Consignes. Les calculatrices sont interdites. Numérotez vos feuilles et faites apparaître les questions dans l'ordre du sujet. Si vous repérez une erreur d'énoncé, signalez-la sur votre copie et poursuivez votre composition.

1 Définition de la numérotation de Sosa

1.1 Définition formelle

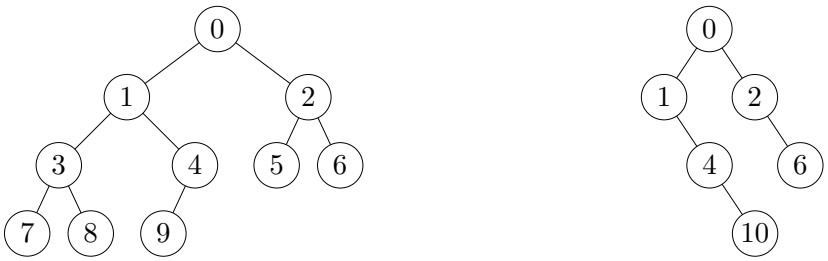
La ***numérotation de Sosa*** permet de donner un identifiant unique à chacun des noeuds d'un arbre binaire. Elle est définie récursivement :

→ Le numéro de Sosa de la racine est 0.

→ Soit x un noeud dont le numéro de Sosa est s, alors le numéro de Sosa du fils gauche de x est $2s+1$ (si ce fils gauche existe) et le numéro de Sosa du fils droit est $2s+2$ (si ce fils droit existe).

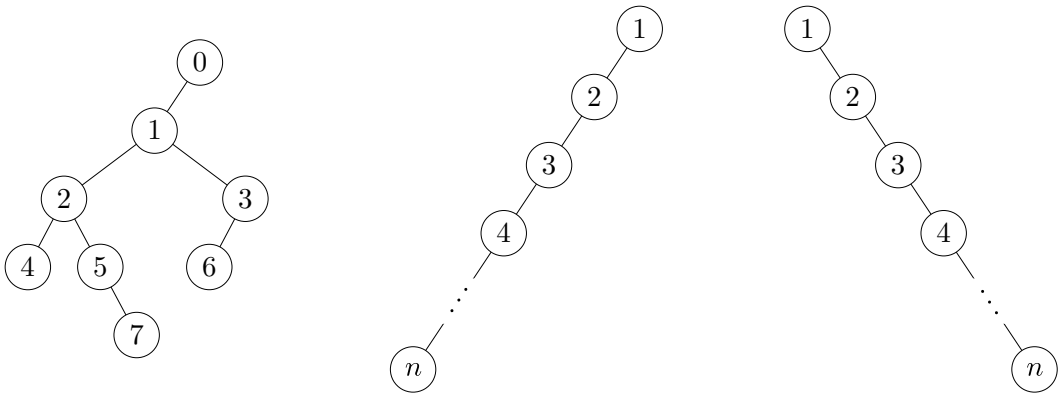
Une conséquence de cette définition est que si x est un noeud différent de la racine dont le numéro de Sosa est s, alors le numéro de Sosa du père de x est $\lfloor (s-1)/2 \rfloor$.

Voici deux arbres binaires où chaque noeud est étiqueté par son numéro de Sosa :



Dans la suite, les étiquettes des arbres ne correspondent plus forcément à la numérotation de Sosa. De plus, on admet que pour tout arbre binaire A et tout $s \in \mathbb{N}$, il existe au plus un noeud de A dont le numéro de Sosa est s.

1. Soit $n \in \mathbb{N}^*$. Réécrire les trois arbres suivants en étiquetant chaque noeud par son numéro de Sosa.



1.2 Définition en OCaml

En OCaml la numérotation de Sosa va permettre de représenter les arbres avec des tableaux via le type « `'a sosa` » :

```
type 'a etiq =
  | Abs
  | E of 'a;;
```

```
type 'a sosa = 'a etiq array;;
```

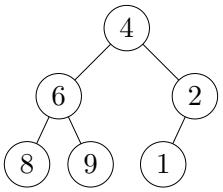
Un tableau « `arb: 'a sosa` » de taille `n` $\in \mathbb{N}$ représente un arbre binaire A lorsque pour tout `i` $\in [\![0, \texttt{n}-1]\!]$, les trois conditions suivantes sont vérifiées :

(i) Si A possède un noeud dont le numéro de Sosa est `i`, alors `arb.(i)` contient « `E e` » où `e` est l'étiquette du noeud.

