TD 1 – Chaînes de caractères

Exercices sur les chaînes

1) Le classique des classiques : recherche de palindromes

Un palindrome est un mot ou une phrase se lisant de la même façon à l'endroit et à l'envers. Par exemple, ROTOR, ou RESSASSER sont des mots palindrome. Les phrases

"Engage le jeu que je le gagne"

"Tu l'as trop écrasé, César, ce Port-Salut !"

sont des phrases palindromes (la dernière est aussi un alexandrin, attribué à Victor Hugo).

Écrire une fonction `palindrome` prenant en argument une chaîne de caractères, et renvoyant `True` ou `False` selon que la chaîne est un palindrome ou pas.

On ne tiendra pas compte des accents, pour éviter de se torturer avec les conventions de FreePascal. Il pourra par contre être intéressant de réfléchir au moyen de ne pas tenir compte des virgules et des espaces pour valider les derniers exemples.

2) Conversion minuscules-majuscules

Écrire les fonction `MinVersMaj` et `MajVersMin` prenant en argument une chaîne, et renvoyant un chaîne représentant le même texte, dont toutes les lettres sont converties en les majuscules (respectivement en minuscules).

Par exemple, si l'on passe la chaîne `'Le petit dejeuner est servi'` à la fonction `MinVersMaj`, celle-ci devra renvoyer `'LE PETIT DEJEUNER EST SERVI'`.

On utilisera uniquement les fonction `chr` et `ord`, ainsi qu'une bonne table ASCII !

3) Compter les lettres, les mots

a) Écrire des fonctions `voyelles` et `consonnes`, prenant en argument une chaîne de caractères et renvoyant respectivement le nombre de voyelles et le nombre de consonnes présentes dans cette chaîne.

b) Écrire une fonction prenant en argument une chaîne de caractères, et renvoyant le nombre de mots

4) Conversions diverses

a) Écrire une procédure `IntToString` prenant en argument un entier `x` et une chaîne `s`, et écrivant dans `s` la chaîne de chiffres représentant l'entier `x`. Par exemple, si `x=17`, alors `s` devra contenir la chaîne `'17'`.

b) Écrire la fonction inverse, prenant en argument une chaîne composée de chiffres, et calculant l'entier qu'elle représente. On pourra réfléchir au cas des entiers relatifs, et au cas plus compliqué des nombres en virgule flottante.

c) Une chaîne représente un entier écrit en base 16, commençant par `0x` ou `0X`. Par exemple, la chaîne `'0xF1A'` représente le nombre d'écriture hexadécimale F1A, et d'écriture décimale 3866.

Écrire la fonction `HexToInt` convertissant une telle chaîne en l'entier qu'elle représente, et la fonction inverse `IntToHex` associant à un entier la chaîne qui le représente en hexadécimal.

d) Un polynôme est représenté par un tableau d'entiers de type

```
polynome = array[0..N] of integer
```

Écrire les fonctions transformant un tel tableau en une chaîne de caractères, ou une chaîne de caractère en le tableau des coefficient du polynôme la représentant.

Par exemple, le tableau `[3,-1,5,0,4,0]` devrait représenter le polynôme $4X^4+5X^2-X+3$.

5) Recherche de numéros et d'adresses

Une page web est découpée en chaînes de caractères. On recherche des numéros de téléphone et des adresses email dans ces chaînes.

Écrire deux fonctions `Extraire_Biniou` et `Extraire_Email` prenant en argument une chaîne de caractères, et remplissant un tableau avec les numéros de téléphone et les adresses email qui s'y trouvent.

On partira du principe qu'un numéro de téléphone est de la forme `nn.nn.nn.nn.nn`, où chaque `n` est un chiffre entre `0` et `9`, et qu'une adresse email est une chaîne de la forme `s1@s2`, où `s1` et `s2` sont des sous-chaînes composées uniquement de lettres, de chiffres et de points (`.`), ne commençant ni ne finissant par un point, et ne contenant pas deux points consécutifs.

6) Cryptologie

a) La légende raconte que Jules César transmettait ses messages en décalant chaque lettre de 3 positions : A devient D, B devient E,... W devient Z, X devient A, Y devient B et Z devient C. Les chiffres sont décalés de la même façon, les autres caractères (espaces, virgules, points...) ne sont pas modifiés.

Écrire deux fonctions `Chiffre_Cesar` et `Dechiffre_Cesar` réalisant le chiffrement et le déchiffrement d'un message représenté par une chaîne.

b) Sur Usenet, il est courant, lorsqu'on veut qu'une partie d'un message ne soit pas directement lisible (pour ne pas gâcher une surprise, par exemple), de coder ce message en ROT13. Cela consiste, comme pour le code de César, à décaler les lettres (et uniquement les lettres) de 13 positions. L'avantage est alors que les fonctions de codage et de décodage sont identiques.

