Password Security Checker

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

Analisi 4

1.4 Analisi del dominio 4

1.5 Analisi e specifica dei requisiti 4

1.6 Diagrammi 7

1.6.1 Activity Diagram 7

1.7 Pianificazione 8

1.8 Analisi dei mezzi 9

1.8.1 Software 9

1.8.2 Hardware 9

2 Progettazione 10

2.1 Design dell’architettura del sistema 10

2.2 Design dei dati e database 10

2.3 Design delle interfacce 11

2.4 Design procedurale 11

3 Implementazione 12

3.1 L’inizio 12

3.1 Attirbuti 12

3.2 Metodi 12

3.2.1 PasswordSecurityChecker(String[] argumetns) 12

3.2.2 loadMostKnownPasswords() 13

3.2.3 add(String word) 13

3.2.4 addCombos(String w1, String w2, String word) 13

3.2.5 addSubStrings(String word) 14

3.2.6 addTempCombosToCombos() 14

3.2.7 makeCombos() 14

3.2.8 printTriesAndTime() 15

3.2.9 finalPrint() 15

3.2.10 endForce(String s) 15

3.2.11 dictionaryForce() 15

3.2.12 argumentsForce() 16

3.2.13 bruteForce(String keys) 16

3.2.14 findPassword() 16

4 Test 17

4.1 Protocollo di test 17

4.2 Risultati test 18

4.3 Mancanze/limitazioni conosciute 18

5 Consuntivo 20

6 Conclusioni 21

6.1 Sviluppi futuri 21

6.2 Considerazioni personali 21

7 Bibliografia 22

7.1 Sitografia 22

Allegati 23

# Introduzione

## Informazioni sul progetto

Sono un allievo della Scuola d’Arti e Mestieri a Trevano, della sezione informatica, nella classe I3AA, e mi chiamo Xavier Horisberger, e ho il ruolo di produttore. Il progetto è supervisionato dai docenti Geo Petrini e Luca Muggiasca che hanno il ruolo di clienti.

Il progetto è iniziato il 09.09.2021 e la consegna è per il 23.12.2021.

## Abstract

Questa documentazione contiene tutte le informazioni riguardante il progetto Password Security Checker. Che ha lo scopo di scoprire una password fornita dall’utente utilizzando diverse tecniche: dictionary attack (scoprire la password confrontandola con le 100000 password più comunemente usate al mondo), brute force (tentare tutte le combinazioni di caratteri possibili per scoprire la password) e un attacco basato sugli argomenti aggiuntivi forniti dall’utente quando inserisce la password, ovvero nome, cognome, data di nascita e una parola aggiuntiva che può essere quello che vuole l’utente (questi argomenti sono tutti opzionali).

L’obbiettivo di questo progetto è che una volta provati tutti questi attacchi bisogna aver scoperto la password.

L’obbiettivo è stato raggiunto, il dictionary attack è molto veloce e funziona correttamente, la scoperta della password tramite gli argomenti passati dall’utente può durare fino a 40 secondi circe, il che è un po’ lento, ma considerando che genera al massimo sulle 100000 combinazioni non è male, mentre il brute forze dura molto se la password inserita è troppo lunga, infatti si spera di scoprila prima di dover ricorrere ad esso, però tutti gli attacchino sono funzionanti, indipendentemente da quanto ci mettono.

## Scopo

Lo scopo di questo progetto è creare un applicativo che passati alcuni dati di partenza, o anche senza, che data una password inserita dall’utente, provi a scoprirla usando diverse combinazioni tra i dati di partenza, una banca dati (o file di testo) con le password più conosciute, e implementando come ultima opzione un algoritmo di brute force. Alla fine una volta trovata la password deve essere comunicato all’utente quanti tentativi e quanto tempo ha impiegato a trovarla.

## Analisi

## Analisi del dominio

Questo prodotto può essere usato su qualsiasi sistema operativo, che ha installato java, e serve una conoscenza minima di come eseguire un programma jar da linea di comando, fornendo ad esso argomenti.

Gli argomenti passati al programma possono essere quanti l’utente vuole, ma il programma terrà in considerazione solo i primi 5, che sono nel seguente ordine: password, nome, cognome, data di nascita e parola extra, che ad esempio potrebbe essere il nome dell’hobby o il segno zodiacale, eccetera.

