Algorithmique et structures de données 2
TP3

am@up8.edu

2022

1 Exercice 1 : vérifier l'optimisation

Récupérez le fichier `tri_a_bulle_corrige.c` dans la section `corrigés/TP1` du moodle. Cet exercice vise à modifier le code pour pouvoir obtenir un affichage du type :

```
$./tri_a_bulle
~~~~ Tri à bulle opti ~~~~
trié avec i=993
Temps d'execution : 0.000052

~~~~~~~ Tri  à bulle ~~~~~~
Temps d'execution : 0.003068
```

Question 1 : mesurer le temps utiliser la fonction `clock()` du module `time.h` pour mesurer le temps passé dans un programme. Le temps passé entre deux instants **begin** et **end** est :

```
1 time_spent = (double)(end - begin) / CLOCKS_PER_SEC
```

Question 2 : définissez le(s) bon(s) tableau(x) de test

- choisissez un tableau de test pour lequel le temps d'exécution du tri à bulle optimisé devrait être significativement plus court que le tri à bulle.
- déclaration :

```
1 T[10] = {1,[1 ... 9] = 10};
```

  permet de déclarer le tableau de taille 10 : { 1, 10, 10, 10, 10, 10, 10, 10, 10, 10 }
- quelle est la fonction qui permet de copier un tableau dans une nouvelle variable ?

Question 3 : Factorisation Pour pouvoir comparer les temps d'exécution des deux fonctions, déclarez une fonction `time_spent`, qui prend en paramètre :

- un pointeur sur une fonction de tri `int (*pf)(int*, int size)`,
- les paramètres de la fonction de tri,

et qui renvoie `time` le temps passé à l'intérieur de la fonction de tri.

2 Exercice 2 : améliorer votre rendu

2.1 Modularité et compilation

Question 1 : définissez un module Si ce n'est pas déjà fait, créez pour votre rendu les fichiers arbres.c et arbres.h afin de disposer d'un module contenant la définition de la structure d'arbre et les fonctions que vous utilisez pour créer l'arbre, vérifier si c'est un ABR, etc.

Votre programme principal (initialisation et test de votre/vos arbre(s)) doit se trouver dans un autre fichier `TP3_ex1.c`.

Question 2 : créez un Makefile En suivant l'exemple du cours, écrivez un fichier `Makefile` vous permettant de compiler votre programme principal. Testez le bon fonctionnement de la compilation.

```
$ make clean
$ make all
```

2.2 Vérification du code

2.3 Question 1 : test

Vérifiez que votre programme fonctionne bien dans les cas suivants :
— l'abre est vide
— l'arbre est un ABR
— l'arbre n'est pas un ABR

2.4 Question 2 : fuites mémoire ?

Utilisez la commande suivante pour vérifier si l'exécution de votre programme génère des fuites mémoires.

```
$ valgrind --leak-check=full ./a.out
```

Si c'est le cas, implémentez une fonction `detruit_arbre` qui prend en paramètre un arbre et libère récursivement la mémoire allouée.

2.5 Nouvelle fonction

Fonction supprime Écrire une fonction qui prend en paramètre un ABR et une valeur et qui supprime le noeud contentant valeur tout en conservant les propriétés d'ABR. La fonction renvoie l'arbre dans lequel le noeud a été supprimé.

N'oubliez pas de faire tourner valgrind sur votre programme pour vérifier que vous ne générez pas de fuite mémoire.