2I003 – Interrogation longue 2 (1h) – Corrigé

30 novembre 2017

On s'intéresse dans cet exercice à un encodage court de **la forme** d'un arbre binaire T, appelé *représentation de Dyck* de T, ou encore *mot de Dyck* associé à T.

Convention d'écriture, notations. Comme seule la forme de l'arbre binaire sera pertinente, un arbre binaire non vide sera représenté comme un couple $T = (G, D)$, où G et D sont ses sous-arbres gauche et droit, et où l'**on omet la clef**. On restreint aussi les fonctions usuelles vues en cours aux fonctions suivantes :

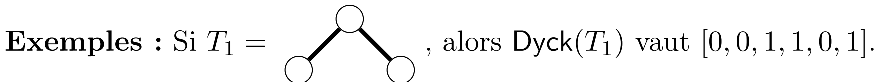
- `estABvide(T)` pour tester si un arbre `T` est l'arbre vide ;
- `T.gauche` et `T.droit` pour accéder respectivement aux sous-arbres gauche et droit de `T` ;
- `ABvide()` pour créer un arbre binaire vide ;
- `AB(G, D)` pour créer l'arbre binaire dont le sous-arbre gauche est `G`, le sous-arbre droit est `D`.

Lorsqu'il vous sera demandé d'écrire une fonction, vous pourrez choisir le langage/pseudo-code que vous souhaitez, tant que vos algorithmes restent clairs et concis.

Définition. La représentation de Dyck d'un arbre binaire T est une liste de bits (c'est-à-dire de 0 et de 1) définie par induction de la manière suivante.

$$\mathsf{Dyck}(T) = \begin{cases} \texttt{[ ]} & \text{si } T \text{ est vide}, \\ \texttt{[0] + } \mathsf{Dyck}(G) \texttt{ + [1] + } \mathsf{Dyck}(D) & \text{si } G \text{ et } D \text{ sont les sous-arbres gauche et droit de } T, \end{cases}$$

l'opération `+` correspondant à la concaténation de listes.

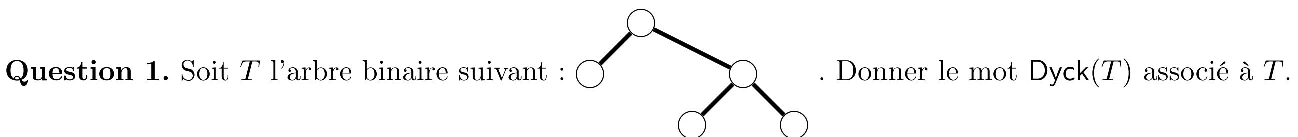


Exemples : Si $T_1 =$ [figure], alors $\mathsf{Dyck}(T_1)$ vaut $[0, 0, 1, 1, 0, 1]$.

Si $T_2 =$ [figure], alors $\mathsf{Dyck}(T_2)$ vaut $[0, 1, 0, 1, 0, 1]$.

1 De l'arbre au mot.

Dans cette partie, on cherche à construire le mot de Dyck associé à un arbre binaire.



Question 1. Soit T l'arbre binaire suivant : [figure]. Donner le mot $\mathsf{Dyck}(T)$ associé à T.

Solution : $[0, 0, 1, 1, 0, 0, 1, 1, 0, 1]$.

Question 2. Rappeler la définition inductive de la taille $n(T)$ d'un arbre binaire T.

Solution : $n(T) = \begin{cases} 0 & \text{si } T \text{ est vide}, \\ 1 + n(G) + n(D) & \text{si } G \text{ et } D \text{ sont les sous-arbres gauche et droit de } T, \end{cases}$

Question 3. Démontrer formellement que la longueur de $\mathsf{Dyck}(T)$ est $2n(T)$.

