very valuable comments on this thesis.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Searching on Data Structures

We have seen various data structures like linked list, Arrays, trees etc. Searching is a very frequent operation on any data structure. If we want to store data we have various types of data structures.

1.1.1 Array

We can store data in an array

2	4	11	43

1.1.2 Tree

We can again store it as a tree

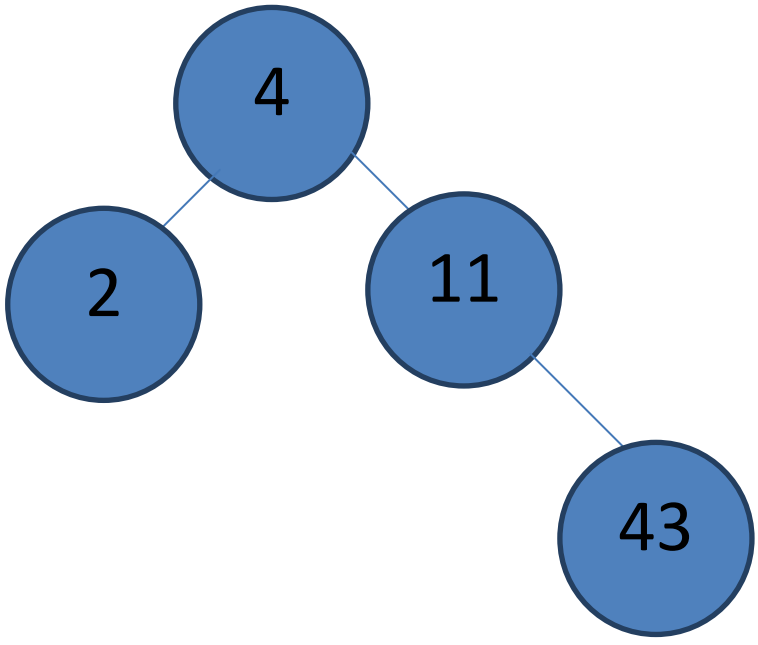

Figure 1: Tree

1.1.3 Linked list

We can also store it in linked list too

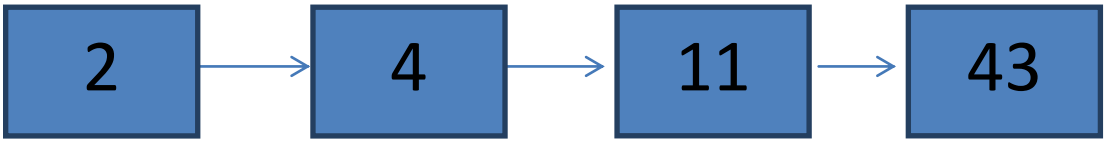

Figure 2: Linked list

1.2 Time Complexity

It actually depends on programmer how he will store the data. Searching is an operation that is used almost everywhere. If we want to search a value in data structure we need searching operation, even if we want to delete a value we also need to search. Since the use of searching operation is more, it needs to be more efficient. That means the time needed to search should be less as much as possible. Whatever data structure we choose, when we search we must consume some time on comparing elements of the item that we want to search .There are different types of time complexity on different types of searching algorithm. For example:

- Linked list- O(n)
- Array(unsorted)-O(n)
- Array(sorted)-O($\log_2 n$)
- Binary search tree-O($\log_2 n$)
- Hash Table-O(1)

Now, the data structure used in hashing is Hash Table. In this case we store data in hash table and the time complexity here to search data is O (1).That means its constant time to search a data. So hash table data structure is most efficient.

1.3 Hash table

Hash table is mainly an array .We can say it a custom array. Example of hash table:

```
{
   int id;
   char name[20];
   int age;}
```

Student s[3]

Id	Name	Age
id	Name	Age
Id	Name	Age

Figure 3: Hash Table

1.4 Hash Function

Hash function means function by which key value is converted into index. Hash function is necessary when-

- We need to insert data into hash table
- We need to search data into hash table

1.5 Hashing

Hashing is the technique in which large values are converted into small ones by hash functions, and then the values are stored in hash table.

Chapter 2

Background

2.1 Cryptographic Hash Functions

Cryptographic Hash Functions has designed at first in 1970, and then in 1980s more proposals have come out. In 1990s so many Hash Function grew up, but in some proposals security problem occurred. Markedly MD5 (Message Digest 5) and SHA-1 (Secure Hash Algorithm-1), at first it get noticed for using in cryptographic scheme with exact mentioned requirements of security, then for some developers and protocol designers it has become a standard expenses.

2.2 MD5 and SHA

Possibly Hash Functions concerned as black boxes. Collisions finding for MD5 become very easy by cryptanalysis done by Wang et al. SHA-1, which embodied the security edge. In the month of November, 2007, the NIST announced that it would arrange a competition which is SHA-3, with the goal that selects a so new Hash function. For the verification of source of data Hash Functions can be merged with cryptographic methods. Message Authentication Code is that in which Hashing algorithms and encryption are being merged, it produces exceptional message view that finds the source of data. In 1976, Diffie and Hellman identified that there is a need for a one-way Hash Function. In the late 1970s, in the work of Rabin [74], Markel [60] and Yuval [99] cryptographic Hash Functions can be found, in that work the analysis, definitions and constructions is done. Yuval showed that finding of collisions for an n-bit hash function in time of 2n/2, based on block cipher DES [37] with 64-bit result a design is proposed by Rabin, and the requirements of collision resistance , preimageresistance, second preimage resistance in done by Markel.

2.3 Impossible to Determine Input

During 1987, the definition of collision resistance is established by Damgard [26] and after couple of years Naor and Yung introduced Universal One Way Hash Function which is a alternative of second preimage resistance functions. During 2004 relations between collisions resistance, preimage resistance and second preimage resistance is studied by Rogaway and Shrimpton [79]. The algebraic structure can be destroyed by Hash Functions, such as Fiat-Shamir heuristic [36] and Coppersmith's attack. This development said that, it needs a requirement that hash functions act as 'ideal. To construct pseudo-random function hash function can be implemented; it is resulted in MAC algorithms construction which is based on the hash functions. Every hash functions has the characteristics is that only knowing the

output it is impossible to determine the input of the function. For example, if the output is 1, we cannot understand or have no information to determine that what can be the input of this function. Another example, a message has digest of ‘1’, main message have ‘112’ or any odd length input, so there is no clue to understand that what could be the original message. This characteristic make the hash function one-way hash function, means it is difficult, is not possible to determine input by the showing output. It’s quite tough to create collisions in the modern hash functions.

Chapter 3

Objectives

3.1 Message Digest

Hash function converts a value which is numerical into another compressed value. The input is an arbitrary length and output is a fixed length. Values are returned by hash function which is called ‘message digest’.

3.2 Message Integrity

Primary application in cryptography of hash function is message integrity. It provides digital fingerprint of the content of message and it just ensures that the message has not alerted by virus or other means.

3.3 Comparing Large Amount of Data

Hash function can be used for comparing large amount of data. We create hashes for key and store those. When we need to compare those data, then we just have to compare those hashes. For index data hashes can be used. It can be used in hash table for pointing the exact row. When we want to find a record more quickly, then by calculating the hash of data we can find where the record is pointing to.

3.4 Cryptographic applications

Cryptographic applications like digital signatures, where hash can be implement. For generating seemingly random strings hash can be used. Hash can be used for finding duplicate records, similar records, similar substrings, geometric hashing.

3.5 Quick Operation

Whenever we need fast key look up times, then we can use hash. Simply it has two components, those are keys and values. Transforming the key into hash is done by hash function. Here, the hash is the index of the data of the array. We just don’t have to traverse the entire array for finding the relevant value and it almost takes no time locate the value. When a hash function maps two different keys into the same table address, then a collision is said to be occurred. Hash function can handle this kind of collisions.

Chapter 4

Different Hashing Techniques

4.1 Hashing Techniques

To get a quick inserting, updating or deleting time we have many hashing techniques. Based on our data type, data size we can select our suitable techniques to keep our data. Simple to complex all types of calculation can be used in hashing techniques. But, it is preferable to keep calculation and steps simple. In hashing techniques we have key and one or more values with one key. Using the unique keys in hash function we get unique addresses and in those unique addresses we keep the values that were with the unique keys.

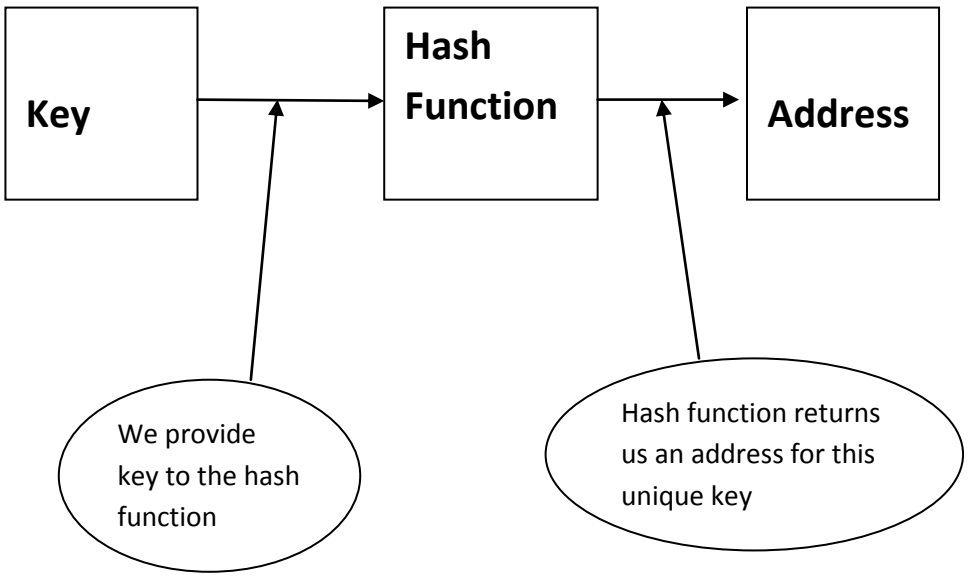

Figure 4: Hashing Technique

4.2 Types of Hashing Technique

Hashing techniques and their procedures are not fixed. All types of keys are not efficient for all types of techniques. For a set of keys one technique may be the best efficient but for the same set of keys other techniques may be not be the best efficient. In this book we will discuss 9 types of hashing techniques.

4.2.1 Extraction [1]

Extraction is the action of taking out something. In this technique we will take some bits from a large binary form so it is called Extraction.

4.2.1.1 Algorithm (Extraction)

```
Procedure Extraction(String word)
    Begin
        For  i = 0 to word.length()-1
            result = concatenate each character's 5 bit binary to result
        End For
        extract = concatenate last, 2nd last, 3rd bit from result  to extract
        index = decimal value of extract
        return index
    End
End Procedure
```

4.2.1.2 Example with flow chart (Extraction)

A=1, B=2, C=3...., Z=26.

