Devoir à faire pour le Lundi 5 Janvier

Exercice 1 (Mots et pliages)

1. Nous considérons l'alphabet $\Sigma = \{a, b\}$. Nous définissons l'ensemble des mots sur Σ noté Σ^* par :

$$\begin{cases} \varepsilon \in \Sigma^* \ (\varepsilon \text{ désigne le mot vide}), \\ \forall \ell \in \Sigma,\ \forall m \in \Sigma^*,\ \ell m \in \Sigma^* \end{cases}.$$

Autrement dit, un mot non vide est une juxtaposition de lettres.

Nous définissons sur Σ^* l'application $r \mapsto \bar{r}$ par les égalités :

$$\bar{\varepsilon} = \varepsilon \quad \text{et} \quad \forall s \in \Sigma^*, \quad \overline{as} = \bar{s}b \quad \overline{bs} = \bar{s}a.$$

   (a) Calculer $\overline{aababa}$.

   (b) Montrer que pour tout $(r, s) \in (\Sigma^*)^2$, $\overline{rs} = \bar{s}\,\bar{r}$.

   (c) Définir en OCaml les types `alphabet` et `mot` pour représenter respectivement les lettres de Σ et les mots de Σ^*.

   (d) Définir une fonction en OCaml effectuant la transformation $r \mapsto \bar{r}$.

2. Prenons maintenant une feuille de papier, et plions-la n fois dans le sens vertical, en repliant à chaque fois la moitié droite sur la gauche. Une fois remise à plat, les plis de la feuille forment une suite de creux et de bosses.

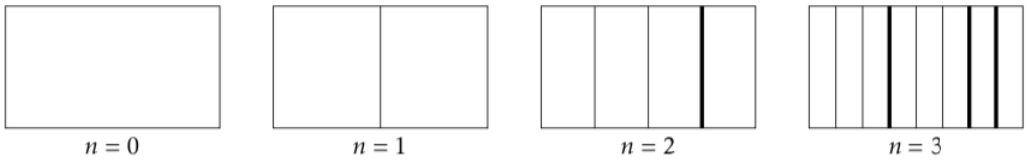


Figure - Un trait fin représente un creux, un trait épais une bosse.

Un pliage de la feuille peut donc être codé par un mot de Σ^* en représentant par a un creux et par b une bosse.

Ainsi, les premiers mots qu'on obtient sont : $\varepsilon, a, aab, aabaabb$.

Nous notons $u_0 = \varepsilon, u_1 = a, u_2 = aab, u_3 = aabaabb, \ldots$ la suite de mots engendrée par les pliages successifs.

   (a) i. Exprimer u_{n+1} en fonction de u_n pour tout entier naturel n. Justifier.

   ii. En déduire une fonction en OCaml construisant les mots de la suite (u_n).

   (b) Les mots de ce langage étant préfixes les uns des autres, on peut considérer le "mot infini" w dont ils sont tous préfixes et nous notons, pour tout entier naturel non nul n, w_n la n-ème lettre de ce mot.

   i. Montrer que, pour tout entier naturel k, $w_{4k+1} = a$ et $w_{4k+3} = b$.

   ii. Montrer que, pour tout $n \in \mathbb{N}$, $w_n = w_{2n}$.

   iii. En déduire une fonction en OCaml calculant w_n.

   iv. Quelle est la 2000-ième lettre de w ?

Exercice 2 (Arbres gauchers)

La structure impérative de tas par des tableaux a été largement étudiée et il est normal de se poser des questions de l'implémentation par structure persistante, c'est-à-dire une structure classique d'arbres binaires. Les opérations principales sont, rappelons-le l'insertion et l'extraction d'un sommet :

- l'insertion peut se faire de deux manières différentes :
  - insertion à la racine puis descente du sommet ;
  - insertion à la dernière feuille puis remontée du sommet ;
- l'extraction de l'élément maximal (pour un tas-max) peut se faire en :
  - supprimant la racine et fusionnant les deux sous-arbres obtenus ;
  - inversant la racine et la dernière feuille puis en descendant le sommet.

Cependant, sans réflexion particulière sur la structure, ces opérations sont toujours réalisées en temps linéaire pour conserver la structure de tas, car il faut savoir dans quelle branche de l'arbre s'effectue la descente ou se situe la dernière feuille.

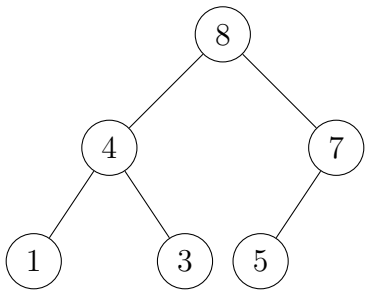
Pour contourner le problème, nous utiliserons un type d'arbre qui ne représente pas structurellement un tas (car il n'est pas parfait), mais qui permet d'implémenter les opérations sur les files de priorités avec les mêmes complexités que la structure impérative de tas.

