HW_1

EECS 565 : Introduction to Information and Computer Security

Satya Ashok Dowluri
dowlurisatyaashok@ku.edu

Professor
Dr. Han Wang

Electrical Engineering and Computer Science
University of Kansas
10 February 2024

Description:

This homework is for us to get familiar with the brute force attack of guessing passwords that are MD5 hashed. There are a total of 3 tasks in this work. A distinct hash is given for each of the tasks to decrypt.

Task 1:
Hash value: 94d9e03c11395841301a7aee967864ec
Password characteristics: 14 character long. Each character can be A-Z or a-z or 0-9 ascii characters with repetitions allowed.
Possible number of passwords: (26+10+26)^14 = 1.24*10^25

Task 2:
Hash value: f593def02f37f3a6d57bcbc9480a3316
Password characteristics: 11 character long. First 4 characters from the left can be any character of A-Z with repetitions allowed. The next 3 characters can be 0-9 with repetitions allowed. The final 4 characters can be from the pool of a-z with repetitions allowed.
Possible number of passwords: 26^4 * 10^3 * 26^4 = 2.08 * 10^14

Task 3:
Hash value: bfb2c12706757b8324368de6a365338b
Password characteristics: 11 character long. 7 out of 11 characters come from ascii characters A-Z and the rest 4 characters will be clubbed together in the same order. Those 4 characters are "1234"
Possible number of passwords: 26^7 * 8 = 6.4 * 10^10

Programming language: Java (SDK 21)
IDE: IntelliJ IDEA CE

Proposed solutions

For brute forcing, we have to check all the possible passwords one by one. Need to take one password at a time and generate the MD5 hash for it and then check it with the hash value of the specific task. This is the general idea to crack passwords in this task. I have formulated two solutions to crack the password. First solution is using recursion. The first solution can also be done iteratively using n for-loops for a n character password (for some tasks). But the code looks clean and neat using recursion and reduces the time complexity overall. Second solution is storing all the possible combination of passwords in a .txt file and accessing it each line iteratively. Both these solutions produce valid results. Albeit the second solution requires us to store all the possible passwords in a file. This would consume lot of memory space on the server running.

Solution 1:

Task 1:

A character array is formed with all the possible ascii characters, and that array is called recursively using generate() function. Inside the generate() function, once the password is constructed, hashGen() function is called to generate the hash and that hash is compared with the Task 1 hash. This process continues either until the hash is matched or until all the possibilities are finished and the password is found.

Task 2:

A character array is formed with A-Z ascii characters, and this array is given as an input to the generate() function. In this generate() function, another character array of 0-9 characters is formed. A concatenated string of 4 first array's characters and second array's 3 characters is formed and this string is given as an input to the generateSecond() function. In this function, there is another character array with a-z ascii characters. A 4-character string is formed here and this is concatenated with the string that is passed from the previous generate() function. Now, the produced string will be of 11-character long. This string (password) is then passed through the hashGen() function to generate it's MD5 hash and that hash is checked with the Task 2 hash. This generation of passwords will continue until the hash is matched or until all the possibilities have been checked.

Task 3:

A character array is formed with A-Z ascii characters, and this array is given as an input to generate() function. Since the password is 11 characters long, and the string formed is 7 characters, "1234" will have 8 places to be placed in that 7-length string. So, concatenate every possibility respectively and the 11-character password string is passed to hashGen() whose output will be checked against the Task 3 hash. This generation and checking of passwords will continue until a hash is matched or until all the possibilities have been checked.

Solution 2:

Task 1:

This solution is about storing all the possible passwords in a file and then checking them iteratively. Cracking this first task using this solution is near impossible with my computational availability. Because the file which contains all the possible passwords (1.24 * 10^25) will take up an enormous 350 petabytes of memory. So, for Task 1, first solution is the most practical one of the two.

Task 2:

A dictionary generator in Linux "crunch" is used to generate all the possible passwords. Two files task2_upper7.txt (3.4 gigabytes) and task2_lower4.txt (2.2 megabytes) have been created to store the two parts of the password combinations. A buffer reader is used in the code to iteratively go through each line in the files, and concatenate strings from both the files and that forms the password. This password is then fed into hashGen() and that hash is compared against Task 2 hash. This process is done until a hash matches or the input file task2_upper7 exhausts.

