

CORRIGÉ : ARBRES CROISSANTS (X-ENS 2014)

Partie I. Structure d'arbre croissant

Question 1. L'élément minimum d'un arbre croissant se trouve à la racine, donc :

```
let minimum = fonction
| E          -> failwith "minimum"
| N(_, x, _) -> x ;;
```

Question 2.

```
let rec est_un_arbre_croissant = fonction
| E          -> true
| N(E, x, E) -> true
| N(E, x, d) -> est_un_arbre_croissant d && x <= minimum d
| N(g, x, E) -> est_un_arbre_croissant g && x <= minimum g
| N(g, x, d) -> est_un_arbre_croissant g && x <= minimum g &&
                est_un_arbre_croissant d && x <= minimum d ;;
```

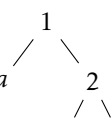
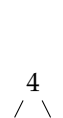
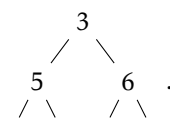
Sachant que la fonction `minimum` est de coût constant, celle-ci a un coût en $O(|t|)$.

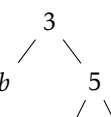
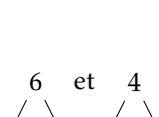
Question 3. Montrons par récurrence sur $n \in \mathbb{N}^*$ qu'il y a exactement $c_n = n!$ arbres croissants possédant n nœuds étiquetés par n entiers deux à deux distincts.

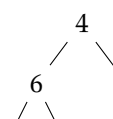
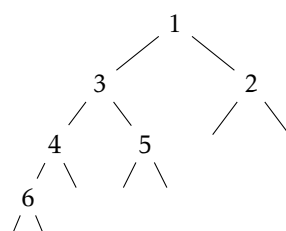
- Si $n = 0$, il y a un seul arbre croissant sans étiquette : **E**.
- Si $n \geq 1$, supposons le résultat acquis jusqu'au rang $n - 1$. Un arbre croissant est nécessairement de la forme $N(\mathbf{g}, \mathbf{1}, \mathbf{d})$, où \mathbf{g} et \mathbf{d} sont deux arbres croissants à respectivement k et $n - 1 - k$ nœuds et $k \in \llbracket 0, n - 1 \rrbracket$. Il y a $\binom{n-1}{k}$ façons de répartir les étiquettes $\llbracket 2, n \rrbracket$ entre ces deux arbres, d'où : $c_n = \sum_{k=0}^{n-1} \binom{n-1}{k} c_k c_{n-1-k}$.

Par hypothèse de récurrence forte, $c_k = k!$ et $c_{n-1-k} = (n - 1 - k)!$ donc $c_n = \sum_{k=0}^{n-1} (n - 1)! = n \times (n - 1)! = n!$.

Partie II. Opérations sur les arbres croissants

Question 4. La fusion de ces deux arbres donne : a  où a est la fusion des arbres  et .

a est donc égal à b  où b est la fusion des arbres .

b est égal à  donc l'arbre final recherché est : .

Question 5. Raisonnons par induction structurelle.

- Si $t_1 = t_2 = \mathbf{E}$ alors $t = \mathbf{E}$ est bien un arbre croissant et pour tout $x \in \mathbb{N}$, $\text{occ}(x, t) = \text{occ}(x, t_1) + \text{occ}(x, t_2)$ puisque $\text{occ}(x, \mathbf{E}) = 0$.
- Si l'un de ces deux arbres, par exemple t_2 , est égal à \mathbf{E} , alors $t = t_1$ est un arbre croissant et pour tout $x \in \mathbb{N}$, $\text{occ}(x, t) = \text{occ}(x, t_1) + \text{occ}(x, \mathbf{E})$ puisque $\text{occ}(x, \mathbf{E}) = 0$.
- Si aucun de ces arbres n'est égal à \mathbf{E} , notons $t_1 = \mathbf{N}(g_1, x_1, d_1)$ et $t_2 = \mathbf{N}(g_2, x_2, d_2)$ et supposons par exemple $x_1 \leq x_2$. Notons t' le résultat de la fusion de d_1 et de t_2 . Par hypothèse d'induction, t' est un arbre croissant, et $t = \mathbf{N}(t', x_1, g_1)$. Pour tout x présent dans d_1 on a $x \geq x_1$ car t_1 est un arbre croissant ; pour tout x présent dans t_2 on a $x \geq x_2 \geq x_1$ car t_2 est croissant. De ceci il résulte que t est croissant.
Enfin, par hypothèse d'induction, pour tout entier x on a $\text{occ}(x, t') = \text{occ}(x, d_1) + \text{occ}(x, t_2)$ donc :
 - pour tout $x \neq x_1$, $\text{occ}(x, t) = \text{occ}(x, t') + \text{occ}(x, g_1) = \text{occ}(x, d_1) + \text{occ}(x, t_2) + \text{occ}(x, g_1) = \text{occ}(x, t_1) + \text{occ}(x, t_2)$;
 - $\text{occ}(x_1, t) = 1 + \text{occ}(x_1, t') + \text{occ}(x_1, g_1) = 1 + \text{occ}(x_1, d_1) + \text{occ}(x_1, t_2) + \text{occ}(x_1, g_1) = \text{occ}(x_1, t_1) + \text{occ}(x_1, t_2)$.

