Exercices

1 Algorithmes récursifs

1.1 Plusieurs algorithmes sur les listes peuvent s'exprimer de façon récursive. Donnez le pseudocode d'un algorithme récursif qui retourne

(a) la longueur d'une liste ;

Solution:

```
fonction LONGUEUR(L : liste) : entier
    si L.ESTVIDE() alors
        retourner 0
    fin si
    retourner 1 + LONGUEUR(L.QUEUE())
fin fonction
```

(b) vrai si et seulement si un élément donné apparaît dans une liste ;

Solution:

```
fonction ESTELEMENT(L : liste, x : élément) : booléen
    si L.ESTVIDE() alors
        retourner Faux
    fin si
    si L.TETE( ) == x alors
        retourner Vrai
    fin si
    retourner ESTELEMENT(L.QUEUE( ), x)
fin fonction
```

(c) le nombre d'occurrences d'un élément donné dans une liste ;

Solution:

```
fonction NBOCCURENCES(L : liste, x : élément) : entier
    si L.ESTVIDE() alors
        retourner 0
    fin si
    si L.TETE( ) == x alors
        retourner 1 + NBOCCURENCES(L.QUEUE( ), x)
    fin si
    retourner NBOCCURENCES(L.QUEUE( ), x)
fin fonction
```

(d) la somme des éléments d'une liste (bien définie si la liste est vide) ;

Solution:

fonction SOMME(L : liste) : élément
  si L.ESTVIDE() **alors**
    retourner 0
  fin si
  retourner L.TETE() + SOMME(L.QUEUE())
fin fonction

On suppose que l'addition est définie sur les éléments de la liste et que 0 est l'élément neutre de l'opération.

(e) le produit des éléments d'une liste (bien défini si la liste est vide) ;

Solution:

fonction PRODUIT(L : liste) : élément
  si L.ESTVIDE() **alors**
    retourner 1
  fin si
  retourner L.TETE() · PRODUIT(L.QUEUE())
fin fonction

On suppose que le produit est définie sur les éléments de la liste et que 1 est l'élément neutre de l'opération.

(f) l'inverse d'une liste (c'est-à-dire la même liste, mais renversée) ;

Solution:

fonction INVERSER(L : liste) : liste
  si L.ESTVIDE() **alors**
    retourner L
  fin si
  retourner INVERSER(L.QUEUE()) + [L.TETE( )]
fin fonction

(g) la i-ème permutation circulaire ou conjuguée d'une liste (c'est-à-dire la liste obtenue en plaçant le premier élément en queue de liste, répété i fois).

Solution:

fonction CONJUGUEE(L : liste, i : entier) : liste
  si $i == 0$ **alors**
    retourner L
  fin si

```
    retourner CONJUGUEE(L.QUEUE() + [L.TETE( )], i − 1)
fin fonction
```

Contrainte : Les seules fonctions disponibles sont
- L.ESTVIDE(), qui retourne vrai si la liste est vide, sinon elle retourne faux ;
- L.TETE(), qui retourne le premier élément de la liste L si elle est non vide ;
- L.QUEUE(), qui retourne la liste obtenue de L en supprimant son premier élément, si L est non vide.
- $L + L'$, qui retourne la concaténation des listes L et L'.
- $x : L$, qui retourne la liste L en insérant l'élément x en première position.

1.2 Soit L une liste triée de longueur n. Donnez le pseudocode d'un algorithme récursif qui vérifie de façon dichotomique si un élément x se trouve ou non dans la liste L. L'en-tête de votre fonction devrait être

fonction CONTIENT(L : liste, x : élément) : boolen

qui retourne vrai si et seulement si x apparaît dans la liste L et elle devrait utiliser une fonction auxiliaire

fonction CONTIENT(L : liste, x : élément, i, j : indices) : boolen

qui retourne vrai si et seulement si x apparaît dans la liste L entre les indices i et j inclusivement.

