Corrigé - X-ENS Informatique A - 2014

Parti I - Structure d'arbre croissant

Question 1

Selon la définition d'un arbre croissant, si $t = N(g, x, d)$, on a x qui est plus petit que tous les éléments de g et de d. Ainsi le minimum est la racine :

```
let minimum (N (_,m,_)) = m;;
```

Question 2

On parcoure l'arbre de manière récursive en vérifiant à chauqe noeud $N(g, x, d)$ que x minore g et d. Pour cela, il suffit de vérifier que x est plus petit que les racines de ces arbres s'ils sont non vides, ce qui s'effectue en temps constant. La complexité correspond donc au nombre d'appels récursifs, i.e. au nombre de noeuds de t : $O(|t|)$.

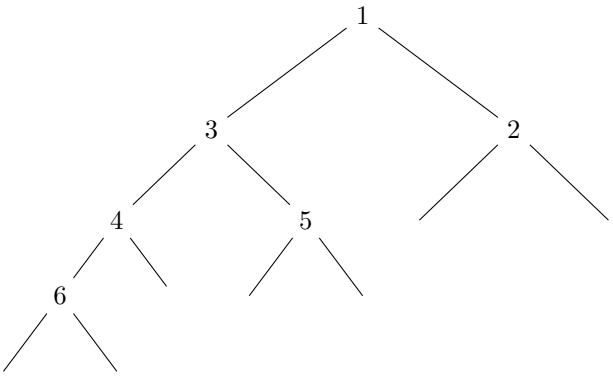
```
let rec est_un_arbre_croissant t =
    let minore x t =
        match t with
        | E -> true
        | N (_,y,_) -> x <= y
    in
    match t with
    | E -> true
    | N (g,x,d) -> minore x g && minore x d
        && est_un_arbre_croissant g
        && est_un_arbre_croissant d;;
```

Question 3

Si t est un arbre croissant contenant n noeuds étiquettés de 1 à n, alors nécessairement le noeud contenant n est de la forme $N(E, n, E)$.
Considérons un arbre croissant contenant $n-1$ noeuds étiquettés de 1 à $n-1$. Il possède n feuilles, et donc il existe exactement n manières de le prolonger en remplacant une feuille par un noeud $N(E, n, E)$.
Comme il n'y a qu'un arbre croissant contenant 1 : $N(E, 1, E)$ on en déduit qu'il y a exactement $n!$ arbres croissants possédant n noeuds étiquettés de 1 à n.

Partie II - Opérations sur les arbres croissants

Question 4



Question 5

Les noeuds de t sont de deux sortes :
- Soit il s'agit de noeuds de t_1 ou de t_2 reproduits tels quels.
- Soit il sont construit lors d'un appel récursif. Dans ce cas si c'est le noeud $N(g_1, x_1, d_1)$ qui est déconstruit alors l'appel récursif porte sur d_1 et t_2, qui contiennent un noeud x_1 de moins que t_1 et t_2, et on construit un noeud d'étiquette x_1.
  Ainsi, le nombre d'occurrences de x_1 est stable.
  Le cas de l'autre appel récursif est symétrique.

Les occurrences sont donc en bijection entre celles de t et celles de (t_1, t_2).

Question 6

On fusionne l'arbre t avec un arbre qui ne contient qu'un noeud d'étiquette x, ainsi, après fusion, on obtiendra, selon la question précédente, un arbre t' tel que $occ(x, t') = occ(x, t) + occ(x, N(E, x, E)) = 1 + occ(x, t)$.

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

Question 7

On sait déjà que si $t = N(g, x, d)$ alors x est une occurrence du minimum de t. On a donc $occ(x, g) + occ(x, d) = occ(x, t) - 1$ et on peut fusionner g et d pour obtenir l'arbre t' ayant une occurrence de moins du minimum.

```
let supprime_minimum (N(g,_,d)) = fusion g d;;
```