Question 6. On obtient la fonction **ajoute** en fusionnant l'arbre croissant t avec l'arbre (croissant) $\mathbf{N}(\mathbf{E}, x, \mathbf{E})$. On commence donc par définir la fonction de fusion :

```
let rec fusion t1 t2 = match (t1, t2) with
| t1      , E                -> t1
| E       , t2               -> t2
| N(g, x1, d), N(_, x2, _) when x1 <= x2 -> N(fusion d t2, x1, g)
| _       , N(g, x2, d)      -> N(fusion d t1, x2, g) ;;
```

Cette fonction est de type *arbre* \rightarrow *arbre* \rightarrow *arbre*. Il reste alors à définir :

```
let ajoute x t = fusion t (N(E, x, E)) ;;
```

Question 7. Si $t = \mathbf{N}(g, m, d)$, on obtient l'arbre demandé t' en fusionnant les arbres g et d :

```
let supprime_minimum = fonction
| E      -> failwith "supprime_minimum"
| N(g, _, d) -> fusion g d ;;
```

Question 8. Il suffit d'appliquer récursivement la fonction **ajoute** pour obtenir :

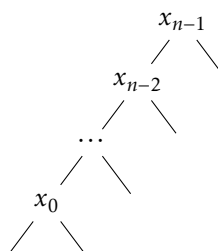
```
let ajouts_successifs x =
  let rec aux = fonction
  | -1 -> E
  | i  -> ajoute x.(i) (aux (i-1))
  in aux (vect_length x - 1) ;;
```

Question 9. Appelons *arbre peigne gauche* un arbre t égal à \mathbf{E} ou à $\mathbf{N}(g, x, \mathbf{E})$ où g est un arbre peigne gauche.

Il est clair que la hauteur d'un arbre peigne gauche non vide est égal à $|t|$.

Considérons une liste d'entiers distincts rangés par ordre décroissant et montrons par récurrence sur n que t_n est un arbre peigne gauche (avec pour conséquence que sa hauteur est égale à n).

- Si $n = 1$ alors $t_1 = \mathbf{N}(\mathbf{E}, x_0, \mathbf{E})$ est bien un arbre peigne gauche.
- Si $n \geq 2$ et si $t_{n-1} = \mathbf{N}(g, x_{n-2}, \mathbf{E})$ est un arbre peigne gauche, on a $x_{n-2} > x_{n-1}$ donc $t_n = \mathbf{N}(t_{n-1}, x_{n-1}, \mathbf{E})$ est bien un arbre peigne gauche.



Question 10. Considérons maintenant une liste d'entiers distincts rangés par ordre croissant, et posons $t_n = \mathbf{N}(g_n, x_0, d_n)$.