Écrire cette fonction `rot13`.

c) Le code de César et ses dérivés sont très simples à craquer. Il suffit de trouver la valeur du décalage pour décoder le message.

L'idée du code de Vigénére est d'utiliser une *clé* de codage qui encode le décalage de chaque lettre. Par exemple, voici comment on code le message `'RENDEZ VOUS DEMAIN'` avec la clé `'BTSSIO'`. Chaque lettre de la clé indique le décalage de la lettre correspondante : A décale de 1, B de 2...

Message	R	E	N	D	E	Z		V	O	U	S		D	E	M	A	I	N
Clé	B	T	S	S	I	O	B	T	S	S	I	O	B	T	S	S	I	O
Code	T	Y	G	W	N	O		P	H	N	B		F	Y	F	T	R	C

Écrire les fonctions `Codage_Vig` et `Decodage_Vig` de codage et de décodage, prenant en argument deux chaînes, le message et la clé, et renvoyant le message transformé.

7) Comparaison de chaînes de caractères

Les systèmes d'exploitation affichent en général les fichiers dans l'ordre *lexicographique*, en utilisant le code ASCII[1]. Ainsi, le fichier `Zigomatique.txt` apparaîtra *avant* le fichier `abricot.jpg`, parce que la lettre 'Z' a un code ASCII inférieur à la lettre 'a'.

Écrire une fonction comparant deux chaînes de caractères `s1` et `s2`, et renvoyant -1, 0 ou $+1$ selon que `s1<s2`, `s1=s2` ou bien `s1>s2` dans l'ordre lexicographique.

Des exercices plus compliqués

1) Recherche d'une sous-chaîne

On dit qu'une chaîne `m` (de longueur `lm`) est une sous-chaîne de la chaîne `s` si il existe un indice entier `k` tel que `m[i] = s[i+k-1]` pour `i` compris entre 1 et `lm`. On dit alors que `m` apparaît dans `s` avec le décalage `k`.

a) Écrire une fonction `sous_chaine` prenant en argument deux chaînes `m` et `s`, et renvoyant -1 si `m` n'apparaît pas dans `s`, et le plus petit entier `k` tel que `m` apparaît dans `s` avec le décalage `k` sinon.

b) Combien de comparaisons fait-on au pire pour vérifier qu'une chaîne longueur `lm` est ou n'est pas une sous-chaîne d'une chaîne de longueur `ls` ?

[1] C'est d'ailleurs pour cela qu'il faut dater ses fichiers "à l'américaine", sous la forme aaaammjj, où 'aaaa' est l'année, 'mm' le mois et 'jj' le jour, pour que les fichiers soient triés dans l'ordre chronologique !

c) Chercher sur Internet des mécanismes plus sophistiqué de recherche. On pourra partir du site `http://www-igm.univ-mlv.fr/~lecroq/string/index.html`.

d) Un problème classique est la prise en considération de caractères spéciaux (des "wildcards" en anglais), pouvant remplacer n'importe quelle lettre, voire n'importe quelle chaîne.

Par exemple, la commande `ls *.tex` dans une console UNIX va lister tous les fichiers du répertoire courant dont le nom se termine par `.tex`, le caractère spécial `*` remplaçant n'importe quelle combinaison de lettres (y compris la chaîne vide).

Sans entre dans des complications faisant intervenir des automates finis, chercher comment on peut prendre en compte le caractère spécial `?`, qui représente *exactement uen lettre quelconque*.

2) Un problème tiré des "Programming Challenges"

On se donne deux chaînes `a` et `b`. Il s'agit de trouver la plus longue chaîne `x` dont une permutation est une sous-séquence de `a`, et une autre permutation une sous-séquence de `b`.

Par exemple, si `a="Babar"` et `b="Arbalete"`, alors `x="aabr"` est la réponse attendue.

Imaginer un algorithme permettant d'effectuer cette recherche, et l'implémenter sous forme d'une fonction.

3) Un autre

Parfois, lorsqu'on est dans le noir, ou mal installé devant un clavier, on tape "de travers". Les lettres sont décalées d'un cran vers la droite, le A devient Z, le D devient F, le M devient ù, etc.

Écrire un programme qui corrige ce problème en associant à une chaîne "décalée" la chaîne correcte qui lui correspond.

4) Et encore un !

Étant donnée une grille de `m` lignes et `n` colonnes composée de lettre et un mot, on cherche la position de ce mot dans la grille.

