TD6 : Algorithme, invariant de boucle et complexité (2/2)

Exercice 1 : Algorithme de chiffrement d'une chaîne de caractères

Considérons l'algorithme de chiffrement de chaînes de caractères dit « de Vigenère ». Il consiste à « additionner » les caractères du texte à chiffrer avec ceux d'une clé de chiffrement. Par exemple, le chiffrement de la chaîne « Cherchez au pied de l'arbre » avec la clé « indice » peut s'illustrer ainsi :

- on place la clé en regard du texte à chiffrer, en répétant la clé autant de fois que nécessaire pour couvrir le texte, et en ignorant les caractères qui ne sont pas des lettres (ils seront laissés inchangés par l'algorithme de chiffrement) :

C	h	e	r	c	h	e	z		a	u		p	i	e	d		d	e		l	'	a	r	b	r	e
i	n	d	i	c	e	i	n		d	i		c	e	i	n		d	i		c	'	e	i	n	d	i

- on remplace chaque lettre de la clé par sa position dans l'alphabet (0 pour 'a', 1 pour 'b'…),

C	h	e	r	c	h	e	z		a	u		p	i	e	d		d	e		l	'	a	r	b	r	e
8	13	3	8	2	4	8	13		3	8		2	4	8	13		3	8		2	'	4	8	13	3	8

- on remplace chaque lettre du texte à chiffrer par la lettre située *d* positions plus loin dans l'alphabet, où *d* est le nombre indiqué par la clé (si on dépasse z, on reboucle sur a, b etc.) :

C	h	e	r	c	h	e	z		a	u		p	i	e	d		d	e		l	'	a	r	b	r	e
K	u	h	z	e	l	m	m		d	c		r	m	m	q		g	m		n	'	e	z	o	u	m

Notez qu'une lettre identique dans le texte à chiffrer ne produit pas forcément le même code (ex. le premier et deuxième e de Cherchez donnent respectivement un h et un m). Et notez qu'un même code n'est pas forcément issu d'une même lettre (ex. les deux m successifs du troisième mot sont issus d'un i et d'un e). Cela rend le processus de décryptage très difficile sans la clé.

a. Si *x* est le code ASCII d'une lettre à chiffrer et *y* le code ASCII de la lettre de la clé située en regard, quelle formule permet de calculer le code ASCII résultant ? On supposera pour cette question que les deux lettres sont des minuscules. Indice : que vaut *y* – 'a' ?

La formule est : ((x – 'a' + y – 'a')%26) + 'a'
En effet, on peut écrire le tableau suivant où on choisit y comme étant la lettre c:

x	x-'a'	y	y-'a'	((x–'a'+y–'a')%26	((x–'a'+y–'a')%26 + 'a'
a	0	c	2	2	c
b	1	c	2	3	d
c	2	c	2	4	e
…	…	…	…	…	…
x	23	c	2	25	z
y	24	c	2	0	a
z	25	c	2	1	b

La formule est équivalente à ((x + y – 2 * 'a')%26) + 'a' que l'on utilisera dans le code.

b. Ecrire en langage algorithmique la procédure de chiffrement, dont voici l'entête :

Procédure chiffrer (texte : chaîne de caractères, cle : chaîne de caractères, result : chaîne de caractères)
Préconditions : texte contient un ou plusieurs caractères suivis d'un caractère de terminaison '\0'. cle contient une ou plusieurs lettres minuscules suivie de '\0'. result est une chaîne déjà allouée en mémoire, assez grande pour contenir le texte crypté (incluant la place pour le caractère de terminaison).
Postcondition : result contient la version cryptée de texte. Seules les lettres (majuscules ou minuscules) non accentuées sont cryptées, les autres caractères sont recopiés tels quels.

Paramètres en mode donnée : texte, cle
Paramètre en mode donnée-résultat : result

