Mini-projet Algorithmes en Python

Hugues Talbot & Jean-Christophe Ketzinger

Ce mini-projet contient deux problèmes qui utilisent tous les trois l'idée de « diviser pour régner ».

1 Tri de valeurs entières

On se propose de trier des valeurs entières. On montre que dans ce cas on peut trier très efficacement.

1.1 Tri par insertion dans une liste

Étant donné un tableau de nombres aléatoires entiers non triés, le tri par insertion consiste à commencer par une liste contenant le premier élément du tableau, puis a insérer chacun des éléments dans la liste au bon endroit, de telle manière que la liste soit triée à la fin.

1. Générez un ensemble de N nombres entiers aléatoires dont la valeur minimum est 0 et la valeur maximum 10^5. On prendra $N = 100$ au départ, mais ensuite on fera varier N entre 10^4 et 10^5. Stockez vos N nombres dans une file d'attente (FIFO) mise en œuvre dans les premiers TPs.
2. Mettez en œvre le tri par insertion. Pour ce faire utilisez la structure de liste que vous avez développée dans les premiers TPs.
3. Testez votre tri par insertion avec un nombre d'entiers N variant entre 10^2 et 10^5. Affichez vos temps de calculs sur une courbe (utilisez le paquet python `matplotlib`).

1.2 Tri par seaux

Vous devez constater que le tri par insertion est rapide pour des petit N mais pas pour des grands N. Comme les nombres à trier sont entiers, une idée est la suivante :

— Diviser l'intervalle entre le plus petit et le plus grand entier en un certain nombre de « seaux », par exemple 100. Dans le premier « seau », on mettra tous les nombres entre 0 et 99, puis dans le 2^e, tous les nombres entre 100 et 199, etc.

— Dans chaque « seau », on maintient une liste triée des nombres qui arrivent. La liste est triée par le tri par insertion vu à la question précédente.

— Une illustration est donnée ici : `https://www.cs.usfca.edu/~galles/visualization/BucketSort.html`.

Ce tri s'appelle « bucketsort » en anglais. Vous devez faire les choses suivantes :

1. Ecrivez une structure de donnée adéquate pour le tri par seaux (par exemple un tableau de listes, etc).
2. Mettez en œuvre le tri par seaux.
3. Testez votre méthode pour 100 seaux, avec N entre 100 et 10^5 entiers. Que constatez vous ?
4. Testez avec un nombre variable de seaux entre 10 et 1000. Que constatez vous ?
5. Comparez votre tri par seaux avec l'algorithme du tri par tas (heapsort) vu au TP précédent.
6. Comparez tous les temps de calculs sur une courbe (matplotlib).

2 Points les plus proches

On se propose de trouver efficacement les deux points les plus proches dans une liste de points géométriques.

1. Ecrivez un programme qui simule N points aléatoirement répartis dans un carré 100×100. On prendra $N = 100$ pour commencer, mais ensuite $N = 10^3, N = 10^4$ et $N = 10^5$ devront être testés. Vous devez stocker les coordonnées de ces points comme vous voulez, par exemple en définissant une classe de points et une liste ou un tableau de tels points. Les coordonnés de ces points seront en virgule flottante.
2. Trouvez les deux points les plus proches dans cette liste au sens de la distance euclidienne. Par exemple, si la liste de points contient les points $A = (10,5;11,2)$, $B = (50,1;30,5)$, $C = (51,3;35,4)$, $D = (87,2;2,3)$ alors les points B et C sont les plus proches.
3. Proposez un algorithme de complexité quadratique pour trouver ces 2 points.
4. Proposer un algorithme de meilleure complexité en utilisant une technique de « diviser pour régner » similaire à celle des tris efficaces.
5. Tester votre algorithme pour différentes valeurs de N.