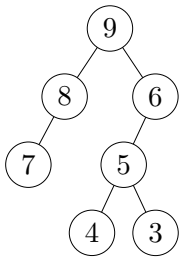
(ii) Si A ne possède pas de noeud dont le numéro de Sosa est `i`, alors `arb.(i)` contient `Abs`.

(iii) Si `n > 0` et `i = n - 1` alors `arb.(i)` ne vaut pas `Abs`.

On veillera à ne pas confondre le nombre de noeuds dans un arbre et la taille du tableau qui le représente. Voici des exemples d'arbres ainsi que leurs représentations par des tableaux de type « `int sosa` » :



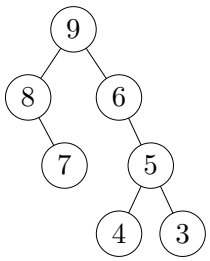
`[|E 4; E 6; E 2; E 8; E 9; E 1|]`




```
[|E 9; E 8; E 6; E 7; Abs; E 5;
Abs; Abs; Abs; Abs; Abs; E 4; E 3|]
```

Sans la condition (iii) énoncée ci-dessus, la représentation ne serait pas unique. En effet, étant donné un tableau « `arb: 'a etiq array` », il suffirait d'ajouter une ou plusieurs cases contenant `Abs` à la fin de `arb` pour obtenir une autre représentation de l'arbre. On admet qu'avec la condition (iii), un arbre est représenté de manière unique.

2. (a) À quel tableau de type « `int sosa` » correspond l'arbre ci-contre ?



   (b) Dessiner l'arbre binaire représenté par le tableau :

`[|E 1; E 9; Abs; E 2; E 8; Abs; Abs; E 3; E 7|]`.

3. (a) À l'aide d'une boucle `while`, écrire une fonction `lastE: 'a etiq array -> int` qui prend en entrée « `tab: 'a etiq array` » et renvoie le plus grand indice `i` tel que `tab.(i) <> Abs`. Si `tab` est vide ou contient uniquement des `Abs`, votre fonction renverra `-1`. Voir les exemples dans le tableau ci-dessous.

   (b) En déduire une fonction `supprAbs: 'a etiq array -> 'a sosa` qui prend en entrée un tableau, et renvoie un autre tableau dans lequel les `Abs` se trouvant à la fin ont été supprimés. Voir les exemples dans le tableau ci-dessous.

`tab`	`[||]` ou `[|Abs; Abs; Abs; Abs|]`	`[|E 1; Abs; E 2; E 3|]` ou `[|E 1; Abs; E 2; E 3; Abs; Abs; Abs; Abs|]`
`lastE tab`	`-1`	`3`
`supprAbs tab`	`[||]`	`[|E 1; Abs; E 2; E 3|]`

À partir de maintenant et jusqu'à la fin du sujet, toutes les variables de type « `'a sosa` » devront respecter la condition (iii). Ainsi, lorsqu'une fonction prend en entrée un tableau de type « `'a sosa` », vous pouvez supposer sans le vérifier que la dernière case ne contient pas `Abs`. De plus, si une fonction renvoie un tableau de type « `'a sosa` », vous devez faire en sorte que la dernière case ne contienne pas `Abs`.

1.3 Tableaux valides

Soit `tab` un tableau de type « `'a sosa` ». On dit que `tab` est ***valide*** lorsqu'il existe un arbre binaire représenté par `tab`.

4. (a) Donner un exemple de tableau « `tab: int sosa` » non valide. Notez que `tab` est de type « `'a sosa` » ce qui implique que sa dernière case ne doit pas contenir `Abs`.
   (b) Écrire une fonction `valide: 'a sosa -> bool` qui teste si l'arbre donné en entrée est valide. Votre fonction devra être de complexité linéaire en la taille du tableau donné en entrée. On expliquera succinctement la procédure utilisée.

