TP7 : Casser (encore) un code secret

Ce TP consistera à utiliser des méthodes avancées d'optimisation pour décrypter un message crypté à l'aide d'une méthode par substitution.

1 Présentation de la méthode de cryptage par substitution (5mn)

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.

2 Description générale de la méthode d'attaque du message codé (10mn)

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.

3 Stocker les quadgrams dans un tableau (1h10)

Dans un premier temps, vous allez devoir construire un tableau de tous les quadgrams et leur fréquence associée d'apparition.

Questions

1. On souhaiterait pouvoir utiliser les quadgrams de la sorte :

```
//On definit qt, pour stocker qt

/* On ecrit du code pour remplir qt */
...

/*Une fois qt initialise, on veut l'utiliser ainsi*/
//Si on souhaite recuperer la valeur de frequence du quadgram ATTE, on fera
double val = qt[a-'a'][t-'a'][t-'a'][e-'a'];
```

   Que doit être le type de *qt* ? Déclarez *qt* dans votre code. Quel sera la taille mémoire utilisée par ce type de donnée ? Est-ce que cela semble raisonnable par rapport à la mémoire disponible sur votre ordinateur ?

2. Le score de chaque quadgram sera stocké à une position spécifique dans *quadgram_tab*.
   Ecrivez un programme qui lit le fichier *english_quadgrams.txt* et remplit *quadgram_tab* avec tous les score des quadgrams contenus dans ce fichier. Attention, le fichier contient certains quadgrams en double (cela vient du fait que les accents des quadgrams ont été retirés). Vous devrez, dans le cas où vous trouverez plusieurs quadgrams égaux, additionner leurs scores.
   Testez ensuite votre tableau en vérifiant que vous êtes capable de trouver le score d'un quadgram donné.

3. Écrivez la fonction **valeur_quadgram** qui prend en paramètre une structure *quadgram_tab* ainsi que quatre caractères $c1, c2, c3, c4$ et renvoie la valeur de fréquence du quadgram $c1c2c3c4$. Si le quadgram a une fréquence nulle, on renverra 1 pour des raisons pratiques qui s'expliqueront plus tard.

4 Attribuer un score à un décryptage, et maximiser ce score (1h35)

Nous allons maintenant nous attaquer à casser le cryptage du texte en anglais (le plus simple). Pour ce faire, on va commencer par écrire une fonction qui attribuera un score à un texte. Le score que l'on attribuera à un texte sera égal à la somme des log des scores de chacun des quadgrams du texte (on pourrait faire la somme des score des quadgrams directement, mais on gardera les logs au vu des différences importantes qu'il existe entre les scores des différents quadgrams).

Par exemple, on aura

$$score(\text{"}COUCOU\text{"}) = log(score(\text{"}COUC\text{"})) + log(score(\text{"}OUCO\text{"})) + log(score(\text{"}UCOU\text{"}))$$

Attention, on aura aussi

$$\begin{aligned} score(\text{"}CAVABIEN\text{"}) = {} & log(score(\text{"}CAVA\text{"})) + log(score(\text{"}AVAB\text{"})) + log(score(\text{"}VABI\text{"})) \\ & + log(score(\text{"}ABIE\text{"})) + log(score(\text{"}BIEN\text{"})) \end{aligned}$$

Questions

1. Écrivez la fonction **score** qui calcule le score d'un texte donné et le retourne sous forme d'un double. Le texte sera donné sous forme d'une chaîne de caractères. Voici le prototype de la fonction :

```
double compute_score(char *input, quadgram_tab tabquad)
```

2. Écrivez une fonction **permute** qui prend un mot de passe déjà existant (un tableau de 26 caractères) et permute, au hasard, deux lettres de ce mot de passe.

3. Écrivez une fonction **decrypt** qui prend un mot de passe et un texte, et réalise le décryptage de ce texte par le mot de passe dans un tableau de sortie passé en paramètre (rappel : ici, les mots de passe sont toujours des tableaux de 26 caractères). Voici le prototype de la fonction :

```
void decrypt(char *input, char *output, char *password)
```

4. La fonction de décryptage sera ainsi : on part d'un mot de passe M (par exemple, l'alphabet classique) et on calcule le score S du texte décrypté par M. Ensuite, on génère M' par permutation de deux lettres au hasard de M, et on décrypte le texte d'origine avec M'. Si le score S' de ce nouveau texte est meilleure que S, alors on conserve ce mot de passe. On réalise cette opération plusieurs fois (jusqu'à, par exemple, ne pas trouver d'amélioration de score pendant 1000 itérations consécutives).

$M = abcdefghijklmnopqrstuvwxyz$
$pas\_mieux = 0$
tant que $pas\_mieux < 1000$ **faire**
  $M' = permute(M)$
  $s = score(decrypt(texte, M))$
  $s' = score(decrypt(texte, M'))$
  si $s' > s$ **alors**
    $M = M'$
    $pas\_mieux = 0$
  sinon
    $pas\_mieux++$
  fin
fin
$Afficher(decrypt(texte, M))$

Algorithme 1 : Première méthode de décryptage : la descente de gradient

En utilisant les fonctions précédemment codées, écrivez la procédure de décryptage correspondant à l'algorithme **??**. Testez-la sur les deux textes en anglais (le long et le court), puis sur le texte en français. Que constatez-vous ? Pourquoi, à votre avis, le texte en français ne se décrypte pas, et le texte anglais court se décrypte moins souvent ?

5 Rendre votre travail

Vous devez rendre le travail réalisé sur Moodle.
Ce rendu doit suivre un format spécifique. Votre programme devra s'exécuter à l'aide de la commande

```
./decryptage_substitution entree.txt fichier_quadgram.txt sortie.txt
```

— **entree.txt** doit être le fichier codé en entrée.
— **fichier_quadgram.txt** doit être le fichier quadgram à utiliser.
— **sortie.txt** contiendra le fichier décodé en sortie..
— Si votre programme ne parvient pas à se terminer correctement (fichier d'entrée introuvable, mauvais nombre de paramètres en entrée du programme, etc), il devra renvoyer **EXIT_FAILURE**. Sinon, il devra se terminer avec le code **EXIT_SUCCESS**.

Votre programme devra être compilé à l'aide de la commande

```
gcc decryptage_substitution.c -lm -o decryptage_substitution
```

Si votre ligne de compilation est plus complexe, vous devrez alors la spécifier dans un *Makefile*. Tous vos fichiers devront être directement placés dans un fichier zip (et ne pas être dans un sous dossier du fichier zip), dont le nom sera

```
VotreNom_VotrePrenom_VotreNumeroEtudiant.zip
```

sans aucun espace (si votre nom ou votre prénom contiennent un espace, ne les faîtes pas figurer). Les formats de compression acceptés sont .zip, .tgz, .tar.gz, .7z, .rar.

Si vous avez un binôme, il vous faudra écrire son nom, prénom et numéro d'étudiant séparés par des espaces, en plus des vôtres, dans un fichier *binome.txt*, dont le format sera

```
Nom1 Prenom1 Numero_etudiant1
Nom2 Prenom2 Numero_etudiant2
```

Vous ne devez rendre le fichier qu'une seule fois par binôme.