La data deve anche essere nel seguente formato: gg.mm.aaaa, altrimenti non viene presa in considerazione.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-01** | |
| **Nome** | Inserimento password |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Deve essere possibile inserire una password. |

|  |  |
| --- | --- |
| **ID: REQ-02** | |
| **Nome** | Parametri aggiuntivi tralasciabili |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Possibilità di inserire solo il nome, solo nome e cognome, solo nome cognome e data di nascita o si possono inserire tutti. |

|  |  |
| --- | --- |
| **ID: REQ-03** | |
| **Nome** | Password ha una lunghezza massima |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | La lunghezza massima della password è di 20 caratteri. |

|  |  |
| --- | --- |
| **ID: REQ-04** | |
| **Nome** | Caratteri della password |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | La password può essere composta solamente dai caratteri premibili sulle tastiere Svizzere. |

|  |  |
| --- | --- |
| **ID: REQ-05** | |
| **Nome** | Usare le password più conosciute al mondo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Controllo con le password più conosciute (usando file di testo) |

|  |  |
| --- | --- |
| **ID: REQ-06** | |
| **Nome** | Scoprire la password |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Cercare di forzare la password inserita |

|  |  |
| --- | --- |
| **ID: REQ-07** | |
| **Nome** | Indicare i numeri di tentativi svolti |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Una volta che il programma è terminato deve indicare quanti tentativi sono stati svolti |

|  |  |
| --- | --- |
| **ID: REQ-08** | |
| **Nome** | Indicare il tempo impiegato |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Una volta che il programma è terminato deve indicare quanto tempo ha impiegato a trovare la password |

|  |  |
| --- | --- |
| **ID: REQ-09** | |
| **Nome** | Stampa in esecuzione |
| **Priorità** | 3 |
| **Versione** | 1.0 |
| **Note** | Mentre il programma è in esecuzione deve stampare i tentativi attualmente svolti e quanto tempo è passato a quel punto, per far capire all’utente che il programma non si è bloccato o interrotto ma che è ancora in funzionamento |

**Spiegazione elementi tabella dei requisiti:**

**ID**: identificativo univoco del requisito

**Nome**: breve descrizione del requisito

**Priorità**: indica l’importanza di un requisito nell’insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

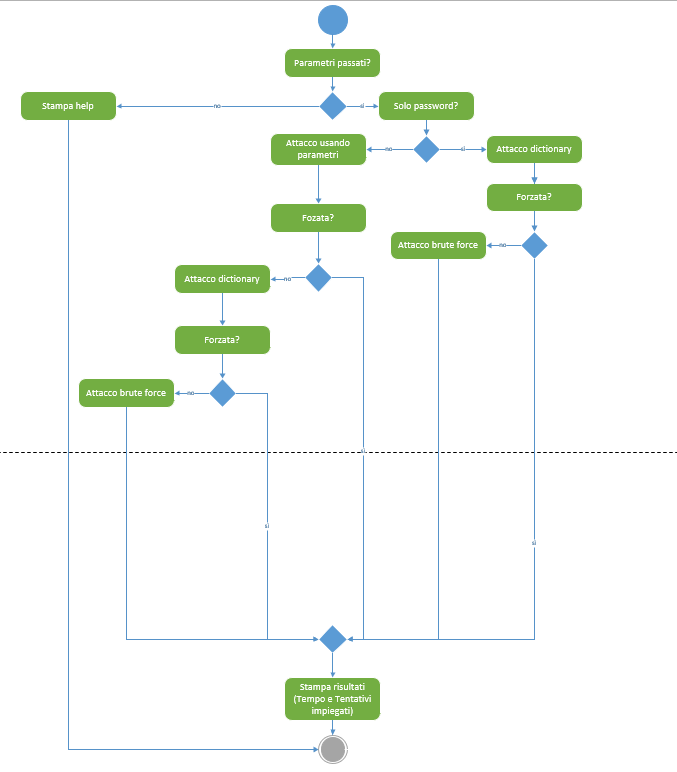
**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l’ultima versione, mentre le vecchie dovranno essere inserite nei diari.