À partir de maintenant et jusqu'à la fin du sujet, toutes les variables de type « `'a sosa` » sont des tableaux valides.

2 Ancêtres dans un arbre

En OCaml, la fonction « `List.map: ('a -> 'b) -> 'a list -> 'b list` » prend en entrée une fonction « `f: 'a -> 'b` » ainsi qu'une liste $[a_1 ; a_2 \dots ; a_n]$ et renvoie la liste $[f\ a_1 ; f\ a_2 \dots ; f\ a_n]$.

5. À l'aide de la fonction `List.map`, écrire une fonction `supprE: 'a etiq list -> 'a list` qui prend en entrée « `li: 'a etiq list` » et renvoie une copie de `li` dans laquelle chaque élément de la forme « `E e` » a été remplacé par `e`. Dans le cas où `li` contient le constructeur `Abs`, votre fonction déclenchera une erreur. Par exemple :

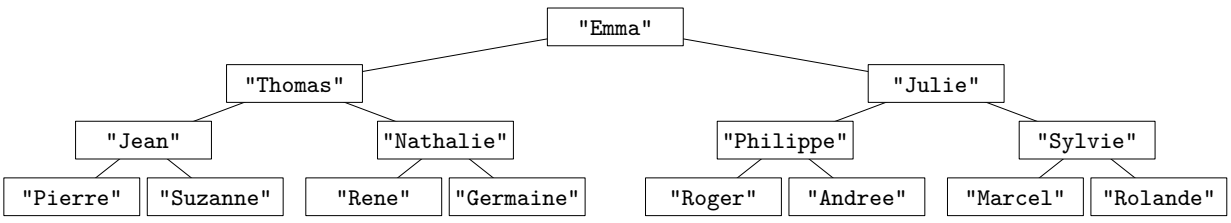
`li`	`[]`	`[E "Philippe"; E "Julie"; E "Emma"]`	`[E "Philippe"; E "Julie"; Abs; E "Emma"]`
`supprE li`	`[]`	`["Philippe"; "Julie"; "Emma"]`	Erreur

Soit x un noeud dans un arbre. On définit récursivement l'ensemble des ancêtres de x :

→ Le père de x est un ancêtre de x.

→ Le père d'un ancêtre de x est un ancêtre de x.

6. Soit A un arbre binaire et « `arb: 'a sosa` » le tableau qui le représente. Écrire une fonction `ancetres: 'a sosa -> int -> 'a list` qui prend en entrée un arbre ainsi qu'un entier s, et renvoie la liste des étiquettes des ancêtres du noeud dont le numéro de Sosa est s. Votre fonction déclenchera une erreur si s ne correspond à aucun noeud. De plus, elle devra s'exécuter en un temps linéaire en la profondeur du noeud dont on cherche les ancêtres. L'ordre des éléments de la liste en sortie n'a pas d'importance. Par exemple avec :



on obtient :

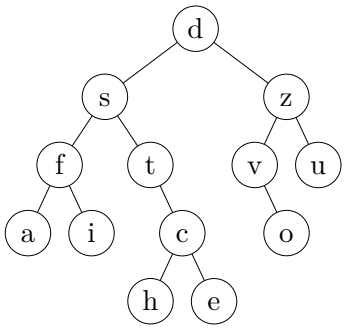
`s`	`0`	`11`	`2`	`4`
`ancetres arb s`	`[]`	`["Philippe"; "Julie"; "Emma"]`	`["Emma"]`	`["Thomas"; "Emma"]`

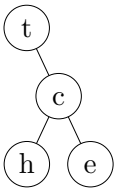
3 Hauteur d'un arbre – version 1

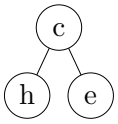
Dans toute cette partie, A est un arbre binaire représenté par une variable « `arb: 'a sosa` » et `n` est la taille du tableau `arb`.

