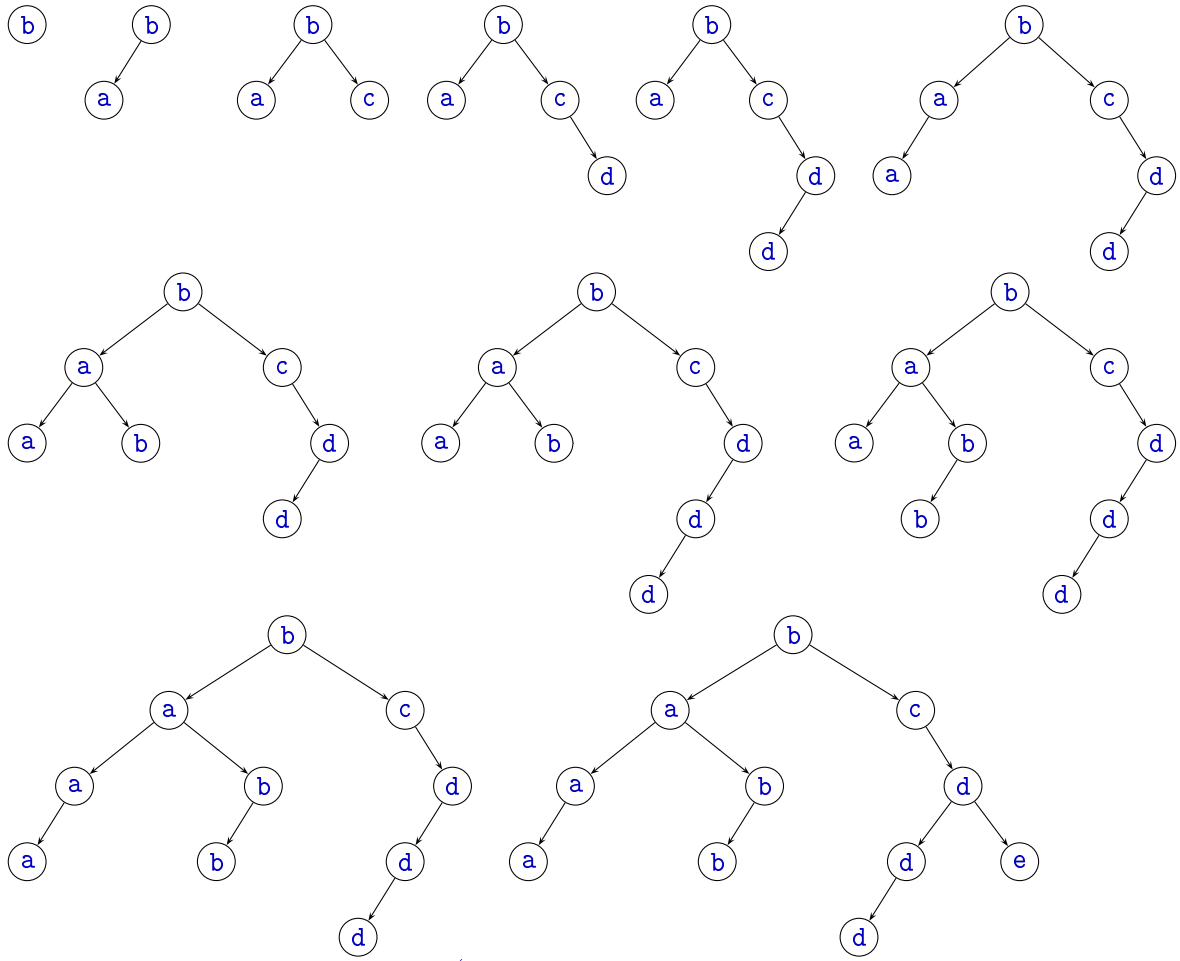
❑ **Exercice** : *type A*

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.