Task 3:

A dictionary generator in Linux “crunch” is used to generate 7 character string consisting of A-Z characters and those are stored in a file task3_upper7.txt (60 gigabytes). Each line of this file is a 7-character string. In the code, “1234” is looped 8 times to be placed in 8 places in that 7-character string and the resulted string (password) is fed into hashGen() whose result is compared against task 3 hash. This process continues either until a match is found or all the strings in the task3_upper.txt file are exhausted.

Codes:

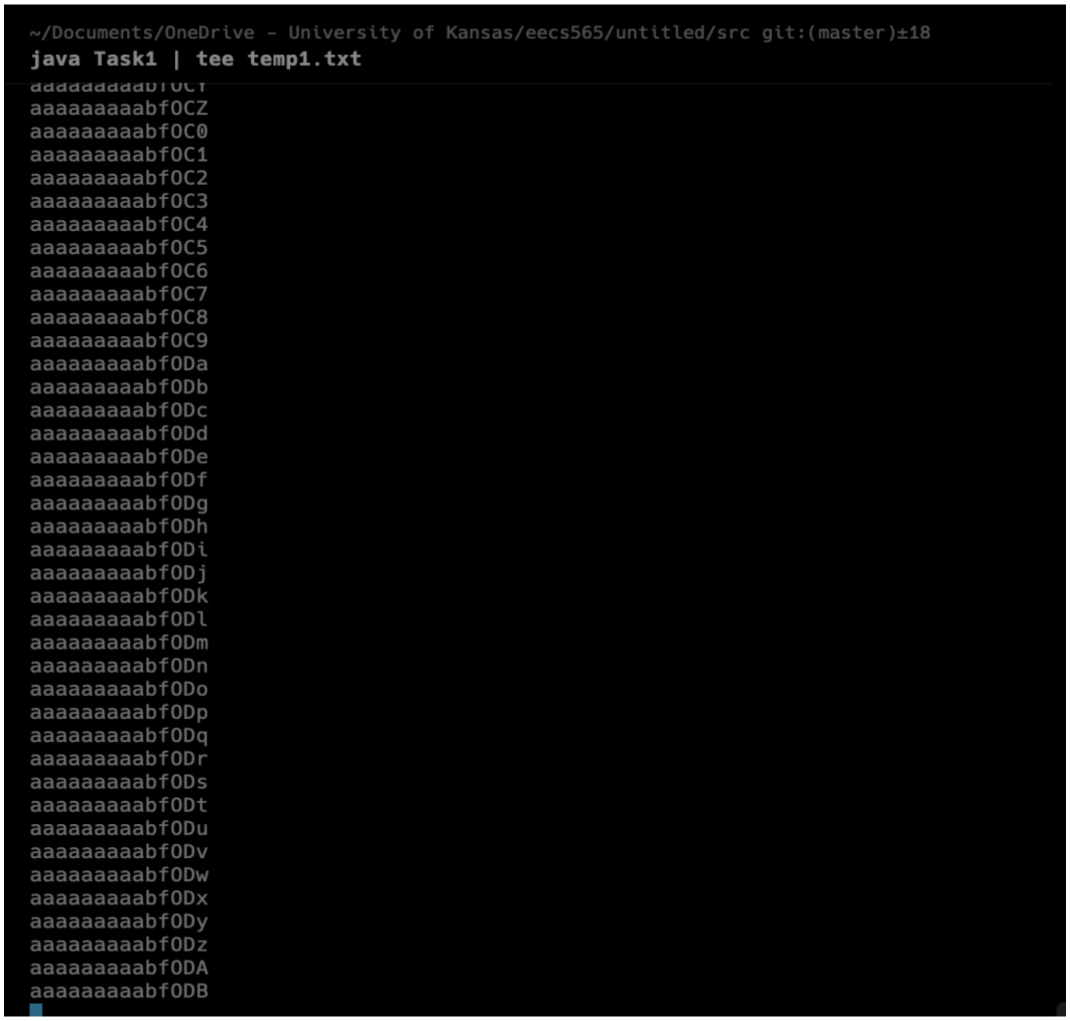
Task1.java -> Solution 1 task 1
Task2.java -> Solution 1 task2
Task3.java -> Solution 1 task3
Main.java -> Solution 2 task2 & task 3