Par récurrence forte sur n.
• Base : si $n(T) = 0$, alors T est vide donc $\mathsf{Dyck}(T) = \texttt{[]}$. Sa longueur est donc bien 0.
• Hérédité : supposons la propriété vraie pour tous les arbres de taille $\leq n-1$. Soit T de taille n. Par définition de Dyck, on a :

$$\mathrm{len}(\mathsf{Dyck}(T)) = 2 + \mathrm{len}(\mathsf{Dyck}(G)) + \mathrm{len}(\mathsf{Dyck}(D))$$

Comme G et D sont de taille $\leq n-1$, par hypothèse de récurrence $\mathrm{len}(\mathsf{Dyck}(G)) = 2n(G)$ et $\mathrm{len}(\mathsf{Dyck}(D)) = 2n(D)$. Donc,

$$\mathrm{len}(\mathsf{Dyck}(T)) = 2 + 2n(G) + 2n(D) = 2(1 + n(G) + n(D) = 2n(T)\,.$$

Question 4. Écrire une fonction récursive `Dyck(T)` qui, étant donné un arbre T, renvoie la représentation de Dyck de T. Pour manipuler vos listes, vous pourrez utiliser les opérations usuelles `L.insert(i, x)`, `L.append(x)` ou encore la concaténation `+`.

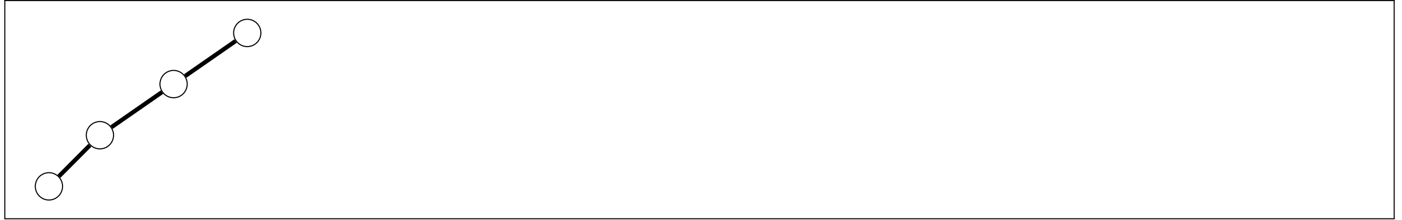
```
def Dyck(T):
    if estABvide(T):
        return []
    g = Dyck(T.gauche)
    d = Dyck(T.droit)
    g.insert(0, 0)
    d.insert(0, 1)
    return g + d
```

Question 5. Quelle est la complexité de votre fonction `Dyck(T)` en fonction de $n(T)$? Vous justifierez votre réponse, et vous supposerez que vos listes sont implantées comme des listes doublement chaînées.

Solution : Soit $c(T)$ la complexité demandée. Montrons que $c(T) \in \mathcal{O}(n(T))$, c'est-à-dire qu'il existe $k > 0$ tel que $c(T) \leq k \cdot n(T)$.
Comme l'insertion en tête et la concaténation sont en temps constant pour des listes doublement chaînées, on a $c(T) = c(D) + c(G) + k$ où k est une constante. S'il on veut être très formel, on peut raisonner par récurrence (mais une bonne justification suffira).
• Base : $c(\varnothing) = 1 \leq k$.
• Hérédité : $c(T) = c(D) + c(G) + k \leq k(n(G) + n(D) + 1) = k \cdot n(T)$.

2 Du mot à l'arbre.

Question 6. Donner un arbre binaire T tel que $\mathsf{Dyck}(T)$ vaut $[0, 0, 0, 0, 1, 1, 1, 1]$.



Question 7. Toute liste de bits de longueur paire correspond-elle à la représentation de Dyck d'un arbre binaire ? Justifier.

Non. Par exemple, un mot de Dyck doit avoir autant de 0 que de 1, ou doit commencer par un 0 et terminer par un 1 (ce qui n'est pas le cas d'une liste quelconque).

On souhaite dorénavant écrire la fonction réciproque de `Dyck(T)`, c'est-à-dire une fonction `Arbre(L)` qui, étant donnée une liste L qui est le mot de Dyck de T, retourne l'arbre T.

