Chapitre 7

Parcours des Graphes et des Arbres

7.1 Parcours des arbres

Nous ne considérons ici que des **arbres ordonnés**.

Parcours en profondeur

Dans un *parcours en profondeur d'abord*, on descend le plus profondément possible dans l'arbre puis, une fois qu'une feuille a été atteinte, on remonte pour explorer les autres branches en commençant par la branche « la plus basse » parmi celles non encore parcourues ; les fils d'un nœud sont bien évidemment parcourus suivant l'ordre sur l'arbre.

PP(A)
  si A n'est pas réduit à une feuille **faire**
    pour tous les fils u de `racine`(A) **faire dans l'ordre** PP(u)

Les parcours permettent d'effectuer tout un ensemble de traitement sur les arbres. La figure 9.1 présente trois algorithmes qui affichent les valeurs contenues dans les nœuds d'un arbre binaire, suivant des parcours en profondeur préfixe, infixe et postfixe, respectivement.

PRÉFIXE(A)
  si $A \neq$ NIL **faire**
    affiche `racine`(A)
    PRÉFIXE(FILS-GAUCHE(A))
    PRÉFIXE(FILS-DROIT(A))

INFIXE(A)
  si $A \neq$ NIL **faire**
    INFIXE(FILS-GAUCHE(A))
    affiche `racine`(A)
    INFIXE(FILS-DROIT(A))

POSTFIXE(A)
  si $A \neq$ NIL **faire**
    POSTFIXE(FILS-GAUCHE(A))
    POSTFIXE(FILS-DROIT(A))
    affiche `racine`(A)

FIGURE 7.1 – Parcours préfixe, infixe et postfixe d'un arbre.

Parcours en largeur

Dans un *parcours en largeur d'abord*, tous les nœuds à une profondeur i doivent avoir été visités avant que le premier nœud à la profondeur $i+1$ ne soit visité. Un tel parcours nécessite que l'on se souvienne de l'ensemble des branches qu'il reste à visiter. Pour ce faire, on utilise une *file* (ici notée F).

PL(A)
  $F \leftarrow \{$`racine`$(A)\}$
  tant que $F \neq \emptyset$ **faire**
    $u \leftarrow$ SUPPRESSION(F)
    pour tous les fils v de u **faire dans l'ordre** INSERTION(F, v)

7.2 Parcours des graphes

Le parcours des graphes se révèle être un peu plus compliqué que celui des arbres. En effet, les graphes peuvent contenir des cycles et nous voulons éviter de parcourir indéfiniment ces cycles ! Pour éviter cet écueil, soit :

— on colorie les sommets des graphes : initialement les sommets sont tous blancs ; lorsqu'il est rencontré pour la première fois un sommet est peint en gris ; lorsque tous ses successeurs dans l'ordre de parcours ont été visités, un sommet est repeint en noir.

— on gére une liste de sommets déjà visités

7.2.1 Parcours en profondeur

Nous pouvons spécifier l'opération de parcours comme une fonction qui renvoie une liste de sommets parcourus :

```
parcprof:      Graphe   S   -->   Liste
pré parcprof(g,x)=exs(g,x)
```

On peut écrire plusieurs versions de cette opération : récursive ou itérative.

Version récursive

On passe par une fonction auxiliaire ayant comme paramètres l'ensemble et la liste en cours :

```
parcprofaux:      Graphe   Ens  Liste   -->   Liste
```

Avec

PARCPROFAUX(G, e, l) =
  si vide(e) **alors** l
  sinon
    avec z=choix(e)
    si appartient(l,z) **alors**
      parcprofaux(g,sup(e,z),l)
    sinon
      parcprofaux(g,sup(e,z),parcprofaux(g,esucc(g,z),adjq(l,z)))

L'appel initial est :

```
parcprof(g,x)=parcprofaux(g,esucc(g,x),adjq(listenouv(),x))
```

On voit qu'à chaque étape on doit choisir un élément z dans l'ensemble courant e : d'où de multiples possibilités.

On peut également mélanger récursif et itératif :

PARCPROFAUX(G, e, l) =
  pour tout $z \in e$ **faire**
    si non appartient(l,z) **alors**
      parcprofaux(g,esucc(g,z),adjq(l,z))

A présent, pour parcourir la totalité d'un graphe il faut répéter l'appel à parcprof(g,x) précédent pour tous les sommets x de es(g), en gardant pour chaque nouveau sommet la liste l obtenue pour les précédents. Ou bien on fait appel à parcprofaux(g,es(g), listenouv()).

