Tables arc-en-ciel

TRANSON Alexandre, RABOT Valentin, MAHIER Awen, FAUQUETTE Mael

L3 Informatique Groupe 3A

29 mars 2024

UNIVERSITÉ
CAEN
NORMANDIE

Table des matières

1 Introduction

1.1 Description générale du projet

Il s'agit d'une table arc-en-ciel programmé en Java, dotée d'une interface graphique, qui consiste à réduire les temps de calcul pour retrouver un mot de passe original au prix d'un coût en espace. Le hash est au cœur de ce système, étant utilisé pour l'identification des mots de passe originaux à partir de leurs hash correspondants.

1.2 Présentation du plan du rapport

Maintenant que nous avons introduit le sujet sur lequel porte notre projet et que nous avons pu lister les divers objectifs à atteindre, nous allons maintenant expliquer le plan global que suivra ce rapport de projet afin d'exposer notre projet de la manière la plus claire possible.

Dans un premier temps nous allons voir chacun des objectifs fixés un peu plus en détails et nous allons expliquer plus concrètement en quoi consistent-ils. Ensuite, nous verrons comment ces objectifs ont pu être traités dans d'autres projets et essayer de s'en inspirer ou de voir une manière différente de répondre à certaines contraintes.

Suite à cela nous expliquerons les principales fonctionnalités à implémenter et comment nous nous sommes organisés au sein de notre groupe afin de mener à bien l'implémentation de ces dernières.

Une fois notre organisation présentée nous verrons les divers éléments techniques auxquels nous devrons apporter des réponses au sein de ce projet. Puis nous verrons comment nous avons élaboré les divers algorithmes imaginés et utilisés dans notre projet. Pour terminer avec cette partie nous verrons comment sont organisées les diverses données utilisées dans nos algorithmes.

Pour la suite du rapport nous exposerons l'architecture de notre projet avec notamment le diagramme des classes.

Ensuite nous testerons notre programme afin d'explorer les divers angles d'usages de notre projet et expliquer quelles sont les données renvoyées et leur utilité.

Pour finir ce rapport nous conclurons par un récapitulatif de la problématique, des fonctionnalités ainsi que des résultats de notre projet puis par la proposition d'axes d'amélioration qui pourraient être à réfléchir à l'avenir.

2 Objectifs du projet

2.1 Problématique du projet

Afin de produire ce projet nous avons suivi une ligne conductrice basée sur une problématique bien précise, **comment créer une table arc-en-ciel fonctionnelle et optimisée ?** En effet, nous avions pour principaux objectifs :

- Générer une base synthétique de mots de passe hachés
- Implémenter une table arc-en-ciel (paramétrable en termes de couleurs)
- Implémenter des optimisations de la table arc-en-ciel

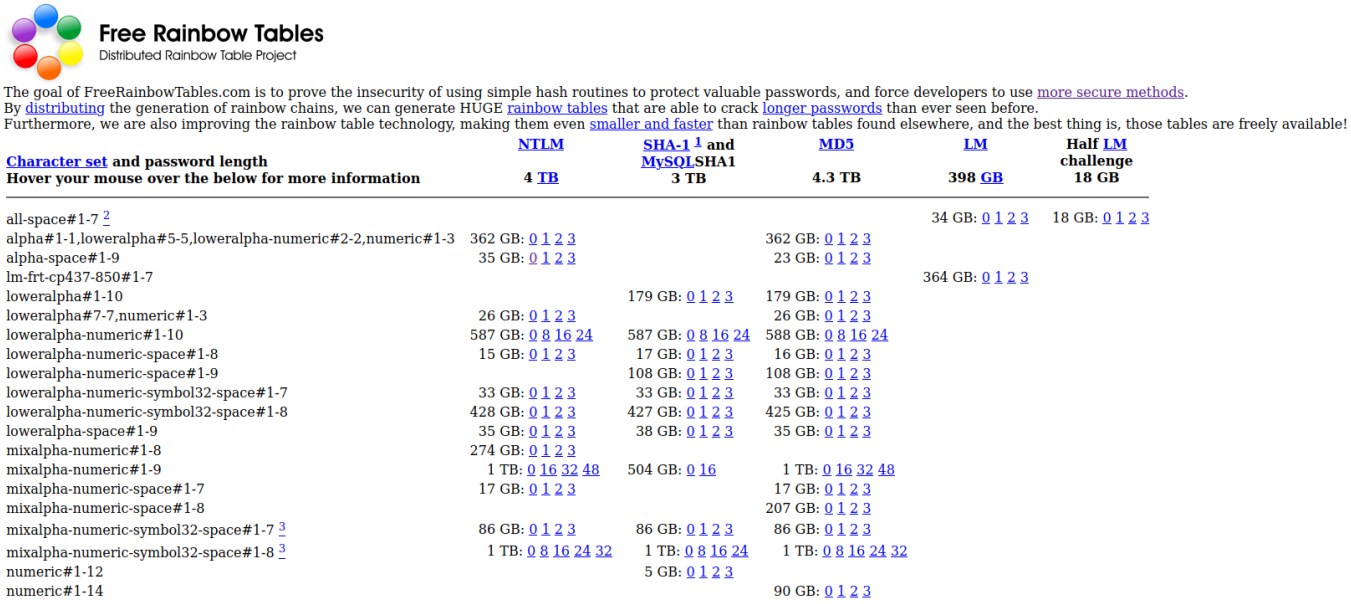
Nous allons voir plus en détail ces différents points dans les prochaines parties.

2.2 Description des points-clés et des grandes étapes

Voici les étapes importantes que nous avons suivis :
Étape 1 : L'analyse des besoins, identification des objectifs du projet, ainsi que des fonctionnalités qu'on souhaité inclure.
Étape 2 : Implémentation des fonctionnalités de base, telle que le hachage, le stockage de mot de passe dans un fichier, etc.
Étape 3 : Développement des fonctionnalités principales, telle que la comparaison de hachages, la recherche de mots de passe, etc.
Étape 4 : Tests et débogage, en effectuant des tests unitaires de l'application afin de vérifier son bon fonctionnement.
Étape 5 : Les optimisations et les améliorations, afin d'avoir de meilleures performances.

2.3 Description de travaux existants sur le même sujet

Le site freerainbowtables.com propose une variété de tables arc-en-ciel pour différents types de hachage, tels que NTLM, SHA-1, MD5, LM, etc..., avec des capacités de cracking allant jusqu'à plusieurs téraoctets en fonction du type de hachage et de la longueur du mot de passe. Ces tables sont souvent disponibles gratuitement en téléchargement via des liens BitTorrent.



Free Rainbow Tables
Distributed Rainbow Table Project

The goal of FreeRainbowTables.com is to prove the insecurity of using simple hash routines to protect valuable passwords, and force developers to use more secure methods.
By distributing the generation of rainbow chains, we can generate HUGE rainbow tables that are able to crack longer passwords than ever seen before.
Furthermore, we are also improving the rainbow table technology, making them even smaller and faster than rainbow tables found elsewhere, and the best thing is, those tables are freely available!

