Contrôle C# 2 : Cryptanalyse

Consignes de rendu

Vous devez mettre tout votre code dans le dossier `submission` qui à été créé automatiquement au début de l'examen. Pour que vos modifications soient prises en compte, vous devez lancer la commande `submission` dans un terminal. Cette commande peut être lancée autant de fois que souhaité, et ce jusqu'à la fin de l'examen.

Faites des rendus!!

Pensez à faire des rendus régulièrement, et n'attendez pas la dernière minute! On est jamais à l'abri d'un problème technique, et tout travail perdu ne pourra pas être récupéré.

Architecture

Vous devez respecter l'architecture suivante :

```
[ACDC@archlinux]$ ls
subject submission$
[ACDC@archlinux]$ tree submission
submission
|-- MidTerm2/
|   |-- MidTerm2.sln
|   |-- Cryptanalysis/
|   |   |-- Everything except bin/ and obj/
|-- README (optionnel)
|-- .gitignore
```

N'oubliez pas de vérifier les points suivants avant de rendre :

— Votre code **DOIT** compiler.
— Votre architecture de fichier **DOIT** correspondre à celle ci-dessus.
— Vous **DEVEZ** respecter les prototypes de fonctions donnés par l'énoncé.

1 Introduction

Lisez bien la totalité du sujet avant de commencer. Chaque fonction est évaluée indépendamment, si vous bloquez sur un exercice, n'hésitez pas à passer à la suite pour y revenir plus tard. **Les fonctions ne sont pas nécessairement classées par ordre de difficulté.**

Cet examen portera sur le thème de la cryptanalyse. La cryptanalyse désigne l'ensemble des techniques qui permettent de décrypter un message qui a subit un chiffrement, sans connaître la clé utilisée. Le but sera de casser deux chiffrements bien connus de tout bon programmeur : le *chiffre de César* et le *chiffre de Vigenère*. Afin de pouvoir tester vos fonctions de décryptage, vous implémenterez les chiffrements avant les algorithmes permettant de les casser. Le *chiffre de César* est un chiffrement monoalphabétique, tandis que le *chiffre de Vigenère* est un chiffrement polyalphabétique. Vous verrez, beaucoup de mécanismes utilisés dans la cryptanalyse du *chiffre de César* sont réutilisables pour le *chiffre de Vigenère*.

2 Outils de base

Nous allons commencer par implémenter des outils de base qui nous seront utiles à plusieurs reprise par la suite. Pensez à les réutiliser le moment venu !

Les fonctions décrites dans cette section sont à implémenter dans la classe `Tools`.

2.1 Modulus

Vous connaissez déjà l'opérateur "modulo" en C#. Il vérifie la propriété suivante :

```
1    b * (a / b) + (a % b) == a
```

Le résultat de `(a % b)` ressemble donc au reste dans la division euclidienne de `a` par `b`. Cela ne tient malheureusement pas dans le cas où au moins un des nombres `a` et `b` est négatif.

Vous allez donc commencer par implémenter la fonction suivante :

```
1    public static int Modulus(int a, int b)
```

Qui correspond au vrai reste (positif) dans la division euclidienne de `a` par `b`. Quelques exemples pour vous aider :

```
1    Modulus(4, 3) == 1
2    Modulus(-4, 3) == 2
3    Modulus(4, -3) == 1
4    Modulus(-4, -3) == 2
```

2.2 LetterIndex

Comme il est plus pratique de manipuler des nombres que des caractères, il nous faut une méthode pour passer de l'un à l'autre.

```
1    public static int LetterIndex(char c)
```

Cette fonction retourne la place dans l'alphabet du caractère passé en argument, la première place étant 0 : la lettre `'A'` correspond à 0, `'B'` à 1, etc... Elle ne doit pas tenir compte de la casse, et retourner `-1` si le caractère n'est pas une lettre.

