INFO0902 - Structures de données et algorithmes

Projet 3: Résolution de problèmes

Jean-Michel Begon, Colin Smetz, Pierre Geurts

15 avril 2016

Dans ce projet, nous allons appliquer les techniques de résolution de problèmes dans le cadre de la cryptanalyse ; nous allons tenter de mettre au point un algorithme permettant de déchiffrer un texte sans connaissance *a priori* de la clé utilisée pour le chiffrer.

Remarque : L'algorithme de chiffrement proposé ci-dessous est inspiré d'un algorithme réel imaginé il y a plus de 200 ans par un mathématicien américain dénommé Robert Patterson. Ce dernier a utilisé cet algorithme pour chiffrer un message qu'il a envoyé à son ami, le président Thomas Jefferson. Ce message a seulement été décrypté en 2007 en utilisant la programmation dynamique (voir par exemple `http://www.wsj.com/articles/SB124648494429082661` *)*

1 Algorithme de chiffrement

Le chiffrement est composé de quatre étapes : le filtrage, la mise en forme, la transposition et le décalage.

1.1 Filtrage

La première étape du chiffrement consiste à enlever tous les caractères non-alphabétiques (espaces, sauts de ligne, ponctuations, etc.) et les majuscules. Par exemple la chaîne suivante (extraite des conditions d'utilisation de Google)

```
We collect information to provide better services to all of our users - from figuring out basic stuff like which language you speak, to more complex things like which ads you'll find most useful, the people who matter most to you online, or which YouTube videos you might like.
```

se transforme en :

```
wecollectinformationtoprovidebetterservicestoallofourusersfromfiguringoutbasicstufflike
whichlanguageyouspeaktomorecomplexthingslikewhichadsyoullfindmostusefulthepeoplewhomat
termosttoyouonlineorwhichyoutubevideosyoumightlike
```

Soit une chaîne unidimensionnelle de 223 caractères.

1.2 Mise en page

La deuxième étape consiste à réorganiser la chaîne unidimensionnelle sous la forme d'un tableau à deux dimensions de taille $N \times P$. Le nombre de colonnes P (qui sera aussi la taille de la clé) est fixé a priori comme un paramètre de l'algorithme de chiffrement. Soit T la longueur du texte après filtrage ($T = 223$ dans notre exemple). Le nombre de lignes N est donné par $N = \lceil T/P \rceil$.

Si $T < NP$, on complétera la dernière ligne avec des caractères choisis de manière arbitraire (par exemple aléatoirement).

Pour une longueur de ligne de $P = 15$ caractères, l'exemple précédent deviendrait :

```
wecollectinform
ationtoprovideb
etterservicesto
allofourusersfr
omfiguringoutba
sicstufflikewhi
chlanguageyousp
eaktomorecomple
xthingslikewhic
hadsyoullfindmo
stusefulthepeop
lewhomattermost
toyouonlineorwh
ichyoutubevideo
syoumightlikepa
```

où on voit que les deux lettres "pa" ont dû être ajoutées à la fin du dernier mot pour compléter le tableau. Le texte peut donc être représenté par un tableau de taille 15×15.

1.3 Transposition

La troisième étape consiste à transposer le tableau précédent afin de le lire colonne après colonne, de bas en haut et de gauche à droite :

```
waeaoscexhsltis
ettlmihatateocy
citlfclkhduwyho
ooeoisatisshoyu
lnrfgtnonyeouom
ltsouugmgofmoui
eoeurfuosuuantg
cprrifarllltluh
trvunlgeilttibt
ioisgieckfhenel
nvceokyoeierevi
fierueomwnpmoik
odsstwuphdeorde
retfbhslimoswep
mboraipecopthoa
```

Le tableau compte maintenant P lignes de taille N, où la j-ème ligne L_j $(0 \leq j \leq P-1)$ contient les caractères $L_j = c_0^{(j)}\, c_1^{(j)}\, \ldots\, c_i^{(j)}\, \ldots\, c_{N-1}^{(j)}$ $(0 \leq i \leq N-1)$. Par exemple, le caractère $c_4^{(0)}$ (ligne 0, colonne 4) est "o".