Character set and password length Hover your mouse over the below for more information	NTLM 4 TB	SHA-1 [1] and MySQLSHA1 3 TB	MD5 4.3 TB	LM 398 GB	Half LM challenge 18 GB
all-space#1-7 [2]				34 GB: 0 1 2 3	18 GB: 0 1 2 3
alpha#1-1,loweralpha#5-5,loweralpha-numeric#2-2,numeric#1-3	362 GB: 0 1 2 3		362 GB: 0 1 2 3		
alpha-space#1-9	35 GB: 0 1 2 3		23 GB: 0 1 2 3		
lm-frt-cp437-850#1-7				364 GB: 0 1 2 3	
loweralpha#1-10		179 GB: 0 1 2 3	179 GB: 0 1 2 3		
loweralpha#7-7,numeric#1-3	26 GB: 0 1 2 3		26 GB: 0 1 2 3		
loweralpha-numeric#1-10	587 GB: 0 8 16 24	587 GB: 0 8 16 24	588 GB: 0 8 16 24		
loweralpha-numeric-space#1-8	15 GB: 0 1 2 3	17 GB: 0 1 2 3	16 GB: 0 1 2 3		
loweralpha-numeric-space#1-9		108 GB: 0 1 2 3	108 GB: 0 1 2 3		
loweralpha-numeric-symbol32-space#1-7	33 GB: 0 1 2 3	33 GB: 0 1 2 3	33 GB: 0 1 2 3		
loweralpha-numeric-symbol32-space#1-8	428 GB: 0 1 2 3	427 GB: 0 1 2 3	425 GB: 0 1 2 3		
loweralpha-space#1-9	35 GB: 0 1 2 3	38 GB: 0 1 2 3	35 GB: 0 1 2 3		
mixalpha-numeric#1-8	274 GB: 0 1 2 3				
mixalpha-numeric#1-9	1 TB: 0 16 32 48	504 GB: 0 16	1 TB: 0 16 32 48		
mixalpha-numeric-space#1-7	17 GB: 0 1 2 3		17 GB: 0 1 2 3		
mixalpha-numeric-space#1-8			207 GB: 0 1 2 3		
mixalpha-numeric-symbol32-space#1-7 [3]	86 GB: 0 1 2 3	86 GB: 0 1 2 3	86 GB: 0 1 2 3		
mixalpha-numeric-symbol32-space#1-8 [3]	1 TB: 0 8 16 24 32	1 TB: 0 8 16 24	1 TB: 0 8 16 24 32		
numeric#1-12		5 GB: 0 1 2 3			
numeric#1-14			90 GB: 0 1 2 3		

FIGURE 1 – Page d'accueil du site freerainbowtables.com

3 Fonctionnalités implémentées

3.1 Fonctionnement

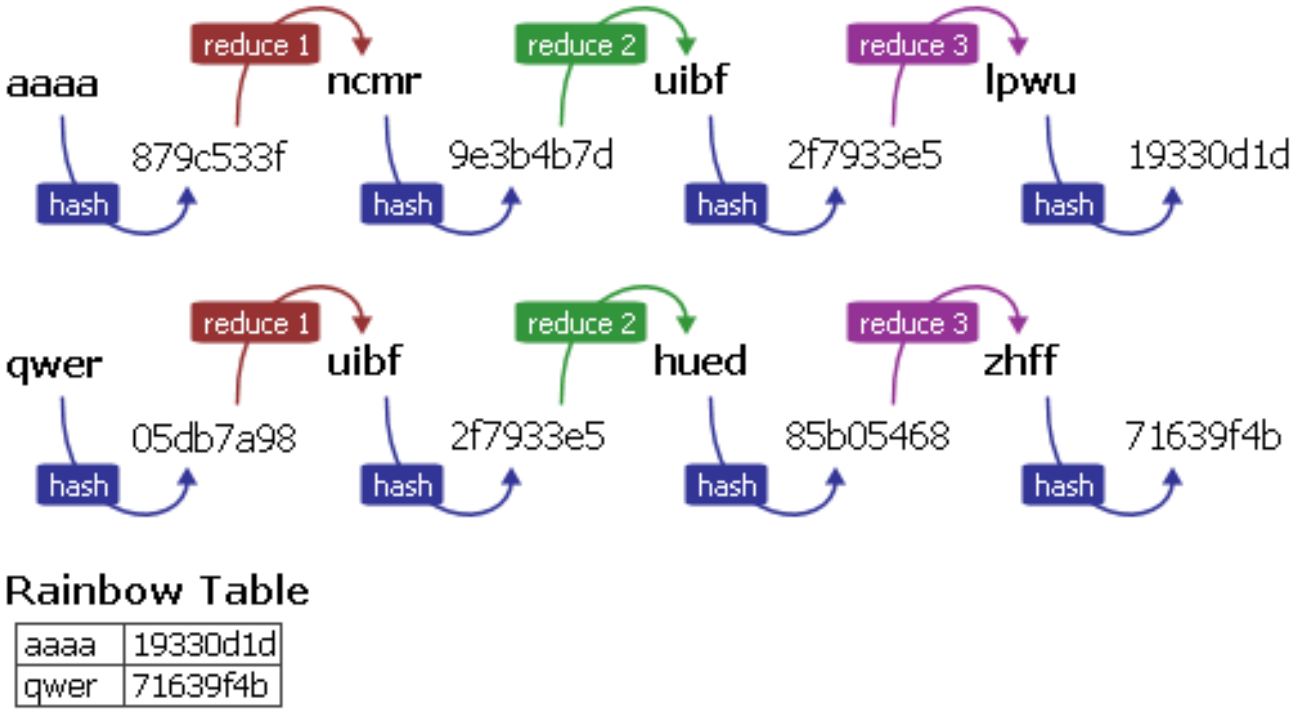


FIGURE 2 – Génération d'une rainbow table

On part d'un mot de passe puis on le hash, le hachage est une fonction qui prend en entrée un mot de passe et produit en sortie une empreinte numérique, appelée le hash. Cette fonction est conçue de telle sorte que deux ensembles de données différents produisent généralement des hashs différents et qu'il est difficile de retrouver le mot de passe à partir du hash. Cependant, plusieurs entrées peuvent générer le même hash (collision de hachage), ce qui est un aspect à considérer lors de l'utilisation de tables arc-en-ciel pour retrouver le mot de passe à partir d'un hash on utilise la réduction qui est le processus inverse du hachage. Elle consiste à prendre un hash et à le convertir en une valeur. Cette valeur réduite est utilisée comme point de départ pour générer une nouvelle chaîne de données, qui sera ensuite hachée à nouveau. La fonction de réduction est conçue pour produire une distribution uniforme des valeurs réduites.

3.2 Description des fonctionalités

3.2.1 Génération des mots de passe

Algorithme 1 : Algorithme de génération de mot de passe

Sortie : Un mot de passe généré "mdp"

```
seed ← 0;
pour c in this.hashpw.toCharArray() faire
    seed ← 31L × seed + c;
generator ← new Random(seed);
alphabet ← "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVW
  XYZ0123456789[] !()*+,-./ :;<>=| ?";
n ← generator.nextInt(32) + 8;
mdp ← "";
pour z ← 0 to n faire
    c ← alphabet.charAt(generator.nextInt(alphabet.length()));
    mdp ← mdp + c;
return mdp;
```

On calcul une "seed" (graine) en parcourant chaque caractère d'un hash donné qui sera utilisée pour créer un objet Random afin de générer des nombres aléatoires. Puis une longueur 'n' est sélectionnée pour le mot de passe entre 8 et 32 caractères, ensuite on itère n fois pour choisir aléatoirement un caractère d'une chaîne contenant les lettres de l'alphabet en minuscule et majuscule, les chiffres et quelques caractères spéciaux et l'ajouter au mot de passe.