Results:

The computational power with me is not enough to be able to crack the Task 1 password. But for task 2, task 3, I was able to crack their passwords.
Password for Task 2: **AAAA123dddd**
Password for Task 3: **ABC1234GHIJ**

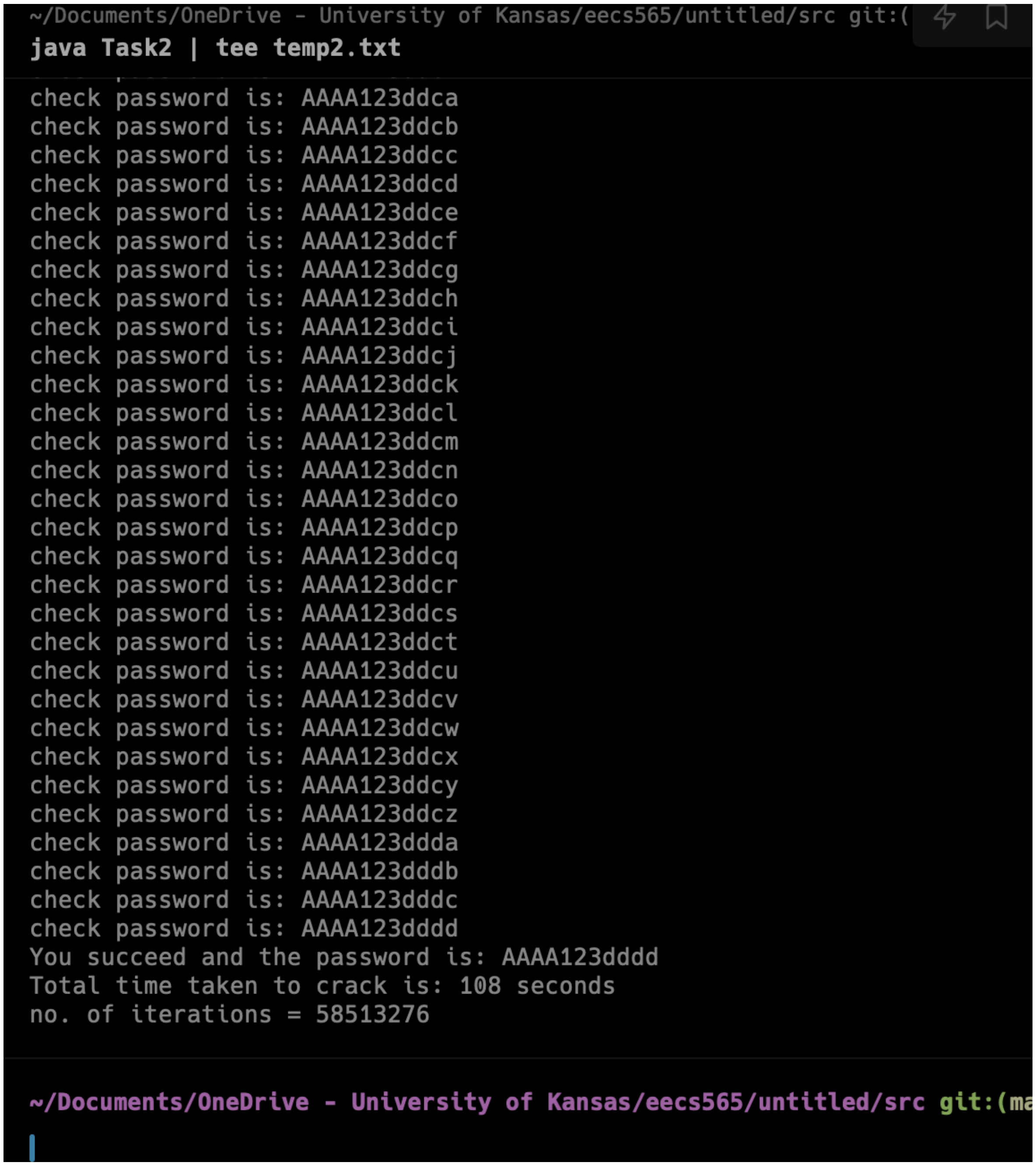
Tasks	Solution 1 (seconds)	Solution 2 (seconds)
Task 1	Unknown	Unknown
Task 2	108	122
Task 3	143	218