2.3 RotChar

Vous devez commencer à connaître le principe. Il faudra appliquer une rotation de `n` positions dans l'alphabet au caractère `c` et le retourner.

```
1    public static char RotChar(char c, int n)
2
3    RotChar('a', 2) == 'c'
4    RotChar('A', 2) == 'C'
5    RotChar('a', 27) == 'b'
6    RotChar('c', -2) == 'a'
7    RotChar('c', -3) == 'z'
8    RotChar('4', 2) == '4'
```

Si le caractère n'est pas une lettre, il faut le retourner tel quel. Des tests unitaires sont disponible dans le fichier `Program.cs` fourni. Ils vous permettront de vous assurer du bon fonctionnement de cette fonction, qui forme la base de ce contrôle.

2.4 Histogram

Enfin, nous allons avoir besoin d'établir quelques statistiques sur des chaînes de caractères. L'utilité de la fonction qui suit peut vous paraître obscure pour le moment mais vous aurez l'occasion de l'utiliser à plusieurs reprises par la suite :

```
1    public static int[] Histogram(string str)
2
3    Histogram("Aa c") == { 2, 0, 1, 0, ... }
```

Le tableau retourné doit avoir autant de cases qu'il y a de lettres dans l'alphabet, et chaque case doit contenir le nombre d'occurences de la lettre correspondante dans `str`.

3 *Le chiffre de César*

Le *chiffre de César* est une méthode de chiffrement extrêmement simple qui aurait été utilisée par Jules César pour certaines de ses correspondances secrètes. Chaque lettre du message d'origine subit une rotation dans l'alphabet de `k` rangs, où `k` est la clef. On dit aussi que le chiffre de Cesar est un chiffrement monoalphabétique, c'est-à-dire que la clé est une seule lettre. Les exercices de cette section se dérouleront dans la classe `Caesar`.

3.1 Chiffrement et Déchiffrement

3.1.1 Constructeur

Votre constructeur respectera le prototype suivant :

```
1    public Caesar(int key)
```

Vous stockerez la valeur de l'argument `key` dans un attribut privé que vous aurez préalablement déclaré.

3.1.2 Encrypt

Le chiffrement des messages se fera par la méthode suivante :

```
1    public string Encrypt(string msg)
```

Vous devrez utiliser l'attribut dans lequel vous avez mis la clé pour chiffrer le message `msg` et retourner le résultat. Seules les lettres doivent être modifiées, les autres caractères (ponctuation, chiffres, caractères spéciaux) doivent rester inchangés.

```
1  Caesar simple = new Caesar(3);
2  simple.Encrypt("My exam is so interesting! Thanks ACDC <3")
3  // Pb hadp lv vr lqwhuhvwlqj! Wkdqnv DFGF <3
```

3.1.3 Decrypt

Passons à l'opération inverse :

```
1    public string Decrypt(string cypherText)
```

```
1  Caesar simple = new Caesar(3);
2  simple.Decrypt("Pb hadp lv vr lqwhuhvwlqj! Wkdqnv DFGF <3")
3  // My exam is so interesting! Thanks ACDC <3
```

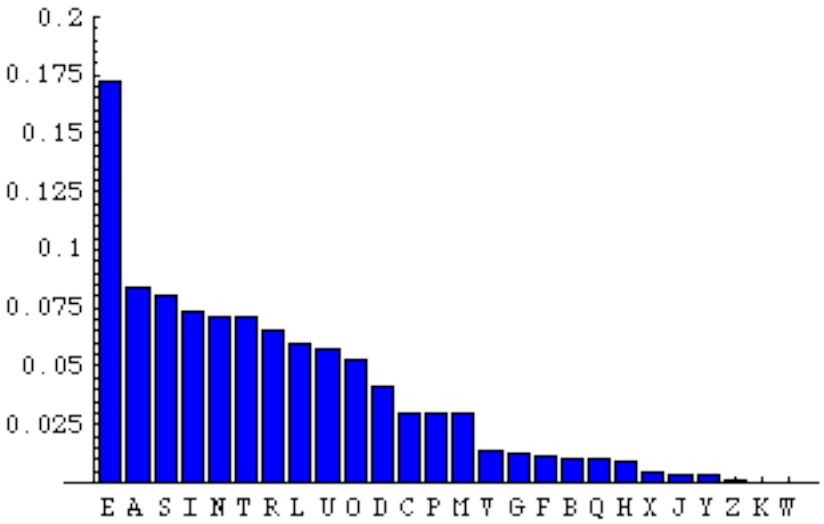
Remarque

Pour déchiffrer un message qui à été chiffré à l'aide du chiffre de César, il suffit de réappliquer l'opération de chiffrement avec l'opposé de la clef d'origine.