1.4 Décalage

La dernière étape consiste à décaler circulairement les lignes. Un décalage circulaire $\Delta_q(L_j)$ de q positions appliqué à la j-eme ligne donnerait la suite de caractères :

$$\Delta_q\left(c_0^{(j)}\, c_1^{(j)}\, \ldots\, c_i^{(j)}\, \ldots\, c_{N-1}^{(j)}\right) = c_{-q \bmod N}^{(j)}\, c_{(1-q) \bmod N}^{(j)} \cdots c_{(i-q) \bmod N}^{(j)} \cdots c_{(N-1-q) \bmod N}^{(j)}$$

Le décalage est appliqué de manière indépendante à chaque ligne. Pour ce faire, on tire aléatoirement un entier compris entre $-R$ et $+R$ ($R \in \mathbb{N}$) pour chaque ligne. Par exemple, pour $R = 3$, une séquence de décalage pourrait être -3, 2, -3, 1, -1, 2, -3, 3, -1, 0, -2, 3, -2, 3, 0, ce qui se traduirait par le texte final :

```
aoscexhsltiswae
cyettlmihatateo
lfclkhduwyhocit
uooeoisatisshoy
nrfgtnonyeouoml
uiltsouugmgofmo
urfuosuuantgeoe
luhcprrifarlllt
rvunlgeilttibtt
ioisgieckfhenel
ceokyoeierevinv
oikfierueomwnpm
sstwuphdeordeod
wepretfbhslimos
mboraipecopthoa
```

2 Cryptanalyse

Connaissant l'algorithme de chiffrement, la cryptanalyse revient à examiner le texte chiffré final afin de réaligner les lignes. Une fois le bon alignement trouvé, il est aisé de se ramener au texte filtré. La dernière étape consiste alors à segmenter le texte filtré afin d'individualiser les mots. La manière d'aborder ces deux problèmes est discutée ci-dessous.

2.1 Recherche de l'alignement optimal

Une approche pour retrouver l'alignement repose sur la définition d'une métrique décrivant la qualité de l'alignement entre deux lignes successives. Le but étant alors de trouver l'alignement (global) qui maximise cette adéquation pour toutes les paires de lignes contiguës.

Dans le cadre de ce projet, la métrique sera basée sur l'analyse statistique des bigrammes (paires de lettres). A supposer que nous connaissions la probabilité d'apparition de chaque paire de lettres dans un texte (qui dépend bien entendu de la langue dans laquelle ce texte est écrit), on peut évaluer la qualité de l'alignement de deux lignes successives j et $j+1$ ($0 \leq j \leq P-2$) respectivement décalées de q_1 et q_2 positions par la fonction suivante :

$$S(j, q_1, q_2) = \sum_{i=0}^{N-1} \log\left(P_{bigram}\left(c^{(j)}_{(i-q_1) \bmod N}, c^{(j+1)}_{(i-q_2) \bmod N}\right)\right)$$

où $P_{bigram}(a, b)$ est la probabilité d'observer le caractère a suivi du caractère b dans un texte quelconque. Le score d'un alignement sera donc d'autant plus élevé que les paires de caractères qui découlent de cet alignement auront une probabilité d'apparition élevée. Les probabilités $P_{bigram}(a, b)$ sont en général inconnues mais elles peuvent par exemple être estimées sur base d'un grand nombre de textes écrits dans la même langue que le message chiffré.

Au final, retrouver l'alignement optimal consiste donc à trouver *efficacement* le vecteur de décalage $Q^* = (q_0^* \dots q_{p-1}^*)^T \in \mathcal{R} = \{-R, \dots, 0, \dots R\}^P$ vérifiant :

$$Q^* = \arg\max_{Q \in \mathcal{R}} \sum_{j=0}^{P-2} S(j, q_j, q_{j+1}). \tag{1}$$

2.2 Segmentation des mots

Une fois l'alignement optimal retrouvé, il est aisé de récupérer le texte filtré en transposant la matrice de caractères obtenue. Pour que le texte soit lisible, il reste encore à séparer la chaîne de caractères en ses différents mots successifs [1]. Le problème est rendu difficile notamment par le fait qu'une même chaîne peut être séparée de plusieurs manières possibles en une suite de mots corrects de la langue sans pour autant avoir du sens (La sous-chaîne "eventsitbecomes" pourrait par exemple correspondre à "event sit be comes" ou à "events it becomes").