Task 1(solution 1):



```
~/Documents/OneDrive - University of Kansas/eecs565/untitled/src git:(master)±18
java Task1 | tee temp1.txt
aaaaaaaaabfOCY
aaaaaaaaabfOCZ
aaaaaaaaabfOC0
aaaaaaaaabfOC1
aaaaaaaaabfOC2
aaaaaaaaabfOC3
aaaaaaaaabfOC4
aaaaaaaaabfOC5
aaaaaaaaabfOC6
aaaaaaaaabfOC7
aaaaaaaaabfOC8
aaaaaaaaabfOC9
aaaaaaaaabfODa
aaaaaaaaabfODb
aaaaaaaaabfODc
aaaaaaaaabfODd
aaaaaaaaabfODe
aaaaaaaaabfODf
aaaaaaaaabfODg
aaaaaaaaabfODh
aaaaaaaaabfODi
aaaaaaaaabfODj
aaaaaaaaabfODk
aaaaaaaaabfODl
aaaaaaaaabfODm
aaaaaaaaabfODn
aaaaaaaaabfODo
aaaaaaaaabfODp
aaaaaaaaabfODq
aaaaaaaaabfODr
aaaaaaaaabfODs
aaaaaaaaabfODt
aaaaaaaaabfODu
aaaaaaaaabfODv
aaaaaaaaabfODw
aaaaaaaaabfODx
aaaaaaaaabfODy
aaaaaaaaabfODz
aaaaaaaaabfODA
aaaaaaaaabfODB
```