Solution:

```
fonction CONTIENT(L : liste triée, x : élément) : booléen
    si L.ESTVIDE() alors
        retourner Faux
    fin si
    retourner CONTIENT(L, x, 0, |L| − 1)
fin fonction
fonction CONTIENT(L : liste triée, x : élément,i, j : indices) : booléen
    si i ≥ j alors
        si L[i] == x alors
            retourner Vrai
        fin si
        retourner Faux
    fin si
    m = (j − i)/2
    si x == L[m] alors
        retourner Vrai
```

```
        fin si
        si x < L[m] alors
            retourner CONTIENT(L, x, i, m − 1)
        fin si
        si x > L[m] alors
            retourner CONTIENT(L, x, m + 1, j)
        fin si
    fin fonction
```

1.3 Pour toute paire d'entiers i et n tels que $0 \leq i \leq n$, on définit le coefficient binomial de paramètres (n, i) par

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}.$$

Il est possible de définir les coefficients binomiaux de façon récursive.

(a) Montrez que la relation

$$\binom{n+1}{i+1} = \binom{n}{i} + \binom{n}{i+1}$$

est satisfaite à partir de la définition de coefficient binomial donnée plus haut.

Solution: On a que

$$\begin{aligned}
\binom{n}{i} + \binom{n}{i+1} &= \frac{n!}{i!(n-i)!} + \frac{n!}{(i+1)!(n-(i+1))!}, \text{ par la définition de } \binom{n}{i} \\
&= \frac{n!}{i!(n-i-1)!(n-i)} + \frac{n!}{i!(n-(i+1))!(i+1)}, \text{ car } k! = (k-1)! \cdot k \\
&= \frac{n!(i+1) + n!(n-i)}{(i+1)!(n-i)!} \\
&= \frac{n!(n+1)}{(i+1)!(n-i)!} \\
&= \frac{(n+1)!}{(i+1)!(n+1-(i+1)!} \\
&= \binom{n+1}{i+1}
\end{aligned}$$

(b) Donnez le pseudocode d'un algorithme récursif qui n'utilise que des additions et qui retourne le coefficient binomial de paramètres (n, i) si $0 \leq i \leq n$.

Solution:

```
    fonction BINOMIAL(n, i : entier) : entier
```

si $i == 0$ ou $i == n$ **alors**
retourner 1
fin si
retourner BINOMIAL$(n-1, i-1)$ + BINOMIAL$(n-1, i)$
fin fonction

(c) Donnez le pseudocode d'un algorithme non récursif qui n'utilise que des additions et qui retourne le coefficient binomial de paramètres (n, i) si $0 \leq i \leq n$. *Indice :* Utilisez deux listes pour vous aider, qui contiennent les coefficients binomiaux pour la valeur de n "courante" et une autre pour $n-1$ (les valeurs "précédentes").

Solution:

fonction BINOMIAL$(n, i$: entier$)$: entier
$precedent = [1]$
pour $j \in \{1, \ldots, n\}$ **faire**
$courant = [\,]$
pour $k \in \{0, \ldots, j\}$ **faire**
si $k == 0$ ou $k == j$ **alors**
$courant$.AJOUTER(1)
sinon si
alors $courant$.AJOUTER$(precedent[k] + precedent[k-1])$
fin si
fin pour
$precedent = courant$
fin pour
retourner $precedent[i]$
fin fonction

(d) Implémentez les deux algorithmes dans SageMath pour vérifier si votre pseudocode fonctionne.