**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.

## Diagrammi

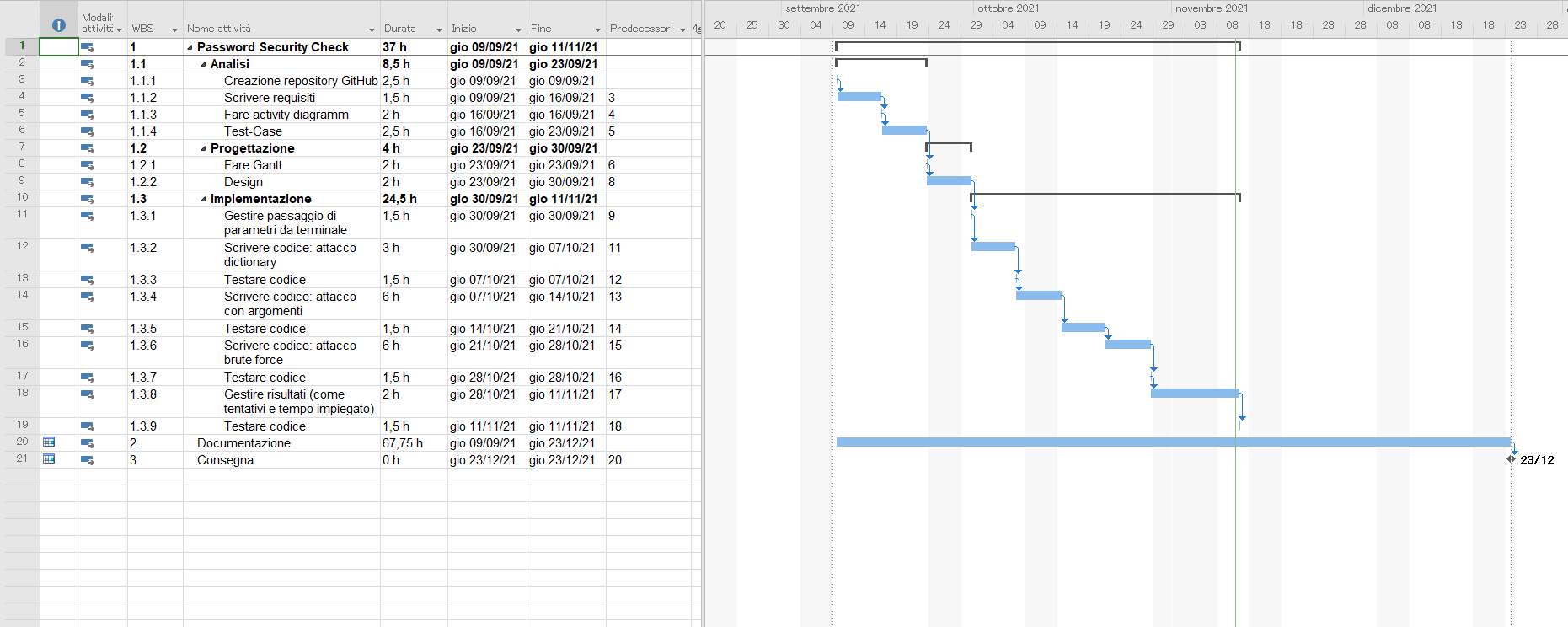
### Activity Diagram



L’activity diagram rappresenta in che ordine le azioni vengono eseguite, e le condizioni perché vengano eseguite,

.

## Pianificazione



## Analisi dei mezzi

Per la realizzazione del progetto è stato utilizzato un computer scolastico e l’applicativo NetBeans, con le seguenti specifiche:

### Software

OS Computer: Windows 10 Enterprise versione 20H2 (build OS 19042.870)

NetBeans: versione 12.4

### Hardware

Processore: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, 3601 Mhz, 4 core, 8 processori logici

