Licence L3 — Mathématiques et Applications (MAP)
Université Paris–Sud — Maths312: Algorithmique et programmation en C
Centre d'Orsay — Année 2009 − 2010

Examen
Mercredi 16 Décembre 2009 - Durée : 3 heures

Seules les notes de cours sur support papier sont autorisées.

Exercice - 1 *Langage C*

Soit une structure définissant une liste chaînée ainsi que certaines fonctions pour sa manipulation (voir en annexe). Deux individus proposent chacun une implémentation suivante de la libération de mémoire et teste leur implémentation dans la fonction principale en vérifiant la fuite de mémoire à l'aide des fonctions **`mtrace()`** et **`muntrace()`** de la bibliothèque standard accessibles via le fichier entête **`< mcheck.h >`**.

```
1  /*-------------------------------------*/
2  void LibereC(TypeCell* L){
3  TypeCell* p;
4  while(L != NULL){
5    p = L;
6    L = L->suivant;
7    free(p); p = NULL;
8  }
9  }
10 /*-------------------------------------*/
11 int main(){
12  mtrace();
13  TypeCell* L;
14  L = Creation(1);
15  InsereEnTete(&L,2); InsereEnTete(&L,3);
16  Affiche(L); LibereC(L); Affiche(L);
17  muntrace();
18  return 0;
19 }
```

```
1  /*-------------------------------------*/
2  void LibereP(TypeCell** pL){
3  TypeCell* p;
4  while((*pL) != NULL){
5    p = *pL;
6   *pL = (*pL)->suivant;
7    free(p); p = NULL;
8  }
9  }
10 /*-------------------------------------*/
11 int main(){
12  mtrace();
13  TypeCell* L;
14  L = Creation(1);
15  InsereEnTete(&L,2); InsereEnTete(&L,3);
16  Affiche(L); LibereP(&L); Affiche(L);
17  muntrace();
18  return 0;
19 }
```

Après compilation et exécution ils obtiennent

```
1  < 3      2          1          >
2  < 0      164098960  164098944  >
3
4  No memory leaks.
```

```
1  < 3      2          1          >
2  < >
3
4  No memory leaks.
```

La ligne 4 indique que dans chacun des cas la mémoire allouée dynamiquement a été convenablement libérée.

Q-1 **:** Expliquer le résultat obtenu. Puis faire un commentaire critique de l'une des approches.

Exercice - 2 *Programmation Dynamique*

Soit la suite de Fibonacci définie par : $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2} \quad \forall n \geq 2$.

On propose la fonction suivante pour son implémentation en langage C.

```
1  int Fibo(int n){
2    if(0 == n)
3      return 0;
4    if(1 == n)
5      return 1;
6    return Fibo(n - 1) + Fibo(n - 2);
7  }
```

Q-1 **:** Montrer que cette implémentation conduit à une complexité au moins exponentielle.

Q-2 **:** Montrer que cet algorithme peut rentrer dans le cadre de la programmation dynamique. Expliquer les transformations à faire.

Q-3 **:** Montrer que l'algorithme transformé est en $\Theta(n)$.

Exercice - 3 *Algorithme "glouton" et "diviser pour régner" dans les tris*

On se propose dans cet exercice de trier un ensemble fini de réels. Le cas illustratif sera le tableau suivant

$$T = \{6, 3, 7, 2, 3, 5\}.$$

On considère deux scénarios :
Scénario 1 : Chaque étape se déroule en deux phases :

1. *Extraction* : on extrait un élément de la partie non triée qui initialement est l'ensemble tout entier.
2. *Insertion* : on place l'élément prélevé à sa position dans la partie triée. On s'arrête lorsque la partie non triée est vide.

Scénario 2 : Chaque étape se déroule en deux phases :

1. *Partition* : On partitionne l'ensemble en deux parties
2. *Fusion* : On fusionne les deux parties triées selon le même scénario.