3.2 Casser César

Passons à la partie amusante : la cryptanalyse. La méthode la plus utilisée pour casser des chiffrements est l'analyse des fréquences. L'analyse des fréquences consiste à calculer la fréquence d'apparition de chaque lettre dans un texte. En faisant l'analyse des fréquences de plusieurs textes français, on obtient l'histogramme suivant :

La lettre la plus fréquente en français est le `e` (en anglais aussi !), donc en calculant le décalage entre la lettre la plus fréquente d'un texte chiffré et la lettre `e`, on obtient la clé de chiffrement. Cependant cette méthode a ses limites : si le texte chiffré est trop court ou que la fréquence d'apparition des lettres est trop éloignée des fréquences classiques pour une raison quelconque [1], le décryptage peut échouer.



Vous devez implémenter la fonction `GuessKey` qui détermine la clé de chiffrement d'un texte français chiffré avec la méthode de César. Vous remarquerez que cette méthode est statique, ce qui signifie qu'elle n'a pas accès à l'attribut contenant la clé, chose assez logique puisque si nous cherchons à deviner la clef c'est que nous ne la connaissons pas.

Vous vous souvenez de la fonction `Tools.Histogram()` ? Vous devez l'utilisez pour casser le chiffre de César.

```
1    public static int GuessKey(string cypherText)
```

1. Le livre La Disparition de Daniel Pennac est un livre qui ne contient aucun `e` (Oui oui). Difficile à décrypter en postulant que le `e` est la lettre la plus fréquente de la langue française...

4 Le *chiffre de Vigenère*

Vous l'aurez constaté, le *chiffre de César* n'est pas très difficile à casser. Il serait intéressant de pouvoir chiffrer un message avec une clé de plusieurs caractères, au lieu d'un simple nombre (chiffrement polyalphabétique). Le *chiffre de Vigenère*, qui utilise un mot (ou une phrase) comme clé fonctionne de la manière suivante :

La clé est étendue en la répétant autant de fois que nécessaire pour que chaque caractère du message soit associé à un caractère de la clé. On applique ensuite la méthode du chiffrement de César sur chaque caractère du message, en prenant pour clé le rang dans l'alphabet de la lettre qu'on lui a associé dans la clé. Le A vaut 0, le B vaut 1 ...

Exemple :

```
Clef: "KEY"
Message: "Hello world!"

Message :      H e l l o   w o r l d !
Clef étendue : K E Y K E ø Y K E Y K ø
Résultat :     R i j v s   u y v j n !
```

Important

Vous remarquerez que, comme pour le *chiffre de César*, les caractères qui ne sont pas des lettres sont complètement ignorés, c'est-à-dire qu'ils ne consomment pas de caractère de la clé mais sont conservés tels quels. De plus, la clé ne doit contenir que des lettres, dont on ignore la casse.

4.1 Chiffrement et Déchiffrement

4.1.1 Constructeur

Vous devez ajouter à votre classe `Vigenere` le même constructeur que pour `Caesar`, à ceci près que la clé est maintenant un `string` :

```
1    public Vigenere(string key)
```

Là encore, vous devrez stocker la valeur de `key` dans un attribut privé. Si la clé est vide ou contient autre chose que des lettres, vous devez lancer une exception de type `ArgumentException`.