Procédure chiffrer (texte : chaîne de caractères, cle : chaîne de caractères, result : chaîne de caractères)
Préconditions : cf énoncé
Postcondition : cf énoncé
Paramètres en mode donnée : texte, cle
Paramètre en mode donnée-résultat : result
Variables locales : i, j entiers
Début

```
1    i ← 1 {rappel : les tableaux/chaînes sont indexés à partir de 1 en langage algorithmique}
2    j ← 1
3    Tant que (texte[i] ≠ '\0') Faire
4        Si (texte[i] >= 'a') et (texte[i] <= 'z')  Alors   {lettre minuscule}
5            result[i] ←  ((texte[i] + cle[j] - 2*'a') mod 26) + 'a'
6            j ← j +1
7            Si (cle[j] = '\0') Alors j ←  1 FinSi
8        Sinon
9            Si (texte[i] >= 'A') et (texte[i] <= 'Z') Alors  {lettre majuscule}
10               result[i] ←  ((texte[i] - 'A' + cle[j] - 'a') mod 26) + 'A' {rappel: cle est toujours minuscule}
11               j ← j + 1
12               Si (cle[j] = '\0') Alors j ←  1 FinSi
13           Sinon  {espace, lettre accentuée, ponctuation…}
14               result[i] ←  texte[i]
15           FinSi
16       FinSi
17       i ← i + 1
18   FinTantQue
19   result[i] ←  '\0'
```

Fin chiffrer

c. Supposons que l'on veuille chiffrer, avec une clé de longueur k, une chaîne constituée exclusivement de L minuscules, avec L > k. Comptez le nombre d'opérations de chaque type (affectations d'entiers, comparaison de caractères, etc.) pour évaluer la complexité en temps de cet algorithme de chiffrement. Qui de L ou de k influence le plus le temps d'exécution ? Ce temps sera-t-il plus long ou plus court avec une plus longue clé ?

Remarques :

- Les accès tableau/chaîne sont négligés.
- Le test de la ligne 3 est effectué L+1 fois (il est fait pour le '\0', ce qui provoque la sortie du Tant que). Le corps du Tant que est, lui, exécuté L fois.
- Le test de la ligne 7 est vrai chaque fois que l'on épuise la clé. En supposant que les caractères de la chaîne à chiffrer sont numérotés à partir de 1, le test sera vrai pour i=k, i=2k, i=3k, etc. Pour une chaîne à chiffrer de longueur L, le test sera donc vrai L/k fois. Ex : pour le texte ''abcd'' et la clé ''xy'', c'est après avoir crypté le 'b' et le 'd' (ligne 5) que j passe à 3 (ligne 6) et que le test rend vrai.
- Les lignes 8 à 16 ne sont pas exécutées du tout (cas où la chaîne est constituée exclusivement de minuscules) donc elles n'apparraissent pas dans le tableau.

Lignes	Aff. entier	Add. entiers	Aff. car.	Comp. Car.	Mult. car.	Add. car.	Soustr. car.	Car. mod car.
1-2	2							
3				L+1				
4				2L				
5			L		L	2L	L	L
6	L	L						
7	L/k			L				
17	L	L						
19			1					
Total	2L + 2 + L/k	2L	L+1	4L+1	L	2L	L	L

Notons t_1 le temps d'exécution d'une affection d'entiers, t_2 le temps d'exécution d'une addition d'entiers, etc. Ce sont des constantes, dans le sens où elles dépendent de la machine mais pas de L ni de k. Alors ce temps d'exécution total T(L,K) est :

$T(L,k) = 2t_1L + 2t_1 + t_1L/k + 2t_2L + t_3L + t_3 + 4t_4L + t_4 + t_5L + 2t_6L + t_7L + t_8L$

$T(L,k) = L(2t_1 + t_1/k + 2t_2 + t_3 + 4t_4 + t_5 + 2t_6 + t_7 + t_8) + 2t_1 + t_3 + t_4$

T(L,k) est $O(L)$.

On a donc un algorithme dont le temps d'exécution augmente linéairement avec la taille du texte à chiffrer. La taille k de la clé influence un peu aussi le temps d'exécution (le temps d'exécution sera légèrement plus court si la clé est longue, en effet il faudra revenir au premier caractère de la clé moins souvent).

d. Quelle serait l'entête de la procédure en C++ ?

```
void chiffrer(const char * texte, const char * cle, char * result);
```

Remarque : Lors de l'appel, il faudra veiller à ce que l'espace nécessaire pour stocker tout le résultat soit alloué. L'adresse dans `result` est soit une adresse sur la pile soit sur le tas, et le tableau de caractères alloué est suffisamment grand.

e. Donner un invariant de la boucle introduite dans la procédure chiffrer.

Un invariant de la boucle de la ligne 3 est : « Au début de toute itération, les éléments du tableau de caractères result entre les indices 1 et i-1 sont les éléments chiffrés par la clé issus des éléments de mêmes indices dans le tableau texte ».

Remarque : attention les indices sont donnés en notation algorithmique (commencent à 1).