Définition. Dans un arbre binaire, le rang d'un sommet est la différence de profondeur entre ce sommet et son sous-arbre vide le plus à droite. Le rang r d'un arbre est le rang de sa racine. L'arbre vide est par convention de rang 0.

Définition. Un arbre binaire est dit gaucher s'il est vide ou s'il vérifie les deux conditions suivantes :

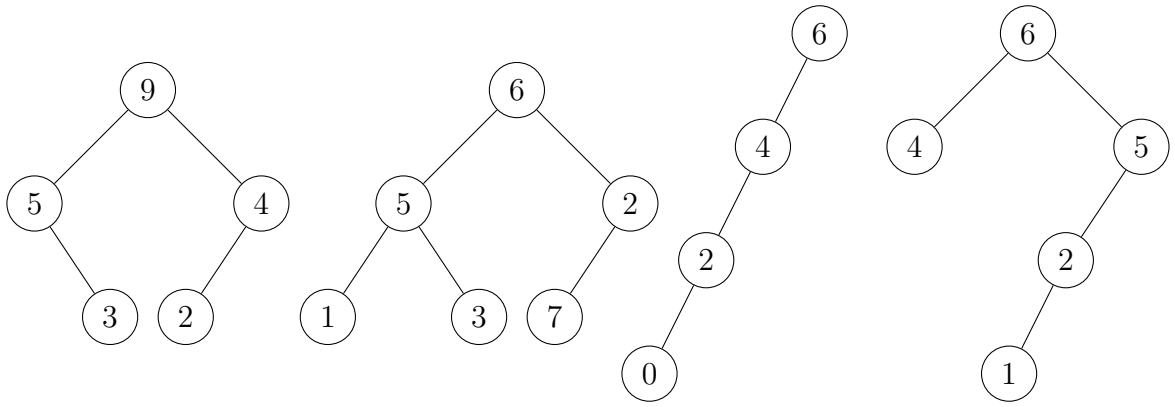
- l'étiquette d'un nœud est toujours supérieure à celles de ses fils ;
- ses deux sous-fils g et d sont gauchers et vérifient $r(g) \geqslant r(d)$.

Exemple. L'arbre binaire suivant est un arbre gaucher :



Les sommets $1, 3, 5$ et 7 sont de rang 1, les sommets 4 et 8 sont de rang 2.

1. Parmi les quatre arbres suivants, déterminer les rangs des différents sommets et déterminer si ces arbres sont gauchers ou non.



2. On considère le type classique des arbres binaires :

   ```
   type 'a arbre = Vide | N of 'a * 'a arbre * 'a arbre ;;
   ```

   Écrire une fonction `rang : 'a arbre -> int` qui détermine le rang d'un arbre.
3. Écrire une fonction `gaucher : 'a arbre -> bool` qui détermine si un arbre est gaucher ou non.
4. Soit a un arbre gaucher. Déterminer et prouver un encadrement optimal de $r(a)$ en fonction de la taille $|a|$ de a.

Pour gagner en complexité temporelle, on code le rang d'un sommet directement dans l'implémentation de l'arbre. On utilise alors le type suivant :

```
type 'a arbre = Vide | N of int * 'a * 'a arbre * 'a arbre ;;
```

L'algorithme de fusion de deux arbres gauchers a et b, pour lesquels la racine de a est supérieure à la racine de b, consiste à fusionner récursivement le sous-arbre droit de a avec b, puis créer un nouvel arbre avec la racine de a, le sous-arbre gauche de a et la fusion effectuée précédemment, en inversant éventuellement ces derniers, pour que l'inégalité sur les rangs soit conservée. Le rang de tous les sommets concernés doit être modifié en conséquence.

5. Réécrire la fonction `rang : 'a arbre -> int` avec ce nouveau type. La complexité de cette fonction doit être en $O(1)$.
6. Écrire une fonction `fusion : 'a arbre -> 'a arbre -> 'a arbre` effectuant l'opération décrite précédemment.
7. En déduire une fonction `insertion : 'a -> 'a arbre -> 'a arbre` qui insère un nouveau sommet dans un arbre gaucher.
8. Écrire une fonction `extraire_max : 'a arbre -> 'a * 'a arbre` qui extrait de l'arbre le sommet d'étiquette maximale et renvoie le couple formé de cette étiquette et de l'arbre restant reformé en arbre gaucher.
9. Déterminer les complexités temporelles des algorithmes de fusion, d'insertion et d'extraction en fonction de la taille des arbres.
10. Écrire une fonction `tri_gaucher : 'a list -> 'a * list` qui trie une liste en utilisant la structure d'arbre gaucher comme intermédiaire.

    La complexité de cette fonction est-elle optimale ?