On ne tient pas compte de la casse des lettres (minuscules ou majuscules), les mots peuvent être écrits dans n'importe laquelle des huit directions. On supposera que $1 \leqslant m, n \leqslant 50$.

Ainsi, par exemple, dans l'exemple ci-dessous :

```
abcDEFGhigg
hEbkWalDork
FtyAwaldOrm
FtsimrLqsrc
byoArBeDeyv
Klcbqwikomk
strEBGadhrb
yUiqlxcnBjf
```

le mot `Bambi` apparaît-il à la deuxième ligne, troisième colonne, et le mot `Waldorf` à la deuxième ligne, cinquième colonne.

Écrire une fonction prenant en argument un tableau de caractères et un mot, et cherchant la position de la première lettre de ce mot dans la grille. S'il y a plusieurs occurences du mot dans la grille, on ne donnera que les coordonnées de celle se trouvant le à gauche dans la grille.

Sujet d'étude : la distance de Levenshtein

La distance de Levenshtein mesure le degré de similarité entre deux chaînes de caractères. Elle est égale au nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne à l'autre. C'est une distance au sens mathématique du terme, donc en particulier c'est un nombre positif ou nul, et deux chaînes sont identiques si et seulement si leur distance est nulle. On a aussi des propriétés de symétrie, et l'*inégalité triangulaire* de la géométrie est ici aussi vérifiée.

Par exemple, pour passer de la chaîne NICHE à la chaîne CHIENS, il faut :

- supprimer les lettres N et I,
- insérer les lettres I, N et S.

donc la distance de Levenshtein entre ces deux chaînes est au plus 5. On peut montrer[2] qu'elle est en fait *exactement* 5. Notons au passage que cette distance ne s'intéresse pas aux déplacements de caractères.

Une des applications de cette distance est la correction orthographique : lorsqu'une personne tape un mot, on le compare à un dictionnaire. Si le mot est présent, on ne fait rien, sinon, on cherche parmi les mots du dictionnaire ceux dont la distance de Levenshtein au mot tapé est inférieure à une limite donnée. Les mots les plus proches sont suggérés comme remplacement en premier.

La mesure de la distance de Levenshtein entre deux chaînes `Chaine1` et `Chaine2`, de longueurs respectives `n1` et `n2`, consiste en la mise en œuvre l'algorithme suivant :

```
fonction DistanceDeLevenshtein(Chaine1,Chaine2 : chaîne) : entier;

// N est un entier représentant la longueur
// de la plus grande chaîne manipulable

variables :
    n1,n2 : entiers      // tailles des deux chaînes
// Le tableau permettant de calculer la distance entre
// les deux chaînes. On en remplira que les n1+1 premières
// lignes, et les n2+1 premières colonnes.
    d     : tableau[0..N,0..N] d'entiers;
    i,j,c : entiers      // c est le coût d'une substitution

n1 <- longueur(Chaine1)
n2 <- longueur(Chaine2)

// initialisation du tableau
pour i de 0 à n1 faire d[i,0] <- i
pour j de 0 à n2 faire d[0,j] <- j

// remplissage du tableau d, ligne par ligne
pour i de 1 à n1 faire
    pour j de 1 à n2 faire
        si Chaine1[i] = Chaine2[i] alors c <- 0 sinon c <- 1
        d[i,j] <- minimum(
                          d[i-1,j  ] + 1  // effacement
                          d[i  ,j-1] + 1  // insertion
                          d[i-1,j-1] + c  // substitution
                         )
retourner d[n1,n2]
```

[2] Les remarques ne garantissent pas qu'on ne peut pas passer de NICHE à CHIENS en faisant moins de 5 opérations.

Pour illustrer le fonctionnement de cet algorithme, observons son déroulement avec `Chaine1='SAMEDI'` et `Chaine2='VENDREDI'`. On commence par initialiser le tableau :

		V	E	N	D	R	E	D	I
	0	1	2	3	4	5	6	7	8
S	1								
A	2								
M	3								
E	4								
D	5								
I	6								

Ensuite on le remplit ligne par ligne, de la gauche vers la droite. Voici deux étapes intermédiaires :

		V	E	N	D	R	E	D	I
	0	1	2	3	4	5	6	7	8
S	1	1	2	*3*	*4*	5	6	7	8
A	2	2	2	*3*	**4**				
M	3								
E	4								
D	5								
I	6								

		V	E	N	D	R	E	D	I
	0	1	2	3	4	5	6	7	8
S	1	1	2	3	4	5	6	7	8
A	2	2	2	3	4	5	6	7	8
M	3	*3*	*3*	3	4	5	6	7	8
E	4	*4*	**3**						
D	5								
I	6								

Dans le tableau de gauche, la case au dessus de celle calculée contient 4, la cas à gauche 3, le minimum est donc 3, auquel on ajoute 1. La case en diagonale contient 3, auquel il faut ajouter 1 parce que la lettre A de SAMEDI n'est pas la lettre D de VENDREDI. Le minimum des deux valeurs est 4, c'est donc ce qu'on inscrit dans cette case.

