Exercices

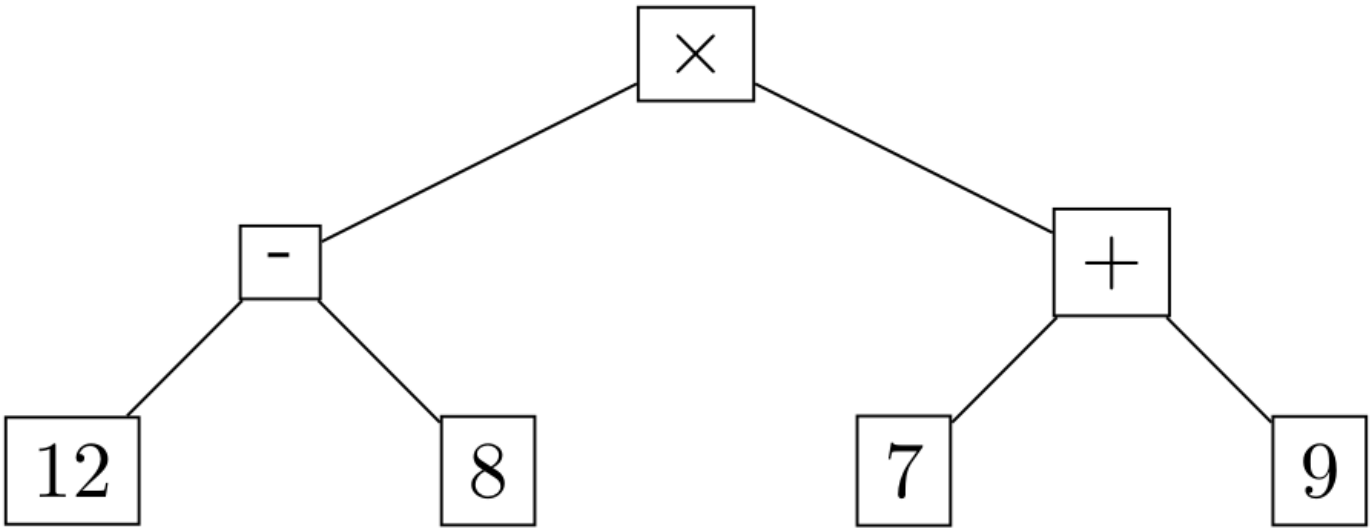

Figure 1: Exercice 1

Exercice 1 * L'arbre binaire ci-dessous représente une opération arithmétique.

1. Parcourir cet arbre en profondeur (préfixe, infixe et postfixe).
2. Donner le résultat du calcul.
3. Pour quel parcours est-il indispensable de rajouter des parenthèses?

Exercice 2 *

1. Effectuer les parcours (largeur, profondeur) sur l'arbre binaire.
2. Quelle est la hauteur de cet arbre. On considère qu'un arbre vide à une hauteur de -1.
3. Cet arbre est-il équilibré?
4. Cet arbre est-il complet?

Exercice 3 **

1. Reprendre la classe **Noeud** construite dans le cours **Algo 07 - notation polonaise**.
2. Construire un arbre équilibré de hauteur 3 et de taille 12.

Pour calculer la taille d'un arbre, il faut:

```
- calculer récursivement la taille du fils gauche,
- calculer récursivement la taille du fils droit,
- ajouter 1 (pour le nœud en cours).
```

3. Dans l'algorithme de calcul de la taille, quel est le cas limite?
4. Écrire la fonction récursive **taille(a: Noeud)** → **int** qui renvoie la taille de l'arbre.
5. Écrire la fonction **maxi(h1: int, h2: int)** → **int** qui renvoie l'entier le plus grand parmi **h1** et **h2**.
6. Écrire la fonction récursive **hauteur(a: Noeud)** → **int** qui renvoie la hauteur de l'arbre binaire. On utilisera la fonction **maxi** pour comparer la taille des fils gauche et droit.
7. Étudier la complexité des fonctions **taille** et **hauteur**.

Exercice 4 ** La France a (brillamment) gagné la coupe du Monde de football 2018 (la coupe du Monde 2022 n'a aucun intérêt).

