UNIVERSITÉ DE MONTRÉAL
Département IRO

Été 2017
Neil Stewart

IFT 2015 E17

Devoir 2.

10/10, soit 10% de la note finale.

Les 10 points "Partie pratique" pour le cours E17 seront distribués de la façon suivante : Devoir #1 (1 point), Devoir #2 (3 points), Devoir #3 (6 points).

1 Partie Pratique (3 points)

Dans cette partie, vous devez implanter un arbre un peu différent de ceux vus en cours, mais d'intérêt, qui s'appelle une *trie* ou *arbre préfixe* (voir : Knuth, Donald, The Art of Computer Programming, Volume 3, Chapitre 6, Section 3, mais faites attention à la définition légèrement différente).

Une trie est une arbre contenant des valeurs associées à des clefs, c'est à dire des tuples (c, V). Les clefs sont formées comme des suites $\sigma_1\sigma_2...\sigma_N$ provenant d'un alphabet Σ, et où N peut varier d'une clef à l'autre. Chaque noeud de la trie peut contenir $|\Sigma|$ enfants étiquettés pour chaque $\sigma \in \Sigma$, et peut ou peut ne pas contenir de valeur. L'accès à une valeur dont la clef est $c = \sigma_1\sigma_2...\sigma_N$ doit se faire comme $root.\sigma_1.\sigma_2.(...).\sigma_N.value$, c'est-à-dire que le chemin menant au noeud contenant la valeur voulue encode directement la clef. Les noeuds ne menant pas à une valeur n'existent pas dans une trie, sauf possiblement la racine. Finalement, nous pouvons définir un ordre total sur les éléments de l'alphabet Σ, ce qui définit également un ordre lexicographique sur les clefs.

Dans le cadre de cette question, nous aurons $\Sigma = \{0, 1\}$ (c'est à dire une *trie binaire* ou *trie digitale*) avec $0 < 1$, mais pour des raisons pratiques nous utiliserons des `boolean` où faux vaut 0 et vrai 1. Donc une clef sera un `boolean[]`, de longueur possiblement égale à 0 (ce qui est la clef de la racine).

À partir du fichier `Trie.java` fourni, vous devez :

1. compléter la méthode `insert`, qui lancera une erreur si on tente d'insérer une valeur à une clef existente,
2. compléter la méthode `find`,
3. compléter la méthode `lexicographic_sort`, qui retourne un ArrayList des valeurs en ordre lexicographique de leurs clefs,
4. compléter la méthode `delete`, qui doit retirer les noeuds rendus superflus par le retrait,

et ce sans modifier la classe interne *Node* ni le constructeur.

Certains indices pour la partie 4 : Vous devez essentiellement parcourir l'arbre pour trouver le noeud dont la valeur est à enlever en conservant trace des noeuds rendus inutiles, mais en garantissant de ne pas rendre une autre valeur encore utile inaccessible. Ci-suivent cinq situations ou des retraits seraient effectués. Vous trouverez un exemple graphique pour chaque cas dans `trie_diagrams.pdf`. Considérez quel devrait être le résultat dans chaque cas, qui couvrent tous les cas que vous pouvez rencontrer :

1. vous retirez la valeur de la clef de longueur 0 et la racine n'a pas d'enfant,
2. vous retirez la valeur d'un noeud sans enfant, qui descend de la racine (qui n'a qu'un enfant elle-même) par une chaîne de noeuds sans valeur avec un seul enfant,
3. vous retirez la valeur d'un noeud sans enfant, qui descend d'un noeud qui a une valeur par une chaîne de noeuds sans valeur avec un seul enfant,
4. vous retirez la valeur d'un noeud sans enfant, qui descend d'un noeud qui a deux enfants par une chaîne de noeuds sans valeur avec un seul enfant,
5. vous retirez la valeur d'un noeud qui a au moins un enfant.

Veuillez remettre votre fichier `Trie.java` complété sur StudiuM avec les noms et matricules des auteurs en commentaire d'entête.