Pour lever une partie de l'ambiguïté, on supposera qu'on dispose d'une structure de données contenant pour chaque mot de la langue ciblée un poids reflétant sa fréquence d'apparition dans des textes de cette langue. Cette structure de données sera accessible par une fonction `Frequency(w)` prenant en entrée un mot `w` (une chaîne de caractère) et renvoyant le poids f du mot. La procédure de séparation des mots d'une chaîne consistera alors à rechercher la segmentation maximisant le produit des poids des mots résultants (ou de manière équivalente la somme du logarithme de ces poids).

Plus formellement, soit $T[1\,..\,L]$ le tableau contenant le texte à segmenter de longueur L. Il s'agit de déterminer le nombre de coupures de ce texte, noté l, et la position de ces l coupures, notées $c_1, c_2, \ldots, c_l$, avec $1 < c_1 < c_2 < \ldots < c_l \leq L$, qui maximisent la somme suivante :

$$\sum_{i=0}^{l} \log\left(\texttt{Frequency}(T[c_i\,..\,c_{i+1}-1])\right), \tag{2}$$

où $c_0 = 1$ et $c_{l+1} = L + 1$.

Remarque : Dans le cas où un mot n'apparaît pas dans la structure de données, la fonction `Frequency` doit quand même renvoyer une valeur cohérente. Une stratégie simple est d'associer la valeur de $10/N_c \times 10^{-s}$, où s est la taille du mot recherché et N_c est le nombre de mots dans le corpus ayant servi à estimer les fréquences. De la sorte, un mot de taille 1 obtient le même poids qu'un mot qui n'apparaît qu'une seule fois et les longs mots sont fortement pénalisés.

3 Analyse théorique

Avant de passer à l'implémentation, on se propose de réfléchir à la pertinence de l'approche de cryptanalyse et de proposer et analyser plusieurs solutions algorithmiques pour les deux étapes : alignement et segmentation.

3.1 Alignement

1. Discutez la pertinence de l'approche proposée, consistant à calculer le vecteur de décalage Q^* en utilisant (1). Est-ce que le vecteur de décalage Q^* ainsi obtenu est unique ? Est-on certain de retrouver exactement le vecteur de décalage effectivement utilisé pour le chiffrement ? Discutez l'impact à la fois de la fonction de score S et du vecteur de décalage sur ces deux questions.
   Suggestion : réfléchissez en particulier au cas où le vecteur de décalage utilisé pour le chiffrement est le vecteur nul. Quelle sera la solution Q^* obtenue dans ce cas ?
2. Expliquez brièvement le principe d'une approche par recherche exhaustive pour trouver le vecteur Q^* selon (1) et exprimez sa complexité.
3. La recherche de Q^* peut s'effectuer par programmation dynamique en s'appuyant sur la fonction de coût $M(i, q)$ (avec $0 \leq i \leq P-1$ et $q \in \{-R, \ldots, 0, \ldots, R\}$), définie de la manière suivante :

$$M(i, q) = \max_{q_0, \ldots, q_{i-1}} \sum_{j=0}^{i-2} S(j, q_j, q_{j+1}) + S(i-1, q_{i-1}, q).$$

1. On n'abordera pas ici le problème de la récupération de la ponctuation.

$M(i, q)$ est donc le score d'alignement maximum que l'on peut obtenir en décalant la ligne i de q positions et optimisant les décalages $q_0, \ldots, q_{i-1}$ des lignes précédentes.

(a) Donnez une formulation récursive de la fonction de coût $M(i, q)$ en précisant bien le(s) cas de base.