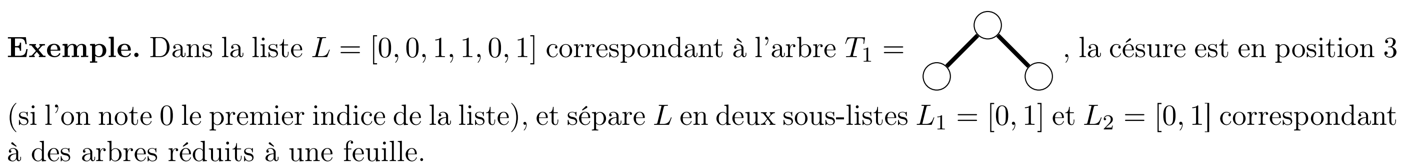
Pour cela, on définit la quantité $\delta(L)$ comme la différence entre le nombre de 0 dans L et le nombre de 1 dans L :

$$\delta(L) = |\{x \in L, x = 0\}| - |\{x \in L, x = 1\}| .$$

On peut alors montrer que si L est la représentation de Dyck d'un arbre $T = (G, D)$, alors il existe un **unique** couple (`L1`, `L2`) de sous-listes de L, telles que

```
L == [0] + L1 + [1] + L2
```

et telles que $\delta($`L1`$) = 0$ et $\delta($`L2`$) = 0$. Alors, la liste `L1` est la représentation de Dyck de G et `L2` est la représentation de Dyck de D. Remarquez que les listes `L1` et `L2` peuvent être des listes vides. L'indice (dans la liste `L`) du bit 1 qui suit la liste `L1` est appelé césure de `L`.

Exemple. Dans la liste $L = [0, 0, 1, 1, 0, 1]$ correspondant à l'arbre $T_1 =$  , la césure est en position 3 (si l'on note 0 le premier indice de la liste), et sépare L en deux sous-listes $L_1 = [0, 1]$ et $L_2 = [0, 1]$ correspondant à des arbres réduits à une feuille.

Dans toute la suite, on dit que L est *bien formée* si L est la représentation de Dyck d'un arbre binaire.

Question 8. Écrire une fonction itérative `Cesure(L)` qui calcule la césure de `L`, en supposant que `L` est bien formée.

```
def Cesure(L):
    s = 1
    i = 1
    m = len(L)
    while i<m and s>0:
        if L[i] == 0:
            s += 1
        else:
            s -= 1
        i += 1
    return i-1
```

Question 9. Donnez la complexité **en pire cas et en meilleur cas** de `Cesure(L)`, où `L` a longueur $m = 2n$, en fonction de n. Vous justifierez votre réponse, et vous supposerez que vos listes sont implantées comme des listes doublement chaînées.

On fait un parcours de liste élément par élément, donc la complexité est en $\mathcal{O}(m) = \mathcal{O}(n)$. Dans le meilleur des cas, L commence par $[0, 1, *, \dots, *]$, c'est-à-dire $i = 1$, et la complexité est constante.

Question 10. En déduire une fonction `Arbre(L)` qui construit l'arbre binaire dont `L` est une représentation de Dyck. Quelle est sa complexité en pire cas ? en meilleur cas ?

```
def Arbre(L):
    if L == []:
        return ABvide()
    i = Cesure(L)
    return AB(Arbre(L[1:i]), Arbre(L[i+1:]))
```

Dans le pire cas, la césure est à la fin de la liste à chaque fois. La fonction `Cesure` a alors une complexité en $\mathcal{O}(n)$ (de même que les opérations sur les listes), et les appels récursifs se font sur des listes de taille $2n-2$ et 0. On a alors une complexité de la forme :

$$c_n = \mathcal{O}(n) + c_{n-1} = \cdots = \mathcal{O}(n^2)$$

Dans le meilleur des cas, la césure est au début de la liste à chaque fois. La fonction `Cesure` a alors une complexité en $\mathcal{O}(1)$ (de même que les opérations sur les listes), et les appels récursifs se font sur des listes de taille 0 et $2n-2$. On a alors une complexité de la forme :

$$c_n = \mathcal{O}(1) + c_{n-1} = \cdots = \mathcal{O}(n)$$

Remarque : l'encodage qui vous a été présenté est presque optimal en ce qui concerne le nombre de bits utilisé. En effet, on peut montrer qu'il y a approximativement $C_n = \frac{1}{n+1}\binom{2n}{n}$ arbres binaires différents de taille n (C_n est appelé le nombre de Catalan). Ainsi, il faut au moins $\log_2(C_n) \simeq 2n - \frac{3}{2}\log_2(n)$ bits pour stocker de manière univoque un arbre binaire de taille n.