Récupérez le fichier annexe disponible sur la page du cours :

`https://informatique-lhp.fr/opt-mpsi.html`

Dans ce fichier sont définis des arbres de types « `'a bin` », « `'a abs` » et « `'a gen` » pour tester vos fonctions.

Exercice 1. Arbres binaires équilibrés

1. Dessiner les arbres `bin_B1`, `bin_B2` et `bin_B3` du fichier annexe. Quel est le type de ces 3 objets ?

Un arbre binaire est dit ***équilibré*** lorsque pour chaque noeud, la hauteur du sous-arbre gauche et du sous-arbre droit diffèrent au plus de 1.

2. `bin_B1`, `bin_B2` et `bin_B3` sont-ils équilibrés ?
3. Écrire une fonction de type « `'a bin -> bool` » qui détermine si un arbre binaire est équilibré. La complexité devra être linéaire en le nombre de noeuds de l'arbre, c'est à dire que chaque noeud doit être exploré au plus une fois.
4. (a) Soit A un arbre binaire équilibré non vide, soit h_G la hauteur du sous-arbre gauche et h_D la hauteur du sous-arbre droit. Montrer que $h_G + h_D \geqslant 2\max(h_G, h_D) - 1$.
   (b) Montrez que pour tout $h \geqslant -1$ et $n \geqslant 0$, si A est un arbre binaire équilibré de hauteur h avec n noeuds alors $h \leqslant \dfrac{3}{2}\log_2(n+1)$. On pourra utiliser l'inégalité :

$$\forall x \in \mathbb{R}_+^*, \forall y \in \mathbb{R}_+^* : \frac{\log_2(x) + \log_2(y)}{2} \leqslant \log_2\left(\frac{x+y}{2}\right)$$

   (c) La réciproque de la question précédente est elle vraie ?

Exercice 2. Parcours d'arbres binaires stricts

1. Dessiner les arbres `abs_A1`, `abs_A2` et `abs_A3` du fichier annexe.

Parcours marqués. Dans cette première partie, un parcours d'arbre est représenté par une liste d'étiquettes dans laquelle deux constructeurs `F` et `N` sont utilisés pour indiquer si l'étiquette est liée à une feuille ou à un noeud interne :

```
type 'a elem_parc =
  | F of 'a
  | N of 'a;;
```

```
type 'a parc =
  'a elem_parc list;;
```

On dira qu'un parcours est ***marqué*** pour indiquer qu'il est de type « `'a elem_parc` », et donc qu'on peut y distinguer les feuilles des noeuds internes. La terminologie "marqué" n'est pas standard, ne l'utilisez pas en dehors de cet exercice. Par exemple, le parcours préfixe marqué de l'arbre `abs_A1` vaut :

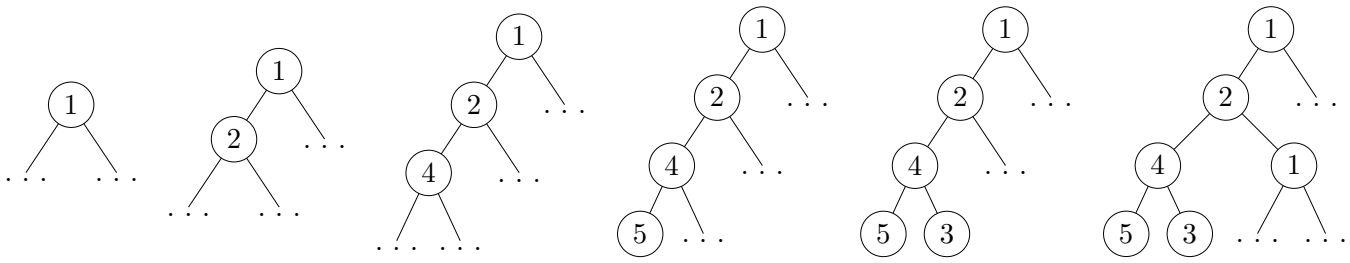
```
[N 1; N 2; N 4; F 5; F 3; N 1; F 10; F 0; N 2; F 1; F 2]
```

2. Trouver 2 arbres binaires stricts différents ayant le même parcours infixe marqué.
3. Écrire les fonctions :

```
get_parc_pre: 'a abs -> 'a parc
get_parc_inf: 'a abs -> 'a parc
get_parc_post: 'a abs -> 'a parc
```

qui prennent en entrée un arbre binaire strict et renvoient ses parcours préfixe, infixe et postfixe marqués. Vos fonctions devront avoir une complexité linéaire en le nombre de noeuds dans l'arbre.

