POLYTECH LILLE

Université Lille1 Sciences et Technologies

Département IMA / 3A (S5) **Programmation Structurée 2010/2011**

Examen - 2h

Calculatrices et livres interdits, traducteurs, documents de cours et de TD autorisés

Les algorithmes seront écrits en pseudo-langage algorithmique. On veillera à bien **indenter** codes et pseudo codes, ainsi qu'à commenter généreusement. On fournit des indications de temps. On numérotera bien les questions. Le problème est long, et il est normal de ne pas tout traiter, mais essayez de ne pas grapiller. Il n'y a aucune subtilité avant la question 10 du problème (qui doit d'ailleurs vous rappeler des choses vues en SD). **Merci de traiter AUSSI les questions de complexité !**

A – Exercice : Schéma d'exécution (20min)

1. Écrire un algorithme qui prend en paramètre un caractère c et un entier x, et qui *modifie* le caractère(passage par adresse) selon le décalage induit par x. Exemple, si x vaut -32, et $c =' a'$ (code 97), alors c devient le caractère A (code ASCII 97-32). Attention au dépassement, un caractère ASCII est entre 0 et 255 (utiliser modulo).
2. Écrire le main faisant un appel à votre algorithme, sur l'exemple précédent, et imprimant le caractère résultat (pas son code ASCII). On pourra supposer l'existence d'une fonction `ImprimerChar(char c)`.
3. Mêmes questions en C. L'impression se fera avec `printf`.
4. Donner le schéma d'exécution de votre programme C.

B – Problème - Recherche d'un mot dans un texte (1h40)

d'après X-PC 2010

Les parties sont indépendantes. Dans toutes les questions, vous avez le droit de faire appel aux fonctions des questions précédentes, même si vous ne les avez pas écrites. Quelle que soit la forme de vos algorithmes (C ou pseudo-codes), ceux-ci seront **expliqués et commentés !**

Dans ce problème, on considère un texte contenu dans un tableau `texte`, de taille N **constante globale** (pour les exemples, $N = 13$). Le tableau est considéré rempli (toutes les cases de 0 à $N-1$ contiennent donc un caractère) par des caractères de 'a' à 'z'. L'objectif est de pouvoir rechercher un mot quelconque dans ce texte, c'est-à-dire pouvoir dire si ce mot apparaît dans le texte, à quel endroit, combien de fois.

Attention, aucune des fonctions de string.h ne pourront être utilisées !

i	0	1	2	3	4	5	6	7	8	9	10	11	12=N-1
$texte[i]$	q	u	e	l	b	o	n	b	o	n	b	o	n

On considère que les mots sont plus petits que le texte, les mots sont donc stockés comme d'habitude dans un tableau de taille N aussi, mais avec le caractère de fin \0 (qui ne sert pas dans toutes les questions cependant).

i	0	1	2	3	4	5	6	7	8	9	10	11	12 =N-1
$mot1[i]$	b	l	a	b	l	a	\ 0	—-	—-	—-	—-	—-	—-

Partie 1 - Méthode simple (50 min)

DÉFINITION 1 *Un mot* `mot` *(de taille m) apparaît dans un texte* `texte` *si il existe un indice i tel que le sous-tableau* `texte[i..i+m-1]` *est égal à* `mot` *(lettre par lettre !).*

DÉFINITION 2 *Le* suffixe *numéro k (k entre* 0 *et* $N-1$*) du tableau* `texte` *est le sous-tableau* `texte[k..N-1]`.

Un mot de taille m apparaît donc dans un texte à l'indice k si il apparaît au début du suffixe numéro k. On dit alors que k est un *indice d'apparition*.

Question 1 Quel est le suffixe numéro 3 du tableau `texte` contenant la chaîne "quelbonbonbon" (celui qui est dessiné) ?

Question 2 Quels sont les indices d'apparition du mot "bon" dans `texte` ?

Question 3 Écrire une **action C** `afficheTexte` qui étant donné un tableau `texte` de taille N affiche ce texte (utilisation obligatoire d'une boucle).

Question 4 Écrire une **fonction C** nommée `enTetedeSuffixe` qui étant donnés `texte`, `mot` (et sa taille `m`) et un indice k, retourne `true` si le mot apparaît à l'indice k, `false` sinon.

Question 5 Écrire une **fonction C** nommée `apparaitV1` qui étant donnés `texte`, `mot` (et sa taille `m`), retourne `true` si le mot apparaît dans le texte, `false` sinon.

Question 6 Écrire une **fonction C** qui compte le nombre d'occurrences d'un mot donné dans un texte, et qui au passage affiche les indices d'occurences.

Question 7 Écrire un programme **C complet** (main, fonctions -déclarations, et code avec des pointillés- , etc) :
- Inclusion des librairies `stdio.h` et `stdbool.h`.
- Déclaration de la constante N avec la taille adéquate.
- Déclaration du tableau `texte` et remplissage avec la chaîne "quelbonbonbon".
- Déclaration du tableau `mot` et remplissage avec la chaîne "bon".
- Impression de tous les indices d'occurence du sous-mot `bon` dans `texte`, ainsi que le nombre d'occurences.

