TD9: Arbres

MP2I Lycée Pierre de Fermat

Pour les questions marquées d'un (*), des indications pour vous aider figurent à la dernière page du TD. Vous pouvez les regarder si vous bloquez ; tentez de ne pas les utiliser si vous avancez vite.

Exercice 1. *Complexité*

Les complexités des algorithmes sur les arbres peuvent s'exprimer en fonction de la taille ou de la hauteur. Intuitivement, lorsqu'une fonction récursive s'appelle sur **un** des sous-arbres, sa complexité va dépendre de la hauteur, mais lorsqu'une fonction récursive s'appelle sur **tous** les sous-arbres, sa complexité va dépendre de la taille. On note $n(a)$ la taille d'un arbre a, et $h(a)$ sa hauteur.

Q1. Rappeler les définitions récursives de $n(a)$ et $h(a)$.

On considère dans cet exercice des arbres binaires (pas forcément stricts), on note $\mathcal{A}_E$ l'ensemble de ces arbres. On considère une fonction récursive $f : \mathcal{A}_E \to X$. Pour $a \in \mathcal{A}_E$, on note $C(a)$ le coût de la fonction f sur l'arbre a.

Q1. Supposons que C vérifie le système d'équations $(*)$ suivant, où $A \in \mathbb{R}^+$:

- $C(\bullet) \leq A$
- $C(N(e, g, d)) \leq A + C(g) + C(d)$ pour tout $e \in E, g, d \in \mathcal{A}_E$.

Montrer par induction structurelle que $\forall a \in \mathcal{A}_E, C(a) \leq (2n(a) + 1)A$

Q2. Supposons maintenant que C vérifie le système d'équations $(**)$ suivant :

- $C(\bullet) \leq A$
- $C(N(e, g, d)) \leq A + \max(C(g), C(d))$ pour tout $e \in E, g, d \in \mathcal{A}_E$.

Montrer par induction structurelle que $\forall a \in \mathcal{A}_E, C(a) \leq (h(a) + 2)A$

Q3. Parmi les fonctions OCaml suivantes, déterminer celles qui vérifient $(*)$ et celles qui vérifient $(**)$, et en déduire leur complexité asymptotique par rapport à h ou n :

```
type 'a arbre =
  | V
  | N of 'a * 'a arbre * 'a arbre (* elem, sous-arbre gauche, sous-arbre droit *)

(* Renvoie la somme des éléments de a *)
let rec sum (a: int arbre) : int =
  match a with
  | V -> 0
  | N (x, g, d) -> x + sum g + sum d

(* Applique f sur chaque étiquette de a *)
let rec map (f: 'a -> 'b) (a: 'a arbre) : 'b arbre =
  match a with
  | V -> V
  | N (x, g, d) -> N (f x, map f g, map f d)
```

```
(* Renvoie l'étiquette du noeud l dans a.
   l est une liste de booléens, où true signifie gauche,
   et false signifie droite. *)
let rec etiq (l: bool list) (a: 'a arbre) : 'a =
  match l, a with
  | [], N (x, g, d) -> x
  | true :: q, N(x, g, d) -> etiq q g
  | false :: q, N(x, g, d) -> etiq q d
  | _ -> failwith "erreur etiq"
```

Q4. On considère maintenant la fonction suivante :

```
(* renvoie true si un sous-arbre non-vide de a est de somme nulle *)
let rec a_somme_nulle (a: int arbre) : bool =
  match a with
  | V -> false
  | N (x, g, d) ->
    sum a = 0 || a_somme_nulle g || a_somme_nulle d
```

Q5. Quel système d'équation vérifie la complexité de la fonction `a_somme_nulle` ?

On appelle **peigne gauche** tout arbre dont les noeuds internes ont pour enfant droit un arbre vide. Visuellement, un peigne gauche est une succession de noeuds n'ayant qu'un seul enfant, à gauche.

Q6. En considérant la famille $(a_n)_n \in \mathcal{A}_E^{\mathbb{N}}$ où a_n est un peigne gauche de taille n, montrer que `a_somme_nulle` est de complexité $\Omega(n^2)$.

