Dans tout ce TP le type `'a btree` fait référence au type OCaml ci-dessous défini.

```
type 'a btree =
| Empty
| Node of 'a * 'a btree * 'a btree
```

Exercice 1 : Jeux de tests

Dans cet exercice on se donne un (mauvais) générateur uniforme d'une permutation de $[\![1, n]\!]$. Cette fonction **devra** être utilisée dans la suite du TP pour tester vos fonctions.

Q. 1 Définir une fonction `remove_nth : 'a list -> int -> 'a * 'a list` prenant en arguments une liste l et un entier i et retournant : l'élément se trouvant en position i dans l et la liste l privée de l'élément se trouvant en position i.

Q. 2 En déduire une fonction `permutation_list : 'a list -> 'a list` générant une permutation de la liste passée en argument. L'algorithme sera le suivant : étant donnée une liste l, on choisit un élément uniformément dans la liste l. Cet élément sera le premier élément de la permutation résultat, on recommence récursivement sur la liste l privée de l'élément choisi. On s'aidera de la fonction `Random.int n` retournant un entier choisi uniformément dans l'intervalle $[\![0, \mathtt{n} - 1]\!]$.

Q. 3 En déduire une fonction `permutation : int -> int list` prenant en argument un entier n et calculant une permutation de $[\![1, n]\!]$.

Exercice 2 : Arbres binaires de recherches

Q. 1 Donner une fonction `mem : 'a btree -> 'a -> bool` permettant de tester si un élément apparaît dans un ABR.

Q. 2 Donner une fonction `inserer : 'a btree -> 'a -> 'a btree` permettant d'insérer un élément dans un ABR.

Q. 3 Donner une fonction `smin : 'a btree -> 'a -> ('a * 'a btree)` permettant simultanément de trouver le minimum d'un ABR et de renvoyer l'ABR privé de ce minimum.

Q. 4 En déduire une fonction `supprimer: 'a btree -> 'a -> 'a btree` permettant de supprimer un élément dans un ABR.

Q. 5 Donner une constante `vide : 'a btree` permettant de représenter l'ensemble vide.

Q. 6 Définir une fonction `est_abr : 'a btree -> bool` permettant de tester qu'un arbre binaire est bien de recherche. Votre fonction devra avoir une complexité en $\Theta(n)$ où n est le nombre d'éléments stockés dans la structure.

Q. 7 Proposer un jeu de test conséquent des fonctions ci-avant. On pourra par exemple générer un ABR par insertion successive des entiers d'une permutation de l'ensemble $\{2, 4, \ldots, 100\}$, on vérifiera à chaque étape que l'arbre obtenu est bien un ABR. On testera ensuite l'appartenance de tous les entiers de l'intervalle $[\![1, 100]\!]$. Pour finir on demandera la suppression de tous les entiers de l'intervalle $[\![1, 100]\!]$, et on vérifiera que la structure obtenue est bien vide.

Exercice 3 : Type de données ensemble

Cette exercice fait suite au précédent. On souhaite enrichir l'implantation, du type de donnée abstrait ensemble, proposée dans l'exercice ci-avant.

Q. 1 Définir une fonction `for_all : ('a -> bool) -> 'a btree -> bool` permettant de tester si toutes les étiquettes d'un ABR satisfont le prédicat passé en argument. Votre implantation **devra** faire intervenir un mécanisme de lever et de rattrapage d'exceptions.

Q. 2 Définir une fonction `exists : ('a -> bool) -> 'a btree -> bool` permettant de tester si une des étiquettes d'un ABR satisfait le prédicat passé en argument. Votre implantation **devra** faire intervenir un mécanisme de lever et de rattrapage d'exceptions.

Q. 3 Définir une fonction `subset : 'a btree -> 'a btree -> bool` permettant de tester si un ensemble, représenté par un ABR, est inclus dans un autre ensemble.

Q. 4 Définir une fonction `fold : ('a -> 'b -> 'b) -> 'a btree -> 'b -> 'b` telle que `(fold f t b)` calcule la valeur :

$$\texttt{(f } x_n \texttt{ (f } x_{n-1} \texttt{ (… (f } x_1 \texttt{ a)…))}$$

lorsque `t` représente l'ensemble $\{x_1, \ldots, x_{n-1}, x_n\}$. On itérera sur les étiquettes dans l'ordre induit par la relation d'ordre totale utilisée pour le stockage des éléments dans l'ABR.

Q. 5 Définir une fonction `cardinal : 'a btree -> int` donnant le cardinal de l'ensemble représenté par l'ABR passé en argument.

Q. 6 Définir une fonction `print : 'a btree -> unit` permettant l'affichage de l'ensemble des étiquettes apparaissant dans un ABR. Vous afficherez les éléments dans l'ordre de la relation d'ordre totale utilisée pour le stockage des éléments dans l'ABR. Par exemple :

```
{1, 3, 7, 8, 10,}
```

Q. 7 Définir une fonction `union: 'a btree -> 'a btree -> 'a btree` permettant de calculer un ABR représentant l'union des ensembles représentés par les deux ABR passés en arguments.

Q. 8 Définir une fonction `intersection: 'a btree -> 'a btree -> 'a btree` permettant de calculer un ABR représentant l'intersection des ensembles représentés par les deux ABR passés en arguments.

Exercice 4 : ABR équilibrés

Q. 1 Proposer un algorithme permettant le calcul, à partir d'une liste l d'étiquettes, d'un ABR complet dont les étiquettes sont les éléments de l. Quelle est la complexité de votre algorithme ?

Q. 2 Implanter votre algorithme en OCaml.

Exercice 5 : Insertion en racine

Q. 1 Proposer un algorithme permettant l'insertion en racine dans un ABR. La complexité de votre algorithme devra être en $\Theta(h(t))$ où $h(t)$ est la hauteur de l'arbre dans lequel on souhaite faire une insertion.

Q. 2 Implanter votre algorithme en OCaml

Exercice 6 : Arbres Rouges et Noirs

Q. 1 Définir un type `'a rbtree` permettant la représentation d'un arbre rouge noir.

Q. 2 Définir une fonction `is_rb_tree: 'a rbtree -> bool` permettant de tester si un `'a rbtree` est bien un arbre rouge noir.

Q. 3 Implanter l'algorithme d'insertion dans un arbre rouge noir.