Devoir à la maison : files min-max

Consignes.
Ce devoir est à réaliser en binôme. Chaque binôme doit rendre une copie (qui peut être manuscrite ou tapée, au choix). Il y aura ensuite un oral pendant lequel vous serez interrogés (en binôme) sur le travail rendu, pour justifier certaines réponses ou éclaircir des points. L'évaluation prendra en compte le rendu écrit et l'oral.

Exercice.
On définit un TAD *file min-max*, qui est une extension du TAD *file de priorité* : une file min-max contient des éléments x avec chacun une priorité p, et possède toutes les opérations d'une file de priorité, auxquelles on rajoute une opération EXTRAIREMIN qui extrait l'élément de priorité minimale. Formellement, la file min-max possède les cinq opérations suivantes (on renomme EXTRAIRE en EXTRAIREMAX) :

— NVFILEMINMAX() crée une nouvelle file min-max vide ;
— ESTVIDE(F) renvoie VRAI si la file F est vide, FAUX sinon ;
— INSÉRER(F, vvp) insère dans F l'élément v avec priorité p ;
— EXTRAIREMAX(F) supprime et renvoie l'élément de priorité maximale de F ;
— EXTRAIREMIN(F) supprime et renvoie l'élément de priorité minimale de F.

Si on réalise ce TAD avec un tas de taille n, les opérations INSÉRER et EXTRAIRE-MAX sont les mêmes que pour une file de priorité, donc avec la même complexité $O(\log n)$. Mais l'opération EXTRAIREMIN est coûteuse : on sait que l'élément de priorité minimale est nécessairement une feuille du tas, mais cela peut être n'importe laquelle. Il faut donc toutes les parcourir pour le trouver, en temps $O(n)$.
Une réalisation efficace du TAD *file min-max* utilise la structure appelée *tas symétrique* (attention, *ce n'est pas un tas* au sens défini dans le cours !). C'est un arbre binaire quasi-complet dont chaque nœud *sauf la racine* contient un couple (v, p). Pour un nœud x, DESCENDANTS(x) est l'ensemble des nœuds du sous-arbre dont x est racine sauf x lui-même (ce sont donc les nœuds *en dessous de* x). Alors un arbre quasi-complet est un *tas symétrique* si chaque nœud (même la racine) satisfait les deux conditions suivantes :

— si x a un enfant gauche, c'est l'élément de priorité minimale de DESCENDANTS(x) ;
— si x a un enfant droit, c'est l'élément de priorité maximale de DESCENDANTS(x).

La figure @tassym illustre la définition de tas symétrique.
Puisqu'un *tas symétrique* est un arbre quasi-complet, on peut le représenter par un tableau de la même manière qu'un tas. On remarque que dans ce tableau, la

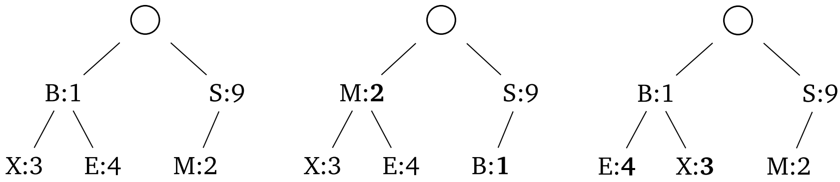

FIGURE 1 – L'arbre de gauche est un tas symétrique ; celui du milieu ne l'est pas car l'élément de priorité minimale de l'arbre n'est pas l'enfant gauche de la racine ; celui de droite non plus car l'élément de priorité minimale *sous* B:1 n'est pas son enfant gauche.

