UdeM –DIRO- IFT2015H16

TP3 : Arbres Binaires de Recherche

La performance de la fouille dans un arbre binaire de recherche (ABR) est cruciale. Si un ABR est dégénéré alors il n'offre pas d'avantage sur la liste car sa fouille est dans $O(N)$. Un arbre complètement balancé (par exemple un arbre rouge-noir) offre la garantie que la fouille est dans $O(\log(N))$. D'un autre côté, pour éviter qu'un arbre ne dégénère lors de l'insertion/suppression d'éléments, il faut effectuer un certain travail et on peut se demander si ce travail en vaut toujours la chandelle. Dans ce travail vous allez comparer des stratégies de balancement d'ABR et en particulier observer la conséquence de balancer un nombre croissant de nœuds.

Notes

Notre ordonnancement

Un ABR est un arbre ordonné dont chaque nœud a au plus deux enfants. La visite en-ordre d'un tel arbre restitue les étiquettes des nœuds dans l'ordre. Dans ce TP, nous nous intéressons à l'ordre croissant. Considérez le sous-arbre Y (dont la racine est y), le sous-arbre X (dont la racine est x) et le sous-arbre Z (dont la racine est z) où x est l'enfant gauche de y et z est l'enfant droit de y. L'ordonnancement recherché peut être obtenu en imposant que toutes les étiquettes de X soient strictement inférieures à l'étiquette de y et qu'aucune étiquette de Z ne soit inférieure à celle de y. Autrement dit :

$$(eq.1)\ \ etiquette(\text{sous-arbre gauche}) < etiquette(\text{racine}) \leq etiquette(\text{sous arbre droit})$$

Définition (profondeur d'un nœud)

La profondeur d'un nœud est le nombre de liens de la racine à ce nœud.

Travail

Téléchargez le texte http://norvig.com/big.txt. Faites un programme qui mesure le nombre de mots lus et insérés dans chaque type d'arbre (voir ci-bas) en 1 seconde (Pour la stratégie 2, utilisez au moins $p_{max} = 1, 2, 3, 4$). Puis, sur chaque arbre construit, le programme relit la même partie du texte, en choisit tous les $10^{\text{ième}}$ mots et mesure le temps nécessaire pour les supprimer de l'arbre. De plus, chaque arbre résultant est écrit dans un fichier nommé strat_1.out, strat_2-pmax[1,2,3,4].out et strat_3.out.

Pour imprimer l'arbre, adaptez la routine suivante :

```
impression(x)
    Si enfant gauche de x non null
        imprimer( parenthèse ouvrante )
        impression( x->gauche )
        imprimer( parenthèse fermante )
    FIN SI
    imprimer( étiquette de x )
    Si enfant droite de x non null
        imprimer( parenthèse ouvrante )
        impression( x->droite )
        imprimer( parenthèse fermante )
```

FIN SI
FIN IMPRESSION

Pour exécuter du code pendant une durée fixe vous pouvez utiliser l'objet Timer du module thread. La fonction start(*lapse*, *f*) invoque f après *lapse* secondes. Codez donc une fonction *f* qui change l'état d'une variable Booléenne visible depuis votre boucle d'essai. Cette méthode n'est pas très précise alors refaites la boucle quelques fois et rapportez les moyennes et écart-types dans votre rapport.

Stratégie 1 : Aucun balancement.
Encodez l'ABR (section 11.1 Goodrich 2013)