Question 8 Combien de comparaisons cet algorithme de recherche fait-il au pour un mot de taille m recherché dans un texte de taille N ?

Partie 2 - Méthode avec tableau des suffixes (50 min)

Dans cette partie nous allons construire un tableau des suffixes, puis l'utiliser pour rechercher les fréquences d'apparition de manière plus efficace.

2.1 Construction du tableau des suffixes (20 min)

Le tableau des suffixes est construit de la façon suivante : tous les suffixes sont caractérisés par leur indice d'apparition dans le tableau `texte`. Le tableau des suffixes est obtenu en triant les suffixes *par ordre alphabétique* (l'ordre du dictionnaire !). La figure suivante montre la construction du tableau des suffixes associé au texte "quelbonbonbon" :

i	suffixe numéro i
0	quelbonbonbon
1	uelbonbonbon
2	elbonbonbon
3	lbonbonbon
4	bonbonbon
5	onbonbon
6	nbonbon
7	bonbon
8	onbon
9	nbon
10	bon
11	on
12	n

i	numero	suffixes dans l'ordre
0	10	bon
1	7	bonbon
3	4	bonbonbon
3	2	elbonbonbon
4	3	lbonbonbon
5	12	n
6	9	nbon
7	6	nbonbon
8	11	on
9	8	onbon
10	5	onbonbon
11	0	quelbonbonbon
12	1	uelbonbonbon

La seconde colonne du tableau de droite donne le tableau des suffixes (qui est donc un tableau **d'entiers**).

i	0	1	2	3	4	5	6	7	8	9	10	11	12 =N-1
$suffixetexte[i]$	10	7	4	2	3	12	9	6	11	8	5	0	1

On remarque qu'au sens de l'ordre alphabétique, le mot "n" est plus petit que le mot "nbon", le mot "onbonbon" est plus petit que le mot "quelbonbonbon".

Nous allons construire ce tableau à l'aide d'une méthode simple (il en existe de plus efficaces).

Question 9 en pseudo-code Écrire une fonction `comparerSuffixes(texte,k1,k2)` qui retourne un entier r :
- $r = 0$ si le suffixe numéro k_1 est égal au suffixe numéro k_2.
- $r = -1$ si le suffixe numéro k_1 du texte est strictement inférieur (au sens de l'ordre alphabétique) au suffixe numéro k_2.
- $r = 1$ sinon.

Question 10 en pseudo-code. En utilisant cette fonction de comparaison, écrire un algorithme qui étant donné `texte`, construit le tableau des suffixes correspondant.
On notera que cette fonction effectue essentiellement un tri, où la fonction de comparaison classique est remplacée par `comparerSuffixes` (on peut supposer l'existence d'une fonction `permuter`). Montrer les premiers pas d'exécution de votre algorithme sur l'exemple.

Question 11 Combien de comparaisons effectue votre algorithme ?

2.2 Utilisation pour la recherche (30 min)

Le mot `mot` apparaît dans le texte `texte` ssi il est en tête d'un suffixe de `texte`. Pour chercher un mot à l'aide du tableau des suffixes nous allons chercher si il existe un indice k tel que `mot` soit au début de `texte[tabsuffixe[k]]`. Le fait que le tableau des suffixes soit trié va nous aider.

Nous allons tout d'abord écrire la recherche d'un entier donné dans un tableau trié, en utilisant la $r$echerche dichotomique. Soit un tableau d'entiers triés, par exemple (ici $N = 6$).

i	0	1	2	3	4	5
$ex[i]$	23	30	34	42	1515	2020

Recherchons 30. On commence par regarder au milieu du tableau, à l'indice $(5+0)/2 = 2$, c'est l'élément 34. Comme $34 > 30$, on sait que si 30 est dans le tableau, il est à un indice entre 0 et 1. On peut donc relancer la recherche dans le sous-tableau entre les indices 0 et 1....

Question 12 en pseudo-code Écrire cet algorithme (en suivant une structure similaire à celle du trifusion vu en cours).

Maintenant, pour appliquer la recherche dichotomique d'un mot dans un texte à l'aide de son tableau des suffixes, on a besoin d'une nouvelle fonction de comparaison entre mot et suffixe.

Question 13 en pseudo-code Écrire une fonction `comparerMotSuffixe(mot, texte, k)` qui prend en arguments un mot `mot`, un texte `texte`, un indice k désignant un suffixe du texte, et qui renvoie un entier r :
- $r = 0$ si `mot` apparaît en tête du suffixe numéro k du texte ;
- $r = -1$ si `mot` est inférieur (au sens lexicographique) du suffixe numéro k ;
- $r = 1$ sinon.

Attention, le mot a un marqueur de fin '\0' !

Question 14 en pseudo-code En utilisant cette fonction de comparaison, et en adaptant l'algorithme de recherche dichotomique écrit auparavant, écrire une fonction de recherche d'un mot dans un texte.

Question 15 Combien de comparaisons cet algorithme fait-il au pire ? Dans quels cas l'utiliser ?