Question 8

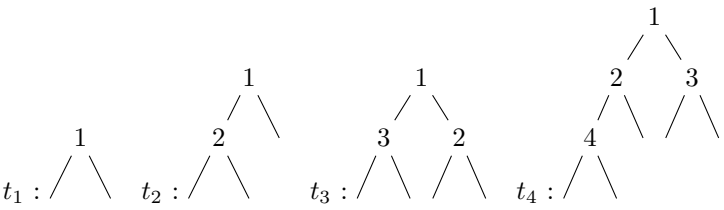
On utilise directement la fonction `ajoute` de la question 6 qui a été formulée pour être compatible avec la construction demandée (fusion à droite) :

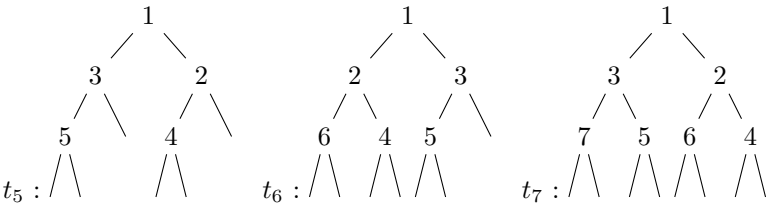
```
let ajouts_successifs v =
    let a = ref E in
    for i = 0 to vect_length v - 1 do
        a := ajoute v.(i) !a
    done;
    !a;;
```

Question 9

Considérons $t = N(N(\cdots N(E, p+k, E)\cdots, p+2,), p+1, E)$, c'est-à-dire un arbre peigne qui ne compte qu'un seul chemin de la racine $p{+}1$ au nœud $p{+}k$. On ajoute p à cet arbre. Comme il est plus petit que tous les nœud, il va forcément être placé comme nouvelle racine de la fusion et t va être le sous arbre gauche de la fusion. Ainsi t' issu de la fusion a la forme suivante $t' = N(t, p, E)$ et est aussi un peigne mais augmenté d'un cran.
On en déduit que l'arbre issu de l'ajout de $x_0 = n > x_1 = n{-}1 > \cdots > x_{n-1} = 1$ est un peigne, et donc qu'il est de hauteur $n \geq n/2$.

Question 10





Un peu d'expérimentation suggère que les arbres t_n vérifient la propriété (*) : pour tout nœud de taille $p \geq 1$ le fils gauche est de taille $\lceil \frac{p-1}{2} \rceil = \lfloor \frac{p}{2} \rfloor$ et le fils gauche est de taille $\lfloor \frac{p-1}{2} \rfloor$
Lors d'un ajout les différentes adjonctions se font en ajoutant un élément supérieur à ceux de l'arbre, on est donc dans le cas de la troisième (ou deuxième) ligne de la définition de `fusion`.
On procède par récurrence.

1. Lors de l'ajout à un nœud de taille 0 on obtient `N(E,n,E)`, qui vérifie (*).
2. Lors de l'ajout nœud de taille $p \geq 1$ qui vérifie $(*)$
   - la taille devient $p+1$
   - le fils droit est l'ancien fils gauche,
     il vérifie (*) et est de taille $\lfloor \frac{p}{2} \rfloor = \left\lfloor \frac{(p+1)-1}{2} \right\rfloor$

- le fils gauche est l'ancien fils droit auquel on ajoute un élément plus grand : par récurrence il vérifie (*) et est de taille $\left\lfloor \frac{p-1}{2} \right\rfloor + 1 = \left\lfloor \frac{p+1}{2} \right\rfloor$
- Ainsi le nouveau nœud vérifie (*)

On prouve alors, toujours par récurrence, que si $2^p \leq n < 2^{p+1} - 1$ alors la hauteur de t_n est $p+1$

Partie III - Analyse

Question 11

Pour garantir la complexité en $O(|t|)$ il faut effectuer en même temps le calcul de taille et le calcul de potentiel.

```
let potentiel t =
    let rec aux t =
        match t with
        | E -> 0, 0
        | N(g,_,d) ->
            let taille_g, pot_g = aux g in
            let taille_d, pot_d = aux d in
            1+taille_g+taille_d,
                (if taille_g < taille_d then 1 else 0)+pot_g+pot_d
    in snd (aux t);;
```