Stratégie 2 : Balancement partiel.
Encodez des arbres dont seuls les nœuds dont la profondeur est inférieure à un seuil ($p_{\max}$) sont balancés.
Augmentez l'implantation de l'ABR afin de maintenir pour chaque nœud, des entiers donnant la taille de ses sous-arbres gauche et droit. Ajustez ces valeurs à chaque insertion/suppression/réarrangement. Notez qu'un nœud balancé possède autant d'éléments dans son sous-arbre droit que dans son sous-arbre gauche (± 1). À intervalles réguliers (à chaque 100 insertions/suppressions), balancez partiellement l'arbre. Étant donné un arbre (ou un sous-arbre) comprenant N nœuds, trouvez le nœud ayant $\left\lceil \frac{N}{2} \right\rceil - 1$ éléments à sa gauche dans la visite *en-ordre*, puis positionnez le à la racine du sous-arbre en réarrangeant ses pointeurs pour lui faire remonter l'arbre. Recommencez avec ses nouveaux sous-arbres si leurs profondeurs sont inférieures à p_{max}. La figure à la fin de ce document montre le processus avec $p_{max} = 1$. Le nœud 8 est déplacé vers la profondeur 0 entre A et B. C montre que le sous-arbre gauche dont la racine est à la profondeur 1 n'a pas besoin de rebalancemement. La racine du sous-arbre droit de profondeur 1 est ajustée entre D et E. L'arbre final partiellement balancé avec $p_{max} = 1$ est montré en F.
Vous devez donc trouver et encoder (a) un algorithme **efficace** pour déplacer les nœuds verticalement jusqu'à leurs destinations et (b) un algorithme **efficace** pour localiser les nœuds à déplacer. Bien sûr il vous faut également actionner ces étapes dans une fonction récursive pour balancer les sous-arbres.

Stratégie 3 : Balancement total.
Encodez l'arbre Rouge-Noir (section 11.6 Goodrich 2013).

Performance :
3 points sont réservés pour la performance de vos algorithmes. La traversée en-ordre n'est pas une approche efficace.

Remise :
Remettez le code et un rapport similaire au rapport du TP2.

Les petits caractères

Ce travail se fait seul ou en équipe de deux.

Avertissement au sujet du plagiat

Tout travail montrant des évidences de plagiat se méritera la note de zéro; les individus dont les travaux sont copiés étant responsables au même titre que les individus qui dupliquent le travail d'autrui. Vous pouvez utiliser des informations obtenues sur Internet (vous ne pouvez pas dupliquer du code obtenu sur Internet), dans un livre ou dans une autre source : citez toutes vos sources! Soyez sérieux, faites votre travail, consultez les démonstrateurs. Si vous vous trouvez dans une discussion avec des collègues et que vous croyez que vous avez échangé suffisamment d'idées pour que vos solutions et analyses se ressemblent, dites-le franchement dans le rapport.

Fichiers à remettre sur Studium

Créez un répertoire dont le nom est matricule-ift2015H16-TP3 (remplacez 'matricule' par votre matricule bien sûr) avec la structure donnée ci-bas. Ensuite, utilisez l'utilitaire zip (ou la commande compress sur un mac) pour créer un fichier d'archive avec l'extension .zip (évitez les .rar et autres variations sur le thème svp). La remise consiste en un seul fichier, l'archive .zip résultante.

Structure du répertoire à remettre :

matricule-ift2015H16-TP3/

- équipiers.txt : ce fichier texte nomme les membres de l'équipe.
- explications.txt : (fichier optionnel) Si vous désirez transmettre de l'information au vous pouvez vous expliquer dans ce fichier texte.
- matricule-ift2015H16-TP3-rapport.pdf : votre rapport.
- code/ : ce répertoire comprend vos fichiers python.

Fonctionnalité

Un code est fonctionnel s'il remplit les spécifications et qu'il ne fait pas autre chose. Un programme qui ne répond pas aux tâches demandées n'est pas fonctionnel bien sûr. Mais un code qui enjolive sa sortie ne l'est pas non plus ! Par exemple : si un programmeur décide d'augmenter son interface d'entrées/sorties il devient impossible d'utiliser son travail dans une chaine d'outils prédéterminée et cela réduit la fonctionnalité de ses programmes. Lorsque vous produisez du code devant remplir des spécifications assurez-vous de remplir exactement les fonctionnalités.

Robustesse

Évidemment plus un programme comprends d'erreurs, moins il est robuste.

De plus, un programme d'ordinateur doit se comporter de manière consistante malgré les variations dans les données qui lui sont fournies en entrée et malgré les conditions variables d'exécution. En particulier lorsqu'une situation incongrue est détectée pendant l'exécution, le programme doit refuser de générer des résultats erronés et aviser l'utilisateur qu'une situation imprévue s'est produite. La politique « crash early, crash hard » est tout à fait bienvenue dans la plupart des cas (sauf évidemment si vous écrivez du code pour piloter un avion.)

