

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: $(26+10+26)^{14} = 1.24 \cdot 10^{25}$

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 its 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: **ABC1234GHUJ**

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
```

```
aaaaaaaaabf0Ct  
aaaaaaaaabf0CZ  
aaaaaaaaabf0C0  
aaaaaaaaabf0C1  
aaaaaaaaabf0C2  
aaaaaaaaabf0C3  
aaaaaaaaabf0C4  
aaaaaaaaabf0C5  
aaaaaaaaabf0C6  
aaaaaaaaabf0C7  
aaaaaaaaabf0C8  
aaaaaaaaabf0C9  
aaaaaaaaabf0Da  
aaaaaaaaabf0Db  
aaaaaaaaabf0Dc  
aaaaaaaaabf0Dd  
aaaaaaaaabf0De  
aaaaaaaaabf0Df  
aaaaaaaaabf0Dg  
aaaaaaaaabf0Dh  
aaaaaaaaabf0Di  
aaaaaaaaabf0Dj  
aaaaaaaaabf0Dk  
aaaaaaaaabf0Dl  
aaaaaaaaabf0Dm  
aaaaaaaaabf0Dn  
aaaaaaaaabf0Do  
aaaaaaaaabf0Dp  
aaaaaaaaabf0Dq  
aaaaaaaaabf0Dr  
aaaaaaaaabf0Ds  
aaaaaaaaabf0Dt  
aaaaaaaaabf0Du  
aaaaaaaaabf0Dv  
aaaaaaaaabf0Dw  
aaaaaaaaabf0Dx  
aaaaaaaaabf0Dy  
aaaaaaaaabf0Dz  
aaaaaaaaabf0DA  
aaaaaaaaabf0DB
```

Task 2 Cracked (Solution 1):

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

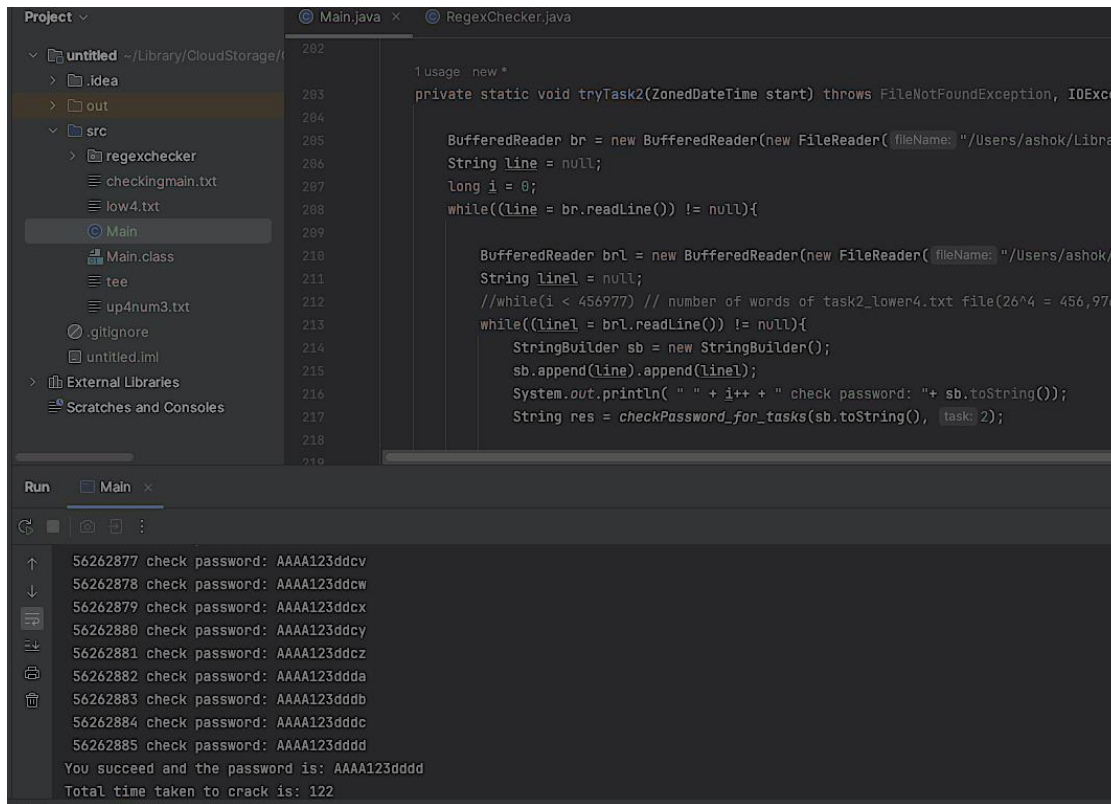
~/Documents/OneDrive - University of Kansas/eecs565/untitled/src git:(ma
```

Task 3 Cracked (Solution 1):

```
~/Documents/OneDrive - University of Kansas/eecs565/untitled/src git:(master) 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)±1
```

Task 2 cracked (Solution 2):