Étant donnés `abs_A1` et `abs_A2` deux arbres binaires stricts de type « `'a abs` », il est possible de montrer que si `abs_A1` et `abs_A2` sont différents alors « `get_parc_pre abs_A1` » et « `get_parc_pre abs_A2` » sont différents. En fait, étant donné un parcours préfixe marqué, on peut reconstruire l'arbre correspondant. Par exemple, les six premiers éléments du parcours préfixe marqué de `abs_A1` sont `N 1`, `N 2`, `N 4`, `F 5`, `F 3` et `N 1`. On peut alors en déduire que `abs_A1` est de la forme :



En continuant ce raisonnement avec tous les éléments du parcours préfixe marqué, on peut reconstruire entièrement l'arbre.

4. Écrire une fonction de type « `'a parc -> 'a abs` » qui prend en entrée un parcours préfixe marqué et renvoie l'arbre binaire strict associé.

Parcours non marqués. On représente maintenant le parcours d'un arbre par une liste d'étiquettes de type « `'a list` ». Ainsi, étant donné le parcours, on ne peut plus distinguer les feuilles des noeuds internes. Dans cet exercice, on dira qu'un parcours est ***non marqué*** pour indiquer qu'il est de type « `'a list` », et donc qu'on ne peut pas y distinguer les feuilles des noeuds internes. Par exemple, le parcours préfixe non marqué de l'arbre `abs_A1` vaut :

```
[1; 2; 4; 5; 3; 1; 10; 0; 2; 1; 2]
```

5. À l'aide de la fonction `List.map` et des fonctions de la question 3, écrire les fonctions :

```
get_parc_pre2: 'a abs -> 'a list
get_parc_inf2: 'a abs -> 'a list
get_parc_post2: 'a abs -> 'a list
```

   qui prennent en entrée un arbre binaire strict et renvoient ses parcours préfixe, infixe et postfixe non marqués. Si vous ne vous souvenez plus de la fonction `List.map`, utilisez l'aide en ligne d'OCaml.

6. Montrer qu'il existe 2 arbres binaires stricts différents de type « `'a abs` » ayant le même parcours préfixe, le même parcours infixe et le même parcours postfixe non marqués (les deux arbres sont les mêmes pour tous les parcours).

On suppose maintenant que toutes les étiquettes d'un arbre sont distinctes deux à deux.

7. Montrer qu'il existe 2 arbres $A_1 \neq A_2$ tels que :
   → Chacun des deux arbres est binaire strict avec des étiquettes distinctes deux à deux.
   → Les parcours préfixes non marqués de A_1 et A_2 sont identiques.

   Idem pour les parcours infixes et postfixes non marqués.

8. Montrez que si $A_1 \neq A_2$ sont deux arbres tels que :
   → Chacun des deux arbres est binaire strict avec des étiquettes distinctes deux à deux.

   Alors les parcours préfixes non marqués de A_1 et A_2 sont différents ou bien les parcours infixes non marqués de A_1 et A_2 sont différents.

9. Écrire une fonction de type « `'a list -> 'a list -> 'a abs` » qui prend en entrée le parcours préfixe non marqué et le parcours infixe non marqué d'un arbre binaire strict A, et qui renvoie A. On testera la fonction avec avec `abs_A4`, `abs_A5`, `abs_A6` (ce sont les mêmes exemples que précédemment mais les étiquettes en double ont été renommées).

