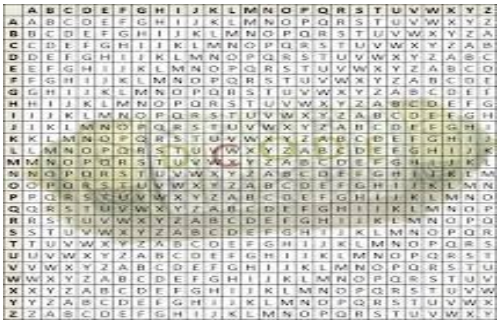
TP 1 Réseau et Sécurité



Réalisé par :

Martin BENITO-RODRIGUEZ

Imene OUAHRANI

Lien github : https://github.com/imouahrani/M1_Reseau_Securite

1. Implémentation du chiffrement de Vigenère

Puisque le TP nous demande de modifier notre programme petit à petit, on met ici des captures d'écrans que nous ne retrouvons pas dans le programme final.

Nous avons implémenté ce TP en JAVA.

Exercice 1 : Implémentation d'un programme en **JAVA** qui demande à l'utilisateur de saisir un texte, et qui l'affiche.

```
System.out.print("Input : ");
String s = new Scanner(System.in).nextLine();
System.out.println("Output: "+s);
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
ouahrani@user:~/Bureau/M1/Réseau et securité/TPs/Tp1/Prog1$ javac Prog1.java
ouahrani@user:~/Bureau/M1/Réseau et securité/TPs/Tp1/Prog1$ java Prog1
Input : CRYPTOGRAPHIE
Output: CRYPTOGRAPHIE
ouahrani@user:~/Bureau/M1/Réseau et securité/TPs/Tp1/Prog1$
```

Exercice 2 : Modification du programme pour qu'il convertisse toutes les lettres en minuscules, et qu'il enlève tous les autres caractères.

```
public static String convert(String s){
    return s.replaceAll("[\\s\\p{Punct}]","").toLowerCase();
}
```

La méthode java *replaceAll()* permet de modifier les caractère d'une chaine de caractère, ici on a remplacé les signes de ponctuations par le caractère «», ce qui permet de supprimer le caractère. Puis on applique la méthode *toLowerCase()* qui transforme les lettres majuscules en minuscules.

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
ouahrani@user:~/Bureau/M1/Réseau et securité/TPs/Tp1/Prog2$ javac Prog2.java
ouahrani@user:~/Bureau/M1/Réseau et securité/TPs/Tp1/Prog2$ java Prog2
Input : Le soleil brille
Output: lesoleilbrille
ouahrani@user:~/Bureau/M1/Réseau et securité/TPs/Tp1/Prog2$
```

Exercice 3 : Implémentation d'une fonction qui prend en entrée deux lettres minuscules (l'une étant une lettre du texte non chiffré, et l'autre une lettre de la clé), et qui retourne la lettre minuscule correspondante chiffrée.

```java
public static char cesar(char c1, char c2){

    int somme=c1+c2-97;

    if(somme>122){
        somme=somme-26;
    }

    return (char)somme;

