Parcours en profondeur en itératif – L3

15 octobre 2020

Correction de l'exercice 4 de la feuille de TD 3.

Version itérative de l'algorithme de parcours en profondeur

Ici l'objectif est de faire l'équivalent de `PPC(G)` mais sans récursivité. Et on veut obtenir les mêmes dates `d` et `f` qu'avec `PPC`.

```
Procedure PPC_iteratif(G)
//G=(S,A)
Pour chaque x dans S :
    Couleur[x] := blanc
    Pi[x] := nil
temps:=0
P := Pile vide
Pour chaque s dans S:
    Si Couleur[s] == blanc Alors :
P.push(s)
Tant que P est non vide Faire
   x := P.pop()
   Si Couleur[x] = blanc Alors
     Couleur[x] := gris
     temps ++
     d[x] := temps
     P.push(x)
     Pour chaque (x,u) dans A
       Si couleur[u] := blanc Alors
         PI[u] := x
         P.push(u)
   Sinon Si couleur[x] = gris :
     couleur[x] := noir
     temps ++
     f[x] := temps
Retourner (d,f,Pi)
```

On va utiliser une pile P pour stocker les sommets en attendant de les explorer. L'idée est, lorsqu'on découvre un sommet x, de le colorier en gris, de le remettre dans la pile et d'empiler tous ses successeurs immédiats encore blancs (tous les sommets y tels que $(x, y) \in A$ et $\mathsf{Couleur}[y] = \mathsf{blanc}$), puis de dépiler un de ses sommets de le colorier en gris *etc.* La difficulté vient dans la gestion du dépilement : entre le moment où un sommet est inséré dans la pile et le moment où il est dépilé, ce sommet a pu changer de couleur parce que découvert entretemps par un autre arc depuis un autre sommet. On distingue trois cas selon la couleur du sommet dépilé x :

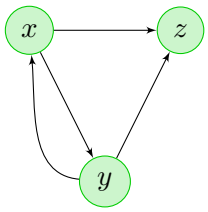
— x est blanc : on le colorie en gris et on empile ses voisins encore à explorer (blancs) ;

— x est gris : on en a fini d'explorer ses voisins, il est temps de colorier x en noir et de dépiler...

— x est noir : il n'y a rien à faire, on dépile...

Prenons l'exemple du graphe 1. Après avoir empilé x (colorié en gris), on peut ajouter z puis y (les deux sont alors blancs). Lorsqu'on dépile y, on le colorie en gris, puis one le ré-empile, ainsi que z. On n'empile pas x car il est gris. On note qu'à ce moment là, la pile P contient $[x, z, y, z]$ avec x et y en gris et z en blanc. Notons que z apparaît deux fois : car on est dans l'attente d'examiner deux arcs menant à z : (x, z) et (y, z). Ensuite l'algorithme va dépiler z, le colorier en gris, le réempiler, comme il n'a pas de successeur blanc, et on va immédiatement le dépiler et comme il est gris, on va le colorier en noir. Puis on dépilera y, lui aussi en gris, on le coloriera en noir, puis on dépilera z, donc en noir, il n'y aura rien à faire, puis enfin x qui est toujours gris, et que l'on coloriera en noir. C'est fini. Les dates d et f sont alors indiquées dans la figure. Notons bien-sûr que ce résultat dépend de la manière d'énumérer les successeurs des sommets (donc de l'ordre d'empiler dans P !).

Plus généralement, la correspondance entre les deux algorithmes peut s'illustrer ainsi : à un moment donné de l'exécution de la version récursive, on a un chemin de sommets gris $x_1 x_2 \dots x_k$ et pour chaque x_i on dispose d'une liste d'arcs (cette information est contenue dans la pile des appels récursifs). Cette situation se traduira dans la version itérative par une pile P qui contient $[x_1] + L_1 + [x_2] + L_2 + \dots + [x_k] + L_k$ où chaque L_i est la liste des arcs restant à explorer depuis x_i...



sommet	x	y	z
d	1	2	3
f	6	5	4

FIGURE 1: Exemple

Et en Python...

Un programme pour tester... Et que faut-il faire sur le graphe pour comparer cet algorithme avec la version récursive ?

```
def PPCit(G) :
    S,A=G[0],G[1]
    Couleur, d, f, Pred, T = { }, { }, { }, { }, 0
    P = []
    for s in G[0] :
        Couleur[s] = 'blanc'
        Pred[s] = None
    for s in G[0] :
        if Couleur[s] == 'blanc' :
            P.append(s)
            while len(P)>0 :
                x=P.pop()
                if Couleur[x]=='blanc' :
                    Couleur[x]='gris'
                    T+=1
                    d[x]=T
                    P.append(x)
                    for y in A[x] :
                        if Couleur[y]=='blanc' :
                            Pred[y]=x
                            P.append(y)
                elif Couleur[x]=='gris' :
                    Couleur[x]='noir'
                    T+=1
                    f[x]=T
    return (d,f,Pred)
```