Université de Liège

INFO0902-1 : Structures des données et algorithmes

Projet 1 : Algorithmes de sélection

Auteurs :
Louis Hogge s192814
Valérian Wislez s200825

Professeur : P. Geurts
Année : 2021-2022

Analyse théorique

1.

Algorithmes	Complexité en temps : pire cas	Complexité en temps : meilleur cas	Complexité en espace : pire cas	Complexité en espace : meilleur cas
SelectByQuicksort	$\Theta(N^2)$	$\Theta(Nlog(N))$	$\Theta(N)$	$\Theta(log(N))$
SelectByHeapsort	$\Theta(Nlog(N))$	$\Theta(Nlog(N))$	$\Theta(1)$	$\Theta(1)$
QuickSelect	$\Theta(N^2)$	$\Theta(N)$	$\Theta(N)$	$\Theta(1)$
MedianOfMedians	$\Theta(N)$	$\Theta(N)$	$\Theta(log(N))$	$\Theta(1)$

TABLE 1 – Complexités en temps et en espace en fonction de N (taille du tableau)

La complexité en temps de QuickSelect dans le pire cas correspond au scénario suivant. Premièrement, si le pivot choisi est toujours le maximum (ou le minimum), le problème de taille N est réduit en un problème de taille N-1 à chaque itération (similairement à QuickSort) constitué d'une part du sous-tableau de taille N-1 et de l'autre du pivot. Deuxièmement, le pire cas est atteint lorsque le pivot ne correspond jamais à l'élément recherché, de sorte qu'on doive trier l'entièreté du tableau avant de retourner la bonne valeur.

On obtient la récurrence suivante (comme dans le cas de QuickSort) :

$$T(N) = T(N-1) + \Theta(N)$$

On peut développer pour obtenir la complexité en temps dans le pire cas de QuickSelect :

$$\begin{aligned} T(N) &= \Theta(N) + \Theta(N-1) + \cdots + \Theta(1) \\ &= \Theta(\frac{N(N+1)}{2}) \\ &= \Theta(N^2) \end{aligned}$$

La complexité en temps de QuickSelect dans le meilleur cas quant à elle correspond au scénario où dès la première itération, on choisit comme pivot l'élément recherché. On retournera alors la bonne valeur après la première itération, qui appelera donc une fois la fonction Partition(), effectuant $\Theta(N)$ opérations. Au final on a donc une complexité en temps qui est $T(N) = \Theta(N)$.

La complexité en espace de QuickSelect dans le pire cas peut être comparée à celle de QuickSort. Suivant notre implémentation, QuickSelect n'est pas récursive terminale car elle effectue encore une opération après la récursion (opération de retour), ce qui indique qu'elle doit mémoriser une valeur à chaque appel. Dans le pire cas, la fonction sera appelée récursivement N fois et donc devra retenir $\Theta(N)$ éléments. On a donc une complexité en espace dans le pire cas de $\Theta(N)$.

La complexité en espace de QuickSelect dans le meilleur cas quant à elle correspond à la complexité en espace d'un seul appel récursif et est donc $\Theta(1)$.

La complexité en espace de MedianOfMedians dans le pire cas est logarithmique car MedianOfMedians assurera d'avoir un pivot assez médian, donc le problème de sélection dans la partie de l'algorithme QuickSelect sera toujours "divisé". Ce qui veut dire que même pour un cas assez défavorable, on aura $\Theta(log(n))$ valeurs à retenir dans la partie QuickSelect.

La complexité en espace de MedianOfMedians dans le meilleur cas correspond au scénario où un seul appel récursif a été nécessaire. De même que pour QuickSelect, on n'a que $\Theta(1)$ valeurs à retenir et donc la complexité en espace dans le meilleur cas de MedianOfMedians est $\Theta(1)$.

2. On repart en partie des hypothèses de QuickSort :

Nombre de comparaisons pour le partitionnement : $n-1$

Probabilité que le pivot soit à la position k : $\frac{1}{n}$

En moyenne, les sous-tableaux seront réduits de moitié.

On a la récurrence suivante :

$$\begin{cases} C_0 = 0 \\ C_n = n - 1 + \sum_{k=1}^{n} \frac{1}{n} C_{k-1} \end{cases}$$

On considère en effet que diminuer le tableau de moitié revient à travailler sur un tableau entre 1 et k-1. On résout la récurrence en commençant par multiplier par n les deux membres :

$$nC_n = n(n-1) + \sum_{k=1}^{n} C_{k-1}$$

On se débarasse des sommes en soustrayant l'équation pour n-1 à celle pour n. Soient :

$$nC_n = n(n-1) + \sum_{k=1}^{n} C_{k-1} \tag{1}$$

$$(n-1)C_{n-1} = (n-1)(n-2) + \sum_{k=1}^{n-1} C_{k-1} \tag{2}$$

En faisant $(1) - (2)$ on a :

$$\begin{aligned} nC_n - (n-1)C_{n-1} &= n(n-1) - (n-1)(n-2) + \sum_{k=1}^{n} C_{k-1} - \sum_{k=1}^{n-1} C_{k-1} \\ nC_n - (n-1)C_{n-1} &= 2(n-1) + C_{n-1} \\ nC_n &= (n-1)C_{n-1} + C_{n-1} + 2(n-1) \\ nC_n &= nC_{n-1} + 2(n-1) \end{aligned}$$

