Assignment 1 - CT255 CyberSecurity

Daniel Hannon (19484286)

November 2020

1 Problem 2

```java
/**
 *
 * @author Michael Schukat, 2019 modified by Daniel Hannon
 */
import java.util.Date;

public class CT255_HashFunction1 {

  public static void main(String[] args) {
    int res = 0;
    int res1 = 0;
    int collisions_found = 0;

    if (args != null && args.length > 0) { // Check for <input> value
      res = hashF1(args[0]); // call hash function with <input>
      if (res < 0) { // Error
        System.out.println("Error: <input> must be 1 to 64 characters long.");
      }
      else {
        System.out.println("input = " + args[0] + " : Hash = " + res);
        Date time = new Date();
        long timeStart = time.getTime();
        System.out.println("Start searching for collisions");
        while(collisions_found < 10) {
          String test = "";
          for(int i = 0; i < 5; i++) { /*Length 5 string*/
            test+=Character.toString((char)( Math.random() * 78)+48); /*ASCII value "0" is 48
            and "~" is 126*/
          }
          res1 = hashF1(test);
          if(res == res1 && args[0].equals(test) == false) { /*Compares hashes and strings to
          make sure that it is not the exact same string if there's a duplicate*/
            time = new Date();
            long currtime = time.getTime() - timeStart;
            System.out.println(currtime+"ms: Collision Found! " + test);
            collisions_found++;
          }
        }
      }
    }
    else { // No <input>
      System.out.println("Use: CT255_HashFunction1 <Input>");
    }
  }

  private static int hashF1(String s){
    int ret = -1, i;
    int[] hashA = new int[]{1, 1, 1, 1};

    String filler, sIn;

    filler = new
     ↪  String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");

    if ((s.length() > 64) || (s.length() < 1)) { // String does not have required length
      ret = -1;
    }
    else {
      sIn = s + filler; // Add characters, now have "<input>ABCDEF..."
      sIn = sIn.substring(0, 64); // // Limit string to first 64 characters
      // System.out.println(sIn); // FYI
      for (i = 0; i < sIn.length(); i++){
        char byPos = sIn.charAt(i); // get i'th character
        hashA[0] += (byPos * 17); // Note: A += B means A = A + B
        hashA[1] += (byPos * 31);
        hashA[2] += (byPos * 101);
        hashA[3] += (byPos * 79);
      }

      hashA[0] %= 255;  // % is the modulus operation, i.e. division with rest
      hashA[1] %= 255;
      hashA[2] %= 255;
      hashA[3] %= 255;

      ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) + (hashA[3] * 256 * 256 *
       ↪  256);
      if (ret < 0) ret *= -1;
    }
    return ret;
  }
}
```

2 Problem 3

```java
/**
 *
 * @author Michael Schukat, 2019 Modified by Daniel Hannon
 */
import java.util.Date;

public class CT255_HashFunction2 {

  public static void main(String[] args) {
    int res = 0;
    int res1 = 0;
    int collisions_found = 0;

    if (args != null && args.length > 0) { // Check for <input> value
      res = hashF2(args[0]); // call hash function with <input>
      if (res < 0) { // Error
        System.out.println("Error: <input> must be 1 to 64 characters long.");
      }
      else {
        System.out.println("input = " + args[0] + " : Hash = " + res);
        /*Added time it takes to compare Algorithms*/
        Date time = new Date();
        long timeStart = time.getTime();
        System.out.println("Start searching for collisions");
        while(collisions_found < 10) {
          String test = "";
          for(int i = 0; i < 5; i++) { /*Length 5 string*/
            test+=Character.toString((char)( Math.random() * 78)+48); /*ASCII value "0" is 48 and "~" is 126*/
          }
          res1 = hashF2(test);
          if(res == res1 && args[0].equals(test) == false) { /*Compares hashes and strings to make sure that it is not the exact same string if there's a duplicate*/
            time = new Date();
            /*Get Time of collision relative to program start*/
            long timecurr = time.getTime() - timeStart;
            System.out.println(timecurr+"ms: Collision Found! " + test);
            collisions_found++;
          }
        }
      }
    }
    else { // No <input>
      System.out.println("Use: CT255_HashFunction1 <Input>");
    }
  }

  private static int hashF2(String s){
    int ret = -1, i;
    int[] hashA = new int[]{1, 1, 1, 1};

    String filler, sIn;
    char space1,space2,space3;
    /* By swapping around one AB in this I have completely removed several duplicates */
    filler = new String("ABCDEFGHBACDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");
  char[] primes = {17,79,91,103,83,67,101,47,89,107,19,41}; /*Random primes in a random order*/
  char[] primes2 = {91,19,107,101,83,89,41,17,103,79,67,47}; /*Different Order of the same primes*/

  if ((s.length() > 64) || (s.length() < 1)) { // String does not have required length
   ret = -1;
  }
  else {
   sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
   sIn = sIn.substring(0, 64); // // Limit string to first 64 characters
   space1 = sIn.charAt(5);
   space2 = sIn.charAt(11);
   space3 = sIn.charAt(43);

   // System.out.println(sIn); // FYI
   for (i = 0; i < sIn.length(); i++){
    char byPos = sIn.charAt(i); // get i'th character
   /*This works like a somewhat Primitive Enigma Device wherein the number
   that the value is changed by depends heavily on the position of the letter
   within the string, to introduce some extra challenge to crack it
   the displacement of the 2nd through 4th rotors depends on the values
   held at fixed arbitrary positions within the string, not only this, the length
   of the rotor which is operated is staggered.
   To reduce the amount of collisions I also changed the filler text to include
   "BACDEFGFH" in one location so duplicates like STRINGABCDEFGH would no longer
   exist.*/
    hashA[0] += (byPos * (primes[i % 12]));
    hashA[1] += (byPos * (primes[(i + space1) % 11]));
    hashA[2] += (byPos * (primes2[(i + space2) % 12]));
    hashA[3] += (byPos * (primes2[(i + space3) % 11]));
   }
   hashA[0] %= 255;  // % is the modulus operation, i.e. division with rest
   hashA[1] %= 255;
   hashA[2] %= 255;
   hashA[3] %= 255;

   ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) + (hashA[3] * 256 * 256 * 256);
   if (ret < 0) ret *= -1;
  }
  return ret;
 }
}
```