TD 3-1. Tableaux dynamiques et analyse amortie

Exercice 1. *Analyse amortie*

1. On considère une pile, avec une opération supplémentaire : MULTIDÉPILE(k, P) dépile les k éléments en haut de la pile et renvoie le $k^{\text{ème}}$. S'il y a moins de k éléments dans la pile, ils sont tous dépilés et le dernier est renvoyé.
   - **i.** Écrire l'algorithme MULTIDÉPILE et analyser sa complexité en nombre d'appels aux opérations de base de la pile.
   - **ii.** On effectue une suite de n opérations EMPILE, DÉPILE et MULTIDÉPILE. Justifier que la complexité amortie par opération est constante.
2. On considère un compteur binaire, qu'on voit comme un tableau de n bits $C_{[0]}, \dots, C_{[n-1]}$, qui représente l'entier $N = \sum_{i=0}^{n-1} C_{[i]} 2^i$. Le compteur possède une unique instruction qui incrémente la valeur de N.
   - **i.** Combien de bits doivent être modifiés, dans le pire cas, lors d'un incrément ? *Donner un exemple qui atteint le pire cas.*

   On incrémente le compteur de 0 à N, et on note $\ell = 1 + \lfloor \log N \rfloor$ le nombre de bits de N.
   - **ii.** Montrer que pour $k \leq \ell$, le $k^{\text{ème}}$ bit est modifié $\lfloor N/2^k \rfloor$ fois au cours des N incréments.
   - **iii.** En déduire que le coût amorti par incrément est $O(1)$.

Exercice 2. *Tableaux dynamiques modifiés*

On rappelle qu'un tableau dynamique $\mathcal{T}$ est représenté par un tableau T de taille N et un nombre d'éléments n qui vérifie à tout instant $n \leq N \leq 4n$. On souhaite utiliser moins de place superflue, et avoir à tout instant $N \leq 2n$.

1. Modifier les algorithmes AJOUT et SUPPRESSION afin de garantir $N \leq 2n$. *On pourra par exemple faire en sorte qu'après redimensionnement, $n \simeq \frac{2}{3}N$.*
2. Adapter l'analyse pour obtenir le coût amorti par opération. *On souhaite une valeur précise en nombre d'affectations, pour pouvoir comparer à la solution initiale.*
3. *(bonus)* On généralise : on souhaite garantir un remplissage minimal αN, et on vise un remplissage βN après redimensionnement. Exprimer la complexité amortie en fonction de α et β, et trouver la valeur optimale de β en fonction de α.

Exercice 3. *Tableau avec recherche efficace*

On souhaite une structure de données pour stocker un ensemble E de n entiers (distincts), avec deux opérations : INSÉRER et RECHERCHER (qui renvoie un booléen).

1. On utilise un tableau dynamique trié.

i. Quel est le coût de RECHERCHER ? *Nommer l'algorithme et donner sa complexité.*

ii. Quel est le coût d'INSÉRER ? *Expliciter l'algorithme.*

2. On veut améliorer la solution précédente. Pour stocker n éléments, on écrit $n = \sum_{i=0}^{k-1} n_i 2^i$ avec $n_i \in \{0, 1\}$) et on on utilise k tableaux $T^0, \ldots, T^{k-1}$, où T^i est de taille 2^i. Chaque tableau est soit plein (si $n_i = 1$) soit vide (si $n_i = 0$). De plus, chaque tableau plein est trié par ordre croissant (mais aucun ordre n'est imposé entre les éléments de tableaux différents).

i. Montrer que le nombre total d'éléments contenu dans les tableaux est bien n et donner une borne sur la place totale utilisée, en fonction de n.

ii. Décrire comment implanter l'opération RECHERCHER, et analyser sa complexité.

iii. Décrire comment implanter l'opération INSÉRER, et analyser sa complexité en pire cas.