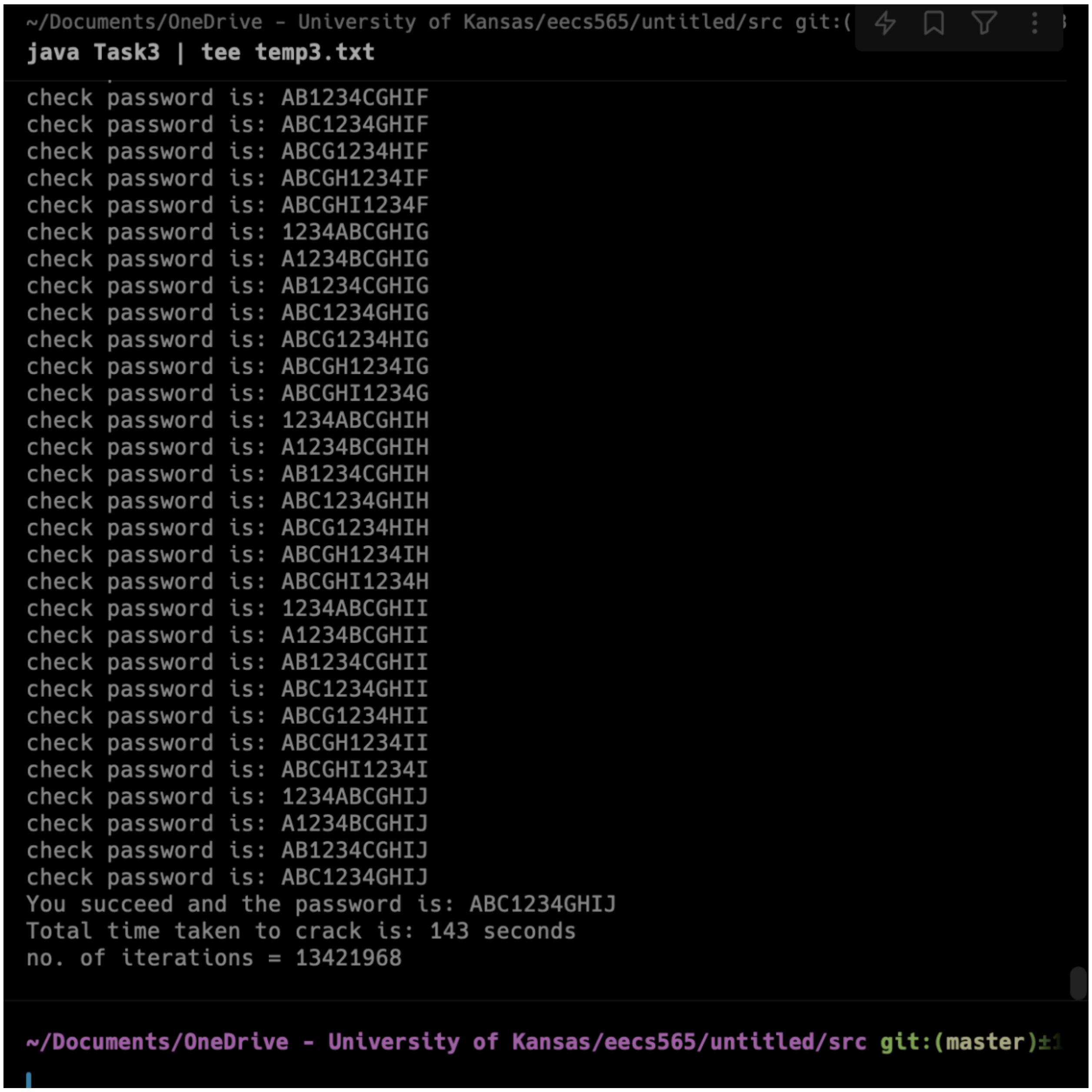
Task 2 Cracked (Solution 1):



```
~/Documents/OneDrive - University of Kansas/eecs565/untitled/src git:(
java Task2 | tee temp2.txt
check password is: AAAA123ddca
check password is: AAAA123ddcb
check password is: AAAA123ddcc
check password is: AAAA123ddcd
check password is: AAAA123ddce
check password is: AAAA123ddcf
check password is: AAAA123ddcg
check password is: AAAA123ddch
check password is: AAAA123ddci
check password is: AAAA123ddcj
check password is: AAAA123ddck
check password is: AAAA123ddcl
check password is: AAAA123ddcm
check password is: AAAA123ddcn
check password is: AAAA123ddco
check password is: AAAA123ddcp
check password is: AAAA123ddcq
check password is: AAAA123ddcr
check password is: AAAA123ddcs
check password is: AAAA123ddct
check password is: AAAA123ddcu
check password is: AAAA123ddcv
check password is: AAAA123ddcw
check password is: AAAA123ddcx
check password is: AAAA123ddcy
check password is: AAAA123ddcz
check password is: AAAA123ddda
check password is: AAAA123dddb
check password is: AAAA123dddc
check password is: AAAA123dddd
You succeed and the password is: AAAA123dddd
Total time taken to crack is: 108 seconds
no. of iterations = 58513276

~/Documents/OneDrive - University of Kansas/eecs565/untitled/src git:(ma
```

