TP6 : Tri rapide

MP2I Lycée Pierre de Fermat

Le but de ce TP est d'implémenter le tri rapide vu en cours, sans réserver de mémoire additionnelle, c'est à dire en travaillant **en place** dans le tableau d'entrée. **Vous pouvez faire le TP en binôme**. Il n'y a aucun rendu obligatoire, mais vous pourrez déposer sur Cahier de prépa un rapport / document-réponse si vous voulez une correction.

Étude théorique

On rappelle tout d'abord l'algorithme du tri rapide pas en place, qui **renvoie** une copie triée de son entrée :

Algorithme 1 : Tri_rapide

Entrée(s) : T un tableau de n éléments
Sortie(s) : T' copie triée de T

si $n \leq 1$ **alors**
  retourner ***une copie de*** T
sinon
  $p \leftarrow$ un indice dans $[\![0, n-1]\!]$ `// choix du pivot`
  $T_{\leq} \leftarrow [T[i] \mid i \in [\![0, n-1]\!], i \neq p, T[i] \leq T[p]]$;
  $T_{>} \leftarrow [T[i] \mid i \in [\![0, n-1]\!], i \neq p, T[i] > T[p]]$;
  Trier récursivement $T_{\leq}$ et $T_{>}$;
  $T' \leftarrow$ concaténation de $T_{\leq}$, $[x]$ et $T_{>}$;
  retourner T'

Question 1. Exécuter l'algorithme du tri rapide, en choisissant toujours comme pivot la première case, pour trier le tableau suivant :

$$[7, 5, 14, 7, 8, 24, 9, 3, 8, 4, 6, 12, 10, 12]$$

Passons maintenant à l'implémentation en place de cet algorithme. Nous allons utiliser un premier algorithme servant à séparer un tableau en deux parties selon une valeur pivot :

Algorithme 2 : Partition

Entrée(s) : T tableau de taille n, $i \in [\![0, n-1]\!]$
Sortie(s) : T est modifié, l'ancienne valeur de $T[i]$ est maintenant à un indice j tel que $\forall k \in [\![0, j-1]\!], T[k] \leq T[j]$ et $\forall k \in [\![j+1, n-1]\!], T[k] > T[j]$

Cet algorithme va donc diviser le tableau en deux parties, en classant les éléments selon leur comparaison avec $T[i]$: les éléments inférieurs ou égaux à gauche, les éléments strictement supérieurs à droite. Tout cela est fait en place, sans réserver de mémoire additionnelle.

Par exemple, si $T = [5, 2, 7, 8, 1, 5, 9, 8, 6, 2]$, après avoir appelé **Partition**$(T, 0)$, on a séparé T en utilisant $T[0] = 5$ comme pivot. Après l'exécution de l'algorithme, T pourrait donc contenir $[2, 1, 5, 2, \mathbf{5}, 7, 8, 9, 8, 6]$.

L'ordre au sein de chaque partie est arbitraire et dépend de l'implémentation de l'algorithme. Dans l'exemple précédent, c'est l'ordre d'apparition dans le tableau T avant la partition, ça ne sera peut être pas ce que votre algorithme donnera.

Question 2. On suppose tout d'abord que l'on a réussi à écrire cet algorithme de partition. En l'utilisant comme brique de base, écrire le pseudo-code du tri-rapide en place. Montrer par récurrence forte que $\forall n \in \mathbb{N}, \forall T$ de taille n, l'algorithme de tri rapide trie T.

On s'intéresse maintenant à l'écriture de l'algorithme de partition.

Question 3. Proposer un algorithme en $\mathcal{O}(n)$ correspondant à la spécification.

Question 4. Justifier la complexité de l'algorithme précédent, et trouver un invariant de boucle permettant de montrer sa correction.

Implémentation en C

Question 5. Implémenter un tri par sélection ou par insertion, et le tester en utilisant une fonction `bool est_trie (int* T, int n)` déterminant si un tableau est trié dans l'ordre croissant.

Question 6. Implémenter le tri rapide en place, et le tester.

Nous allons comparer les performances du tri rapide avec celles du tri quadratique choisi. Pour cela, nous devons mesurer leur temps d'exécution, mais la fonction `time()` ne permet de faire des mesures qu'à la seconde près.

La librairie `<time.h>` contient la fonction `clock_t clock()`. Cette fonction renvoie le temps écoulé depuis le lancement du programme, dans une unité arbitraire appelé les *clocks*.[1] Un **clock** vaut environ un millionième de seconde, ce qui permet d'avoir des mesures plus précises. Le nombre exact de **clocks** par seconde est accessible avec `CLOCKS_PER_SEC`.

Question 7. Écrire une fonction `float test_tri_rapide (int n)` qui génère 20 tableaux aléatoires de taille n, les trie avec le tri rapide, et renvoie le nombre moyen de millisecondes écoulé par tableau.

Question 8. Écrire une fonction analogue pour le tri quadratique choisi.

Question 9. Mesurer le temps moyen d'exécution du tri rapide sur $n \in [100, 200, 300, \ldots 10000]$ et stocker les valeurs mesurées dans un fichier.

Question 10. Faire de même avec le tri quadratique.

Question 11. En python, tracer les courbes afin de comparer les deux algorithmes.

Question 12. Comment peut-on se convaincre en traçant autrement les données que le tri quadratique est bien en $\Theta(n^2)$ et que le tri rapide est bien en $\Theta(n \log n)$?

1. Si vous avez des notions d'architecture des ordinateurs : attention, un clock ne correspond pas du tout à un tic d'horloge du CPU !