3.2.2 Stockage des mots de passe

Ils sont stockés dans un fichier texte avec comme première ligne la profondeur de la compression et la liste des fonctions de réductions représentés par une couleur. Chaque ligne représente un mot de passe d'origine et son hash terminal.

```
5 [Purple]
oYI671zd5fTRc 9190b9202f0aaf9ae55292f0eb980848d6d9dc4709ad42a44194cc3d1c8171d3
gR5Ok1M7ti fc923e8132dbfd9d1006ce6cee23d159e0794abcbd54ed19de65440b15b61694
wQcRPKVLhn f9c4ef091c17b37d35b8f410984b7c1e752b4953dccbde53c21a0277feec06fd
JOBq 69eb8919619196e50640fb80d1989eae99eca5d4584c835f1dc92bd50ddc6be8
cvUPQjJXMct3k 31da95a13afc6b52a15f0a02177fa662f14b7769ce4601cceaaba67452977907
```

FIGURE 3 – Données dans le fichier texte

La profondeur correspond au nombre d'itérations du processus de compression et la couleur à la fonction de réduction qui vise à diminuer la probabilité de collision.

3.2.3 Récupération des données

L'objet BufferedReader est utilisé pour lire le fichier texte ligne par ligne, la première ligne est extraite pour obtenir la profondeur et les fonctions de réductions puis les lignes suivantes sont parcourues. BufferedReader est une classe Java utilisée pour la lecture de fichiers texte en mémoire tampon. Plutôt que de lire chaque caractère individuellement, ce qui peut être inefficace, le BufferedReader lit les données en blocs, ce qui réduit les appels coûteux. Cela en fait un choix idéal pour la lecture de fichiers texte ligne par ligne.

3.2.4 Comparaison des mots de passe

On utilise des fonctions de réduction pour obtenir le mot de passe. S'il n'y a aucune correspondance on effectue une recherche par couleur pour générer de nouveaux hash à partir du hash fourni jusqu'à trouver une correspondance ou atteindre la profondeur maximale.

3.2.5 Fonction de réduction

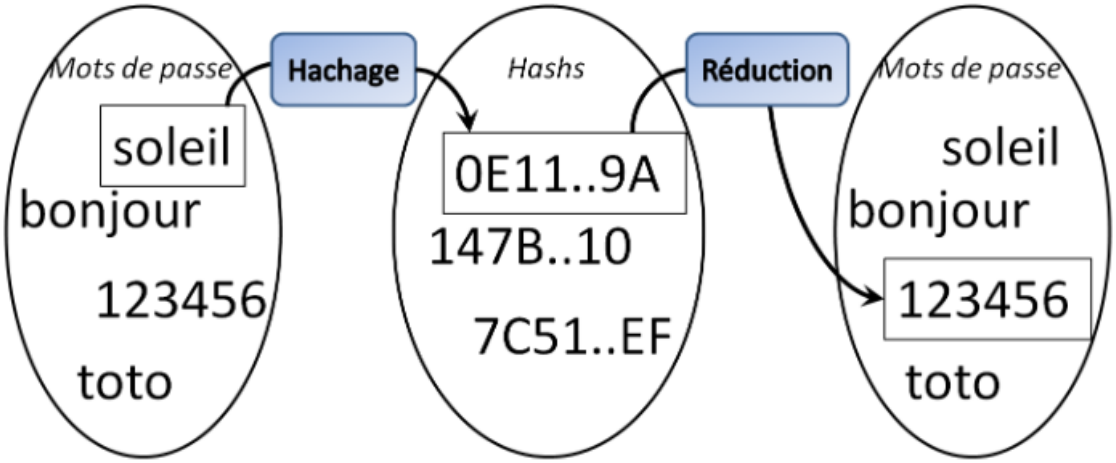


FIGURE 4 – Structure d'un maillon de chaîne

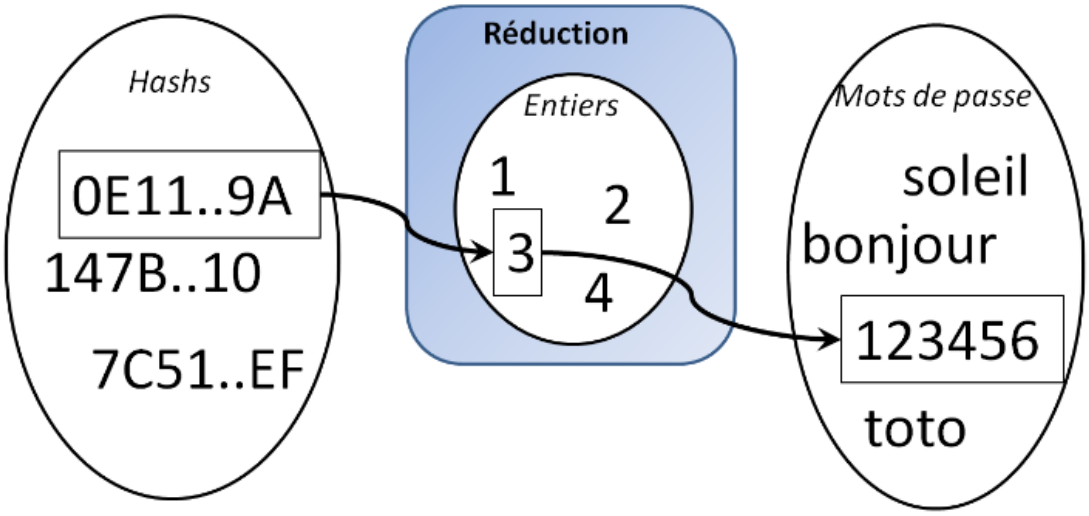


FIGURE 5 – Décomposition d'une fonction de réduction

Les valeurs de hachage qui sont dans les tables arc-en-ciel sont créées à l'avance. Les personnes peuvent obtenir les tables arc-en-ciel et les utiliser pour chercher des mots de passe. Cependant, ces fichiers sont très volumineux donc on utilise une fonction de réduction pour éviter l'encombrement de l'espace de stockage. Dans une table arc-en-ciel une nouvelle valeur de hachage est générée à partir d'un mot de passe plusieurs fois, de sorte qu'une chaîne est créée. Dans la table finale seul le premier mot de passe et la dernière valeur de hachage de la chaîne apparaissent. Grâce a ses informations et en tenant compte des fonctions de réduction utilisées toutes les autres valeurs peuvent aussi être déterminées. La valeur de hachage à craquer est

réduite de façon répétée et hachée selon les mêmes règles, chaque résultat intermédiaire étant comparé aux valeurs du tableau.

Dans notre projet, nous avons 6 fonctions de réduction différentes (Orange, Blue, Purple, Yellow, Green et Red) chacune d'entre elles applique une réduction de hash différente. Par exemple "Orange" prend les caractères du hash aux positions impaires tandis que "Purple" prend les 32 premiers caractères du hash sur 64.

