TP1 : Casser un code secret

La cryptographie (en fait, on parle de chiffrement) consiste à protéger le contenu d'un message en s'aidant d'algorithmes. Elle est largement utilisée aujourd'hui afin de sécuriser les communications sur Internet (communications bancaires, militaires, ...). Dans ce projet, vous allez voir comment casser des messages initialement cryptés avec des techniques assez basiques. Vous allez voir, en tout, trois méthodes de protection de messages, et vous devrez pour chacune écrire un programme permettant de casser la protection du message. Dans chacun des cas, des messages cryptés vous seront fournis et vous devrez les déchiffrer.

Vous trouverez tous les fichiers à déchiffrer à cette adresse, dans un fichier zip :

`http://goo.gl/EzEf1J`

Dans tous les cas, seules les lettres ont été chiffrées : les nombres, les espaces et les signes de ponctuation sont inchangés.

1 Le chiffre de César

Le chiffre de César consiste à décaler, dans un message à transmettre, toutes les lettres d'un certain nombre de places. Par exemple, si le décalage vaut trois, alors la lettre 'a' devient 'd', 'b' devient 'e', ..., 'z' devient 'c'. Le destinataire du message doit connaître ce décalage afin de pouvoir l'appliquer (de façon inversée) sur le message chiffré et obtenir le message

La faiblesse de ce chiffrage est son nombre limité de possibilités... Si quelqu'un souhaite casser ce code, il peut essayer tous les décalages possibles (26 en tout) sur une partie du message, et conserver le décalage qui fait apparaître des mots intelligibles.

Ici, nous utiliserons une méthode plus simple et automatique. Les deux textes chiffrés sont des messages en français, la lettre qui y apparait le plus souvent doit donc être le 'e' du message déchiffré (en français, la lettre 'e' est celle qui apparait le plus souvent). A vous de chercher, dans les deux messages, quelle lettre apparait le plus souvent et calculer alors le décalage à appliquer. Par exemple, si vous trouvez que le message chiffré contient une majorité de 'o', cela signifie que toutes les lettres du message d'origine ont été décalées de 10 places dans l'alphabet, et que le 'e' est ainsi devenu le 'o'.

Votre programme doit lire le fichier chiffré d'entrée, trouver la lettre qui y apparait le plus souvent, et en déduire le décalage inverse à appliquer au texte pour retrouver le message d'origine. Notez bien que cette méthode, comme toutes les méthodes de déchiffrage d'un message, ne fonctionne que si le message est assez long...

Méthode à suivre :

— Faîtes une fonction qui prend en paramètre un pointeur sur fichier, et renvoie la prochaine lettre (en ignorant les espaces et ponctuations) du fichier, ou EOF si le fichier est terminé.

— Faîtes, dans votre programme, une fonction qui prend comme entrée un fichier texte et renvoie en sortie le décalage à appliquer.

2 Le chiffre de Vigenère

2.1 Fonctionnement de Vigenère

Le chiffre de Vigenère ressemble au chiffre de César, si ce n'est que le décalage à effectuer n'est pas le même selon la position de la lettre dans le texte.

Imaginons que l'on souhaite chiffrer la phrase "la programmation, c'est vraiment bien !" (juste les lettres, on ne touche pas aux signes de ponctuation). On choisit tout d'abord un mot de passe que le destinataire du message doit aussi connaître : par exemple, choisissons "java" comme mot de passe. Le mot de passe doit alors être transformé en une série de nombres, chacun correspondant à la position de la lettre dans l'alphabet : ici, "java" devient 10-1-22-1.

Pour chiffrer le message, on décalera la première lettre de 10 places, la seconde d'une place, la troisième de 22 places, la quatrième de 1 place, puis on boucle : la cinquième lettre sera décalée de 10 places, la sixième lettre sera décalée d'une place, etc...

Notre message devient alors :

```
l a    p r o g r a m m a t i o n  ,  c ' e s t   v r a i m e n t   b i e n  !
10 01  22 01 10 01 22 01 10 01 22 01 10 01 22     01   10 01 22   01 10 01 22 01 10 01 22   01 10 01 22
-------------------------------------------------------------------------------------------------------
 v b    l s y h n b w n w u s p j  ,  d ' o t p   w b b e n o o p   c s f j  !
```

On voit dans le résultat que deux lettres différentes peuvent, au final, apparaître comme la même lettre dans le message crypté.