Puisque $t_{n-1} = \mathbf{N}(g_{n-1}, x_0, d_{n-1})$, nous avons $t_n = \mathbf{N}(\mathbf{fusion}(d_{n-1}, \mathbf{N}(\mathbf{E}, x_{n-1}, \mathbf{E})), x_0, g_{n-1})$ donc $\begin{cases} g_n = \mathbf{fusion}(d_{n-1}, \mathbf{N}(\mathbf{E}, x_{n-1}, \mathbf{E})) \\ d_n = g_{n-1} \end{cases}$

Autrement dit :

- lorsque n est pair, g_n est obtenu par ajouts successifs à \mathbf{E} des entiers $x_1, x_3, x_5, \dots, x_{n-1}$ et d_n par ajouts successifs des entiers x_2, x_4, \dots, x_{n-2} ;
- lorsque n est impair, g_n est obtenu par ajouts successifs à \mathbf{E} des entiers x_2, x_4, \dots, x_{n-1} et d_n par ajouts successifs des entiers $x_1, x_3, x_5, \dots, x_{n-2}$;

Prouvons alors par récurrence sur n que la hauteur h_n de t_n est égale à $\lceil \log(n+1) \rceil$ en raisonnant par récurrence sur $n \in \mathbb{N}$.

- C'est clair lorsque $n = 0$ puisque $t_0 = \mathbf{E}$ donc $h_0 = 0$.
- Si $n > 0$, supposons le résultat acquis jusqu'au rang $n-1$ et appliquons-le aux arbres g_n et d_n .
 - Si $n = 2p$ alors $h(g_n) = \lceil \log(p+1) \rceil$ et $h(d_n) = \lceil \log(p) \rceil$ donc $h(t_n) = \lceil \log(p+1) \rceil + 1 = \lceil \log(2p+2) \rceil = \lceil \log(n+2) \rceil = \lceil \log(n+1) \rceil$ car $n+1$ est impair donc ne peut être une puissance de 2.
 - Si $n = 2p+1$ alors $h(g_n) = h(d_n) = \lceil \log(p+1) \rceil$ donc $h(t_n) = \lceil \log(p+1) \rceil + 1 = \lceil \log(2p+2) \rceil = \lceil \log(n+1) \rceil$.

Dans les deux cas on a bien obtenu que $h_n = \lceil \log(n+1) \rceil$.

Partie III. Analyse

Question 11. Une traduction littérale de la définition conduirait à un coût trop important ; il faut utiliser une fonction auxiliaire qui renvoie le couple $(\Phi(t), |t|)$:

```
let potentiel t =
  let rec aux = fonction
    | E -> (0, 1)
    | N(g, -, d) -> let (p1, n1) = aux g and (p2, n2) = aux d in
                    (p1 + p2 + (if n1 < n2 then 1 else 0), n1 + n2 + 1)
  in fst (aux t) ;;
```

Question 12. Nous allons prouver l'inégalité demandée par induction structurale.

- Si $t_1 = t_2 = \mathbf{E}$ alors $t = \mathbf{E}$ donc $C(t_1, t_2) = 0$, $\Phi(t_1) = \Phi(t_2) = \Phi(t) = 0$ et l'inégalité s'écrit : $0 \leq 0$.
- Si l'un de ces deux arbres, par exemple t_2 , est égal à \mathbf{E} , alors $t = t_1$ et $C(t_1, t_2) = 0$, $\Phi(t_1) = \Phi(t)$, $\Phi(t_2) = 0$ donc l'inégalité demandée s'écrit : $0 \leq 2 \log |t_1|$, ce qui est vrai.
- Si aucun de ces arbres n'est égal à \mathbf{E} , notons $t_1 = \mathbf{N}(g_1, x_1, d_1)$ et $t_2 = \mathbf{N}(g_2, x_2, d_2)$ et supposons par exemple $x_1 \leq x_2$. Notons t' le résultat de la fusion de d_1 et de t_2 . Alors $t = \mathbf{N}(t', x_1, g_1)$ et $C(t_1, t_2) = 1 + C(d_1, t_2)$.

Par hypothèse d'induction, $C(d_1, t_2) \leq \Phi(d_1) + \Phi(t_2) - \Phi(t') + 2(\log |d_1| + \log |t_2|)$ donc :

$$C(t_1, t_2) \leq 1 + \Phi(d_1) + \Phi(t_2) - \Phi(t') + 2(\log |d_1| + \log |t_2|).$$

$$t_1 = \mathbf{N}(g_1, x_1, d_1) \text{ donc } \Phi(t_1) = \Phi(g_1) + \Phi(d_1) + \begin{cases} 1 & \text{si } |g_1| < |d_1| \\ 0 & \text{sinon} \end{cases}$$

$$t = \mathbf{N}(t', x_1, g_1) \text{ donc } \Phi(t) = \Phi(t') + \Phi(g_1) + \begin{cases} 1 & \text{si } |t'| < |g_1| \\ 0 & \text{sinon} \end{cases}$$

Nous avons donc $\Phi(t_1) - \Phi(t) = \Phi(d_1) - \Phi(t') + \varepsilon$ avec $\varepsilon \in \{-1, 0, 1\}$.

