Lycée Louis-Le-Grand, Paris
MPSI 4 – Informatique pour tous
A. Troesch, J.-P. Becirspahic

2013/2014

TP 5 bis : le problème de la sélection

Un *médian* d'un ensemble $X = \{e_1, e_2, \ldots, e_n\}$ de n nombres entiers *distincts* est un nombre e dans X tel que les nombres d'éléments strictement plus petits et strictement plus grands que e dans X diffèrent d'au plus de 1. Si n est impair, le médian est unique ; si n est pair il y a deux médians possibles.

Le *problème de la sélection* consiste à trouver l'élément de rang k dans X, c'est-à-dire l'élément e se trouvant en k-ième position quand X est trié par ordre croissant.
Par la suite, nous supposerons l'ensemble X représenté par la liste de ses éléments.

Résolution par segmentation

1. Soit p un entier quelconque ; par la suite, on sera amené à partitionner l'ensemble X en éléments plus grands, égaux ou plus petits que p. Écrire la fonction `nPetits` qui prend en arguments un entier p et l'ensemble X et qui retourne le nombre d'éléments strictement inférieurs à p dans X.
   Quelle est la complexité temporelle de cette fonction par rapport à n ?
2. Pour sélectionner le k-ième élément de X, on prend un nombre p bien choisi dans X appelé *pivot* et on effectue une partition de X par rapport à p.
   Écrire la fonction `partitionP` qui prend en arguments un entier p et l'ensemble X et qui retourne la liste de tous les éléments de X strictement plus petits que p.

   Modifier cette fonction pour obtenir la fonction `partitionG` qui retourne la liste de tous les éléments de X strictement plus grands que p. Quelles sont les complexités temporelles de ces fonctions par rapport à n ?
3. Écrire la fonction `elementDeRang` qui prend en arguments un nombre entier naturel k et l'ensemble X et qui retourne l'élément de rang k dans X, en choisissant comme pivot le premier élément de la liste représentant X.
4. Le nombre d'opérations effectuées par cette fonction varie en fonction du choix du pivot. Donner un ordre de grandeur, par rapport à n, du nombre maximum $M(n)$ d'opérations utilisées par la fonction précédente. Dans quelle situation ce nombre maximal est-il atteint ?
5. Il est cependant possible de démontrer qu'en moyenne le coût de cette fonction est linéaire. Nous allons expérimenter cette assertion en considérant pour X une permutation de l'ensemble $[\![1, n]\!]$.
   Vous allez tracer sur un graphe le temps $T(n)$ mis par la fonction `elementDeRang` pour trouver l'élément de rang $\lfloor n/2 \rfloor$ dans une permutation arbitraire de $[\![1, n]\!]$, et ce pour $n = 100, 150, 200, 250, \ldots, 10000$.

   Pour tracer un graphe, utiliser la fonction `plot` du module `matplotlib.pyplot`. Cette fonction prend en paramètres deux listes $[a_1, a_2, \ldots, a_n]$ et $[b_1, b_2, \ldots, b_n]$ et ouvre une nouvelle fenêtre dans laquelle sera tracée la ligne brisée reliant les points de coordonnées (a_i, b_i).
   Vous utiliserez la fonction `time()` du module `time` pour mesurer le temps d'exécution, et la fonction `permutation` du module `numpy.random` pour générer une permutation arbitraire de $[\![1, n]\!]$.

Une solution de coût linéaire

On cherche désormais à optimiser le coût dans le pire des cas en choisissant judicieusement le pivot. Pour ce faire, on considère les sous-ensembles $X_i = \{e_{5i+1}, e_{5i+2}, e_{5i+3}, e_{5i+4}, e_{5i+5}\}$ de 5 éléments consécutifs de X, avec $0 \leqslant 5i \leqslant n - 5$.
On note Y l'ensemble des médians de chacun des X_i.

6. Écrire une fonction `medians` qui prend en argument l'ensemble X et qui retourne l'ensemble Y.
   Quelle est la complexité temporelle de cette fonction par rapport à n ?
7. Pour améliorer la vitesse de la fonction de sélection, on prend à présent comme pivot le médian de l'ensemble Y.
   Écrire la fonction `elementDeRangBis` qui prend en argument un entier naturel k et l'ensemble X et qui retourne l'élément de rang k dans X, en prenant comme pivot le médian de l'ensemble Y.
8. Le choix d'un tel pivot permet de s'assurer que la recherche se poursuivra dans une liste de taille « sensiblement plus petite » que la liste initiale, et il est en effet possible de prouver que dans ces conditions le coût dans le pire des cas est linéaire.
   En guise d'illustration, comparer les performances des fonctions `elementDeRangBis` et `elementDeRang` dans le cas le plus défavorable à cette dernière, puis avec des listes arbitraires.