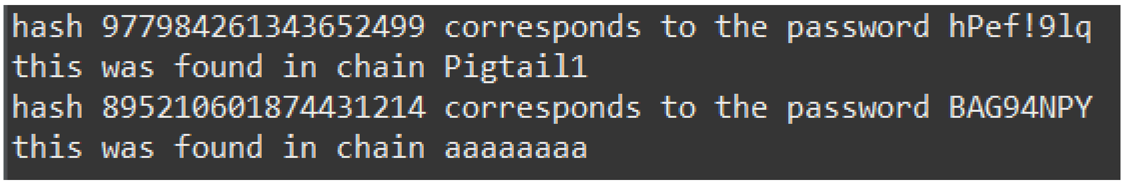
Assignment 2 - 21347021

Éidhne Kennedy

Passwords found:



```
hash 977984261343652499 corresponds to the password hPef!9lq
this was found in chain Pigtail1
hash 895210601874431214 corresponds to the password BAG94NPY
this was found in chain aaaaaaaa
```

Source code:

```
package ct255_assignment_2;

/*  CT255 Assignment 2
 *  This class provides functionality to build rainbow tables (with a different reduction
function per round) for 8 character long strings, which
    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 iterations of hashFunction() and reductionFunction():
          start value  -  end value
          Kermit12        lsXcRAuN
          Modulus!        L2rEsY8h
          Pigtail1        R0NoLf0w
          GalwayNo        9PZjwF5c
          Trumpets        !oeHRZpK
          HelloPat        dkMPG7!U
          pinky##!        eDx58HRq
          01!19!56        vJ90ePjV
          aaaaaaaa        rLtVvpQS
          036abgH#        klQ6IeQJ


 *
 * @author Michael Schukat
 * @version 1.0
 */
public class RainbowTables {
    public static void main(String[] args) {
              long res = 0;
              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 {
                            String[] chains = {
                                          "Kermit12",
                                   "Modulus!",
                                   "Pigtail1",
                                   "GalwayNo",
                                   "Trumpets",
                                   "HelloPat",
                                   "pinky##!",
                                   "01!19!56",
                                   "aaaaaaaa",
                                   "036abgH#"
                            };
                            // list of possible chain beginnings
```

```
                              long[] hashes = {
                                             895210601874431214L,
                                             750105908431234638L,
                                             111111111115664932L,
                                             977984261343652499L
                              };
                              // hashes to look for

                              String working;
                              long working_hash;

                              for (int i = 0; i < chains.length; i++) {
                                     working = chains[i];  // start a new chain

                                     for (int j = 0; j < 10000; j++) {
                                             working_hash = hashFunction(working);
                                             // hash the current 8-char string

                                             for (int k = 0; k < 4; k++) {
                                                    //  compare  (note:  inefficient,  could  be
refactored)
                                                    if (working_hash == hashes[k]) {
                                                           System.out.println("hash    "    +
hashes[k] + " corresponds to the password " + working);
                                                           System.out.println("this      was
found in chain " + chains[i]);
                                                    }
                                             }

                                             working = reductionFunction(working_hash, j);
                                             // reduce to another 8-char string and continue
                                     }
                              }




                      }
               }
               else { // No <input>
                      System.out.println("Use: RainbowTable <Input>");
               }
        }

    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("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");

        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;
    }
}
```