- Fiche de TD2 : Arbres binaires, tas -

- Arbre binaire -

- Exercice 1 - Opération sur les arbres binaires -

Étant donné un arbre binaire A, donner le pseudo code des fonctions suivante :

a. `Supp`(x) qui teste si le nœud x est une feuille de A qui n'est pas la racine et le supprime de A dans ce cas.

b. `AddFG`(y, x) qui teste si le nœud y a un fils gauche vide et, dans ce cas, ajoute le nœud x à A comme fils gauche de y.

- Tas -

- Exercice 2 - Tas ou pas tas -

Le tableau $[25, 18, 13, 7, 12, 10, 6, 7, 9]$ est-il un tas (c-à-d est-il le tableau des valeurs d'un tas donné selon l'ordre naturel) ?

- Exercice 3 - Où sont les feuilles ? -

Monter que si on utilise la réprentation en tableau pour un tas à n éléments alors les feuilles du tas sont exactement les nœuds ayant pour numéro d'ordre $\lfloor n/2 \rfloor, \lfloor n/2 \rfloor + 1, \ldots, n-1$.

- Exercice 4 - Sous tas -

Soient A un tas et T un sous-arbre binaire de l'arbre quasi-complet formé par A. On note $\text{rac}(T)$ le nœud de T de hauteur minimale dans A. Montrer que pour tout nœud x de T on a $\text{val}(x) \leq \text{val}(\text{rac}(T))$.

- Exercice 5 - Long entassement -

Donner un exemple d'arbre binaire quasi-complet à n nœuds dont l'entassement sur un nœud bien choisi demande $\Omega(\log n)$ appels récursifs (on veut un tel exemple pour n quelconque).

- Exercice 6 - Créer un tas -

a. Exécuter l'algorithme CREER-TAS sur le tableau $[1, 2, 3, 4, 5, 6, 7]$.

b. Montrer que sur tout tableau trié par ordre décroissant, CREER-TAS ne fait aucun entassement.

- Exercice 7 - Implémenter une file de priorité -

Donner le pseudo-code des primitives nécessaires pour dans une file de priorité : obtenir l'élément le plus prioritaire, retirer cet élément, baisser la priorité d'un élément, augmenter cette priorité, insérer un nouvel élément et supprimer un élément. Indiquer pour chacune de ses primitves sa complexité. On pourra appeler des fonctions existantes sur les tas.

- Exercice 8 - Fusion ! -

On dispose de k listes $L_1, \ldots, L_k$ triées en ordre croissant. Pour $i = 1, \ldots, k$ on a accès au dernier élément de la liste L_i par `fin`(L_i) et à sa valeur par `val`(`fin`(L_i)), on peut tester si la liste est vide par `EstVide`(L_i) et on peut supprimer le dernier élément de L_i par `SupFin`(L_i). Toutes ces opérations s'effectuent en temps $O(1)$. Enfin, le nombre total d'éléments dans ces k listes est noté n.
On veut fusionner ces k listes en une seule liste L triée par ordre croissant. Montrer :

a. Comment implémenter cette fusion en un temps $O(n.k)$.

b. Comment améliorer l'implémentation précédente pour avoir une complexité en $O(n. \log k)$ (on pourra utiliser un tas).