}
```

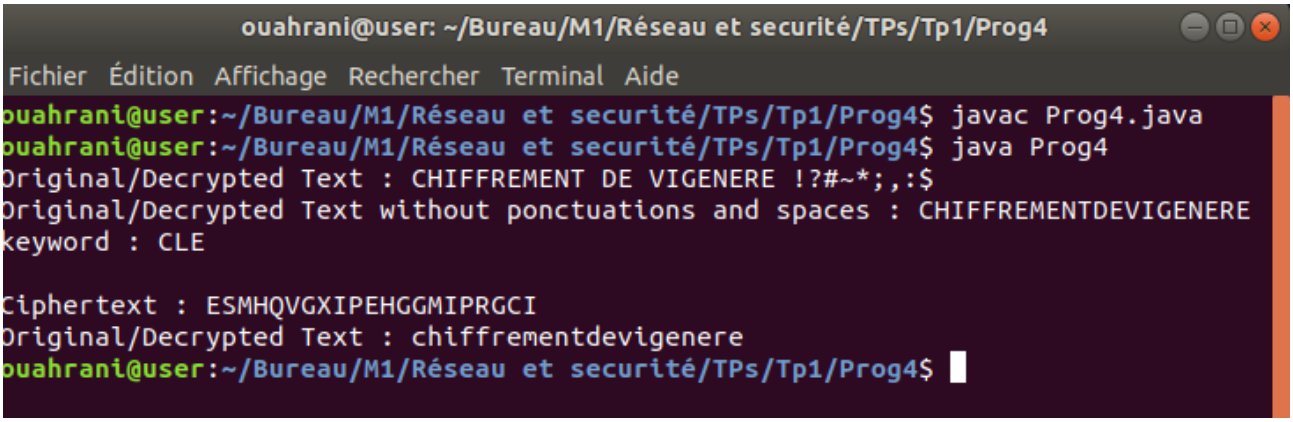
- Ici, nous exploitons la conversion des caractères et des entiers. On peut faire des opérations arithmétiques avec des entiers avec leur code ASCII, les lettres minuscules sont dans l'intervalle [97;122].

- Le 'a' minuscule est le 97^{e} caractère en ASCII, si on veut connaître la 'somme' de 2 caractères, il suffit de leur soustraire 97 pour avoir un résultat entre 97 (a+a=a => 97+97-97=97) et 244 (z+z).
Si la somme est supérieure à 122, c'est-à-dire si la somme dépasse z, on soustrait 26 pour retomber dans l'intervalle des minuscules (z+z=y => 122+122-97=147, 147-26=121).

Exercice 4 : Modification du programme pour qu'il demande à l'utilisateur deux textes (l'un étant le texte non chiffré, et l'autre la clé), qui les convertit tous les deux (selon l'exercice 2), et qui chiffre le texte avec le chiffrement de Vigenère et la clé donnée.

```java
public static String chiffreVigenere(String text, String key)
{
    text=convert(text);
    key=convert(key);

    StringBuffer buff = new StringBuffer(text);

    for(int i = 0; i < text.length(); i++)
    {
        char kIndex=key.charAt(i % key.length());
        char tIndex=text.charAt(i);
        char newChar = cesar(tIndex , kIndex );
        buff.setCharAt(i, newChar);
    }