Observons maintenant que t' est la fusion de d_1 et de t_2 donc $|t'| > |d_1|$. Ceci permet de distinguer trois cas.

- Si $|g_1| < |d_1|$ alors $\varepsilon = 1$ et :

$$C(t_1, t_2) \leq \Phi(t_1) + \Phi(t_2) - \Phi(t) + 2(\log |d_1| + \log |t_2|) \leq \Phi(t_1) + \Phi(t_2) - \Phi(t) + 2(\log |t_1| + \log |t_2|).$$

- Si $|d_1| \leq |g_1| \leq |t'|$ alors $\varepsilon = 0$ et :

$$C(t_1, t_2) \leq \Phi(t_1) + \Phi(t_2) - \Phi(t) + 1 + 2(\log |d_1| + \log |t_2|) \leq \Phi(t_1) + \Phi(t_2) - \Phi(t) + 2(\log |t_1| + \log |t_2|).$$

car $1 + 2 \log |d_1| \leq 2 \log 2 |d_1| \leq 2 \log (|d_1| + |g_1|) \leq 2 \log |t_1|$.

- Si $|g_1| > |t'|$ alors $\varepsilon = -1$ et :

$$C(t_1, t_2) \leq \Phi(t_1) + \Phi(t_2) - \Phi(t) + 2 + 2(\log |d_1| + \log |t_2|) \leq \Phi(t_1) + \Phi(t_2) - \Phi(t) + 2(\log |t_1| + \log |t_2|).$$

car $2 + 2 \log |d_1| = 2 \log 2 |d_1| \leq 2 \log (|d_1| + |g_1|) \leq 2 \log |t_1|$.

La formule est bien prouvée par induction.

Question 13. Notons c_k le coût total de la construction de t_k . Nous avons $c_k = c_{k-1} + C(t_{k-1}, \mathbf{N}(\mathbf{E}, x_{k-1}, \mathbf{E}))$ donc d'après la question précédente,

$$c_k \leq c_{k-1} + \Phi(t_{k-1}) + 0 - \Phi(t_k) + 2(\log |t_{k-1}| + \log 3).$$

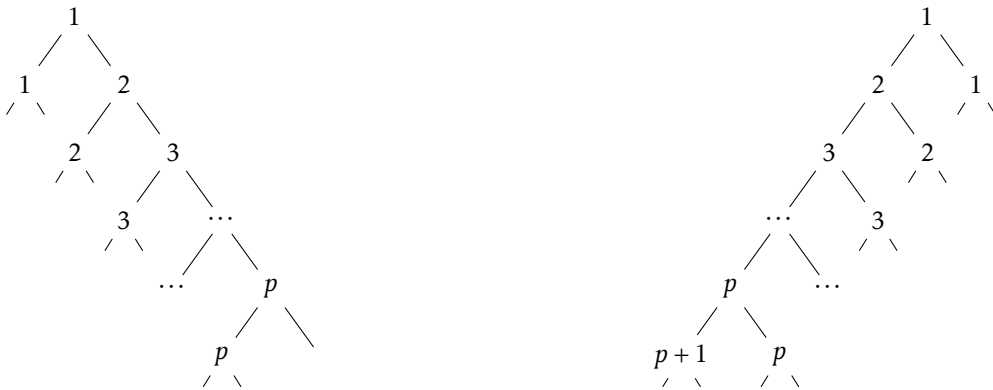
t_{k-1} contient $k - 1$ nœuds donc $|t_k| = 2k - 1$ et :

$$c_k \leq c_{k-1} + \Phi(t_{k-1}) - \Phi(t_k) + 2(\log(2k - 1) + \log 3).$$

Par télescopage, on en déduit : $c_n \leq c_0 + \Phi(t_0) - \Phi(t_n) + 2 \sum_{k=1}^n \log(2k - 1) + 2n \log 3 \leq 2n + 2 \log(n!) + 2n \log 3$.