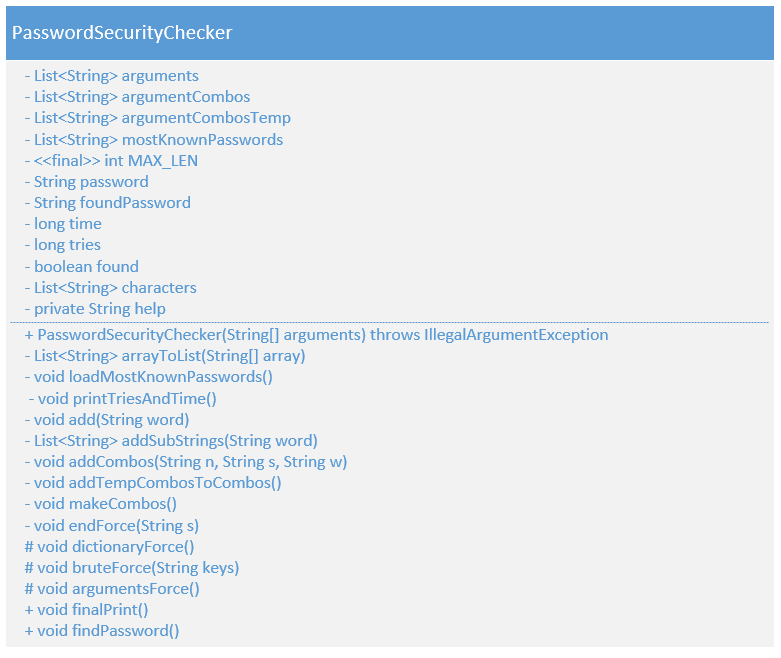
RAM: 16 GB

# Progettazione

Questo capitolo descrive esaustivamente come deve essere realizzato il prodotto fin nei suoi dettagli. Una buona progettazione permette all’esecutore di evitare fraintendimenti e imprecisioni nell’implementazione del prodotto.

## Design dell’architettura del sistema

Diagramma delle classi:



## Design dei dati e database

Le password più conosciute sono salvate in un file chiamato 100000-most-known-passwords.txt, e sono tutte una sotto l’altra.

## Design procedurale

Descrive i concetti dettagliati dell’architettura/sviluppo utilizzando ad esempio:

### Diagramma di flusso



# Implementazione

## L’inizio

All’inizio del lavoro ho creato un progetto con NetBeans creando la classe PasswordSecurityChecker, ovvero il programma principale del progetto, e il file 100000-most-known-passwords.txt che conterrà le 100000 password più conosciute al mondo.

Dopo ciò ho cercato su internet le 100000 password più conosciute al mondo trovando un file su github a questo indirizzo: <https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-100000.txt#L100000> che conteneva esattamente quello che cercavo. Ho preso il suo contenuto e l’ho messo nel mio file file 100000-most-known-passwords.txt.

In seguito ho iniziato a scivere codice nel file PasswordSecurityChecker.java:

## Attirbuti

Ho creato i seguenti attributi:

* arguments: lista di stringhe che conterrà i primi 4 argomenti passati dall’utente escludendo il primo (quindi gli argomenti da args[1] a args[4]).
* argumentCombos: lista di stringhe che conterrà le diverse combinazioni tra gli argomenti passati dall’utente.
* argumentCombosTemp: lista di stringhe che conterrà delle combinazioni tra argomenti che verranno poi aggiunte ad argumentCombos, essendo che non si possono fare aggiunte ad una lista mentre la si percorre, il che causerebbe una java.util.ConcurrentModificationException.
* mostKnownPasswords: lista di stringhe che conterrà le 100000 password più conosciute al mondo.
* MAX\_LEN: constante int che contiene la lunghezza massima possibile della password inserita dall’utente.
* password: questa variabile di tipo stringa conterrà il primo argomento passato dall’utente nell’array args, ovvero la password da trovare.
* foundPassword: questa variabile di tipo stringa conterrà la password scoperta dal codice.
* time: questa variabile conterrà il tempo complessivo che il programma ha impiegato a scoprire la password in millisecondi.
* tries: questa variabile conterrà il numero di tentativi svolti dal programma per scoprire la password.
* found: variabile booleana che indica se la password è stata scoperta (true) o no (false).
* characters: questa lista di stringhe conterrà tutti i caratteri necessari per trovare la password usando l’algoritmo brute force, ovvero tutti i caratteri della tastiera (più altri).
* help: questa variabile di tipo stringa contiene varie indicazioni utili a chi usa il programma nel caso non sappia come funziona e viene rilevata un’eccezione, in tal caso verrà stampato il contenuto di questa variabile.