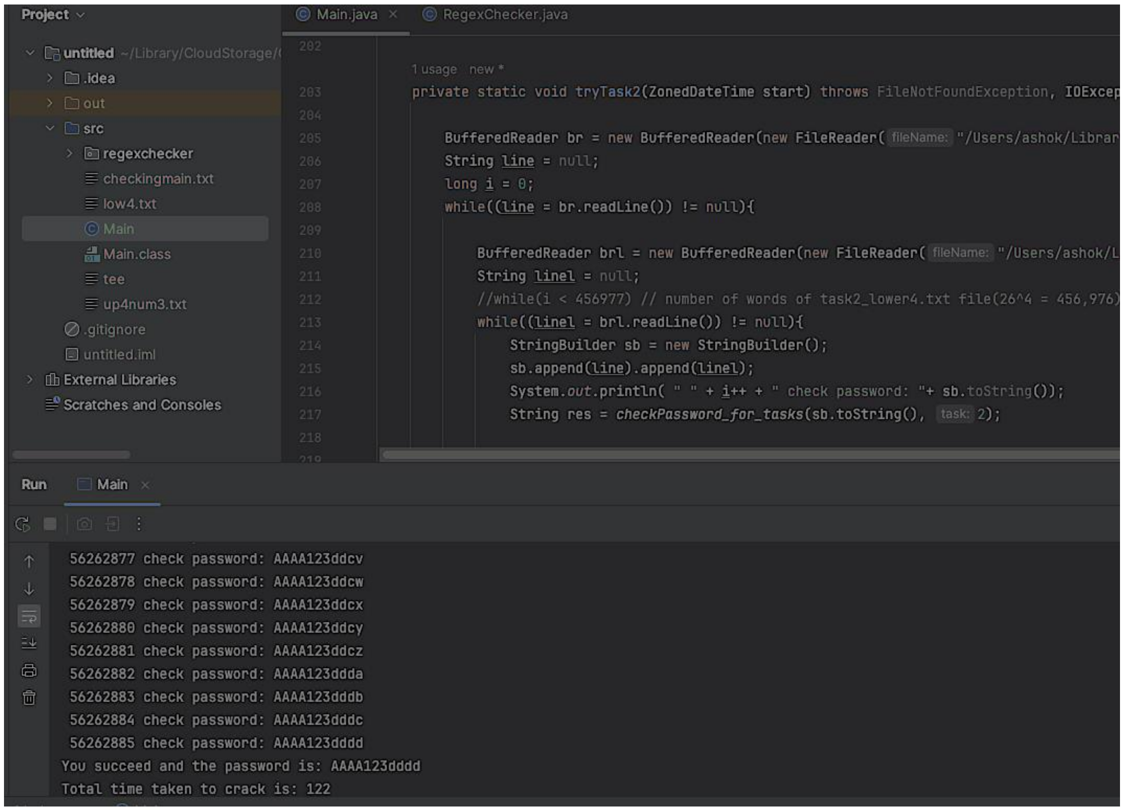
Task 3 Cracked (Solution 1):



```
~/Documents/OneDrive - University of Kansas/eecs565/untitled/src git:(
java Task3 | tee temp3.txt
check password is: AB1234CGHIF
check password is: ABC1234GHIF
check password is: ABCG1234HIF
check password is: ABCGH1234IF
check password is: ABCGHI1234F
check password is: 1234ABCGHIG
check password is: A1234BCGHIG
check password is: AB1234CGHIG
check password is: ABC1234GHIG
check password is: ABCG1234HIG
check password is: ABCGH1234IG
check password is: ABCGHI1234G
check password is: 1234ABCGHIH
check password is: A1234BCGHIH
check password is: AB1234CGHIH
check password is: ABC1234GHIH
check password is: ABCG1234HIH
check password is: ABCGH1234IH
check password is: ABCGHI1234H
check password is: 1234ABCGHII
check password is: A1234BCGHII
check password is: AB1234CGHII
check password is: ABC1234GHII
check password is: ABCG1234HII
check password is: ABCGH1234II
check password is: ABCGHI1234I
check password is: 1234ABCGHIJ
check password is: A1234BCGHIJ
check password is: AB1234CGHIJ
check password is: ABC1234GHIJ
You succeed and the password is: ABC1234GHIJ
Total time taken to crack is: 143 seconds
no. of iterations = 13421968

~/Documents/OneDrive - University of Kansas/eecs565/untitled/src git:(master)±
```

Task 2 cracked (Solution 2):