5 bits Binary values are: A = 00001, B = 00010, C = 00011.... Z = 11010.

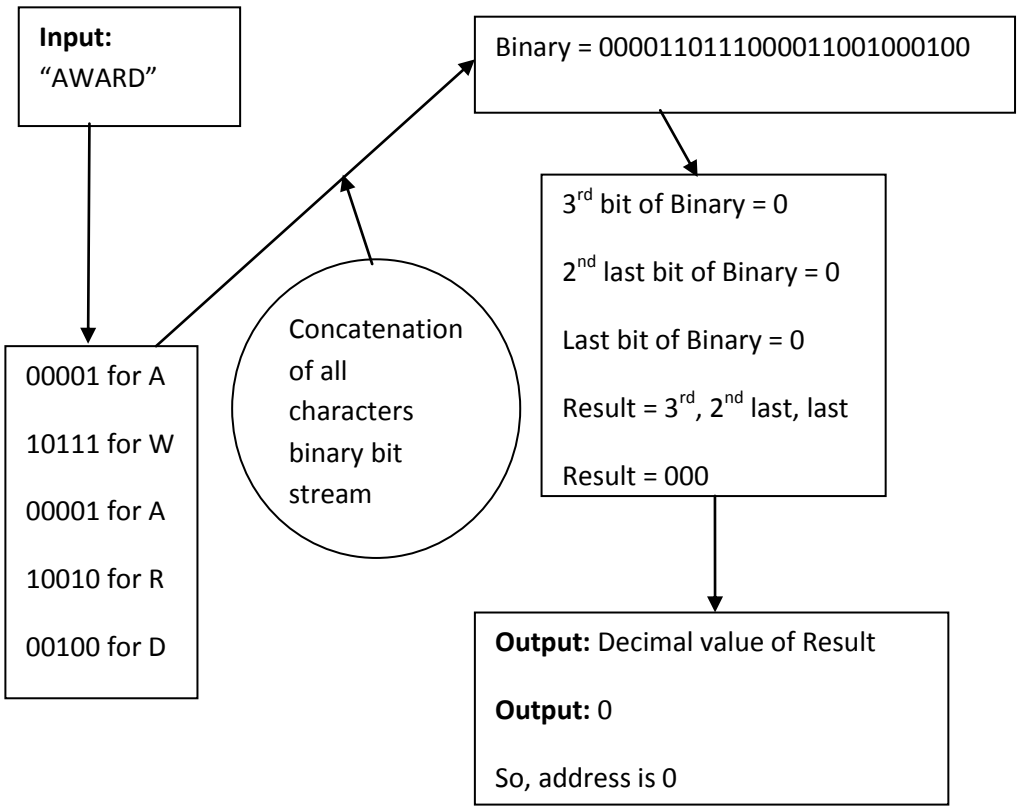

Figure 5: Extraction

4.2.2 Compression [1]

We compress a large index into small index. As we are compressing this is called Compression.

4.2.2.1 Algorithm (Compression)

```
Procedure Compression (String word)
    Begin
        For i = 0 to word.length()-1
            result = exclusive or operation between all character's 5 bits binary to result
        End For
        index = decimal value of result
        return index
    End
End Procedure
```

4.2.2.2 Example with flow chart (Compression)

A=1, B=2, C=3... Z=26.

5 bits Binary values are: A = 00001, B = 00010, C = 00011, ..., Z = 11010.

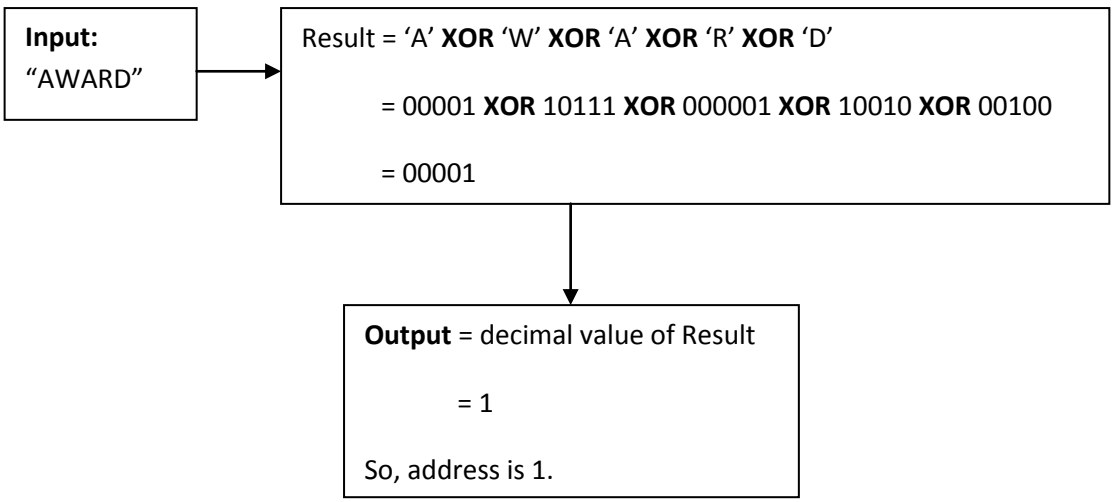

Figure 6: Compression

4.2.3 Radical

In radical we can get address as a decimal or octal or hexadecimal base.

4.2.3.1 Algorithm (Radical)

```
Procedure Radical (int num)
    Begin
        return num%10
    End
End Procedure
```

4.2.3.2 Example with flow chart (Radical)

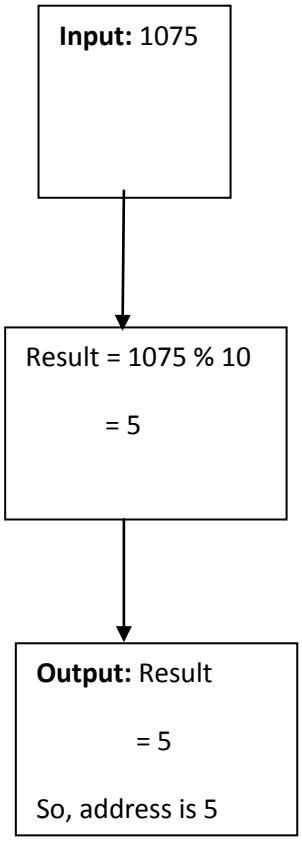

Figure 7: Radical

4.2.4 Midsquare [2]

In Midsquare method we square the index value and take the value from middle address, for this it is Midsquare method.

4.2.4.1 Algorithm (Midsquare)

Procedure Midsquare (String word)

Begin

For i = 0 to word.length()-1

result = exclusive or operation between all character's 5 bits binary to result

End For

index = decimal value of result

index = $(index)^2$

index = take the value of middle address from index
return index

End

End Procedure

4.2.4.2 Example with flow chart (Midsquare)

A=1, B=2, C=3...., Z=26.

5 bits Binary values are: A = 00001, B = 00010, C = 00011, ..., Z = 11010.

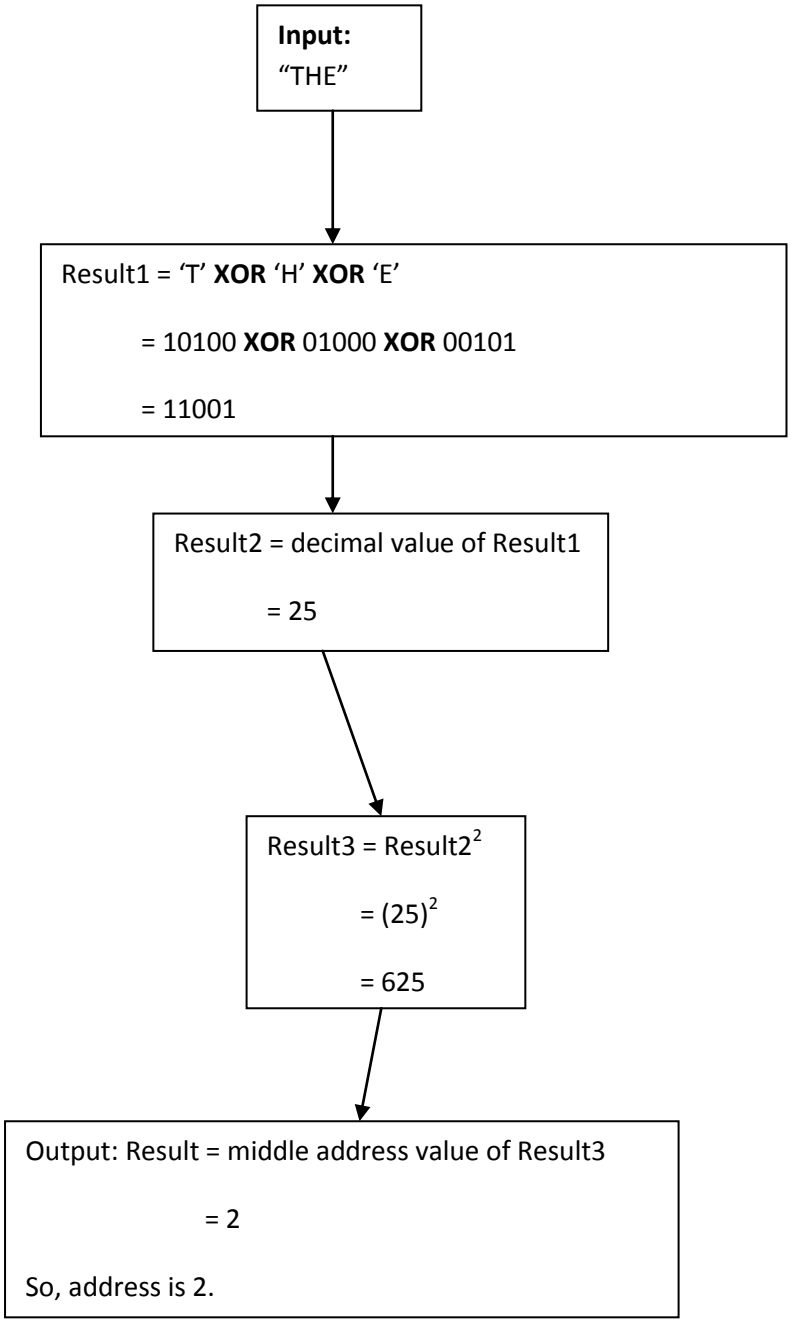

Figure 8: Mid square

4.2.5 Folding [2]

This method will fold the index number size and we will get a small number of indexes.