Voici une version récursive qui parcourt la totalité des sommets d'un graphe en utilisant un coloriage des sommets plutôt que des listes :

PARCPROF(g)
  pour chaque sommet u de g **faire** $couleur[u] \leftarrow$ BLANC
  pour chaque sommet u de g **faire si** $couleur[u] =$ BLANC **alors** VISITER-PP(g, u, $couleur$)

VISITER-PP(g, s, *couleur*)
  couleur$[s] \leftarrow$ GRIS
  pour tout $v \in esucc(G,s)$ **faire**
    si *couleur*$[v]$ = BLANC **alors** VISITER-PP(g, v, *couleur*)
  couleur$[s] \leftarrow$ NOIR

Notez que l'utilisation d'un coloriage revient à définir une table qui associe à chaque sommet une couleur. Cela revient donc à remplacer la liste et la fonction d'appartenance à la liste par une table dont la complexité est constante : O(1).

A la place d'une liste, il est aussi possible d'obtenir une **forêt de recouvrement** : au lieu de renvoyer une liste, on renvoie une forêt, c'est à dire un ensemble d'arbres généraux. Les arcs du graphe se classent alors par rapport à cette forêt en trois types : arcs en arrière, arcs en avant, arcs croisés. Nous verrons un exemple de forêt de recouvrement en TD.

Version itérative

Une solution simple consiste à dérécursiver l'algorithme du parcours auxiliaire (récursivité multiple) en utilisant une pile dont les éléments sont des **ensembles**.

PARCPROFAUX(g, x)
  avec p =empiler(pilenouv(),esucc(g,x)) **et** l =adjq(listenouv(),x)
  tant que non vide(p) **faire**
    e =sommet(p)
    si vide(e) **alors** p=depiler(p)
    sinon
      avec z =choix(e)
      si appartient(l,z)
        alors p =empiler(depiler(p),sup(e,z))
        sinon
          p =empiler(empiler(depiler(p),sup(e,z)),esucc(g,z))
          l =adjq(l,z)

Ici les appels à empiler correspondent trivialement aux appels de parcprofaux dans le cas récursif. Mais l'utilisation d'une pile d'ensembles peut s'avérer incommode dans certains cas de figure. Il est alors possible de n'utiliser qu'une pile de sommets du graphe. Dans ce cas, l'écriture du parcours non récursif est un peu plus complexe. L'écriture de cet algorithme est laissé en exercice.

Une variante du parcours en profondeur de tous les sommets du graphes est la recherche d'un chemin de x à y. Si l'on ne recherche qu'un chemin (ou à déterminer l'existence d'un chemin), il suffit d'ajouter une condition d'arrêt : dès que y est atteint. Pour trouver l'ensemble de tous les chemins de x à y il faut gérer une liste de listes, chaque liste de sommets représentant un chemin. Le recherche d'un chemin ou de l'ensemble des chemins sera vu en TP.

7.2.2 Parcours en largeur

Dans ce type de parcours la stratégie consiste à traiter les sommets accessibles par ondes successives : comme pour le parcours d'arbre, on utilise une file, où l'on place en queue les sommets issus de l'ensemble des successeurs. On utilise une fonction intermédiaire parclargf :

```
parclarg:   Graphe   S          -->   Liste
parclargf:  Graphe  File Liste  -->  Liste

pré parclarg(g,x)=exs(g,x)

parclarg(g,x)=parclargf(g, adjq(filenouv(),x), listenouv())
```

Avec

```
PARCLARGF(g, f, l)
  si non vide(f) faire
    z =tete(f)
    si appartient(l,z) alors
      parclargf(g,suptete(f),l)
    sinon
      parclargf(g,adjqes(suptete(f), g, z),adjq(f,z))
```

Ici la fonction adjqes(f,g,z) met en queue de la file f tous les éléments de esucc(g,z).

Notez que cet algorithme est trivial à dérécursiver puisque c'est une récursivité terminale. Voici un exemple de parcours complet, utilisant une table de marqueurs, en version itérative :

```
PL(G, s)
  couleur[s] ← GRIS
  pour chaque sommet u de G, u ≠ s faire couleur[u] ← BLANC
  F ← {s}
  tant que F ≠ ∅ faire
    u ← SUPPRESSION(F)
    pour tout v ∈ esucc(G,u) faire
      si couleur[v] = BLANC
        alors couleur[v] ← GRIS
              ADJQ(F,v)
    couleur[u] ← NOIR
```

Comme pour les arbres, le traitement effectué à chaque sommet peut être préfixe ou postfixe (pour infixe cela dépend du nombre de successeurs d'un sommet - il y a plusieurs infixe possibles). Il est aussi possible de marquer le nombre de passage $k(x)$ d'un sommet x plutot que de marquer le sommet par une couleur. On peut alors effectuer un traitement $pr(x,k)$ qui dépend du nombre k, et appliquer pr autant de fois que l'on rencontre le sommet x (et pas uniquement un traitement lors de la première visite). Au premier passage le traitement est pr(x,0), au second pr(x,1), etc. Pour savoir si le sommet a déjà été visité, et donc éviter les boucles infinies, il suffit de vérifier que $k == 0$.