7. Rappeler la définition de la hauteur d'un arbre binaire.

Soit `i` $\in \mathbb{N}$. On appelle sous-arbre d'indice `i` de A l'arbre ayant pour racine le noeud dont le numéro de Sosa est `i`. Par exemple, pour tout arbre binaire, le sous-arbre d'indice 0 est l'arbre lui-même, le sous-arbre d'indice 1 est le sous-arbre gauche et le sous-arbre d'indice 2 est le sous-arbre droit. De plus, si on définit A_1, A_2, A_3 et A_4 les arbres ci-dessous, alors A_2 est le sous-arbre d'indice 4 de A_1, A_3 est le sous-arbre d'indice 10 de A_1 et A_4 est le sous-arbre d'indice 6 de A_1 :

$A_1 =$ 

$A_2 =$ 

$A_3 =$ 

$A_4 =$ 

Enfin, par convention, si `i` $\geqslant$ `n` ou `arb.(i) = Abs`, alors on considère que le sous-arbre d'indice `i` de A est vide.

8. Soit A un arbre binaire représenté par une variable « `arb: 'a sosa` ». Afin de déterminer la hauteur de A, on propose de parcourir tous ses noeuds.
   (a) Écrire une fonction récursive `hautAux: 'a sosa -> int -> int` qui prend en entrée `arb` ainsi qu'un entier `i` $\in \mathbb{N}$, et renvoie la hauteur du sous-arbre d'indice `i` de A.
   (b) En déduire une fonction `haut: 'a sosa -> int` qui renvoie la hauteur de A.
   (c) Déterminer le nombre exact d'appels à la fonction `hautAux` lors de l'évaluation de « `haut arb` ». Le montrer à l'aide d'une récurrence.
   (d) En déduire la complexité en temps de la fonction `haut`. Justifier.

4 Représentation alternative des arbres binaires

Définissons un second type pour représenter les arbres binaires :

```
type 'a bin =
  | Vide
  | N of 'a * 'a bin * 'a bin;;
```

4.1 Parcours en largeur

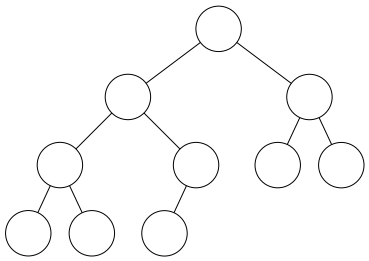
9. Sans convertir vers le type `'a sosa`, écrire une fonction `larg1: 'a bin -> 'a list` qui calcule le parcours en largeur d'un arbre. Vous devez utiliser la méthode vue en cours en utilisant une file.
10. Sans convertir vers le type `'a bin`, écrire une fonction `larg2: 'a sosa -> 'a list` qui calcule le parcours en largeur d'un arbre. On expliquera succinctement la procédure utilisée.

4.2 Conversions

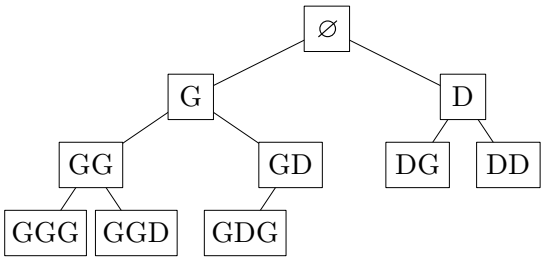
11. Écrire une fonction `binOfSosa: 'a sosa -> 'a bin` qui prend en entrée un arbre représenté par le type « `'a sosa` », et renvoie sa représentation par le type « `'a bin` ». Votre fonction doit être de complexité linéaire en la taille du tableau en entrée.
    Indication. On pourra écrire une fonction intermédiaire similaire à celle de la question 8a.
12. (a) Écrire une fonction `pref : ('a -> int -> unit) -> 'a bin -> unit` qui prend en entrée une fonction « `f: 'a -> int -> unit` » ainsi qu'un arbre « `arb: 'a bin` » et lance un parcours préfixe de `arb`. À chaque fois qu'un noeud x est visité, vous devez exécuter l'instruction « `f e s` » où `e` et `s` sont respectivement l'étiquette et le numéro de Sosa de x.
    (b) En déduire une fonction `sosaOfBin: 'a bin -> 'a sosa` qui prend en entrée un arbre représenté par le type « `'a bin` », et renvoie sa représentation par le type « `'a sosa` ». Votre fonction doit être de complexité linéaire en la taille du tableau renvoyé.