4.2.5.1 Algorithm (Folding)

Procedure Folding (String word)

Begin

For i = 0 to word.length()-1

result = exclusive or operation between all character's 5 bits binary to result

End For

index = (decimal value of result)^2

index = summation of all the digits from index and keep to index

return index

End

End Procedure

4.2.5.2 Example with flow chart (Folding)

A=1, B=2, C=3...., Z=26. 5 bits Binary values are: A = 00001, B = 00010, C = 00011, ..., Z = 11010

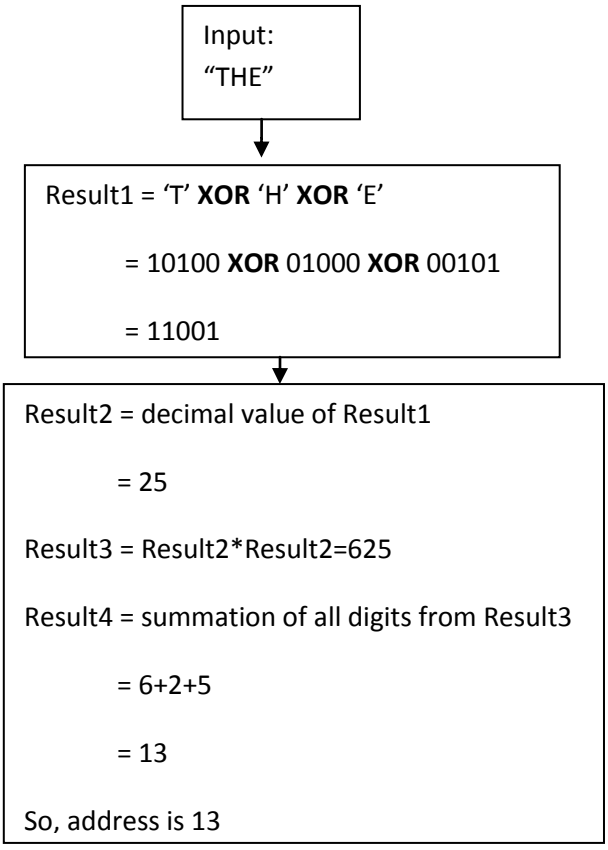

Figure 9: Folding

4.2.6 Radix Conversion

We are changing the index from one base to another (decimal to octal), so this method is called Radix Conversion.

4.2.6.1 Algorithm (Radix Conversion)

```
Procedure RadixConversion (String word)
    Begin
        For i = 0 to word.length()-1
            result = exclusive or operation between all character's 5 bits binary to result
        End For
        index = decimal value of result
        index = convert the index value from decimal to octal and keep in index
        return index
    End
End Procedure
```

4.2.6.2 Example with flow chart (Radix Conversion)

A=1, B=2, C=3...., Z=26.

5 bits Binary values are: A = 00001, B = 00010, C = 00011, ..., Z = 11010.

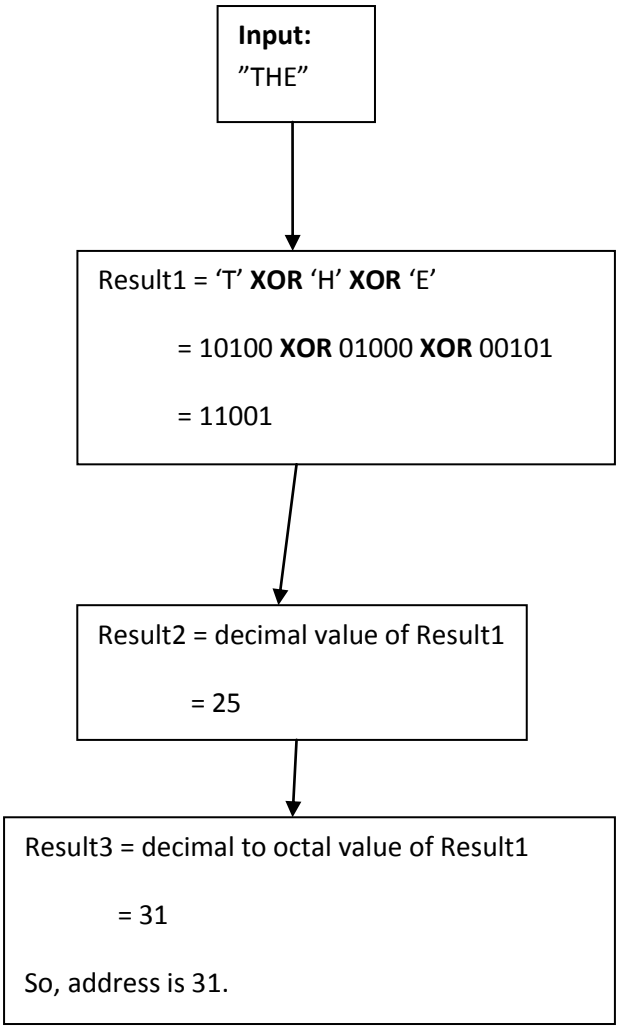

Figure 10: Radix Conversion

4.2.7 Database Hashing

This hashing is designed so that we can put multiple values in a unique address. It is not a hashing technique. But here we can learn how to put several data in a unique address.

4.2.7.1 Procedure (Database Hashing)

1. Take a linked list and linked list type will be a class type of several data
2. Initialize all the index of linked list
3. Now, we can put several list of data in a unique address

4.2.8 Multiplication [1]

In this technique we fix a constant with a range and then perform some multiplication calculation so that we can get our address.

4.2.8.1 Algorithm (Multiplication)

```
Procedure Multiplication (String word)
    Begin
        For i = 0 to word.length()-1
            result = concatenate each character's 5 bit binary to result
        End for
        comp = decimal value of result
        frac = 0.6125423471 * comp
        frac = frac % int value of frac
        index = 1+(30 * frac)
        index = int value of index
        return index
    End
End Procedure
```

4.2.8.2 Example with flow chart (Multiplication)

A=1, B=2, C=3...., Z=26.

5 bits Binary values are: A = 00001, B = 00010, C = 00011.....Z = 11010.

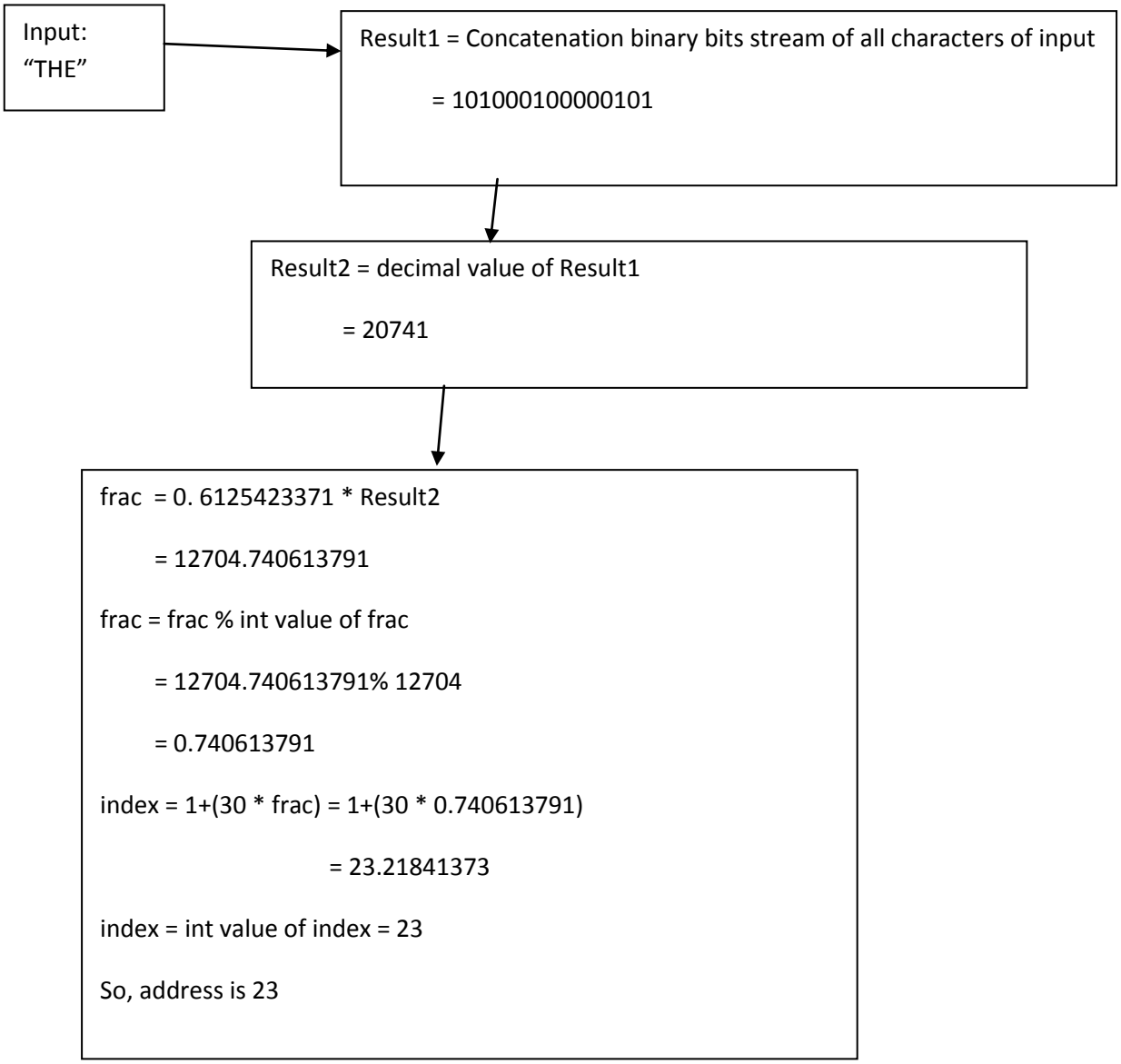

Figure 11: Multiplication

4.2.9 Division [1]

It is most using technique. In this technique, we divide the index with the size and take the remainder so that our index value does not be greater than the size.

