TP n°3 - Introduction aux arbres binaires

À rendre au plus tard le 31 mars à 20h

- Programmer en OCaml dans un seul fichier intitulé `TP03_NOM_Prenom.ml` (où `NOM` est votre nom de famille et `Prenom` votre prénom, sans utiliser d'accent).
- Votre script doit pouvoir fonctionner sans intervention extérieure. Avant de rendre votre TP, il est conseillé redémarrer l'interpréteur puis de vérifier que votre script s'exécute sans erreur.
- Pour les questions qui ne demandent pas de programmation, on donnera la réponse dans un commentaire.
- Respecter scrupuleusement les noms des fonctions donnés.
- Rendre votre fichier avant le mercredi 31 mars à 20h en le déposant à l'adresse : `https://maths.mlong.fr/nextcloud/index.php/s/Gr4cEK9xcaYq9y3`

1 Un peu de vocabulaire

Un *arbre binaire* est une structure de données hiérarchique dont les éléments sont appelés des *nœuds*, chaque nœud possédant au plus deux éléments au niveau inférieur, appelés *fils gauche* et *fils droit* du nœud (qui est alors le *père* de ses éventuels fils).

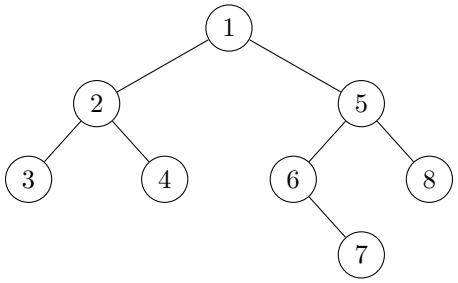

Figure 1 – Un arbre binaire

Le nœud initial d'un arbre binaire non vide est appelé la *racine* de l'arbre. Un élément n'ayant aucun fils est appelé une *feuille* ; un nœud qui possède au moins un fils est appelé un *nœud interne*.

Pour l'arbre de la figure 1, la racine est le nœud 1, les feuilles sont les nœuds 3, 4, 7 et 8.

La *profondeur* d'un nœud est sa distance par rapport à la racine. La *hauteur* d'un arbre est la profondeur maximale de ses nœuds. [1]

Dans l'arbre de la figure 1, la profondeur du nœud 6 est 2 ; la hauteur de l'arbre est 3.

Chaque nœud est la racine d'un arbre constitué de lui-même et de sa descendance : on parle alors de *sous-arbre* de l'arbre initial.

Enfin, un *arbre binaire strict* est un arbre binaire tel que chaque nœud interne possède exactement deux fils.

2 Implémentation et fonctions élémentaires

Dans un premier temps, pour implémenter les arbres binaires, on utilisera le type suivant :

```
type 'a arbre =
  | Vide
  | Noeud of 'a * 'a arbre * 'a arbre
;;
```

Q1 Définir en OCaml l'arbre `mon_arbre` correspondant à l'arbre de la figure 1. On pensera par la suite à utiliser cet arbre pour tester les différentes fonctions écrites.

1. Il existe en réalité deux définitions de la hauteur d'un arbre ; nous choisirons celle-ci pour ce TP.

Q2 Écrire une fonction `est_strict : 'a arbre -> bool` prenant en argument un arbre et renvoyant `true` s'il s'agit d'un arbre binaire strict, `false` sinon.

Q3 Écrire une fonction `nb_noeuds : 'a arbre -> int` et une fonction `nb_feuilles : 'a arbre -> int` prenant en argument un arbre et renvoyant respectivement le nombre de nœuds et le nombre de feuilles de l'arbre.

Q4 Écrire une fonction `hauteur : 'a arbre -> int` prenant en argument un arbre et renvoyant sa hauteur.

Q5 Écrire une fonction `recherche : 'a -> 'a arbre -> bool` prenant en argument un élément et un arbre et renvoyant `true` si l'élément est présent dans l'arbre, `false` sinon.

3 Quelques fonctions sur des arbres étiquetés par des entiers

On considère ici des arbres dont les étiquettes sont de type `int`.

Q6 Écrire une fonction `maxi : 'a arbre -> 'a` prenant en argument un arbre et renvoyant le maximum de ses étiquettes.

Q7 Écrire une fonction `somme : int arbre -> int` prenant en argument un arbre et renvoyant la somme de ses étiquettes.

Q8 On considère pour chaque branche de l'arbre (i.e un chemin de la racine à une feuille) la somme des étiquettes de ses nœuds. Écrire une fonction `somme_max_branche : int arbre -> int` donnant la somme maximale des branches d'un arbre.

