❑ **Exercice 1** : *Doublons autorisés dans un arbre binaire de recherche* 🎓 Oraux CCINP

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. Pour l'implémentation on propose d'utiliser le type `abr` suivant :

```
type abr = Vide | Noeud of abr*char*abr;;
```

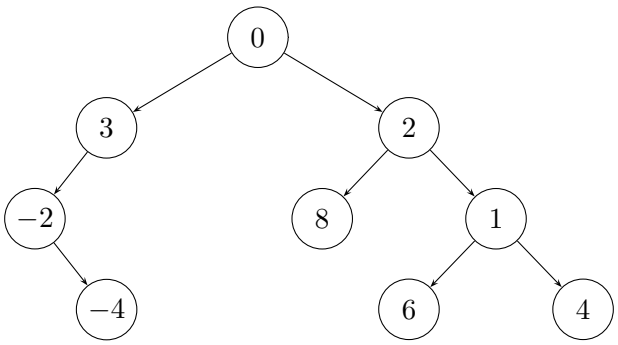
Ecrire la fonction `insere abr -> char -> abr` qui prend en argument un arbre `abr` et une lettre et renvoie l'arbre obtenu en insérant cette lettre dans `abr`.

6. Proposer une implémentation en OCaml pour l'algorithme de la question 4

❑ **Exercice 2** : *Minima locaux dans des arbres* 🎓 Oraux CCINP

Dans cet exercice, on considère des arbres étiquetés par des entiers relatifs deux à deux distincts. On dit qu'un noeud est un *minimum local* d'un arbre si son étiquette est plus petite que celle de son éventuel père et celles de ses éventuels fils.

1. Déterminer le ou les minima locaux de l'arbre $A$ suivant :




2. Donner la définition inductive des arbres binaires ainsi que la définition de la hauteur d'un arbre. Quelle est la hauteur de l'arbre $A$ ?
3. Montrer que tout arbre non vide possède un minimum local.
4. Proposer un algorithme permettant de trouver un minimum local d'un arbre non vide et déterminer sa complexité.
5. Ecrire une implémentation de cet algorithme en OCaml en utilisant le type arbre binaire suivant :

```
type ab =
 |Vide
 |Noeud of ab *  int * ab;;
```

6. Ecrire une fonction `min_locaux ab -> int list` qui prend en argument un arbre binaire et renvoie la liste de tous ses minima locaux.

❑ **Exercice 3** : *Tableaux autoréférents* 🎓 Oraux CCINP

On dit qu'un tableau `tab` de taille $n$ est *autoréférent* si pour tout entier `i` $\in [\![0; n-1]\!]$, `tab.(i)` est le nombre d'occurences de `i` dans `tab`. Par exemple, le tableau `ex=[| 1; 2; 1; 0 |]` est autoréférent, en effet :

— `ex.(0) = 1` et `0` apparaît bien une fois dans le tableau

— `ex.(1) = 2` et `1` apparaît bien deux fois dans le tableau

— `ex.(2) = 1` et `3` apparaît bien une fois dans le tableau

— `ex.(3) = 0` et `3` n'apparaît pas dans le tableau

1. Justifier rapidement que si `tab` est un tableau autoréférent de taille `n` alors les éléments de `tab` sont tous inférieurs ou égaux à `n`
2. Montrer que pour $n \in [\![1;3]\!]$, il n'existe aucun tableau auto référent de taille $n$.
3. Déterminer un autre tableau autoréférent de taille 4 que celui donné en exemple.
4. Soit `n`$\geq 7$, on définit le tableau `tab` de taille `n` par :
   — `tab.(0) = n-4`
   — `tab.(1) = 2`
   — `tab.(2) = 1`
   — `tab.(n-4) = 1`
   — `tab.(i) = 0` si `i` $\notin \{0, 1, 2, n-4\}$

   Prouver que `tab` est autoréférent
5. Montrer que si `tab` est un tableau autoréférent de taille `n` alors la somme des éléments de `tab` vaut `n`. La réciproque est-elle vraie ?
6. Ecrire en OCaml une fonction `est_autoreferent int array -> bool` qui prend en argument un tableau d'entiers et renvoie `true` si ce tableau est autoréférent et `false` sinon. On attend une complexité en $O(n)$ où $n$ est la taille du tableau.