1. Quel est le nœud racine de cet arbre binaire?
2. Quelle est la taille de l'arbre?
3. Quelle est la hauteur de l'arbre?
4. Télécharger et extraire l'annexe **cdm2018.zip**.
5. Ouvrir le fichier **cdm2018.json** avec Notepad++ et vérifier *à la main* que le tableau représente bien l'arbre binaire des phases finales de la coupe du Monde 2018.

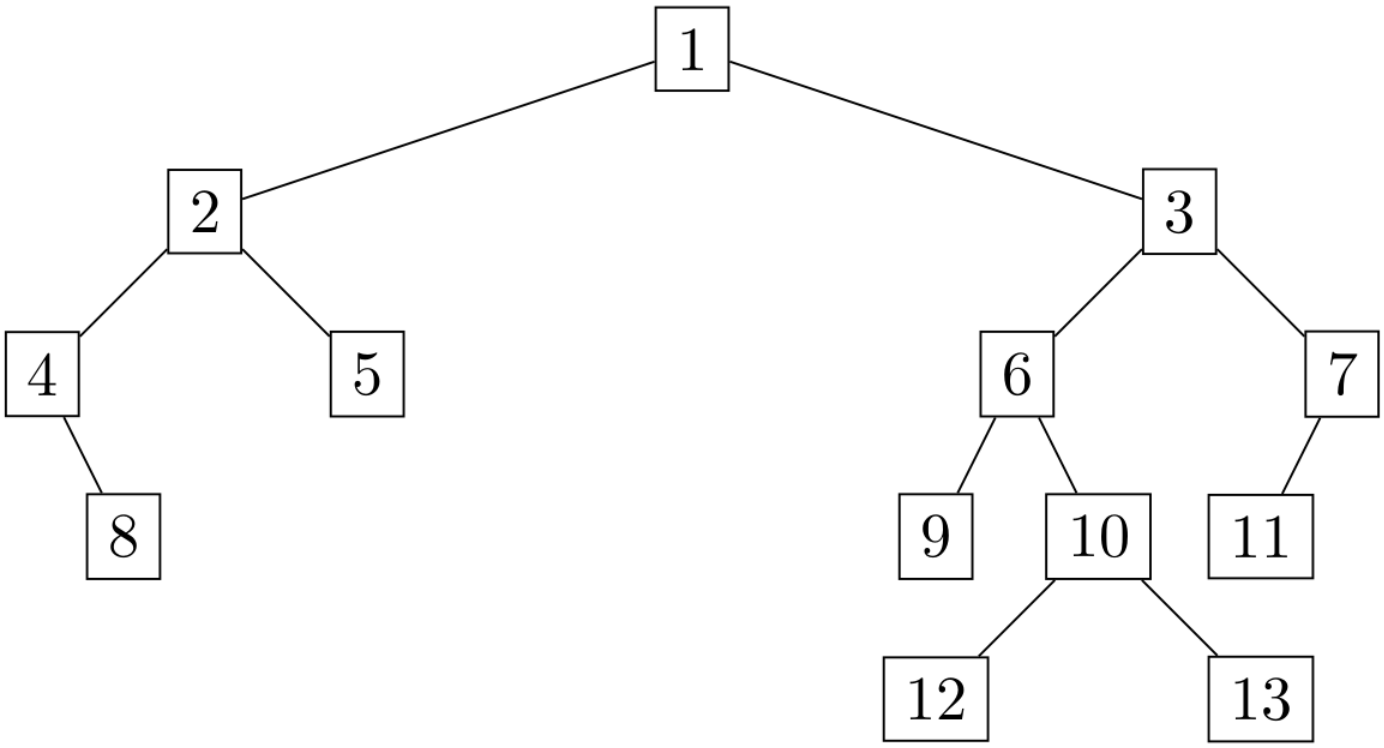

Figure 2: Exercice 2

6. Dans un programme Python, importer le fichier **json** dans un tableau.
7. Écrire la fonction **i_feuille_gauche(arbre: list) → int** qui renvoie l'indice de l'équipe situé sur la feuille la plus à gauche de l'arbre.
8. Écrire alors la fonction **get_matchs(arbre: list) → list** qui renvoie la liste des matchs de huitième de finale sous la forme d'un tableau de tuples.

Exercice 5 *** L'objectif est de créer une classe permettant de construire un arbre binaire de chaînes de caractères *distinctes*.