Q7. Écrire une fonction ayant la spécification suivante, sans réutiliser les fonctions précédentes, en $\mathcal{O}(n)$:

```
(* Renvoie un couple (s, b), où
   - s est la somme de a
   - b est un booléen indiquant si un des sous-arbres non vide
     de a est de somme nulle *)
let somme_et_nulle (a: int arbre) : int * bool = ...
```

Q8. En déduire une version linéaire de `a_somme_nulle`.

Exercice 2.

Parcours

On considère dans cet exercice des arbres binaires. Les arbres de taille n seront étiquetés par $[\![1, n]\!]$, et l'on considèrera uniquement des arbres dont les étiquettes sont toutes distinctes. Autrement dit, les noeuds d'un arbre de taille n auront comme étiquettes exactement $1, 2, \ldots, n$.

Q1. Trouver deux arbres distincts d'au moins 5 noeuds avec des étiquettes toutes distinctes, dont les parcours préfixes donnent la même liste.

Q2. Idem pour les parcours infixes et postfixes.

Q3. Trouvez deux arbres distincts avec des étiquettes toutes distinctes ayant le même parcours postfixe et le même parcours préfixe.

Q4. (*) Montrer que si deux arbres ont le même parcours préfixe et le même parcours infixe, alors ils sont égaux, en proposant un algorithme permettant de reconstruire un arbre à partir de ses deux parcours.

Exercice 3.

Arbres d'arités quelconques

On considère les arbres binaires et les arbres d'arité quelconque stricte, que l'on peut définir selon le code OCaml suivant :

```
1 type 'a ab = (* arbres binaires *)
2   | V (* vide *)
3   | N2 of 'a * 'a ab * 'a ab (* etiquette, gauche, droite *)
4
5 type 'a arbre = (* arbres généraux *)
6   | N2 of 'a * ('a arbre list)
```

On cherche à construire une bijection entre ces deux ensembles, dont le principe est que la relation "enfant gauche de" dans les arbres binaires correspondra à la relation "premier enfant de" dans les arbres généraux, tandis que la relation "enfant droit de" dans les arbres binaires correspondra à la relation "frère de" dans les arbres généraux.

Q1. En suivant cette indication, construire deux fonctions inverses l'une de l'autre entre les types `'a ab` et `'a arbre`.

Exercice 4.

Dénombrement des arbres

On s'intéresse au nombre de noeuds des arbres binaires et des arbres binaires stricts, en faisant abstraction des étiquettes. Autrement dit, on considèrera des arbres dont tous les noeuds ont une seule et même étiquette : c'est la forme des arbres qui nous intéresse. On parle parfois de squelette d'arbre.

Q1. Montrer par induction que tout arbre binaire strict non vide a un nombre impair de noeuds.

Q2. Proposer une preuve plus directe utilisant des propriétés vues en cours.

Pour $n \in \mathbb{N}$, on note $\mathbf{AB}_n$ l'ensemble des arbres binaires de taille n, et $\mathbf{ABS}_n$ l'ensemble des arbres binaires stricts de taille $2n+1$. Pour $n \in \mathbb{N}$, on note $C_n = |\mathbf{AB}_n|$ et $D_n = |\mathbf{ABS}_n|$.

Q3. Donner C_0, C_1, C_2, C_3, C_4.

Q4. Donner D_0, D_1, D_2, D_3.

Q5. (*) Trouver une bijection entre $\mathbf{AB}_n$ et $\mathbf{ABS}_n$ pour $n \in \mathbb{N}$.

Q6. Montrer que pour $n \in N^*$, on a :

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1}$$

On peut montrer (avec les séries entières, que vous verrez l'année prochaine) que cette équation a pour solution $C_n = \frac{1}{n+1}\binom{2n}{n}$. On appelle cette suite les ***nombres de Catalan***

Q7. (*) Pour $n \in \mathbb{N}$, montrer que le nombre de mots bien parenthésés de taille $2n$ est C_n.

Les nombres de Catalan permettent de compter de très nombreux objets : les mots bien parenthésés, les triangulations de polygones, les marches aléatoires revenant en 0 en restant positives, etc... Chacune de ces situations représente donc une bijection avec les arbres binaires !

Exercice 5.

