Hashing

Objective

The objective of this lab is to understand the implementation of hashing using open addressing technique that include linear probing and quadratic probing.

Task

Implement hash table using open addressing technique and follow good hash function properties.

a. Apply linear probing and analyze the number of collisions,
b. Apply quadratic probing and analyze the number of collisions.
c. Finally understand the big O of insertion, and search operations in the hash table. Hint: Count the number of rehashes at each insertion to analyze the worst case. Count number of occupied cells to estimate uniform distribution of keys in hash table.

Procedure

- Create an array to build a hash table. Use random function to generate keys of 3-digit. When collision occurred call rehash, the key using open addressing techniques. Assum no duplicate keys are allowed.
- Analyze which method is more suitable with minimum collisions. Count number of collisions of each key to compare both methods (linear and quadratic) for the same data.

```
class HashingDemo {

  public static void main(String args[]){

   HashTable H=new HashTable(100);

  Random R = new Random
    for (int i = 0; i < 100; i++) {
          R.nextInt(900) + 100;          // generate hundred 3-digit random number
         H.insert(R);
  }
  System.out.println(H);

  H.find(R); // find number you already insert , also check the not found case
  System.out.println(H);
}
}
```

```
public class HashTable {

  int[] Table;

     int numofCollisions = 0;
     int numofOccupiedCells = 0;

   HashTable(int s){
      // table size should be a prime number and 1/3 extra.
         int size=s+(s/3);
        int newSize = getPrime(size);

         Table=new int [newSize];   // if value is 0 for integer the cell will consider empty.
   }
   private int getPrime(int n) {
          while(true) {
                if (isPrime(n)) return n;
                     n++;
           }
   }
    private boolean isPrime(int n) {
          for (int i = 2; i <= n/2; i++) {
               if (n % i == 0) return false;
                     }
                     return true;
          }

   public int Hash(int key){
      //compute hash value by taking mod on key value and return remainder
   }
   public int Rehash(int key, int i){
          // first test linear probing, then test Quadratic probing and  compare both technique on same data
          with respect to number of collision
   }
   public void insert(int key){   // keep maintain 1/3 empty cells
       // call Hash(key) and save return hash-value
       // if (no collision on hash-value) then place in table
       //else call rehash function until empty cell found or reached to threshold limit of rehashes
       // also count number of collisions on each key insertion
   }
   public boolean search (int key) {
       // search word in a hash table
       // call Hash(key) and save return hash-value
       // if (value match at hash-value index) return true
       //else call rehash function until empty cell found or it reaches threshold limit of rehashes.
   }
    public String toString(){
      String str="";
      for (int i = 0; i < Table.length; i++) {
         str=str+"["+i+"] "+Table[i]+"\n";
      }
     return str;
    }
}
```