4.1.2 Encrypt

Implémentez la méthode de chiffrement (référez-vous aux explications plus haut) :

```
1    public string Encrypt(string msg)
```

Voici un exemple d'utilisation :

```
1    Vigenere cypher = new Vigenere("KEY");
2
3    Console.WriteLine(cypher.Encrypt("Hello world!"));
4    // Affiche "Rijvs uyvjn!"
5    Vigenere longKey = new Vigenere("loveIsAllYouNeed");
6    cypher.Encrypt("Lorem ipsum dolor sit amet, consectetur adipiscing elit."
7                   +"Praesent mattis neque nec aliquam auctor. Morbi.");
8    // Wcmiu apdfk riysv vth vqml, czyqswgixxc oymxasntlu yymx.
9    //                Scozwmft xlrhcf ritfs iik sltbsog nygwzf. Hszti.
```

4.1.3 Decrypt

Vient ensuite en toute logique l'opération de déchiffrage, le principe est le même que ce vous avez vu précedement :

```
1    public string Decrypt(string cypherText)
```

4.2 Casser Vigenère en connaissant la longueur de la clé

Cette partie est consacrée à une des méthodes possibles pour casser le *chiffre de Vigenère*. L'analyse des fréquences comme pour le *chiffre de César* ne fonctionne pas ici : chaque lettre n'a pas le même décalage. Un histogramme des occurrences des lettres du texte n'apporte pas d'information utilisable.

Cependant, imaginez que vous connaissez la taille k de la clé qui a servi à chiffrer un texte. Il est possible de séparer le texte en prenant une lettre sur k, de façon à isoler les caractères qui ont été chiffrés avec la même lettre. À partir de là, on peut déterminer le décalage pour les k différents sous-ensembles du message. Vous l'aurez compris, cela nécessite néanmoins un texte beaucoup plus long que pour la méthode de César si l'on souhaite faire des histogrammes représentatifs.

Fonctionnement :

```
Texte original: La triche consiste a recevoir, en situation de concurrence,
une recompense indue grace a ses capacites a contourner ou enfreindre les
regles juridiques ou morales (principes de vie en societe, reglementation,
conventions), ou a trouver un moyen facile de se sortir d'une situation
desagreable par des moyens malhonnetes. Cette definition large comprend
necessairement les actes de corruption, de copinage, de nepotisme et toute
situation dans laquelle des individus ont la preference en utilisant des
criteres inappropries1.Les regles enfreintes peuvent etre explicites ou
provenir d'un code de conduite non ecrit base sur la moralite,
l'ethique ou la coutume. L'identification de la tricherie est un
processus potentiellement subjectif.La tricherie peut faire specifiquement
reference a l'infidelite.Une personne connue pour avoir triche est
qualifiee de << tricheur >>. Une personne decrite comme un << tricheur >>
ne triche pas necessairement tout le temps, mais se repose plutot sur
une tactique trompeuse au point d'en acquerir la reputation.
Clé: key
```

```
Texte chiffré et normalisé, les lettres d'une même couleur ont été chiffrées
avec le même caractère de la clé :
Verbmariayrqswroepogcfsgbilcmreerssslniayraevporaoy
lovcmskzilcigxhsokpkgckwccgyzeasxcceayrryypxipyycxjpomlnvcviqbieviqtypshgayc
csswspkpcctpsrastcchcfmcorqyggoxcbievikorrkxgyrayrtorrsslcsskxpyytovsxqmiilp
easpcniqowmbxgbhsxiqsxskxgyrbowyqvckfjotybhccqmiilcqyvlmxrcdiqMirdibojgxmrss
lvepqiayqnbilnrcmiqcegbikorrviqkgrowbogmbvszxgyrbogmzmlkkcnilotmdmqwicdxmexc
cmreerssslnelcpyaycvpcniqsrbszgnyqyrrvenbidovcxgcorsdmjswyxxbowabmrovccmlktnb
snbmccPccvcqpccilpvcsrrownoytorroxpoivzpgmmrowmetpyzcxmpnylmsbohcmslnygdilyr
cmvgdfyciqevjkqmbejsxcvirrmoeimepymssdykoPgnildmdsgydmmxhcverbmaripsiccxsxtp
ygccwsctmdildmcvpcwildwslncmxgpPydvgmlcbmczisdjysvcctcmmdsusoqcxxpojcbilmiyv
mlpmbopgdiSxinovqyrlogmxrsotmevyfsgbxpsgfoiqduskpgpmcohcdvgmlcevSxinovqyrloh
cmvgdiayqkoyldvgmlcevloxpsgfotycrcmiqcegbikorrdssdpcdikzwkkmqcipotmcinvyryxq
evsxirkgrsusoxpyqnoyqoeszsgxxborymusovgbpybinexydmmx

k: 3

Textes extraits en commançant avec le caractère:
0 - Vbrysoofbcesnyeooomzcxokkczscyyxyxonvbvtsacwkcsscfoyobvokyyosckyoxipsnobb
  xskyoqkobcicvxdMdoxsvqybnmcbovkoobzyozknodwdecesncavnssnyvboxodsxobockbbccq
  cpsooooozmoeyxnmomndymdcekbsvreemdondsdxvbrscxyccddvwdlmpdmbzdscmsoxobmvpod
  xoyoxoefbsodkpodmexoyomdyodmeosocmcbodddzkcocvyexksoyoozxomobbedx
1 - emirwegsimesirvryvsiihkgwgexeryiyjmviiiyhyssptrthmrgxiirxrrrssxyvqiepiwxh
  ixxrwvfthqiqlriiijmseiqirieirigwgvxrgmkitmixxmesepypirzyreivgrmwxwmvmtsmPvp
  ivrwyrxipmwtzmyshsyirvfivqeximipsyPimgmhemiixtgwtimpiwnxPvlmijvtmuqxjiimmpi
  ivrgrtvsxgiupmhvlvivrhviqyvlvxgtrieirspiwmitiyxviguxqyesxruvpixm
2 - raaqrpcglrrlaapalcklgspccyacarppcplcqeqpgcspcpacccqgcekrgatrlsptsmlacqmgs
  qsgbycjycmlymcqrbgrlpanlcqgkrqrbmsgbmlclmqcmcrrllyccqbgqrndccsjybarclnncccc
  lcrntrpvgrmpcplbclglcgyqjmjcromyskgldymcrapcspcsmlcclscgygccsyccdscpclylbgS
  nqlmsmygpfqsgccgcSnqlcgaklgclpfycqgkrsckkqpmnrqsrrspnqsgbysgynym

Clé obtenue avec le texte extrait, après création et étude d'un histogramme
pour chaque sous-ensemble du texte principal:
0 - k
1 - e
2 - y
```