Dans le tableau de droite, le principe est le même, mais la lettre E étant commune aux deux mots, on n'ajoute rien à la case en diagonale, le minimum est donc 3.

À la fin du remplissage, on obtient le tableau suivant :

		V	E	N	D	R	E	D	I
	0	**1**	**2**	3	4	5	6	7	8
S	1	1	2	**3**	4	5	6	7	8
A	2	2	2	3	**4**	5	6	7	8
M	3	3	3	3	4	**5**	6	7	8
E	4	4	3	4	4	5	**5**	6	7
D	5	5	4	4	4	5	6	**5**	6
I	6	6	5	5	5	5	6	6	**5**

Le chemin en gras indique les opérations à exécuter pour passer du premier mot au second :

- un déplacement horizontal indique une insertion,
- un déplacement vertical indique une suppression,
- un déplacement en diagonal indique une substitution.

Ainsi, la séquence permettant de passer de `SAMEDI` à `VENDREDI` est :

```
SAMEDI -> VSAMEDI -> VESAMEDI -> VENAMEDI -> VENDMEDI -> VENDREDI
```

soit deux insertions et trois substitutions. D'autres solutions sont possibles : il suffit d'aller de case en case en partant de la case inférieure droite, et en allant vers une case réalisant le minimum calculé dans cette case. Dans l'exemple qui nous occupe, peu importe à quel moment on fait les insertions et les substitutions. On aurait ainsi pu faire :

```
SAMEDI -> VAMEDI -> VEMEDI -> VENEDI -> VENDEDI -> VENDREDI
```

Le but de ce sujet d'étude est d'étudier cet algorithme sur quelques exemples, et de réaliser quelques programmes le mettant en œuvre et illustrant son fonctionnement.

Étude de quelques exemples

Pour bien comprendre comment fonctionne cet algorithme, étudions des exemples dans lesquels une seule opération est à réaliser.

a) Construire la matrice issue du calcul de la distance entre `PAPA` et `PARA` (exemple d'une unique **substitution**).

b) Construire la matrice issue du calcul de la distance entre `FORT` et `FORET` (exemple d'une unique **insertion**).

c) Construire la matrice issue du calcul de la distance entre `LEON` et `LEO` (exemple d'une unique **suppression**).

d) Construire la matrice issue du calcul de la distance entre `NICHES` et `CHINE`.

Déduire de cette matrice tous les moyens de passer d'un mot à l'autre en effectuant à chaque étape une seule opération.

Mise en œuvre de l'algorithme

a) Écrire une fonction Pascal prenant en argument deux chaînes, et calculant la distance de Levenshtein entre elles.

Insérer cette fonction dans un programme permettant de la tester sur quelques chaînes saisies par l'utilisateur.

b) Améliorer la fonction précédente de manière à lui faire afficher la matrice calculée, ainsi qu'une suite de transformations possibles permettant de passer d'une chaîne à l'autre.

c) Améliorer encore la fonction précédente de manière à lui faire afficher *toutes* les suites de transformations possibles.

Un exemple de correction orthographique

On va étudier sur un exemple le mécanisme de suggestion de mots de substitutions lors d'une correction orthographique.

a) Écrire une fonction prenant en argument une chaîne `c`, un entier `k` et un tableau `t` de `N` chaînes, et renvoyant un tableau `r` de chaînes, ne contenant aucune chaîne si la chaîne `c` est une des chaînes de `t`, et contenant sinon *toutes les chaînes* de `t` dont la distance à `c` est inférieure à `k`.

b) Tester votre fonction à l'aide d'un programme demandant à l'utilisateur de remplir un dictionnaire (le tableau `t`), puis lui demandant une chaîne (la chaîne `c`) et une borne (l'entier `k`), et renvoyant toutes les suggestions issues du dictionnaire si le mot tapé n'est pas dans le dictionnaire.

On pourra ajouter une option permettant à l'utilisateur d'ajouter le mot tapé au dictionnaire !

Recherches complémentaires

Se documenter sur les sujets connexes aux notions étudiées dans ce sujet d'étude : distance entre deux mots, correction orthographique...

Sources :

- http://en.wikipedia.org/wiki/Levenshtein_distance
- http://jean-paul.davalan.pagesperso-orange.fr/lang/algo/lev/index.html
- http://rosettacode.org/wiki/Levenshtein_distance