MP2I – DM pour le 26 février 2024

Le code demandé est à rendre sur cahier de prépa, le reste sur une copie. Tout le code est à faire en `C` et en `OCaml`.
Pour chaque exercice de la programmation :

- Vérifiez que votre code compile.
- S'il y a quelque chose à exécuter, testez-le sur un exemple (au minimum sur l'exemple de la figure 1).

1 Arbres d'arité supérieure

Jusqu'à présent on n'a vu d'un point de vue théorique que des arbres binaires, c'est-à-dire tels que chaque nœud possède au plus deux fils (dans les exemples néanmoins, on a eu l'occasion de travailler avec des structures arborescentes plus générales). On peut bien sûr généraliser une telle structure, en établissant une borne différente sur le nombre de fils de chaque nœud, ou non.

Définition

Le nombre de fils d'un nœud est son *arité*. Un arbre est d'*arité bornée* si l'ensemble des arités de ses nœuds est borné et d'*arité non bornée* si ce n'est pas le cas.
Bien entendu, un arbre fini et figé est d'arité bornée. Mais on a déjà croisé des arbres dont on ne peut pas majorité l'arité, comme le SGF.

Exercice 1 Montrer que la hauteur d'un arbre de taille n et d'arité bornée dont tous les nœuds internes ont au moins deux fils est en $\Omega(\log n)$.

On peut parcourir un arbre d'arité quelconque et on retrouve de façon immédiate les parcours préfixe, postfixe, largeur (infixe n'a pas de signification ici).

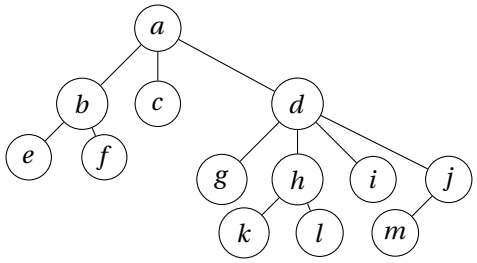
Implémentation Il existe plusieurs implémentations naturelles pour les arbres d'arité quelconque.

Tous les fils dans une même structure

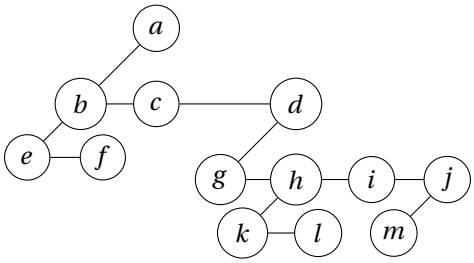
Une implémentation immédiate consiste à représenter tous les fils d'un nœud dans une même structure (tableau ou liste par exemple, voire n-uplet). Nous verrons cette implémentation en TP.

Fils gauche / frère droit

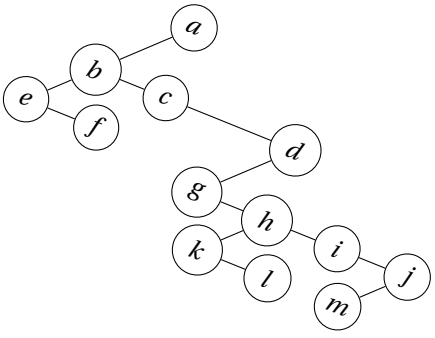
Une autre implémentation d'un arbre d'arité quelconque peut se faire par un arbre binaire : chaque nœud contient la référence de deux autres nœuds, son fils gauche et son frère droit.
Ainsi l'arbre :



est représenté de la manière suivante :



si on tourne un peu la figure précédente, on voit bien un arbre binaire :

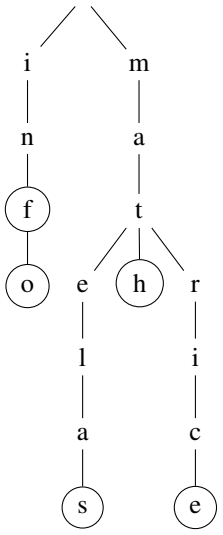


Exercice 2 Montrer que le parcours préfixe de l'arbre d'origine et le parcours préfixe de l'arbre binaire correspondant à la représentation fils gauche / frère droit coïncident.