## Metodi

In seguito agli attributi ho creato i seguenti metodi:

### PasswordSecurityChecker(String[] argumetns)

Il costruttore della classe, che serve ad istanziare un oggetto di tipo PasswordSecurityChecker.

Per prima cosa viene assegnato un contenuto all’attributo help.

Questo metodo riceve come parametro un array di stringhe, il primo elemento verrà assegnato all’attributo password, se non è presente verrà rilevata un IllegalArgumentException, se invece è presente il suo valore verrà assegnato all’attributo password.

In seguito viene richiamato il metodo loadMostKnownPasswords() (che varrà descritto in seguito).

Dopodiché vengono salvati nell’attributo arguments i primi 4 elementi presenti dopo il primo, scartando quelli in più se presenti o prendendo quelli rimanenti se gli elementi dell’array sono minori di 5.

Dopo ciò se ci sono elementi nell’attributo arguemnts viene richiamato il metodo makeCombos() (che varrà descritto in seguito).

Infine l’attributo characters viene riempito con i caratteri della tabella ascii dal 33 al 255.

### loadMostKnownPasswords()

Questo metodo legge il contenuto del file 100000-most-known-passwords.txt che su ogni riga contiene una password e ognuna di queste password verrà inserite nella lista mostKnownPasswords.

Essendo che il file in questione si trova nella cartella src nel progetto di NetBeans ho dovuto leggere il suo contenuto usando un metodo diverso da quello che userei normalmente:

Questo è dovuto al fatto che il contenuto della cartella src diventerà poi un file jar, che si legge diversamente da file di testo.

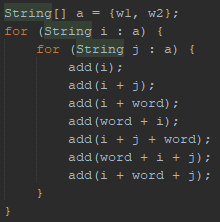
### add(String word)

Questo metodo aggiunge word a argumentCombosTemp ma controlla solo che rispetti il limite di lunghezza stabilito da MAX\_LEN.

### addCombos(String w1, String w2, String word)

Questo metodo serve a creare varie combinazioni tra i parametri passati, w1 e w2 sono delle stringhe contenenti nulla, allora a argumentCombosTemp verranno aggiunte solo le varie versioni di word (originale, maiuscolo e minuscolo).

Se solo w2 è una stringa vuota verranno aggiunte a argumentCombosTemp: w1 concatenato davanti a word (con tutte le varie versioni di w1, originale, maiuscolo e minuscolo) e la stessa cosa viene fatta concatenando w1 alla fine di word.

Se tutti i parametri contengono qualcosa viene fatto il seguente:

Viene fatto un array contenente w1 e w2, poi viene percorso con un doppio for e vengono aggiunte ad argumentCombosTemp diverse combinazioni tra w1, w2 e word

Tutte le aggiunte ad argumentCombosTemp vengono fatte con il metodo add().

### addSubStrings(String word)

Questo metodo ritorna una lista contenente: se possibile il primo carattere di word all’indice 0 della lista, all’indice 1 della lista vengono aggiunti se possibile i primi due caratteri di word, e all’indice 2 della lista viene aggiunto se possibile vengo aggiunti i primi tre caratteri di word. Il contenuto di questa lista viene poi aggiunto a argumetnCombosTemp usando addCombos:

così da ottenere tutte le varie versioni delle sotto stringhe (ovvero originale, minuscolo e maiuscolo).

### addTempCombosToCombos()

Questo metodo serve ad aggiungere il contenuto di argumentCombosTemp a argumentCombos e svuotare argumentCombosTemp.

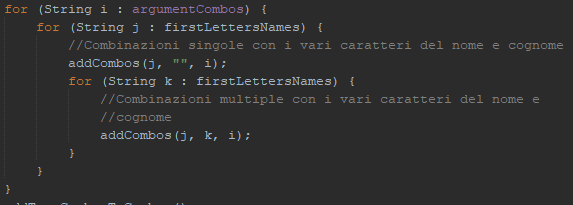
### makeCombos()

Questo metodo serve a creare diverse combinazioni tra i vari argomenti passati dall’utente e salvarli nell’attributo argumentCombos.