(b) Représentez le graphe des appels récursifs pour un problème de petite taille.

(c) Donnez le pseudo-code d'un algorithme efficace de calcul des valeurs de cette fonction (par l'approche ascendante ou descendante).

(d) Modifiez ce pseudo-code pour récupérer la solution optimale.

(e) Donnez la complexité en *temps* et en *espace* de cet algorithme.

Remarques :

— Les complexités demandées ci-dessus devront être exprimées en fonction de N, la taille des lignes, de P, le nombre de lignes et de R, le décalage maximum d'une ligne. Rechercher la probabilité d'un bigramme est $\Theta(1)$ (accès dans un tableau).

— Il y a plusieurs degrés de liberté pour implémenter la solution par programmation dynamique. Nous accorderons une attention particulière aux optimisation réalisées qui mèneront à une diminution de la complexité asymptotique par rapport à des implémentations plus directes.

3.2 Segmentation

1. Expliquez brièvement le principe d'une approche par recherche exhaustive pour trouver la meilleure segmentation et exprimez sa complexité.
2. Proposez une approche gloutonne pour résoudre le problème (sans donner de pseudo-code) et montrez par un contre-exemple qu'elle ne permettra pas de maximiser la somme (2) dans le cas général où on ne fait pas d'hypothèse sur les fréquences des mots.
3. Proposez une solution optimale au problème utilisant la programmation dynamique en passant par les étapes suivantes :
   (a) Soit $M[L]$ la somme du logarithme des poids des mots d'une segmentation optimale de la chaîne $T[1\,..\,L]$. Formulez $M[L]$ de façon récursive. Précisez le(s) cas de base.
   (b) Donnez le pseudo-code d'un algorithme efficace de calcul des valeurs de cette fonction (par l'approche ascendante ou descendante).
   (c) Modifiez ce pseudo-code pour récupérer la segmentation optimale.
   (d) Donnez la complexité en *temps* et en *espace* de cet algorithme en fonction de la taille L du texte.

Remarque : Vous pouvez supposer dans vos développements que la longueur l_{max} du mot le plus long dans la structure est connue.

4 Implémentation

Contrairement aux précédents projets, nous ne vous fournissons pas de fichiers *header* mais un `Makefile` à compléter. Celui-ci doit comporter au moins cinq *target*s : `all`, `cipher`, `decipher`, `shifts` et `segment`. `all` est la commande par défaut qui appelle les quatre autres. Ces dernières compilent les programmes de mêmes noms.

4.1 Le programme `cipher`

Le programme `cipher` chiffre un texte donné. Il prend deux arguments (dans cet ordre) :

1. `plain_text` : un chemin vers le texte à chiffrer. Nous vous fournissons le fichier `plain_text.txt` à titre d'exemple. Vous pouvez supposer que les lignes du fichier texte ne dépassent jamais 1024 caractères.
2. `key` : un chemin vers le fichier contenant la clé. Celui-ci doit contenir les P entiers représentant les décalages sur sa première ligne. Les nombres doivent être délimités par des espaces.

Ce programme affiche le résultat du chiffrement sur la sortie standard (un bloc de P lignes, chacune de taille N). Les caractères de *paddings* peuvent être choisis arbitrairement.

4.2 Le programme `decipher`

Le programme `decipher` déchiffre un texte donné. Il prend deux arguments (dans cet ordre) :

1. `ciphered_text` : un chemin vers le texte à déchiffrer (un bloc de P lignes, chacune de taille N).
2. `key` : un chemin vers le fichier contenant la clé au même format que pour le chiffrement.

Ce programme affiche le résultat du déchiffrement sous la forme d'une chaîne unidimensionnelle sans espace sur la sortie standard.

4.3 Le programme `shifts`

Le programme `shifts` calcule le décalage optimal par programmation dynamique. Il prend trois arguments (dans cet ordre) :

1. `max_shift` : un entier positif représentant le décalage maximum R d'une ligne.
2. `ciphered_text` : un chemin vers le texte à déchiffrer (un bloc de P lignes, chacune de taille N).
3. `bigrams` un chemin vers un fichier contenant les bigrammes. Celui-ci doit contenir un bigramme par ligne. Les deux caractères du bigramme doivent être séparés de sa probabilité (un `double`) par un point-virgule. Nous vous fournissons le fichier `bigrams.csv` qui correspond à l'anglais.