3.2.6 Interface graphique

L'interface graphique de notre application a été réalisée en utilisant Swing qui est une bibliothèque de composants graphiques avancée qui étend et améliore les fonctionnalités d'AWT.

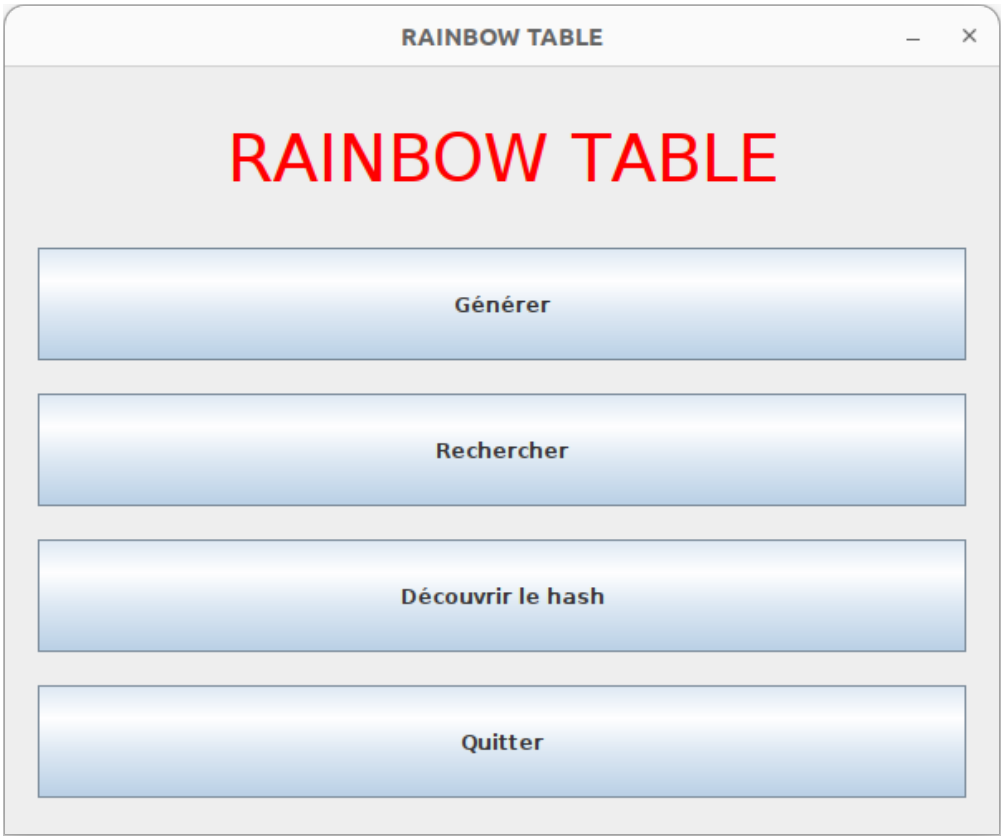


FIGURE 6 – Fenêtre principale de l'interface graphique

Si nous voulons générer une base de hash de mot passe, il faut cliquer sur le bouton "Générer". Puis de suivre les étapes suivantes ci dessous.

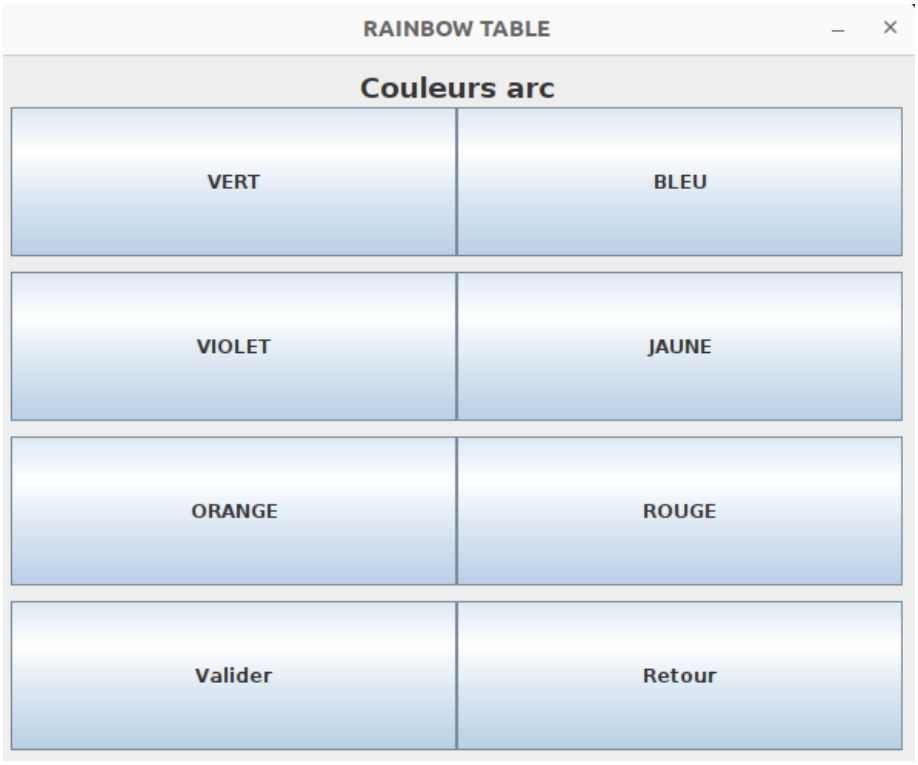


FIGURE 7 – Fenêtre du choix des fonctions de réduction

Il faut désormais choisir les fonctions de réductions que l'on veux (minimum 1) en cliquant dessus.

FIGURE 8 – Fenêtre de la taille du fichier

Et enfin préciser la profondeur ainsi que le nombre de ligne. Une profondeur de 1000 associée à un nombre de lignes de 1000 générera au total (1000 x 1000) 1 000 000 de chaînes de hachage. Donc le fichier texte résultant occupera environ 80ko d'espace de stockage. De même, pour

une profondeur de 500 avec un nombre de lignes de 500, le fichier texte totalisera une taille d'environ 40ko.

FIGURE 9 – Fenêtre de recherche d'empreinte

Si nous voulons rechercher un mot de passe grâce a son hash, il faut de cliquer sur le bouton "Rechercher" de la fenêtre principale puis d'entrer le hash dans la nouvelle fenêtre. Le hash entré doit être d'une longueur de 64 caractères puis enfin cliquer sur le bouton "Recherche". Il se peut que aucun mot de passe ne soit trouver.

FIGURE 10 – Fenêtre de transformation en hash

Si nous voulons découvrir le hash d'un mot de passe, il faut cliquer sur le bouton "Découvrir le hash" de la fenêtre principale puis d'entrer le mot de passe dans la nouvelle fenêtre. Le hash entré doit être d'une longueur de 8 à 32 caractères.

3.3 Organisation du projet

3.3.1 Répartition des tâches

Pour la répartition des tâches, Awen s'est occupé de la génération des mots de passe et du stockage de leurs hash, Mael des fonctions de réduction ainsi que de la comparaison des hash, Valentin de la récupération des données dans un fichier et des tests unitaires puis pour finir Alexandre de l'interface graphique et du hachage des mots de passe. Le rapport et la soutenance ont étaient rédigé par Valentin.

