Examen

CPES2: Algorithmique et Applications

Pierre SENELLART

21 janvier 2021

Les seuls documents autorisés sont deux feuilles A4, recto-verso (4 pages) avec le contenu de votre choix. Cet examen dure deux heures et contient un unique problème, formé de 5 parties. Il est noté sur 20 points.

Tiers-temps : *Les étudiants et étudiantes disposant d'un tiers-temps doivent l'indiquer sur la copie et ne doivent traiter ni la partie D (questions 7 à 9) ni la question 18. Ils et elles seront notés uniquement sur les questions restantes (1 à 6, 10 à 17, 19). Le résultat de la question 18 pourra être admis pour faire la question 19.*

Une *file de priorité* est une structure de données utilisée pour stocker des objets avec une *valeur de priorité* (un nombre réel) et qui supporte les opérations suivantes :

constructeur() Construire une file de priorité vide.

insérer(objet, priorité) Ajouter un objet à la file, avec sa valeur de priorité.

dépiler() Supprimer l'objet dont la valeur de priorité est la plus haute dans la file, et le renvoyer.

augmenter(objet, priorité) Mettre à jour la valeur de priorité d'un objet, avec la condition que la nouvelle valeur de priorité doit être plus grande que l'ancienne.

Le but de cet examen est d'étudier plusieurs manières d'implémenter une telle structure de file de priorité. Dans tout l'examen, on notera n le nombre d'objets actuellement stocké dans une file de priorité, et donc on parlera par exemple d'une opération ayant une complexité en $\Theta(n)$ quand cette complexité est linéaire en le nombre d'objets stockés.

A. Préliminaires (1 point)

1. (0,5 point) On suppose que l'on souhaite réaliser les opérations suivantes avec une file de priorité :
   - On construit une file de priorité vide.
   - On y insère l'élément x avec la priorité 12.
   - On y insère l'élément y avec la priorité 23.
   - On y insère l'élément z avec la priorité 13.
   - On augmente la priorité de x à 35.
   - On dépile et on affiche l'objet correspondant.
   - On dépile et on affiche l'objet correspondant.
   - On dépile et on affiche l'objet correspondant.

   Quel est le résultat de l'exécution de ce programme ?
2. (0,5 point) Proposer une manière d'implémenter augmenter en utilisant directement les autres opérations. Si insérer a une complexité $\Theta(f(n))$ et dépiler une complexité $\Theta(g(n))$, quelle est la complexité d'une telle implémentation de augmenter ?

B. Tableau non trié (2,5 points)

On considère d'abord une implémentation dans laquelle on stocke dans un tableau dynamique des paires (objet, priorité), dans l'ordre où celles-ci ont été insérées dans le tableau.

3. (1,5 point) Écrire sous la forme d'une classe Python une telle implémentation d'une file de priorité (avec son constructeur, et ses méthodes `insérer`, `dépiler`, `augmenter`).
4. (1 point) Quelle est la complexité asymptotique dans le pire des cas des opérations insérer, dépiler, augmenter avec cette implémentation ? Justifier brièvement.

C. Liste chaînée triée par priorité (2 points)

On suppose maintenant qu'on utilise une liste doublement chaînée stockant des paires (objet, priorité) triées par ordre décroissant de priorité.

5. (1 point) Décrire, en français ou en pseudo-code, une implémentation possible des opérations insérer, dépiler et augmenter dans une telle implémentation.
6. (1 point) Quelle est la complexité asymptotique dans le pire des cas des opérations insérer, dépiler, augmenter avec cette implémentation ? Justifier brièvement.

D. Arbre binaire de recherche équilibré (2,5 points)

On suppose maintenant qu'on utilise une structure d'arbre binaire de recherche équilibré (par exemple, un arbre rouge–noir) pour stocker les paires (objet, priorité), la priorité étant la clef utilisée pour comparer des nœuds de l'arbre.

7. (0,5 point) Dans une telle implémentation, où se trouve l'objet de plus haute priorité ?
8. (1 point) Décrire, en français ou en pseudo-code, une implémentation possible des opérations insérer, dépiler et augmenter dans une telle implémentation.
9. (1 point) Quelle est la complexité asymptotique dans le pire des cas des opérations insérer, dépiler, augmenter avec cette implémentation ? Justifier brièvement.

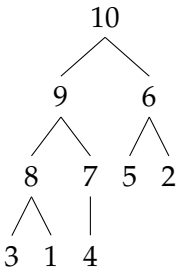
E. Tas binaire (6 points)

Comme vu en cours, un *tas* ou *tas binaire* est l'implémentation d'une file de priorité comme un arbre binaire, stockant les objets avec leur priorité. L'arbre binaire est plein à tous les niveaux, sauf possiblement le dernier (c.-à-d., si l'arbre est de profondeur p, tous les nœuds de profondeur $\leqslant p-2$ ont au plus deux enfants). On impose de plus la condition que la priorité d'un nœud est toujours supérieure ou égale à la priorité de ses descendants (on appelle ceci la *condition de tas de priorité*).

Un arbre binaire de n nœuds est stocké dans un tableau A de taille n de la manière suivante : $A[0]$ stocke le nœud racine ; si un nœud u est stocké en $A[k]$ et a pour enfants u_1 et u_2, alors $A[2k+1]$ stocke le nœud u_1 et $A[2k+2]$ stocke le nœud u_2.

Une opération importante des tas binaires est la procédure tasser qui est utilisée pour corriger une violation élémentaire de la condition de tas de priorité. Elle prend en entrée le tableau A et un index i, de sorte que les sous-arbres enracinés en $2i+1$ et $2i+2$ dans le tableau A sont des tas respectant la condition de tas de priorité, mais avec la priorité du nœud i qui peut être plus petite que celle de ses enfants. Elle renvoie un tableau A dans laquelle le sous-arbre enraciné en i est un tas de priorité correct.