    return buff.toString();
}
```

- Pour commencer on convertit les 2 strings pour ne pas avoir de caractères gênants. On créer aussi un buffer pour stocker une chaine de caractère de la taille du texte à chiffrer. Puis pour chaque caractère du message, on additionne avec césar un caractère de la clé. Le nouveau caractère est placé dans le buffer jusqu'à ce que la boucle chiffre chaque caractère du message. Après cela on retourne le buffer qui contient le message chiffré.

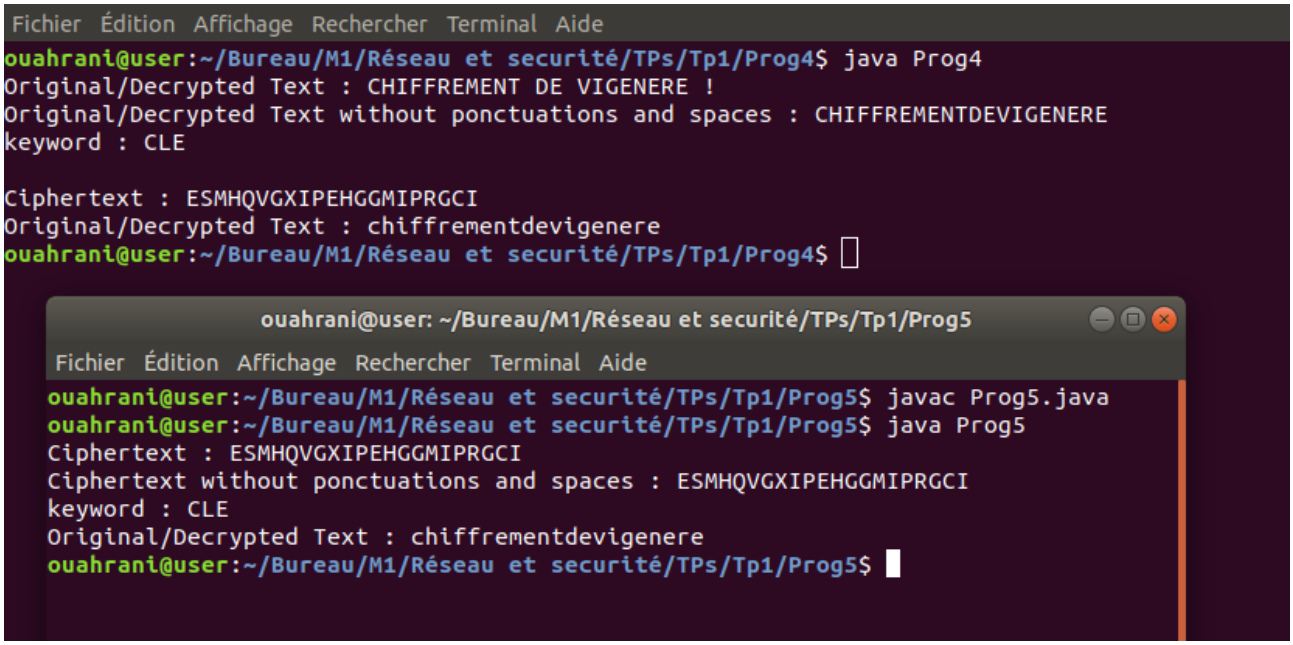


Exercice 5 : Implémentation d'un programme qui demande à l'utilisateur deux textes (l'un étant le texte chiffré, et l'autre la clé), et qui déchiffre le texte.

```java
public static String dechiffreVigenere(String text, String key)
{
    text=convert(text);
    key=convert(key);

    StringBuffer buff = new StringBuffer(text);
    int somme;

    for(int i = 0; i < text.length(); i++)
    {
        char kIndex=key.charAt(i % key.length());
        char tIndex=text.charAt(i);

        somme=tIndex-kIndex+97;
        if(somme<97){
            somme=somme+26;
        }

        buff.setCharAt(i, (char)somme);
    }

    return buff.toString();
}
```

- La structure du programme est identique à *chiffreVigenere()*. La seule différence est qu'on n'utilise pas la fonction césar. On refais le parcours de César à l'envers.

- Dans cet exercice j'ai utilisé le même principe de l'exercice précédent. Mais en commençant par le déchiffrement au lieu du chiffrement. J'ai pris le résultat de l'exécution de l'exercice précédent pour vérifier que le déchiffrement marche bien et c'est le cas car j'ai bien obtenue le résultat correspondant au texte saisie à l'exercice précédent.



2. Cryptanalyse par estimation de la longueur de la clé et analyse fréquentielle

Lorsque la longueur de la clé est connue, un chiffrement de Vigenère peut être cassé par une analyse fréquentielle.

2.1 Méthode de Babbage et Kasiki

Exercice 6 : Implémentation un programme qui prend en paramètre un texte chiffré, et qui affiche toutes les occurrences de séquences de 3 lettres ou plus qui se répètent.

Exemple :

user$ prog6

cipher: abcdefghijklmnopqrstuvwxyzabcdmnoabc

abc trouvé 3 fois

bcd trouvé 3 fois

abcd trouvé 2 fois

- Afin de répondre à cette requête j'ai utilisé la **classe buffer Reader** présente dans le **package io (Input Output)** de **JAVA8** pour la saisie au clavier d'une chaîne de caractère. J'ai fixé un mot ' pat ' représenté par un tableau qui est comparé avec la chaîne de caractère saisie au clavier. A chaque itération le mot 'pat' est glissé dans le chaîne de caractères afin de compter le nombre d'occurrence de ce mot dans la chaîne et pour cela j'ai utilisé 2 boucles imbriquées, une pour parcourir le tableau pat et un autre qui représente un itérateur de chaîne de caractère.

Exercice 7 : Modification du programme pour qu'il calcule la longueur de la clé à partir des

distances entre les répétitions.

- Afin de répondre à cette requête j'ai modifié l'exercice 6 comme indiqué. Une clé peut avoir plusieurs tailles possibles, ces tailles représentent les diviseurs de la distance entre les répétitions pour cela j'ai ajouté une fonction qui retourne tous les diviseurs d'un nombre dont j'ai fait un appel dans la méthode calculant les tailles possibles de la clé.

Résultat de l'exécution des 2 exercices 6 et 7 :

