Projet LSINF1121 - Algorithmique et structures de données

-

Bilan Mission 3

Groupe 26

Laurian Detiffe (6380-12-00)

Sundeep Dhillon (6401-11-00)

Alexis Macq (5910-12-00)

Xavier Pérignon (8025-11-00)

Thibaut Piquard (4634-13-00)

Thomas Wyckmans (3601-12-00)

Année académique 2015-2016

Questions et réponses

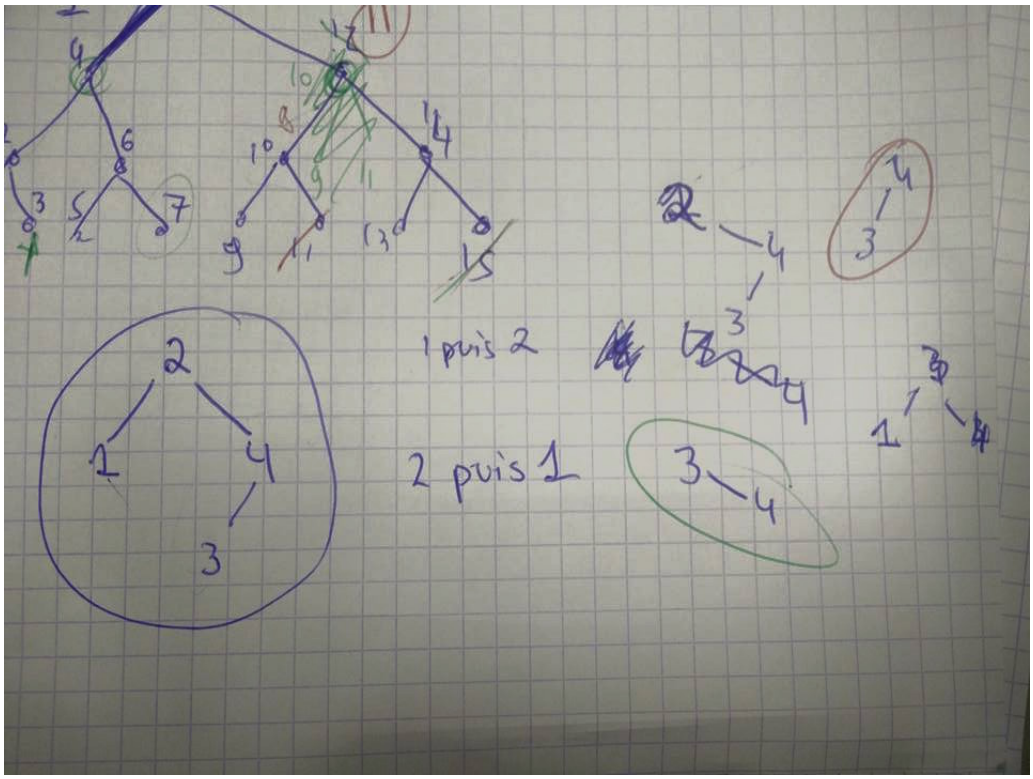
1. Vrai ou Faux (BST s'entend ici comme l'implémentation du livre Algorithms-4)
   (a) Etant donné un parcours infixe d'un BST contenant N clefs distinctes. Est-il possible de reconstruire la forme du BST sur base du résultat du parcourt ?
   FAUX. Car l'arbre n'est pas forcément équilibré.
   (b) Dans le meilleur de cas, le nombre de comparaisons entre clefs pour une recherche binaire d'une clef particulière dans un tableau trié de N clefs distinctes est $\sim$ lgN.
   FAUX. Ca peut être direct et en moyenne ce sera du log(n).
   (c) Etant donné un arbre ordonné de N clefs distinctes et une clef x, est-il possible de trouver la plus petite clef strictement plus grande que x en temps logarithmique dans le pire cas ?
   FAUX. Car l'arbre n'est pas forcément équilibré.
   (d) La hauteur attendue (au sens statistique) d'un BST résultant de l'insertion de N clefs distinctes dans un ordre aléatoire dans un arbre initialement vide est logarithmique.
   VRAI. Vrai seulement dans le sens statistique, on considère que c'est toujours en log mais sinon, c'est faux !
   (e) Soit x un noeud dans un BST. Le successeur de x (le noeud contenant la clef suivante dans l'ordre croissant) est le noeud le plus à gauche dans l'arbre de droite de x.
   FAUX. Dans le cas d'une feuille, c'est faux ! Attention au piège...

