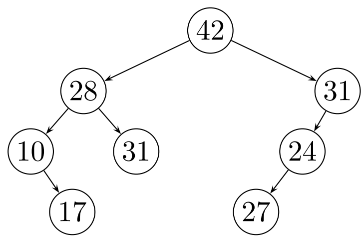
Devoir surveillé d'informatique

Consignes

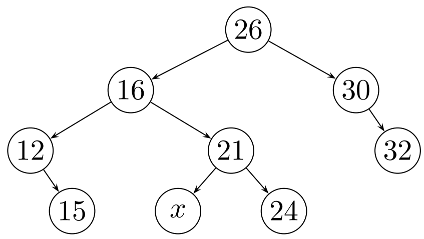
- Les programmes demandés doivent être écrits en C ou en OCaml. Dans le cas du C, on suppose que les librairies standards usuelles (`<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`, `<stdassert.h>`, ...) sont déjà importées.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

❑ **Exercice 1** : *Questions de cours*

1. Donner la définition d'un arbre binaire.
2. Donner les définitions de la hauteur et de la taille d'un arbre binaire.
3. Donner la définition d'un arbre binaire de recherche.
4. Prouver le parcours infixe d'un arbre binaire de recherche fournit les clés dans l'ordre croissant.
   Indication : on pourra raisonner par récurrence sur la taille de l'arbre.
5. Donner l'ordre des noeuds lors des parcours prefixe, infixe et suffixe de l'arbre suivant :



6. On considère l'arbre binaire suivant :



   Donner les valeurs de l'étiquette x pour lesquelles cet arbre est un arbre binaire de recherche.
7. On implémente les arbres binaires de recherche en OCaml à l'aide du type suivant :

```ocaml
type abr =
  | Vide
  | Noeud of  abr * int * abr;;
```

   Ecrire une fonction `insere : int -> abr -> abr` qui prend en argument un entier x et un arbre binaire de recherche a et renvoie un arbre binaire de recherche contenant x et tous les éléments de a.

❑ **Exercice 2** : *Valeur plus petite la plus proche*
On considère un tableau d'entiers *positifs* et on s'intéresse au problème de la recherche pour chacun de ces entiers de la valeur plus petite la plus proche située à gauche dans le tableau. Dans le cas où aucune valeur située à gauche dans le tableau n'est plus petite que la valeur considérée alors on renverra -1.
Par exemple dans le tableau $\{2, 1, 7, 9, 8, 3\}$:

— Il n'y a aucune valeur à gauche de 2, donc la valeur plus petite la plus proche est -1,

- Pour 1, aucune valeur située à gauche n'est plus petite, donc on renvoie aussi -1,
- Pour 7, la valeur plus petite la plus proche est 1.
- Pour 9, c'est 7.
- Pour 8 c'est 7.
- Pour 3, c'est 1.

Et donc le tableau des valeurs plus petites les plus proches dans cet exemple est $\{-1,-1,1,7,7,1\}$

1. Donner le tableau des valeurs plus petites les plus proches pour le tableau $\{5,7,11,6,9,2\}$
2. On propose l'algorithme suivant pour résoudre ce problème : pour chaque élément `tab[i]` du tableau on parcourt les valeurs `tab[i-1], ..., tab[0]` *dans cet ordre*, si on trouve un élément strictement inférieur à `tab[i]` alors c'est la valeur plus petite la plus proche, sinon la valeur plus petite la plus proche est -1. Ecrire une implémentation de cet algorithme en C sous la forme d'une fonction de signature `int *vpp_naif(int tab[], int size)` qui prend en argument un tableau d'entiers `tab` ainsi que sa taille `size` et un renvoie un tableau de taille `size` contenant à l'indice `i` la valeur strictement inférieure la plus proche de `tab[i]`.
3. Justifier rapidement que l'algorithme précédent a une complexité quadratique On considère maintenant l'algorithme suivant qui utilise une pile dotée de son interface usuelle (`est_vide, empiler, depiler`) et de la fonction `sommet` qui renvoie la valeur située au sommet de la pile sans la dépiler.