4.2.1 FilterLetters

Pour effectuer la cryptanalyse, nous aurons besoin de normaliser le texte pour que seules les lettres soit analysées. Implémentez donc la fonction `FilterLetters` qui retourne la chaîne passée en argument privée de tous les caractères qui ne sont pas de lettres (les lettres accentuées doivent également être retirées). Cette fonction doit être écrite dans la classe `Tools`.

```
1    public static string FilterLetters(string str)
2
3    FilterLetters("01 A_bc éï #") == "Abc"
```

4.2.2 Extract

Implémentez la fonction suivante qui prend la chaîne passée en argument, supprime les `start` premiers caractères, et sélectionne un caractère sur `step`. Cette fonction doit être écrite dans la classe `Tools`. Si l'argument `step` est inférieur ou égal à zéro, la fonction doit lever une `ArgumentException`.

```
1    public static string Extract(string str, int start, int step)
2
3    Extract("0123456789", 3, 2) == "3579"
4    Extract("ABCDEFGHIJKLMNO", 5, 3) == "FILO"
```

4.2.3 GuessKeyWithLength

Le décryptage du message chiffré se fait en appliquant la même méthode que pour le chiffre de César sur chaque sous-partie du message. On entend ici par sous-partie un ensemble de lettres qui ont été chiffrées par un même caractère de la clé. Ici aussi, vous pouvez (et devez !) réutiliser les fonctions déjà implémentées plus tôt.

```
1    public static string GuessKeyWithLength(string cypherText, int keyLength)
```

4.3 Casser Vigenère

Essayons maintenant de deviner la taille de la clé. Nous allons pour cela utiliser l'*indice de coïncidence*, outil qui fût mis au point par le cryptologue William Friedman en 1920.