2. Vrai ou Faux (Les 2-3/red-black BST sont ceux du livre Algorithms-4 qui diffère de l'implémentation classique qui correspond à un arbre 2/4).
   (a) La hauteur maximum d'un arbre 2-3 avec N clés est $\sim log_3(N)$.
   FAUX. Elle est en $log_2(n)$.
   (b) L'insertion de N clefs dans l'ordre croissant dans un red-black BST initialement vide résulte en un nombre de changement de couleur d'au plus N.
   VRAI.
   (c) Un red-black BST obtenu après insertion de N $>$ 1 clefs dans un arbre initialement vide possède au moins un lien rouge.
   FAUX.
   (d) Dans un red-black BST de N noeuds, la hauteur noire (i.e. le nombre de liens noirs de chaque chemin depuis la racine vers un lien null) est maximum lgN.
   VRAI.

(e) Un red-black tree classique est un BST dans lequel les liens sont colorés soit rouge soit noir de sorte que tout les chemins de la racine vers un lien null a le même nombre de liens noirs (équilibre noir parfait) et dans lequel il n'y a jamais plus d'un lien rouge consécutif le long d'un chemin (mais ceux-ci peuvent pencher à gauche ou à droite et les deux liens d'un noeud peuvent être rouges). Dans un red-black BST classique, la hauteur est entre $\sim$ lgN et $\sim$ 2lgN.
VRAI. Dans l'implémentation classique expliquée à la question, Vrai.

3. Imaginez un algorithme de tri utilisant un BST. Quelle serait la complexité de votre algorithme si le BST est remplacé par un red-black BST ?
Red Black tree $O(n * log(n) + n)$
Pour un arbre non-équilibré, c'est du n^2 dans le pire des cas. En moyenne, ce sera du nlog(n). Pour le red/black tree, le pire des cas serait en log(n) + n à cause du parcours infixe.

4. Est-ce que l'opération de suppression est "commutative" dans le sens que supprimer x et ensuite directement y d'un BST laisse l'arbre dans même état que si on avait d'abord supprimé y et puis x ? Donnez un contre exemple ou argumentez pourquoi c'est effectivement toujours le cas.
Demander les contre-exemples faits au cours, cf feuille



5. Arbres doublement cousus. Imaginez un algorithme non récursif de parcourt infixe d'un BST qui n'utilise pas de stack et qui laisse le BST inchangé. Pour se faire, il faut repenser l'utilisation des liens null dans le BST de cette manière : remplacez un lien null gauche par un pointeur vers son prédécesseur infixe du noeud (s'il existe), et remplacez un lien null droit par son successeur infixe. Mettez à jour put(), deleteMin(), deleteMax() et delete() pour maintenir ces invariants. Hint : vous pourriez avoir besoin d'ajouter deux boolean's dans les noeuds pour indiquer la nature des liens.

```
Si on suit les coutures de droites c'est bon.
       Comment implementer les coutures :
       if(je suis une feuille){
              if(j'ai un noeud a ma droite){
                     ma couture de droite = noeud a ma droite
              }
              else if (je n'ai pas de noeud a ma droite){
                     je remonte jusqu'a ce que je puisse prendre a
                         droite et le premier noeud
                     que je rencontre en allant a droite est ma couture
                         de droite.
              }
              if (j'ai un noeud a ma gauche){
                     ma couture de gauche = noeud a ma gauche
              }
              else if (je n'ai pas de noeud a ma gauche){}
                     je remonte jusqu'a ce que je puisse prendre a
                         gauche et le premier noeud
                     que je rencontre en allant a gauche est ma couture
                         de gauche
              }
       }
       Apres il suffit de suivre les coutures.
```
