Contrôle continu

La durée du contrôle continu est 1h30. Il est constitué de quatre exercices indépendant. Toute question non résolue peut être admise dans la suite. Les réponses doivent être correctement rédigées. Aucun document n'est autorisé.

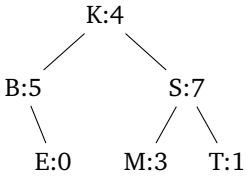
Exercice 1. (sur 5 pts) *Dessins*

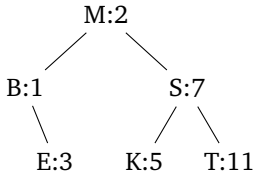
1. Soit $T = [21, 17, 18, 5, 15, 13, 11, 4, 1, 8]$.

i. Dessiner ce tas sous forme d'arbre binaire.

ii. On applique l'algorithme de suppression sur la racine de T. Représenter le résultat sous forme de tableau et sous forme d'arbre binaire.

2. On considère les deux arbres binaires ci-dessous, dans lesquels les clés sont les lettres et les valeurs sont les entiers.





i. Lequel des deux est un arbre binaire de recherche ? *Donner explicitement le problème pour celui qui n'en est pas un.*

ii. On insère les couples $(G, 3)$ puis $(S, 4)$ dans le dictionnaire représenté par l'arbre binaire de recherche. Dessiner l'arbre binaire de recherche obtenu après insertion.

3. Soit h une fonction de hachage définie par $h(k) = ((2k + 7) \bmod 41) \bmod 10$ pour tout $k \in \mathbb{Z}$. On définit une table de hachage $\mathcal{T} = (T, h)$ de taille $m = 10$, dans laquelle les collisions sont résolues par adressage ouvert en essayant les cases d'indices $h(k)$, puis $h(k) + 1 \bmod m$, puis $h(k) + 2 \bmod m$, ...
On insère les clefs $k = 8$, 11, 13, 16 et 24, dans cet ordre, dans la table initialement vide. Donner l'état de la table après les cinq insertions.

Exercice 2. (sur 5 pts) *Questions de cours*

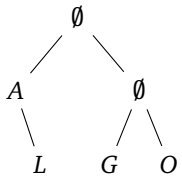
1. Définir, en deux phrases, les modèles universel et idéalisé des fonctions de hachage.

2. Un algorithme parcourt toutes les permutations de $\{0, \dots, n-1\}$ et effectue pour chacune un test dont le coût est $O(n^2)$. Quelle est sa complexité ?

3. On effectue une suite de n opérations sur une structure de données. La $i^{\text{ème}}$ opération coûte i si i est une puissance de 2, et 1 sinon. Quel est le coût amorti par opération ? *On pourra commencer par calculer le coût total.*

Exercice 3. (sur 5 pts) *Arbre radix*

Un arbre radix est un arbre binaire, qui permet d'implanter le TAD dictionnaire dans le cas où les clefs sont des mots binaires. La *clé* d'un nœud est définie de manière récursive : la clé de la racine est le mot vide ; si la clé d'un nœud est le mot c, la clé de son enfant gauche (s'il existe) est $c{\cdot}0$ et celle de son enfant droit (s'il existe) est $c{\cdot}1$. Un nœud peut être vide, ou contenir une valeur v : cela signifie que le dictionnaire contient le couple (c, v) où c est la clé du nœud. Par exemple, l'arbre radix suivant représente le dictionnaire $\{0 : A, 01 : L, 10 : G, 11 : O\}$.



1. **i.** Dessiner l'arbre radix obtenu si on insère les couples $(1, I)$ et $(111, T)$ dans l'arbre ci-dessus.
   ii. Exprimer la hauteur de l'arbre en fonction des clés contenues dans le dictionnaire.
2. Proposer une implantation de l'opération RECHERCHER(D, c) du TAD dictionnaire et analyser sa complexité.
3. On admet qu'on peut effectuer l'opération INSÉRER en complexité $O(\ell)$ où ℓ est la longueur de la clef à insérer. Décrire un algorithme, basé sur un parcours d'arbre radix, 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.

Exercice 4. (sur 5 pts) *Différence maximale*

Soit T un tableau d'entiers, de taille $n > 1$. On souhaite trouver la différence maximale $T_{[j]} - T_{[i]}$ où $i < j$. *Cette valeur est négative si le tableau est décroissant.*

1. Proposer un algorithme de complexité quadratique pour le problème.
2. Proposer un algorithme de type « diviser-pour-régner », qui coupe le tableau en deux, pour le problème. *On suppose avoir accès à deux fonctions* $\min(T)$ *et* $\max(T)$ *qui calculent le minimum et le maximum de* T *en temps* $O(n)$.
3. Démontrer, par récurrence sur n, la correction de votre algorithme.
4. Analyser sa complexité.
5. *(bonus)* Montrer que si au lieu d'utiliser les fonctions min et max, on calcule les minima et maxima nécessaires au cours des appels récursifs, on peut obtenir un algorithme de complexité linéaire.