Arbres #4 : arbres n-aires

Pierre Karpman

Lycée Champollion MP2I
https://membres-ljk.imag.fr/Pierre.Karpman/CPGE/2025/

Motivation

On peut représenter n'importe structure d'arbre n-aire avec n borné en définissant un nœud comme ayant n enfants (éventuellement vides)

Par ex. en OCaml :

```
type 'a ttree =
    | E
    | N of 'a * 'a ttree * 'a ttree * 'a ttree
```

- adapté pour des *n pas trop grands*, mais vite laborieux

Quid de n non bornés ou grands ?

Principe

On peut représenter un arbre n-aire en définissant un nœud par son étiquette, et la forêt (l'ensemble) de ses enfants

Par ex. en OCaml :

```
type 'a ntree = N of 'a * 'a ntree list
```

- Représenter un arbre vide n'est plus nécessaire *au sein du type lui-même* ; on peut utiliser un `type 'a ntree option` pour ça si besoin
- Les enfants deviennent désordonnés
  - On *pourrait* les ordonner et matérialiser des enfants vides, mais probablement peu pertinent

Remarque

À peu de choses près une forêt de tels arbres ainsi représentés aura essentiellement la même représentation que celle d'un graphe par (liste de) listes d'adjacence

Représentation C

Plusieurs options de la forme :

```
struct ntree
{
    t children;
    int data;
};
```

avec `t` un type permettant de stocker un ensemble d'enfants, par ex. une liste chaînée, un tableau (avec information de taille)…

Représentation C *bis*

Alternativement :

```
struct ntree
{
    struct ntree *first_child;
    struct ntree *next_sibling;
    int data;
};
```

- `first_child` pointe vers le « premier » enfant du nœud ou est nul s'il n'y en a pas
- `next_sibling` pointe vers son « prochain » adelphe, ou est nul s'il n'y en a plus

Illustrations

Fonctions sur les arbres n-aires : exemple OCaml

Taille

Définie comme le nombre de nœuds

- Version littérale mutuellement récursive :

```
let rec size_forest
    = function
    | [] -> 0
    | x::xs -> (size_tree x) + (size_forest xs)
and
size_tree (N (_, kids)) = 1 + (size_forest kids)
```

- Version avec itérateur :

```
let rec size (N (_, kids)) =
    List.fold_left (+) 1 (List.map size kids)
```

Fonctions sur les arbres *n*-aires *bis*

Hauteur

Définie récursivement comme un plus la hauteur maximale des enfants ; pareil que la taille *mutatis mutandis*

Parcours

Un parcours d'un arbre *n*-aire est (toujours) une fonction permettant de visiter tous ses nœuds exactement une fois. On peut toujours définir un parcours en largeur et des parcours en profondeur (préfixe et postfixe), qui seront assez similaires aux parcours de graphe (la gestion des éventuels cycles en moins)

Conversion arbre n-aire $\leftrightarrow$ arbre binaire

On peut convertir de façon réversible un arbre n-aire en un arbre binaire, par exemple avec la construction « enfant gauche / adelphe droit », qui est précisément celle utilisée dans la seconde représentation C ci-dessus !

- L'enfant gauche d'un nœud binaire est utilisé pour stocker le « premier » enfant du nœud n-aire correspondant
- L'enfant droit est utilisé pour stocker son « prochain » adelphe

Remarques

- En convertissant ainsi un arbre n-aire, on obtient un arbre binaire qui n'a pas de sous-arbre droit (à la racine), puisque la racine d'un arbre n-aire n'a pas d'adelphe
- Un arbre binaire quelconque représente alors une forêt d'arbres n-aires

Exemple de mise en œuvre OCaml

Les conversions s'écrivent plus aisément à partir de la correspondance arbre binaire $\leftrightarrow$ forêt d'arbres n-aires

On redéfinit les :

```
type 'a ntree = NN of 'a * 'a ntree list
type 'a btree = E | BN of 'a * 'a btree * 'a btree
```

Arbre n-aire vers arbre binaire

```
let rec nf2b
    = function
    | [] -> E
    | (NN (x, kids))::sibs
        -> BN (x, nf2b kids, nf2b sibs)

let ntree2btree t = nf2b [t]
```

Exemple de mise en œuvre OCaml *bis*

Arbre binaire vers arbre *n*-aire

```
let rec b2f
    = function
    | E -> []
    | BN (x, kids, sibs)
        -> (NN (x, b2f kids))::b2f sibs

let btree2ntree
    = function
    | BN (x, k, E) -> NN (x, b2f k)
    | _ -> raise (InvalidArgument "btree2ntree")
```

Une application des arbres *n*-aires : les arbres préfixes (en anglais : *tries*)

Objectif

Représenter de façon compacte un ensemble de listes pouvant avoir des *préfixes communs*

Approche

Les *chemins* dans une *forêt* d'arbres *n*-aires représentent les listes ; chaque nœud indique si le chemin s'y terminant fait partie de l'ensemble

- Gain en espace possiblement quadratique en le nombre d'éléments dans l'ensemble
- Chercher la présence d'un élément dans l'ensemble : en suivant le chemin : linéaire en la taille