Algorithme : Valeurs plus petites les plus proches

Entrées : Un tableau t d'entiers positifs de taille n
Sorties : Un tableau s d'entiers positifs de taille n tel que $s[i]$ soit la valeur plus petite la plus proche de $t[i]$

```
s ← tableau de taille n
p ← pile de taille maximale n
pour i ← 0 à p − 1 faire
    tant que p n'est pas vide et sommet(p) ⩾ t[i] faire
        depiler(p) ;
    fin
    si p est vide alors
        s[i] ← −1
    fin
    sinon
        s[i] ← sommet(p)
    fin
    empiler t[i] dans p
fin
return s
```

4. On fait fonctionncer cet algorithme sur le tableau $\{2,7,5,8,6,3\}$. Recopier et compléter le tableau suivant qui indique pour chaque valeur de l'indice `i` de la boucle `for` l'état de la pile et du tableau s après l'exécution de la boucle pour les valeurs de `i` de 0 à 5 (on note une pile avec les extrémités `|` et `>` pour indiquer le sommet de la pile)

<table>
<tr><th>i</th><th>État de la pile</th><th>État du tableau s</th></tr>
<tr><td>Initialement</td><td><code>|></code></td><td>$\{-1,-1,-1,-1,-1,-1\}$</td></tr>
<tr><td>0</td><td><code>|2></code></td><td>$\{-1,-1,-1,-1,-1,-1\}$</td></tr>
<tr><td>1</td><td><code>|2, 7></code></td><td>$\{-1,2,-1,-1,-1,-1\}$</td></tr>
<tr><td>2</td><td>...</td><td>...</td></tr>
<tr><td>3</td><td>...</td><td>...</td></tr>
<tr><td>4</td><td>...</td><td>...</td></tr>
<tr><td>5</td><td>...</td><td>...</td></tr>
</table>

5. On suppose qu'on a *déjà implémentée* en C une structure de donnée de pile qu'on manipule à l'aide des fonctions suivantes :
   - `est_vide` de signature `bool est_vide(pile p)`,
   - `empiler` de signature `void empiler(pile *p, int v)`,
   - `depiler` de signature `int depiler(pile *p)`.

Ecrire en utilisant ces fonctions une fonction `sommet` de signature `int sommet(pile *p)` qui renvoie le sommet de la pile sans le depiler si la pile n'est pas vide et -1 sinon.

6. Ecrire une implémentation en C de l'algorithme des valeurs plus petites les plus proches donné ci-dessus et utilisant une pile sous la forme d'une fonction de signature `int *vpp_pile(int tab[], int size)` qui renvoie le tableau des valeurs plus petites les plus proches.
7. Prouver que cet algorithme est de complexité linéaire, on pourra vérifier que chaque élément du tableau t est empilé une fois et dépilé au plus une fois.

❑ **Exercice 3** : *Base de données de publications scientifiques*
On utilise le schéma relationnel suivant afin de modéliser une base de données de publications scientifiques. Chaque article publié ayant un ou plusieurs auteurs.

- **Article** (<u>IdArticle</u>, titre, revue, volume, annee)
- **Auteur** (<u>IdAuteur</u>, nom, prenom)
- **Publie** (<u>#Article,#Auteur</u>)

La clé étrangère #Article de la table **Publie** fait référence à la clé primaire de la table **Article** et la clé étrangère #Auteur de la table **Publie** fait référence à la clé primaire de la table **Auteur**. Les attributs titre, revue, nom et prénom sont des chaines de caractères, les autres sont des entiers.

1. Justifier que l'attribut #Article de la table **Publie** seul, ne peut pas servir de clé primaire pour cette table.
2. Expliquer ce qu'affiche la requête suivante :

```sql
SELECT nom, prenom
FROM Auteur
JOIN Publie ON Auteur.IdAuteur = Publie.Auteur
WHERE Publie.Article = 42
```

3. Ecrire les requêtes permettant d'afficher les informations suivante :
   a) La liste des titres des articles parus en 2022 listé par ordre alphabétique.
   b) Les noms des revues listé par ordre alphabétique, sans répetition.
   c) Les noms et prénoms des auteurs qui ont publié dans la revue "Nature" en 2000.
   d) Les titres et revues des articles écrits (ou co-écrit) par Donald KNUTH en 2010.
   e) La liste des volumes de la revue "Nature" en 2020 avec le nombre d'article qu'il contient.
   f) Pour chaque revue, son nom et l'année de publication de son article le plus ancien.