4.3.1 IndexOfCoincidence

Cet indice correspond à la probabilité, lorsque l'on choisit un couple de lettres dans un texte, de tomber sur deux lettres identiques. On le calcule comme suit :

$$IC = \sum_{k=0}^{25} \frac{n_k(n_k - 1)}{N(N-1)}$$

Où n_k est le nombre d'occurences de la $k^{ème}$ lettre de l'alphabet (valeur à l'indice k dans le tableau retourné par `Histogram`), et N est le nombre total de lettres dans le texte. **N'ayez pas peur**, c'est une simple somme qui parcourt chaque case de l'histogramme !

Implémentez donc la fonction qui calcule cet *indice de coïncidence*, à l'intérieur de la classe `Vigenere` :

```
1    public static float IndexOfCoincidence(string str)
```

Astuce

Pour tester votre fonction, il peut être bon de savoir que *l'indice de coïncidence* d'un texte en français tourne autour de 0,0778, et celui d'un texte aléatoire autour de 0,0385.

4.3.2 GuessKeyLength

Vous allez maintenant pouvoir implémenter la méthode du dit Friedman. Vous devez, pour chaque longueur de clé possible `k`, sélectionner une lettre sur `k` du message chiffré et calculer l'indice de coïncidence sur la sous-chaîne ainsi obtenue. Pour chaque indice calculé, on prend sa distance à l'indice de coïncidence de la langue française (utilisez la constante `FrenchIndexOfCoincidence`).

La taille de clé qui sera considérée la plus probable sera la première à être une distance de moins de 0.01, ou à défaut celle avec la plus petite distance. On choisira de ne prendre en compte que les tailles de clés plus petites que la moitié du message.

Exemple :

```
Avec le texte suivant: abcdefghijklmnopqrstuvwxy
On calcule l'indice de coïncidence des sous-chaînes suivantes:
- longueur de clé = 1 : abcdefghijklmnopqrstuvwxy
- longueur de clé = 2 : acegikmoqsuwy
- longueur de clé = 3 : adgjmpsvy
...
Et on choisi la première longueur de clé qui done un indice de coïncidence
suffisamment proche de celui du français.
```

Dans la classe `Vigenere`, implémentez la fonction suivante :

```
1    public static int GuessKeyLength(string cypherText)
```

4.3.3 GuessKey

Vous pouvez finalement recoller les morceaux et implémenter la méthode qui permet de casser le chiffrement d'un texte dans la classe `Vigenere` :

```
1    public static string GuessKey(string cypherText)
```

5 Le mot de la fin

Vous voilà arrivé à la fin du sujet. Cela ne veut pas dire que vous avez terminé pour autant ! Relisez vous et faites autant de tests que vous pouvez. Pensez à nettoyer votre code et à documenter les parties les plus obscures, il faut que votre travail soit compréhensible !

En bonus (ce n'est pas un exercice et ne sera pas noté), voici un petit texte chiffré à l'aide de l'algorithme de Vigenère :

> Evmvwo, uen z'blbz, o m'yepff fu wzbeccwu ca xongabbf, Ae koskimoj. Modg-ul, jz gbzs lif ku h'oukeirt. A'imoj gam zb womsu, a'imoj gam zb doihbxnz. Xf ee kijj dzaflrzf mfii rf kod dmls gcoxtzaqj.
>
> Jz abiccssri gst pepl gzxzg tlr hst geigfvs, Nooj rdso modf bl dzvpis, nooj eihfedms blcpb ciudh, Tvug, wotoibv, ce yct topfcv, lzg nriig diodgfvs, Ofjjtz, su ce ecvi pjis dod gfia xcnde go olio.
>
> Xf ee mshrryssri iw m'fr yi tfim evz tjacv, Nd zfj vjwmvs vi mfii rfjczberno jfis Coswlzis, Vt libed e'osiiqssri, es nvtofbz spf ur tjacv Ui pplqpsu ue ccvo vzfu vt ys ciutssv ei tmvum.
>
> Jjttjf Ilgj, Zfj Cjbuvmkzbkijbt

If you knew time as well as I do, you wouldn't talk about wasting it