❑ **Exercice** : *type A*

Dans cet exercice, on autorise les doublons dans un arbre binaire de recherche et pour le cas d'égalité on choisira le sous-arbre gauche. On ne cherchera pas à équilibrer les arbres.

1. Rappeler la définition d'un arbre binaire de recherche.
2. Insérer successivement et une à une dans un arbre binaire de recherche initialement vide toutes les lettres du mot `bacddabdbae`, en utilisant l'ordre alphabétique sur les lettres. Quelle est la hauteur de l'arbre ainsi obtenu ?
3. Montrer que le parcours en profondeur infixe d'un arbre binaire de recherche de lettres est un mot dont les lettres sont rangées dans l'ordre croissant. On pourra procéder par induction structurelle.
4. Proposer un algorithme qui permet de compter le nombre d'occurrences d'une lettre dans un arbre binaire de recherche de lettres. Quelle est sa complexité ?
5. On souhaite supprimer une occurrence d'une lettre donnée d'un arbre binaire de recherche de lettres. Expliquer le principe d'un algorithme permettant de résoudre ce problème et le mettre en œuvre sur l'arbre obtenu à la question 2. en supprimant successivement une occurrence des lettres e, b, b, c et d. Quelle en est la complexité ?

❑ **Exercice** : *type B*

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} u_0 = e - 1 \\ u_{n+1} = (n+1)\,u_n - 1 \end{cases}$$

On note

$$S_n = \sum_{k=0}^{n} \frac{1}{k!}$$

On pourra utiliser sans justification le résultat suivant : pour tout $n \in \mathbb{N}$: $S_n \leq e \leq S_n + \dfrac{1}{n\,n!}$

1. Montrer que $e = \lim\limits_{n\to+\infty} S_n$
2. Monter que pour tout $n \in \mathbb{N}$, $u_n = n!(e - S_n)$
3. En déduire que $(u_n)_{n\in\mathbb{N}}$ converge et donner sa limite.
4. Ecrire en langage C, une fonction `main` qui prend un entier n en argument sur la ligne de commande et affiche les n premières valeurs de la suite u_n. On utilisera le type `double` pour les flottants et la valeur `M_E` de `<math.h>` pour représenter le nombre e.
5. Tester votre fonction pour $n = 17$, le comportement observé est-il conforme à celui établi à la question 3 ?
6. Tester votre fonction pour $n = 25$, commenter le résultat obtenu.
7. La fonction `nextafter` disponible dans `<math.h>` de signature `double nextafter(double x, double y)` renvoie le nombre flottant en double précision qui suit exactement `x` en allant vers `y`. Remplacer `M_E` comme valeur de u_0 par le flottant suivant et tester de nouveau le comportement de la suite. Commenter.