Calcul de complexité du TD n°2

AP3 Algorithmique et programmation — L2 mathématiques

4 octobre 2017

(a) Rappelons qu'un *passage* sur la liste `li` est donné par le bout de code suivant

```
for j in range(len(li)-1):
  if li[j] > li[j+1]: li[j],li[j+1] = li[j+1],li[j]
```

On prouve par récurrence qu'au bout de k passages, les k plus grand éléments de `li` sont placés dans l'ordre croissant au k dernières places de `li`.

Initialisation : $k = 0$. La condition étant complètement creuse pour $k = 0$, elle est bien entendue vérifiée.

Hérédité. Supposons la propriété vrai pour un certain $k \geq 0$, et montrons qu'elle reste vraie pour $k+1$. Appelons x le $k+1$-ième plus grand élément de `li`. Pour montrer la propriété au rang $k+1$, il s'agit de montrer qu'après un nouveau passage, l'élément x est à la position `len(li)-k-1`. Par hypothèse de récurrence, x est maximal pour la liste `[li[0],...,li[len(li)-k-1]]` : ainsi, si on appelle `i` la position de départ de l'élément x, alors tous les échanges ont lieu pendant le passage entre les tours de boucles `j` $=$ `i` et `j` $=$ `len(li)-k-2`. Comme de plus x est plus petit que les k plus grand éléments de `li`, aucun échange n'est effectué entre les tours `j` $=$ `len(li)-k-1` et `j` $=$ `len(li)-2`. On a donc bien, après le passage, que `li[len(li)-k-1]` vaut x.

Cette propriété au rang $n-1$ dit exactement que la liste est triée.

(d) Rappelons la fonction de tri par bulles dont on veut calculer la complexité :

```
def bubble_sort(li):
  tmp = li[:]
  for i in range(len(li)-1):
    for j in range(len(li)-1):
      if tmp[j] > tmp[j+1]: tmp[j],tmp[j+1] = tmp[j+1],tmp[j]
  return tmp
```

Notons L, T, A le nombre d'opérations effectuées respectivement par une assignation, un test entre entiers et une addition d'entiers. Et notons $\gamma(n)$ le nombre d'opérations effectuées par l'algorithme ci-dessus sur une liste de longueur n. Le corps de la boucle intérieure effectue (au pire) un test, 3 additions et 2 assignations, pour un nombre d'opérations totale de $T+3A+2L$. Ainsi le nombre total d'opérations effectuées par la double boucle est :

$$\sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(T+3A+2L) = (T+3A+2L)(n-1)^2$$

On suppose que l'instruction de copie `tmp = li[:]` se fait en un temps constant[1] C. On a alors une complexité totale donnée par

$$\gamma(n) = (T + 3A + 2L)(n-1)^2 + C$$

Mettons, *de force*, n^2 en facteur :

$$\gamma(n) = \left((T + 3A + 2L)\left(\frac{n-1}{n}\right)^2 + \frac{C}{n^2}\right) \times n^2$$

et donc, dès que $n \geq 2$, en utilisant qu'alors $1/2 \leq (n-1)/n \leq 1$, on a :

$$\frac{T + 3A + 2L + C}{4} \times n^2 \leq \gamma(n) \leq (T + 3A + 2L + C) \times n^2$$

Ce qui montre que $\gamma(n) = \Theta\left(n^2\right)$

(e) Si aucun échange n'est effectué pendant un passage, cela siginifie que pour tout `j` ente `0` et `len(li)-2`, on a `li[i]` $\leq$ `li[i+1]`. C'est la définition même d'une liste triée.

(f) Rappelons le code de la fonction dont on veut déterminer la complexité :

```
def bubble_sort_better(li):
  tmp = li[:]
  for i in range(len(li)):
    swap = False
    for j in range(len(li)-1):
      if tmp[j] > tmp[j+1]:
        tmp[j],tmp[j+1] = tmp[j+1],tmp[j]
        swap = True
    if not swap: break
  return tmp
```

On reprend les notations L, T, A données précédemment. On note $\gamma(n)$ le nombre d'opérations effectuées par l'algorithme ci-dessus **dans le pire des cas**[2] sur une liste de taille n. Le corps de boucle intérieure effectue un test, 3 assignations, et 3 additions, donc le nombre d'opérations de la boucle

```
for j in range(len(li)-1):
  if tmp[j] > tmp[j+1]:
    tmp[j],tmp[j+1] = tmp[j+1],tmp[j]
    swap = True
```

est $\sum_{j=0}^{n-2}(T+3L+3A)$. Outre cette boucle, le corps de la boucle sur `i` effectue une assignation et un test, donc le nombre d'opérations de la double boucle s'élève dans le pire des cas à :

$$\sum_{i=0}^{n-2}\left(\sum_{j=0}^{n-2}(T+3L+3A)+A+T\right) = (T+L+3A)(n-1)^2+(A+T)(n-1)$$

1. Ce n'est bien sûr pas vrai, mais cette copie ne fait pas vraiment partie de l'algorithme de tri à bulles et cela permet de simplifier.

