DOUBLE HASHING

Double hashing is a collision resolving technique in Open Addressed Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

Advantages of Double hashing

The advantage of Double hashing is that it is one of the best form of probing, producing a uniform distribution of records throughout a hash table.

This technique does not yield any clusters.

It is one of effective method for resolving collisions.

Double hashing can be done using :

(hash1(key) + i \* hash2(key)) % TABLE\_SIZE

Here hash1() and hash2() are hash functions and TABLE\_SIZE

is size of hash table.

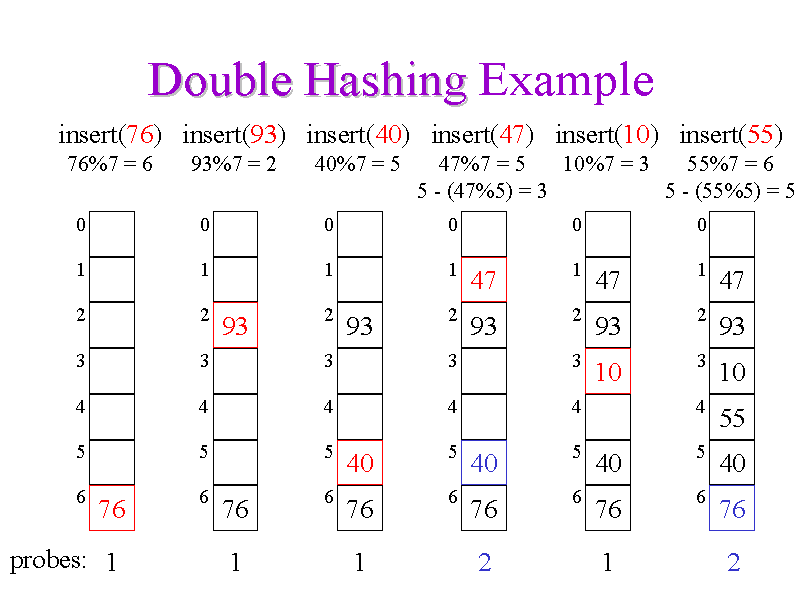
(We repeat by increasing i when collision occurs)

First hash function is typically hash1(key) = key % TABLE\_SIZE

A popular second hash function is : hash2(key) = PRIME – (key % PRIME) where PRIME is a prime smaller than the TABLE\_SIZE.

A good second Hash function is:

It must never evaluate to zero



import java.util.Scanner;

import java.math.\*;

/\* Class LinkedHashEntry \*/

class HashEntry

{

String key;

int value;

/\* Constructor \*/

HashEntry(String key, int value)

{

this.key = key;

this.value = value;

}

}

/\* Class HashTable \*/

class HashTable

{

private int TABLE\_SIZE;

private int size;

private HashEntry[] table;

private int primeSize;

/\* Constructor \*/

public HashTable(int ts)

{

size = 0;

TABLE\_SIZE = ts;

table = new HashEntry[TABLE\_SIZE];

for (int i = 0; i < TABLE\_SIZE; i++)

table[i] = null;

primeSize = getPrime();

}

/\* Function to get prime number less than table size for myhash2 function \*/

public int getPrime()

{

for (int i = TABLE\_SIZE - 1; i >= 1; i--)

{

int fact = 0;

for (int j = 2; j <= (int) Math.sqrt(i); j++)

if (i % j == 0)

fact++;

if (fact == 0)

return i;

}

/\* Return a prime number \*/

return 3;

}

/\* Function to get number of key-value pairs \*/

public int getSize()

{

return size;

}

public boolean isEmpty()

{

return size == 0;

}

/\* Function to clear hash table \*/

public void makeEmpty()

{

size = 0;

for (int i = 0; i < TABLE\_SIZE; i++)

table[i] = null;

}

/\* Function to get value of a key \*/

public int get(String key)

{

int hash1 = myhash1( key );

int hash2 = myhash2( key );

while (table[hash1] != null && !table[hash1].key.equals(key))

{

hash1 += hash2;

hash1 %= TABLE\_SIZE;

}

return table[hash1].value;

}

/\* Function to insert a key value pair \*/

public void insert(String key, int value)

