

Correction TP n° 3 (X2004 - première partie)

BATOG Guillaume

10 janvier 2007

```
let rec card = function
  | [] -> 0
  | t::q -> 1+(card q);;

let rec partitionP p = function
  | [] -> []
  | t::q -> let res = partitionP p q in
            if (t<p) then t::res
            else res;;

let rec partitionG p = function
  | [] -> []
  | t::q -> let res = partitionG p q in
            if (t>p) then t::res
            else res;;
```

Le pivot est en tête. La fonction proposée est récursive. On prendra garde à chercher l'élément de rang $k-p-1$ dans l'ensemble des éléments plus grand que le pivot si $k > p+1$ où p est le nombre d'éléments plus petits que le pivot. Le pire des cas se produit lorsque la liste est triée dans l'ordre décroissant en cherchant l'élément de rang 1 : on obtient une complexité quadratique (on applique la fonction de partition – linéaire – successivement à des listes de taille $n, n-1, \dots, 1$).

```
let rec elementDeRang k = function
  | [] -> failwith "l'élément n'existe pas"
  | [t] -> if k=1 then t
           else failwith "l'élément n'existe pas"
  | t::q -> let petit = partitionP t q
            and grand = partitionG t q in
            let p = card petit in
            if (p>=k) then elementDeRang k petit
            else if (p=(k-1)) then t
            else elementDeRang (k-p-1) grand;;
```

Établissons la complexité moyenne de cet algorithme en supposant équiprobables toutes les permutations possibles sur les éléments d'une liste : à une liste de taille n correspond une permutation de \mathfrak{S}_n à travers la fonction rang. Voyons les conséquences de cette hypothèse à travers le lemme suivant.

Lemme 1. Soit I_i (resp J_i) l'ensemble des permutations produites par `partitionP pivot sigma` (resp `partitionG pivot sigma`) où le pivot est de rang i et où σ parcourt \mathfrak{S}_n de façon

équiprobable. I_i (resp J_i) est isomorphe à \mathfrak{S}_{i-1} (resp \mathfrak{S}_{n-i}) et tous ses éléments sont équiprobables.

Ce lemme est fondamental pour l'analyse de la complexité. Imaginons que `partitionP` retourne toujours un résultat tel que le premier élément soit de rang 1, il n'y a plus équirépartition des permutations produites ! Au cours du premier appel de `elementDeRang` sur une liste σ de taille n :

- les calculs de `petit`, `grand` et `p` sont linéaires en n donc on peut majorer leur temps total d'exécution par ln où l est une constante indépendante de σ donc le temps moyen de ces exécutions est majoré par ln ;
- notons $\sigma_1 = \text{partitionP } \sigma(1) \sigma$ et $\sigma_2 = \text{partitionG } \sigma(1) \sigma$, on appelle alors la fonction récursive `elementDeRang` sur σ_1 ou σ_2 . Notons $T'(n)$ le temps moyen d'exécution de cet appel et $T'(n, i)$ ce même temps lorsqu'en plus le pivot est de rang i , ie avec probabilité $(n-1)!/n! = 1/n$. La formule de la moyenne donne $T'(n) = \frac{1}{n} \sum_{i=1}^n T'(n, i)$. L'appel récursif se fait sur $\sigma_1 \in I_i$ ou $\sigma_2 \in J_i$ avec les notations du lemme. On obtient la majoration

$$T'(n, i) \leq \max(T(i-1 | \sigma_1 \in I_i), T(n-i | \sigma_2 \in J_i))$$

où on note $T(k | \sigma \in K)$ le temps moyen d'exécution de l'algorithme sur l'ensemble des permutations de K sous-ensemble de \mathfrak{S}_k . D'après le lemme, $T(i-1 | \sigma_1 \in I_i) = T(i-1)$ et $T(n-i | \sigma_2 \in J_i) = T(n-i)$.

- finalement, on obtient la majoration de complexité suivante :

$$T(n) \leq ln + \frac{1}{n} \sum_{i=1}^n \max(T(i-1), T(n-i)).$$

On fait un raisonnement par analyse/synthèse pour montrer que $T(n)$ est linéaire. Analyse : supposons qu'il existe une constante c telle que pour tout $n \geq 1$, $T(n) \leq cn$. La majoration précédente devient alors pour tout $n \geq 1$:

$$\begin{aligned} T(n) &\leq ln + \frac{c}{n} \sum_{i=1}^n \max(i-1, n-i) \leq ln + \frac{2c}{n} \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} i - 1 \\ &\leq ln + \frac{2c}{n} \frac{n/2(n/2+1)}{2} \leq (l + \frac{c}{4})n + \frac{c}{2} \leq (l + \frac{3c}{4})n. \end{aligned}$$

En choisissant $c = 4l$, on obtient $T(n) \leq (l + 3l)n = 4ln$. Synthèse : on montre par récurrence sur n que $T(n) \leq 4ln$ pour tout $n \geq 1$ (on a tout fait pour).

