DM 2 - ABRnois

Dans ce DM, on se propose d'explorer une structure d'arbres intermédiaires entre les arbres binaires de recherches et les arbres tournois, les `ABRnois`s.

Sur internet, les professeurs des écoles dispose d'une ressource pédagogique établie par le linguiste Étienne Brunet. Il s'agit de la liste des 1500 mots environ les plus fréquents de la langue française. Ces mots, extraits de textes littéraires ou non, y ont été ramenés à leur base lexicale.

On se propose de reconstruire cette liste en oubliant la fusion des mots de même base lexicale, qui est le propre du travail du linguiste.

Pour reconstruire notre table des mots les plus fréquents, on souhaite définir une structure de données qui combine à la fois une recherche efficace des éléments stockés (comme dans un arbre binaire de recherche) et qui prenne en compte aussi une notion de priorité (comme dans un tas binaire) afin de supprimer facilement un élément dont la priorité est maximale.

Malheureusement, les notions d'arbre binaire de recherche et d'arbre presque complet sont incompatibles. On affaiblit donc légèrement la notion de tas binaire en ne gardant que la structure d'arbre tournois : un `ABRnois` sera alors un arbre binaire qui est à la fois un arbre binaire de recherche et qui possède aussi une structure d'arbre tournois.

1 Définition d'un arbre de type ABRnois

Plus formellement, considérons deux ensembles ordonnés V et P, V pour les valeurs et P pour des priorités associées à chaque élément. Un `ABRnois` sur $V \times P$ est un arbre binaire dont chaque nœud possède deux informations, une valeur $v \in V$ et une priorité $p \in P$, vérifiant :

- les valeurs sont ordonnées dans l'arbre comme dans un arbre binaire de recherche : pour tout nœud `n`, la valeur v stockée dans `n` est supérieure à chaque valeur stockée dans le sous-arbre gauche de `n`, et est inférieure à chaque valeur stockée dans le sous-arbre droit de `n` ;
- la priorité d'un nœud est supérieure ou égale à la priorité de ses enfants.

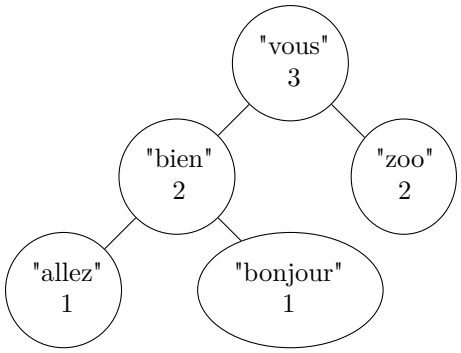
Pour reconstruire la liste des mots les plus fréquents de la langue française, nous utiliserons les structures suivantes :

```c
typedef struct _noeud {
    char * mot;
    int nb_occ;
    struct _noeud * fg, * fd;
} Noeud, * ABRnois;
```

```c
typedef struct _cell {
    Noeud * n;
    struct _cell * suivant;
} Cell, * Liste
```

L'attribut `mot` sera celui sur lequel la structure d'arbre binaire de recherche devra avoir lieu ; l'ordre des valeurs sera donc l'ordre lexicographique. `nb_occ` designera le nombre de fois où `mot` aura été vu dans un corpus et sera considéré comme la priorité sur lequel la structure d'arbre tournois aura lieu.

Pour illustrer la définition, notons $\mathcal{M}$ l'ensemble des mots construits sur l'alphabet classique $\{'a', \cdots, 'z'\}$. Alors, l'arbre suivant est un `ABRnois` sur $\mathcal{M} \times \mathbb{N}^*$:



2 Opérations d'insertion et de suppressions dans un `ABRnois`

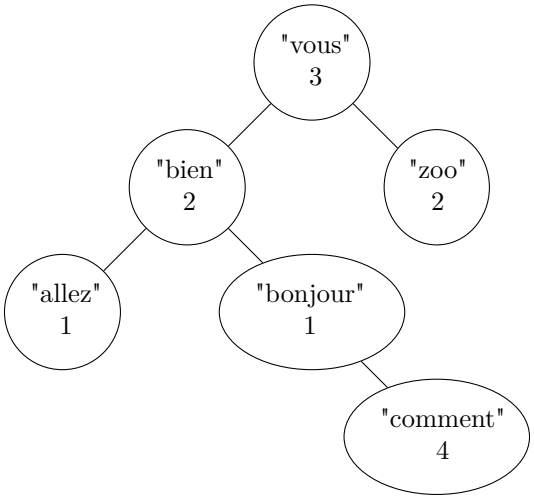
2.1 Insertion dans un `ABRnois`

Supposons que l'on ait un `ABRnois` A et que l'on veuille y insérer la valeur v avec la priorité p. Cette insertion va s'effectuer en deux temps :

1. On commence par insérer la valeur v dans A comme dans un arbre binaire de recherche.
   Si A contient déjà un nœud ayant v comme valeur, sa priorité est augmentée de p.
2. On rétablie dans un second temps la structure d'arbre tournoi sur la priorité en effectuant non pas des échanges comme dans un tas binaire, mais en effectuant des rotations pour remonter le nœud ayant pour valeur v le long de la branche le reliant à la racine.

