Examen d'Algorithmique - (2011/2012 session 1)

Durée : 1heure 30 minutes - Aucuns documents autorisés

Formation Ingénieurs CNAM

Ce sujet comporte 3 pages et 2 parties indépendantes. On s'attachera à soigner la présentation du code afin qu'il soit le plus lisible possible. Il ne suffit en aucun cas d'écrire le code d'une fonction, il faut expliquer (i.e. commenter) les choix faits pour l'implantation. Il est de plus indispensable de respecter les notations données dans l'énoncé.

1 Echauffement

On rappelle la structure de données définie pour représenter les arbres d'entiers ainsi que les opérations de base `vide` et `e` (enracinement). Afin de pouvoir tester notre implantation, on programme également une fonction d'affichage `print`.

```
#include<stdio.h>

typedef struct arbre
{
  int x;
  struct arbre *g, *d;
} Sarbre, *Arbre;

Sarbre *vide()
{ return (NULL); }

Sarbre *e(Sarbre *ag, int v, Sarbre *ad) /* enracinement */
{
  Sarbre *m=(Sarbre *)malloc(sizeof(Sarbre));
  m->g=ag;
  m->x=v;
  m->d=ad;
  return m;
}

void print(Sarbre *a) /* affichage infixe */
{
  if (a!=NULL)
    {
      print(a->g);
      printf("%d ",a->x);
      print(a->d);
   }
```

```
  printf("\n");
}
```

Question 1 Programmer la fonction `oppose` (qui transforme chaque étiquette x en son opposé -x). On procèdera de manière récursive avec des effets de bord (le type de retour de la fonction demandée est `void`).

```
void oppose (Sarbre *a)
{ ... }
```

Question 2 Programmer de manière *récursive* une fonction `tous_positifs` qui vérifie que toutes les étiquettes de l'arbre sont positives.

```
bool tous_positifs(Sarbre *a)
{ ... }
```

Question 3 Programmer de manière *récursive* une fonction `nb_positifs` qui compte le nombre d'étiquettes positives dans un arbre.

```
int nb_positifs(Sarbre *a)
{ ... }
```

Question 4 Ecrire un programme de test dans la fonction `main` qui produit la sortie suivante :

```
int main()
{ ... }
```

```
user@computer$ ~/test-arbre
3 6 8
-3 -6 -8
tous_positifs ? = true
4 7 -9
nb_positifs = 2
-4 -7 9
nb_positifs = 1
user@computer$
```

2 Problème : le tri par compteur

On s'intéresse au problème de trier des donnés. Le tri par compteur est un algorithme très efficace quand toutes les données du tableau sont toutes incluses dans un petit intervalle.

On suppose que l'on veut trier un tableau t de n valeurs entières. Pour simplifier, on suppose que toutes les valeurs sont positives ou nulles. On commence par rechercher la valeur maximale (`max`) contenue dans le tableau t. On compte ensuite le nombre d'occurrences des éléments de t (qui sont compris en 0 et `max`). Pour cela, on utilise un tableau intermédiaire u de taille (`max`$+1$) où chaque case d'indice i représente le nombre d'occurrences de la valeur i dans le tableau t à trier. Il ne reste plus qu'à parcourir ce nouveau tableau u pour construire un nouveau tableau t' contenant exactement les mêmes éléments que le tableau initial t, et qui sera trié dans l'ordre croissant.

Question 5 On considère le tableau suivant (à 6 éléments) :

```
8 5 2 3 5 4
```

Construire le tableau des occurrences. Quelle est sa taille ? Expliquer comment l'utiliser pour obtenir le tableau trié résultat :

```
2 3 4 5 5 8
```

Question 6 Programmer une fonction qui calcule la valeur maximale des éléments d'un tableau t de taille n :

```
int max(int *t, int n)
{ ... }
```

Question 7 Construire une fonction `nb_occurrences` qui crée un tableau de taille `max` + 1 où `max` est la valeur retournée à la question précédente. On va le remplir avec le nombre d'occurrences des éléments du tableau t.

```
int * nb_occurences(int *t, int n)
{...}
```

Question 8 Programmer ensuite une fonction qui produit le tableau trié résultat t' à partir du tableau des occurences. `occ` représente le tableau des occurences produit à la question précédente et m sa taille. On pourra compter le nombre d'occurrences total (somme des éléments du tableau `occ`) pour déterminer la taille du tableau à construire. Expliquer pourquoi.

```
int * trie(int *occ, int m)
{...}
```

Question 9 Assembler tous les morceaux pour obtenir un programme de tri d'un tableau t de taille n en utilisant les fonctions des questions précédentes :

```
int *tri_compteur(int *t, int n)
{...}
```