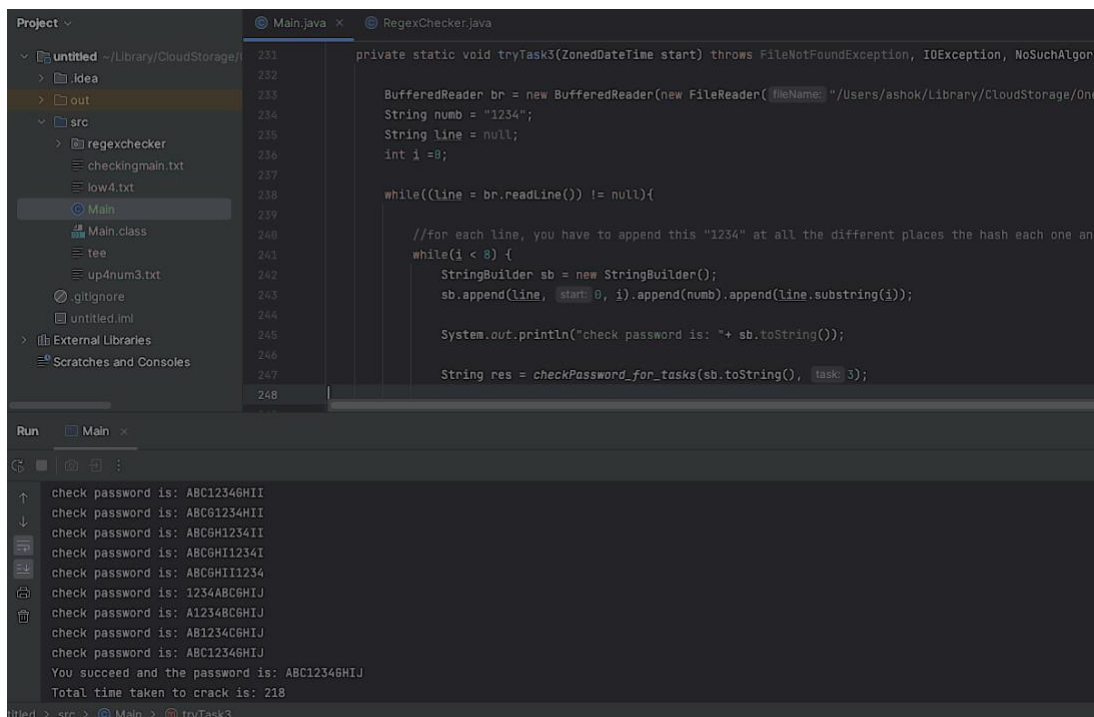
```
Project
└─ untitled - ~/Library/CloudStorage/OneDrive
   └─ idea
      └─ out
         └─ src
            └─ regexchecker
               ├── checkingmain.txt
               ├── low4.txt
               └─ Main
                  ├── Main.class
                  ├── tee
                  └─ up4num3.txt
            ├── .gitignore
            ├── untitled.iml
            └─ External Libraries
               └─ Scratches and Consoles

Main.java
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319

RegexChecker.java
1 usage: new *
private static void tryTask2(ZonedDateTime start) throws FileNotFoundException, IOException {
    BufferedReader br = new BufferedReader(new FileReader("Users/ashok/Library/CloudStorage/OneDrive/idea/out/src/regexchecker/low4.txt"));
    String line = null;
    long i = 0;
    while((line = br.readLine()) != null){
        BufferedReader brl = new BufferedReader(new FileReader("Users/ashok/Library/CloudStorage/OneDrive/idea/out/src/regexchecker/low4.txt"));
        String line1 = null;
        //while(i < 456977) // number of words of task2_lower4.txt file(26^4 = 456,976)
        while((line1 = brl.readLine()) != null){
            StringBuilder sb = new StringBuilder();
            sb.append(line).append(line1);
            System.out.println(" " + i++ + " check password: " + sb.toString());
            String res = checkPassword_for_tasks(sb.toString(), task: 2);
        }
    }
}
```

```
Run Main
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):



```
Project
└─ untitled - ~/Library/CloudStorage/OneDrive
   └─ idea
      └─ out
         └─ src
            └─ regexchecker
               ├── checkingmain.txt
               ├── low4.txt
               └─ Main
                  ├── Main.class
                  ├── tee
                  └─ up4num3.txt
            ├── .gitignore
            ├── untitled.iml
            └─ External Libraries
               └─ Scratches and Consoles

Main.java
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

RegexChecker.java
private static void tryTask3(ZonedDateTime start) throws FileNotFoundException, IOException, NoSuchAlgorithmException {
    BufferedReader br = new BufferedReader(new FileReader("Users/ashok/Library/CloudStorage/OneDrive/idea/out/src/regexchecker/low4.txt"));
    String numb = "1234";
    String line = null;
    int i = 8;
    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);
        }
    }
}
```

```
Run Main
check password is: ABC12346HII
check password is: ABC61234HII
check password is: ABC6H1234II
check password is: ABC6HI1234I
check password is: ABC6HII1234
check password is: 1234ABC6HIJ
check password is: A12348C6HIJ
check password is: AB1234C6HIJ
check password is: ABC12346HIJ
You succeed and the password is: ABC12346HIJ
Total time taken to crack is: 218
```