PC 8
Graphes 2

Exploration des graphes

Deux stratégies possibles:

1. Exploration en profondeur ('depth-first search'): généralisation de l'ordre préfixe dans les arbres.
2. Exploration en largeur ('breadth-first search'): généralisation de l'ordre militaire.

Nous allons voir qu'on peut explorer les graphes en *temps linéaire* $O(f)$ en fonction du nombre de flèches.

On pourra ainsi faire en temps linéaire:

1. existence d'un chemin de i à j.
2. test d'existence d'un circuit
3. ...

Conséquence: fermeture transitive en $O(nf)$.

Exploration en profondeur

On suppose le graphe représenté par liste de successeurs.

On peut utiliser un tableau de listes `l[s]` ou une représentation en tableau `succ[s,i]` pour chaque sommet `s`. On place une valeur ω en fin de liste.

Le calcul à partir de la matrice d'adjacence se fait ainsi:

```
for i:= 1 to n do
 begin
  k:= 1;
  for j:= 1 to n do
   if A[i,j] then
    begin
     succ[i,k]:= j;
     k:= k+1
    end;
   succ[i,k]:=Omega;
  end;
```

L'algorithme

Couleur des sommets: blanc= initiale, gris= en cours, noir= terminé.

```
procedure exploration( s: sommet);
 var
  i: integer;
  t: sommet;
 begin

  marque[s]:= gris;
  t:= succ[s,1]; i:= 1;
  while t <> Omega do begin
   if marque[t] = blanc then
     exploration(t);
    i:= i+1; t:= succ[s,i]
  end;

  marque[s]:= noir
 end;
```

Analyse de l'algorithme

1. **Correction**: Chaque sommet est marqué une fois et une seule par la procédure `exploration`.

   $$\begin{aligned} d[s] &= \text{numéro de } s \text{ pour le début} \\ f[s] &= \text{numéro de } s \text{ pour la fin} \end{aligned}$$

   La famille des intervalles $[d[u], f[u]]$ a une structure de système de parenthèses.

2. **Complexité**: La procédure est appelée sur chaque sommet un fois et une seule. La boucle `while` est exécutée autant de fois que s a de successeurs. Le temps d'exécution est donc

   $$T = \Theta(\sum_{s \in S} |l[s]|) = \Theta(f)$$

Calcul des indices de début et fin

`temps`= variable globale initialisee a 1.

```
d[s]:= temps; temps:= temps+1;
```

```
f[s]:= temps; temps:= temps+1;
```

Forêt recouvrante

Arbre recouvrant d'un graphe G: arbre formé de flèches de G avec le même ensemble de sommets que G.

Forêt : union d'arbres disjoints.

Les flèches utilisées dans l'exploration pour atteindre un sommet non marqué forment une forêt recouvrante.

Exemple

Classification des flèches

On classe les flèches en trois catégories:

1. *en avant*: descendent dans l'arbre.
2. *en arrière*: remontent dans l'arbre
3. *de travers*: autres.

1. (u, v) en avant $\Leftrightarrow d[u] < d[v]$
2. (u, v) en arrière $\Leftrightarrow f[u] < f[v]$

Recherche de cycles

Un cycle (ou circuit) est un chemin

$$(s_0, s_1), (s_2, s_3), \ldots, (s_{n-1}, s_n)$$

tel que $s_0 = s_n$.

Un graphe est acyclique ssi on ne trouve pas de flèche arrière dans une exploration en profondeur

Tri topologique

Ordre sur les sommets tel que

$$(u, v) \in S \Rightarrow u < v$$

Application:problèmes d'ordonnancement.

Algorithme:

1. Explorer en profondeur.
2. Quand l'exploration d'un sommet est terminée, le placer en tête d'une liste.
3. La liste fournit l'ordre cherché.

Variante: on calcule les indices `f[u]` et on range u à l'indice $n - f[u]$.

Composantes fortement connexes

Classes de l'équivalence: $u \equiv v$ ssi il existe un chemin de u à v *et* un de v à u.

Algorithme:

1. Explorer pour calculer `f[u]`
2. Calculer G^t (graphe renversé)
3. Explorer G^t par ordre décroissant de f[u] (calculé en 1)
4. Les arbres de la forêt recouvrante sont les composantes f.c.