4.3 Numérotation de Sosa en base 2

Considérons les arbres A et B suivants :



Arbre A



Arbre B

Les arbres A et B ont la même forme. Dans l'arbre B, chaque noeud est étiqueté par le chemin qu'il faut emprunter depuis la racine pour accéder au noeud. La lettre G indique que le noeud se trouve dans le sous-arbre gauche et la lettre D indique que le noeud se trouve dans le sous-arbre droit. Par exemple, pour accéder au noeud dont le numéro de Sosa est 9 à partir de la racine, il faut aller dans le sous-arbre gauche, puis dans le sous-arbre droit et enfin dans le sous-arbre gauche.

13. (a) Dessiner l'arbre A en étiquetant chaque noeud x par l'écriture en base 2 de $s+1$ où s est le numéro de Sosa de x.
    (b) Que remarquez vous en comparant l'arbre obtenu dans la question précédente et l'arbre B ? Montrez que c'est le cas pour tout arbre binaire.
14. Déduire de la question précédente une fonction `etiq: 'a bin -> int -> 'a` qui prend en entrée un arbre binaire ainsi qu'un entier $s \in \mathbb{N}^*$, et renvoie l'étiquette du noeud dont la numérotation de Sosa est s. Si aucun noeud n'a pour numérotation de Sosa s, votre fonction déclenchera une erreur. La complexité devra être linéaire en la profondeur du noeud.

5 Hauteur d'un arbre – version 2

En fait, la hauteur d'un arbre « `arb: 'a sosa` » peut être calculée beaucoup plus rapidement que par la fonction `haut`. En effet, elle est directement liée à la taille du tableau `arb`.

15. Soit A un arbre binaire non vide représenté par une variable « `arb: 'a sosa` ». Soit `n` $\in \mathbb{N}^*$ la taille du tableau `arb` et $h \in \mathbb{N}$ la hauteur de A. Montrer que $h = \left\lfloor \log_2(n) \right\rfloor$.

16. (a) Soit $n \in \mathbb{N}^*$. Quel est le lien entre $\left\lfloor \log_2(n) \right\rfloor$ et le nombre de bits dans la représentation de n en base 2 ? Justifier.
    (b) Sans utiliser la fonction `log` de OCaml, écrire une fonction `nbBits: int -> int` qui prend en entrée un entier $n \in \mathbb{N}^*$, et renvoie le nombre de bits dans la représentation de n en base 2.

17. (a) Déduire des questions précédentes une fonction `hautBis: 'a sosa -> int` qui renvoie la hauteur de l'arbre donné en argument.
    (b) Quelle est la complexité en temps de la fonction `hautBis` ?

6 Généralisation pour des arbres d'arité bornée

Soit $k \in \mathbb{N}^*$ un entier. On souhaite généraliser la représentation sous forme de tableaux utilisée dans ce sujet au cas des arbres d'arité au plus k. On note $\mathcal{T}$ l'ensemble des arbres d'arité au plus k défini récursivement :

→ L'arbre vide appartient à $\mathcal{T}$.

→ Si $e \in E$ est une étiquette et $(A_1, \dots A_k) \in \mathcal{T}^k$ alors le $(k+1)$-uplet $(e, A_1, \dots, A_k)$ appartient à $\mathcal{T}$.

18. En utilisant la même idée que pour le type « `'a sosa` », proposez une manière de représenter les éléments de $\mathcal{T}$ avec des tableaux. Afin de minimiser la mémoire utilisée, vous devez faire en sorte pour tout `i` $\in \mathbb{N}$, il existe un tableau `tab` représentant un arbre de $\mathcal{T}$ tel que `tab.(i)` ne vaut pas `Abs`.