Examen
Programmation Structurée IMA3
9 décembre 2011
Durée 2H

Calculatrices, téléphones et livres interdits
Traducteurs, polycopié de cours et notes de TD et TP autorisés

Instructions **à lire attentivement !**

1. Les questions précisent si les algorithmes sont à écrire en pseudo-langage algorithmique ou en C.
2. Les codes écrits seront expliqués (au besoin avec des exemples), *commentés*, indentés correctement (en C comme en pseudo-code).
3. Les réponses aux questions seront numérotées, et on évitera le grapillage.
4. Des indications de temps sont fournies. Ne pas passer plus d'une demi-heure sur l'exercice.

Pour le problème (partie B) :

1. Les consignes précédentes s'appliquent encore !
2. Les deux parties se suivent.
3. Dans toutes les questions, vous avez le droit de faire appel aux fonctions des questions précédentes, même si vous ne les avez pas écrites.
4. **Merci de traiter AUSSI les questions de complexité !**
5. **Le problème est long**, et il est normal de ne pas tout traiter (la notation en tiendra compte), un objectif raisonnable étant d'aller jusqu'à la question 11. Il n'y a aucune subtilité avant la question 12.

A – Exercice : Programme Mystère (30min)

adapté d'un examen de Programmation java (INF311), Polytechnique, 2008

Voici un programme C (à lire colonne par colonne) :

```c
#include <stdio.h>
#define N 3

void display (int t[N]){
  int i;
  for(i=0;i<N;i=i+1){
    printf("%d ",t[i]);
  }
  printf("\n");
}

void swap3 (int t[N],int i,int j){
  int tmp = t[i];
  t[i]=t[j];
  t[j]=tmp;
}

void init(int tt[N]){
  int i;
  for (i=0;i<N;i=i+1){
    tt[i] = i+1;
  }
}
```

```c
void mysteriousRec(int tab[N],int k) {
  int j;
  if ( k == N-1 ) display (tab);
  for (j = k ; j < N ; j ++) {
    swap3 (tab,k,j);
    mysteriousRec(tab,k+1);
    swap3 (tab,k,j);
  }
}

int main(){
  int montab[N];
  init(montab);

  mysteriousRec(montab,0);
  return 0;
}
```

Question 1

Après avoir compilé, quelles impressions sont effectuées par ce programme sur le terminal ? *quelques lignes*

Question 2

Pourquoi `mysteriousRec(t,0)` termine-t-il ?*quelques lignes*

Question 3

Que fait ce programme dans le cas général ? (N quelconque)*10 à 20 lignes. La justification est difficile, mais sera bien payée.*

B – Problème (BZIP) - 1h30

d'après Concours d'entrée Polytechnique 2007, MP/PC, et TP de OCaml de MP 2007-2008 par K. Huguenin et C. Avenel.

Dans ce problème, nous allons coder une partie de l'algorithme de compression `bzip`, en fait la partie préparatoire du texte, qui consiste à faire apparaître des caractères identiques dans le texte à compresser. La partie compression des données n'est pas abordée.

1 - Transformation de Burrows-Wheeler - 1h10

Cette transformation réalise une permutation des lettres du mot d'entrée (c'est-à-dire que toutes les lettres du mot d'entrée s'y retrouveront). Cette transformation a deux caractéristiques :

- dans le mot de sortie les répétitions de lettres identiques sont plus fréquentes.
- elle est bijective, c'est-à-dire qu'il existe une transformation inverse qui calcule à partir du mot transformé, le mot initial.

L'objet de cette partie est de calculer le codage d'un mot quelconque par la transformée.

Soit w un mot quelconque de taille ℓ, par exemple `banane`, qui est de taille 6. Dans un premier temps, on ajoute un marqueur de fin de mot, $\sharp$ (on considère que $\sharp$ n'est pas un caractère du mot). Ensuite, on construit une matrice qui contient toutes les permutations **circulaires** de ce texte, en décalant successivement le mot d'entrée vers la droite (matrice de gauche du dessin1).