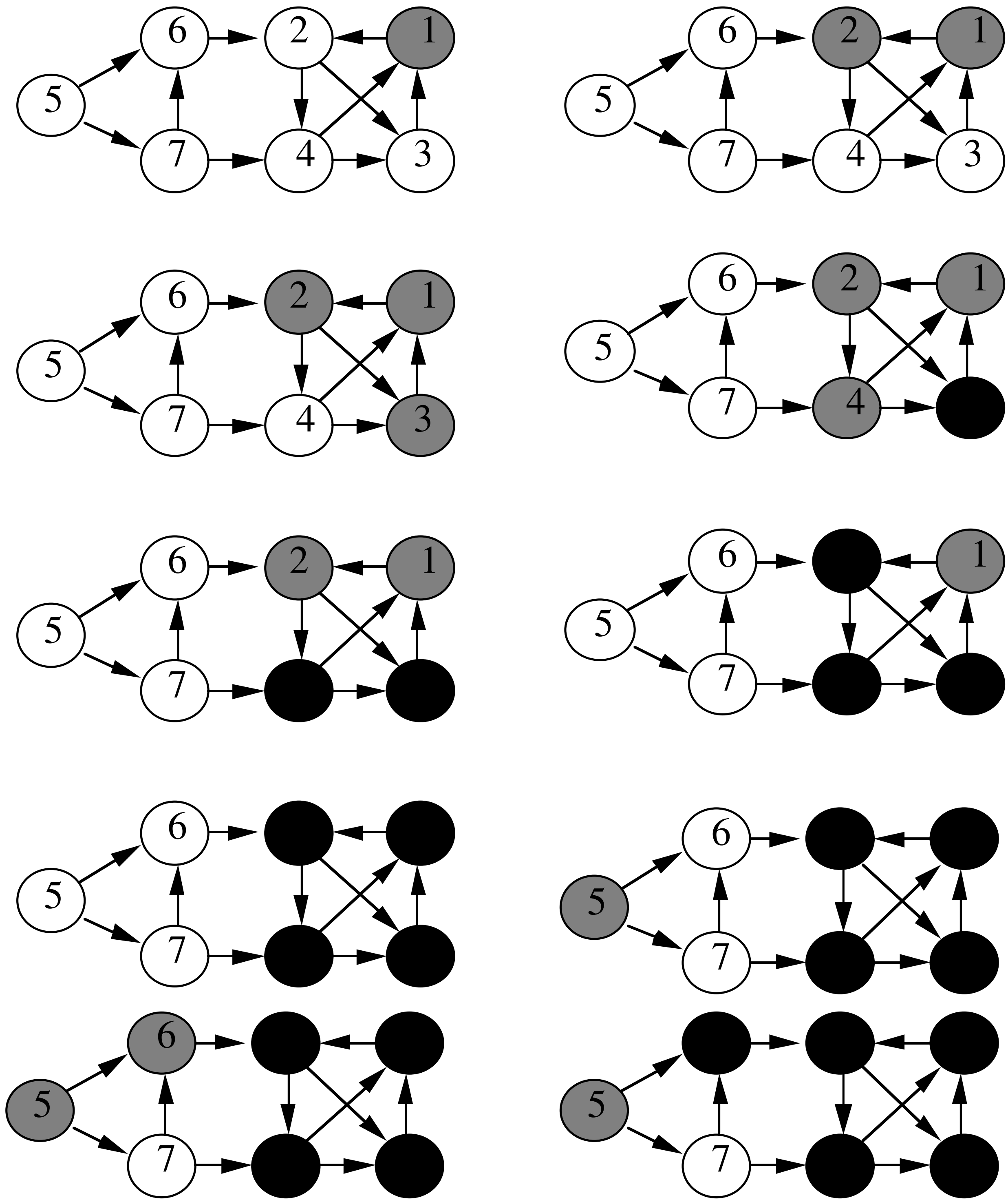

Figure 1: évolution de l'exploration

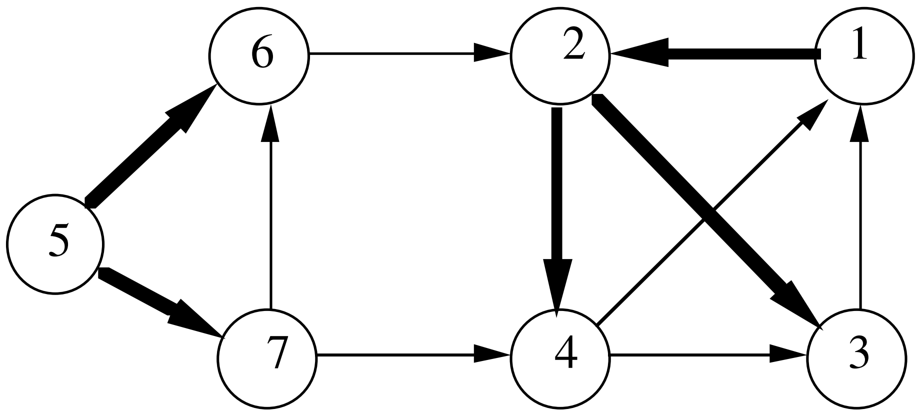

Figure 2:

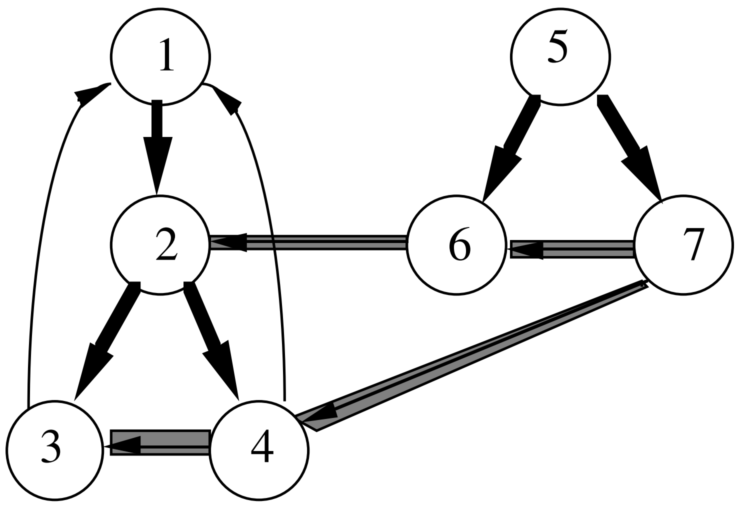

Figure 3: les trois sortes de flèches

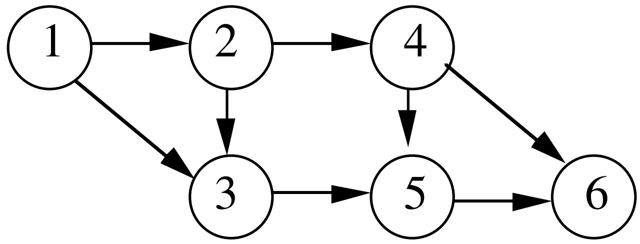

Figure 4: graphe acyclique

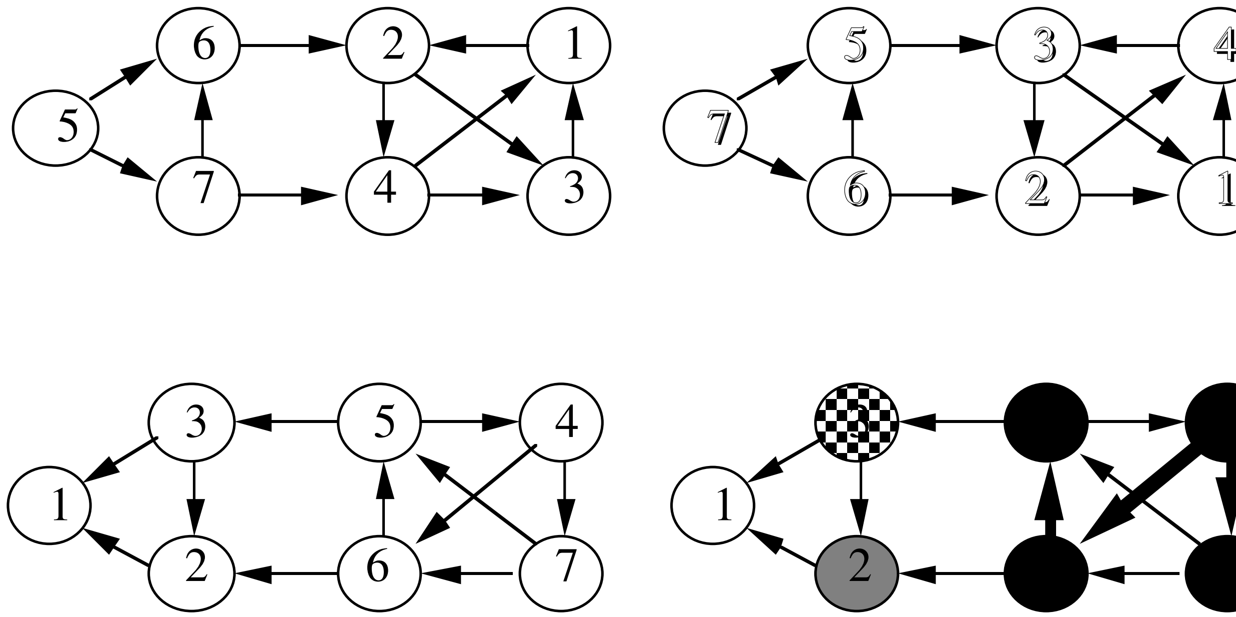

Figure 5: calcul des composantes f.c.