Solution: Voici l'implémentation de récursive de l'algorithme :

```
def binomial(n,i):
    if i == 0 or i == n:
        return 1
    return binomial(n-1,i-1) + binomial(n-1,i))))
```

Maintenant, voici l'implémentation itérative de l'algorithme :

```
def binomial(n,i):
    precedent = [1]
    for j in range(1,n+1):
        courant = []
        for k in range(j+1):
            if k == 0 or k == j:
```

```
                courant.append(1)
            else:
                courant.append(precedent[k]+precedent[k
                    -1])
        precedent = courant
    return precedent[i]
```

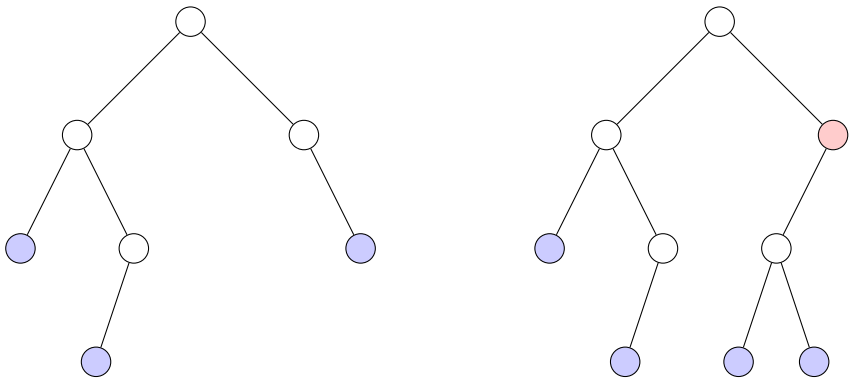
1.4 Un *arbre binaire* A est une structure récursive définie comme suit :

(i) Soit A est l'*arbre vide*, qu'on dénote par ϕ ou

(ii) A est composé d'un *noeud*, d'un *sous-arbre gauche* et d'un *sous-arbre droit*.

Un noeud d'un arbre est appelé *feuille* si ses sous-arbres gauche et droit sont vides. On dit d'un arbre qu'il est *équilibré* si, pour chacun de ses noeuds, le sous-arbre gauche et le sous-arbre droit ont des hauteurs qui diffèrent d'au plus 1. Dans les deux questions qui suivent, si A n'est pas un arbre vide, alors *A.gauche()* retourne son sous-arbre gauche et *A.droit()* retourne son sous-arbre droit. De plus, vous pouvez supposer que l'expression *hauteur*(A) retourne la hauteur de l'arbre binaire A.

L'image ci-bas illustre la situation. Les feuilles sont colorées en bleu. L'arbre de gauche est équilibré alors que l'arbre de droite ne l'est pas : il y a un déséquilibre au noeud en rouge, puisque le sous-arbre gauche est de hauteur 2 et le sous-arbre droit est de hauteur 0 (différence de $2 > 1$.



(a) Écrivez un algorithme récursif qui calcule le nombre de feuilles présentes dans un arbre binaire.

Solution: Pour compter le nombre total de feuilles, il suffit d'additionner le nombre de feuilles dans les sous-arbres gauche et droit. Il y a par contre deux cas de base. Si on a un arbre vide, celui-ci n'est pas une feuille, alors il faut retourner zéro. Par contre, si le noeud courant n'a aucun fils, alors il faut retourner un car il s'agit bien d'une feuille.

fonction NbFeuilles(A : arbre binaire) : entier positif
  si $A = \phi$ **alors**
    retourner 0

```
    sinon si A.gauche() = φ et A.droit() = φ alors
        retourner 1
    sinon
        retourner NbFeuilles(A.gauche()) + NbFeuilles(A.droit())
    fin si
fin fonction
```

(b) Écrivez un algorithme récursif qui vérifie si un arbre binaire est équilibré.

Solution: Il suffit de vérifier récursivement que les sous-arbres gauche et droit sont équilibrés. De plus, il faut s'assurer qu'il n'y a pas de déséquilibre au noeud courant, c'est-à-dire que la différence entre la hauteur des sous-arbres est entre -1 et 1. Dans ce cas, il n'y a qu'un cas de base, qui est l'arbre vide (celui-ci est équilibré).

```
fonction EstEquilibré(A : arbre binaire) : booléen
    si A = φ alors
        retourner vrai
    sinon
        retourner EstEquilibre(A.gauche())∧
            EstEquilibre(A.droit())∧
            |hauteur(A.gauche())−hauteur(A.droit())| ≤ 1
    fin si
fin fonction
```

Dans les deux cas, indiquez bien l'en-tête de la fonction, les paramètres d'entrée et le type de la valeur de retour.