❑ **Exercice 4** : *Représentations classiques d'ensembles* d'après CCSE 2021 - MP (Partie 2)
Les programmes de cet exercice doivent être écrits en OCaml.
On s'intéresse dans cet exercice à des structures de données représentant des ensembles d'entiers naturels. On notera $|E|$ le cardinal d'un ensemble E.

■ **Partie I** : Avec une liste d'entiers triés

Dans cette partie uniquement, on implémente un ensemble d'entiers positifs par la liste des ses éléments rangés *dans l'ordre croissant.*. Par exemple la liste `[2; 7; 11]` représente l'ensemble $\{2, 7, 11\}$.

1. Ecrire la fonction `intersection : int list -> int list -> int list` qui prend en argument deux listes d'entiers triés représentant des ensembles et renvoyant leur intersection sous la forme d'une liste triée d'entiers.
2. Ecrire une fonction `succ_list` de signature `int list -> int -> int` prenant en arguments une liste d'entiers *distincts* dans l'ordre croissant et un entier x et renvoyant le successeur de x dans la liste, c'est à dire le plus petit entier strictement supérieur à x de la liste (-1 si cela n'existe pas). Par `succ_list [2; 7; 11] 5` doit renvoyer 7.
3. Donner la complexité de cette fonction dans le pire des cas.

■ **Partie II** : Avec un tableau trié

Soit N un entier naturel strictement positif, fixé pour toute cette partie. On choisit de représenter un ensemble d'entiers E de cardinal $n \leq N$ par un tableau de taille $N + 1$ dont la case d'indice 0 indique le nombre

n d'éléments de E et les cases d'indices 1 à n contiennent les éléments de E *rangés dans l'ordre croissant*, les autres cases étant non significatives. Par exemple, le tableau `[| 3; 2; 5; 7; 9; 1; 14 |]` représente l'ensemble $\{2, 5, 7\}$. En effet, cet ensemble contient 3 éléments car la case d'indice 0 du tableau contient 3 et ces 3 éléments sont 2, 5, 7 (cases d'indice 1 à 3).

1. Pour une telle implémentation d'un ensemble E, décrire brièvement des méthodes permettant de réaliser chacune des opérations ci-dessous (on ne demande pas d'écrire des programmes) et donner leurs complexités dans le pire cas :
   — déterminer le maximum de E,
   — tester l'appartenance d'un élément x à E
   — ajouter un élément x dans E (on suppose que $x \notin E$ et que la taille du tableau est suffisante)
2. Par une méthode dichotomique, écrire une fonction `succ_vect` de signature `int array -> int -> int` prenant en arguments un tableau `t` codant un ensemble E comme ci-dessus et un entier x et renvoyant le successeur de x dans E (-1 si cela n'existe pas.)
3. Calculer la complexité dans le pire cas de la fonction `succ_vect` en fonction de n.
4. Ecrire une fonction `union_vect` de signature `int array -> int array -> int array` prenant en arguments deux tableaux `t_1` et `t_2`, de taille N, codant deux ensembles E_1 et E_2 et renvoyant le tableau correspondant à $E_1 \cup E_2$. On supposera que $|E_1 \cup E_2| \leq N$.

■ **Partie III** : Avec une table de hachage

Soit K un entier naturel strictement positif. On choisit de représenter un ensemble d'entiers E de cardinal n par une table de hachage de taille K avec résolution des collisions par chainage. La fonction de hachage est $h(i) = i \mod K$.

1. Dans le cas où $K = 10$, représenter la table de hachage qui correspond à l'ensemble $\{2, 5, 7, 15\}$.
2. A quelle condition, portant sur K et sur n, la fonction h génère-t-elle forcément des collisions ?
3. Décrire brièvement (on ne demande pas d'écrire un programme) une fonction permettant de renvoyer le maximum d'un ensemble E représenté par une table de hachage. Donner sa complexité.
4. En OCaml, un ensemble est donc représenté par un tableau (de taille K) de listes d'entiers c'est à dire par le type `array int list`. Ecrire la fonction `appartient : int -> int array list -> bool` qui prend en arguments un entier et un ensemble représenté par une table de hachage et renvoie un booléen indiquant si cet élément appartient ou non à l'ensemble.