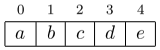
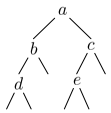
Partie II. Tableaux flexibles

Cette partie doit être traitée dans le langage OCaml.

Dans ce problème, on représente un tableau par un arbre binaire. Si le tableau est vide, il est représenté par l'arbre vide. Sinon, l'élément d'indice 0 du tableau est stocké à la racine de l'arbre, le sous-arbre gauche contient une représentation des éléments d'indices de la forme $2i+1$ et le sous-arbre droit contient une représentation des éléments d'indices de la forme $2i+2$. Ainsi, le tableau de taille 5



est représenté par l'arbre binaire



avec l'élément a d'indice 0 à la racine, le sous-tableau $[b, d]$ des éléments d'indices impairs récursivement organisé pour former le sous-arbre gauche et le sous-tableau $[c, e]$ des éléments d'indices pairs récursivement organisé pour former le sous-arbre droit.

Question 2.1. Dessiner l'arbre binaire correspondant à un tableau de taille 12.

Mise en œuvre en OCaml. On suppose que les éléments sont d'un type `elt` donné qui n'est pas précisé. Un tableau flexible est représenté par une valeur du type OCaml suivant :

```
type tree = E | N of tree * elt * tree
```

La valeur `E` représente l'arbre binaire vide. Une valeur $\texttt{N}(\ell, x, r)$ représente un arbre binaire non vide, avec la valeur x à la racine, un sous-arbre gauche ℓ et un sous-arbre droit r. La taille d'un arbre t, notée $|t|$, est son nombre de nœuds. On a donc :

$$\begin{aligned} |\texttt{E}| &= 0 \\ |\texttt{N}(\ell, x, r)| &= 1 + |\ell| + |r| \end{aligned}$$

Question 2.2. Écrire une fonction `get: int -> tree -> elt` qui reçoit en arguments un entier i et un tableau flexible t, avec $0 \leq i < |t|$, et qui renvoie l'élément d'indice i du tableau représenté par t. La complexité doit être proportionnelle à la hauteur de t. On ne demande pas de la justifier.

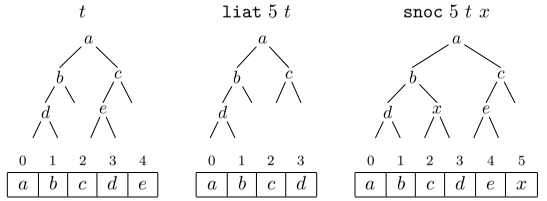
Tournez la page S.V.P.

Invariant structurel. On remarque que dans un tableau flexible, le sous-arbre gauche a toujours au moins autant de nœuds que le sous-arbre droit, car il y a au moins autant d'indices de la forme $2i+1$ que d'indices de la forme $2i+2$. Plus précisément, *pour tout sous-arbre* $\texttt{N}(\ell, x, r)$ d'un tableau flexible, on a

$$|r| \leq |\ell| \leq |r| + 1 \tag{1}$$

Question 2.3. Montrer que la hauteur d'un tableau flexible de taille n est en $\mathcal{O}(\log n)$.

Agrandir/rétrécir du côté droit. Notre objectif est de réaliser quatre opérations pour respectivement agrandir ou rétrécir le tableau d'une case sur l'une ou l'autre de ses extrémités. Commençons par le côté droit, c'est-à-dire celui des indices les plus grands. L'opération `liat` n t supprime le dernier élément d'un (arbre représentant le) tableau t de taille n. L'opération `snoc` n t x ajoute un élément x à la fin d'un (arbre représentant le) tableau t de taille n. Si on reprend l'exemple du tableau donné en introduction, ces deux opérations donnent respectivement les arbres suivants :



En dessous de chaque arbre, on a dessiné le tableau qu'il représente. Voici par exemple le code de la fonction `liat` :

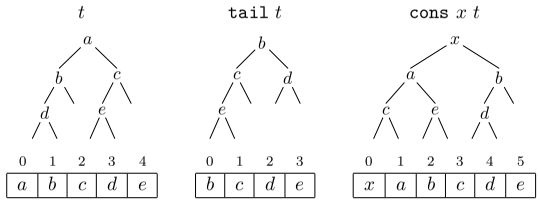
```
let rec liat n = function
  | N(E, _, E) -> E
  | N(l, x, r) -> if n mod 2 = 0 then N(liat (n / 2) l, x, r)
                                 else N(l, x, liat (n / 2) r)
  | E -> assert false
```

Question 2.4. Pour un tableau flexible t de taille n, donner la complexité de `liat` n t en fonction de n.

