IFT2015 hiver 2010 — Notes de cours

Miklós Csűrös

10 février 2010

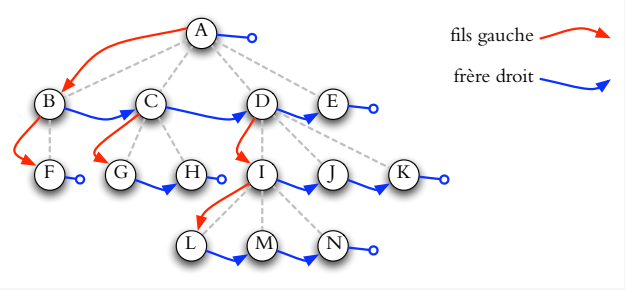
7 Arbres — représentation et parcours

7.1 Représentation d'un arbre numeroté

. Arbre = ensemble d'objets représentant de nœuds + relations parent-enfant. En général, on veut retrouver facilement le parent et les enfants de n'importe quel nœud.

```
public class TreeNode
{
   TreeNode parent;
   TreeNode enfant_gauche;
   TreeNode enfant_droit;
   // ... d'autre information
}
```

Si l'arbre est d'arité k, alors on peut avoir `TreeNode[] enfants` avec `enfants.length` $= k$.



Si l'arité de l'arbre n'est pas connu en avance (ou la plupart des nœuds ont très peu d'enfants), on peut utiliser une liste pour stocker les enfants : c'est la représentation **fils-gauche**, **frère-droit** (*first-child, next-sibling*)

7.2 Parcours

Dans un parcours, tous les nœuds de l'arbre sont visités. Dans un **parcours préfixe** (*preorder traversal*), chaque nœud est visité avant que ses enfants soient visités. Dans un **parcours postfixe** (*postorder traversal*), chaque nœud est visité après que ses enfants sont visités.

Algo PARCOURS-PRÉFIXE(x)
1 **si** $x \neq$ null **alors**
2 visiter x
3 **pout** $i \leftarrow 1, \ldots, k$ **faire**
4 PARCOURS-PRÉFIXE(enfant(x, i))

Algo PARCOURS-POSTFIXE(x)
1 **si** $x \neq$ null **alors**
2 **pour** $i \leftarrow 1, \ldots, k$ **faire**
3 PARCOURS-POSTFIXE(enfant(x, i))
4 visiter x

(enfant(x, i) donne l'enfant de x étiqueté par i — s'il n'y en a pas, alors null). Maintenant PARCOURS-. . . (racine) visite tous les nœuds dans l'ordre souhaité.

On peut parcourir un arbre binaire aussi dans l'ordre infixe. Dans un **parcours infixe** (*inorder traversal*), chaque nœud est visité après son enfant gauche mais avant son enfant droit.

```
Algo PARCOURS-INFIXE(x)
1 if x ≠ null then
2     PARCOURS-INFIXE(gauche(x))
3     visiter x
4     PARCOURS-INFIXE(droit(x))
```

Exercice 7.1. *Montrer le code pour un parcours postfixe quand l'arbre est stocké en format fils-gauche-frère-droit.*

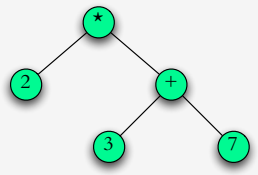
Un parcours préfixe ou postfixe se peut faire aussi à l'aide d'une pile. Si au lieu de la pile, on utilise une queue, alors on obtient un **parcours par niveau**.

```
Algo PARCOURS-PILE
1 initialiser la pile P
2 P.push(racine)
3 tandis que P n'est vide
4     x ← P.pop()
5     si x ≠ null alors
6         visite préfixe de x
7         pour i ← 1, . . . , k faire
8             P.push(enfant(x, i))
9         visite postfixe de x
```

```
Algo PARCOURS-NIVEAU(x)
1 initialiser la queue Q
2 Q.enqueue(racine)
3 tandis que Q n'est vide
4     x ← Q.dequeue()
5     si x ≠ null alors
6         visite de x en parcours par niveau
7         pour i ← 1, . . . , k faire
8             Q.enqueue(enfant(x, i))
```

La structure récursive de l'arbre permet des solutions naturelles par récurrences. Logique générale : traiter la racine, sous-arbre gauche, et le sous-arbre droit + calcul avec les valeurs. Exemples : calculer taille de l'arbre, la hauteur ou le niveau des nœuds (en parcours postfixe, postfixe et préfixe, dans cet ordre).

7.3 Expressions arithmétiques



notation infixe: 2*(3+7)
notation préfixe: * 2 + 3 7
notation postfixe: 2 3 7 + *

Une expression arithmétique peut être representée par un **arbre syntaxique**. Parcours différents du même arbre mènent à des représentations différentes de la même expression. (L'arbre montre l'application de règles dans une grammaire formelle pour expressions : $E \to E + E | E * E | \textit{nombre}$).

Une opération arithmétique a op b est écrite en **notation polonaise inverse** ou notation «postfixée» par $a\, b$ op. Avantage : pas de parenthèses ! Exemples : $1 + 2 \to 1\,2\,+$, $(3 - 7) * 8 \to 3\,7\,-\,8\,*$.

PostScript est un langage de programmation qui utilise une pile dans l'exécution de code. Opérations arithmétiques en Postscript : `add`, `sub` `mul` `div`. P.e., la suite d'instructions `5 2 sub` place 5 et 2 sur la pile (dans cet ordre), et l'opérateur `sub` prend les deux éléments en haut de la pile pour les remplacer par le résultat de la soustraction. Dans ce cas-ci, la pile contiendra le seul élément 3 à la fin. Toutes les fonctions et commandes de contrôle enlèvent leurs arguments de la pile et y placent les valeurs retournées (s'il y en a). P.e., le code "`b {10 20 moveto 30 40 lineto} if`" dessine une ligne entre les points $(10, 20)$ et $(30, 40)$ si b est vrai.