4.2.9.1 Algorithm (Division)

```
Procedure Division (String word)
    Begin
        For  i = 0 to word.length()-1
            result = concatenate each character's 5 bit binary to result
        End For
        index = decimal value of result
        return index % 31
    End
End Procedure
```

4.2.9.2 Example with flow chart (Division)

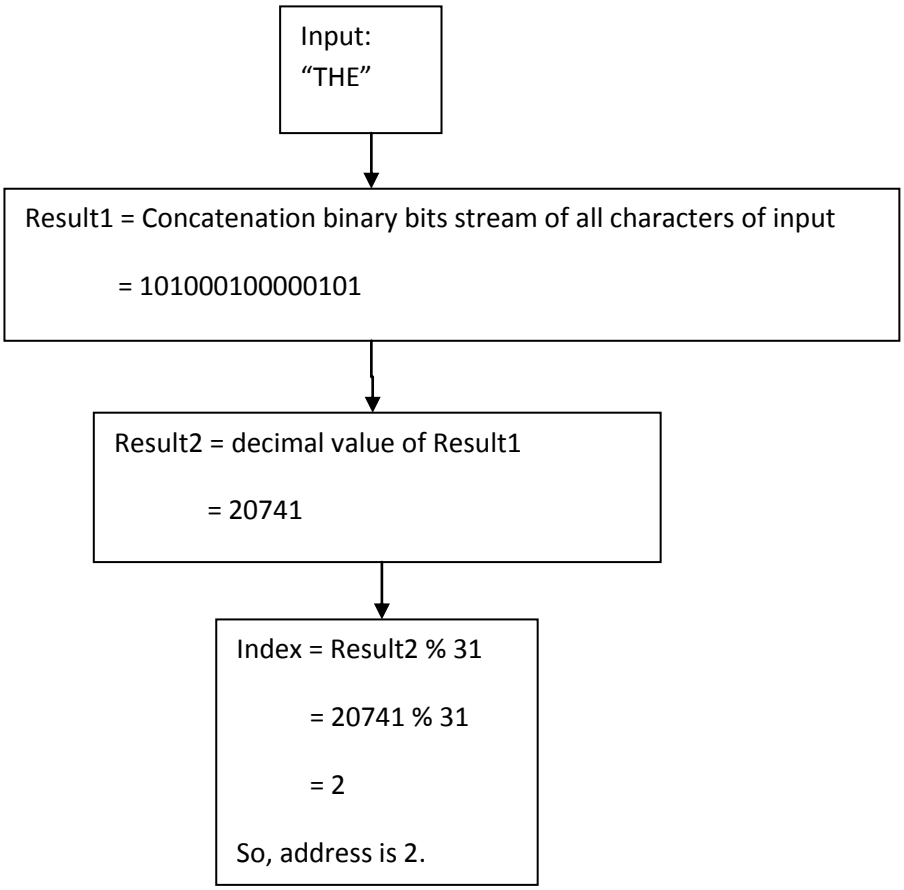

Figure 12: Division

Chapter 5

Collision Resolution

5.1 Collision

After providing a key to a hash function, we will get an address for each key. Sometimes, we get the same address for two or more keys and that is called collision.

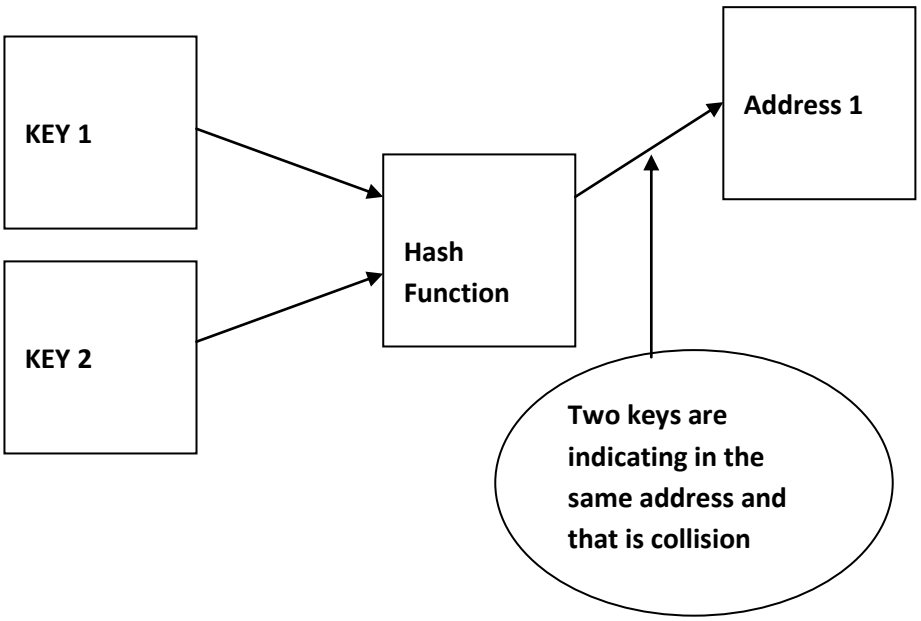

Figure 13: Collision

Example: Two keys are K1 = 5001 and K2 = 301

For key K1 address is A1

A1 = K1 mod 10

= 5001 mod 10

= 1

For key K2 address is A2

A2 = K2 mod 10

= 301 mod 10

= 1

Here, for keys K1 and K2, we got the same address. So, that is a collision.

5.2 Collision Resolution

To solve the collision problem we have some solutions, so that we get only one address for only one key. These solutions are called collision resolution.

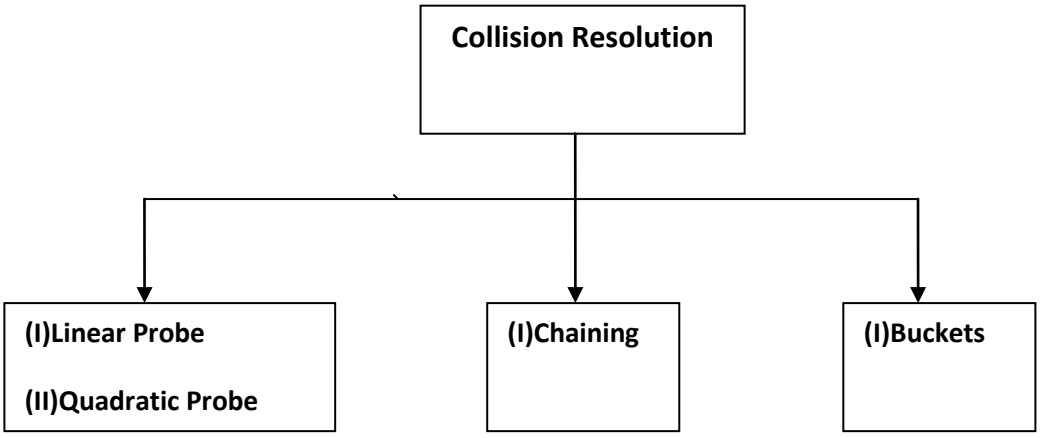

Figure 14: Different types of Collision

5.2.1.1 Linear Probe [3]

This process will be completed by adding constant ‘a’ to current address. a=1, 2, 3, 4...

Suppose, we have 4 key values K1 = 5001, K2 = 301, K3 = 402, K4 = 101.From those 4 keys we get 4 addresses.

A1= 5001 mod 10 or, A1 = 1

A2= 301 mod 10 or, A2 = 1

A3= 402 mod 10 or, A3 = 2

A4= 101 mod 10 or, A4 = 1

A1 = 1, index 1 is empty. So, we can keep K1 in 1.K1 = 5001

A2 = 1, now index 1 is not empty. So, increment index by 1.

Then, index will be 2. Index 2 is empty. So, we can keep K2 in 2. K2 = 301

A3 = 2, index 2 is not empty. So, increment index by 1.

Then, index will be 3. Index 3 is empty. So, we can keep K3 in index 3. K3 = 402

A4 = 1, index 1 is not empty. So, increment index by 1.

Then, index will be 2. Index 2 is also not empty.

So, increasing index by 1.After that, index is 3. Index 3 is again not empty. Index 3 is the bottom index. So, it will go to the beginning index which is address 0. Index 0 is empty. Finally we can keep K4 in index 0.K4=101

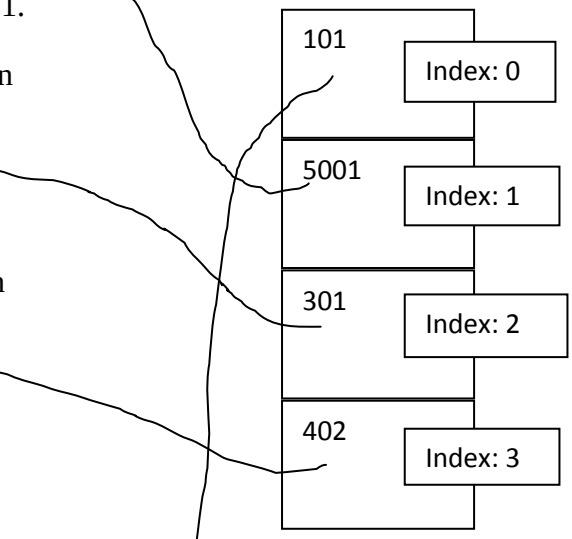

Figure 15: Linear Probe

5.2.1.2 Quadratic Probe [3]

Quadratic probe is completed by adding constant 'a+a^2' to current address. a= 1,2,3,4,.....

$(1+1^2)$ mod SIZE

$(2+2^2)$ mod SIZE

$(3+3^2)$ mod SIZE

$(4+4^2)$ mod SIZE

5.2.1.3 Algorithm (Probe)

Making hash table:

```
Procedure Hashing(int tablesize)
    Begin
      size = tablesize
      hashData = new String[tablesize]
    End
End Procedure
```

Putting via Linear Probe:

```
Procedure linearProbe (int key, String value)
  Begin
    index=hashFunction(key)
    While(hashData[index]!=null)
        index+=1
        if(index==size)
            index=0
        End if
    End While
    hashData[index]=value
  End
End Procedure
```

Putting via Quadratic Probe:

```
Procedure QuadraticProbe(int key, String value)
  Begin
      Index=hashFunction(key)
      Index=(index+index^2)%size
      While(hashData[index]!=null)
          index+=1
          if(index==size)
              index=0
          End if
      End While
      hashData[index]=value
  End
End Procedure
```

Showing Data:

```
Procedure showData()
  Begin
    For i=0 to hashData.length-1
        println(hashData[i])
    End For
  End
End Procedure
```

5.2.1.4 Limitation of Probing

Positive side is all the values are in one array but the table may be plenary so that it is infeasible to insert any new record.

5.2.1.5 Chaining [4]

In chaining collision resolution each index of hash table points to a linked list of records.

Assume that, we have 7 key values K1 = 5001, K2 = 301, K3 = 402, K4 = 101, K5 = 306, K6 = 405 and K7 = 305. From those 7 keys we get 7 addresses.