2. Un bon nombre d'entre vous m'ont majoré la complexité dans le pire des cas et minoré dans le meilleur des cas. Ca n'a pas de sens : ou bien on détermine la complexité dans le pire des cas, ou bien dans le cas moyen (ce qui est souvent difficile à définir), ou bien dans le meilleur des cas (ce qui a souvent aucun intérêt). Dans ce cours, ce sera toujours la complexité dans le pire des cas : on cherche le comportement asymptotique du nombre d'opérations que l'algorithme peut être amené à faire.

En ajoutant le nombre d'opérations de l'instruction de copie qu'on suppose encore constante égale à C, on obtient pour tout n :

$$\gamma(n) = (T + 3L + 3A)(n-1)^2 + (A+T)(n-1) + C$$

et donc, dès que $n \geq 2$,

$$\frac{2T + 3L + 4A + C}{4} \times n^2 \leq \gamma(n) \leq (2T + 3L + 4A + C) \times n^2$$

On en déduit donc $\gamma(n) = \Theta(n^2)$.

(g) On propose l'amélioration suivante de l'algorithme précédent :

```
def bubble_sort_best(li):
  tmp,n = li[:],len(li) # copy li
  pos = n-1
  for i in range(n):
    swap = False
    for j in range(pos):
      if tmp[j] > tmp[j+1]:
        tmp[j],tmp[j+1] = tmp[j+1],tmp[j]
        swap,pos = True, j
    if not swap: break
  return tmp
```

Non seulement, on se rappelle si un échange a eu lieu lors d'un passage, mais on se rappelle même la position de cet échange : le passage suivant peut alors s'arrêter à cette position. En effet, si aucun échange n'a lieu après une position `i`, c'est que la liste `li[i:]` (i.e. la liste à partir de la positioin `i` et jusqu'à la fin) est déjà triée (c.f. question (e)).

Le calcul de la complexité change un peu ici, car la boucle sur la variable `j` n'opère pas troujours $n-1$ tour. Une rapide analyse similaire à celle de la question (a) montre que lors du tour de boucle `i`, la variable `pos` est au plus `len(li)-1-i`. Ainsi, en comptant un test, 3 additions et 4 assignations dans le pire des cas pour le corps de la boucle portant sur `j`, on a que le nombre d'opérations effectuées par cette boucle

```
for j in range(pos):
  if tmp[j] > tmp[j+1]:
    tmp[j],tmp[j+1] = tmp[j+1],tmp[j]
    swap,pos = True, j
```

est au pire de $\sum_{j=0}^{n-2-i}(T + 3A + 4L)$. En observant qu'on fait également une addition et deux affectations de plus dans le corps de la fonction en dehors de la double boucle, on obtient un nombre d'opérations total sur une entrée de taille n de :

$$\begin{aligned}
\gamma(n) &= \sum_{i=0}^{n-2}\left(\sum_{j=0}^{n-2-i}(T + 3A + 4L) + A + T\right) + A + 2L + C \\
&= (T + 3A + 4L)\sum_{i=0}^{n-2}(n-1-i) + (A+T)(n-1) + A + 2L + C \\
&= (T + 3A + 4L)\frac{(n-1)n}{2} + (A+T)(n-1) + A + 2L + C
\end{aligned}$$

et donc, dès que $n \geq 2$,

$$\frac{2T + 5A + 6L + C}{4} \times n^2 \leq \gamma(n) \leq (2T + 5A + 6L + C) \times n^2$$

On en déduit donc que $\gamma(n) = \Theta\left(n^2\right)$.

Une remarque importante : bien que les version en (f) et (g) soient plus malines et plus rapides en pratique que la version en (b) du tri à bulles, leur complexité est la même : c'est-à-dire que leur comportement asymptotique sur de grandes valeurs de n sont d'ordres de grandeur comparables. Attention aux amalgames donc : un programmeur peut très bien chercher à rendre son code plus rapide sans en changer la complexité ! Bien entendu, on se place plutôt dans ce cours dans le rôle du mathématicien étudiant un algorithme : c'est donc le comportement asymptotique qui nous intéresse et pas les détails de l'implémentation.