Per prima cosa vengono salvate in argumentCombosTemp tutte le varianti degli argomenti originali passati (maiuscoli. Minuscoli e originali) subito dopo viene chiamato il metodo addTempCombosToCombos(), in seguito vengono aggiunte a argumentCombosTemp tutte te combinazioni (tra 2 elementi) tra le varie versioni degli argomenti, subito dopo viene richiamato il metodo addTempCombosToCombos().

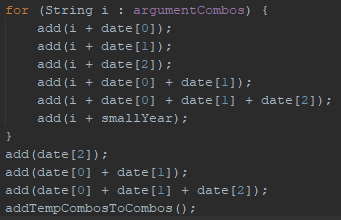
In seguito viene creata una lista di stringhe chiamata firstLettersNames a cui viene assegnato il valore ritornato da addSubStrings(arguments.get(0)), ovvero tutte le sotto stringhe del nome, poi se il cognome è presente vengono concatenate a firstLettersNames le sotto stringhe del cognome. In seguito viene svuotata la lista firstLettersNames e le vengono aggiunte tutti gli elementi di argumentCombosTemp (essendo che a questo punto argumentCombosTemp conterrà tutte le versioni delle sotto stringhe del nome e se presente anche del cognome) e viene richiamato addTempCombosToCombos().

Dopo questo punto verranno aggiunte a argumentCombos tutte le combinazioni fatte con addCombos passando ogni elemento in firstLettersNames e ogni elemento in argumentCombos , poi verrà fatta la stessa cosa passando però a addCombos in entrambi i primi due parametri ogni elemento in firstLettersNames e ogni elemento in argumentCombos, in questo modo:



Dopo questo viene richiamato il metodo addTempCombosToCombos() per aggiungere effettivamente gli elementi presenti il argumentCombosTemp a argumentCombos.

Dopo ciò se presente verrà preso il terzo elemento di arguments, ovvero la dati di nascita e in un array verranno messe le varie parti di essa, giorno, mese, anno, e verrà creata una variabile contenente solo le ultime due cifre dell’anno di nascita, in seguito per ogni elemento all’interno di argumentCombos verranno aggiunti questi quattro numeri, una versione dove si aggiungono tutte e tre i numeri della data di nascita e una dove vengono aggiunte gionro+mese, e infine vengono aggiunte singolarmente a argumentCombos giorno+mese e giorno+mese+anno se possibile, in questo modo:



Per ogni aggiunta alla lista argumentCombosTemp fatta in questo metodo viene utilizzato il metodo add(String) per evitare che password superino il limite e che quindi siano inutili.

### printTriesAndTime()

Questo metodo serve a stampare i tentativi svolti al momento che il metodo viene richiamato e il tempo che ha attualmente impiegato a fare questi tentativi. Questo metodo viene verrà usato nei metodi per scoprire la password così che l’utente che usa il programma vedrà che il programma sta lavorando e non avrà il dubbio che si è bloccato.

### finalPrint()

Questo metodo viene richiamato una volta che la password è stata scoperta e il programma è terminato, stampando la foundPassword, tries e time.

### endForce(String s)

Questo metodo viene richiamato per terminare un force, ovvero gli algoritmi per cercare di scoprire la password, impostando l’attributo foundPassword a s, aumentando di uno l’attributo tries e impostando a true l’attributo found.

### dictionaryForce()

Questo metodo percorre ogni elemento della lista mostKnownPasswords e le confronta con la password passata dall’utente, se essa combacia a una di queste password verrà invocato il metodo endForce passando la password in questione. Ogni 100 tentativi viene richiamato printTriesAndTime().

### argumentsForce()

Questo metodo percorre ogni elemento della lista argumentCombos e le confronta con la password passata dall’utente, se essa combacia a una di queste password verrà invocato il metodo endForce passando la password in questione. Ogni 10000 tentativi viene richiamato printTriesAndTime().