$$\begin{pmatrix} b & a & n & a & n & e & \sharp \\ \sharp & b & a & n & a & n & e \\ e & \sharp & b & a & n & a & n \\ n & e & \sharp & b & a & n & a \\ a & n & e & \sharp & b & a & n \\ n & a & n & e & \sharp & b & a \\ a & n & a & n & e & \sharp & b \end{pmatrix} \qquad \begin{pmatrix} \sharp & b & a & n & a & n & \mathbf{e} \\ a & n & a & n & e & \sharp & \mathbf{b} \\ a & n & e & \sharp & b & a & \mathbf{n} \\ b & a & n & a & n & e & \mathbf{\sharp} \\ e & \sharp & b & a & n & a & \mathbf{n} \\ n & a & n & e & \sharp & b & \mathbf{a} \\ n & e & \sharp & b & a & n & \mathbf{a} \end{pmatrix}$$

FIGURE 1 – Matrice des permutations circulaires, puis la même, triée, et en gras le mot résultat.

Ensuite, les **lignes** de la matrice ainsi obtenue sont triées par ordre alphabétique (ordre du dictionnaire), en considérant les lignes comme des mots, $\sharp$ étant inférieur à tous les autres caractères. Notre matrice devient alors la matrice de droite de la figure 1. (En effet, la ligne 1 : `anane#b` est inférieure à la ligne 2 : `ane#ban` car les deux premiers caractères sont identiques et le caractère 'a' est inférieur au caractère 'e'.)
Enfin, le mot résultat de la transformation est la dernière colonne de la matrice, ici en **gras**, c'est-à-dire `ebn#naa`.

Tous les tableaux de caractère (chaînes) auront une taille fixée à N (constante), N considéré toujours plus grand que la taille du mot à coder. Les tableaux de chaînes de caractères seront des matrices de caractères de taille $N \times N$. On rappelle qu'une chaîne de caractère possède un marqueur de fin, le caractère \0.

Question 1

Soit w le mot suivant (pour cette question on suppose N=9) :

h	e	l	l	o	\ 0			

Quelle est la taille du mot w ?

Question 2

Calculer la transformée du mot `ima`.*On demande les calculs et le résultat, pas un algorithme*

Question 3

Écrire en C une fonction `taille` qui prend en entrée une chaîne de caractères et qui renvoie sa taille.

Question 4

Écrire en C une action `copie_tab` qui prend en paramètre deux chaînes de caractères, un entier ℓ, et qui copie la première chaîne dans la seconde sachant que la taille du mot est ℓ. *On n'oubliera pas de mettre le caractère de fin de chaîne à la bonne place.*

Question 5

Écrire en C une action `decalage_droite` qui prend en entrée deux chaînes de caractères, un entier ℓ (la taille de la première chaîne), et qui modifie la deuxième chaîne de façon à ce qu'elle contienne la première chaîne décalée vers la droite. *Sur notre exemple, le décalé de* `banane#` *est* `#banane`*, le décalé de* `#banane` *est* `e#banan`.

Question 6

Écrire en C une action `imprime_mat_carree` qui étant donnés une matrice *mat* de caractères de taille $N \times N$, et un entier ℓ imprime les caractères $mat[i][j]$ avec $0 \leq i \leq \ell - 1$ et $0 \leq j \leq \ell - 1$. *Cette fonction nous sera bien utile, car on va encoder des mots de taille quelconque dans des lignes de taille N, et on ne veut pas imprimer des caractères en trop.*

Question 7

Écrire en C une action `mat_permut` qui étant donnés un mot d'entrée t, une matrice *mat* de caractères de taille $N \times N$, et un entier ℓ (taille du mot d'entrée), remplit la matrice *mat* avec les permutations du mot t. *On pourra s'apercevoir que $mat[i]$ est un tableau de taille N, et il est fortement recommandé d'utiliser les actions/fonctions précédentes.*

Question 8

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 valant 60
- Déclaration et initialisation d'un tableau t contenant le mot `banane#`
- Déclaration, construction et impression de la matrice de permutation de t.

Question 9

Quelle est la complexité en nombre d'affectations de votre algorithme de construction de la matrice de permutations, en fonction de ℓ taille du mot initial ?

