UQAR
Université du Québec à Rimouski
300, allée des Ursulines, C. P. 3300, succ. A
Rimouski (Québec) G5L 3A1, CANADA
www.uqar.ca

Solutions

DÉPARTEMENT DE MATHÉMATIQUES, D'INFORMATIQUE ET DE GÉNIE

Structures de données et algorithmes
Examen final — Hiver 2023

SIGLE : INF21307
TITRE : Structures de données & algorithmes
GROUPE : MS
PROFESSEUR : Steven Pigeon
K-212
steven_pigeon@uqar.ca
DURÉE : **3h00**

➜ Nom : _____
Code permanent : _____

Question	Points	Bonus	Obtenus
1	10	0	
2	10	0	
3	10	0	
4	0	5	
Total:	30	5	

Modalités : Toutes les notes (données en classe, manuscrites) sont permises. Aucun appareil électronique n'est permis. La durée de l'examen est 3h00. Toute contravention aux directives expose l'étudiant aux mesures disciplinaires prévues au règlement 5, article 15. Répondez à l'encre sur le formulaire (dûment identifié à votre nom).

1. Questions vrai ou faux, à choix multiple et « buffet. » Les questions marquées à choix multiple demandent de ne faire qu'un seul choix, tandis que les questions « buffet » demandent de cocher tout ce qui s'applique. Pour les questions vrai ou faux et à choix multiple, une bonne réponse donne les points, mais une mauvaise donne zéro. Pour les questions « buffet » chaque bon choix donne un point (+1), une erreur enlève un point (−1) mais on arrête à zéro.

   (a) (1 point) (Choix multiple) Plus l'arité d'un arbre est grande, plus efficace est la recherche.

   ☐ vrai ☑ faux la profondeur est moindre, mais on fait plus de comparaisons par noeud.

   (b) (1 point) (Vrai ou faux) Toutes les opérations d'insertion et de suppression sur une liste simplement chaînée s'effectuent en temps constant.

   ☐ vrai ☑ faux (ex: insérer au $i^{ème}$)

   (c) (1 point) (Choix multiple) Ce n'est *pas* une façon de gérer les collisions :

   A. Le sondage linéaire
   B. Le sondage biquadratique
   C. Le chaînage
   D. Le chaînage coagulé (p. 297 !)

   (d) (1 point) (Vrai ou faux) La notation polonaise inversée des expressions arithmétiques se prête plutôt mal à l'évaluation avec une pile.

   ☐ vrai ☑ faux

   (e) (1 point) (Vrai ou faux) Un b-monceau qui habite dans un tableau de longueur n aura une profondeur $O(2^b)$.

   ☐ vrai ☑ faux (profondeur $\log_b n$!)

   (f) (1 point) (Vrai ou faux) Avec une recherche séquentielle, si on déplace les items recherchés (et trouvés) vers le début de la liste, on peut réduire le nombre d'étapes de la recherche.

   ☑ vrai ☐ faux ... en faisant l'hypothèse que certains sont plus souvent cherchés que d'autres...

(g) (1 point) (Vrai ou faux) Une fonction de hachage primaire n'est vraiment utile que si elle distribue uniformément les clefs sur les nombres de 0 à $k-1$ (où k est la taille de la table).

☑ vrai ☐ faux

(h) (1 point) (Vrai ou faux) Toutes les comparaisons s'effectuent en temps constant.

☐ vrai ☑ faux ex: comparaisons de chaînes de caractères!

(i) (1 point) (Vrai ou faux) Dans une arborescence, le coût d'accès aux nœuds est toujours essentiellement nul.

☐ vrai ☑ faux ils peuvent être un digne!

(j) (1 point) (Vrai ou faux) Dans la structure d'ensemble disjoints, il y a autant de sous-ensembles que d'éléments qui sont des racines.

☑ vrai ☐ faux

2. Questions à développement court. Les questions qui suivent demandent une réponse courte (quelques phrases) mais claire.

(a) (2 points) Expliquez pourquoi il pourrait être intéressant de pousser les nœuds effacés d'une liste dans une seconde liste de nœuds prêts à être réutilisés.

S'éviter d'appeler new/delete (malloc/free, ...) car ces opérations ne sont pas gratuites (et même plutôt lentes).

(b) (2 points) Expliquez pourquoi une table de hachage doit demeurer assez faiblement peuplée.

Si la fct de hachage est adéquate, alors les cases sont également susceptibles d'être choisies, et une collision arrivera avec une probabilité qui dépend du α, le facteur de charge : plus petit le α, plus faibles les chances.