### bruteForce(String keys)

Questo metodo riceve inizialmente una stringa vuota, poi controlla che la lunghezza di keys non superi la lunghezza massima stabilita dalla costante. Poi inizia a percorrere l’attributo characters, controlla se l’attributo found sia false e se keys concatenato all’elemento corrente di characters sia uguale a password, se si verrà richiamato il metodo endForce e il programma terminerà, altrimenti verrà richiamato bruteForce passando keys + l’elemento corrente di characters, entrando in un ciclo che terminerà solo quando la password verrà trovata. Ogni 100000 tentativi viene richiamato printTriesAndTime().

### findPassword()

Questo è il metodo più importante, che mette insieme tutti gli altri metodi creati fino ad adesso. Per prima cosa viene richiamato il metodo argumentsForce(), se lui trova la password viene richiamato finalPrint(), altrimenti si prova con il dictionaryForce(), se lui trova la password viene richiamato il finalPrint(), altrimenti viene richiamato il bruteForce(), e una volta che lui trova la password viene richiamato finalPrint(), altrimenti il programma non finirà.

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-01  REQ-04  REQ-06  REQ-07  REQ-08 | **Nome:** | Test inserimento password |
| **Descrizione:** | Controllo che sia possibile inserire una password. | | |
| **Prerequisiti:** | Avere il programma. | | |
| **Procedura:** | Aprire il terminale. Andare nella cartella contenente PasswordSecurityChecker.jar  Utilizzare il programma da linea di commando inserendo una password qualsiasi. (Esempio: java -jar PasswordSecurityChecker.jar Admin01!).  Poi premere “enter” per far partire il programma. | | |
| **Risultati attesi:** | Il programma funziona senza problemi e alla fine stampa i tentativi e il tempo che ha impiegato a forzare la password. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-02 | **Nome:** | Test inserimento argomenti aggiuntivi |
| **Descrizione:** | Controllo che sia possibile inserire quanti argomenti si vogliono. | | |
| **Prerequisiti:** | Avere il programma. | | |
| **Procedura:** | Aprire il terminale. Andare nella cartella contenente PasswordSecurityChecker.jar  Utilizzare il programma da linea di commando inserendo una password per provare e il nome. (Esempio: java -jar PasswordSecurityChecker.jar Admin Admin).  Poi premere “enter” per far partire il programma.  Ripetere lo stesso processo passando anche un cognome. (Esempio: java -jar PasswordSecurityChecker.jar AadministratorA Admin Administrator).  Poi premere “enter” per far partire il programma.  Ripetere lo stesso processo passando anche una data di nascita. (Esempio: java -jar PasswordSecurityChecker.jar AAdministrator70 Admin Administrator 1.1.1970).  Poi premere “enter” per far partire il programma.  Ripetere lo stesso processo passando anche una parola extra (Esempio: AdminAcquario1970 Admin Administrator 1.1.1970 Acquario), poi premere “enter” per far partire il programma. | | |
| **Risultati attesi:** | Il programma funziona allo stesso modo in tutti i casi senza problemi e alla fine stampa i tentativi e il tempo che ha impiegato a forzare la password. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-03 | **Nome:** | Test lunghezza password |
| **Descrizione:** | Controllo che non sia possibile inserire una password più lunga di 20 caratteri. | | |
| **Prerequisiti:** | Avere il programma. | | |
| **Procedura:** | Aprire il terminale.  Utilizzare il programma da linea di commando inserendo una password di lunghezza 21 caratteri. (Esempio: aaaaaaaaaaaaaaaaaaaaa). Poi premere “enter” per far partire il programma. | | |
| **Risultati attesi:** | Il programma da errore e stampa un piccolo testo che descrive come usare correttamente esso. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-5 | **Nome:** | Test utilizzo delle password più conosciute al mondo |
| **Descrizione:** | Controllo che il programma utilizzi il file contenente le password più conosciute al mondo. | | |
| **Prerequisiti:** | Avere il programma. | | |
| **Procedura:** | Aprire il terminale.  Utilizzare il programma da linea di commando inserendo una password qualsiasi contenuta nel file delle password più conosciute (Esempio: qwerty), poi premere enter per far partire il programma. | | |
| **Risultati attesi:** | Il programma funziona senza problemi e alla fine stampa i tentativi e il tempo che ha impiegato a forzare la password. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-09 | **Nome:** | Test stampa di informazioni mentre il programma è in esecuzione |
| **Descrizione:** | Mentre il programma funziona, deve stampare su una riga del terminale i tentativi svolti al momento e quanto tempo è passato in quel momento. | | |
| **Prerequisiti:** | Avere il programma. | | |
| **Procedura:** | Aprire il terminale.  Utilizzare il programma da linea di commando inserendo una password qualsiasi (Esempio: Admin01!), poi premere “enter” per far partire il programma. | | |
| **Risultati attesi:** | Mentre il programma è in esecuzione vengono stampati i tentativi attualmente svolti e il tempo passato in quel momento. | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