Certaines questions sont à rédiger sur feuille. Si vous le souhaitez, vous pouvez me rendre ces questions lors de la prochaine séance de TP le 07/06/2024. En revanche, les programmes doivent être envoyés par email comme habituellement.

Exercice 3. Parcours d'un arbre d'arité quelconque

1. Dessiner les arbres `gen_G1` et `gen_G2` du fichier annexe.

Comme pour les arbres binaires, le parcours préfixe consiste à visiter d'abord la racine puis les sous-arbres. De même, le parcours postfixe consiste à visiter d'abord les sous-arbres puis la racine. En revanche, il n'existe pas de parcours infixe pour les arbres d'arité quelconque.

2. Écrire une fonction « `parc_pre_gen: 'a gen -> 'a list` » qui prend en entrée un arbre et renvoie son parcours préfixe. Vous devez obtenir :

```
[1;5;2;7;0;9;3;4;6;8]        ['c';'m';'p';'s';'i';'d';'e';'u';'x';'a';'b']
```

3. Idem pour le parcours postfixe : « `parc_post_gen: 'a gen -> 'a list` ». Vous devez obtenir :

```
[2;5;0;9;7;4;6;8;3;1]        ['s';'i';'p';'d';'u';'x';'e';'b';'a';'m';'c']
```

4. Idem pour le parcours en largeur : « `parc_larg_gen: 'a gen -> 'a list` ». Vous devez obtenir :

```
[1;5;7;3;2;0;9;4;6;8]        ['c';'m';'p';'d';'e';'a';'s';'i';'u';'x';'b']
```

Exercice 4. Nombre de Strahler

Le ***nombre de Strahler*** d'un arbre binaire strict A est défini par induction :

→ Si A est réduit à une feuille alors son nombre de Strahler est 1.

→ Sinon, soient s_G le nombre de Strahler du sous-arbre gauche et s_D celui du sous-arbre droit. Si $s_G = s_D$, le nombre de Strahler de A est $s_G + 1$, sinon c'est $\max(s_G, s_D)$.

Pour information, étant donnée une expression arithmétique représentée sous la forme d'un arbre binaire strict A, le nombre de Strahler de A est le nombre de registres (cases mémoires) nécessaires pour évaluer l'expression arithmétique.

1. Écrire une fonction de type « `nb_strahler: 'a abs -> int` » qui prend en entrée un arbre et renvoie son nombre de Strahler.
2. Soit $h \in \mathbb{N}$. Quel est le nombre de Strahler maximal pour un arbre de hauteur h ? On attend une démonstration détaillée.
3. Soit $h \in \mathbb{N}$. Quel est le nombre de Strahler minimal pour un arbre de hauteur h ? On attend une démonstration détaillée.

Exercice 5. Arbres de Fibonacci (exercice facultatif)

Les arbres de Fibonacci $F_0, F_1, \ldots$ sont des arbres binaires stricts définis par récurrence :

→ $F_0 = F_1$ est un arbre réduit à une feuille

→ Pour $n \geqslant 2$, F_n est l'arbre dont le sous-arbre gauche est F_{n-1} et le sous-arbre droit est F_{n-2}.

Étant donné qu'on ne s'intéresse pas aux étiquettes de l'arbre, on pourra utiliser le type « `unit abs` » pour représenter les arbres de Fibonacci.

1. Sur feuille, dessiner F_0, F_1, F_2, F_3, F_4 et F_5.
2. Écrire une fonction « `make_fibo: int -> unit abs` » qui prend en entrée un entier n et renvoie F_n. Vérifiez que votre programme parvient à calculer F_{1000}.

3. Soit $(f_n)_{n\in\mathbb{N}}$ la suite de Fibonacci définie par :

$$f_0 = f_1 = 1 \qquad\qquad f_{n+2} = f_n + f_{n+1} \text{ pour tout } n \in \mathbb{N}$$

   Calculez le nombre de feuilles, le nombre de noeuds internes et le nombre de noeuds de F_n en fonction de f_n.

4. Montrer qu'un arbre de Fibonacci est équilibré.

5. Déterminer en le justifiant le nombre de Strahler d'un arbre de Fibonacci.