CS228 - TP1 - Tris de Tableaux et complexité expérimentale.

Grenoble INP – Esisar – année 2023-24

Version du 16 juin 2024

Objectif pédagogique

- Réviser les tris.
- Pratiquer/réviser les fiche de syntaxe C “niveau 1”, et “niveau 2”.
- Comparer expérimentalement les complexités des tris classiques en nombre de comparaisons.

Ni le code ni les réponses ne sont évaluées, l'objectif est de **pratiquer**.

Etape 1 : prise en main du code fourni

- Récupérer l'archive fournie, et la désarchiver (commande tar).
- Observer le code et le Makefile fournis (rapidement, pour vérifier qu'il ne s'agit pas de logiciel malveillant).
- Compiler avec la commande `make`. Comprendre le makefile fourni. Ajouter une option de compilation optimisante (O2), modifier le nom du compilateur utilisé (clang). Relancer la compilation
- Exécuter : `./main`

Note Le processus de compilation séparée sera vu au prochain TP.

Etape 2 : questions de compréhension et premier tri.

- Observer la déclaration des 3 tableaux non triés tab1, tab2, tab3. Comment leur taille est-elle récupérée dans le `main` ?
- Expliquer pourquoi on réalise une copie de tableau avant de faire un appel à une fonction de tri.
- Décommenter la ligne du main commençant par `assert`, compiler, exécuter et expliquer le résultat obtenu.
- Tester la fonction `swap` fournie (dans `fonction.c`) à partir du `main`. Il pourra être utile d'éditer le `.h`, pourquoi ?
- Observer et expliquer la signature des fonctions de tri dans `fonction.h`. Donner une alternative à la syntaxe `int *a`.
- Implémenter `insertion_sort` et vérifier que la fonction exécute correctement sur le tableau 1. Attention à bien calculer le nombre de comparaisons effectuées.

2B : si vous n'êtes pas en retard, ajouter le tri selection.

Etape 3 : comprendre la structure et implémenter le tri fusion.

- Observer la structure fournie pour l'implémentation du tri fusion. Quelles modifications voyez-vous par rapport au pseudo-code fourni dans le TD2 ?
- Expliquer le passage du paramètre compteur (cnt).
- Expliquer comment le tableau nécessaire pour la fusion est alloué, utilisé, recopié.

- Implémenter les TODO et tester le tri fusion.

Etape 4 : utiliser la fonction qsort de la librairie standard

- En regardant le man, réaliser un appel à la fonction `qsort` sur une copie du tableau 1.

Aide : on doit passer une fonction compare qui réalise une comparaison d'entiers, mais ceux-ci sont passés en paramètre comme ceci:

```
int compare_ints(const void *p1, const void *p2){
     ...
}
```

comment donc transformer un pointeur constant void en un entier "int"?

Etape 5 : générer des tableaux de tailles variables, comparer

L'objectif est de comparer les performances de nos implémentations du tri par insertion et du tri-fusion. Nous allons pour cela générer des tableaux de tailles croissantes, et imprimer le nombre de comparaisons obtenues. Il est fourni une fonction `init_random_array`.

- Essayer l'usage d'un tableau aléatoire à la place des tableaux fournis statiquement.
- Réaliser une fonction qui réalise un affichage sur la sortie standard de cette forme :

```
100,2422,316
200,9705,732
300,21988,1180
400,39136,1664
500,67748,2216
600,87867,2660
700,123417,3172
800,157143,3728
900,199768,4304
1000,253648,4932
```

la première colonne étant la taille étudiée (un tableau de cette taille, tiré au hasard), la deuxième le nombre de comparaisons du tri par insertion, la troisième le nombre de comparaisons du tri fusion, sur le même tableau, bien sûr.

- Verifier avec valgrind que vos tableaux dynamiquement alloués ont bien été désalloués.
- Au lieu d'un calcul de nombre de comparaison par taille par algorithme, faire une moyenne sur une 20 aine de cas. Attention à bien utiliser les mêmes tableaux.
- Utiliser un tableur pour regarder la croissance de ces comparaisons.

Etape 6 : utiliser des compteurs de temps.

Grenoble INP – Esisar – année 2023-24

Même exercice avec les "vraies" durées : utiser les fonctions de bibliothèque `time`.