3.3.2 Logiciels utilisés

Pour la gestion des versions du projet nous avons utilisé l'outil "Subversion" (SVN). SVN est un système de gestion de version centralisé, utilisé pour suivre et contrôler les modifications apportées à un ensemble de fichiers ou de dossiers au fil du temps. Ainsi que le site de la Forge de l'université de Caen, nous avons créé un groupe Discord pour la communication entre les différents membres du groupe. Pour la génération de diagramme nous avons utilisé le logiciel Dia. Pour l'écriture de ce rapport nous avons utilisé l'éditeur LaTeX en ligne Overleaf. Logiciel utilisé également pour la soutenance. Pour finir pour réaliser notre projet nous avons fait appel à un outil de construction de logiciel en Java nommé Ant permettant d'automatiser l'initialisation, la compilation, l'exécution, la génération de documentation, l'archivage au format JAR et l'exécution des tests.

4 Éléments techniques

4.1 Description des paquetages non standards

4.1.1 Test unitaire

Pour les tests unitaires, nous avons utilisé le framework de tests unitaires JUnit en version 3.8.2. Il est conçu pour le langage de programmation Java. Il est utilisé pour faciliter le processus de test en permettant d'écrire des tests unitaires pour vérifier la fonctionnalité des méthodes. Ce framework fournit une série de classes et de méthodes pour écrire des tests unitaires, qui sont des morceaux de code qui testent une méthode ou une classe individuelle. Nos tests unitaires sont écrits à l'aide d'assertions, qui vérifient que le résultat d'une méthode ou l'initialisation d'une variable est conforme à ce qui est attendu. Afin de réaliser les tests nous avons téléchargé le fichier .jar de JUnit puis nous l'avons mis dans le répertoire "lib".

Voici un aperçu des tests réalisés :

— **Test de la classe HashCompare :** Ce test vérifie le bon fonctionnement de la comparaison de hachages. Nous avons vérifié que la création d'un hachage, la comparaison avec un hachage existant dans le fichier et la recherche d'un mot de passe associé à un hachage fonctionnent comme prévu.

— **Test de la classe Menu :** Ce test garantit que le bouton de recherche dans le menu est initialisé correctement après la création de l'interface. Cela assure que l'interface est correctement configurée et prête à être utilisée.

— **Test de la classe RainbowGUI :** Ce test vérifie si la fenêtre de l'interface graphique RainbowGUI est visible après sa création. Il s'assure que l'interface est rendue correctement et prête à être utilisée.

— **Test de la classe Search :** Ce test vérifie si les fenêtres de recherche Search sont visibles, que ce soit avec un mot de passe trouvé ou non. Cela garantit que les fenêtres de recherche fonctionnent comme prévu.

— **Test de la classe Rainbow :** Ce test vérifie le bon fonctionnement de la classe Rainbow en vérifiant si la méthode `ApplicationColor()` produit le résultat attendu. Cela assure que la logique de compression des couleurs fonctionne correctement.

Voici ce qui est affiché dans le terminal après avoir exécuté les tests unitaires. Cela nous permet de connaître le nombre de tests réussis/échoués et de repérer les erreurs dans les tests. Nous remarquons que sur les 5 tests lancés aucun n'a échoué.

```
test:
    [junit] Running test.Test
    [junit] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0,58 sec
```

FIGURE 11 – Résultat des tests unitaires dans le terminal

Ces tests unitaires assurent une validation automatique du fonctionnement du code, ce qui contribue à sa fiabilité et à sa qualité.

4.2 Descriptions des algorithmes

4.2.1 Génération de hash

Algorithme 2 : Algorithme de création du hash SHA-256

Sortie : Hash du mot de passe

$md \leftarrow$ MessageDigest.getInstance("$SHA-256$");
$digest \leftarrow md.digest(password.getBytes(StandardCharsets.UTF_8))$;
$sha \leftarrow$ **new** StringBuilder();
pour b ***in*** $digest$ **faire**
    sha.append(String.format("%02x", b));
retourner *sha.toString().toLowerCase()*;

Cet algorithme prend en entrée un mot de passe et retourne le hash SHA-256 correspondant. Il utilise l'algorithme de hachage SHA-256 fourni par la bibliothèque Java. Ensuite, il calcule le hash du mot de passe en convertissant d'abord le mot de passe en un tableau d'octets, puis en appliquant le hachage SHA-256. Le hash est converti en une chaîne de caractères hexadécimale pour le rendre lisible, puis il est retourné en minuscules.

4.2.2 Génération d'un nouveau hash avec une réduction

Algorithme 3 : Fonction hashIsNot

Entrées : String *couleur*, String *hash*
Sortie : Nouveau hash

```
1 rainbow ← new Rainbow();
2 hashReduc ← rainbow.ApplicationColor(couleur, hash);
3 generate ← new Generate(hashReduc);
4 tmpMdp ← generate.generatePassword();
5 hashable ← new Hash(tmpMdp);
6 hash ← hashable.creeHash();
7 retourner hash ;
```

Cet algorithme prend en entrée une couleur et un hash, puis il utilise la classe Rainbow pour appliquer une transformation de hash à l'aide de la couleur spécifiée. Ensuite, il génère un mot de passe à partir du hash réduit à l'aide de la classe Generate, puis recalcule le hash du mot de passe généré avec la classe Hash. Enfin, il renvoie le nouveau hash ainsi obtenu. Cette fonction est utilisée dans les méthodes de la classe HashCompare pour générer de nouveaux hash à partir de couleurs spécifiques.

4.3 Descriptions des structures de données

Pour stocker les couleurs, nous avons utilisé une ArrayList. C'est une structure de données dynamique dans Java, ce qui signifie qu'elle peut grandir ou rétrécir dynamiquement pendant l'exécution du programme. Chaque couleur est représentée par une chaîne de caractères (String). Cela permet aux classes d'accéder facilement aux différentes couleurs nécessaires pour compresser les mots de passe.

Pour l'utiliser en Java nous devons l'importer avec la ligne :

```
1 import java.util.ArrayList;
```

Pour ajouter une couleur dans l'Arraylist nous devons faire par exemple :

```
1 tabCouleur.add("Green");
```

Pour récupérer une couleur dans l'ArrayList nous devons faire :

```
1 tabCouleur.get(indiceCouleur);
```

4.4 Description des données

4.4.1 Données utilisateur

— Le hash
C'est le résultat l'application d'une fonction de hachage, c'est une valeur obtenue à partir de données d'entrée par exemple un mot de passe. Cette valeur est généralement de taille fixe, quelle que soit la taille des données d'origine. Les propriétés principales d'un bon algorithme de hachage incluent la résistance à la collision (deux mots de passe ne doivent pas générer le même

hash) et la non-réversibilité (il doit être difficile de retrouver les données d'origine à partir du hash). Les fonctions de hachage sont utilisées pour sécuriser les données en cryptographie et pour vérifier l'intégrité des données.

4.4.2 Données de configuration

— La profondeur

Elle fait référence au nombre d'itérations effectuées dans le processus de génération des rainbow tables. Chaque itération consiste à appliquer une fonction de hachage puis une fonction de réduction sur une valeur donnée, et ce processus est répété jusqu'à ce qu'une chaîne de longueur désirée soit obtenue. Plus la profondeur est grande, plus la rainbow tables est longue, ce qui peut potentiellement augmenter la capacité de recherche de mots de passe.

