TP « Benchmark de tri »

N. Delestre

Sujet

L'objectif de ce TP est d'écrire un programme C qui compare les 4 tris de base : le tri par insertion, le tri par sélection (ou tri par minimum successif), le tri rapide et le tri par fusion. Voici un exemple d'exécution de ce programme :

```
$ bin/benchTri
Nb d'entiers :50000
Test des tris sur 50000 entiers :
  Tri Par Minimum Successif : 3.284840s (temps CPU)
  Tri Par Insertion : 1.297865s (temps CPU)
  Tri Rapide : 0.006665s (temps CPU)
  Tri Par Fusion : 0.010241s (temps CPU)
```

Comportement du programme

L'exécution du programme suit les étapes suivantes :

— Remplir un tableau avec *n* entiers aléatoires (*n* saisi par l'utilisateur). n et le contenu du tableau seront de type `long int`.

— Pour chaque tri, un benchmark (fonction `benchmarkerUnTri`) réalise les actions suivantes :
  — allouer dynamiquement une zone mémoire temporaire de la même taille que le tableau à trier (utilisation de `malloc`),
  — copier le tableau à trier dans la zone mémoire temporaire (utilisation de la fonction `memcpy`) ;
  — stocker dans une variable de type `clock_t` l'heure courante (fonction `clock` de `time.h`) ;
  — lancer le tri sur cette zone mémoire temporaire ;
  — stocker dans une seconde variable de type `clock_t` l 'heure courante ;
  — libèrer la zone mémoire temporaire ;
  — retourner le temps (en seconde) mis pour effecteur le tri (différence entre les deux heures récupérées, transtypée en `double` et divisée par la constante `CLOCKS_PER_SEC`).

Fichiers du programme

Ce programme est composé des fichiers suivants :

— `src/main.c` : le programme principal

— `include/echanger.h` et `src/echanger.c` : les fonctions permettant d'échanger deux variables de même type (entre autres pour le type `int`, pour le type `long int`, etc.)

— `include/triParMinSuccessif.h` et `src/triParMinSuccessif.c`

— `include/triParInsertion.h` et `src/triParInsertion.c`
— `include/triRapide.h` et `src/triRapide.c`
— `include/triFusion.h` et `src/triFusion.c`
— `lib/libechanger.a` : la bibliothèque statique qui contient `echanger.o`
— `lib/libtris.a` : la bibliothèque statique qui contient `triParMinimumSuccessof.o`, `triParInsertion.o`, `triRapide.o` et `triFusion.o`
— `makefile`, tel que make créera le programme `bin/benchTri`

Consignes pour réaliser le TP

Suivez les étapes suivantes (on ne passe à l'étape $n+1$ que si l'étape n fonctionne sans problème) :

Étape 1 :

1. Créer les `.h` et les `.c` (fonctions sans corps pour les `.c`) : bien faire attention au nom des fichiers, des fonctions et aux types des paramètres formels (cf. le `main.c`) ;
2. Compiler et exécuter les tests unitaires (`make tests` et `tests/testTris`)
3. Compiler et exécuter le programme principal (`make` et `bin/benchTris`).

Étape 2 :

1. Développer la fonction `benchmarkerUnTri` de `main.c` ;
2. Compiler le programme principal.

Étape 3 :

1. Développer le tri par minimum successif ;
2. Compiler et exécuter les tests unitaires ;
3. Décommenter le `printf` de la fonction `benchmark` affichant les résultats du benchmark sur ce tri ;
4. Compiler et exécuter le programme principal.

Étape 4 :

1. Développer le tri par insertion ;
2. Compiler et exécuter les tests unitaires ;
3. Décommenter le `printf` de la fonction `benchmark` affichant les résultats du benchmark sur ce tri ;
4. Compiler et exécuter le programme principal.

Étape 5 :

1. Développer le tri rapide ;
2. Compiler et exécuter les tests unitaires ;
3. Décommenter le `printf` de la fonction `benchmark` affichant les résultats du benchmark sur ce tri ;
4. Compiler et exécuter le programme principal.

Étape 6 :

1. Développer le tri par fusion ;
2. Compiler et exécuter les tests unitaires ;
3. Décommenter le `printf` de la fonction `benchmark` affichant les résultats du benchmark sur ce tri ;
4. Compiler et exécuter le programme principal.

Étape 7 :

1. Tester le benchmark avec 10000, 20000, 50000 et 100000 éléments Modifier le main remplir au hasard de façon à trier un tableau déjà trié ;
2. Refaire le benchmark.