

IFT2015 :: hiver 2020

Cours de structures de données, Université de Montréal

JANVIER 28, 2020 JANVIER 28, 2020 *by* CSUROS M

File de priorité: diapos, notes de cours révisées, et exercices


- EXERCICES, NOTES DE COURS

- 2 COMMENTAIRES

□

Il y avait trop d'erreurs dans les notes de cours affichées hier.

File de priorité:  06note-pq.pdf

(<https://ift2015h20code.files.wordpress.com/2020/01/ift2015h20-06note-pq-1-2.pdf>) (notes de cours révisées le 28 janvier)  06prez-pq.pdf

(<https://ift2015h20code.files.wordpress.com/2020/01/ift2015h20-06prez-pq.pdf>) (diapos)

Exercices: file de priorité et tas binaire

pq.1 File de priorité avec tableau trié

Donner une implémentation complète (en pseudocode ou Java) du TA file de priorité avec les opérations `insert`, `deleteMin` et `size` (taille = nombre d'éléments dans la file), en utilisant un tableau trié avec gestion dynamique de capacité. Analyser le temps de calcul des opérations en fonction de la taille.

pa.2 Tas binaire avec sentinelles

On veut maintenir un tas binaire aux indices $1..n$ d'un tableau $H[0..L-1]$. On se sert des cases inutilisées à la gauche et à la droite ($L > n+1$) des éléments du tas en y mettant des sentinelles: $H[0] = -\infty$, $H[n+1] = H[n+2] = \dots = H[L-1] = +\infty$. Ici, $-\infty$ a une priorité inférieure à tout élément et $+\infty$ a une priorité supérieure à tout élément. Montrer une implémentation complète avec des opérations `deleteMin` et `insert` en exploitant les sentinelles. La solution doit inclure aussi l'expansion/réduction dynamique du tableau sous-jacent: éviter le débordement mais assurer $L \leq c \times n$ avec une constante raisonnable (p.e. $c=4$).

Indice. Les conditions d'arrêt dans les boucles sont plus simples avec les sentinelles: il n'est pas nécessaire de tester si l'indice du parent > 0 dans `swim`, ou si l'indice du deuxième enfant est toujours $\leq n$ dans `sink`. Par contre, il faut placer les sentinelles lors d'initialisation, suppression et réallocation.

pq.3 Les tout petits

Donnez un algorithme pour énumérer les éléments avec une priorité inférieure ou égale à un argument k dans un tableau ordonné comme min-tas binaire. L'algorithme doit prendre $O(m)$ temps si le tas contient m tels éléments.

Indice. Considérez l'arbre binaire représentant le tas, et élaborer comment parcourir les éléments demandés par récursion.

pq.4 File de priorité binaire

Implémenter file de priorité quand il y a juste deux valeurs possibles pour la priorité: 0 ou 1. Toute opération doit s'exécuter en $O(1)$.

Indice. Il est possible d'utiliser un seul tableau comme structure sous-jacente.

pq.5 Minimax

On a un tableau $A[0..n-1]$ et un paramètre $0 < d \leq n$. On veut calculer $Q(A, d) = \max_i \min_{0 \leq j < d} A[i+j]$. Donner un algorithme qui calcule $Q(A, d)$ en temps $O(n \log d)$, avec $O(d)$ espace de travail au plus.

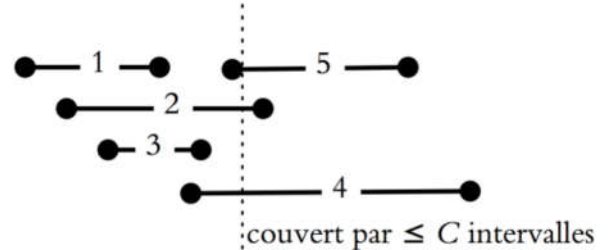
Indice. L'algorithme parcourt A en «glissant» une fenêtre de taille d au long du tableau: $m \leftarrow -\infty$; **for** $i \leftarrow 0, \dots, n-d$: $m \leftarrow \max\{m, M(i, d)\}$ où $M(i, d) = \min\{A[i], A[i+1], A[i+2], \dots, A[i+d-1]\}$ est la valeur minimale dans la fenêtre. Une solution naïve calcule $M(i, d)$ dans une boucle interne qui prend $O(d)$ temps. Une meilleure solution utilise une structure de données qui permet la mise à jour efficace

maintenir une structure de données qui permet la mise à jour efficace $M(i, d) \rightarrow M(i + 1, d)$, en temps $O(\log d)$.