2 Partie Théorique (7 points)

1. ($\frac{1}{2}$ point)
   Considérons le problème simple de chercher une clef dans un tableau linéaire. Disons que si nous trouvons la clef dans la i'ième case, cela prend i comparaisons, $i = 1, \ldots, N$, parce que nous devons passer à travers les premières i cases pour tomber sur la clef. Si nous pouvons mettre les clefs les plus probables plus au début du tableau, le coût moyen de recherche sera inférieur à $N/2$.
   Si la probabilité de la clef i est p_i, écrivez l'expression pour le coût moyen de trouver une clef, en nombre de comparaisons.
   Dans la vraie vie, il arrive assez souvent que les probabilités p_i, en ordre décroissant, sont proportionnelles à $1/i$, $i = 1, \ldots, N$ (loi de Zipf). Dans ce cas :
   (a) Donnez l'expression mathématique pour p_i en utilisant H_N, le nombre harmonique.
   (b) Quel est le coût moyen de trouver une clef dans ce cas ?
2. ($\frac{1}{2}$ point)
   Pour l'insertion dans un arbre AVL
   (a) j'ai dit que "nous revenons sur nos pas" pour ajuster les facteurs de balancement dans l'arbre. C'est très facile à dire : mais comment nous pouvons faire cela, "revenir sur nos pas", dans la pratique ? En utilisant quelle structure de données simple ?
   (b) nous étions très contents de constater que la hauteur de la sous-arborescence ne changeait pas, après la ou les rotation(s), lors de l'insertion. Cela nous a permis d'affirmer qu'il n'est pas nécessaire de continuer plus loin : les facteurs de balancement plus hauts dans l'arbre ne vont pas changer. C'est bizarre quand même : si nous continuons d'ajouter des noeuds dans un arbre, la hauteur doit finir par augmenter quand même. Comment résoudre ce "paradoxe" ?

3. ($\frac{1}{2}$ point)

   Dans la preuve sur la profondeur maximale d'un arbre AVL, j'ai laissé tomber quelques détails simples.

   Le coeur da la preuve était de montrer que

$$G_d > \frac{\phi^{d+3}}{\sqrt{5}} - 2, \;\; d \geq 2, \tag{1}$$

   de sorte que $d < \log_\phi \left(\frac{\sqrt{5}(N+2)}{\phi^3} \right)$, grâce à la monotonicité de la fonction logarithme.

   (a) Pour démontrer (1), il a fallu montrer que $\left| \frac{\hat{\phi}}{\sqrt{5}} \right| < 1, i \geq 5$. Démontrez cela.

   (b) Après avoir démontré (1) j'ai affirmé que cela implique $d < 1.441 \lg N$, $N \geq 3$. Démontrez cela en démontrant que $\log_\phi \left(\frac{\sqrt{5}(N+2)}{\phi^3} \right) \leq \log_\phi(N), N \geq 3$, et en prenant pour acquis que la réciproque du logarithme (base 2) de ϕ est égal à $1.4404\ldots$

4. (1 point)

   Regardez la page 132 de Weiss, et acceptons son idée d'emmagasiner les hauteurs des sous-arbres AVL, plutôt que les facteurs de balancement.

   Weiss, Exercice 4.25, page 163.

   Dans la partie (a), une réponse $O(\ldots)$ suffit. Dans la partie (b), donnez une réponse en interpretant "smallest" dans le sens de "What is the HEIGHT of the smallest AVL tree $\ldots$?", et aussi en interpretant "smallest" comme la plus petite valeur de N. Des valeurs approximatives suffisent.

   Suivons Weiss aussi en disant qu'un octet ne nous donne que 7 bits, parce qu'il travaille avec des "int-octet" dans $[-128, 127)$.

5. (1 point)

   Weiss, Exercice 2.7, page 50 : partie (a) seulement, et seulement les cas (2), (3), (4) et (5).

6. (1 point)

   Ajoutons les "noeuds d'échec" dans un arbre de recherche binaire avec N noeuds : à la place de chaque référence nulle dans les feuilles de l'arbre, mettons un nouveaux noeud (un carrée plutôt qu'un cercle). Apellons ces nouveaux noeuds les "noeuds d'échec". Un tel noeud correspond à une recherche échouée : la clef n'est pas présente.

   Il y a combien de noeuds d'échec dans un arbre avec N noeuds ? Démontrez votre réponse en utilisant un théorème déjà connu (mais exprimez-vous soigneusement).

7. (1 point)

   Weiss, Exercice 5.4, page 218.

   Supposons que nous fassions le réhâchage quand le nombre d'éléments est égal à p fois la taille τ de la table. Weiss entend que, après avoir fait un réhâchage vers une table plus petite, le nombre de retraits ou le nombre d'insertions avant le prochain réhâchage serait le même. Trouvez la valeur de p.

8. ($1\frac{1}{2}$ point)

   Il a été dit dans le cours que si nous prenons M (la taille de la table de hâchage) un nombre premier, alors indépendamment de la valeur de $p(x)$, $p(x) < M$, dans la méthode de hâchage double, nous visiterons toutes les cases de la table en regardant $h(x) - i \cdot p(x) \pmod{M}$, $i = 0, 1, \ldots, M-1$. Démontrez formellement que ceci est vrai.

À réaliser en équipes de 1 ou 2. À remettre le 21 juin, 2017, avant `9:00`. Les solutions seront affichées le 21 juin. Les devoirs en retard ne seront pas acceptés.