A1= 5001 mod 10 or, A1 = 1 (index 1, first node)

A2= 301 mod 10 or, A2 = 1 (index 1, second node)

A3= 402 mod 10 or, A3 = 2 (index 2, first node)

A4= 101 mod 10 or, A4 = 1 (index 1, third node)

A5= 306 mod 10 or, A1 = 6 (index 6, first node)

A6= 405 mod 10 or, A2 = 5 (index 5, first node)

A7= 304 mod 10 or, A3 = 4 (index 4, first node)

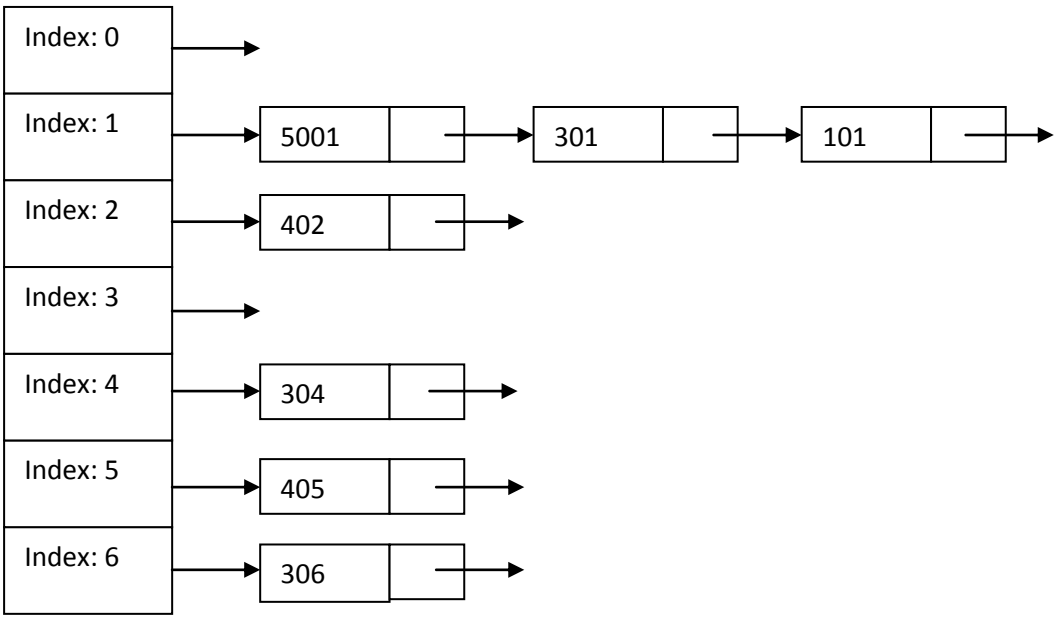

Figure 16: Chaining

5.2.1.6 Algorithm (Chaining):

Making hash table:

```
Procedure HashTableArray (int size)
    Begin
        arrayHash=new LinkedList[size]
        For i=0 to size-1
            arrayHash[i]=new LinkedList()
        End For
    End
End Procedure
```

Putting data:

```
Procedure Put (String word)
    Begin
        Index=hashFunction(word)
        arrayHash[index].add(word)
    End
End Procedure
```

Showing Data:

```
Procedure displayTheTable()
    Begin
        For i=0 to arrayHash.length-1
            print(i);
            for j=0 to arrayHash[i].size()-1
                print(arrayHash[i].get(j))
            End for
            println()
        End For
    End
End Procedure
```

5.2.1.7 Load factor for chaining example in (4.2.3) [3]

The ratio of total number of keys (n) and number of hash addresses generated by hash function (m) is called load factor (λ).

Load factor, λ = n/m

= number of keys (K1,K2,K3,K4,K5,K6,K7) / number of hash addresses generated by hash function(1,2,4,5,6)

= 7/5

= 1.4

If load factor is high, hash function will be less effective.

If load factor is closed to 1, hash function will be more effective

Complexity is O(1+λ)

5.2.1.8 Advantage and Disadvantage of chaining

Advantages	Disadvantages
1. Implementation is easy	1. Data are put in linked list
2. No overflow occurs. We can put more data to chain	2. Memory wastage
3. Not much sensitive with load factor	3. If the chain is too long, then searching a data is O(n)
4. For unknown number of data we can use it	4.We need extra space for links

5.2.1.9 Buckets

In bucket hashing, we can take each index as a bucket. An in every buckets we can put several data.

Assume that, we have 7 key values K1 = 5001, K2 = 301, K3 = 402, K4 = 101, K5 = 306, K6 = 405 and K7 = 305. From those 7 keys we get 7 addresses.

A1= 5001 mod 10 or, A1 = 1 (Bucket 1, index: 0)

A2= 301 mod 10 or, A2 = 1 (Bucket 1, index: 1)

A3= 402 mod 10 or, A3 = 2 (Bucket 2, index: 0)

A4= 101 mod 10 or, A4 = 1 (Bucket 1, index: 2)

A5= 306 mod 10 or, A1 = 6 (Bucket 6, index: 0)

A6= 405 mod 10 or, A2 = 5 (Bucket 5, index: 0)

A7= 304 mod 10 or, A3 = 4 (Bucket 4, index: 0)

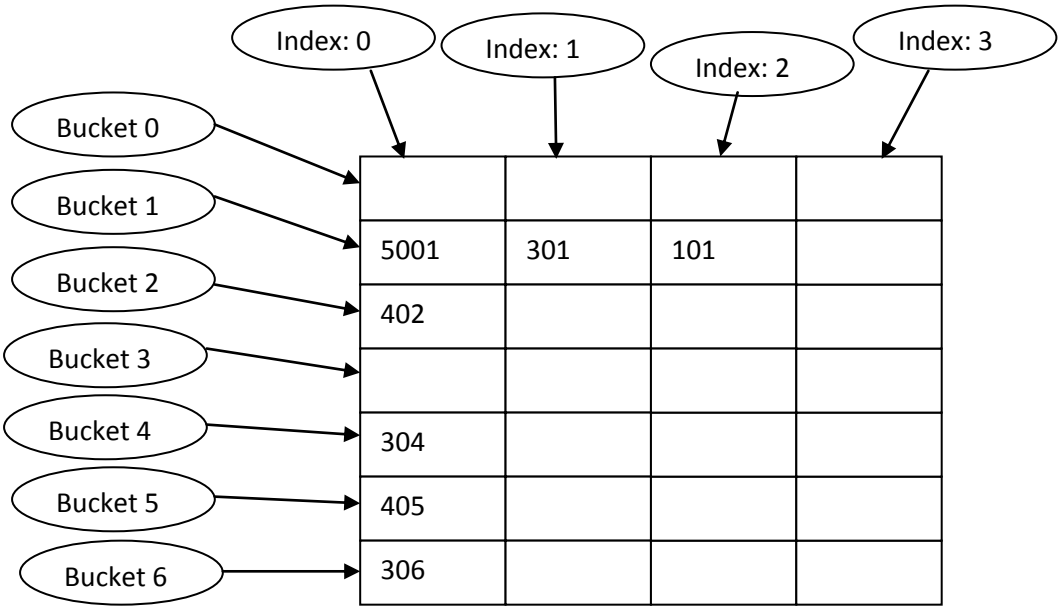

Figure 17: Bucket hashing

5.2.1.10 Algorithm (Buckets)

Making the Buckets:

```
Procedure bucketHashing (int size)
    Begin
        y=0      //Bucket index
        dataInBucket = new String[size][size]
    End
End Procedure
```

Putting data in Buckets:

```
Procedure putDataInBucket(int key, int value)
    Begin
        index=hashFunction(key)
        While(dataInBucket[index][y]!=null)
                y+=1
        End While
        dataInBucket[index][y]=value
    End
End Procedure
```

Showing the Buckets:

```
Procedure showBuckets()
    Begin
        For i=0 to size-1
                print(i)
                j=0
                While(dataInBucket[i][j]!=null)
                        print(dataInBucket[i][j])
                End While
                println()
        End For
    End
End Procedure
```

5.2.1.11 Bucket Overflow

If the index number is filled by data then, we cannot put more data in the filled bucket. That is a problem in bucket hashing and it is called bucket overflow.

We can solve the problem if we initially increase the size of index so that index is not filled by data.

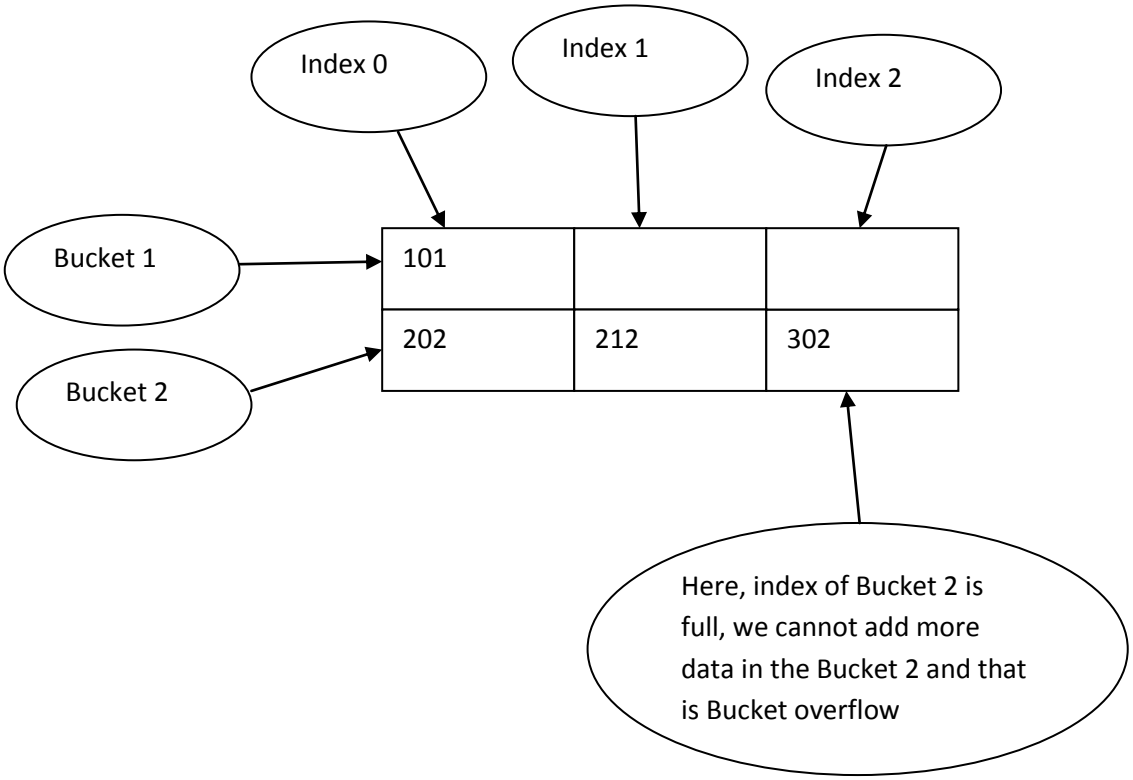

Figure 18: Bucket Overflow

Experimental Results

Here used only 16 words to show clearly the process how to determine most efficient hashing techniques for a specific type of data.

The more data we will take the more accurate result will be found

Using 16 words (keys): THE, OF, AND, TO, A, IN, THAT, IS, FROM, THEY, ONE, DRY, TEA, GLUM, WE, AWARD

Method Name	No. Of keys (n)	No. Of address (m)	Load factor (λ=n/m)
Extraction	16	8	2
Compression	16	14	1.143
Multiplication	16	13	1.231
Division	16	13	1.231
Folding	16	10	1.6
Midsquare	16	8	2
Radix conversion	16	14	1.143

Chapter 6

Conclusion & Future Work

6.1 Conclusion

Everyday almost quintillion bytes of data are generated because of business transactions, social networking and many others. Therefore sorting, processing and analyzing of these voluminous data is vital. Hashing is one the most effective techniques for data compression, processing and analysis. We have studied and analysis on different types of hashing techniques and have tried to find out the most effective techniques. According to our experiment we can see [5.2.1.12] that load factor of Extraction and Midsquare method is 2, Compression and Radix conversion is 1.143, Multiplication and division is 1.231 and Folding is 1.6 .Compression and Radix Conversion method work better since they have the lowest load factor which is 1.143 whereas Extraction and Midsquare method is not so efficient since they have the highest load factors which is 2.