Question 2.5. Montrer la correction totale de la fonction `liat`, c'est-à-dire que, sous l'hypothèse que t est un tableau flexible de taille $n \geq 1$, alors `liat` n t termine et renvoie bien un tableau flexible de taille $n-1$ avec les éléments attendus.

Question 2.6. Donner un code pour la fonction `snoc: int -> tree -> elt -> tree`. Pour un tableau flexible t de taille n, et un élément x, l'appel `snoc` n t x renvoie un tableau flexible de taille $n+1$ avec les mêmes éléments que t aux indices $0 \leq i < n$ et l'élément x à l'indice n. La complexité doit être ~~$\mathcal{O}(n)$.~~ On ne demande pas de la justifier.
$\mathcal{O}(\log n)$

Agrandir/rétrécir du côté gauche. Agrandir ou rétrécir le tableau du côté gauche, c'est-à-dire du côté des indices les plus petits, est plus délicat à réaliser, car on décale alors les indices des éléments. L'opération `tail` t supprime le premier élément du tableau t. L'opération `cons` x t ajoute un élément x au début du tableau t. Si on reprend toujours le même exemple, alors ces deux opérations donnent respectivement les arbres suivants :



On prendra le temps de bien comprendre ce qui se passe ici. En particulier, on observera comment les éléments passent d'un côté à l'autre de l'arbre. En effet, les éléments situés à des indices pairs (resp. impairs) avant se retrouvent à des indices impairs (resp. pairs) après.

Question 2.7. Donner un code pour la fonction `tail: tree -> tree`. Pour un tableau flexible t de taille $n \geq 1$, l'appel `tail` t renvoie un tableau flexible de taille $n-1$ où le premier élément de t a été supprimé. La complexité doit être $\mathcal{O}(\log n)$ et on demande de la justifier.

Question 2.8. Donner un code pour la fonction `cons: elt -> tree -> tree`. Pour un tableau flexible t de taille n, l'appel `cons` x t renvoie un tableau flexible de taille $n+1$ où un premier élément x a été ajouté à gauche des éléments de t. La complexité doit être $\mathcal{O}(\log n)$. On ne demande pas de la justifier.

Construction à partir d'un vrai tableau. Si on se donne un (vrai) tableau `a`, de type `elt array` et de taille n, on peut chercher à construire un tableau flexible contenant les mêmes éléments que `a`. Si on le fait naïvement en appelant n fois la fonction `snoc` (ou n fois la fonction `cons`) alors la complexité sera $\mathcal{O}(n \log n)$ au total. Mais on peut faire mieux.

Question 2.9. Écrire une fonction `of_array: elt array -> tree` qui reçoit en argument un tableau `a` de taille n et construit un tableau flexible ayant les mêmes éléments que `a`, en temps $\mathcal{O}(n)$. Indication : on pourra introduire une fonction auxiliaire qui, pour deux entiers $m \geq 1$ et $k \geq 0$ donnés, construit le tableau flexible des éléments de `a` d'indices $mi + k$.

Tableau flexible constant. Pour cette dernière question, on cherche à construire un tableau flexible d'une certaine taille n dont tous les éléments sont identiques à un élément donné. Comme pour la question précédente, on pourrait le construire à partir des fonctions `cons` ou `snoc`, avec une complexité totale $\mathcal{O}(n \log n)$. Mais là encore, on peut faire mieux, et même beaucoup mieux !

Tournez la page S.V.P.

Question 2.10. Écrire une fonction `make: int -> elt -> tree` qui reçoit en arguments un entier $n \geq 0$ et un élément x, et renvoie un tableau flexible de taille n dont tous les éléments sont égaux à x. La complexité en temps doit être $\mathcal{O}(\log n)$. On demande de la justifier. Donner par ailleurs la complexité en espace.