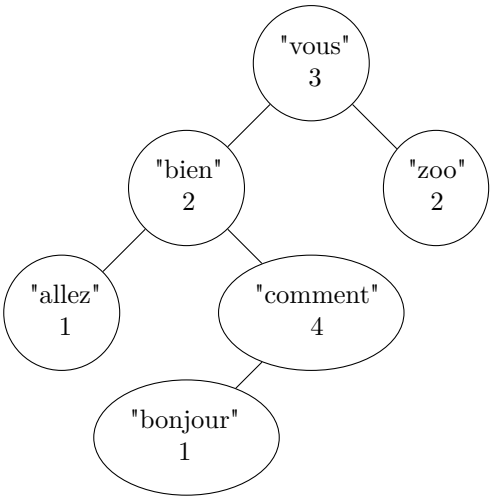
Pour illustrer l'algorithme, insérons le mot `"comment"`, avec la priorité 4 dans le `ABRnois` précédent :

Etape 1 : Insertion comme dans un ABR

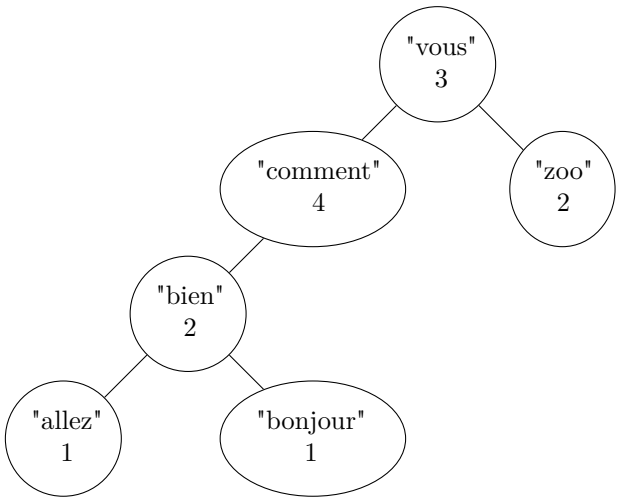


Etape 2 : Remontée du nœud `"comment"` pour rétablir la structure d'arbre tournoi.

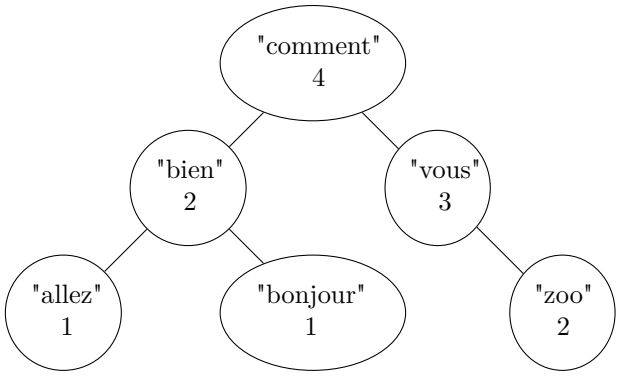
1. Rotation gauche au niveau du nœud `"bonjour"` :



2. Rotation gauche au niveau du nœud `"bien"` :



3. Rotation droite au niveau du nœud `"vous"` :



2.2 Suppression dans un `ABRnois`

Le principe de l'algorithme de suppression dans un `ABRnois` A d'une valeur v est exactement l'inverse de celui de l'algorithme d'insertion.