Le pivot est le médian des médians. Plus précisément ici, c'est le médian des médians des paquets de cinq éléments consécutifs dans la liste. Puisque l'énoncé demande exactement $\lfloor n/5 \rfloor$ médians, on ne se soucie pas du dernier groupement de cardinal strictement inférieur à 5. Pour calculer le médian sur ces paquets de taille bornée, on peut utiliser la fonction quadratique `elementDeRang`. Par contre, pour calculer le pivot, qui est le médian des médians des groupements, on ne doit pas réutiliser la fonction `elementDeRang` si on veut améliorer la complexité de recherche du k^e élément !

```
let rec medians = fonction
| [t1;t2;t3;t4;t5]@q -> (elementDeRang 3 [t1;t2;t3;t4;t5])::(medians q)
| _ -> [];;
```

```

let rec elementDeRangBis k = fonction
  | [] -> failwith "l'élément n'existe pas"
  | [t] -> if k=1 then t
            else failwith "l'élément n'existe pas"
  | l -> let med = medians l in
         let k = card med in
         let pivot = elementDeRangBis (k/2) med in
         let petit = partitionP pivot q
         and grand = partitionG pivot q in
         let p = card petit in
         if (p>=k) then elementDeRangBis k petit
         else if (p=(k-1)) then t
         else elementDeRangBis (k-p-1) grand;;

```

Calculons la complexité de ce nouvel algorithme dans le pire des cas. Au cours du premier appel de la fonction :

- les calculs de `med`, `k`, `p`, `petit` et `grand` se font en temps linéaire, on peut majorer le temps total de ces exécutions par $l'n$ où l' ne dépend pas de la liste de taille n choisie ;
- le calcul de `pivot` se fait en temps $M'(\lfloor n/5 \rfloor)$;
- l'appel récursif à `elementDeRangBis` se fait soit sur `petit` soit sur `grand`, son temps d'exécution est donc majoré par $\max(M'(|\text{petit}|), M'(|\text{grand}|))$. Utilisons la spécificité du pivot pour préciser cette majoration. On notera $x = 1/2\lfloor n/5 \rfloor$, $y = \lfloor n/10 \rfloor$ et r le reste de n modulo 5 : on remarque selon la valeur de n que $x = y$ ou $x = y + 1/2$ donc $\lfloor x \rfloor = y$.
- * Les trois plus grands éléments d'un paquet de cinq dont le médian est strictement supérieur au pivot sont plus grands que le pivot, il y a exactement y tels paquets donc $|\text{grand}| \geq 3y + 2$ en comptant les deux éléments du paquet dont le médian est le pivot ; ainsi $|\text{petit}| \leq n - 3y - 2 = 10x - 3y + r - 2 \leq 7y + 7$.
- * Symétriquement, on obtient les mêmes résultats si $x = y + 1/2$ et dans le cas où $x = y$, on obtient $|\text{grand}| \leq 7y + 5$.

Le temps d'exécution de l'appel récursif est donc majoré par $M'(7y + 7)$.

On obtient finalement

$$M'(n) \leq l'n + M' \left(\left\lfloor \frac{n}{5} \right\rfloor \right) + M' \left(7 \left\lfloor \frac{n}{10} \right\rfloor + 7 \right).$$

Supposons que pour tout n suffisamment grand, $M'(n) \leq c'n$. L'inégalité ci-dessus se transforme en $M'(n) \leq (l' + \frac{9}{10}c')n + 7 \leq c'n$ pour $c' = 11l'$ et pour $n \geq 70/l'$. On montre que $M'(n) \leq 11l'n$ pour tout $n \geq 70/l'$ par récurrence sur n en ajustant la constante suivant le cas de base : Si k désigne le nombre maximal d'opérations élémentaires pour le cas de base, alors on peut montrer que $M'(n) \leq (11l' + k/770)n$ pour tout $n \geq 70/l'$.

Si on avait fait des paquets de trois pour chercher le pivot, une analyse similaire aurait conduit à l'inégalité $l' + c' + 2c'/n \leq c'$ qui ne permettait pas de conclure. Pour des paquets de 7, on obtiendrait la relation $M'(n) \leq l'n + M'(n/7) + M'(5n/7 + q)$ et on aurait c' de l'ordre de $70l'$ ce qui est une constante largement trop grande pour de petites données.