6.2 Future Work

So, In future we will try to work with those inefficient functions to improve their efficiency. We will also implement other Hash functions and will experiment with more datasets. We will try to find out more efficient hash functions and better ways to reduce collisions.

References

[1] 7.4.1 Hash Functions, Data Structuresby Edward M. Reingold (Author), Wilfred J. Hansen (Author)

[2] https://www.tutorialspoint.com/data_structure/hashing.asp

[3]https://www.tutorialspoint.com/data_structure/hashing_collision_and_collision_resolution.asp

[4] https://www.tutorialspoint.com/data_structure/hashing_chaining.asp

[5] Hash functions: Theory, attacks, and applications IlyaMironov Microsoft Research, Silicon Valley Campus mironov@microsoft.com November 14, 2005

[6] The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition Bart PreneelKatholiekeUniversiteit Leuven and IBBT Dept. Electrical Engineering-ESAT/COSIC, KasteelparkArenberg 10 Bus 2446, B-3001 Leuven, Belgium bart.preneel@esat.kuleuven.be

[7] John Edward Silva January 15, 2003 GIAC Security Essentials Practical Version 1.4b Option 1 An Overview of Cryptographic Hash Functions and Their Uses

Appendix

HashTableArrayDemo.java

```
packagehashtablearraydemo;

importjava.io.File;

importjava.io.FileNotFoundException;

importjava.util.Scanner;


public class HashTableArrayDemo {

public static void main(String[] args) throws FileNotFoundException {

        File file = new File("input.txt");

        File intFile =new File("intInput.txt");

        File databaseFile=new File("databaseinput.txt");

        File OpenAddressingInput=new File("OpenAddressingInput.txt");

        File bucketFile=new File("OpenAddressingInput.txt");


BucketHashing n1=new BucketHashing(5);//BucketHashing

        Hashing a1=new Hashing(5);//LinearProbing

        Hashing a2=new Hashing(5);//QuadraticProbing

HashTableArray hash1= new HashTableArray(8);//extraction ,chaining

HashTableArray hash2=new HashTableArray(32);//compression ,chaining

HashTableArray hash3=new HashTableArray(32);//division ,chaining

HashTableArray hash4=new HashTableArray(32);//multiplication ,chaining

HashTableArray hash5=new HashTableArray(10);//midsquare ,chaining

HashTableArray hash6=new HashTableArray(28);//folding ,chaining

HashTableArray hash7=new HashTableArray(38);//radixConversion ,chaining

HashTableArray hash8=new HashTableArray(10);//radical ,chaining

HashTableArrayDatabase hash9=new HashTableArrayDatabase(15);//databaseHashing ,chaining

        Scanner scan1 = new Scanner(file);

while(scan1.hasNext()){

hash1.putExtraction(scan1.next());

        }
```

```
        Scanner scan2 = new Scanner(file);
while(scan2.hasNext()){
hash2.putCompression(scan2.next());
        }
        Scanner scan3 = new Scanner(file);
while(scan3.hasNext()){
hash3.putDivisionCompression(scan3.next());
        }
        Scanner scan4 = new Scanner(file);
while(scan4.hasNext()){
hash4.putMultiplicationCompression(scan4.next());
        }
        Scanner scan5 = new Scanner(file);
while(scan5.hasNext()){
hash5.putMidsquare(scan5.next());
        }
        Scanner scan6 = new Scanner(file);
while(scan6.hasNext()){
hash6.putFolding(scan6.next());
        }
        Scanner scan7 = new Scanner(file);
while(scan7.hasNext()){
hash7.putRadixConversion(scan7.next());
        }
        Scanner scan8 = new Scanner(intFile);
while(scan8.hasNext()){
hash8.putRadical(scan8.nextInt());
        }
        Scanner scan9 = new Scanner(databaseFile);
int a=-1,k=0;
        String b="",c="",d="";
```

```
while(scan9.hasNext()){
if(k==4){
                k=0;
            }
if(k==0){
                a=scan9.nextInt();
                k+=1;
continue;
            }
if(k==1){
                b=scan9.next();
                k+=1;
continue;
            }
if(k==2){
                c=scan9.next();
                k+=1;
continue;
            }
if(k==3){
                d=scan9.next();
                k+=1;
            }
hash9.putDataHash(a, b, c, d);
        }
        Scanner scan10 = new Scanner(OpenAddressingInput);
while(scan10.hasNext()){
int key=scan10.nextInt();
            String value=scan10.next();
a1.putDataLinearProbing(key, value);
        }
```

```
 Scanner scan11 = new Scanner(OpenAddressingInput);
while(scan11.hasNext()){
int key=scan11.nextInt();
          String value=scan11.next();
a2.putDataQuadraticProbing(key, value);
      }
      Scanner scan12 = new Scanner(bucketFile);
while(scan12.hasNext()){
int key=scan12.nextInt();
          String value=scan12.next();
n1.putDataInBucket(key, value);
      }

System.out.println("Hash data in Buckets");
n1.showBuckets();
System.out.println("Hash data using Linear Probing");
a1.showData();
System.out.println("Hash data using Quadratic Probing");
a2.showData();

hash1.deleteExtraction("OF");
System.out.println("Hash table using Extraction chaining");
hash1.displayTheTable();
System.out.println(hash1.getExtraction("THE"));
hash1.searchExtraction("THE");
System.out.println();

hash2.deleteCompression("THE");
System.out.println("Hash table using Compression chaining");
hash2.displayTheTable();
System.out.println(hash2.getCompression("THE"));
hash2.searchCompression("OF");
```

```
System.out.println();


hash3.deleteDivisionCompression("THE");
System.out.println("Hash table using DivisionCompression chaining");
hash3.displayTheTable();
System.out.println(hash3.getDivisionCompression("IN"));
hash3.searchDivisionCompression("ONE");
System.out.println();


hash4.deleteMultiplicationCompression("IN");
System.out.println("Hash table using MultiplicationCompression chaining");
hash4.displayTheTable();
System.out.println(hash4.getMultiplicationCompression("THE"));
hash4.searchMultiplicationCompression("THEE");
System.out.println();


hash5.deleteMidSquare("IN");
System.out.println("Hash table using MidSquare chaining");
hash5.displayTheTable();
System.out.println(hash5.getMidSquare("THE"));
hash5.searchMidSquare("THEE");
System.out.println();


hash6.deleteFolding("IN");
System.out.println("Hash table using Folding chaining");
hash6.displayTheTable();
System.out.println(hash6.getMidSquare("THE"));
hash6.searchMidSquare("THEE");
System.out.println();


hash7.deleteRadixConversion("IN");
System.out.println("Hash table using RadixConversion chaining");
```

```
hash7.displayTheTable();
System.out.println(hash7.getRadixConversion("THE"));
hash7.searchRadixConversion("THEE");
System.out.println();

hash8.deleteRadical(3);
System.out.println("Hash table using Radical chaining");
hash8.displayTheTable();
System.out.println(hash8.getRadical(101));
hash8.searchRadical(102);
System.out.println();

hash9.deleteData(15);
System.out.println("Hash table file of Database chaining");
hash9.displayData();
hash9.updateAddress(16, "London");
hash9.updateName(2, "Yasin");
hash9.displayData();
hash9.searchDatabase(10, "Habib");
    }
}
```

BucketHashing.java

```
packagehashtablearraydemo;


public class BucketHashing {

int size;

int y=0;

String[][] dataInBucket;


publicBucketHashing(int size){

this.size=size;


dataInBucket = new String[size][size];

    }

inthashFunction(int key){

returnkey%size;

    }


public void putData(intindex,String value){



while(dataInBucket[index][y]!=null){

                y+=1;

           }

dataInBucket[index][y]=value;


    }

public void putDataInBucket(intkey,String value){

int index=hashFunction(key);

putData(index,value);

    }


public void showBuckets(){
```

```
for(int i=0;i<size;i++){
System.out.print("Bucket"+i+": ");
int j=0;
while(dataInBucket[i][j]!=null){
if(j==0){
System.out.print(dataInBucket[i][j]);
                j+=1;
            }
else{
System.out.print(","+dataInBucket[i][j]);
                j+=1;


            }


        }
System.out.println();


    }


  }
}
```

DatabaseHashing.java

```
packagehashtablearraydemo;


public class DatabaseHashing {

privateint id;

private String name;

private String dept;

private String address;


publicDatabaseHashing(intid,Stringname,Stringdept,String address){

        this.id=id;

        this.name=name;

this.dept=dept;

this.address=address;

    }


publicintgetId() {

return id;

    }


public void setId(int id) {

        this.id = id;

    }


public String getName() {

return name;

    }


public void setName(String name) {

        this.name = name;

    }
```

```
public String getDept() {
returndept;
    }

public void setDept(String dept) {
this.dept = dept;
    }

public String getAddress() {
return address;
    }

public void setAddress(String address) {
this.address = address;
    }
}
```

