2I003 – Interrogation longue 2 (1h)

30 novembre 2017

On s'intéresse dans cet exercice à un encodage de **la forme** d'un arbre binaire T en une suite de bits, qu'on appelera *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 :

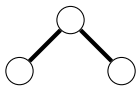
- `estABvide(T)` pour tester si un arbre `T` est l'arbre vide ;
- `T.gauche` et `T.droit` pour accéder respectivement aux sous-arbres gauche et droit de `T` ;
- `ABvide()` pour créer un arbre binaire vide ;
- `AB(G, D)` pour créer l'arbre binaire dont le sous-arbre gauche est `G`, le sous-arbre droit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

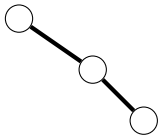
Par convention, tout arbre sera dessiné avec sa racine en haut. Aussi, lorsqu'il vous sera demandé d'écrire une fonction, vous pourrez choisir le langage/pseudo-code que vous souhaitez, tant que vos algorithmes restent clairs et concis. Si un langage autre que `python` ou `C` est choisi, veuillez le préciser en commentaire.

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.

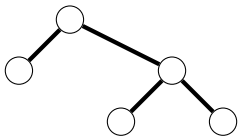
$$\mathsf{Dyck}(T) = \begin{cases} \texttt{[ ]} & \text{si } T \text{ est vide}, \\ \texttt{[0]} + \mathsf{Dyck}(G) + \texttt{[1]} + \mathsf{Dyck}(D) & \text{si } G \text{ et } D \text{ sont les sous-arbres gauche et droit de } T, \end{cases}$$

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

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

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

1 De l'arbre au mot.

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

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

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

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 `+`.

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.

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.

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.

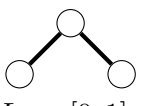
Pour cela, on définit la quantité $\delta(L)$ comme la différence entre le nombre de 0 dans L et le nombre de 1 dans L. Ainsi, pour une liste `L` de longueur $2n$,

$$\delta(\texttt{L}) = |\{i \in \{0,\dots,2n-1\}, \texttt{L[i]} = 0\}| - |\{i \in \{0,\dots,2n-1\}, \texttt{L[i]} = 1\}|\,.$$

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

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

et telles que $\delta(\texttt{L1}) = 0$ et $\delta(\texttt{L2}) = 0$. 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.

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.

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 ?

Remarque : l'encodage d'arbre binaire 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 n-ème 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.