pq.6 Tri d'intervalles

On a une séquence d'intervalles

$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ (stockée dans un tableau $T[0..n-1]$) avec $x_i < y_i$, triées par leur côté gauche: $x_1 < x_2 < x_3 < \dots < x_n$. Les intervalles peuvent être de longueurs différentes. On sait que partout il y a tout au plus C intervalles chevauchants. Dans d'autres mots, pour tout z , il existe tout au plus C intervalles (x_i, y_i) avec $x_i \leq z < y_i$. Donner un algorithme pour trier les intervalles par leur côté droit. L'algorithme doit trier $T[]$ en temps $O(n \log C)$, avec un espace de travail de $O(C)$ au plus.



Remarque. Tri par insertion prendrait $O(n C)$ temps et $O(1)$ espace de travail.

pq.7 ☞ Difficile à comparer

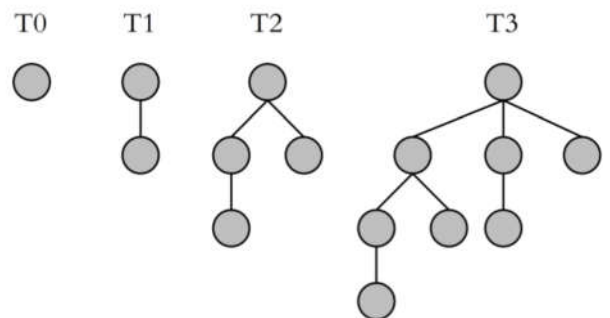
Dans l'implantation usuelle du tas binaire, une insertion nécessite $O(\log n)$ comparaisons entre priorités et $O(\log n)$ affectations (de cases dans le tableau du tas). Montrer comment faire l'insertion avec $O(\log \log n)$ comparaisons et $O(\log n)$ affectations. (Une telle solution est utile quand la comparaison est plus coûteuse que l'affectation —p.e., avec des chaînes de caractères.)

Indice. Considérez les indices visités dans la boucle de swim entre la dernière case et la racine. Identifiez rapidement où la boucle va arrêter.

pq.8 ☞ Arbre binomial

La structure du tas binomial est basée sur une forêt d'arbres binomiaux, chacun dans l'ordre de tas, contenant les données aux noeuds. On définit les **arbres binomiaux** T_0, T_1, T_2, \dots par récursion:

- T_0 contient un seul noeud (sa racine);
- T_k contient un noeud, appelé la racine, avec les enfants $T_{k-1}, T_{k-2}, \dots, T_0$ dans cet ordre de gauche à droit.



a. Dessiner l'arbre T_4 .

b. Démontrer formellement que T_k contient exactement 2^k noeuds. **Indice.** Exploitez la structure récursive des arbres. et procédez par induction.

c. Démontrer formellement qu'il y a exactement $\binom{k}{d}$ noeuds au niveau d dans T_k , où $\binom{k}{d} = \frac{k!}{d!(k-d)!}$ dénote le coefficient binomial. (D'où le nom de ces arbres.) **Indice.** Servez-vous des identités $\binom{k+1}{d+1} = \binom{k}{d} + \binom{k}{d+1} = \sum_{j=d}^k \binom{j}{d}$.

d. Supposons qu'on a deux arbres T_k avec des éléments stockés aux noeuds dans l'ordre de tas. Montrer comment fusionner les deux arbres pour obtenir T_{k+1} , en préservant l'ordre de tas, et sans créer de nouveaux noeuds. **Indice.** Déterminez les relations récursives entre T_k et T_{k+1} .

Remarque. En exploitant la propriété en **b.**, le tas binomial place n éléments dans un ensemble d'arbres T_i où i est les indices des bits '1' dans la représentation binaire de n . Par exemple, pour $n = 21$ on utilise les arbres T_4 , T_2 et T_0 avec $2^4 = 16$, $2^2 = 4$, $2^0 = 1$ noeuds. Afin de fusionner deux tas binomiaux, avec n et m éléments, on suit la logique de l'addition binaire $n + m$ sur les forêts sous-jacentes: si chacun contient un T_k , on les fusionne dans T_{k+1} selon la technique en **d.** (comme addition de bits de poids k). L'insertion d'un élément correspond au cas $m = 1$. Lors de deleteMin, on supprime la racine minimale d'un arbre T_k , et on fusionne les sous-arbres coupés. Toutes opérations prennent alors $O(\log n)$.

Publicités