Dénombrement 2

Pour $h \in \mathbb{N}$, On note $H(h)$ le nombre d'arbres binaires de hauteur h.

Q1. Donner $H(-1), H(0), H(1)$.

Q2. Donner une formule de récurrence vérifiée par H.

Vous pouvez trouver plus d'information sur la suite $H(n)$ en cherchant les premiers termes sur oeis.org/

Exercice 6. *Rotations d'arbres*

On note $\mathcal{A}$ l'ensemble des arbres binaires stricts étiquetés aux feuilles par $\mathbb{N}$, défini inductivement :

— Pour $n \in \mathbb{N}$, $F(n) \in \mathcal{A}$, c'est une feuille ;

— Si $g, d \in \mathcal{A}$, $N(g, d) \in \mathcal{A}$, c'est un nœud interne.

On considère donc des arbres binaires n'ayant pas d'informations sur les nœuds internes.

Pour $n \in \mathbb{N}$, on note $\mathcal{A}_n \subseteq \mathcal{A}$ l'ensemble des arbres dont les étiquettes sont exactement $1, 2, \ldots, n$, sans doublons, et dont les feuilles sont ordonnées de gauche à droite.

Q1. Donner tous les éléments de $\mathcal{A}_n$ pour $n = 1, 2, 3, 4$.

Définition 1. Soit $A = N(x, N(y, g', d'), d)$ un arbre dont l'enfant gauche est non-vide. L'arbre obtenu en appliquant une rotation droite sur A est $A' = N(y, g', N(x, d', d))$. Inversement, l'arbre obtenu en appliquant une rotation gauche sur A' est A.
Notons $R_g : \mathcal{A}_E \to \mathcal{A}_E$ la rotation gauche et $R_d : \mathcal{A}_E \to \mathcal{A}_E$ la rotation droite.
On peut appliquer les rotations gauche et droite au sein d'un arbre, sur un nœud interne quelconque. On notera $R_d^x(A)$ l'arbre obtenu en effectuant une rotation droite sur le sous-arbre enraciné en x dans A, lorsque c'est possible. On note de même $R_g^x(A)$ pour une rotation gauche.

Q2. Représenter graphiquement l'effet des rotations droites et gauche.

Q3. Montrer que $\mathcal{A}_n$ est stable par rotations.

On appelle ***peigne gauche*** tout arbre dont tous les nœuds internes ont pour enfant droit une feuille. On définit de manière analogue les peignes droit.

Q4. Quel est l'unique peigne droit dans $\mathcal{A}_n$? Et l'unique peigne gauche ?

Q5. Donner une suite de rotations permettant de transformer le peigne droit en peigne gauche.

Q6. Montrer que pour $A, B \in \mathcal{A}_n$, il existe une suite de rotations permettant de transformer A en B.

On appelle ***distance de rotation*** de deux arbres A, B le nombre minimal de rotations à appliquer pour passer de A à B.

Q7. Donner une borne supérieure sur la distance de rotation de deux arbres $A, B \in \mathcal{A}_n$.

Il n'existe pas à ce jour d'algorithme efficace (i.e. polynomial) permettant de calculer la distance de rotation exacte de deux arbres : c'est un problème ouvert.

Indications

- **Exercice 2, Question 4** : Commencez par déterminer la racine de l'arbre
- **Exercice 3** : On pourra commencer par coder la fonction auxiliaire suivante :

```
1 (* ab_fratrie [a1; a2; ... ak] renvoie un arbre binaire contenant les mêmes étiquettes que
2    a1, a2 ... et ak, et transformant la relation ``être frère de" en ``être enfant droit de"
3    et la relation ``être premier enfant de" en ``être enfant gauche de" *)
4 let rec ab_fratrie (l: 'a arbre list) : 'a ab =
5   match l with
6   | [] -> V
7   | N (x, lf) :: q -> ...
```

- **Exercice 4, Question 5** : Qu'obtient-t'on si l'on enlève les feuilles d'un arbre binaire strict ?
- **Exercice 4, Question 7** : On pourra décomposer un mot bien parenthésé selon l'indice de la parenthèse fermante correspondant à la première parenthèse ouvrante.