TP 5 : tri par pivot

Judicaël Courant

2 janvier 2017

Ce problème est en partie inspiré de l'article *Engineering a Sort Function* de Jon L. Bentley et M. Douglas McIlroy pour l'implantation d'un algorithme de tri par pivot efficace.

Vous trouverez sur le site web de la classe le code vu en classe pour trier un tableau avec l'algorithme de tri par pivot dans un fichier appelé `quicksort.py`.

Ce fichier contient trois fonctions permettant de trier un tableau t : `tri_selection(t)`, `tri_insertion(t)` et `tri_rapide(t)`.

Il contient également des fonctions permettant de générer un tableau de taille n arbitraire : `tableau_croissant(n)` (resp. `tableau_croissant(n)`) retournant les n premiers entiers naturels, dans l'ordre croissant (resp. décroissant), `tableau_dense(n)` retournant un tableau ne comportant que des 0 et des 1, choisis de façon équiprobables, `tableau_aleatoire(n)` retournant des valeurs dans $[\![0, n[\![$, tirées de façon équiprobables et enfin `tableau_triangle(n)` retournant un tableau dont la première moitié croît puis la seconde décroît.

Enfin, il contient une fonction `bench(tri, methode, duree, fichier)`. Voyons sur un exemple comment elle s'utilise. L'exécution de

```
bench(tri_insertion, tableau_aleatoire, 1., 'tri_insertion_aleatoire.dat')
```

note le temps de calcul de la fonction `tri_insertion(t, 0, len(t))` sur des tableaux t de tailles n de plus en plus grandes, construits par la fonction `tableau_aleatoire(n)`. Pour chaque essai elle écrit, dans le fichier «`tri_insertion_aleatoire.dat`», une ligne contenant le logarithme décimal de la taille du tableau n, puis un caractère espace, puis le logarithme décimal du temps d'exécution t.

Vous trouverez dans le fichier `lance_tests.py` les commandes permettant de générer les douze combinaisons possibles (quatre méthodes pour générer un tableau, trois fonctions de tri).

Pour regarder les résultats, on utilisera un outil appelé `gnuplot`. Avec n'importe quel éditeur de texte, on écrit un fichier contenant des commandes pour gnuplot, puis on exécute la commande gnuplot dans un terminal sur ce fichier. Par exemple, j'ai écrit un fichier `tris.plot` contenant des commandes pour afficher les résultats générés par mon script `lance_tests.py`. Pour le lancer, il me suffit de taper, dans un terminal, la commande «`gnuplot -p tris.plot`». Vous pouvez vous inspirer de ce fichier pour tracer vos graphiques.

Q0 Écrire une fonction `tableau_constant(n)` retournant un tableau de `n` éléments, tous égaux (par exemple tous nuls).

Q1 En vous inspirant de `lance_tests.py`, créer des fichiers de données pour comparer les différentes fonctions de tri sur des tableaux constants, les unes par rapport aux autres et par rapport à ce qu'il se passe sur un tableau aléatoire. Les afficher pour comparer visuellement.

Q2 Expliquer. Expliquer également les mauvais résultats du tri rapide sur un tableau ne contenant que des 0 et des 1.

La méthode de partionnement utilisée dans le cours pour le tri pivot est attribuée à Lomuto. On propose maintenant une autre méthode de partitionnement, adaptée du code proposé par Bentley et McIlroy, effectuant un partitionnement trois voies. Le code en est donné dans le fichier `bentleyqsort.py`.

Q3 Lire la *spécification* de la fonction `bentley_partition(t, g, d)`.[1] Faire un dessin. Ne pas chercher à comprendre l'*implantation* de la fonction pour l'instant.

Q4 Dans cette question, et dans cette question seulement, on fait l'hypothèse que la fonction `bentley_partition(t, g, d)` respecte sa spécification. Montrer alors que la fonction `tri_rapide_b(t)` trie le tableau qui lui est passé en argument.

Montrons maintenant que la fonction `bentley_partition(t, g, d)` respecte la spécification qui est donnée en documentation.

Q5 Représenter l'invariant (1) par un dessin.

Q6 Justifier que l'invariant (1) est vérifié avant l'entrée dans la boucle.

On considère maintenant un tour de boucle au début duquel l'invariant (1) est vérifié.

Q7 Montrer qu'alors, en arrivant à la ligne marquée (2), la propriété $j \leq k$ est vérifiée.

Q8 Montrer qu'alors en arrivant aux lignes marquées (3) et (4) la propriété (3) est vérifiée et en déduire que la propriété (4) est vérifiée.

Q9 Montrer qu'alors l'invariant (1) est vérifié en fin du tour de boucle.

Q10 Montrer que la boucle `while True:` termine (i.e. il y a un moment où on exécute l'instruction `break`.

Q11 Montrer qu'à la sortie de la boucle, la propriété notée (4) est vérifiée.

Q12 En déduire que la fonction `bentley_partition(t, g, d)` respecte sa spécification.

Q13 Étudier maintenant les performances de cette fonction par rapport aux fonctions précédentes, notamment sur des tableaux constants.

Q14 Écrire une fonction effectuant le partitionnement en suivant la méthode de Bentley et McIlroy mais choisissant pour pivot la valeur située au milieu de la partie du tableau à trier. Comparer avec les méthodes précédentes.

Q15 Même question mais cette fois-ci prendre pour pivot le médiane des trois valeurs situées au début, à la fin et au milieu du tableau à trier.

Q16 Modifiez votre fonction de façon à ce que, lorsque la portion du tableau à trier est de taille suffisamment petite (par exemple moins de 20 éléments), elle soit triée par insertion. Étudiez sa performance par rapport aux versions précédentes.

1. C'est-à-dire les 9 lignes de documentation au début de la fonction.