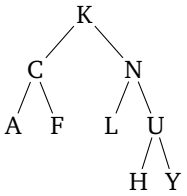
iv. Calculer le coût amorti par insertion lorsqu'on part d'un ensemble vide et qu'on insère n éléments. *On peut réutiliser certains calculs effectués pour le compteur binaire.*

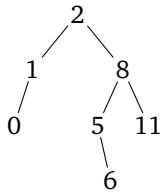
TD 3-2. Arbres binaires de recherche

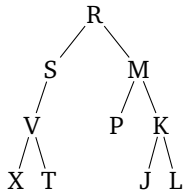
Exercice 1. *Manipuler des ABR*

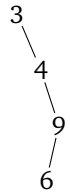
1. Dessiner des ABR de hauteur 6, 3 et 2 dont les nœuds ont le même ensemble de clés : $\{2, 4, 6, 8, 9, 11, 13\}$. Dans chaque cas, donner l'ordre d'insertion qui produit l'arbre dessiné.

2. Parmi les arbres suivants, déterminer ceux qui sont des ABR. *Seules les clés sont représentées.*



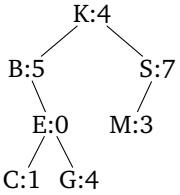






3. Dessiner l'état de l'ABR ci-dessous après chacune des opérations suivantes (appliquées dans l'ordre) :

1. Insérer $(Q, 8)$
2. Supprimer B
3. Insérer $(A, 5)$
4. Supprimer K



4. **i.** Appliquer le parcours infixe sur l'ABR de la question précédente. Que remarque-t-on ?

ii. Énoncer et démontrer le résultat observé.

Exercice 2. *Algorithmes sur les ABR*

1. **i.** Écrire un algorithme qui renvoie la valeur associée à la clé maximale dans un ABR. Quelle est sa complexité ?

ii. Que doit-on changer pour obtenir la valeur associée à la clé minimale ?

2. **i.** Écrire un algorithme SUIVANT qui, étant donné une clé k et un ABR $\mathcal{A}$, renvoie la valeur associée à la plus petite clé de $\mathcal{A}$ qui est supérieure à k. *Si aucune clé n'est supérieure à k, on renvoie une erreur.*

ii. Que doit-on changer pour obtenir PRÉCÉDENT ?

3. Écrire un algorithme qui prend en entrée un ABR $\mathcal{A}$ et un entier k et affiche les valeurs de nœuds de $\mathcal{A}$ dont les clés sont supérieures à k. *Minimiser le nombre de tests effectués.*

Exercice 3. *File de priorité*

✎ Expliquer comment implanter une file de priorité avec un ABR, et indiquer la complexité des opérations.

Exercice 4. *Arbre radix*

Un arbre radix, ou trie, est une autre implantation possible du TAD dictionnaire dans le cas où les clefs sont des mots binaires. C'est un arbre binaire, dont chaque nœud possède une clé de la manière suivante : la racine possède la clé vide (mot de longueur 0) ; si un nœud a la clé c, son enfant gauche (s'il existe) a la clé $c0$ et son enfant droit (s'il existe) la clé $c1$. Chaque nœud est soit vide, soit contient une valeur. S'il est vide, la clé de ce nœud n'est pas dans le dictionnaire. Sinon, la valeur associée à sa clé est la valeur du nœud.

1. **i.** Dessiner l'arbre radix pour le dictionnaire suivant : $\{0 : x, 10 : r, 011 : b, 100 : h, 1011 : p\}$.
   ii. Exprimer la hauteur de l'arbre en fonction des clés contenues dans le dictionnaire.
   iii. Justifier que les feuilles vides sont inutiles.
2. On implante le TAD dictionnaire à l'aide d'un arbre radix. L'opération DICTIONNAIREVIDE est simplement ARBREVIDE des arbres binaires. On suppose qu'il existe une valeur spéciale VIDE pour indiquer qu'un nœud est vide.
   i. Proposer une implantation de RECHERCHER(D, c) et analyser sa complexité.
   ii. Proposer une implantation de INSÉRER(D, c, v) et analyser sa complexité.
   iii. Proposer une implantation de SUPPRIMER(c) et analyser sa complexité. *Attention, l'arbre radix ne doit pas contenir de nœud inutile après suppression.*
3. Montrer comment utiliser cette implantation pour trier un ensemble de mots binaires par ordre lexicographique, en temps $O(N)$ où N est la somme des longueurs des mots de l'ensemble..