Maintenant, nous allons trier la matrice de permutation (en place, sans utilisation d'une matrice auxiliaire), par lignes, selon l'ordre alphabétique, toujours en considérant que *les lignes sont des mots.*

Question 10

En pseudo-code, écrire une fonction `comp_tab_alpha` qui prend en argument deux chaînes de caractères de taille ℓ, ainsi que l'entier ℓ et qui retourne :
- $r = 0$ si les deux chaînes sont identiques
- $r = -1$ si la première chaîne est strictement inférieure (au sens de l'ordre alphabétique) à la deuxième chaîne.
- $r = 1$ sinon.

Question 11

En pseudo-code, écrire une action `echange_lignes` qui prend en paramètres une matrice *mat* de caractères de taille $N \times N$, deux entiers $i1$ et $i2$, un entier ℓ, et qui échange le contenu des lignes $i1$ et $i2$ de la matrice *mat* (en considérant que seules les cases d'indice 1 à $\ell - 1$ doivent être copiées dans une ligne). *On utilisera la fonction* `copie_tab` *de la question 4.*

Question 12

En utilisant les deux fonctions précédentes, écrire un algorithme **en pseudo-code** qui étant donné une matrice *mat* de caractères de taille $N \times N$, ainsi que l'entier ℓ donnant la taille réelle de la sous-matrice à trier, trie la matrice en lignes par ordre alphabétique. *On modifiera un tri connu, par exemple le tri sélection, pour l'adapter au cas matriciel.*

Question 13

Quelle est la complexité de votre algorithme précédent en nombre d'affectations ?

Question 14

À l'aide de toutes les fonctions/actions écrites précédemment, écrire un algorithme **en pseudo-code** qui réalise le codage d'un mot initial quelconque (initialement sans marqueur de fin ♯). Quelle est la complexité de cet algorithme en nombre d'affectations ?

2 - Transformation de Burrows-Wheeler inverse - 20 min

L'objet de cette partie est de calculer la transformée inverse. L'algorithme consiste à construire une matrice à l'aide du mot codé (voir la figure 2 pour les étapes) :
- On trie les lettres du mot codé (ici donc, `ebn#naa`) par ordre alphabétique et on le stocke dans la colonne 0 de la matrice à construire.
- On ajoute à gauche de la matrice (en décalant la colonne déjà construite) le mot codé (étape (a) sur la figure). On trie par ordre alphabétique les *lignes de* la matrice obtenue. (étape (t))
- On recommence jusqu'à remplissage complet de la matrice. Le résultat (le mot décodé) est lisible sur la première ligne de la matrice.

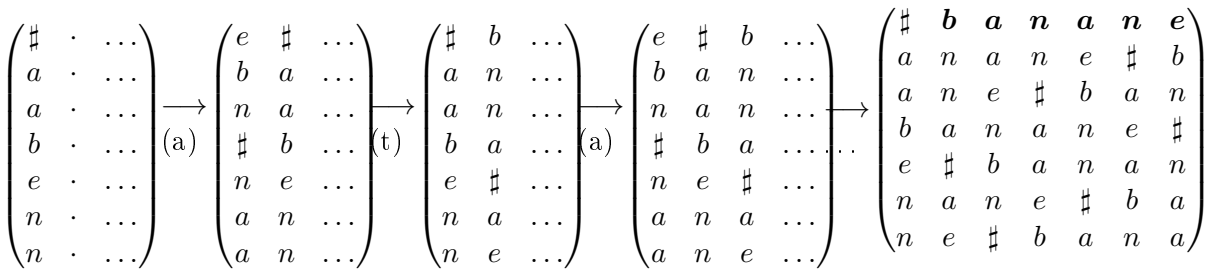

FIGURE 2 – Les différentes étapes du décodage

Question 15

Écrire **en pseudo-code** un algorithme qui réalise l'ajout à gauche d'un mot t de taille ℓ dans une matrice m dont les j premières colonnes sont remplies.

Question 16

Écrire **en pseudo-code** un algorithme qui calcule la transformée inverse d'un mot quelconque. *Bien détailler l'analyse ...*

Question 17

Quelle est la complexité de votre algorithme précédent ?

Remarques :

- Une correction de cet examen sera publiée sur le Twiki/Annales juste après la session.
- L'étudiant curieux pourra se reporter à la page Wikipédia expliquant Bzip et aux références citées au début du problème, pour comprendre l'étape de compression "par redondance" qui suit l'étape de préparation du code.