1. Créer la classe **Arbre_binaire** et son constructeur. On passera un paramètre **h** qui initialisera l'attribut **hauteur** de l'arbre. Le constructeur initialisera:
   - un tableau **arbre** de la taille de l'arbre binaire parfait correspondant à **h** et rempli d'objet **None** .
   - la racine de l'arbre avec la chaîne de caractère **"r"**.
2. Écrire la méthode **get_taille(self) → int** qui renvoie le nombre de nœuds de l'arbre.
3. Écrire la méthode **get_indice(self, chaine: str) → int** qui renvoie la position de la chaîne dans le tableau.
4. Écrire la méthode **inserer(self, pere: str, gauche: str, droit: str) → None** qui ajoute les fils **gauche** et **droit** au nœud **pere**. La méthode lèvera une erreur d'assertion si le nœud **pere** ne peut pas avoir de fils (sort du tableau).
5. Créer une instance de la classe **Arbre_binaire** et reproduire l'arbre binaire ci-dessus.
6. Écrire la méthode récursive **prefixe(self, position: int, parcours: list) → None** qui effectue un parcours préfixe et complète le tableau **parcours** au fur et à mesure.
7. Écrire sur le même modèle les méthodes **infixe** et **postfixe**.
8. **Pour les plus avancés:** Écrire la méthode récursive **prefixe_2(self, position: int) → list** qui construit (par concaténation) le tableau de parcours.