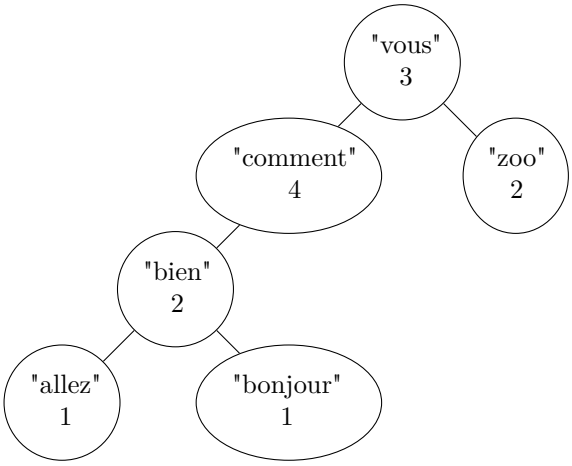
1. On commence par faire descendre le nœud n contenant la valeur v, s'il existe, en effectuant des rotations gauche et droite pour qu'il devienne une feuille de A :
   - ⇝ si le nœud n n'a qu'un seul enfant, il n'y a pas de choix sur la rotation à effectuer.
   - ⇝ si le nœud n a deux enfants g et d pour la gauche et la droite respectivement, il faut remonter celui qui a la plus grande priorité. Autrement dit, si g est prioritaire sur d, on effectuera une rotation droite, sinon ce sera une rotation gauche.

     En cas d'égalité des priorités des deux enfants, les deux rotations sont possibles. Nous privilégierons dans ce devoir une rotation gauche.
2. Une fois devenu une feuille, on supprime alors purement et simplement le nœud n de A.

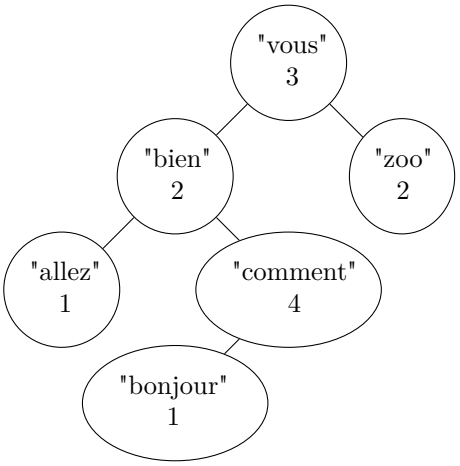
Pour illustrer l'algorithme, supprimons le mot `"comment"` du `ABRnois` obtenu après son insertion.

Etape 1 : Descente du nœud "comment" pour devenir une feuille

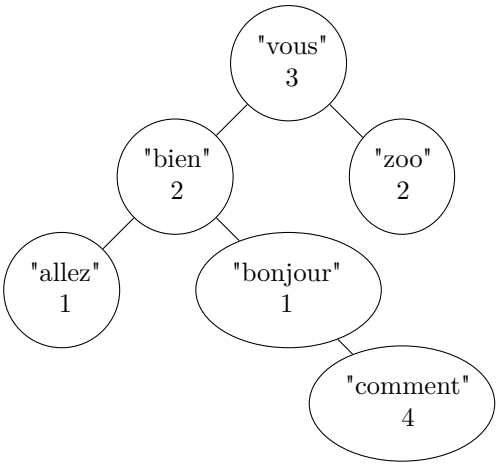
1. La racine possède deux enfants, on va donc remonter celui de priorité maximale, à savoir `"vous"` de priorité 3 en effectuant une rotation gauche sur lui :



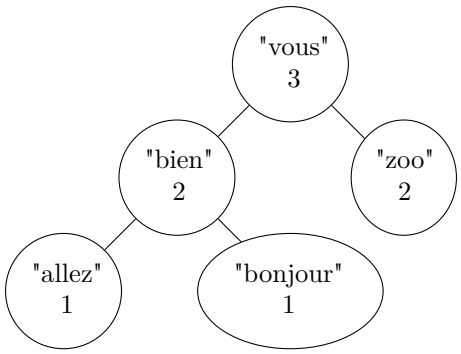
2. Le nœud `"comment"` possède désormais un seul enfant (gauche) ; on va faire une rotation droite dessus :



3. Le nœud `"comment"` possède à nouveau un seul enfant (gauche) ; on va faire une rotation droite dessus :



Etape 2 : Suppression de la feuille contenant la valeur "comment"



On retrouve ici exactement le même `ABRnois` qu'avant l'insertion du mot "comment". Néanmoins, il n'est pas difficile de constater que selon les rotations faite à lors de l'insertion d'une valeur, sa suppression peut produire un arbre différent de celui que l'on avait initialement (par exemple, en insérant "xylophone" dans l'arbre précédent puis en le supprimant immédiatement).

3 Création de la table des mots fréquents à l'aide d'un arbre `ABRnois`

Dans un fichier `abrnois.c`, écrire au moins les fonctions suivantes :

1. `Noeud * alloue_noeud(char * mot)` qui crée sur le tas un nouveau `Noeud` avec une occurence initialisée à 1.
2. `int exporte_arbre(char * nom_pdf, ABRnois A)` qui crée une représentation graphique d'un arbre de type `ABRnois` dans un fichier `pdf`.
3. `void rotation_gauche(ABRnois * A)` (resp. `void rotation_droite(ABRnois * A)`) qui effectue une rotation gauche (resp. droite) sur l'arbre `*A` lorsque celui-ci possède un enfant droit (resp. gauche) non nul.
4. `int insert_ABRnois(ABRnois * A, char * mot)` qui insère dans l'arbre `*A` un nouveau nœud contenant une copie du mot passé en paramètre, avec une occurrence de 1.