première case d'indice 0 est vide puisque la racine ne contient pas d'élément. Pour réaliser les opérations INSÉRER, EXTRAIREMIN et EXTRAIREMAX, on introduit du vocabulaire. On appelle *voisins* deux nœuds d'un arbre binaire qui ont le même parent. Si un nœud x est à hauteur ≥ 2, x possède un *grand-parent* g, défini comme le parent de son parent. On appelle alors *pseudo-parent gauche de* x l'enfant gauche de g et *pseudo-parent droit de* x l'enfant droit de g. On remarque que l'un des deux pseudo-parents de x est son parent. *Par exemple sur le tas symétrique de la figure @tassym (à gauche), le nœud X:3 est le voisin de gauche du nœud E:4. Le nœud vide ◯ est le grand-parent de E:4, et B:1 et S:9 sont ses pseudo-parents gauche et droit.*

1. On cherche à déterminer des fonctions qui, étant donné l'indice i d'un nœud dans le tableau, calculent l'indice de son grand-parent et de ses pseudo-parents.

- **i.** Si un nœud est en case i, comment déterminer s'il possède un voisin de gauche ? Et quel est son indice dans ce cas ?
- **ii.** Dans quelle case GRAND-PARENT(i) se trouve le grand-parent du nœud en case i ? *Donner l'expression la plus simple possible.*
- **iii.** De même, donner les fonctions PSEUDOG et PSEUDOD qui calculent l'indice des cases des pseudo-parents gauche et droit de i.

L'algorithme ci-dessous réalise l'opération INSÉRER(F, v, p). *On note p_i la priorité du nœud en case i : si $F_{[i]} = (v, p)$, $p_i = p$.*

INSÉRER(F, v, p)

```
0. n ← #F
1. Agrandir F d'une case
2. F[n] ← (v, p)
3. Si n est pair et p_{n−1} > p_n :
4.    échanger F[n] et F[n−1]
5.    n ← n − 1
```

Tant que $n > 2$:
    Si $p_n < p_{\text{PSEUDOG}(n)}$:
        échanger $F_{[n]}$ et $F_{[\text{PSEUDOG}(n)]}$
        $n \leftarrow \text{PSEUDOG}(n)$
    Sinon si $p_n > p_{\text{PSEUDOD}(n)}$:
        échanger $F_{[n]}$ et $F_{[\text{PSEUDOD}(n)]}$
        $n \leftarrow \text{PSEUDOD}(n)$
    Sinon : renvoyer F

2. Quelle est la complexité de l'algorithme INSÉRER ?

On veut montrer la correction de l'algorithme INSÉRER : on suppose que F est un tas symétrique, et on veut montrer qu'INSÉRER(F, v, p) renvoie un tas symétrique. Pour un nœud x, on définit $\text{DESCENDANTS}^\star(x) = \text{DESCENDANTS}(x) \setminus \{(v, p)\}$ (on supprime (v, p) des descendants, s'il en faisait partie). On propose alors l'invariant suivant : à l'entrée de la boucle « Tant que », chaque nœud satisfait la propriété de tas symétrique, éventuellement en remplaçant DESCENDANTS par DESCENDANTS*.

3. **i.** Montrer que l'invariant est satisfait avant la première itération.
ii. Montrer que si l'invariant est satisfait avant la boucle « Tant que », il l'est toujours après la boucle « Tant que ».
iii. Montrer qu'à la sortie de l'algorithme, l'invariant assure que l'arbre est un tas symétrique.
iv. En déduire la correction de l'algorithme.

On cherche maintenant à extraire l'élément de priorité minimale, ou celui de priorité maximale, du tas symétrique. Les deux cas sont similaires : on traite uniquement celui de priorité minimale. L'idée de départ est la même que pour les tas *standard* : on supprime l'élément qu'on souhaite, on le remplace par le dernier élément du tableau, et on fait redescendre cet élément pour qu'il aille à la bonne place.

4. **i.** Écrire l'algorithme EXTRAIREMIN(F). *Cette question est volontairement non guidée : il faut déterminer quel élément extraire, puis comment faire redescendre l'élément qu'on a mis à la place de celui extrait. On demande l'algorithme et une justification qu'il est correct.*
ii. Quelle est la complexité de l'algorithme ?