Beauté

Un programme d'ordinateur n'est pas qu'une suite d'instructions destinées à être exécutée par un ordinateur ! C'est un objet abstrait destiné à être modifié par des humains. Un programme doit pouvoir être :

- corrigé de ses erreurs (même les programmeurs très expérimentés font des erreurs de codage),
- modifié (le comportement d'un programme devrait pouvoir être ajusté pour de nombreuses raisons n'ayant rien à voir avec le programmeur),
- validé (la relecture critique de code reste une méthode très efficace d'assurance qualité), etc.

Ainsi, un programmeur utile écrit son code avec deux cibles en tête : la machine et l'humain. Parmi les alternatives, on doit favoriser l'usage d'idiomes communs et faciles à comprendre (par

un programmeur compétent) et constamment s'efforcer à produire du code limpide. La marque d'un bon programmeur est la beauté de son code.

Correction négative

Le code non fonctionnel n'est pas recevable et la correction sera très sévère en ce qui concerne les remises partiellement fonctionnelles. La remise de TP ne réalisant pas les spécifications pourrait vous couter jusqu'à la totalité des points.
Vous perdrez des points si vous utilisez un utilitaire produisant une archive qui n'est pas au format demandé ou si la structure de votre répertoire de remise n'est pas conforme aux demandes.

Correction positive

fonctionnalité: 2.33 pts,
performance des algorithmes : 3 pts.
robustesse : 2.33, pts
beauté : 2.33 pts.

Retard

20% par jour (tranches de 24h)

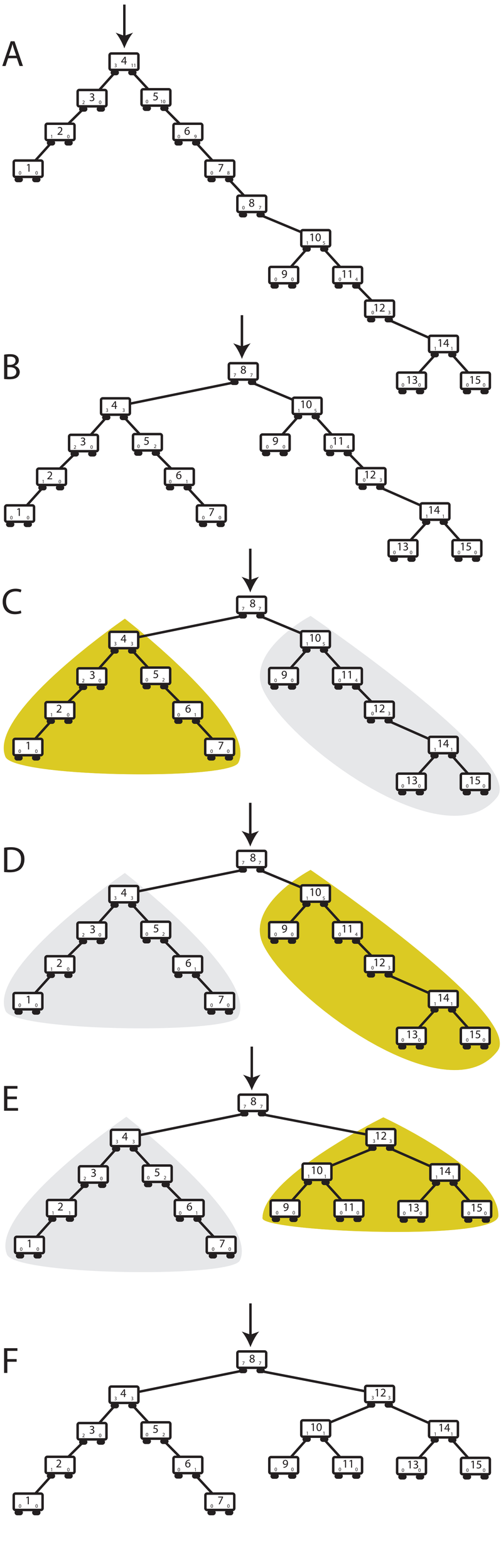
Soupçon de plagiat

zéro, 0, nil, zip, quedalle.

Date de remise

Mardi le 12 avril 23h55.

Bon travail et bonne chance


A
B
C
D
E
F