2.2 Attaquer le chiffre de Vigenère, étape 2 (l'étape 1 est après)

Imaginons que l'on connaisse la longueur du mot de passe (ici, pour notre exemple, quatre) utilisé pour crypter le message (juste la longueur, pas le mot de passe lui-même). On pourra alors dire que, toutes les quatre lettres, le décalage est le même...

On peut alors étudier le sous-message obtenu en conservant uniquement une lettre sur quatre du message à partir de la première lettre (on ne conserve donc que la 1°, 5°, 9°, etc... lettre de notre message). Par exemple, de notre message codé " vb lsyhnbwnwuspj, d' otp wbbenoop csfj !", on obtiendrait "vywsobos" (les symboles et espaces ne sont pas comptés).

Dans ce sous-message, on sait que chaque lettre a subi le même décalage (dans notre exemple, ces lettres ont subi un décalage de 10, correspondant à la lettre "j" - la première lettre - de notre mot de passe "java"). On se retrouve alors à devoir résoudre, pour trouver le décalage, un simple problème de César. En répétant ce processus quatre fois (quatre étant la longueur du mot de passe), en prenant une lettre sur quatre à partir de la 1° lettre, puis à partir de la 2° lettre, etc... on peut déchiffrer le message entier.

2.3 Attaquer le chiffre de Vigenère, étape 1

Pour trouver la longueur du mot de passe, on va utiliser un indice statistique précieux en linguistique : l'indice de coïncidence. L'indice de coïncidence d'un texte est la probabilité, si on prend deux lettres au hasard dans un texte, que ces deux lettres soient les mêmes. Pour calculer l'indice de coïncidence d'un texte, c'est vraiment très simple.

Posons N le nombre de lettres du texte que l'on examine, N_a le nombre de 'a' dans le texte, N_b le nombre de 'b' dans le texte, etc... L'indice de coïncidence du texte est égal à

$$\frac{N_a * (N_a - 1) + N_b * (N_b - 1) + ... + N_z * (N_z - 1)}{N * (N - 1)}$$

Remarquez que le fait de décaler toutes les lettres **d'un même cran** dans le texte ne modifie pas l'indice de coïncidence du texte (donc, le codage de César ne modifie pas l'indice de coïncidence). Par contre, le fait de décaler les lettres d'un décalage différent (comme dans le code de Vigenère) modifie l'indice de coïncidence (comme on l'a dit précédemment, deux lettres différentes dans le message de départ peuvent se retrouver codées par la même lettre dans le message codé).

En quoi l'indice de coïncidence peut nous aider ? Il se trouve que si l'on met des lettres au hasard dans un texte (le texte n'aura alors aucun sens), l'indice de coïncidence de ce dernier sera à peu près égal à 0,0385. Par-contre, si vous écrivez un texte en français (vous ne choisissez donc pas les lettres au hasard), l'indice de coïncidence sera à peu près égal à 0,0785. De plus, le fait de ne prendre qu'une lettre sur n dans un texte ne modifie que très peu son indice de coïncidence.

Reprenons l'exemple d'un message chiffré en Vigenère avec le mot de passe "java". Si vous calculez l'indice de coïncidence de ce message, vous obtiendrez probablement un score aux alentours de 0,04 (car le message crypté ne ressemble en rien à du français). Si vous ne prenez qu'une lettre sur deux du message, l'indice de coïncidence du nouveau texte sera aussi assez proche de 0,04. Si vous ne prenez qu'une lettre sur trois du message, l'indice de coïncidence du nouveau texte sera aussi assez proche de 0,04. Par contre, si vous ne prenez qu'une lettre sur quatre du message (quatre étant la longueur du mot de passe), vous créez un message où toutes les lettres proviennent d'un texte en français et ont été décalées d'un même cran : vous constaterez alors que l'indice de coïncidence de ce nouveau texte sera très proche de 0,0785.

Pour résumer, on trouve la longueur du mot de passe utilisé pour chiffrer un texte en Vigenère en ne prenant qu'une lettre sur n dans le message, avec n variant de 1 à N (à vous de décider la valeur de N). A chaque fois, calculez l'indice de coïncidence de ce sous-message, et dès que ce dernier est proche de 0,0785, vous avez trouvé la longueur du mot de passe : vous pouvez alors utiliser la méthode décrite dans la section précédente pour décrypter complètement le message.

Cette méthode ne fonctionne que si le texte est assez long par rapport au mot de passe utilisé. De plus, le français possédant un indice de coïncidence très élevé, cette méthode s'applique bien pour des textes en français.