HashTableArray.java

```
packagehashtablearraydemo;

importjava.util.LinkedList;

public class HashTableArray{


LinkedList[] arrayHash;


publicHashTableArray(int Size){

arrayHash = new LinkedList[Size];

for(int i=0;i<arrayHash.length;i++){

arrayHash[i]=new LinkedList();

        }


    }


    //Hash function radical

int radical(intnum){

return num%10;

    }

    //Hash function extraction

int extraction(String word) {


        String result="";

        String extract="";

for(int i=0;i<word.length();i++){

int a=word.charAt(i)-'A'+1;//1 for A,2 for B.....

            String binary= Integer.toBinaryString(a);//1 for 1,10 for 2,11 for 3

            String fivezeroes = "00000";

intlen=binary.length();

if(len<5){

binary=fivezeroes.substring(0, 5-len).concat(binary);//making 5 bits:::: 00001 for 1 ， 00010 for 10 ， 00011 for 11

            }
```

```
result=result.concat(binary);//Concating each character's 5 bits binary to result;
        }

extract=result.charAt(result.length()-1)+"".concat(extract);//taking last bit
extract=result.charAt(result.length()-2)+"".concat(extract);//taking second last bit
extract=result.charAt(2)+"".concat(extract);//taking 3rd bit

int index= Integer.parseInt(extract, 2);//binary string to decimal value of extract

return index;

    }
    //Hash Function compression
int compression(String word){

int result=0;
int temp;
for(int i=0;i<word.length();i++){
int a=word.charAt(i)-'A'+1;//1 for A,2 for B.....
            String binary= Integer.toBinaryString(a);//converting integer to binary string: 10 for 2
            String fivezeroes = "00000";
intlen=binary.length();
if(len<5){
binary=fivezeroes.substring(0, 5-len).concat(binary);//00001 for 1,00010  for 2,making 5 bits
            }
temp=Integer.parseInt(binary,2);
if(i==0){
result=Integer.parseInt(binary,2);
}else{
result=result^temp;//Exclusive-0r operation
            }
```

```
        }
return result;
    }

    //Hash Function Division
intdivisionCompression(String word){

        String result="";
for(int i=0;i<word.length();i++){
int a=word.charAt(i)-'A'+1;//1 for A,2 for B.....
            String binary= Integer.toBinaryString(a);//converting integer to binary string: 01 for 2
            String fivezeroes = "00000";
intlen=binary.length();
if(len<5){
binary=fivezeroes.substring(0, 5-len).concat(binary);//00001 for 1,00010  for 10,making 5 bits
            }
result=result.concat(binary);//Concating each character's 5 bits binary to result;
        }
int index=Integer.parseInt(result, 2)%31;//binary result to decimal value,after that mod by 31

return index;
    }

    //Hash Function Multiplication
intmultiplicationCompression(String word){
        String result="";
for(int i=0;i<word.length();i++){
int a=word.charAt(i)-'A'+1;//1 for A,2 for B.....
            String binary= Integer.toBinaryString(a);//converting integer to binary string: 01 for 2
            String fivezeroes = "00000";
intlen=binary.length();
if(len<5){
```

```
binary=fivezeroes.substring(0, 5-len).concat(binary);//00001 for 1,00010  for 10,making 5 bits
            }
result=result.concat(binary);//Concating each character's 5 bits binary to result;
        }
int comp=Integer.parseInt(result, 2);//binary result to decimal value
doublefrac=0.6125423371*comp;
frac=frac%(int)frac; //taking fractional part from frac;
double index = 1+(30*frac);
return (int)index;
    }

    //Hash Function MidSquare
intmidSquare(String word){

int result=0;
int temp;
for(int i=0;i<word.length();i++){
int a=word.charAt(i)-'A'+1;//1 for A,2 for B.....
            String binary= Integer.toBinaryString(a);//converting integer to binary string: 01 for 2
            String fivezeroes = "00000";
intlen=binary.length();
if(len<5){
binary=fivezeroes.substring(0, 5-len).concat(binary);//00001 for 1,00010  for 2,making 5 bits
            }
temp=Integer.parseInt(binary,2);
if(i==0){
result=Integer.parseInt(binary,2);
}else{
result=result^temp;//Exclusive-0r operation
            }

        }
```

```
int index=result*result;//squaring the result and keeping to index


int[] a= new int[3];


for(int k=a.length-1;k>=0;k--){
if(index==0){
a[k]=0;
        }
else{
a[k]=index%10;
index=index/10;


        }


      }


return a[a.length/2];
   }


   //Hash Function Folding
publicint folding(String word){


int result=0;
int temp;
for(int i=0;i<word.length();i++){
int a=word.charAt(i)-'A'+1;//1 for A,2 for B.....
        String binary= Integer.toBinaryString(a);//converting integer to binary string: 01 for 2
        String fivezeroes = "00000";
intlen=binary.length();
if(len<5){
binary=fivezeroes.substring(0, 5-len).concat(binary);//00001 for 1,00010  for 2,making 5 bits
        }
```

```
temp=Integer.parseInt(binary,2);
if(i==0){
result=Integer.parseInt(binary,2);
}else{
result=result^temp;//Exclusive-0r operation
            }

        }
int index=result*result;//squaring the result and keeping to index

int[] a= new int[3];

for(int k=a.length-1;k>=0;k--){
if(index==0){
a[k]=0;
            }
else{
a[k]=index%10;
index=index/10;

            }

        }

int sum=0;
for(int k=0;k<a.length;k++){
sum=sum+a[k];
        }

return sum;
    }
```

```
   //Hash Function radixConversion


publicintradixConversion(String word){


int result=0;

int temp;

for(int i=0;i<word.length();i++){

int a=word.charAt(i)-'A'+1;//1 for A,2 for B.....

          String binary= Integer.toBinaryString(a);//converting integer to binary string: 01 for 2

          String fivezeroes = "00000";

intlen=binary.length();

if(len<5){

binary=fivezeroes.substring(0, 5-len).concat(binary);//00001 for 1,00010  for 2,making 5 bits

          }

temp=Integer.parseInt(binary,2);

if(i==0){

result=Integer.parseInt(binary,2);

}else{

result=result^temp;//Exclusive-0r operation

          }


       }

       String a=Integer.toOctalString(result);

int b=Integer.parseInt(a);//converted to octal integer


return b;


   }


public void put(String word,int index){

arrayHash[index].add(word);
```

```
    }

public void putRadical(int word){

int index=radical(word);
        String v=word+"";
put(v,index);
    }

public void putExtraction(String word){
int index=extraction(word);
put(word,index);
    }

public void putCompression(String word){
int index=compression(word);
put(word,index);
    }
public void putDivisionCompression(String word){
int index=divisionCompression(word);
put(word,index);
    }
public void putMultiplicationCompression(String word){
int index=multiplicationCompression(word);
put(word,index);
    }
public void putMidsquare(String word){
int index=midSquare(word);
put(word,index);
    }
public void putFolding(String word){
int index=folding(word);
```

```
put(word,index);
    }
public void putRadixConversion(String word){
int index=radixConversion(word);
put(word,index);
    }

public void displayTheTable(){

for(int i=0;i<arrayHash.length;i++){
System.out.print("index: "+i+":");
for(int j=0;j<arrayHash[i].size();j++){
if(j==arrayHash[i].size()-1){
System.out.print(arrayHash[i].get(j));
              }
else{
System.out.print(arrayHash[i].get(j));
System.out.print(",");
              }

           }
System.out.println();
       }

    }

public String getRadical(int value){
       String result="";
       String v=value+"";
int index=radical(value);
for(int j=0;j<arrayHash[index].size();j++){
```

```
if(arrayHash[index].get(j).equals(v)){
result=arrayHash[index].get(j).toString();
          }
       }
return result;
   }

public String getExtraction(String value){
       String result="";
int index=extraction(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
result=arrayHash[index].get(j).toString();
          }
       }
return result;
   }

public String getCompression(String value){
       String result="";
int index=compression(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
result=arrayHash[index].get(j).toString();
          }
       }
return result;
   }

public String getDivisionCompression(String value){
       String result="";
int index=divisionCompression(value);
```

```
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
result=arrayHash[index].get(j).toString();
        }
      }
return result;
   }

public String getMultiplicationCompression(String value){
      String result="";
int index=multiplicationCompression(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
result=arrayHash[index].get(j).toString();
        }
      }
return result;
   }

public String getMidSquare(String value){
      String result="";
int index=midSquare(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
result=arrayHash[index].get(j).toString();
        }
      }
return result;
   }
public String getFolding(String value){
      String result="";
int index=folding(value);
```

```
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
result=arrayHash[index].get(j).toString();
        }
      }
return result;
   }
public String getRadixConversion(String value){
      String result="";
int index=radixConversion(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
result=arrayHash[index].get(j).toString();
        }
      }
return result;
   }



public void searchRadical(int value){
int m=0;
int index=radical(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value+"")){
System.out.println("Found");
            m=1;
break;
        }
      }
if(m==0){
System.out.println("Not Found");
      }
```

```
    }
public void searchMultiplicationCompression(String value){
int m=0;
int index=multiplicationCompression(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
System.out.println("Found");
                m=1;
break;
            }
        }
if(m==0){
System.out.println("Not Found");
        }

    }
public void searchDivisionCompression(String value){
int m=0;
int index=divisionCompression(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
System.out.println("Found");
                m=1;
break;
            }
        }
if(m==0){
System.out.println("Not Found");
        }

    }
```

```
public void searchCompression(String value){
int m=0;
int index=compression(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
System.out.println("Found");
            m=1;
break;
        }
      }
if(m==0){
System.out.println("Not Found");
      }

  }
public void searchExtraction(String value){
int m=0;
int index=extraction(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
System.out.println("Found");
            m=1;
break;
        }
      }
if(m==0){
System.out.println("Not Found");
      }

  }
public void searchMidSquare(String value){
int m=0;
```

```
int index=midSquare(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
System.out.println("Found");
            m=1;
break;
          }
       }
if(m==0){
System.out.println("Not Found");
       }

   }
public void searchFolding(String value){
int m=0;
int index=folding(value);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).equals(value)){
System.out.println("Found");
            m=1;
break;
          }
       }
if(m==0){
System.out.println("Not Found");
       }

   }
public void searchRadixConversion(String value){
int m=0;
int index=radixConversion(value);
for(int j=0;j<arrayHash[index].size();j++){
```

```
if(arrayHash[index].get(j).equals(value)){
System.out.println("Found");
            m=1;
break;
          }
       }
if(m==0){
System.out.println("Not Found");
       }

   }


public void deleteRadical(int value){
int index=radical(value);
arrayHash[index].remove(value+"");
   }

public void deleteExtraction(String value){
int index=extraction(value);
arrayHash[index].remove(value);
   }
public void deleteCompression(String value){
int index=compression(value);
arrayHash[index].remove(value);
   }
public void deleteDivisionCompression(String value){
int index=divisionCompression(value);
arrayHash[index].remove(value);
   }
public void deleteMultiplicationCompression(String value){
int index=multiplicationCompression(value);
```

```
arrayHash[index].remove(value);
    }
public void deleteMidSquare(String value){
int index=midSquare(value);
arrayHash[index].remove(value);
    }
public void deleteFolding(String value){
int index=folding(value);
arrayHash[index].remove(value);
    }
public void deleteRadixConversion(String value){
int index=radixConversion(value);
arrayHash[index].remove(value);
    }
}
```