Il est bien connu que $\log(n!) \sim n \log n$, donc $c_n = O(n \log n)$.

Question 14. Considérons la liste d'entiers $p, p, p - 1, p - 1, p - 2, p - 2, \dots, 2, 2, 1, 1$. Il n'est pas difficile de montrer par récurrence que la construction de la question précédente donne l'arbre de gauche :



Ajoutons maintenant l'élément $p + 1$ à la suite de cette liste. Il est là encore possible de montrer par récurrence qu'on réalise p appels récurifs à la fonction **fusion** avant d'obtenir l'arbre dessiné à droite.

Avec $n = 2p + 1$ nous avons exhibé un exemple pour lequel la dernière opération de fusion a un coût égal à $n/2$.

En général, lorsqu'on analyse le coût cumulé d'une suite d'opérations de coût linéaire on obtient un coût quadratique ; ce n'est pas le cas ici puisque l'exemple ci-dessus montre que même si certains des coûts élémentaires peuvent être linéaires, le coût total reste un $O(n \log n)$. Ainsi, lors de la construction de cette suite d'arbres croissants, le coût d'une fusion reste *en moyenne* logarithmique. C'est dans ce sens qu'on peut parler de complexité amortie : le pire des cas se produit suffisamment peu pour être amorti lorsqu'on réalise une successions d'opérations de même type¹.

Question 15. D'après la question 12, le coût de la fusion de g_i et de d_i est majoré par :

$$C(g_i, d_i) \leq \Phi(g_i) + \Phi(d_i) - \Phi(t_{i+1}) + 2(\log |g_i| + \log |d_i|) \leq \Phi(t_i) - \Phi(t_{i+1}) + 2(\log |g_i| + \log |d_i|)$$

La concavité de la fonction \log donne l'inégalité : $\frac{1}{2}(\log a + \log b) \leq \log\left(\frac{a+b}{2}\right)$ qui conduit à la nouvelle majoration :

$$C(g_i, d_i) \leq \Phi(t_i) - \Phi(t_{i+1}) + 4(\log(|g_i| + |d_i|)) - 4 \log 2 = \Phi(t_i) - \Phi(t_{i+1}) + 4 \log(|t_i| - 1) - 4 \log 2.$$

Sachant que t_i possède $n - i$ nœuds on a $|t_i| = 2(n - i) + 1$ donc $C(g_i, d_i) \leq \Phi(t_i) - \Phi(t_{i+1}) + 4 \log(n - i)$.

Le coût total de la construction est donc majoré par :

$$\sum_{i=0}^{n-1} C(g_i, d_i) \leq \Phi(t_0) - \Phi(t_n) + 4 \sum_{i=0}^{n-1} \log(n - i) \leq n + \log(n!).$$

Sachant que $\log(n!) \sim n \log n$ le coût total est bien un $O(n \log n)$.

Partie IV. Applications

Question 16. Les principe du tri est très voisin de *heapsort*, le tri par tas : on commence par créer l'arbre croissant décrit à la question 8 puis on récupère un par un les éléments triés à l'aide des fonctions **minimum** et **supprime_minimum** des questions 1 et 7.

1. En ce sens, la complexité amortie se distingue de la complexité en moyenne.

```

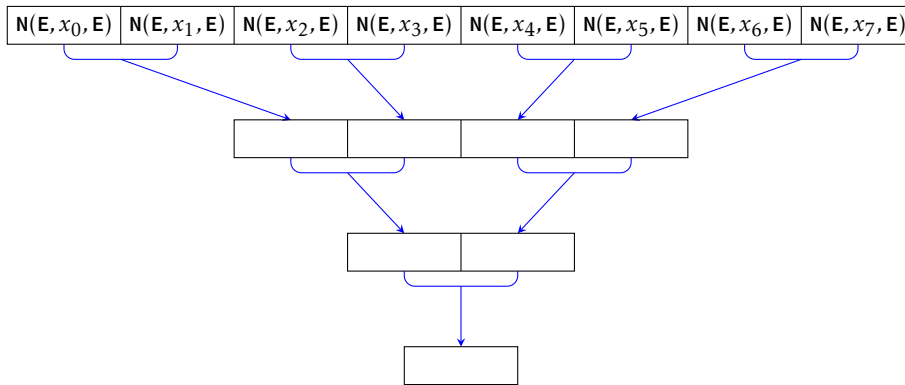
let tri x =
  let rec aux t = function
    | k when k = vect_length x -> ()
    | k -> x.(k) <- minimum t ; aux (supprime_minimum t) (k+1)
  in aux (ajouts_successifs x) 0 ;

