Algorithmique des graphes : TD1

Alain Cournier Stéphane Devismes

Dans ces corrections OuSinon désigne l'opérateur logique du ou paresseux. De même Le EtAlors désignera l'opérateur logique du et paresseux.

1 Exercice 2

Algorithme 1 Fonction *DegSortant* avec résultat entier (version matricielle)

Entrées

1: G : graphe de n sommets
2: x : sommet

Variables

3: d, i : entiers

Programme

4: $d \leftarrow 0$
5: **Pour** i **de** 0 **à** $n-1$ **faire**
6: **Si** $G[x][i]$ **alors**
7: $d \leftarrow d+1$
8: **Fin Si**
9: **Fin Pour**
10: `renvoyer` d

Algorithme 2 Fonction *DegSortant* avec résultat entier (version listes d'adjacence)

Entrées

1: G : graphe de n sommets
2: x : sommet

Programme

3: `renvoyer` $Longueur(G[x])$

2 Exercice 3

Algorithme 3 Procédure *CalculDegresEntrants* (version matricielle)

Entrées

G : graphe de n sommets

Sorties

$DegEnt[0..n-1]$: tableau de n entiers

Variables

i, j : entiers

Programme

Pour i **de** 0 **à** $n-1$ **faire**
    $DegEnt[i] \leftarrow 0$
Fin Pour
Pour i **de** 0 **à** $n-1$ **faire**
    Pour j **de** 0 **à** $n-1$ **faire**
        Si $G[i][j]$ **alors**
            $DegEnt[j] \leftarrow DegEnt[j] + 1$
        Fin Si
    Fin Pour
Fin Pour

Algorithme 4 Procédure *CalculDegresEntrants* (version listes d'adjacence)

Entrées

G : graphe de n sommets

Sorties

$DegEnt[0..n-1]$: tableau de n entiers

Variables

i : entier
L : liste de sommets

Programme

Pour i **de** 0 **à** $n-1$ **faire**
    $DegEnt[i] \leftarrow 0$
Fin Pour
Pour i **de** 0 **à** $n-1$ **faire**
    $L \leftarrow G[i]$
    Tant que `NON` $TestListeVide(L)$ **faire**
        $DegEnt[Premier(L)] \leftarrow DegEnt[Premier(L)] + 1$
        $L \leftarrow Suite(L)$
    Fin Tant que
Fin Pour

3 Exercice 4

Algorithme 5 Fonction *SansCircuits* avec résultat booléen (version générale)

Entrées

1: G : graphe de n sommets

Variables

2: L : liste de sommets
3: $DegEnt[0..n-1]$: tableau de n entiers
4: u, v : sommets
5: Cpt : entier

Programme

6: $Cpt \leftarrow 0$
7: $CalculDegresEntrants(G, DegEnt)$
8: $L \leftarrow ListeVide()$
9: **Pour tout** $u \in X$ **faire**
10: **Si** $DegEnt[u] = 0$ **alors**
11: $InsererTete(L, u)$
12: **Fin Si**
13: **Fin Pour**
14: **Tant que** `NON` $TestListeVide(L)$ **faire**
15: $u \leftarrow RetirerPremier(L)$
16: $Cpt \leftarrow Cpt + 1$
17: **Pour tout** $v \in Succ(u)$ **faire**
18: $DegEnt[v] \leftarrow DegEnt[v] - 1$
19: **Si** $DegEnt[v] = 0$ **alors**
20: $InsererTete(L, v)$
21: **Fin Si**
22: **Fin Pour**
23: **Fin Tant que**
24: `renvoyer` $Cpt = n$

4 Exercice 5

Algorithme 6 Procédure *Dual* (version matricielle)

Entrées

G : graphe de n sommets

Sorties

GD : graphe de n sommets

Variables

u, v : sommets

Programme

```
Creer(GD, n)
Pour tout u ∈ X faire
    Pour tout v ∈ X faire
        GD[v, u] ← G[u, v]
    Fin Pour
Fin Pour
```

Algorithme 7 Procédure *Dual* (version listes d'adjacence)

Entrées

G : graphe de n sommets

Sorties

GD : graphe de n sommets

Variables

L : liste

u : sommet

Programme

```
Creer(GD, n)
Pour tout u ∈ X faire
    L ← G[u]
    Tant que NON TestListeVide(L) faire
        InsererTete(GD[Premier(L)], u)
        L ← Suite(L)
    Fin Tant que
Fin Pour
```

5 Exercice 6

Algorithme 8 Fonction *Symetrique* avec résultat booléen (version générale)

Entrées

G : graphe de n sommets

Variables

u, v : sommets

Programme

Pour tout $u \in X$ **faire**
    Pour tout $v \in Succ(u)$ **faire**
        Si `NON` $u \in Succ(v)$ **alors**
            `renvoyer` $faux$
        Fin Si
    Fin Pour
Fin Pour
`renvoyer` $vrai$

Algorithme 9 Fonction *Symetrique* avec résultat booléen (version optimisée pour listes d'adjacence)

Entrées

G : graphe de n sommets

Variables

GD, IGD : graphes de n sommets
u : sommet
$L1, L2$: listes de sommets

Programme

$Dual(G, GD)$
$Dual(GD, IGD)$
Pour tout $u \in X$ **faire**
    $L1 \leftarrow GD[u]$
    $L2 \leftarrow IGD[u]$
    Tant que `NON` $TestListeVide(L1)$ **faire**
        Si $TestListeVide(L2)$ $OuSinon$ $Premier(L1) \neq Premier(L2)$ **alors**
            $Liberer(GD)$
            $Liberer(IGD)$
            `renvoyer` $faux$
        Fin Si
        $RetirerPremier(L1)$
        $RetirerPremier(L2)$
    Fin Tant que
    Si `NON` $TestListeVide(L2)$ **alors**
        $Liberer(GD)$
        $Liberer(IGD)$
        `renvoyer` $faux$
    Fin Si
Fin Pour
`renvoyer` $vrai$