Question 12

On effectue une récurrence sur la taille de la fusion

- On a $\Phi(E) = 0$ et $\log|E| = 0$ et la fusion de t et de E est t.
  On en déduit $0 = C(t, E) \leq \Phi(t) + \Phi(E) - \Phi(t) + 2(\log|t| + \log|E|) = 2\log|t|$.
  La formule (1) est symétrique ; ainsi elle est vérifiée si t_1 ou t_2 est vide.
- On suppose maintenant t_1 et t_2 non vides.
  Soit t le résultat de la fusion de $t_1 = N(g_1, x_1, d_1)$ et de $t_2 = N(g_2, x_2, d_2)$.
  On suppose $x_1 \leq x_2$.
  Alors $t = N(t'_2, x_1, g_1)$ où t'_2 est le résultat de la fusion de d_1 et de t_2.
  On a $\Phi(t) = \Phi(t'_2) + \Phi(g_1) + \alpha$ avec $\alpha = 1$ si t est lourd et $\alpha = 0$ sinon.
  De même $\Phi(t_1) = \Phi(d_1) + \Phi(g_1) + \alpha_1$ avec $\alpha_1 = 1$ si t_1 est lourd ou $\alpha_1 = 0$n.
  On a toujours $\log|t_1| \geq \log|d_1|$.
  Si $\alpha_1 = 0$ alors $|g_1| \geq |d_1|$ d'où $|t_1| = |g_1| + |d_1| + 1 \geq 2|d_1|$ donc
  $\log|t_1| \geq \log|d_1| + 1$. Dans tous les cas on a $\log|d_1| + 1 - \alpha_1 \leq \log|t_1|$.
  L'hypothèse de récurrence donne
  $C(d_1, t_2) \leq \Phi(d_1) + \Phi(t_2) - \Phi(t'_2) + 2(\log|d_1| + \log|t_2|)$ d'où

$$
\begin{aligned}
C(t_1,t_2) &= C(d_1,t_2)+1 \\
&\leq \Phi(d_1)+\Phi(t_2)-\Phi(t_2')+2(\log|d_1|+\log|t_2|)+1 \\
&\leq \Phi(t_1)-\Phi(g_1)-\alpha_1+\Phi(t_2)-\Phi(t_2')+2(\log|d_1|+\log|t_2|)+1 \\
&\leq \Phi(t_1)+\Phi(t_2)-\Phi(g_1)-\Phi(t_2')+2(\log|d_1|+\log|t_2|)+1-\alpha_1 \\
&\leq \Phi(t_1)+\Phi(t_2)-\Phi(t)-\alpha+2(\log|d_1|+\log|t_2|)+1-\alpha_1 \\
&\leq \Phi(t_1)+\Phi(t_2)-\Phi(t)+\log|d_1|+1-\alpha_1+\log|d_1|+2\log|t_2| \\
&\leq \Phi(t_1)+\Phi(t_2)-\Phi(t)+\log|t_1|+\log|d_1|+2\log|t_2| \\
&\leq \Phi(t_1)+\Phi(t_2)-\Phi(t)+2(\log|t_1|+\log|t_2|)
\end{aligned}
$$

(1) est vérifiée.
Le cas $x_1 > x_2$ se fait de même.

Question 13

Soit C le cout total, on a $C = \sum_{i=1}^{n} C(t_{i-1}, N(E,x_i,E)) \leq \sum_{i=1} \Phi(t_{i-1}) + \Phi(N(E,x_i,E)) - \Phi(t_i) + 2(\log|t_{i-1}| + \log|N(E,x_i,E)|) \leq \Phi(t_0) - \Phi(t_n) + 2\sum_{i=1}^{n} \log|t_{i-1}| = 2\sum_{i-1}^{n} \log i - \Phi(t_n) \leq 2n\log n$.
Donc $C = O(n\log n)$.