   On cherche maintenant à construire un tableau autoréférent de taille `n` en utilisant un algorithme de recherche par retour sur trace (*backtracking*) qui valide une solution partielle construite jusqu'à un index `idx` donné, on propose pour cela la fonction suivante :

```
    if tab.(i)=elt then res := !res + 1;
  done;
  !res;;

  let partiel t  k =
    let n = Array.length t in
    let cpt = Array.make  n  0 in
    try
    for i=0 to k-1 do
      cpt.(t.(i)) <- cpt.(t.(i)) + 1
    done;
    for i=0 to k-1 do
      if cpt.(i) > t.(i) then  raise Exit;
      done;
```

7. Ecrire une fonction de validation partielle qui pour le moment renvoie toujours `true` et tester cette fonction pour de petites valeurs de `n` (attention, il faut aussi écrire la fonction d'affichage du tableau), que constater vous ?
8. Proposer et implémenter des améliorations de la fonction de validation partielle en utilisant les résultats établis sur les tableaux autoréférents aux questions précédentes.

❑ **Exercice 4** : *Convergence d'une suite*

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

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

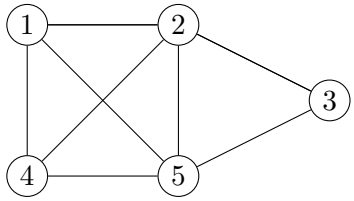
On pourra utiliser sans justification le résultat suivant : pour tout $n \in \mathbb{N}$ : $S_n \leq e \leq S_n + \dfrac{1}{n\,n!}$

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 on pourra utiliser 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.

❑ **Exercice 5** : *Triangle dans un graphe*

On considère un graphe non orienté $G = (S, A)$ où $A = \{1, \dots, n\}$. On dit qu'un sous ensemble de $V$ à trois éléments $\{x, y, z\}$ est un *triangle* de $G$ lorsque $\{x, y\}, \{y, z\}$ et $\{x, z\}$ appartiennent à $A$. Par exemple, dans le graphe $g$ suivant, $\{1, 2, 4\}$ est un triangle.




1. Donner tous les autres triangles du graphe $g$.
2. Rappeler la défintion d'un graphe complet. Donner le nombre de triangle d'un graphe complet à $n$ sommets.
3. On dit qu'un graphe est bipartite lorsqu'on il existe une partition de l'ensemble des sommets $S$ en deux ensembles $S_1$ et $S_2$ tel que tout arête ait un élément dans $S_1$ et l'autre dans $S_2$. Dessiner un graphe bipartite à 6 sommets et 9 arêtes.
4. Montrer qu'un graphe bipartite ne contient pas de triangles.
5. Afin de lister les triangles d'un graphe, on propose de tester si chaque partie de $A$ à trois éléments est un triangle. Indiquer la complexité d'un tel algorithme.
6. Dans cette question uniquement, on utilise des graphes représentés par matrice d'adjacence à l'aide du type structuré en C :

```c
// Nombre maximal de sommets
#define NMAX 100
// La matrice d'adjacence du graphe
typedef bool graphe[NMAX][NMAX];
```

la fonction d'ajout d'un arc dans un tel graphe s'écrit :

```c
void cree_arete(graphe g, int i, int j){
    g[i][j] = true;
    g[j][i] = true;}
```

Ecrire la fonction `liste_triangle`, de prototype `void liste_triangle(graphe g, int n)` qui prend en argument un graphe `g` et son nombre de noeud `g` et affiche sur la sortie standard les triangles de ce graphe en utilisant l'algorithme décrit à la question précédente.

7. Un autre algorithme possible pour lister les triangles d'un graphe consiste pour chaque arête $\{x, y\}$ à chercher l'intersection de l'ensemble des sommets $z$ adjacent à $x$ et à $y$. Donner la complexité de cet algorithme si on suppose que l'intersection est calculée en temps linéaire du nombre de sommets.
8. Ecrire en OCaml, une fonction `intersection int list -> int list -> int list` qui calcule en temps linéaire l'intersection de deux listes *en les supposant triées*.
9. Dans la suite de l'exercice, on utilise des graphes représentés par liste d'adjacence en OCaml avec le type :

```
type graphe = {
  taille : int;
  ladj : int list array};;
```

Et on donne la fonction permettant de créer un graphe de taille donnée `n` :

```
let cree_graphe n =
  {taille=n; ladj = Array.make n []};;
```

Ecrire une fonction permettant d'ajouter une arête dans un graphe en *maintenant triée* les listes d'adjacence.

10. Implémenter l'algorithme décrit à la question 7 pour lister les triangles d'un graphe.