HashTableArrayDatabase.java

```
packagehashtablearraydemo;

importjava.util.LinkedList;

public class HashTableArrayDatabase {



LinkedList<DatabaseHashing>[] arrayHash;

publicHashTableArrayDatabase(int Size){
arrayHash = new LinkedList[Size];
for(int i=0;i<arrayHash.length;i++){
arrayHash[i]=new LinkedList<>();
      }

   }

intdatabaseHash(int id){
return id%15;
   }

public void put(intindex,intid,Stringname,Stringdept,String address){
arrayHash[index].add(new DatabaseHashing(id,name,dept,address));
   }
public void putDataHash(intid,Stringname,Stringdept,String address){
int index=databaseHash(id);
put(index,id,name,dept,address);
   }

public void deleteData(int id){
```

```
int index=databaseHash(id);

for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).getId()==id){
arrayHash[index].remove(j);

        }

        }

    }
public void updateName(intid,String name){
int index=databaseHash(id);

for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).getId()==id){
arrayHash[index].get(j).setName(name);

        }

        }

    }
public void updateDept(intid,Stringdept){
int index=databaseHash(id);

for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).getId()==id){
arrayHash[index].get(j).setDept(dept);

        }
```

```
        }


    }
public void updateAddress(intid,String address){
int index=databaseHash(id);

for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).getId()==id){
arrayHash[index].get(j).setAddress(address);

        }


        }


    }

public void searchDatabase(intid,String name){
int m=0;
int index=databaseHash(id);
for(int j=0;j<arrayHash[index].size();j++){
if(arrayHash[index].get(j).getName().equals(name)){
System.out.println("Found");
                m=1;
break;
            }
        }
if(m==0){
System.out.println("Not Found");
        }


    }
public void displayData(){
```

```
for(int i=0;i<arrayHash.length;i++){
System.out.print("index: "+i+":");
for(int j=0;j<arrayHash[i].size();j++){
if(j==arrayHash[i].size()-1){
System.out.print(arrayHash[i].get(j).getId());
System.out.print(",");
System.out.print(arrayHash[i].get(j).getName());
System.out.print(",");
System.out.print(arrayHash[i].get(j).getDept());
System.out.print(",");
System.out.print(arrayHash[i].get(j).getAddress());
            }
else{
System.out.print(arrayHash[i].get(j).getId());
System.out.print(",");
System.out.print(arrayHash[i].get(j).getName());
System.out.print(",");
System.out.print(arrayHash[i].get(j).getDept());
System.out.print(",");
System.out.print(arrayHash[i].get(j).getAddress());
System.out.print(";;");
            }

        }
System.out.println();
      }
   }


}
```

Hashing.java

```
packagehashtablearraydemo;

public class Hashing {
int size;
String[] hashData;

Hashing(int size){
this.size=size;
hashData =new String[size];
    }

inthashFunction(int key){
returnkey%size;
    }

    //Linear Probing
public void putLinearProbing(intindex,String value){

while(hashData[index]!=null){
index+=1;
if(index==size){
index=0;
            }
        }
hashData[index]=value;
    }
    //Quadratic Probing
public void putQuadraticProbing(intindex,String value){

index=(index+(index*index))%size;
```

```
while(hashData[index]!=null){
index+=1;
if(index==size){
index=0;
            }
        }
hashData[index]=value;
    }
public void putDataLinearProbing(intkey,String value){
int index=hashFunction(key);
putLinearProbing(index,value);

    }
public void putDataQuadraticProbing(intkey,String value){
int index=hashFunction(key);
putQuadraticProbing(index,value);
    }

public void showData(){
for(int i=0;i<hashData.length;i++){
System.out.println("index: "+i+":"+hashData[i]);
        }
    }
}
```

Input.txt

THE OF AND TO A IN THAT IS

FROM THEY ONE DRY TEA GLUM WE

AWARD

Databaseinput.txt

1 Nazmul CSE Dhaka

2 Jack EEE Comilla

3 John BBB Dhaka

4 Rahim CSE Dhaka

5 Karim EEE Comilla

6 Suraiya BBB Noyakhali

7 Samia CSE Dhaka

8 Fahim EEE Comilla

9 Tanzim BBB Rajshahi

10 Habib CSE Dhaka

11 Islam EEE Comilla

12 Khan BBB Dhaka

13 Sumaiya CSE Dhaka

14 Nayem EEE Dinajpur

15 Rasel BBB Dhaka

16 Kabir CSE Barishal

17 Mamun EEE Comilla

18 Tarek BBB Dhaka

19 Sishir CSE Dhaka

20 Bithi EEE Comilla

21 Shahriar BBB Dhaka

22 Toya CSE Dhaka

23 Jannat EEE Bogra

24 Sabrina BBB Dhaka

25 Arafat BBB Sylhet

intInput.txt

101 102 7 14 18

25 26 3 15

OpenAddressingInput.txt

501 Nazmul

402 Rahul

403 Jack

204 John

101 Smith

Output:

run:

Hash data in Buckets

Bucket0:

Bucket1: Nazmul,Smith

Bucket2: Rahul

Bucket3: Jack

Bucket4: John

Hash data using Linear Probing

index: 0:Smith

index: 1:Nazmul

index: 2:Rahul

index: 3:Jack

index: 4:John

Hash data using Quadratic Probing

index: 0:John

index: 1:Rahul

index: 2:Nazmul

index: 3:Jack

index: 4:Smith

Hash table using Extraction chaining

index: 0:AND,AWARD

index: 1:A

index: 2:IN

index: 3:IS

index: 4:THAT

index: 5:THE,FROM,THEY,ONE,DRY,TEA,GLUM,WE

index: 6:

index: 7:TO

THE

Found

Hash table using Compression chaining

index: 0:THEY

index: 1:A,AWARD

index: 2:

index: 3:

index: 4:ONE

index: 5:

index: 6:

index: 7:IN

index: 8:

index: 9:OF,THAT

index: 10:

index: 11:AND

index: 12:

index: 13:

index: 14:

index: 15:DRY

index: 16:TEA

index: 17:

index: 18:WE

index: 19:GLUM

index: 20:

index: 21:

index: 22:FROM

index: 23:

index: 24:

index: 25:

index: 26:IS

index: 27:TO

index: 28:

index: 29:

index: 30:

index: 31:

Found

Hash table using DivisionCompression chaining

index: 0:

index: 1:A

index: 2:THE

index: 3:ONE

index: 4:TO

index: 5:

index: 6:

index: 7:

index: 8:

index: 9:

index: 10:

index: 11:

index: 12:

index: 13:

index: 14:

index: 15:

index: 16:DRY

index: 17:

index: 18:THAT

index: 19:AND

index: 20:

index: 21:OF,FROM

index: 22:GLUM

index: 23:IN

index: 24:

index: 25:

index: 26:TEA

index: 27:THEY

index: 28:IS,WE

index: 29:

index: 30:

index: 31:

IN

Found

Hash table using MultiplicationCompression chaining

index: 0:A

index: 1:THEY,GLUM

index: 2:IS

index: 3:FROM

index: 4:AND,ONE,DRY

index: 5:

index: 6:

index: 7:TO

index: 8:

index: 9:

index: 10:

index: 11:

index: 12:

index: 13:

index: 14:

index: 15:TEA

index: 16:

index: 17:THAT

index: 18:

index: 19:

index: 20:

index: 21:OF

index: 22:

index: 23:THE

index: 24:

index: 25:AWARD

index: 26:

index: 27:WE

index: 28:

index: 29:

index: 30:

index: 31:

THE

Not Found

Hash table using MidSquare chaining

index: 0:A,THEY,AWARD

index: 1:ONE

index: 2:THE,AND,TO,DRY,WE

index: 3:

index: 4:

index: 5:TEA

index: 6:GLUM

index: 7:IS

index: 8:OF,THAT,FROM

index: 9:

THE

Not Found

Hash table using Folding chaining

index: 0:THEY

index: 1:A,AWARD

index: 2:

index: 3:

index: 4:AND

index: 5:

index: 6:

index: 7:ONE

index: 8:

index: 9:OF,THAT,DRY,WE

index: 10:GLUM

index: 11:

index: 12:

index: 13:THE,TEA

index: 14:

index: 15:

index: 16:FROM

index: 17:

index: 18:TO

index: 19:IS

index: 20:

index: 21:

index: 22:

index: 23:

index: 24:

index: 25:

index: 26:

index: 27:

Not Found

Hash table using RadixConversion chaining

index: 0:THEY

index: 1:A,AWARD

index: 2:

index: 3:

index: 4:ONE

index: 5:

index: 6:

index: 7:

index: 8:

index: 9:

index: 10:

index: 11:OF,THAT

index: 12:

index: 13:AND

index: 14:

index: 15:

index: 16:

index: 17:DRY

index: 18:

index: 19:

index: 20:TEA

index: 21:

index: 22:WE

index: 23:GLUM

index: 24:

index: 25:

index: 26:FROM

index: 27:

index: 28:

index: 29:

index: 30:

index: 31:THE

index: 32:IS

index: 33:TO

index: 34:

index: 35:

index: 36:

index: 37:

THE

Not Found

Hash table using Radical chaining

index: 0:

index: 1:101

index: 2:102

index: 3:

index: 4:14

index: 5:25,15

index: 6:26

index: 7:7

index: 8:18

index: 9:

101

Found

Hash table file of Database chaining

index: 0:

index: 1:1,Nazmul,CSE,Dhaka;;16,Kabir,CSE,Barishal

index: 2:2,Jack,EEE,Comilla;;17,Mamun,EEE,Comilla

index: 3:3,John,BBB,Dhaka;;18,Tarek,BBB,Dhaka

index: 4:4,Rahim,CSE,Dhaka;;19,Sishir,CSE,Dhaka

index: 5:5,Karim,EEE,Comilla;;20,Bithi,EEE,Comilla

index: 6:6,Suraiya,BBB,Noyakhali;;21,Shahriar,BBB,Dhaka

index: 7:7,Samia,CSE,Dhaka;;22,Toya,CSE,Dhaka

index: 8:8,Fahim,EEE,Comilla;;23,Jannat,EEE,Bogra

index: 9:9,Tanzim,BBB,Rajshahi;;24,Sabrina,BBB,Dhaka

index: 10:10,Habib,CSE,Dhaka;;25,Arafat,BBB,Sylhet

index: 11:11,Islam,EEE,Comilla

index: 12:12,Khan,BBB,Dhaka

index: 13:13,Sumaiya,CSE,Dhaka

index: 14:14,Nayem,EEE,Dinajpur

index: 0:

index: 1:1,Nazmul,CSE,Dhaka;;16,Kabir,CSE,London

index: 2:2,Yasin,EEE,Comilla;;17,Mamun,EEE,Comilla

index: 3:3,John,BBB,Dhaka;;18,Tarek,BBB,Dhaka

index: 4:4,Rahim,CSE,Dhaka;;19,Sishir,CSE,Dhaka

index: 5:5,Karim,EEE,Comilla;;20,Bithi,EEE,Comilla

index: 6:6,Suraiya,BBB,Noyakhali;;21,Shahriar,BBB,Dhaka

index: 7:7,Samia,CSE,Dhaka;;22,Toya,CSE,Dhaka

index: 8:8,Fahim,EEE,Comilla;;23,Jannat,EEE,Bogra

index: 9:9,Tanzim,BBB,Rajshahi;;24,Sabrina,BBB,Dhaka

index: 10:10,Habib,CSE,Dhaka;;25,Arafat,BBB,Sylhet

index: 11:11,Islam,EEE,Comilla

index: 12:12,Khan,BBB,Dhaka

index: 13:13,Sumaiya,CSE,Dhaka

index: 14:14,Nayem,EEE,Dinajpur

Found