Ce programme affiche deux lignes sur la sortie standard. La première contient les décalages trouvés : les P entiers sont séparés par des espaces. La seconde ligne contient le score associé aux décalages renvoyés. La redirection de la sortie standard vers un fichier est donc directement exploitable par le programme `decipher`.

4.4 Le programme `segment`

Le programme `segment` permet de découper une chaîne unidimensionnelle en mots. Il prend deux arguments (dans cet ordre) :

1. `line_txt` : un chemin vers la chaîne unidimensionnelle.
2. `words` : un chemin vers un fichier contenant pour chaque mot son nombre d'occurrences dans un corpus de textes de la langue cible. Celui-ci doit contenir une paire mot-comptage par ligne. Les deux éléments de la paire sont séparés par un point-virgule. Nous vous fournissons le fichier `words.csv` qui correspond à des textes anglais.

Ce programme affiche deux lignes sur la sortie standard. La première est la chaîne unidimensionnelle segmentée et la seconde est le score correspondant.

Comme structure pour stocker les mots et leur comptage, nous vous fournissons une table de hachage (`HashTable.h` et `HashTable.c`).

Notez que pour dériver les fréquences à partir des comptages, il vous faudra diviser le nombre d'occurences de chaque mot par la somme totale de ces nombres sur tous les mots. Cette somme totale vous donnera également le paramètre N_c vous permettant de calculer la fréquence associées aux mots non présents dans le fichier de comptage (voir la remarque à la section 2.2).

Remarques Nous vous fournissons également le fichier `ciphered.txt` qui est le chiffrement d'un texte en anglais (autre que celui contenu dans `plain_text.txt`). Pour le décoder, il faut d'abord utiliser le programme `shifts` afin de trouver les décalages, ensuite utiliser ceux-ci avec le programme `decipher` et finalement utiliser le programme `segment` pour séparer les mots.

Les programmes ne peuvent afficher sur la sortie standard que les éléments mentionnés. En cas d'erreur, veillez à bien rediriger les messages vers `stderr`. Assurez-vous aussi que les programmes renvoient un code d'erreur en cas de problème (*i.e.* un nombre différent de zéro) et quitte proprement (libération des ressources, etc.) ; ils ne doivent pas s'arrêter brusquement en cas d'erreur de segmentation, par exemple.

Veillez également à bien découper votre code pour éviter les redondances tout en différenciant les parties "bibliothèques" (pas de `printf`, etc.) du "*business code*" (robustesse face aux entrées, etc.).

Finalement, notez que les *flags* de compilation ne peuvent être modifiés.

5 Deadline et soumission

Le projet est à réaliser **par groupe de deux** pour le **vendredi 13 mai 2015** à **23h59** au plus tard. Le projet est à remettre via la plateforme de soumission :

`http://submit.run.montefiore.ulg.ac.be/`.

Il doit être rendu sous la forme d'une archive `tar.gz` contenant :

— Votre rapport (8 pages maximum) au format PDF. Soyez bref mais précis et respectez bien l'ordre des (sous-)questions.

— Le `Makefile` modifié.

— Les fichiers sources.

Concaténez vos identifiants sxxxxxx pour les noms des groupes.

Vos fichiers seront évalués sur les machines ms8xx avec les implications habituelles :

— Le projet doit être réalisé dans le standard C99.

— La présence de *warnings* impactera négativement la cote finale.

— Un projet qui ne compile pas avec cette commande sur ces machines recevra une cote nulle (pour la partie code du projet).

Un projet non rendu à temps recevra une cote globale nulle. En cas de plagiat avéré, l'étudiant se verra affecter une cote nulle à l'ensemble des projets.

Les critères de correction sont précisés sur la page web des projets.

Bon travail !