— La longueur

Elle représente la taille des chaînes arc-en-ciel générées. Une longueur plus grande peut potentiellement augmenter la probabilité de couvrir un espace de recherche plus large pour les mots de passe.

— Les couleurs

Elles font référence aux fonctions de réduction utilisées dans le processus de génération. Chaque couleur correspond à une fonction de réduction différente qui est appliquée à chaque itération pour transformer les valeurs hachées en nouvelles valeurs. Ces fonctions de réduction sont utilisées pour réduire la taille de l'espace de recherche des mots de passe et réduire les probabilités de collision comme dit précédemment.

4.5 Se protéger des rainbow tables

Pour se protéger contre les attaques utilisant des rainbow tables, il est essentiel d'adopter des pratiques de sécurité, notamment :

— Utiliser des fonctions de hachage robustes

Optez pour des algorithmes de hachage sécurisés et réputés comme SHA-256 ou SHA-3 plutôt que des algorithmes plus faibles comme MD5 ou SHA-1, qui sont vulnérables aux attaques par rainbow tables.

— Utiliser des techniques de salage (salt)

Le salage consiste à ajouter une chaîne aléatoire de données à chaque mot de passe avant de le hacher. Cela rend les attaques par rainbow tables beaucoup plus complexes, car chaque mot de passe haché est unique, même s'il s'agit du même mot de passe utilisateur.

— Implémenter des politiques de complexité de mot de passe

Encouragez les utilisateurs à choisir des mots de passe longs, complexes et uniques, et imposez des politiques de renouvellement régulier des mots de passe pour réduire la probabilité qu'un mot de passe puisse être compromis et utilisé dans une attaque par rainbow tables.

— Surveiller et détecter les activités suspectes

Mettez en place des mécanismes de surveillance pour détecter les tentatives d'authentification anormales, ce qui peut indiquer une tentative d'utilisation de rainbow tables ou d'autres techniques d'attaque.

5 Architecture du projet

5.1 Description des paquetages

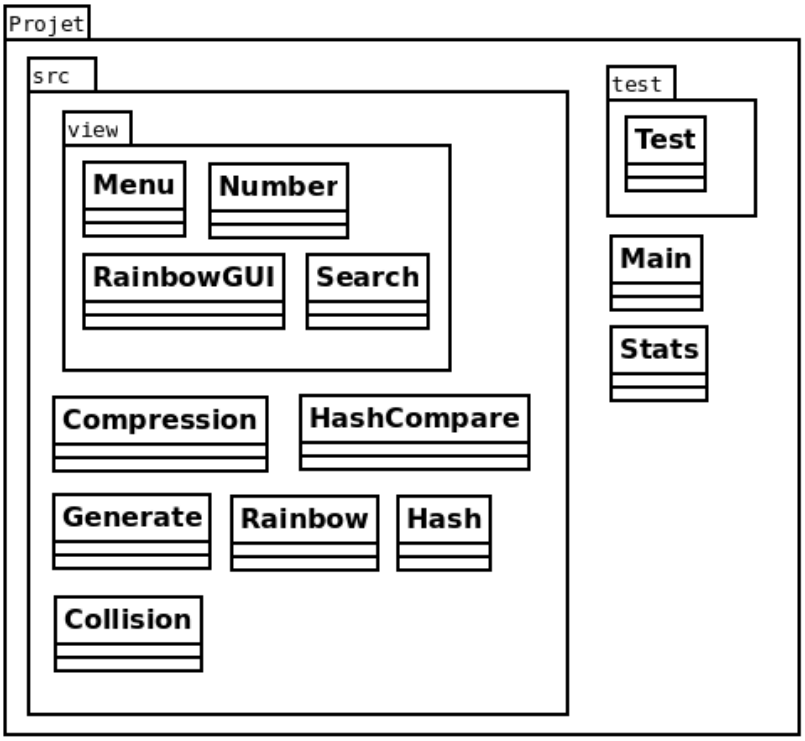
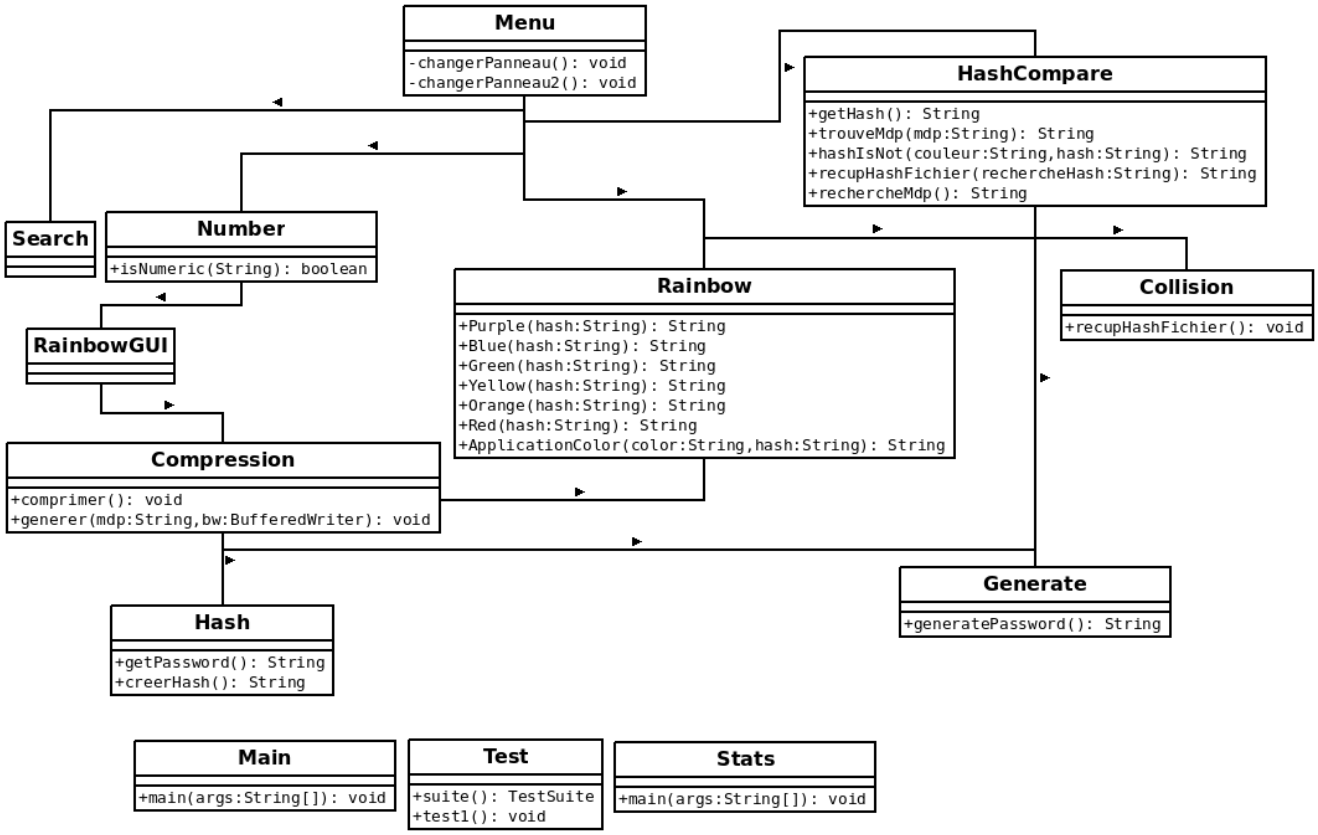


FIGURE 12 – Diagramme des paquetages

Ces paquetages sont divisés ainsi parce que nous avons eu la volonté de bien séparer l'aspect visuel de l'aspect technique de notre projet. Les classes liées à l'interface utilisateur sont regroupées dans le paquetage view.