Q-1 **: Cas du scénario 1**

Q-1-1 : Ce scénario peut-il conduire à un tri de l'ensemble en question ? Est-il de type "glouton" ou du type "diviser pour régner" ?

Q-1-2 : Citer un algorithme qui réalise l'extraction avec une complexité en $\Theta(1)$ et une insertion avec une complexité en $\Theta(n)$, où n est la taille de la partie triée. Évaluer cet algorithme sur l'ensemble T. Cet algorithme est-il stable ? Est-il sur place ?

Q-1-3 : Citer un algorithme qui réalise l'extraction avec une complexité en $\Theta(n)$ et une insertion avec une complexité en $\Theta(1)$, où n est la taille de la partie non triée. Évaluer cet algorithme sur l'ensemble T. Cet algorithme est-il stable ? Est-il sur place ?

Q-1-4 : Montrer qu'un tel algorithme a un complexité temporelle quadratique, c'est-à-dire en $\Theta(n^2)$.

Q-2 **: Cas du scénario 2**

Q-2-1 : Ce scénario peut-il conduire à un tri de l'ensemble en question ? Est-il de type "glouton" ou du type "diviser pour régner" ?

Q-2-2 : Citer un algorithme qui réalise la partition avec une complexité en $\Theta(1)$ et une fusion avec une complexité en $\Theta(n)$, où n est la taille de l'ensemble. Évaluer cet algorithme sur l'ensemble T. Cet algorithme est-il stable ? Est-il sur place ?

Q-2-3 : Citer un algorithme qui réalise la partition avec une complexité en $\Theta(n)$ et une fusion avec une complexité en $\Theta(1)$, où n est la taille de l'ensemble. Évaluer cet algorithme sur l'ensemble T. Cet algorithme est-il stable ? Est-il sur place ?

Q-2-4 : Montrer qu'un tel algorithme a un complexité temporelle en $\Theta(n \log_2 n)$.

Q-3 **: Un tris par tas**

Q-3-1 : Appliquer le tris par tas sur l'ensemble T.

Q-3-2 : Ce tri rentre-t-il dans le cadre de l'un des scénarios précédents ?
(*Une justification rigoureuse est nécessaire, notamment les complexités de chaque phase d'une étape.*)

Exercice - 4 *Tri par distribution et sa programmation en C*

Q-1 **:**

Soit a un entier naturel. On sait que a admet une écriture binaire :

$$a = a_0 + 2a_1 + 4a_2 + \ldots + 2^{n-1}a_{n-1},$$

où $a_i \in \{0, 1\}$ et n est un entier suffisamment grand. Le nombre a_i est appelé $i-$**ème bits** de a ou aussi **bit** i de a.

Q-1-1 : Donner une expression simple calculant a_0 en fonction de a.

Q-1-2 : Quelle est l'écriture binaire de $\lfloor a/2 \rfloor$ en fonction de celle de a ($\lfloor \cdot \rfloor$ désigne la partie entière) ?

Q-1-3 : En déduire une fonction C **`int bit(int p, int a)`** qui retourne le bit p de a si p et a sont deux entiers. on donnera deux versions, une récursive et l'autre itérative.

Q-2 **:**

Soit $v = \{v_0, v_1, \ldots, v_{n-1}\}$ un vecteur à éléments entiers naturels. On veut réordonner ce vecteur de sorte que les v_i pairs soient placés en tête et tous les v_i impairs en queue, l'ordre relatif des éléments étant conservé à l'intérieur de chaque groupe. Par exemple si :

$$v = \{3, 1, 4, 1, 5, 9, 2, 6\}$$

alors on souhaite obtenir après transformations :

$$v = \{4, 2, 6, 3, 1, 1, 5, 9\}$$

Il est autorisé pour ce faire d'utiliser un vecteur auxiliaire w de longueur n.