{

if (size == TABLE\_SIZE)

{

System.out.println("Table full");

return;

}

int hash1 = myhash1( key );

int hash2 = myhash2( key );

while (table[hash1] != null)

{

hash1 += hash2;

hash1 %= TABLE\_SIZE;

}

table[hash1] = new HashEntry(key, value);

size++;

}

/\* Function to remove a key \*/

public void remove(String key)

{

int hash1 = myhash1( key );

int hash2 = myhash2( key );

while (table[hash1] != null && !table[hash1].key.equals(key))

{

hash1 += hash2;

hash1 %= TABLE\_SIZE;

}

table[hash1] = null;

size--;

}

/\* Function myhash which gives a hash value for a given string \*/

private int myhash1(String x )

{

int hashVal = x.hashCode( );

hashVal %= TABLE\_SIZE;

if (hashVal < 0)

hashVal += TABLE\_SIZE;

return hashVal;

}

/\* Function myhash function for double hashing \*/

private int myhash2(String x )

{

int hashVal = x.hashCode( );

hashVal %= TABLE\_SIZE;

if (hashVal < 0)

hashVal += TABLE\_SIZE;

return primeSize - hashVal % primeSize;

}

/\* Function to print hash table \*/

public void printHashTable()

{

System.out.println("\nHash Table");

for (int i = 0; i < TABLE\_SIZE; i++)

if (table[i] != null)

System.out.println(table[i].key +" "+table[i].value);

}

}

/\* Class DoubleHashingHashTableTest \*/

public class Main

{

public static void main(String[] args)

{

Scanner scan = new Scanner(System.in);

System.out.println("Hash Table Test\n\n");

System.out.println("Enter size");

/\* Make object of HashTable \*/

HashTable ht = new HashTable(scan.nextInt() );

char ch;

/\* Perform HashTable operations \*/

do

{

System.out.println("\nHash Table Operations\n");

System.out.println("1. insert ");

System.out.println("2. remove");

System.out.println("3. get");

System.out.println("4. check empty");

System.out.println("5. clear");

System.out.println("6. size");

int choice = scan.nextInt();

switch (choice)

{

case 1 :

System.out.println("Enter key and value");

ht.insert(scan.next(), scan.nextInt() );

break;

case 2 :

System.out.println("Enter key");

ht.remove( scan.next() );

break;

case 3 :

System.out.println("Enter key");

System.out.println("Value = "+ ht.get( scan.next() ));

break;

case 4 :

System.out.println("Empty Status " +ht.isEmpty());

break;

case 5 :

ht.makeEmpty();

System.out.println("Hash Table Cleared\n");

break;

case 6 :

System.out.println("Size = "+ ht.getSize());

break;

default :

System.out.println("Wrong Entry \n ");

break;

}

/\* Display hash table \*/

ht.printHashTable();

System.out.println("\nDo you want to continue (Type y or n) \n");

ch = scan.next().charAt(0);

} while (ch == 'Y'|| ch == 'y');

}

}

Output

Enter size

100

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

1

Enter key and value

prime 97

Hash Table

prime 97

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

1

Enter key and value

even 24

Hash Table

prime 97

even 24

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

1

Enter key and value

odd 63

Hash Table

prime 97

even 24

odd 63

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

1

Enter key and value

composite 6

Hash Table

prime 97

even 24

odd 63

composite 6

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

1

Enter key and value

armstrong 153

Hash Table

prime 97

even 24

odd 63

armstrong 153

composite 6

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

3

Enter key

prime

Value = 97

Hash Table

prime 97

even 24

odd 63

armstrong 153

composite 6

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

3

Enter key

even

Value = 24

Hash Table

prime 97

even 24

odd 63

armstrong 153

composite 6

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

2

Enter key

composite

Hash Table

prime 97

even 24

odd 63

armstrong 153

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

6

Size = 4

Hash Table

prime 97

even 24

odd 63

armstrong 153

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

5

Hash Table Cleared

Hash Table

Do you want to continue (Type y or n)

y

Hash Table Operations

1. insert

2. remove

3. get

4. check empty

5. clear

6. size

Empty Status true

Hash Table

Do you want to continue (Type y or n)

n