Méthode à suivre :

— Faîtes une fonction qui prend en paramètre un fichier texte, un nombre N et un nombre d, et génère un fichier obtenu en ne prenant qu'une lettre sur N du fichier d'entrée à partir de la d-ième lettre.

— Faîtes une fonction calculant l'indice de coïncidence d'un fichier texte.

3 Le chiffrement par substitution

3.1 Le principe

Le chiffrement par substitution consiste à remplacer, dans un message, chaque lettre par une autre lettre. Deux lettres égales dans le texte original doivent être chiffrées de la même manière, et deux lettres différentes doivent être chiffrées différemment. C'est comme si on prenait les lettres de l'alphabet, rangées par ordre croissant, qu'on les mélangeait (obtenant alors un nouvel alphabet) et qu'on utilisait le nouvel alphabet pour chiffrer notre message : les "a" de notre message deviendraient alors la première lettre de notre nouvel alphabet, etc...

Par exemple, considérons cette substitution :

```
lettre originale :  a b c d e f g h i j k l m n o p q r s t u v w x y z
mot de passe     :  w q a x s z c d e v f r b g t n h y j u k i l o m p
```

Les 'a' du texte original deviendrons des 'w' dans le texte chiffré, les 'b' deviendrons des 'q', etc.. Notre message sur la programmation deviendra :

```
 l a    p r o g r a m m a t i o n  ,  c ' e s t   v r a i m e n t   b i e n  !
-------------------------------------------------------------------------------------------------
 r w    n y t c y w b b w u e t g  ,  a ' s j u   i y w e b s g u   q e s g  !
```

Remarquez que le chiffrement de César est un cas particulier de chiffrement par substitution.

3.2 L'attaque

Comme pour César, on peut rechercher la lettre la plus fréquente, ce qui permettra de trouver le "e" du texte. Malheureusement, contrairement à César, trouver le "e" ne permet pas de trouver toutes les autres lettres. L'indice de coïncidence du texte ne vas pas nous aider car, comme cet indice n'est pas sensible au décalage de lettres, on risque fort de trouver une valeur proche de 0,0785.

On va se baser sur les quadgrams, qui sont tous les groupes de quatre lettres consécutives (en ignorant tous les espaces, chiffres et signes de ponctuation) dans le texte, afin de casser le code secret. Chaque quadgram est plus ou moins populaire selon la langue utilisée (par exemple, en français, TION est un quadgram très utilisé car il apparait souvent). Vous trouverez, dans le fichier zip du TP (ceci a été récupéré d'Internet), une valeur de popularité de chaque quadgram, que l'on utilisera pour tester si le décryptage réalisé donne un texte qui semble français.

Par exemple, si vous lisez dans le texte la phrase "ca va bien ?", il faudra aller chercher la valeur du quadgram "cava", celle du quadgram "avab", celle de "vabi", de "abie" et de "bien", et additionner toutes les valeurs récupérées.

Pour déchiffrer le texte, il faut trouver le bon mot de passe (qui est l'alphabet mélangé qui annulera les effets du cryptage effectué)... Pour ce faire, commencez par générer le mot de passe "abcdefgh....xyz" (c'est à dire que la lettre a est remplacée par a, b par b, etc... bref, rien ne change) et calculez le score du texte complet (en mesurant le score de chaque quadgram).

Ensuite, permutez au hasard deux lettres de votre mot de passe, et refaîtes le calcul du score : si ce dernier est meilleur que le précédent, conservez le nouveau mot de passe ; sinon, revenez au précédent. Répétez ce processus jusqu'à ce qu'aucune permutation ne permette d'améliorer le score (par exemple, si après 1000 permutations vous n'avez pas réussi à améliorer le score, considérez que vous pouvez arrêter la boucle). Vous aurez alors normalement le bon mot de passe qui vous permettra de décrypter le texte.

Méthode à suivre :

- Faîtes une fonction qui renvoie, étant donné un quadgram, son score.
- Faîtes une fonction qui prend en paramètre un fichier texte et un mot de passe, et calcule le score du fichier texte décrypté par ce mot de passe.
- Réfléchissez à comment représenter efficacement un quadgram... Peut-on représenter chaque quadgram comme un nombre entier ? Et comment stocker les valeurs de popularité des quadgrams (combien y a-t-il de quadgrams en tout ?) ?