   Un arbre binaire sur un ensemble d'étiquettes E, peut se définir inductivement par l'axiome $\varnothing$ (arbre vide), et la règle d'inférence d'arité 2 : $(g, d) \mapsto (g, x, d)$ où $x \in E$ et toute étiquette de g est inférieure à x et toute étiquette de d est supérieure à x.

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 ?



   La hauteur de l'arbre obtenu est 4 (la programme donne -1 comme hauteur de l'arbre vide.)

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.

On procède par induction structurelle, pour tout ABR a, on note $P(a)$ la propriété « le parcours infixe de a est un mot dont les lettres sont rangées dans l'ordre croissant ».

— $P(\varnothing)$ est vraie et donc la propriété P est vérifiée pour tous les axiomes.

— Montrons à présent la conservation de la propriété P par application de la règle d'inférence, si g et d sont deux ABR vérifiant P, et x une lettre telle que toutes les lettres de g sont avant x et toutes les lettres de d sont après x dans l'ordre alphabétique. Le parcours infixe de l'ABR (g, x, d) est par définition le parcours infixe de g suivi de x suivi du parcours infixe de d. Par hypothèse d'induction, les lettres du parcours de g et celles du parcours de d sont rangées par ordre croissant. Comme de plus x est après toute les lettres de g et avant toutes celles de d, le parcours de (g, x, d) est bien formé de lettres rangées par ordre croissant.

Par application du principe d'induction structurelle, pour tout ABR a, $P(a)$ est vraie.

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 ?

On part de la racine, à chaque étape, on ajoute 1 au nombre d'occurence si l'étiquette est la lettre cherchée et on descend dans le sous arbre gauche si l'étiquette est inférieure ou égale à la racine et dans le sous arbre droit sinon. On s'arrête en rencontrant une feuille. Comme à chaque étape on descend d'un niveau dans l'orbre la complexité est en $O(h)$ où h est la hauteur de l'arbre.

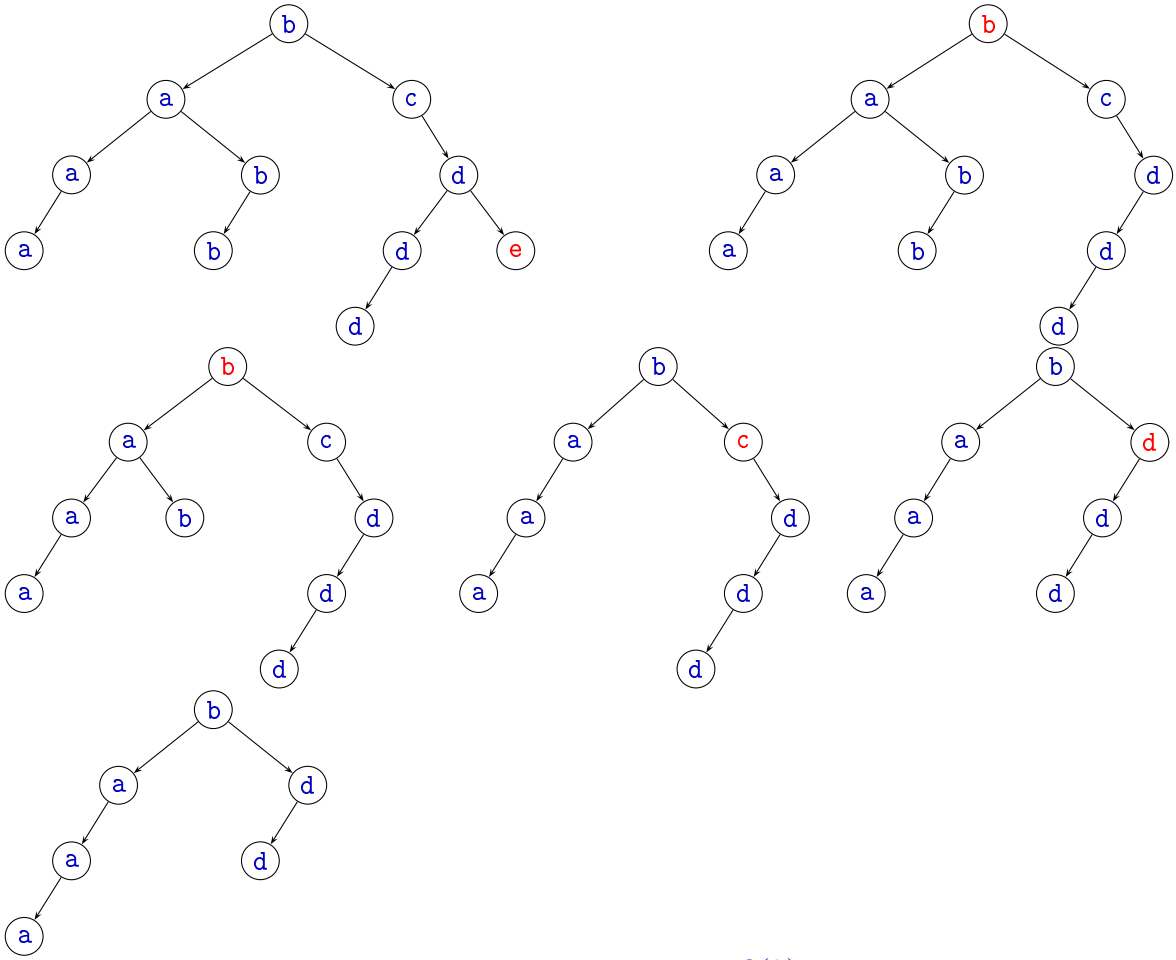
5. On souhaite supprimer une occurrence d'une lettre donnée d'un arbre binaire de recherche de lettres. Expliquer le principe d'un algorithme permettant de résoudre ce problème et le mettre en œuvre sur l'arbre obtenu à la question 2. en supprimant successivement une occurrence des lettres e, b, b, c et d. Quelle en est la complexité ?