## Mancanze/limitazioni conosciute

La limitazione più grande di questo programma è il fatto che se la password non viene trovata dai metodi argumentsForce o dictionaryForce e quindi potrà solo essere trovata con il bruteForce, e la password è più lunga di 4 caratteri il bruteForce non finirà mai, o per lo meno ci metterà troppo tempo a terminare, essendo che deve provare tutte le combinazioni possibili tra 223 caratteri, più la password è lunga più tempo impiegherà.

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

# Consuntivo

# Conclusioni

## Sviluppi futuri

Uno sviluppo futuro sarebbe di trasformare il programma com’è ora in un programma multi thread, per velocizzare il brute force soprattutto, e anche far lavorare tutti i diversi attacchi in “contemporanea”.

## Considerazioni personali

Da questo progetto ho imparato alcune cose: ad esempio come leggere il contenuto di un file .jar, come creare un array contenente quasi tutta la tabella ascii di 255 caratteri con una solo riga di codice, come cronometrare le varie operazioni del programma e come lavorare meglio con le liste e stringhe in generale.

Tutto sommato credo che questo progetto sia andato abbastanza bene, la parte più difficile di esso era sicuramente la parte del brute force, sulla quale ho impiegato diverso tempo, ma anche parlandone con alcuni colleghi sono riuscito ad arrivare ad una soluzione.

Credo di aver gestito abbastanza bene le tempistiche, anche se le mie stime iniziali erano sbagliate, avendo sottovalutato quanto tempo avrei impiegato a fare il brute force, e non essendo completamente in chiaro sulla consegna per l’attacco basato sugli argomenti inseriti dall’utente, sulla quale ho dovuto lavorare più del previsto.

Alla fine di questo progetto devo dire che un po’ mi pento della mia scelta, vedendo tutti i lavori dei miei colleghi mi rendo conto che il mio progetto è forse quello meno significativo, complesso e soddisfacente da realizzare. La mia scelta basata solamente sul fatto che dalla lista di possibili progetti ritenevo questo il più semplice da realizzare, non è stato proprio un ottimo criterio per la mia decisione.

Il programma funziona, fa quello che dovrebbe fare come stabilito dai requisiti, anche se c’è quella pecca del brute force che con una certa lunghezza di password non finisce più. Sicuramente questo progetto non cambierà il mondo e non è un successo importante, sono quasi tentato a dire che è stata una perdita di tempo considerando quello che ho imparato dal mio primo progetto da solo, e considerando anche i risultati ottenuti dai miei colleghi, il che mi rattrista un po’.

# Bibliografia

## Sitografia

* <https://stackoverflow.com/questions/20389255/reading-a-resource-file-from-within-jar>, *java - Reading a resource file from within jar - Stack Overflow*, 2021-09-23
* <https://stackoverflow.com/questions/11377113/java-printing-lines-from-buffered-reader>, *Java - printing lines from buffered reader - Stack Overflow*, 2021-09-23
* <https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-100000.txt#L100000>, *SecLists/10-million-password-list-top-100000.txt at master · danielmiessler/SecLists*, 2021-09-16
* <https://gist.github.com/jwoschitz/1129249>, *Brute force implementation / C#,* 2021-09-30
* <https://replit.com/@CyanCoding/C-Brute-Force-Password-Cracker>, *C# Brute Force Password Cracker - Replit*, 2021-09-30

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Codici sorgente/documentazione macchine virtuali
* Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
* Documentazione di prodotti di terzi
* Eventuali guide utente / Manuali di utilizzo
* Mandato e/o QdC
* Prodotto
* …