EA4 – Éléments d'algorithmique
TD n° 8 : arbres binaires de recherche (suite)

Exercice 1 : ABR presque-parfait et opérations ensemblistes

Un arbre binaire est dit *presque-parfait* si tous ses niveaux sont complètement remplis, excepté éventuellement le dernier.

1. Donner un algorithme (efficace !) transformant une liste triée en ABR presque-parfait.
2. En déduire un algorithme qui transforme un ABR en ABR presque-parfait.
3. Quelle est sa complexité ?
4. En utilisant les questions précédentes, proposer un algorithme qui, étant donnés deux ABR (quelconques) A et B, renvoie un nouvel ABR presque-parfait représentant l'union des ensembles représentés par A et par B.
5. Que pensez-vous d'autres opérations ensemblistes binaires (intersection, différence, ...) ?

Exercice 2 : sélection efficace dans les ABR

On s'intéresse ici au problème `selection(E, k)`, consistant à déterminer le k^{e} plus petit élément d'un ensemble de n entiers représenté par une structure `E` sans doublons ($k \in [\![1, n]\!]$).

1. Rappeler rapidement comment résoudre ce problème de manière optimale (au moins en moyenne) dans les cas suivants :
   - **a.** `E` est un tableau trié,
   - **b.** `E` est une liste chaînée triée,
   - **c.** `E` est une liste non triée,
   - **d.** `E` est un ABR.

   Préciser dans chaque cas la complexité en temps (en fonction de n et k) au pire et en moyenne, et justifier l'optimalité.

On souhaite enrichir la structure d'ABR pour effectuer la sélection de manière plus efficace (d'une manière analogue à QuickSelect). Pour cela, on stocke dans chaque nœud, en plus des données usuelles, la `taille` du sous-arbre correspondant (c'est-à-dire le nombre de nœuds du sous-arbre, y compris le nœud lui-même).

2. Réécrire l'algorithme d'insertion `insertionABR(A, x)` pour que le champ `taille` soit correctement tenu à jour. Que devient la complexité ?
3. Si les tailles des deux sous-arbres d'un ABR `A` sont respectivement égales à g et d, où le k^{e} plus petit élément de `A` se trouve-t-il, en fonction de k ?
4. Décrire un algorithme, aussi efficace que possible, répondant au problème de la sélection pour ces ABR enrichis à l'aide du champ `taille`.
5. Analyser la complexité de cet algorithme, au pire et en moyenne.

Exercice 3 : arbres 2-3-4 *(examen 2020-2021)*

Les arbres 2-3-4 sont des cas particuliers de B-arbres, dont les nœuds ont arité 2, 3 ou 4, définis de la manière suivante :

- chaque nœud ou feuille d'un arbre 2-3-4 contient entre 1 et 3 clés ;
- un nœud d'arité $k+1$ contient exactement k clés ;
- ***toutes les feuilles ont la même profondeur.***

La propriété d'ordre des ABR s'étend quant à elle simplement aux nœuds d'arité $k+1$: si un nœud contient les clés $c_0 < c_1 < \cdots < c_{k-1}$ et possède les sous-arbres A_0, A_1... A_k, toutes les clés de A_i sont supérieures à c_{i-1} (si $i > 0$) et inférieures à c_i (si $i < k$).

1. a. Décrire toutes les formes possibles pour un arbre 2-3-4 de hauteur 1. Combien de clés un tel arbre peut-il contenir ?

b. Quelles sont les hauteurs possibles pour un arbre 2-3-4 contenant 15 clés ? Donner un exemple pour chacune (avec comme clés les entiers de 1 à 15).

2. Donner une minoration du nombre de clés à profondeur k, pour $k > 0$. En déduire que la hauteur d'un arbre 2-3-4 contenant n clés est en $\Theta(\log n)$ dans tous les cas.

On suppose toutes les clés distinctes, et on considère que chaque `sommet` contient :
- un champ booléen `feuille` indiquant s'il s'agit d'une feuille ou non,
- un champ entier `taille` compris entre 1 et 3 indiquant le nombre de clés qu'il contient,
- un tableau `cles` de longueur 3, trié, contenant les clés (et `None` dans les cases inutilisées),
- un tableau `fils` de longueur 4 contenant les fils[1], dans l'ordre,

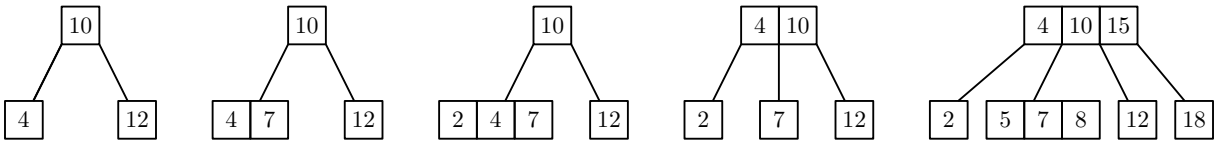
pour lesquels on dispose de tous les accesseurs nécessaires – par exemple, `getTaille(sommet)`, `getCles(sommet)`, `getCle(i, sommet)`...

3. Décrire un algorithme `minimum234(racine)` qui renvoie la plus petite clé d'un arbre 2-3-4 dont `racine` est la racine. Quelle est sa complexité ?

4. Décrire un algorithme `cherche234(c, racine)` *le plus efficace possible* renvoyant le nœud de l'arbre 2-3-4 de racine `racine` contenant la clé `c`, s'il en existe, et `False` sinon. Quelle est sa complexité ?

L'ajout de nouvelles clés est plus complexe, du fait de la contrainte sur la profondeur des feuilles : comme dans un ABR, on cherche à ajouter la clé dans une feuille, mais si celle-ci est déjà *saturée*, c'est-à-dire si elle contient déjà 3 clés, il est nécessaire de l'*éclater* : créer une feuille supplémentaire, nécessairement au même niveau, et faire remonter une clé dans son père – ce qui peut avoir des répercussions sur celui-ci, voire sur toute sa lignée ancestrale ; s'il faut augmenter la hauteur de l'arbre, cela ne peut se faire qu'au niveau de la racine.

5. Insérer la clé 6 dans chacun des arbres ci-dessous.



Vous avez dû constater qu'un éclatement au niveau d'une feuille peut se répercuter sur toute la branche jusqu'à la racine. Il est en fait plus simple de réaliser, lors de la descente, un « éclatement préventif » de chaque nœud saturé rencontré pour s'assurer que de telles répercussions ne se produiront pas.

6. Décrire l'opération d'éclatement d'un nœud, selon qu'il s'agit de la racine ou d'un autre nœud (dont le père a lui-même déjà subi un éclatement si nécessaire).

7. (∗) Écrire un algorithme `insertion234(c, racine)` permettant d'insérer la clé `c` dans l'arbre 2-3-4 de racine `racine`. Quelle est sa complexité ?

Pour la suppression, le problème inverse se pose : celui des nœuds ne contenant qu'une seule clé, qui doivent être *étoffés* en y ajoutant une clé (prise d'un nœud proche). À nouveau, il est plus simple de traiter le problème à la descente plutôt qu'à la remontée, en déplaçant des clés vers les nœuds ayant une seule clé rencontrés lors de la descente.

8. Décrire l'opération d'étoffage d'un nœud, selon qu'il s'agit de la racine ou d'un autre nœud (dont le père a lui-même été étoffé si nécessaire).

9. (∗) Écrire un algorithme `suppression234(c, racine)` permettant de supprimer la clé `c` de l'arbre 2-3-4 de racine `racine`.