Question 14

Lors d'un ajout d'un élément, on fusionne t avec un arbre ne contenant aucun nœud mis à part la racine. Le cout de la fusion correspond donc exactement au nombre de nœuds sur le chemin le plus à droite de t plus 1 (le noeud ajouté).
Supposons $n = 2p-1$, on pose $x_0 = p, x_1 = p+1, x_2 = p-1, x_3 = p+2, x_4 = p-2, \cdots, x_{n-4} = 2p-1, x_{n-3} = 1, x_{n-2} = 2p, x_{n-1} = n$.
L'arbre t_{n-1} obtenu a un chemin tout à droite qui contient les nœuds $1, 2, \cdots, p$ et ces nœuds ont pour fils gauche des nœuds simples contenant, dans l'ordre, les nœuds $2p, \cdots, p+1$.
Le cout de fusion avec $N(E,n,E)$ est donc de $p+1 \geq n/2$.
Le cout total de la construction étant en $O(nlogn)$ c'est que ce surplus de cout sur une fusion est compensée par des fusions moins couteuses. Pour la construction globale tout se passe comme si les fusions avait un cout en $O(logn)$, on a *amorti* le surcout.

Question 15

Soit C le cout total, on a $C = \sum_{i=0}^{n-1} C(g_i,d_i) \leq \sum_{i=1}^{n-1} \Phi(g_i)+\Phi(d_i)-\Phi(t_{i+1})+2(\log|g_i|+\log|d_i|) \leq \sum_{i=0}^{n-1} \Phi(t_i)-\Phi(t_{i+1})+4\log|t_i| = \Phi(t_0)-\Phi(t_n)+4\sum_{i=0}^{n-1}\log|t_i| \leq 2n+4n\log n$.
Donc $C = O(n\log n)$.

Partie IV - Applications

Question 16

On remarque que dans la construction de la question 15 on passe de t_i à t_{i+1} en supprimant le minimum. On peut donc effectuer cette construction et à chaque étape stocker dans le tableau le minimum courant. On aura ainsi trié le tableau pour un cout en $O(n \log n)$. De plus, on note que chaque arbre aura une taille plus petite que $2n + 1$ et qu'on ne peut garder en mémère que deux arbres : l'arbre courant et le prochain. Ainsi, en espace la complexité est en $O(n)$ (*non demandé mais cela semble crucial dans ce genre d'algorithmes qui allouent des structures*).

```
let tri v =
    let t = ref (ajouts_successifs v) in
    let n = vect_length v in
    for i = 0 to n -1 do
        v.(i) <- minimum !t;
        t := supprime_minimum !t
    done;;
```

Question 17

On remarque facilement que t_i^j contient 2^i nœuds et donc $\log |t_i^j| \leq i + 2$ car $|t_i^j| = 2^{i+1} + 1 \leq 2^{i+2}$.
Soit C le cout total on a $C = \sum_{i=0}^{k-1} \sum_{j=0}^{2^{k-i-1}} C(t_i^{2j}, t_i^{2j+1}) \leq \sum_{i=0}^{k-1} \sum_{j=0}^{2^{k-i-1}-1} \Phi(t_i^{2j}) + \Phi(t_i^{2j+1}) - \Phi(t_{i+1}^j) + 2(\log |t_i^{2j}| + \log |t_i^{2j+1}|) \leq 4 \sum_{i=0}^{k-1} 2^{k-i-1}(i+2) - \Phi(t_k^0) \leq 2^k \times 2 \sum_{i=0}^{k-1} \frac{i+2}{2^i}$
Or $\sum_{i=0}^{k-1} \frac{i+2}{2^i} \xrightarrow{k \to +\infty} 2$ donc est bornée à partir d'un certain rang. Ainsi $C = O(2^k)$.

Question 18

On effectue directement le calcul précédent à l'aide d'une fonction auxillaire portant sur i et sur j. Aucun appel récursif n'est effectué inutilement deux fois car on ne fusionne jamais deux fois un arbre dans la construction précédente.

```
let construire v =
    let rec aux i j =
        if i = 0
        then N(E,v.(j),E)
        else fusion (aux (i-1) (2*j)) (aux (i-1) (2*j+1))
    in
    let n = vect_length v in
    let k = log2 n in
    aux k 0;;
```

Question 19

Si n n'est pas une puissance de 2 mais $k = \lceil \log n \rceil$, il suffit de compléter les t_0^j avec $j > n-1$ par des arbres vides. Ensuite, on procéde comme précédemt pour une complexité en $O(2^k) = O(n)$.