```
ouahrani@user:~/Bureau/M1/Réseau et securité/reseau_securite-main/Prog6$ javac Prog6.java
ouahrani@user:~/Bureau/M1/Réseau et securité/reseau_securite-main/Prog6$ java Prog6
Cipher:abcdefghijklmnopqrstuvwxyzabcdmnoabc

++++++++ Exercice 6 +++++++++

abc trouvé 3 fois
bcd trouvé 2 fois
abcd trouvé 2 fois
mno trouvé 2 fois

+++++++++ Exercice 7 +++++++++

La taille de la clé abc peut faire 2 tailles tels que :
  1)          1
  2)          3


La taille de la clé bcd peut faire 2 tailles tels que :
  1)          1
  2)          2


La taille de la clé abcd peut faire 2 tailles tels que :
  1)          1
  2)          2


La taille de la clé mno peut faire 2 tailles tels que :
  1)          1
  2)          2
```

Voici la mise en œuvre de l'idée :

```
// Compter les occurrences
//de pat dans txt.
import java.io.*;

class Prog6 {

    static int countFreq(String pat, String txt) {
        int M = pat.length();
        int N = txt.length();
        int res = 0;

        /* Une boucle pour faire glisser pat [] un par un */
        for (int i = 0; i <= N - M; i++) {
            /*  Pour l'index courant i, vérifier la correspondance  */
```

```
                int j;
                for (j = 0; j < M; j++) {
                        if (txt.charAt(i + j) != pat.charAt(j)) {
                                break;
                        }
                }

                // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
                if (j == M) {
                        res++;
                        j = 0;
                }
        }
        return res;
    }

    // Cette méthode compte et retourne le nombre de diviseurs
  // d'un nombre donné
  static int nbDiv(int nombre)
  {
      int n = 0;
      for(int i = 1 ; i <= nombre; i++)
              if(nombre % i == 0)
                  n++;
    return n;
  }


  // Calculer la taille de la clé
      static void keyLength(String pat, String txt) {
        System.out.printf("La taille de la clé "+pat+" peut faire %d tailles tels que :\n",
nbDiv(countFreq(pat, txt)));
```

```
  int rang = 0 ;
  for(int i = 1; i <= countFreq(pat, txt); i++)
          if (countFreq(pat, txt) % i == 0)
                  System.out.printf("%3d) %10d\n", ++rang, i) ;
System.out.printf("\n\n");
  }

  /* Programme principale */
  static public void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Cipher:");
String txt = br.readLine();



          String pat1 = "abc";
          String pat2 = "bcd";
          String pat3 = "abcd";
          String pat4 = "mno";
          System.out.println("\n+++++++++ Exercice 6 +++++++++\n");
          System.out.println("abc trouvé "+countFreq(pat1, txt)+" fois");
          System.out.println("bcd trouvé "+countFreq(pat2, txt)+" fois");
          System.out.println("abcd trouvé "+countFreq(pat3, txt)+" fois");
     System.out.println("mno trouvé "+countFreq(pat4, txt)+" fois");