```

D'après la question 13, la construction de l'arbre (donc de la fonction `ajouts_successifs`) a un coût en $O(n \log n)$. Les n appels à la fonction `minimum` ont chacun un coût constant donc un coût total en $O(n)$. Enfin, chaque appel à la fonction `supprime_minimum` réalise la fusion entre fils gauche et droit donc d'après la question 15, cette opération a un coût total en $O(n \log n)$.

La complexité temporelle de ce tri est donc en $O(n \log n)$.

Question 17. Pour construire plus rapidement un arbre croissant contenant les n éléments d'un tableau on adopte maintenant une stratégie *diviser pour régner* : partant de $n = 2^k$ arbres à un nœud, on fusionne deux arbres voisins jusqu'à n'en obtenir plus qu'un.



D'après la question 12, le coût de la construction de t_{i+1}^j est majoré par :

$$C(t_i^{2^j}, t_i^{2^{j+1}}) \leq \Phi(t_i^{2^j}) + \Phi(t_i^{2^{j+1}}) - \Phi(t_{i+1}^j) + 2(\log |t_i^{2^j}| + \log |t_i^{2^{j+1}}|)$$

Il est facile de prouver que t_i^j possède 2^j nœuds et donc que $|t_i^j| = 2^{i+1} - 1$ donc :

$$C(t_i^{2^j}, t_i^{2^{j+1}}) \leq \Phi(t_i^{2^j}) + \Phi(t_i^{2^{j+1}}) - \Phi(t_{i+1}^j) + 4 \log(2^{i+1} - 1) \leq \Phi(t_i^{2^j}) + \Phi(t_i^{2^{j+1}}) - \Phi(t_{i+1}^j) + 4(i+1).$$

Le coût total de cette construction est donc majoré par :

$$\sum_{\substack{0 \leq i < k \\ 0 \leq j < 2^{k-i-1}}} C(t_i^{2^j}, t_i^{2^{j+1}}) \leq \sum_{j=0}^{n-1} \Phi(N(E, x_j, E)) - \Phi(t_k^0) + 4 \sum_{\substack{0 \leq i < k \\ 0 \leq j < 2^{k-i-1}}} (i+1) \leq 4 \sum_{i=0}^{k-1} (i+1) 2^{k-i-1} = 4 \sum_{i=1}^k i 2^{k-i} = 4 \cdot 2^k \sum_{i=1}^k \frac{i}{2^i}.$$

Il n'est pas nécessaire de calculer explicitement cette dernière somme ; il suffit d'observer que la série $\sum \frac{n}{2^n}$ converge pour en conclure que le coût total est un $O(2^k) = O(n)$.

Question 18. On utilise un tableau à $n = 2^k$ cases contenant initialement les arbres $N(E, x_i, E)$ et on fusionne deux cases voisines jusqu'à ce qu'il n'en reste plus qu'une :

```

let construire x =
  let n = vect_length x in
  let a = make_vect n E in
  let rec aux = function
    | 1 -> a.(0)
    | k -> for j = 0 to k/2-1 do a.(j) <- fusion a.(2*j) a.(2*j+1) done ;
          aux (k/2)
  in
  for j = 0 to n-1 do a.(j) <- N(E, x.(j), E) done ;
  aux n ;

```

Question 19. Pour traiter le cas d'un nombre quelconque d'éléments il suffit, dans la fonction précédente, de garder inchangé le dernier arbre de la liste lorsqu'à une étape le nombre d'arbres à traiter est impair.

```
let construire2 x =  
  let n = vect_length x in  
  let a = make_vect n E in  
  let rec aux = function  
    | 1 -> a.(0)  
    | k -> for j = 0 to k/2-1 do a.(j) <- fusion a.(2*j) a.(2*j+1) done ;  
           if k mod 2 = 1 then a.(k/2) <- a.(k-1) ;  
           aux ((k+1)/2)  
  in  
  for j = 0 to n-1 do a.(j) <- N(E, x.(j), E) done ;  
  aux n ;;
```

