

Algorithmes et structures de données avancés : TD 3(Corrigé)

Graphes - Chaînes - Algorithme de Warshall

Exercice 3.1 *Connexité*

Rappeler vous du théorème suivant :

Théorème 1 *Un graphe G est connexe si et seulement si $\forall s, t \in V(G)$, il existe une chaîne entre s et t .*

- Supposons qu'il existe un algorithme en $O(|V(G)| \cdot |E(G)|)$ qui vérifie s'il y a une chaîne entre deux sommets s et t . Quelle est la complexité d'un algorithme qui vérifie si G est connexe en utilisant ce théorème ? Quelle est la règle de complexité que vous avez appliquée ?

Il y a $|V(G)|^2$ différentes permutations pour les sommets s et t . Si on doit vérifier s'il y a une chaîne en $O(|V(G)| \cdot |E(G)|)$ entre les $O(|V(G)|^2)$ permutations, la complexité de cet algorithme est de $O(|V(G)|^2) \cdot O(|V(G)| \cdot |E(G)|) = O(|V(G)|^3 \cdot |E(G)|)$. Nous avons appliqué la règle de complexité du produit.

- Considérer maintenant le théorème suivant :

Théorème 2 *Un graphe est connexe si et seulement si à partir de n'importe quel sommet $s \in V(G)$, il existe une chaîne entre s et chacun des sommets du graphe.*

- Quelle est la complexité d'un algorithme qui vérifie si G est connexe en utilisant ce nouveau théorème ?

Cette fois si le sommet s peut être fixé, il reste donc uniquement $|V(G)|$ différentes permutations pour le sommet t . Si on doit vérifier s'il y a une chaîne en $O(|V(G)| \cdot |E(G)|)$ entre ces $O(|V(G)|)$ permutations, la complexité de cet algorithme est de $O(|V(G)|) \cdot O(|V(G)| \cdot |E(G)|) = O(|V(G)|^2 \cdot |E(G)|)$. Nous avons de nouveau appliqué la règle de complexité du produit.

- Prouver ce théorème par **contradiction**.

Cette preuve se trouve sur les support du cours 8 : La preuve est par **contradiction**.

Partie "si" : Nous allons supposer qu'à partir de n'importe quel sommet s du graphe, il existe une chaîne entre s et chacun des sommets du graphe, mais, $\exists s_1, s_2 \in V(G)$, tels qu'il n'existe pas de chaîne entre s_1 et s_2 , puis montrer que cela nous donne un résultat absurde.

Il suffit pour cela de construire les chaînes $C_1 = s, \dots, s_1$ et $C_2 = s, \dots, s_2$.

Puis, on regarde $\overline{C_1}C_2 = s_1, \dots, s, \dots, s_2$ qui est une chaîne entre s_1 et s_2 .

Partie "seulement si" : Il faut prouver que si un graphe est connexe, alors à partir de n'importe quel sommet s du graphe, il existe une chaîne entre s et chacun des sommets du graphe.

C'est une conséquence immédiate de la définition de connexité.

Exercice 3.2 *Connexité (par récurrence)*

Considérer la matrice d'adjacence **A** ainsi qu'un tableau **marquer** pour marquer des sommets (ces variables seront globales pour assurer la simplicité) :

```
const n = 8;
type t_matrice_adjacence=array[1..n,1..n] of integer;
type t_tableau=array[1..n] of boolean;
var A,Aetoile : t_matrice_adjacence;
var marquer : t_matrice_tableau;
```

1. Ecrire une fonction `function connexe : boolean;` qui teste si un graphe avec la matrice d'adjacence **A** est connexe.

```
type t_tableau=array[1..n] of boolean;
var marquer : t_tableau;

procedure marquerSommets(s : integer);
var i : integer;
begin
    marquer[s] := True;
    for i:=1 to n do begin
        if ((marquer[i]=False) AND (graphe[s][i]=1)) then
            begin
                marquerSommets(i); { Appel récursif ! }
            end;
        end;
    end;
end;

function estConnexe : boolean;
var i:integer;
begin
    for i:=1 to n do
        marquer[i]:= False;

    marquerSommets(1);

    result := True;
    for i:=1 to n do
        begin
            if marquer[i]= False then
                begin
                    result := False;
                end;
            end;
        end;
    end;
end;
```

2. Ecrire une fonction `function existeChaine(s,t) : boolean;` qui teste s'il y a une chaîne entre le sommet `s` et le sommet `t`.
3. Ecrire une procédure `function fermetureTransitive;` qui calcule la fermeture transitive `Aetoile` de la matrice d'adjacence `A`. Quelle est la complexité de votre procédure ?

Exercice 3.3 Complexité

Pour calculer la fermeture transitive G^* d'un Graphe G , on peut déterminer la matrice d'adjacence A^* du Graphe G^* de la manière suivante :

$$A_c = I + A + A^2 + A^3 + \dots + A^{n-1}$$
$$A_{ij}^* = 1 \text{ si } A_{i,j} > 0 \text{ et } A_{ij}^* = 0 \forall i, j$$

1. Quelle est la complexité de calculer ce terme pour un graphe de $n=|V(G)|$ sommets ?

Exercice 3.4 Connexité (par itération)

Considérer la matrice d'adjacence A :

```
const n = 8;  
type t_matrice_adjacence=array[1..n,1..n] of integer;  
type t_tableau=array[1..n] of integer;  
var A,Aetoile : t_matrice_adjacence;
```

1. Ecrire une procédure `function fermetureTransitive`; qui calcule la fermeture transitive `Aetoile` de la matrice d'adjacence `A` avec l'algorithme de Warshall. Quelle est la complexité de votre procédure ?
2. Ecrire une fonction `function connexe : boolean`; qui teste si un graphe avec la matrice d'adjacence `A` est connexe. Quelle est la complexité de votre procédure ?
3. Ecrire une fonction `function existeChaine(s,t) : boolean`; qui teste s'il y a une chaîne entre le sommet `s` et le sommet `t`. Quelle est la complexité de votre procédure ?