4 Diamètre d'un arbre

Étant donné un arbre binaire $\mathcal{A}$, on appelle *diamètre* de $\mathcal{A}$ la longueur (i.e le nombre d'arêtes) d'un plus long chemin dans $\mathcal{A}$.

Q9 Quel est la longueur d'un plus long chemin passant par la racine ?

Q10 Écrire une fonction `diametre : 'a arbre -> int` calculant le diamètre d'un arbre binaire ; on distinguera trois cas.

5 Peignes

On considère désormais des arbres binaires stricts non vides dont les nœuds internes ne sont pas étiquetés et les feuilles sont étiquetés par des entiers. On utilisera donc le type suivant :

```
type arbre =
   |Feuille of int
   |Noeud of arbre * arbre ;;
```

On dit qu'un arbre est un *peigne* si tous les noeuds à l'exception éventuelle de la racine ont au moins une feuille pour fils. On dit qu'un peigne est un *peigne strict* si sa racine a au moins une feuille pour fils, ou s'il est réduit à une feuille. On dit qu'un peigne est *rangé* si le fils droit d'un noeud est toujours une feuille. Un arbre réduit à une feuille est un peigne rangé.

Q11 Donner la représentation en OCaml du peigne à 5 feuilles de la figure 2. Est-il strict ? Est-il rangé ?

Q12 Quelle est la hauteur d'un peigne rangé à n feuilles ? On justifiera la réponse.

Q13 Ecrire une fonction `est_range : arbre` $\to$ `bool` qui renvoie `true` si l'arbre donné en argument est un peigne rangé.

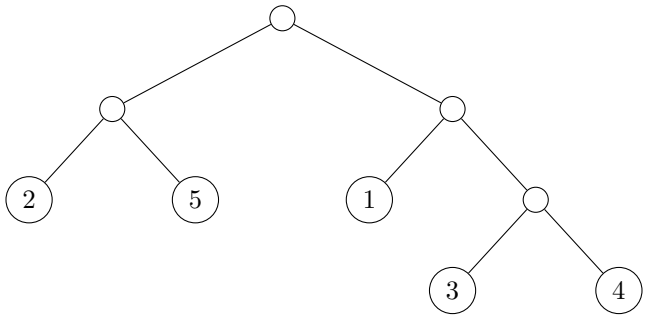

FIGURE 2 – Un peigne à cinq feuilles

Q14 Ecrire une fonction `est_peigne_strict : arbre → bool` qui renvoie `true` si l'arbre donné en argument est un peigne strict. En déduire une fonction `est_peigne : arbre → bool` qui renvoie `true` si l'arbre donné en argument est un peigne.

Q15 On souhaite ranger un peigne donné. Supposons que le fils droit N de sa racine ne soit pas une feuille. Notons A_1 le sous-arbre gauche de la racine, f l'une des feuilles du noeud N et A_2 l'autre sous-arbre du noeud N. On va utiliser l'opération de *rotation* qui construit un nouveau peigne où

- le fils droit de la racine est le sous-arbre A_2 ;
- le fils gauche de la racine est un noeud de sous-arbre gauche A_1 et de sous-arbre droit la feuille f.

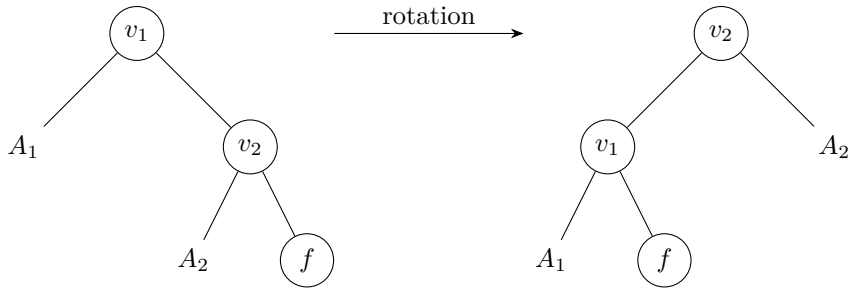

FIGURE 3 – Une rotation

(a) Donner la représentation en OCaml du peigne obtenu après une rotation sur le peigne de la figure 2.

(b) Écrire une fonction `rotation : arbre → arbre` qui effectue l'opération décrite ci-dessus. La fonction renverra l'arbre initial si une rotation n'est pas possible.

(c) Écrire une fonction `rangement : arbre → arbre` qui range un peigne donné en argument, c'est à dire qui renvoie un peigne rangé ayant les mêmes feuilles que celui donné en argument. La fonction renverra l'arbre initial si celui-ci n'est pas un peigne.

(d) Prouver que la fonction `rangement` termine.