(c) (2 points) Expliquez pourquoi c'est toujours une feuille qui est supprimée dans un arbre de recherche.

Quand la clef à supprimer se trouve dans un noeud interne, on échange avec une clef qui se trouve plus bas dans l'arbre. On répète jusqu'à ce that se trouve dans une feuille, que l'on peut supprimer.

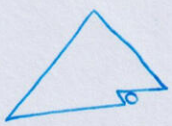
(d) (2 points) Expliquez pourquoi une fonction de hachage doit se comporter essentiellement comme un générateur (pseudo)aléatoire uniforme dont l'amorce est la clef à hacher.

La partie amorce : avec la même amorce, la fct génère la même resultat (utile pour retrouver les choses !)

La partie pseudo aléatoire uniforme : elle doit choisir n'importe quelle case avec une probabilité égale, indépendamment des « tirages » précédents.

(e) (2 points) Expliquez pourquoi un monceau, contrairement à un arbre de recherche binaire, ne peut pas dégénérer en une « liane ».

On s'assure d'ajouter, ou de supprimer, une feuille seulement à la dernière profondeur de l'arbre (et les feuilles sont au plus sur deux niveaux différents)



3. Questions à développement. Les questions qui suivent demandent un développement un peu plus long des idées ; privilégiez une approche « droit au but ».

   (a) (5 points) Si nous proposions de résoudre les collisions dans les tables à adressage dispersé (table de *hash*) par des arbres binaires de recherche plutôt que par des listes, est-ce que le gain serait important ? Si oui, pourquoi (et sinon, pourquoi aussi) ?

Si le facteur de charge, α, est beaucoup plus petit que 1, ça ne change rien; la plupart des cases seront vides, on contiendrait 1 item: Tableau, liste, arbre ... c'est juste 1 item!

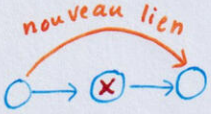
si le facteur de charge, α, commence à être grand (la table est plus-que-pleine), un arbre offrira une recherche $O(\log \alpha)$ (ou $O(\sqrt{\alpha})$!), ce qui est mieux qu'un $O(\alpha)$ avec une recherche séquentielle.

Donc: Il y a un gain si α est grand, sinon, ça n'en vaut pas la peine!

(b) (5 points) La structure d'ensembles disjoints nous garantit que les opérations de fusion d'ensembles et de test d'appartenance sont réalisées très rapidement. Cependant, *retirer* un élément d'un sous-ensemble est beaucoup plus complexe. Expliquez pourquoi retirer un élément est une opération qui s'effectue en temps linéaire, c'est-à-dire $O(n)$ pour un ensemble universel contenant n éléments.

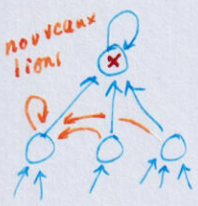
Si l'élément à supprimer est dans une chaîne (ou il est la racine) de son ensemble, il faut trouver un remplaçant.

Or, la relation « est dans le même ensemble que » est unidirectionnelle : la seule façon de trouver qui pointe sur un élément, c'est de les examiner un par un ! (donc c'est $O(n)$).



Si l'élément à supprimer n'est pas une racine, on fait pointer les éléments qui pointaient sur l'élément à supprimer sur l'élément pointé par l'élément à supprimer : on fait seulement un « saute-mouton ».

Si l'élément à supprimer est une racine, on choisit (au hasard ?) un des éléments qui pointaient sur l'élément à supprimer pour être la nouvelle racine, et on fait pointer tous les autres qui pointaient sur l'élément à supprimer sur cette nouvelle racine.



4. (5 points (bonus)) Pour énumérer toutes les clefs contenues dans un arbre dans l'ordre habituel (croissant), il suffit de faire une fouille en profondeur de type en-ordre (on visite récursivement le sous-arbre de gauche, puis la racine, puis récursivement le sous-arbre de droite). Sauf que cette procédure toute simple est récursive et il faut utiliser une pile. Or, cette pile sera aussi haute/profonde que le plus long chemin entre une feuille et la racine de l'arbre. Proposez une façon d'énumérer — dans l'ordre — les clefs, mais sans utiliser une pile. Vous pouvez ajouter des pointeurs ou utiliser une autre technique. Détaillez votre solution.

Il y a plusieurs façons:

- « Coudre » l'arborescence en ajoutant une liste qui relie tous les noeuds (p. 212)

- Organiser l'arbre en mémoire comme pour un monceau (ou presque!)

- parcourir l'arborescence « à main gauche » (p. 213).