TD 2

Algorithmique

Exercice 1: Max

On veut calculer le nombre moyen d'affectations de l'algorithme naïf sur un tableau de n éléments. Soit $P_{n,k}$ le nombre moyen de permutations ayant k Maximum Provisoire de Gauche à Droite (MPGD).

1. Montrer que $Moy(n) = \sum_{k=1}^{n} k\frac{P_{n,k}}{n!}$.
2. Montrer que le nombre de permutations ayant k MPGD vérifie:
   - $P_{1,1} = 1,\ P_{n,0} = 0$, pour $n \geq 1$
   - $P_{n,k} = (n-1)P_{n-1,k} + P_{n-1,k-1}$, pour $n \geq 2,\ k \geq 1$
3. Pour résoudre cette relation de récurrence, utiliser la série génératrice associée aux $(P_{n,k})_{k\in\mathbb{N}}$ et montrer que si $G_n(z) = \sum_{k\geq 0} P_{n,k}z^k$; alors pour $n \geq 1$, on a

$$G_n(z) = z(z+1)\dots(z+n-1).$$

4. En conclure que $Moy(n) = G'_n(1)/G_n(1)$ et que $Moy(n) = H_n$ le n nombre harmonique, $H_n = 1 + 1/2 + \ldots + 1/n$ de l'ordre de $\Theta(\log n)$.

Exercice 2: Min et Max

1. Écrire un algorithme naïf qui calcule le min et le max et donner sa complexité dans le pire cas.
2. Écrire un algorithme récursif qui calcule le min et le max en utilisant une méthode divide-and-conquer. Donner sa complexité.
3. Écrire un algorithme non récursif qui calcule le min et le max et donner sa complexité.

Exercice 3: Médiane et tri rapide en moyenne

Soit un ensemble de n entiers $S = \{a_1, a_2, \ldots, a_n\}$. Le *médian* est l'élément qui se trouve au milieu quand on les trie. Il y a un problème si n est pair. Le médian est le k-ième plus grand élément de S, où $k = (n+1)/2$. On suppose que les élements sont tous distincts.

1. Montrer comment calculer le médian en $O(n \log n)$.
2. Étant donné l'algorithme `partition` qui choisit un pivot $a_i \in S$ et retourne dans S^- tous les éléments plus petit que a_i et dans S^+ les autres. Montrer comment implémenter cet algorithme pour que son coût soit proportionnel au nombre d'éléments de S.
3. Donner un algorithme pour calculer le médian qui utilise la fonction `partition`.
4. Donner le coût de cet algorithme dans le pire cas.

5. Si le pivot est choisi uniformément au hasard et que l'on vérifie qu'il y a au moins $n/4$ éléments dans S^- et dans S^+, quel est le coût de cette variante en moyenne ?
6. Montrer que le coût du tri rapide est $O(n \log n)$.

Exercice 4: Médiane dans le pire cas
SELECT(k,S)

1. If $|S| < 50$, sort S and return the kth largest element
2. Otherwise
   (a) Divide S into $\lfloor |S|/50 \rfloor$ sequences of 5 elements
   (b) Sort each 5-element sequence
   (c) Let M be the sequence of medians of the 5-element sequence
   (d) m=SELECT($\lceil |M|/2 \rceil, M$)
   (e) Let S_1, S_2, and S_3 be the sequence of the elements in S less than, equal to, and greater than m
   (f) If $|S_1| \geq k$, return SELECT(k,S_1)
   (g) Else if ($|S_1| + |S_2| \geq k$) then return m
   (h) Else return SELECT($k - |S_1| - |S_2|, S_3$)

1. Montrer que le temps nécessaire pour sélectionner le k-ième plus grand élément parmi n élément est
   - $T(n) \leq cn$ pour $n \leq 49$
   - $T(n) \leq T(n/5) + T(3n/4) + cn$ pour $n \geq 50$

   Montrer que les suites S_1 et S_3 sont de taille au plus $3n/4$.
2. Prouver par récurrence que $T(n) \leq 20cn$. En déduire que le coût de l'algorithme ci-dessus est en $O(n)$ dans le pire cas.

Exercice 5: Premier et second maxima
Soit T un tableau de n entiers.

1. Donner un algorithme naïf qui calcule les deux premiers maxima et donner sa complexité.
2. Donner un algorithme non naïf pour la même tâche et donner sa complexité.
3. Montrer qu'il est optimal.

Exercice 6: Double hachage

On a un ensemble N d'éléments d'un univers totalement ordonné U à mettre dans une table T avec $|T| = t$ et $|N| = n$. On a une famille de fonction de hachage universelle, *i.e.* telle que $\Pr_{s \in S}[h_s(x) = h_s(y)] = 1/t$ pour $x \neq y \in U$.

1. Montrer que la famille de fonction de hachage $h_{a,b}(x) = ax + b \mod p$ est universelle.

2. Si on utilise un hachage à un seul niveau, quel est le nombre moyen de collision ? Que peut-on dire si $t = n^2$ et si $t = n$? Quels sont les implications de ces deux cas sur la complexité en temps et en mémoire ?
3. Fredman, Komlós et Szemerédi ont proposé d'utiliser une table T de $O(n)$ entrées qui demande un temps $O(n)$ à construire et des tests d'appartenance en temps contant.
   (a) Choisir $s \in S$ et envoyer N dans T. Pour chaque $i \in T$, soit N_i le sous-ensemble de N envoyé dans i par h_s, et $n_i = |N_i|$. Soit $C = \sum_{i \in T} \binom{n_i}{2}$ le nombre de collisions. Si $C > n$, recommencer avec un s différent, sinon aller à l'étape 2.
   (b) Pour chaque $i \in T$, si $n_i \geq 1$ allouer une table T_i de taille n_i^2 et S_i l'ensemble des indices pour la famille de fonctions de hachage de U dans T_i. Choisir $s_i \in S_i$ et utiliser h_{s_i} pour envoyer S_i vers T_i. Si h_{s_i} est injective dans T_i, c'est un bon choix pour s_i, sinon recommencer avec un autre s_i.

   Montrer que la construction demande un temps $O(n)$, un espace de stockage $O(n)$ et un temps d'accès constant.

Exercice 7: Bloom Filters

On va maintenant définir une structure de données qui permet de stocker n éléments et qui va donner un compromis intéressant entre espace nécessaire et probabilité de faux positif.

Le Bloom filter consiste en un tableau de m bits, $A[0]$ à $A[m-1]$ initialisé à zéro. On a k fonctions de hachage aléatoires et indépendantes $h_1, \ldots, h_k$ à image dans $\{0, \ldots, m-1\}$. On fait l'hypothèse que les fonctions de hachage envoient chaque élément vers un élément aléatoire dans $\{0, \ldots, m-1\}$. Pour placer un élément x dans cette structure, on applique chaque fonction de hachage et on place un 1 dans $A[h_i(x)]$. Ensuite, pour recherche si un élément y appartient à l'ensemble, on calcule tous les $h_i(y)$ et on vérifie s'ils valent tous 1.

Déterminer la probabilité de faux positifs.