Projet d'Algorithmique et Programmation

AP3

—Licence Informatique, 3ième année, 1ier semestre—

Ce projet est à réaliser à deux ou trois. Il est à rendre au plus tard le 4 décembre 2023 avant 20h par le dépôt d'une archive au format `zip` à l'emplacement prévu sur UPdago.

Vous pouvez constituer votre groupe comme vous le souhaitez, mais les monômes (une seule personne dans un groupe) sont interdits et **l'inscription dans un groupe sur UPdago est obligatoire** (voir dans l'onglet « Le projet » sur UPdago).

---

1 Arbres binaires de recherche

Les arbres binaires de recherche (ABR) sont très utiles pour implanter des structures de données pour lesquelles les opérations de recherche, d'insertion et de suppression doivent être rapides (i.e. avoir une bonne complexité en temps).

Les ABR sont généralement assez performants lorsque les données insérées sont aléatoires, mais ils se comportent plutôt mal dans le cas contraire. Même s'il facile de voir qu'en insérant une liste ordonnée d'entiers dans un ABR on obtient un arbre filiforme, il vous est proposé d'expérimenter pour vérifier ce point.

1. À partir du module sur les ABR que vous devez réaliser comme TP4 et en utilisant la fonction `Random.self_init` du module[(1)] `Random` qui permet d'initialiser un générateur de nombres aléatoires et la fonction `Random.int borne` qui donne un nombre aléatoire compris entre `0` et `borne - 1` (le paramètre `borne` doit être inférieur à 2^{30}), écrivez une fonction `bst_rnd_create` qui crée un arbre binaire de recherche.
2. Le déséquilibre d'un arbre est la différence entre la hauteur de son fils gauche et la hauteur de son fils droit. À l'aide de la fonction `bst_rnd_create` estimez le déséquilibre moyen des arbres binaires de recherche construits à partir de suites de nombres entiers aléatoires. Pour avoir des estimations significatives, il est nécessaire de répéter un grand nombre de fois l'estimation de la moyenne faite sur un grand nombre d'arbres.
3. Reprenez le même processus d'estimation du déséquilibre moyen d'arbres binaires de recherche, mais cette fois-ci avec des suites de nombres entiers qui contiennent des sous-suites ordonnées de longueurs variables. Les longueurs de ces sous-suites pourront être choisies aléatoirement ou bien de longueur fixe ou encore de longueurs croissantes ou décroissantes.

(1). Pour faire des expérimentations, la fonction `Random.init` peut aussi vous être utile. Il vous est conseillé d'aller consulter la documentation du module `Random` dans le manuel d'OCaml disponible sur `https://v2.ocaml.org/releases/4.14/api/Random.html`

4. Pour chacune des expérimentations précédentes, établissez un compte-rendu d'expérimentation. Vous pouvez illustrer votre compte-rendu avec des graphiques et des courbes (les fonctions qui vous ont été fournies en AP2 peuvent être utiles ou bien un tableur est aussi une option).

2 Arbres AVL

Dans le cours, on a vu que les opérations de recherche, d'insertion et de suppression dans un ABR se font en moyenne en $\Theta(\log n)$ où n est la taille de l'arbre. Dans le cas le pire, quand l'ABR est filiforme, la complexité de ces opérations est en $\Theta(n)$ et on ne gagne rien par rapport à l'utilisation d'une liste. Avec les expérimentations précédentes vous avez sans doute observé que l'on peut assez facilement obtenir des ABR déséquilibrés. Un enjeu important est donc de maintenir l'équilibrage d'un ABR lors des opérations sans pour autant que les opérations de rééquilibrage ne soient trop coûteuses. Historiquement la première classe d'ABR équilibrés a été introduite dans les années 60 par Adelson-Velskii et Landis (deux chercheurs de l'URSS à l'époque, la traduction anglaise de l'article originel, en russe, est sur UPdago). Il s'agit des arbres AVL (d'après les noms de leurs inventeurs).

Depuis les années 60 de nombreuses autres structures de données d'arbres équilibrés ont été inventées. Que ce soit des arbres binaires, arbres Rouge-Noir, arbres AA, arbres à « bouc émissaire », arbres évasés, etc. ou des arbres généraux, $2-3$ arbres, $2-3-4$-arbres, B-arbres (ces derniers sont utilisés pour l'implantation des systèmes de gestion des bases de données), leur importance fait que ces structures sont largement étudiées.

2.1 Implantation d'un module `Avl`

Dans cette partie, il vous est demandé, en utilisant le module `Bst`, d'implanter un module pour les arbres AVL à partir du type abstrait des arbres AVL que l'on a vu en cours.

1. Implantez les rotations à partir des axiomes et des exemples fournis.
2. Implantez l'opération `reequilibrer` à partir des axiomes et en supposant que le déséquilibre d'un arbre est stocké à la racine de cet arbre (proposez une manière de réaliser ce stockage à partir du type `'a t_btree`).
3. Implantez les opérations d'insertion et de suppression dans un arbre AVL.
4. Pouvez-vous réutiliser l'opération de recherche du module `Bst` ? Si ce n'est pas le cas modifiez votre code afin de pouvoir le faire.

2.2 Expérimentations avec les arbres AVL

Vous pouvez illustrer vos réponses aux questions à l'aide de courbes ou de graphiques comme pour la partie précédente.

1. Définissez une fonction `avl_rnd_create` qui crée des arbres AVL à partir de suites d'entiers aléatoires et vérifiez expérimentalement que les opérations de recherche, d'insertion et de suppression ont bien une complexité en $\Theta(\log n)$ où n est la taille de l'arbre.

2. En créant des arbres AVL avec des suites de nombres entiers qui contiennent des sous-suites ordonnées de longueur variable [(2)], estimez le nombre moyen de rotations qui sont effectuées pour garder l'arbre équilibré. Comment ce nombre évolue-t-il en fonction de la taille de l'arbre ?

(2). Les longueurs de ces sous-suites pourront être choisies aléatoirement ou bien de longueur fixe ou encore de longueurs croissantes ou décroissantes.