On peut ensuite diviser par n :

$$C_n = C_{n-1} + \frac{2(n-1)}{n}$$

On réalise finalement un téléscopage en remarquant la récurrence et en la traduisant par une somme :

$$\begin{aligned} C_n &= C_{n-1} + \frac{2(n-1)}{n} \\ &= C_{n-2} + \frac{2(n-2)}{n-1} \\ &= C_{n-3} + \frac{2(n-3)}{n-2} \\ &= \cdots \end{aligned}$$

$$\implies C_n = C_0 + 2\sum_{k=1}^{n} \frac{k-1}{k}$$

On a finalement :

$$C_n = \underbrace{C_0}_{0} + 2(\underbrace{\sum_{k=1}^{n} \frac{k}{k}}_{n} - \sum_{k=1}^{n} \frac{1}{k})$$
$$C_n = 2(n - H_n)$$
$$C_n \in \Theta(n)$$

En sachant que la série harmonique $H_n \in \Theta(log(n))$.

Expérimentations

1.

Type de liste	aléatoire			croissante			décroissante		
Taille	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
SelectByQuicksort	0,0019	0,0388	2,9009	0,0011	0,0089	0,1159	0,0013	0,0103	0,1254
SelectByHeapsort	0.0023	0,0268	0,3741	0,0020	0,0239	0,3040	0,0019	0,0235	0,2984
QuickSelect	0,0003	0,0025	0,0249	0,0002	0,0016	0,0146	0,0002	0,0016	0,0155
MedianOfMedians	0,0024	0,0106	0,1264	0,0016	0,8806	0,0828	0,0008	0,0085	0,1173

TABLE 2 – recherche de la médiane (k = N/2)

Type de liste	aléatoire			croissante			décroissante		
Taille	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
SelectByQuicksort	0,0017	0,0386	2,9714	0,0009	0,0086	0,1220	0,0012	0,0106	0,1341
SelectByHeapsort	0,0025	0,0292	0,3938	0,0020	0,0246	0,3182	0,0020	0,0237	0,3111
QuickSelect	0,0002	0,0017	0,0186	0,0001	0,0010	0,0102	0,0002	0,0014	0,0122
MedianOfMedians	0,0011	0,0110	0,1384	0,0093	0,8957	9,0151	0,0011	0,0096	0,1662

TABLE 3 – recherche du 10-ième centile (k = N/10)

2. (a) On remarque dans le cas aléatoire de SelectByQuicksort que l'augmentation du temps est plus que linéaire mais n'est pas non plus quadratique. On se situe donc bien dans les valeurs de complexités théoriques. Pour les cas croissant et décroissant, on observe cependant plutôt une complexité linéaire. Notons que nous avons choisi d'implémenter la variante de l'algorithme de Quicksort utilisant Insertionsort pour les sous-tableaux de longueur inférieure ou égale à 5, Quicksort étant trop lourd pour les tableaux de cette longueur là où Insertionsort les trie en temps constant. Nous avons de plus effectuer le choix du pivot de manière aléatoire, ce qui permet de rendre très improbable la possibilité de se retrouver dans le pire cas de complexité temporelle. Ces ajouts permettent d'améliorer grandement l'efficacité en temps de SelectByQuicksort.

   Concernant HeapSort, on remarque une complexité pratique moins bonne que linéaire mais non quadratique. La complexité théorique en $Nlog(N)$ est donc représentative.

   QuickSelect présente, pour tous les cas (aléatoire, croissant, et décroissant), une complexité pratique plutôt linéaire, ce qui correspond bien à la complexité théorique.

   MedianOfMedians ne semble pas suivre une complexité linéaire comme prévue dans l'analyse théorique, mais plutôt supralinéaire (autant dans le cas aléatoire, que croissant ou décroissant).

   Remarque : La valeur correspondant à un tableau de taille 10^6 *dans le cas d'une liste croissante (k = N/2) est aberrante. Lors des tests, le temps d'exécution du programme dépassait les quinze secondes mais la valeur retournée par le main est celle présentée dans le tableau. Par soucis de rigueur, nous l'avons reportée mais nous sommes conscients du problème soulevé ici.*

   (b) On remarque que dans l'ensemble, QuickSelect est plus performant que HeapSort, lui-même étant plus performant que QuickSort, comme annoncé dans l'analyse théorique. Cependant, MedianOfMedians n'est pas plus performant que QuickSelect, malgré la technologie supérieure employée et la prédiction de l'analyse théorique. Cela est sûrement dû à l'implémentation de la recherche du pivot, qui d'un point de vue pratique prend beaucoup de temps (allocation dynamique de mémoire dans le code).

   (c) Selon les résultats expérimentaux obtenus, on a que dans le cas de QuickSort et HeapSort, une recherche du dixième centile est plus longue que la recherche de la médiane.

De même pour MedianOfMedians. A contrario, QuickSelect semble plus rapide dans la recherche du dixième centile que de la médiane. On peut expliquer que la recherche du dixième centile prend plus de temps par le fait que rechercher une valeur proche du maximum ou du minimum requiert en général de trier un plus grand nombre de fois le tableau. Dans le cas où le pivot, choisi aléatoirement, est entre un extrema et la valeur recherchée, la partie du tableau ignorée ne sera pas importante et donc toute l'opération n'est pas fort satisfaisante. Tandis que chercher une valeur médiane s'opère plus rapidement car on arrive tout de suite à dégager une partie conséquente du tableau (en vérifiant si le pivot est plus grand ou plus petit que la valeur recherchée).