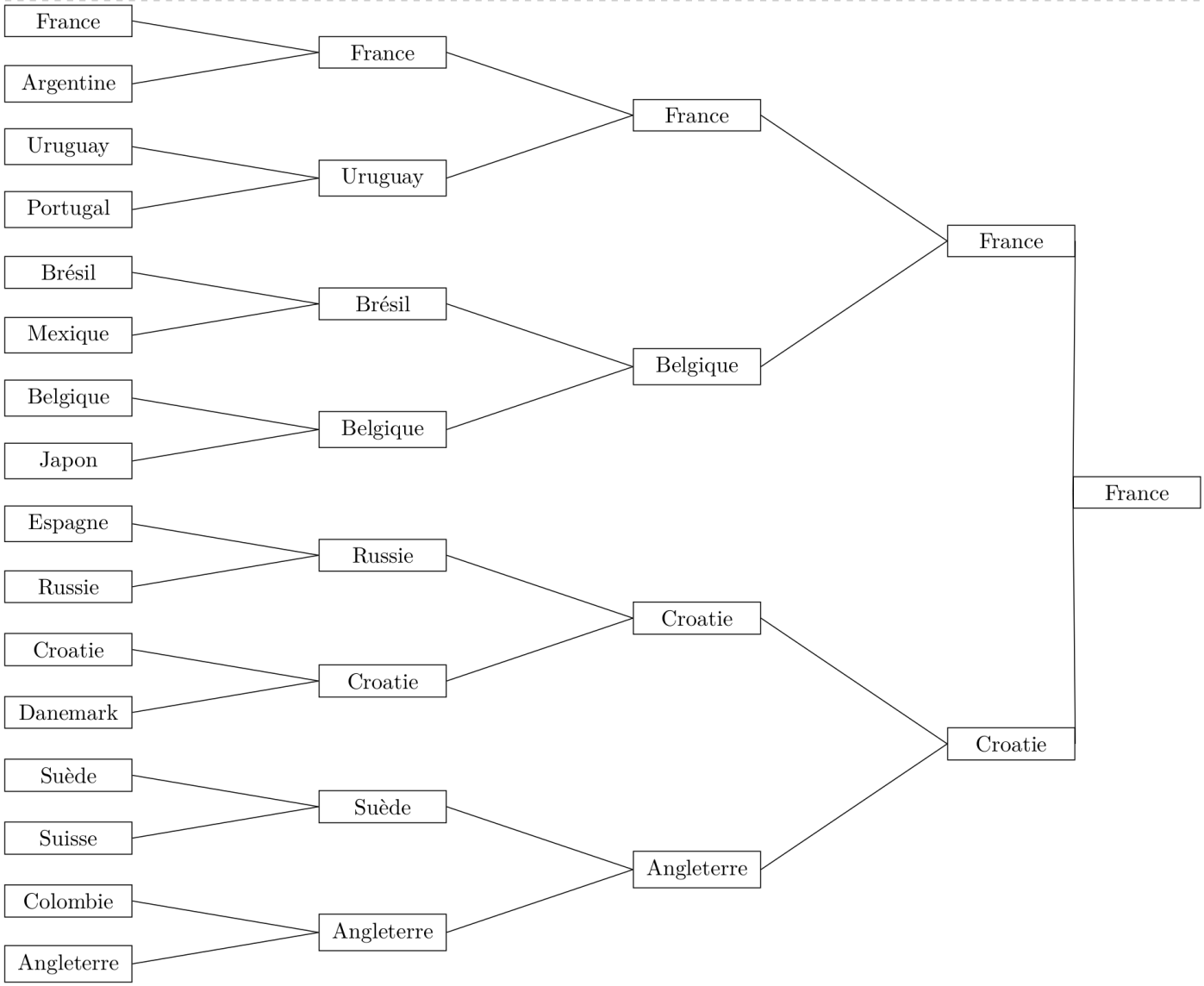

Figure 3: Exercice 4

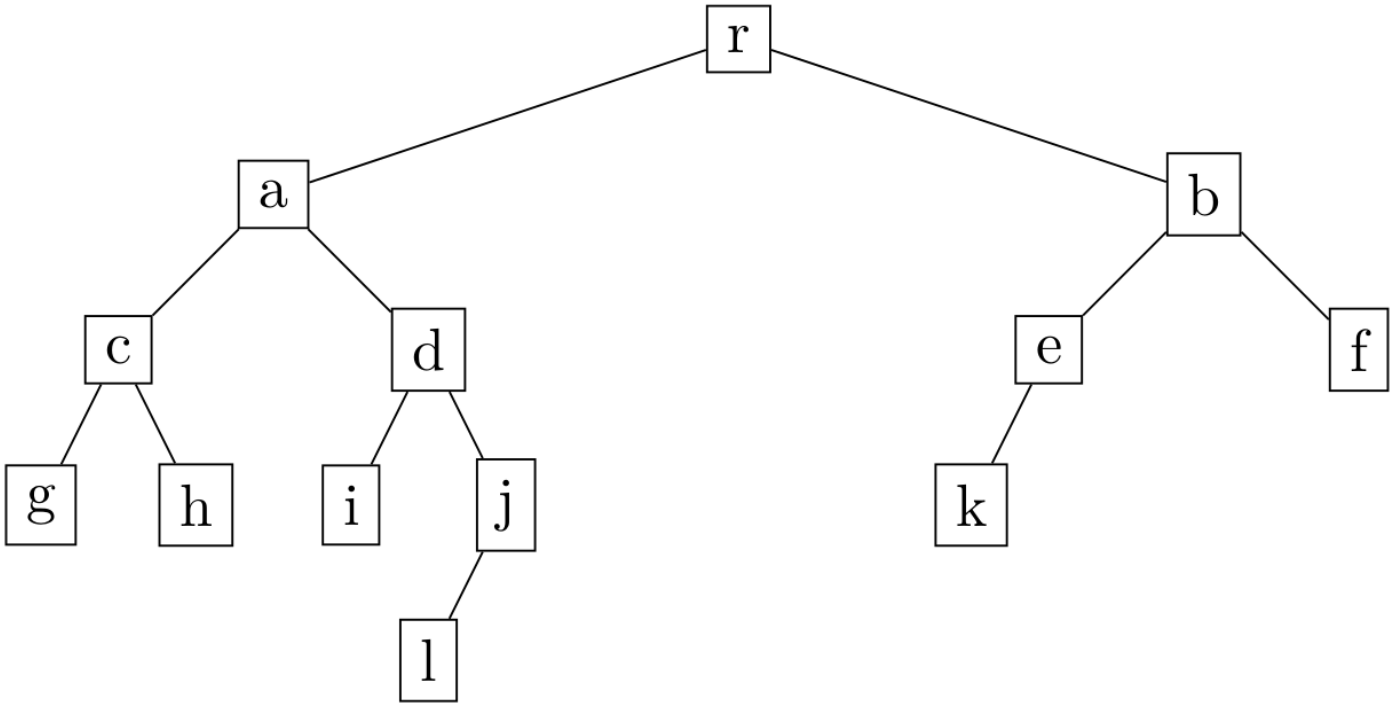

Figure 4: Exercice 5