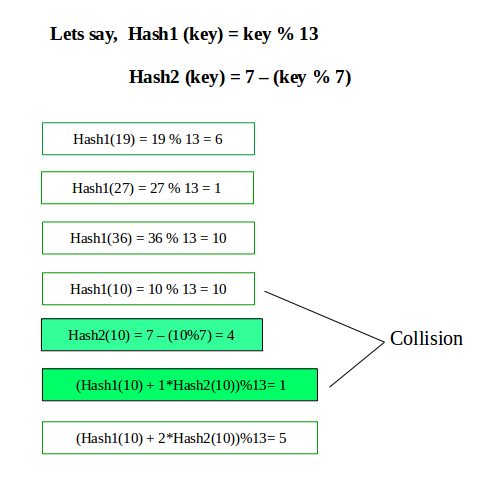
**Double hashing** is a collision resolving technique in **Open Addressed** Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

**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.



There are two conditions we need to keep in mind.

* It must never evaluate to zero
* Must make sure that all cells can be probed

|  |
| --- |
| // Java Program to implement hashtable in  // double hashing  // Here performing additional task  // which is to remove the entered items  // Importing input output classes  import java.io.\*;  // Importing all classes from java.util package  import java.util.\*;  // Class 1  // Class LinkedHashEntry  class ValueEntry {  // Member variables of this class  String key;  int value;  // Constructor of this class  // Parameterized constructor  ValueEntry(String key, int value)  {  // 'This' keyword refers to the current object itself  // to assign the values  this.key = key;    // This keyword is pointer which contains location  // of that container that have key and value pairs  this.value = value;  }  }  // Class 2  // Helper class  // Class HashTable  class HashTable {  // Member variable of this class  private int HASH\_TABLE\_SIZE;  private int size;  private ValueEntry[] table;  private int totalprimeSize;  // Constructor of this class  // Parameterized constructor  public HashTable(int ts)  {  // Initially, initializing the parameters  // to some values  size = 0;  HASH\_TABLE\_SIZE = ts;  table = new ValueEntry[HASH\_TABLE\_SIZE];  // Iterating using for loop over table  for (int i = 0; i < HASH\_TABLE\_SIZE; i++)  // Initially table is empty  table[i] = null;  totalprimeSize = getPrime();  }  // Method 1  // To check for the prime number  public int getPrime()  {  // Iterating over hashtable using nested for loops  // in reverse order  for (int i = HASH\_TABLE\_SIZE - 1; i >= 1; i--) {  // Initially count is zero  int cnt = 0;  for (int j = 2; j \* j <= i; j++)  if (i % j == 0)  cnt++;  if (cnt == 0)  return i;  }  // Returning a prime number  return 3;  }  // Method 2  // To get snumber of key-value pairs  public int getSize()  { return size; }  public boolean isEmpty()  { return size == 0; }  // Method 3  // To clear the hash table  public void makeEmpty()  {  // Initially size set to zero  size = 0;  for (int i = 0; i < HASH\_TABLE\_SIZE; i++)  table[i] = null;  }  // Method 3  // To get value of a key  public int getkey(String key)  {  int hash1 = myhash1(key);  int hash2 = myhash2(key);  while (table[hash1] != null  && !table[hash1].key.equals(key)) {  hash1 += hash2;  hash1 %= HASH\_TABLE\_SIZE;  }  return table[hash1].value;  }  // Method 4  // To insert a key-value pair  public void insert(String key, int value)  {  // Checking the size of table and  // comparing it with users input value  if (size == HASH\_TABLE\_SIZE) {  // Display message  System.out.println("Table is full");  return;  }  int hashing1 = myhash1(key);  int hashing2 = myhash2(key);  while (table[hashing1] != null) {  hashing1 += hashing2;  hashing1 %= HASH\_TABLE\_SIZE;  }  table[hashing1] = new ValueEntry(key, value);  size++;  }  // Method 4  // To remove a key from hashtable  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 %= HASH\_TABLE\_SIZE;  }  table[hash1] = null;  size--;  }  // Method 5  // This method returns a hash value for a given  // string as it is linear probing  private int myhash1(String y)  {  int myhashVal1 = y.hashCode();  myhashVal1 %= HASH\_TABLE\_SIZE;  if (myhashVal1 < 0)  myhashVal1 += HASH\_TABLE\_SIZE;  return myhashVal1;  }  // Method 6  // In this function, 'myhash'function for double hashing  // after linear probing. A quadratic probing is used in  // which two 'myhash' functions are used  // as it is double chaining  private int myhash2(String y)  {  int myhashVal2 = y.hashCode();  myhashVal2 %= HASH\_TABLE\_SIZE;  if (myhashVal2 < 0)  myhashVal2 += HASH\_TABLE\_SIZE;  return totalprimeSize - myhashVal2 % totalprimeSize;  }  // Method 7  // To print hash table  public void printHashTable()  {  // Display message  System.out.println("\nHash Table");  // Iterating over the table  for (int i = 0; i < HASH\_TABLE\_SIZE; i++)  if (table[i] != null)  System.out.println(table[i].key + " "  + table[i].value);  }  }  // Class 3  // Main class  // Class for DoubleHashingHashTableTest  public class GFG {  // Main driver method  public static void main(String[] args)  {  // Display message  System.out.println("Hash Table Testing");    // Step 1: Creating an object of hashtable  // of custom size 100 which signifies  // table can hold 100 key-value pairs  HashTable ht = new HashTable(100);  // Step 2: Adding(appending) the values to  // the hashtable object  // Custom inputs of key-value pairs  ht.insert("prime", 97);  ht.insert("even", 96);  ht.insert("odd", 95);    // Step 3: Printing hash table after insertion  // of key-value pairs    // Calling print hash table function using object  // we call it with object.function\_name  ht.printHashTable();    // Primarily goal of the program  // Step 4: Removing elements with using key values  // using the remove() method  ht.remove("prime");  ht.printHashTable();  }  } |

**Output**

|  |
| --- |
| Hash Table Testing  Hash Table  prime 97  even 96  odd 95  Hash Table  even 96  odd 95 |

|  |
| --- |
| *Similarly, we can get the size of hashed table, can clear the elements from hash table, can get our desired element in hash function. In order to get*   * *For size can use ht.getSize()* * *For element can use ht.get(String)*   *Where ht is object name. In the same way, we can call our other functions in the above code.* |