   Si le mot est déjà présent dans un nœud n, son occurrence sera augmentée d'une unité.

   Enfin, la fonction renverra 0 en cas d'échec de l'ajout et un entier strictement positif sinon.
5. `int extrait_priorite_max(ABRnois * A, Liste * lst)` extrayant tous les nœuds de l'arbre `*A` ayant dans l'attribut `nb_occ` l'occurrence maximale (c'est-à-dire l'occurence du mot à la racine de l'arbre `*A`) et renvoyant la liste chainée de ces nœud de `*A`.

Les cellules (de type `Noeud`) de la liste `*lst` seront triées par ordre alphabétique dans le cas où plusieurs mot auraient le nombre maximal d'occurrence.

Enfin, la fonction renverra 0 en cas d'échec lors des extraction ; elle renverra le nombre de mots extrait de l'arbre `*A` sinon.

Écrire aussi dans ce fichier un `main` qui prends au moins deux paramètres définis par :

— le nom du premier argument décrits le nom d'un premier fichier texte `frequents.txt`

— les autres arguments sont des autres noms de fichiers texte `corpus_1.txt`, ..., `corpus_n.txt` contenant un texte (littéraire par exemple), .

Ce `main` permettra :

1. de lire mot à mot le contenu de chaque fichier `corpus.txt` et d'ajouter dans un arbre de type `ABRnois` initialement vide les mots qu'ils contiennent un à un avec la fonction `inser_ABRnois` ;
2. de supprimer ensuite de cet arbre les mots les plus fréquents de l'ensemble des fichiers `corpus.txt` jusqu'à obtenir l'arbre vide, puis de les écrire dans le fichier `frequents.txt`, les plus fréquents en premier, suivi d'un espace et de leur pourcentage de fréquence (à 2 chiffres après la virgule) en regard. En cas d'égalité de fréquence, l'ordre alphabétique sera alors respecté.

Ce `main` acceptera aussi deux options :

— `-g` pour créer une représentation sous forme de fichiers `.pdf` des arbres intermédiaires crées au cours du processus de création de la liste des mots les plus fréquents.

— `-n` suivi d'un entier p pour créer uniquement une liste des p premiers mots les plus fréquents (par exemple $p = 1500$ dans le cas du linguiste Étienne Brunet).

Si l'option option `-g` est activée, un fichier `pdf` sera créer après chaque insertion et chaque suppression dans l'arbre de type `ABRnois` dans le répertoire courant. Ceux-ci seront nommés `insertion_1.pdf`, `insertion_2.pdf`, $\cdots$, puis `suppression_1.pdf`, `suppression_2.pdf`, $\cdots$

En cas d'utilisation de l'option `-n`, seuls les p mots les plus fréquents seront supprimés de l'arbre et insérés dans le fichier `frequents.txt`. Si après les p extractions, il reste des mots avec le même nombre d'occurrence que le dernier mot extrait, ceux-ci seront aussi extrait de l'arbre.

Par exemple, le fichier `frequents.txt` obtenu sur l'arbre de l'exemple, après insertion de `"comment"` sera :

```
comment 30.77%
vous 23.08%
bien 15.38%
zoo 15.38%
allez 7.69%
bonjour 7.69%
```

4 Conditions de rendus

Ce devoir est à réaliser en binôme au sein d'un même groupe de TP. Le devoir est à rendre sur le e-learning du cours pour le dimanche 18 mai 2025, 23h59.

Le code sera réalisé dans un seul fichier nommé `abrnois.c`. Les structures et prototypes de fonctions proposées dans le sujet ne doivent pas être modifiées. En cas de besoin, des fonctions intermédiaires peuvent être ajoutées.

Un fichier `.zip` y sera déposé contenant a minima le fichier de code `abrnois.c` que vous avez développé ainsi qu'un fichier `rapport.pdf` décrivant succinctement les fonctions implémentées, les difficultés rencontrées, la

répartition du travail au sein du binôme et votre réponse aux questions de la fin de la partie "Comparaison des méthodes".

L'archive et le répertoire qu'elle contiendra seront nommés selon la nomenclature `nom1_nom2.zip` où `nom1` et `nom2` désigne les noms de famille des membres du binôme. Le rapport contiendra l'intégralité des noms et prénoms des membres du binômes.