Université Lille1 Sciences et Technologies

ASD

2012-2013
Licence STS - Semestre 4

TP2 : le tri rapide

L'objectif de ce TP est de programmer le tri rapide et d'évaluer l'incidence du choix du pivot.

Récupérez auparavant l'archive disponible sur le portail. Vous trouverez dans un répertoire `tp2` un module de génération de tableaux d'entiers, un module de tris avec le tri fusion déjà implémenté et un programme de test minimal nommé `testtp2`. Vous pourrez compiler votre programme grâce à un `Makefile` et tester votre programme en lançant `./testtp2`

1 Le tri rapide

Le principe du tri rapide consiste à partitionner le tableau à trier autour d'un pivot, les éléments inférieurs à ce pivot étant placés au début du tableau, à gauche du pivot, et les éléments supérieurs sont placés à droite du pivot, en fin de tableau. Puis on appelle récursivement ce tri sur chaque sous-tableau : la tranche contenant les éléments à gauche du pivot (`t1` ci-dessous), puis la tranche contenant les éléments à droite du pivot (`t2` ci-dessous). Le principe de l'algorithme peut être décrit ainsi :

```
tri_rapide t
   soit t1,t2 = partitionner t
   tri_rapide t1
   tri_rapide t2
```

Un exemple de partionnement est donné ci-dessous, où le pivot est le premier élément du tableau (ici 3) :

3 8 9 6 2 1 5 7 4
↓
2 1 **3** 4 7 5 6 9 8

La propriété du partitionnement est que **tous** les éléments plus petits que le pivot sont à gauche, et que **tous** les éléments les plus grands sont à droite. On remarque alors que le pivot est nécessairement à la position qu'il occupera lorsque le tableau aura été trié. Ainsi, après chaque partition un élément du tableau est correctement mis en place.

Avec le tri rapide il n'est pas nécessaire de copier le tableau à chaque étape, contrairement au tri fusion, puisque le pivot est directement mis à sa place définitive dans le tableau trié à chaque étape.

On appelle ces tris qui ne nécessitent pas de recopie de tableau des **tris sur place** (ou en place).

Q 1 Citez d'autres exemples de tris sur place.

Si on ne recopie pas le tableau, il faut passer en paramètre à la fonction de tri le tableau ainsi que les positions entre lesquelles on souhaite trier.

Plutôt que de passer le tableau et les indices définissant la tranche de tableau à traiter, nous allons définir un type `tranche` qui permettra de représenter cela (bien entendu, on ne créera pas de tranche de tableau par une copie). Le type tranche est donc un enregistrement qui contient une référence sur le tableau ainsi que deux entiers permettant de définir les positions gauche et droite de la tranche qui nous intéresse.

Q 2 Définissez le type `tranche` dans le module `Tris` :

```
type 'a tranche = { t : 'a array; g : int; d : int }
```

Ainsi, les définitions :

```
1 let mon_tableau = [| 1 ; 2 ; 3 ; 4 ; 5 |];;
2 let ma_tranche_1 = { t = mon_tableau; g = 2; d = 3 };;
3 let ma_tranche_2 = { t = mon_tableau; g = 0; d = 0 };;
```

permettent de définir la tranche qui contient les éléments `3; 4; 5` du tableau `mon_tableau` puis la tranche qui contient l'unique élément `1`.

Q 3 Réfléchissez sur la façon de partitionner un tableau sans disposer d'un tableau supplémentaire. Puis programmez dans le module `Tris` la fonction

`val partitionner : tranche -> ('a␣->␣'a -> int) -> tranche * tranche.`

qui réalise la partition d'une tranche en deux tranches. On choisira comme pivot l'élément situé en première position de la tranche passée en paramètre. Testez (vraiment).

Q 4 Programmez, en vous inspirant de l'écriture réalisée en cours, l'algorithme du tri rapide sur une tranche dans le module `Tris` :

```
val tri_rapide_tranche : tranche -> ('a -> 'a -> int) -> tranche.
```

Q 5 Enfin, programmez l'algorithme de tri rapide qui s'applique sur un tableau et qui utilise la fonction de la question précédente. Testez.

Q 6 D'après vous, quel est l'espace mémoire supplémentaire utilisé lors d'un tri rapide d'un tableau de longueur n ?

2 Sélection du pivot

Le choix du pivot dans le tri rapide est primordial. Un pivot mal choisi (le minimum ou le maximum) par exemple, et le tri ne sera pas plus rapide qu'un tri bulle.

2.1 Pivot aléatoire

Q 7 Dans le modules `Tris`, écrivez une fonction

```
pivot_aleatoire : tranche -> int
```

qui retourne une valeur alétoire de la tranche du tableau.

Q 8 Ajoutez un paramètre qui donnera le pivot à utiliser à la fonction `partitionner` (on supposera que le pivot est toujours un élément de la tranche à partitionner).

Q 9 Mofifiez vos codes pour compter le nombre de comparaisons effectuées par le tri rapide.

Q 10 Pour chacune des deux versions du tri rapide (pivot en première position, pivot aléatoire), calculez et stockez ces décomptes pour des tableaux de taille 1 à 100 tirés alétoirement (on effectuera la moyenne sur 100 tirages). Concluez.

2.2 Pivot optimal

Q 11 Quelle est théoriquement la meilleure valeur à choisir pour le pivot ?

Q 12 Dans le module `Tris`, programmez une fonction

```
pivot_optimal : tranche -> int
```

qui calcule cette valeur (**sans vous soucier de l'efficacité de ce calcul**). Testez.

Q 13 En utilisant ce choix de pivot, calculez et stockez le nombre de comparaisons pour des tableaux de taille 1 à 100 tirés alétoirement (on effectuera la moyenne sur 100 tirages).

Q 14 Réalisez une courbe compilant les chiffres produits au questions précédentes et permettant de voir l'évolution du nombre de comparaisons dans les trois versions. Concluez.

Q 15 Ajoutez aux comparaisons comptées celles de la fonction calculant le pivot. La tri rapide avec choix du pivot optimal est-il toujours le meilleur ?

Q 16 **Défi !** Sinon, trouvez un algorithme efficace du calcul du pivot optimal (vous pouvez bien sûr utiliser toute ressource qui vous permettra d'avancer dans votre quête).