1.5 Un *arbre binaire de recherche* (ABR) est un arbre binaire dans lequel chaque noeud contient une valeur dans $\mathbb{Z}$ qu'on appelle *clé* et vérifiant les propriétés suivantes :

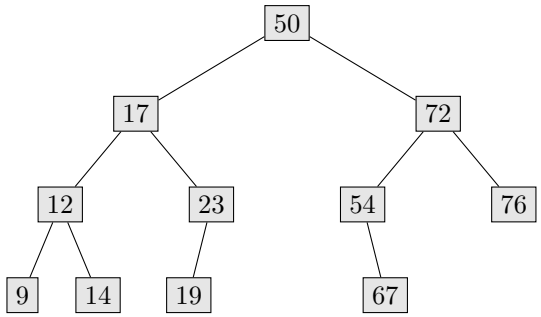
(i) Toutes les clés dans le sous-arbre gauche sont strictement plus petites que la clé de la racine ;

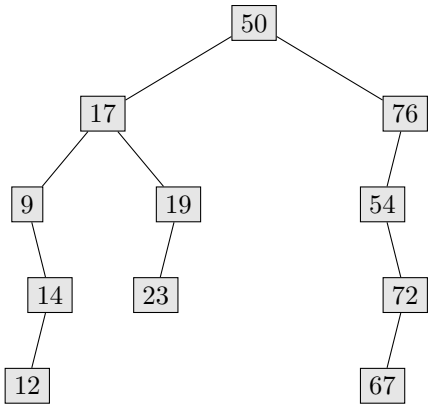
(ii) Toutes les clés dans le sous-arbre droit sont strictement plus grandes que la clé de la racine ;

(iii) Les sous-arbres gauche et droit sont aussi eux-même des arbres binaires de recherche ;

(iv) Aucune clé n'apparaît deux fois ou plus.

Par exemple, l'arbre binaire de gauche est un ABR, mais pas celui de droite, car la clé 23 apparaît à gauche de la clé 19.





Donnez le pseudocode d'un algorithme récursif qui vérifie si un arbre binaire est bien un arbre binaire de recherche. L'en-tête de votre fonction devrait être

fonction EstABR(A : arbre binaire) : booléen

Suggestion : Utilisez une fonction auxiliaire dont l'en-tête est

fonction EstABR(A : arbre binaire, i, j : entiers ou infini) : booléen

qui retourne vrai si et seulement si A est un arbre binaire dont les clés se trouvent entre i et j inclusivement.
Vous pouvez supposer que les services suivants sont disponibles :

1. A.GAUCHE() retourne le sous-arbre gauche de A ;
2. A.DROIT() retourne le sous-arbre droit de A ;
3. A.CLÉ() retourne la clé qui se trouve dans le noeud à la racine de A.

Solution: La foncion récursive est la suivante :

fonction EstABRRecursif(A : arbre binaire) : booléen
  si $A = \phi$ **alors**
    retourner *vrai*
  sinon
    retourner EstABRRecursif(A.gauche(), 0, A.clé() - 1) et
      EstABRRecursif(A.droit(), A.clé() + 1, ∞)
  fin si
fin fonction

Voici la fonction auxiliaire :

fonction EstABRRecursif(A : arbre binaire, i, j : entiers ou infini) : booléen
  si $A = \phi$ **alors**
    retourner *vrai*
  sinon si A.clé() $< i$ **ou** A.clé() $> j$ **alors**

```
        retourner faux
    sinon
        retourner EstABRRecursif(A.gauche(), i, A.clé() - 1) et
          EstABRRecursif(A.droit(), A.clé() + 1, j)
    fin si
fin fonction
```