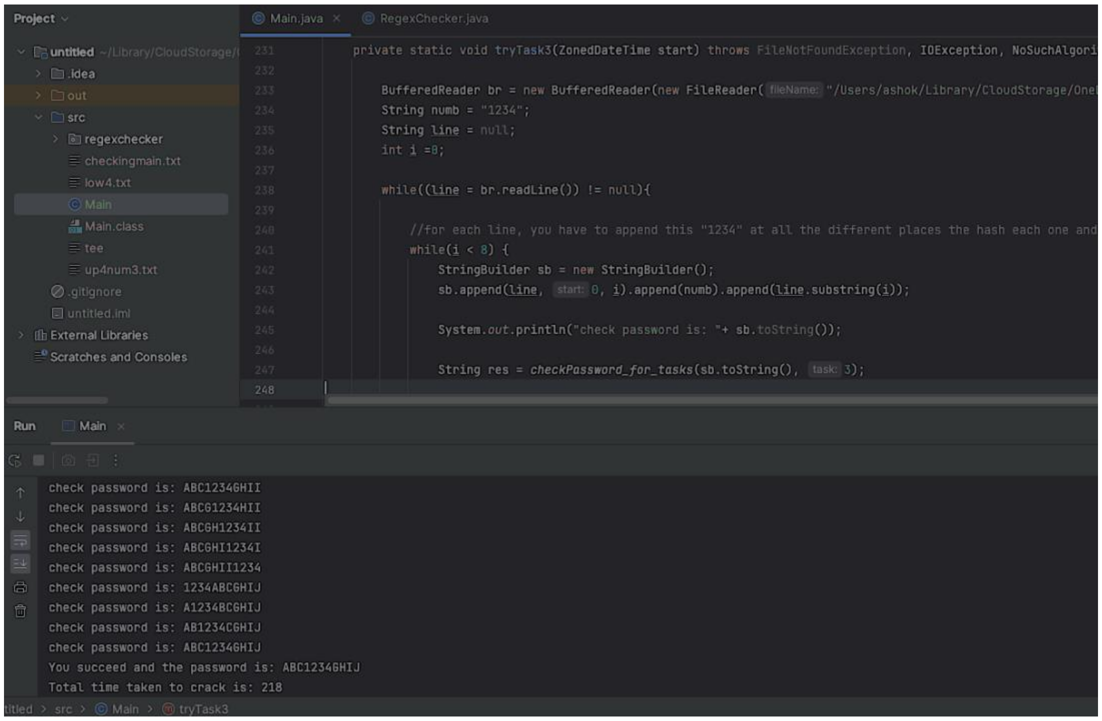
```java
    1 usage  new *
    private static void tryTask2(ZonedDateTime start) throws FileNotFoundException, IOExcep

        BufferedReader br = new BufferedReader(new FileReader( fileName: "/Users/ashok/Librar
        String line = null;
        long i = 0;
        while((line = br.readLine()) != null){

            BufferedReader brl = new BufferedReader(new FileReader( fileName: "/Users/ashok/L
            String linel = null;
            //while(i < 456977) // number of words of task2_lower4.txt file(26^4 = 456,976)
            while((linel = brl.readLine()) != null){
                StringBuilder sb = new StringBuilder();
                sb.append(line).append(linel);
                System.out.println( " " + i++ + " check password: "+ sb.toString());
                String res = checkPassword_for_tasks(sb.toString(),  task: 2);
```

```
 56262877 check password: AAAA123ddcv
 56262878 check password: AAAA123ddcw
 56262879 check password: AAAA123ddcx
 56262880 check password: AAAA123ddcy
 56262881 check password: AAAA123ddcz
 56262882 check password: AAAA123ddda
 56262883 check password: AAAA123dddb
 56262884 check password: AAAA123dddc
 56262885 check password: AAAA123dddd
You succeed and the password is: AAAA123dddd
Total time taken to crack is: 122
```

Task 3 cracked (solution 2):



```java
    private static void tryTask3(ZonedDateTime start) throws FileNotFoundException, IOException, NoSuchAlgori

        BufferedReader br = new BufferedReader(new FileReader( fileName: "/Users/ashok/Library/CloudStorage/One
        String numb = "1234";
        String line = null;
        int i =0;

        while((line = br.readLine()) != null){

            //for each line, you have to append this "1234" at all the different places the hash each one and
            while(i < 8) {
                StringBuilder sb = new StringBuilder();
                sb.append(line,  start: 0, i).append(numb).append(line.substring(i));

                System.out.println("check password is: "+ sb.toString());

                String res = checkPassword_for_tasks(sb.toString(),  task: 3);
```

```
check password is: ABC1234GHII
check password is: ABCG1234HII
check password is: ABCGH1234II
check password is: ABCGHI1234I
check password is: ABCGHII1234
check password is: 1234ABCGHIJ
check password is: A1234BCGHIJ
check password is: AB1234CGHIJ
check password is: ABC1234GHIJ
You succeed and the password is: ABC1234GHIJ
Total time taken to crack is: 218
```