5.2 Diagrammes des classes



Les diagrammes des classes et des paquetages ont étaient fait grâce au logiciel de création de diagramme Dia. Il représente l'architecture logicielle de l'application. La classe Menu crée une interface utilisateur permettant à l'utilisateur d'interagir avec différentes fonctionnalités. Cette classe est associée à plusieurs autres classes telles que HashCompare, Number, Rainbow, et Search, qui sont responsables de différentes étapes du processus de génération et de recherche de mots de passe. La classe HashCompare effectue la comparaison d'empreintes et utilise la classe Rainbow pour générer des couleurs. La classe Number permet à l'utilisateur de spécifier la taille du fichier à générer, tandis que la classe RainbowGUI gère l'interface graphique pour afficher le processus de génération de mots de passe. La classe Search affiche le résultat de la recherche d'un mot de passe puis la classe Collision permet de vérifier si il y a une ou des collisions de hash.

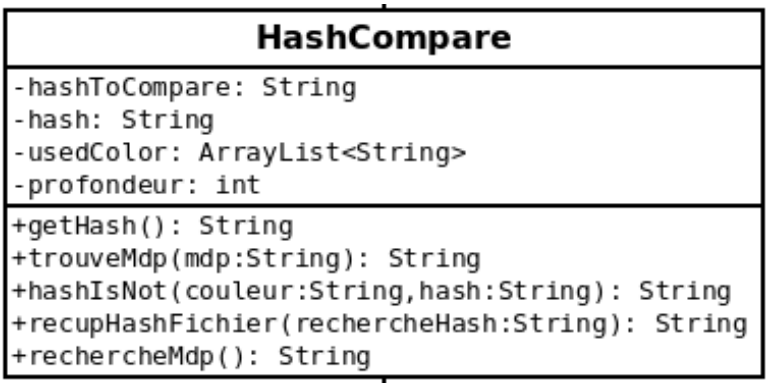


FIGURE 13 – Diagramme de la classe HashCompare

Voyons plus en détail la classe HashCompare. Cette classe joue un rôle crucial dans la com-

paraison des hashs et la récupération des mots de passe associés. Elle est conçue pour interagir avec un fichier texte ("mdp.txt") contenant des associations de mots de passe et de hashs. La classe prend en attribut un hash, le hash à comparer et stocke également une liste de couleurs utilisées ainsi que la profondeur du processus de réduction de hash.
La méthode `trouveMdp` compare le hash fourni avec ceux stockés dans le fichier texte et retourne le mot de passe correspondant s'il existe. Elle utilise un processus de réduction de hash basé sur une liste de couleurs pour retrouver le mot de passe associé.
La méthode `hashIsNot` génère un nouveau hash en appliquant une réduction de hash basée sur une couleur spécifiée.
La méthode `recupHashFichier` recherche dans le fichier "mdp.txt" le mot de passe associé à un hash donné. Elle extrait les informations du fichier et utilise la méthode trouveMdp pour vérifier si le hash fourni correspond à celui de la ligne.
Et enfin la méthode `rechercheMdp` orchestre l'utilisation des méthodes précédentes pour rechercher le mot de passe associé à un hash donné. Elle commence par utiliser la méthode recupHashFichier pour récupérer le mot de passe correspondant au hash dans le fichier, puis, si nécessaire, génère de nouveaux hash à partir de différentes couleurs jusqu'à trouver le mot de passe correspondant.

5.3 Chaînes de traitement

Voici les étapes de la chaîne de traitement :

— Menu principal
L'utilisateur interagit avec le menu principal pour choisir entre la génération de mots de passe avec leur hash associé, la recherche d'un mot de passe ainsi que découvrir le hash d'un mot de passe.

— Génération de mots de passe
L'utilisateur sélectionne cette option pour générer une table arc-en-ciel. Une fois l'option sélectionnée, l'interface lui permet de choisir les couleurs à inclure dans le processus de génération. L'utilisateur spécifie ensuite la taille du fichier à générer en termes de profondeur et de nombre de lignes. Une fois que toutes les informations sont fournies, le fichier de mots de passe est généré puis une interface indique à l'utilisateur que la génération est terminée.

— Recherche de mot de passe
L'utilisateur sélectionne cette option pour rechercher un mot de passe à partir d'une empreinte spécifiée. L'interface lui permet d'entrer une empreinte à rechercher. Une fois l'empreinte entrée, l'application compare cette empreinte aux empreintes existantes. Si un mot de passe correspondant est trouvé, il est affiché à l'utilisateur. Sinon, l'application indique à l'utilisateur que le mot de passe est introuvable.

— Découvrir le hash d'un mot de passe
L'utilisateur sélectionne cette option pour rechercher le hash d'un mot de passe à partir du mot de passe. L'interface lui permet d'entrer une empreinte à rechercher. Une fois le mot de passe entré, l'application affiche à l'utilisateur son hash.

— Retour et quitter
À chaque étape, l'utilisateur a la possibilité de retourner au menu principal pour effectuer une autre action ou de quitter l'application.

6 Expérimentations

6.1 Cas d'utilisation

6.1.1 Documentation

Pour générer la documentation, il suffit de se placer dans le dossier de la table arc-en-ciel puis d'écrire "ant javadoc" dans le terminal. Cela générera la documentation automatiquement à partir des contrats des classes et des méthodes et sera consultable directement sur un navigateur web. La documentation est stockée dans le dossier "doc".

6.1.2 Lancement du jeu

Afin de lancer l'interface graphique, il suffit d'écrire "ant run" dans le terminal qui s'occupera de créer le dossier build, de compiler les classes, de vérifier que les tests sont réussis puis d'exécuter la classe principale. Ensuite, vous pourrez choisir entre la génération d'une rainbow table, la recherche de mot de passe ou la découverte du hash d'un mot de passe selon vos besoins. Cela est possible grâce au fichier build.xml qui définit les différentes tâches à exécuter lors de la commande "ant run". L'utilisation de "ant run" simplifie le processus en automatisant les étapes nécessaires à l'exécution de l'application, ce qui facilite grandement son utilisation pour l'utilisateur final.