Q-2-1 : Donner un algorithme en français effectuant la transformation demandée. On démontrera que cet algorithme répond effectivement au problème posé.

Q-2-2 : Écrire une fonction C **`void permute(int* v, int n)`** qui implémente cet algorithme. L'entier n est le nombre d'éléments du tableau v.

Q-2-3 : On appelle **transfert** une opération **`v[i] = qqch`** ou **`w[i] = qqch`** .
Calculer le nombre de transferts effectués par votre programme en fonction des nombres a et b d'entiers pairs et impairs dans v.

Q-2-4 : On généralise le problème en ajoutant à **`permute`** un paramètre p entier naturel de sorte que l'appel de **`permute(p,v,n)`** place en tête de v les éléments dont le bit p vaut 0 et en queue ceux dont le bit vaut 1, l'ordre initial des entiers étant conservé à l'intérieur de chaque groupe. Indiquer quels sont les modifications à apporter à votre code donné en **Q-2-2**.

Q-3 **:**

Soit $v = \{v_0, v_1, \ldots, v_{n-1}\}$ un vecteur à éléments entiers naturels que l'on veut trier par ordre croissant. On exécute l'algorithme suivant :

α) *Calculer* $M = \max(v_0, v_1, \ldots, v_{n-1})$.
β) *Déterminer un entier* K *tel que* $M < 2^K$.
γ) *Pour* $p = 0, 1, \ldots, K-1$ *faire* **`permute (p, v, n)`** *fin pour.*

Q-3-1 : Exécuter cet algorithme sur le vecteur $v = \{3, 1, 4, 1, 5, 9, 2, 6\}$. On indiquera les valeurs de M et K, et la valeur de v à la fin de chaque itération de γ).

Q-3-2 : Traduire cet algorithme en C. On rappelle qu'en C il existe une fonction **`int fmax(int a, int b)`** qui retourne le pus grand de ses deux arguments. Il n'y a pas d'élévation à la puissance dans les entiers et il est interdit d'utiliser celle des flottants.

Q-3-3 : Montrer qu'à la fin de chaque itération de la boucle γ) la propriété suivante est vérifiée :

$$\text{la suite} \qquad (v_0 \bmod 2^{p+1}, v_1 \bmod 2^{p+1}, \ldots, v_{n-1} \bmod 2^{p+1}) \quad \text{est croissante.}$$

Où "$a \bmod b$" est le reste de la division euclidienne de a par b.
En déduire que le vecteur v est trié par ordre croissant à la fin de l'algorithme.

Q-3-4 : On suppose que tous les entiers v_i son compris entre 0 et $2^{30} - 1$ (taille maximale d'un entier en C sur une machine 32 bits). Quelle est la complexité asymptotique de cet algorithme de tri comptée en nombre de transferts effectués ?

Annexe

Voici le listing pour l'Exercice 1.

```
1  /*-------------------------------------------*/
2  typedef  int TypeDonnee;
3
4  typedef struct Cell{
5  TypeDonnee donnee;
6  struct Cell* suivant;
7  }TypeCell;
8
9  /*-------------------------------------------*/
10
11 TypeCell* Creation(TypeDonnee a){
12 TypeCell* nouv;
13  nouv = (TypeCell*) malloc(sizeof(TypeCell));
14  nouv->suivant = NULL;
15  nouv->donnee = a;
16  return nouv;
17 }
```

```
1  /*-------------------------------------------*/
2  void InsereEnTete(TypeCell** pL, TypeDonnee a){
3  TypeCell *nouv;
4   nouv = Creation(a);
5   nouv->suivant=*pL;
6   *pL = nouv;
7  }
8  /*-------------------------------------------*/
9  void Affiche(TypeCell* L){
10  TypeCell* p;
11    printf("< ");
12    for(p = L; p!=NULL; p=p->suivant){
13      printf("%d\t", p->donnee);
14    }
15    printf("> \n");
16 }
```