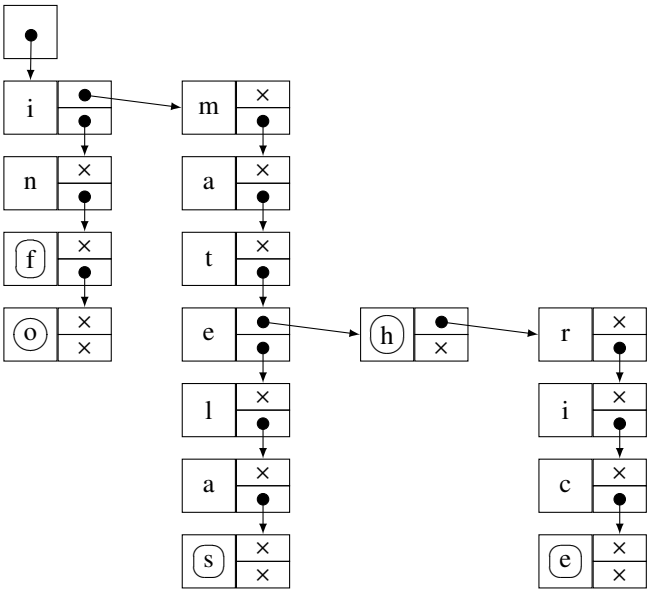
Exercice 3 Montrer que le parcours postfixe de l'arbre d'origine et le parcours infixe de l'arbre binaire correspondant à la représentation fils gauche / frère droit coïncident.

2 Arbres lexicographiques

Un *arbre lexicographique* (*trie* en anglais) est un arbre qui sert à représenter un ensemble fini de mots (un mot étant une suite de finie de lettres). On lit un mot en suivant un chemin simple de la racine à un nœud qui est marqué comme correspondant à la fin d'un mot.
Par exemple, l'ensemble des mots `"inf"`, `"info"`, `"matelas"`, `"math"` et `"matrice"` est représenté par l'arbre lexicographique de la figure 1a.



(a) Arbre lexicographique.



(b) Représentation fils gauche / frère droit.

Figure 1 – Représentations de l'ensemble des mots `"inf"`, `"info"`, `"matelas"`, `"math"` et `"matrice"`.

Pour simplifier, on suppose dans la suite que les seules lettres utilisées sont les lettres latines minuscules non accentuées. On peut alors implémenter l'arbre lexicographique en représentation fils gauche / frère droit, comme représenté en figure 1b.

Exercice 4 Proposer en `C` et en `OCaml` une implémentation de la structure de données qui respecte les contraintes ci-dessus (notamment concernant la représentation fils gauche / frère droit).

Exercice 5 Construire à la main en `C` et en `OCaml` l'arbre lexicographique correspondant au lexique donné en exemple.

Exercice 6 Écrire en `C` et en `OCaml` une fonction qui prend en argument un arbre lexicographique et affiche la liste des mots qu'il contient.
Quelle sont les complexités de vos fonctions (`C` et `OCaml`) par rapport à la taille des données ?
Comme on n'a pas vraiment encore fait les chaînes de caractères, ni en `C`, ni en `OCaml`, on va considérer à la place :

- des tableaux de caractères en `C`, avec le caractère `'\0'` comme sentinelle [1],
- des listes de caractères en `OCaml`.

Exercice 7 Écrire en `C` et en `OCaml` une fonction permettant d'ajouter un mot dans un arbre lexicographique.
Quelle sont les complexités de vos fonctions (`C` et `OCaml`) par rapport à la taille des données ?

Exercice 8 Écrire une fonction permettant de supprimer un mot d'un arbre lexicographique.
Quelle sont les complexités de vos fonctions (`C` et `OCaml`) par rapport à la taille des données ?

1. Bon et pour ne rien vous cacher, c'est ça une chaîne de caractères en `C`.