6.2 Résultats quantifiables

Profondeur 1000, longueur 1
— Temps de compression : 279 ms
— Temps de recherche : 1097 ms
— Taille : 102 octets
Profondeur 10, longueur 10
— Temps de compression : 152 ms
— Temps de recherche : 591 ms
— Taille : 831 octets
Profondeur 10, longueur 100
— Temps de compression : 109 ms
— Temps de recherche : 398 ms
— Taille : 8169 octets
Profondeur 1, longueur 1000
— Temps de compression : 89 ms
— Temps de recherche : 301 ms
— Taille : 81000 octets

Voici les résultats sous forme de graphique :

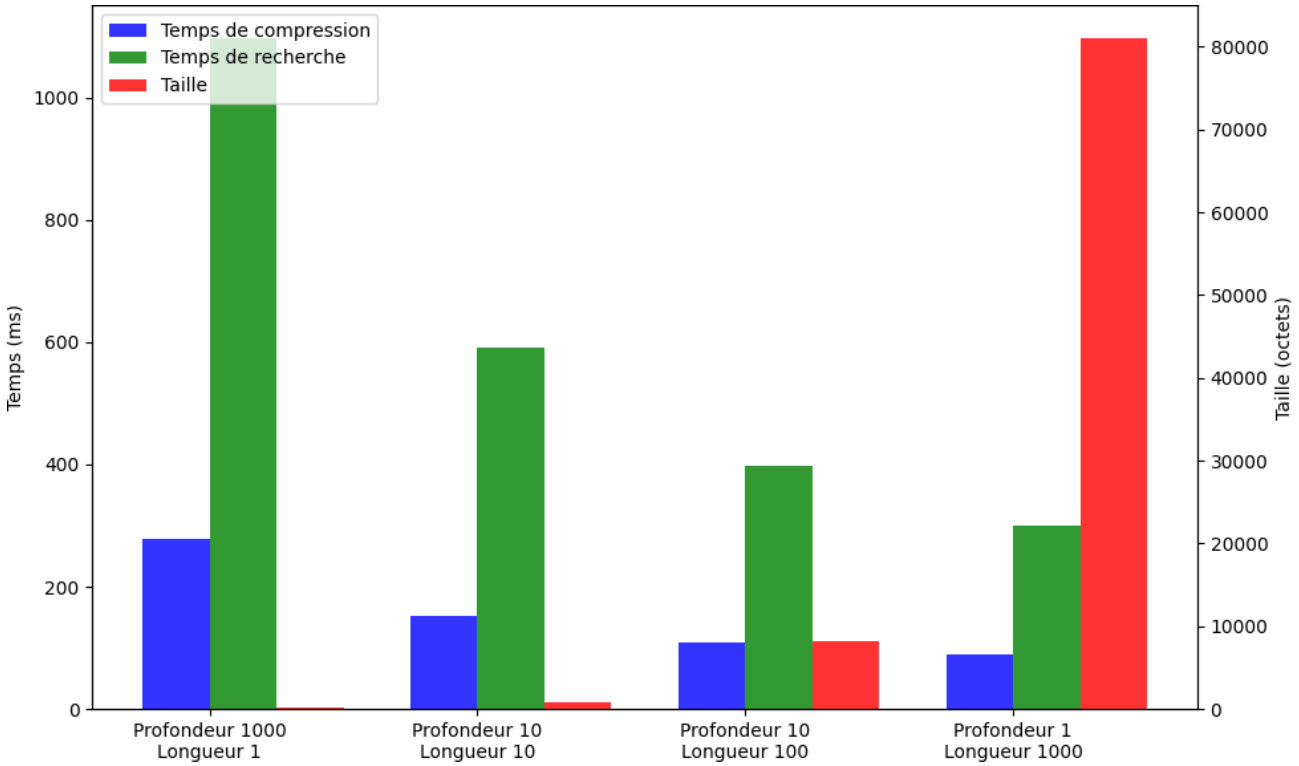


FIGURE 14 – Graphique représentant le temps de compression, de recherche et la taille en fonction de la profondeur et de la longueur

6.3 Analyse des résultats

Chaque barre du graphique représente une mesure pour une profondeur et une longueur spécifique. Les valeurs sur l'axe des x indiquent la longueur et la profondeur, tandis que les valeurs sur l'axe des y représentent le temps nécessaire pour effectuer les opérations, mesuré en millisecondes (ms) pour les temps de compression et de recherche, et en octets pour la taille. Ainsi, ce graphique permet de visualiser l'évolution du temps de compression, du temps de recherche et de la taille en fonction de la longueur pour différentes profondeurs, offrant ainsi des informations précieuses pour l'optimisation des performances.

D'après les données fournies, la configuration optimale semble être une profondeur de 10 et une longueur de 100. Cette configuration offre un bon compromis entre le temps de compression, le temps de recherche et la taille de la table. Augmenter la profondeur augmente le temps de compression et de recherche mais diminue la taille de la table. Augmenter la longueur diminue le temps de compression et de recherche mais augmente la taille de la table. Le choix de la configuration optimale dépendra de vos besoins spécifiques.

7 Conclusion

7.1 Récapitulatif de la problématique et de la réalisation

Notre objectif principal était de concevoir et de mettre en œuvre une table arc-en-ciel fonctionnelle et optimisée en utilisant le langage de programmation Java. Pour ce faire, nous

avons dû relever plusieurs défis, notamment la génération de mots de passe, le stockage des hashs associés et la mise en place d'une interface graphique. Ce projet nous a permis d'approfondir nos connaissances en programmation Java, en algorithmique et en gestion de projet.

7.2 Récapitulatif des résultats

Le temps nécessaire pour générer une rainbow tables avec une longueur de 1 et une profondeur de 1000 serait d'environ 279 ms avec une taille de 102 octets, tandis que pour une chaîne avec une longueur de 10 et une profondeur de 10, ce temps serait réduit à environ 152 ms avec une taille de 831 octets.

7.3 Propositions d'améliorations

Nous avons plusieurs idées pour d'éventuelles améliorations :

Ajouter de nouveaux tests unitaires : Nous pourrions étendre notre suite de tests unitaires pour couvrir davantage de cas d'utilisation et garantir une plus grande robustesse de notre application. Cela permettrait de détecter plus facilement d'éventuels bugs ou erreurs de fonctionnement.

Version de JUnit : Envisager de passer à la dernière version de JUnit, la 5.9.2 au lieu de la 3.8.2 afin de bénéficier des dernières fonctionnalités et améliorations de JUnit, ainsi que de bénéficier d'un support actif et de correctifs de bogues.

Implémenter des optimisations connues : Il existe plusieurs techniques d'optimisation des tables arc-en-ciel, telles que l'utilisation de chaînes de Markov ou de fonctions de réduction plus complexes. Nous pourrions améliorer significativement les performances de notre application.

Ajouter un système de logging : L'intégration d'un système de logging nous permettrait de suivre et d'enregistrer les actions et les événements importants se produisant lors de l'exécution de l'application. Cela faciliterait le processus de débogage en fournissant des informations détaillées sur les erreurs, les avertissements et les activités ce qui serait utile pour diagnostiquer et résoudre les problèmes.

Ajouter la prise en charge de nouveaux algorithme de hachage : Pour rendre notre application plus polyvalente et adaptable, nous pourrions étendre son support pour inclure une variété d'algorithmes de hachage supplémentaires. Cela offrirait aux utilisateurs la possibilité de choisir parmi une gamme plus large d'options de hachage, en fonction de leurs besoins spécifiques.

Implémenter une interface de statistique : Il serait intéressant d'implémenter des options pour l'analyse et la visualisation des statistiques directement sur l'interface. Par exemple, il pourrait être possible d'afficher des graphiques pour visualiser l'efficacité des tables en fonction du temps, de la longueur, de la profondeur et des fonctions de réductions ce qui permettrait d'avoir des résultats encore plus concrets.

En mettant en œuvre ces propositions, nous pourrions rendre notre projet plus performant, plus robuste et plus adaptable aux besoins des utilisateurs.