Afin de supprimer une valeur dans un ABR on recherche cette valeur dans l'arbre puis, :

— s'il s'agit d'une feuille, on la supprime de l'arbre

— si le sous arbre gauche est vide alors on remplace par le sous arbre droit

— sinon, on remplace par la plus grande valeur présente dans le sous arbre gauche en y supprimant cette valeur



La recherche de la valeur dans l'ABR a une complexité en $O(h)$ de même que la recherche de la plus grand valeur du sous arbre gauche, donc la complexité est en $O(h)$.

❑ **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 $e = \lim\limits_{n\to+\infty} S_n$

   On soutrait S_n à chaque membre dans l'inégalite $S_n \leq e \leq S_n + \dfrac{1}{n\,n!}$ et on passe à la limite.

2. Monter que pour tout $n \in \mathbb{N}$, $u_n = n!(e - S_n)$

   Récurrence tranquille !

3. En déduire que $(u_n)_{n\in\mathbb{N}}$ converge et donner sa limite.

On utilise de nouveau l'inégalité $S_n \leq e \leq S_n + \frac{1}{n\,n!}$ en soustrayant S_n puis en mutipliant par $n!$.

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.

```c
void u(int n)
{
    double v = nextafter(M_E - 1,3.0);
    printf("v0=%.16f\n",v);
    for (int i=1;i<=n;i++)
    {
        v = (double)i*v - 1.0;
        printf("v%d=%.16f\n",i,v);
    }
}

int main(int argc, char* argv[])
{
    u(atoi(argv[1]));
}
```

5. Tester votre fonction pour $n = 17$, le comportement observé est-il conforme à celui établi à la question 3 ?

La suite a l'air de tendre vers 0 comme prévu.

6. Tester votre fonction pour $n = 25$, commenter le résultat obtenu.

La suite prend des valeurs négatives de plus en plus grandes.

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.

Cette fois les valeurs sont positives et de plus en plus grandes. Le problème est du à l'irrationnalité de e qui n'est donc pas représentable exactement par un flottant. Or en remplaçant e par $e + \delta$ dans la définition de la suite u_n on change son comportement (suivant le signe de δ).