10. (1 point) Considérons le tas représenté par l'arbre binaire suivant :



Construire le tableau correspondant.

11. (1,5 point) Proposer (ou rappeler du cours) un algorithme pour la procédure tasser qui soit en $O(h)$ où h est la hauteur du nœud d'entrée (c.-à-d. la distance de ce nœud à une feuille).
12. (1,5 point) Décrire, en français ou en pseudo-code, une implémentation possible des opérations insérer, dépiler et augmenter en utilisant un tas binaire, qui soient aussi efficace que possible ; on pourra utiliser la procédure tasser quand nécessaire.
13. (1 point) Quelle est la complexité asymptotique dans le pire des cas des opérations insérer, dépiler, augmenter avec cette implémentation ? Justifier brièvement.
14. (1 point) Commenter les différences de complexités avec celles obtenues par les implémentations précédemment proposées. Les tas ont-ils d'autres avantages pour implémenter des files de priorité ?

F. Tas de Fibonacci (6 points)

Un *tas de Fibonacci* est une structure de données complexe utilisée pour implémenter des files de priorité. Formellement, c'est une collection de t arbres (non nécessairement binaires) dont chacun vérifie la *condition de tas de priorité* : la priorité d'un nœud est toujours supérieure ou égale à la priorité de ses descendants. De plus, on mémorise un certain nombre d'informations supplémentaires :

— La collection des arbres est stockée comme une liste doublement chaînée *circulaire* des racines des arbres (une liste circulaire est sans queue ni tête, mais chaque racine pointe vers la suivante et la précédente, formant un cycle de toutes les racines).

— Dans chaque arbre, chaque nœud a un pointeur vers l'un de ses enfants (s'il y en a), un pointeur vers son parent (sauf la racine) et un pointeur vers ses frères gauche et droit (comme une liste doublement chaînée circulaire, de sorte que le frère droit de l'enfant le plus à droite d'un nœud est l'enfant le plus à gauche).

— Chaque nœud d'un arbre a une *marque* supplémentaire qui est une variable booléenne, initialement mise à Faux. La marque est mise à vraie si le nœud *a perdu un enfant depuis la dernière fois qu'il a changé de parent*.

— Une variable spéciale *max* stocke un pointeur vers l'arbre dont le nœud racine a la plus haute valeur de priorité.

— Le nombre t d'arbres et le nombre $\delta(u)$ d'enfants de chaque nœud est également gardé en mémoire.

15. (0,5 point) Proposer (en français ou en pseudo-code) une manière d'implémenter le constructeur d'un tas de Fibonacci vide.
16. (1 point) On considère la *fonction de potentiel* suivante, pour un tas de Fibonacci H : $\phi(H) = \gamma \times (t + 2m)$ où t est le nombre d'arbres dans le tas, m le nombre de nœuds marqués (à Vrai) et

γ une constante positive que l'on définira plus loin. On va utiliser cette fonction de potentiel pour faire une analyse de la complexité amortie du coût des différentes opérations.

a) (0,5 point) Quel est le potentiel du tas de Fibonacci vide ?

b) (0,5 point) Pour une opération x qui transforme un tas de Fibonacci H en un nouveau tas de Fibonacci $x(H)$ avec coût $c_x(H)$, on considère

$$\hat{c}_x(H) \coloneqq c_x(H) + \phi(x(H)) - \phi(H).$$

Montrer que pour toute séquence d'opérations $x_1 \dots x_k$ à partir du tas de Fibonacci vide H_0, avec $H_i \coloneqq x_i(H_{i-1})$:

$$\frac{1}{k}\sum_{i=1}^{k} c_{x_i}(H_{i-1}) \leqslant \frac{1}{k}\sum_{i=1}^{k} \hat{c}_{x_i}(H_{i-1}).$$

On appellera donc $\hat{c}_x(H)$ le *coût amorti* de l'opération x et on l'utilisera pour caractériser la complexité des opérations dans les tas de Fibonacci.

17. (1 point) Proposer une implémentation la plus simple possible de l'opération insérer dont la complexité (directe, pas amortie) est $O(1)$.

18. (2,5 points) Dans les tas de Fibonacci, dépiler fonctionne comme suit : on enlève le nœud pointé par le pointeur *max* et on fait de chacun de ses enfants une nouvelle racine d'un arbre. Ensuite, on modifie la liste circulaire des arbres pour s'assurer que chaque racine r a un nombre d'enfants $\delta(r)$ distinct : pour ce faire, chaque fois que deux racines ont le même nombre d'enfants, l'une devient le fils de l'autre (en respectant la condition de tas de priorité), et on répète jusqu'à ce que toutes les racines ont des degrés distincts. Pendant ces opérations, on met à jour la marque quand c'est nécessaire.

Montrer que, en choisissant une valeur appropriée de γ, la complexité amortie de dépiler dans un tas de Fibonacci H est en : $O\left(\max\left(\max_{u \text{ nœud de } H} \delta(u), \max_{u \text{ nœud de dépiler}(H)} \delta(u)\right)\right)$.

19. (1 point) On admettra les deux points suivants, qui peuvent être démontrés :

— En choisissant bien γ (de manière compatible avec le choix qui a déjà été fait), la complexité asymptotique de augmenter est de $O(1)$.

— Le nombre maximum d'enfants d'un nœud dans un tas de Fibonacci de n nœuds est en $O(\log n)$.

Qu'en conclure sur la complexité de dépiler ? Comment les tas de Fibonacci se comparent-ils aux autres structures de données permettant de représenter des files de priorité ?