     System.out.println("\n+++++++++ Exercice 7 +++++++++\n");
     keyLength(pat1, txt);
     keyLength(pat2, txt);
     keyLength(pat3, txt);
     keyLength(pat4, txt);

  }
```

}

Exercice 8 : Améliorez votre programme pour qu'il supprime les répétitions peu probables (par exemple, 10% des répétitions). Expliquez ce que vous considérez comme peu probable.

2.2 Test de Friedman

Exercice 9 : Soit Tr un grand texte, généré aléatoirement, en utilisant seulement des lettres minuscules. Quelle est la probabilité Kr que deux lettres choisies aléatoirement soient égales, dansTr ?

1/26 = 0.038

Exercice 10 : Soit Te un grand texte rédigé en anglais, et utilisant uniquement des lettres

minuscules. La probabilité que deux lettres choisies aléatoirement soient égales dans Te est environ

Ke≈0.067. Expliquez pourquoi cette valeur est différente de la valeur de l'exercice 9.

- Dans l'exo 9, le texte était généré aléatoirement, alors qu'ici le texte a été écrit en anglais. Il y a donc des lettres qui se repetent plus que d'autres, d'ou la difference de fréquence de répétitions.

Exercice 11 : Soit T un texte utilisant uniquement des lettres minuscules. Écrivere un programme qui calcule la probabilité K que deux lettres choisies aléatoirement soient les mêmes dans T.

```
public static float probaRepet(String text){

    text=convert(text);

    int size=text.length();

    int tab[ ] = new int[size];

    for(int i=0; i<size; i++){
        tab[i]=1;
    }

    for(int i=0; i<size-1; i++){
        for(int j=i+1; j<size; j++){
            if( text.charAt(i)==text.charAt(j) ){
                tab[i]++;
                tab[j]++;
            }
        }
    }
```

Dans un premier temps, on convertit la chaine de caractères et on stocke la taille de la chaine dans une variable pour gagner du temps de calcul et de la lisibilité.

Pour compter le nombre d'occurrences, on va créer un tableau d'entiers de la taille de la chaine qu'on initialise à 1. Puis la double boucle va comparer tous les caractères du string et si 2 caractères sont identiques, on incrémente la cellule i du tableau d'entiers. Si on affiche le tableau *tab* sous le tableau de caractère *text*, on verra l'occurrence de chaque caractère.

```
decode
221122
```

```
float numerateur=0;
float denominateur=0;

for(int i=0; i<size; i++){

    if(tab[i]!=1){//le caractère est il rendondant ?
        numerateur+= (float)(tab[i]*(tab[i]-1)) / (float)(size-1);
        denominateur+=(float)tab[i];
    }
}

if(numerateur==0){//on a trouvé aucune redondance
    return 0;
}
else{
    return numerateur/denominateur;
}
```

- Pour trouver la formule de probabilité de répétition, imaginons qu'un premier caractère C soit choisi aléatoirement et qu'il soit répété J fois dans un texte de N caractères. La probabilité qu'un second caractère aléatoire soit identique est de (J-1)/(N-1). On applique donc cette formule à chaque caractère possible en pondérant les fractions avec le nombre d'occurrence du caractère (ici J) ce qui nous donne J*(J-1)/(N-1) pour chaque caractère différent. Puisqu'on veut une moyenne, il faut aussi penser à diviser le numérateur qu'on vient de calculer par un dénominateur qui est la somme des facteurs de pondérations.

Exercice 12 : Le test de Friedman estime la longueur de la clé L comme (Ke-Kr)/(K-Kr). Calculez L.

L= (Ke-Kr)/(K-Kr) = (0.067-0.038)/(K-0.038) => L(K) = 0.29/(K-0.038)