Assignment 2 - CT255 CyberSecurity

Daniel Hannon (19484286)

November 2020

1 Problem 1 code

```
/**
 *  This class provides functionality to build rainbow tables (with a different reduction function per
  consist of the symbols "a .. z", "A .. Z", "0 .. 9", "!" and "#" (64 symbols in total).
  Properly used, it creates the following value pairs (start value - end value) after 10,000 iteration
      start value  -  end value
      Kermit12        lsXcRAuN
      Modulus!        L2rEsY8h
      Pigtail1        RONoLfOw
      GalwayNo        9PZjwF5c
      Trumpets        !oeHRZpK
      HelloPat        dkMPG7!U
      pinky##!        eDx58HRq
      01!19!56        vJ9OePjV
      aaaaaaaa        rLtVvpQS
      036abgH#        klQ6IeQJ


 *
 * @author Michael Schukat
 * @version 1.0
 */
public class RainbowTable
{
  /**
   * Constructor, not needed for this assignment
   */
  public RainbowTable() {

  }

  public static void main(String[] args) {
    long res = 0;
    int i;
    String start;

    if (args != null && args.length > 0) { // Check for <input> value
      start = args[0];

      if (start.length() != 8) {
        System.out.println("Input " + start + " must be 8 characters long - Exit");
      }
      else {
          // Your code for problem 1 starts here
          res = hashFunction(start); //Need to generate all 10000 hashes
        for (i = 0; i<10000;i++) {
          start = reductionFunction(res, i); //Get ith string
          res = hashFunction(start); //get ith hash
        }
        System.out.println(start); //print end string
      }
    }
    else { // No <input>
```

```
53        System.out.println("Use: RainbowTable <Input>");
54      }
55    }
56
57    private static long hashFunction(String s){
58      long ret = 0;
59      int i;
60      long[] hashA = new long[]{1, 1, 1, 1};
61
62      String filler, sIn;
63
64      int DIV = 65536;
65
66      filler = new
          ↪ String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");
67
68      sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
69      sIn = sIn.substring(0, 64); // // Limit string to first 64 characters
70
71      for (i = 0; i < sIn.length(); i++) {
72        char byPos = sIn.charAt(i); // get i'th character
73        hashA[0] += (byPos * 17111); // Note: A += B means A = A + B
74        hashA[1] += (hashA[0] + byPos * 31349);
75        hashA[2] += (hashA[1] - byPos * 101302);
76        hashA[3] += (byPos * 79001);
77      }
78
79      ret = (hashA[0] + hashA[2]) + (hashA[1] * hashA[3]);
80      if (ret < 0) ret *= -1;
81      return ret;
82    }
83
84    private static String reductionFunction(long val, int round) {
        ↪ // Note that for the first function call "round" has to be 0,
85      String car, out;
          ↪ // and has to be incremented by one with every subsequent call.
86      int i;
          ↪ // I.e. "round" created variations of the reduction function.
87      char dat;
88
89      car = new String("0123456789ABCDEFGHIJKLMNOPQRSTUNVXYZabcdefghijklmnopqrstuvwxyz!#");
90      out = new String("");
91
92      for (i = 0; i < 8; i++) {
93        val -= round;
94        dat = (char) (val % 63);
95        val = val / 83;
96        out = out + car.charAt(dat);
97      }
98
99      return out;
100   }
101 }
```

2 Problem 2 code

```java
/**
 *  This class provides functionality to build rainbow tables (with a different reduction function per
  consist of the symbols "a .. z", "A .. Z", "0 .. 9", "!" and "#" (64 symbols in total).
  Properly used, it creates the following value pairs (start value - end value) after 10,000 iteration
      start value  -  end value
      Kermit12        lsXcRAuN
      Modulus!        L2rEsY8h
      Pigtail1        RONoLfOw
      GalwayNo        9PZjwF5c
      Trumpets        !oeHRZpK
      HelloPat        dkMPG7!U
      pinky##!        eDx58HRq
      01!19!56        vJ9OePjV
      aaaaaaaa        rLtVvpQS
      036abgH#        klQ6IeQJ
    *
    * @author Michael Schukat
    * @version 1.0
    */
public class RainbowTableFind
{
  /**
   * Constructor, not needed for this assignment
   */
  public RainbowTableFind() {

  }

  public static void main(String args[]) {
    long res = 0;
    int i, myindex;
    String start, current;
    /*List of Hashes to look for instances of*/
    long hashes[] =
     ↪ {8952106018744312l4L,750105908431234638L,1111111111156649 32L,9779842613436524 99L};
    /*List of start values for rainbow tables*/
    String hashchains[] =
     ↪ {"Kermit12","Modulus!","Pigtail1","GalwayNo","Trumpets","HelloPat","pinky##!","01!19!56" "aaaaa
    for (int j = 0; j < hashchains.length; j++) {
      start = hashchains[j]; //Check jth hashchain
      System.out.println("Start: "+start);
      current = start;
      /*0th run by doing this, it allows to check the hash of the 10000th string*/
      res = hashFunction(current); //check Hash
        myindex = indexOfLongArray(hashes, res); /*Checks for collision here*/
        if(myindex != -1) {
          /*Outputs collision*/
          System.out.println("String found for " + hashes[myindex] +" : " + current +" in
           ↪ Chain: "+start);
        }
      for(i = 0; i < 10000; i++) {
        current = reductionFunction(res, i);
        res = hashFunction(current); //check Hash
        myindex = indexOfLongArray(hashes, res); /*Checks for collision here*/
        if(myindex != -1) {
          /*Outputs collision*/
          System.out.println("String found for " + hashes[myindex] +" : " + current +" in
           ↪ Chain: "+start);
```

```
      }
    }
    System.out.println("End: " +current+"\n");
  }
}

private static long hashFunction(String s){
  long ret = 0;
  int i;
  long[] hashA = new long[]{1, 1, 1, 1};

  String filler, sIn;

  int DIV = 65536;

  filler = new
   ↪  String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");

  sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
  sIn = sIn.substring(0, 64); // // Limit string to first 64 characters

  for (i = 0; i < sIn.length(); i++) {
    char byPos = sIn.charAt(i); // get i'th character
    hashA[0] += (byPos * 17111); // Note: A += B means A = A + B
    hashA[1] += (hashA[0] + byPos * 31349);
    hashA[2] += (hashA[1] - byPos * 101302);
    hashA[3] += (byPos * 79001);
  }

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

private static String reductionFunction(long val, int round) {
 ↪  // Note that for the first function call "round" has to be 0,
  String car, out;
   ↪  // and has to be incremented by one with every subsequent call.
  int i;
   ↪  // I.e. "round" created variations of the reduction function.
  char dat;

  car = new String("0123456789ABCDEFGHIJKLMNOPQRSTUNVXYZabcdefghijklmnopqrstuvwxyz!#");
  out = new String("");

  for (i = 0; i < 8; i++) {
    val -= round;
    dat = (char) (val % 63);
    val = val / 83;
    out = out + car.charAt(dat);
  }

  return out;
}
/*Made this so I can find check if the hash is in the array of hashes */
public static int indexOfLongArray(long longArray[],long number) {
  int returnValue = -1;
    for(int i = 0; i < longArray.length; i++) {
      if(longArray[i] == number) return i;
    }
```

```
111         return returnValue;
112       }
113     }
```