Techniques de parcours de graphes

Parcours en profondeur d'abord (depth-first search)

- Input : graphe
- Output : forêt d'exploration couvrante

Programme principal :

```
Tant qu'il existe sommet s tel que marque (s) = false
dfs (s) ;
fin tant que
```

Version récursive

```
procédure recursif_dfs (s : sommet) ;
var v : sommet ; marque : booléen ;
début
        marque (s) := vrai ;
        imprimer (s) ; {ordre préfixe}
        pour tout (v successeur de s) faire :
                si v non marqué alors recursif_dfs (v) ;
                finsi
        finfaire
        imprimer (s) ; {ordre suffixe}
fin.
```

Version itérative

```
procédure iter_dfs (s : sommet) ;
var v, u : sommet ; marque : booléen ; p : pile ;
début
        vide (p) ;
        marque (s) := vrai ;
        push (s, p) ;
        répéter
                u := tête (p) ;
                pour tout (v successeur de u) faire :
                        si v non marqué alors
                                marque (v) := vrai ;
                                imprimer (v) ; {ordre préfixe}
                                push (v, p) ;
                        finsi
                finfaire
                imprimer (u) ; {ordre suffixe}
                pop (p) ;
        jusqu'à p = vide ;
fin.
```

Parcours en largeur d'abord (breadth-first search)

- Input : graphe
- Output : forêt d'exploration couvrante

Programme principal :

```
Tant qu'il existe sommet s tel que marque (s) = false
bfs (s) ;
fin tant que
```

Version itérative

```
procédure bfs (s : sommet) ;
var v, u : sommet ; marque : boolean ; q : file ;
début
        vide (q) ;
        marque (s) := vrai ;
        inject (s, q) ;
        répéter
            u := tête (q) ;
            pour tout (v successeur de u) faire :
                si v non marqué alors
                    marque (v) := vrai ;
                    inject (v, p) ;
                finsi
            finfaire
            pop (q) ;
        jusqu'à q = vide ;
fin.
```

Calcul de la fermeture transitive

Algorithme de Roy-Warshall

Calcul de la fermeture transitive d'un graphe G décrit par sa matrice d'adjacence M.

```
procédure Roy-Warshall  (A : tableau [1..n, 1..n] de booléen) ;
var i, j, k : entier ;
début
   {initialisation}
   pour i := 1 à n faire
         pour j := 1 à n faire
               A[i, j] := M[i, j] ;
         finfaire
   finfaire
   {calcul de la matrice d'adjacence A de G}
   pour k = 1 à n faire
         pour i := 1 à n faire
               pour j := 1 à n faire A[i, j] := A[i, j] ou (A[i, k] et A[k, j]) ;
               finfaire
         finfaire
   finfaire
fin.
```

Plus courts chemins

Algorithme Dijkstra

Calcule l'arborescence des plus courts chemins issus d'un sommet donné « source » numéroté « 1 ». Le graphe $G(X, U, V)$ est valué positivement : $V(x, y) \geq 0\ \forall (x, y) \in U$.

$\overline{X}$: ensemble d'**entier** ;
d, père : **tableau** [1..n] **de entier** ;
début
    début {initialisation}
        $\overline{X}$ = {2, 3, .., n} ;
        d(1) := 0 ;
        pour i := 2 **à** n **faire**
            père (i) := 1 ;
            si (i successeur de 1) **alors** d(i) := V(1,i)
            sinon d(i) := $+\infty$;
        finfaire
    fin
    tant que $\overline{X} \neq \varnothing$
        sélectionner $j \in \overline{X} \,/\, d(j) = \min\limits_{i \in \overline{X}} d(j)$
        début
            $\overline{X} := \overline{X} - \{j\}$;
            pour tout $i \in \overline{X} \,/\, d(j) + V(j, i) < d(i)$ **faire**
                d(i) := d(j) + V(j,i) ;
                père (i) := j ;
            finfaire
        fin
fin.

Algorithme de Bellmann-Ford

Calcule l’arborescence des plus courts chemins issus du sommet s dans un graphe à valuation quelconque.

X : ensemble de sommets ;
s : sommet ; {source}
v, last : sommet ;
dist : **réel** ; q : **file** ; pass : **entier** ;
début
    début {initialisation}
        vide (q) ;
        enfiler (s, q) ;
        dist (s) := 0 ;
        pass := 0
        last := s ;
        pour tout $v \in X, v \neq s$ **faire**
            d(v) := $+\infty$;
            père (v) := 0 ;
        finfaire
    fin
    répéter
        v := tête (file) ;
        pop (q) ;
        pour tout $w \in succ(v) / dist(v) + val(v,w) < dist(w)$ **faire**
            dist (w) := dist (v) + val(v,w) ;
            père (w) := v ;
            si $w \notin q$ **alors** enfiler (w,q) ;
        finfaire
        si (q ≠ ∅ et v = last) **alors**
            pass := pass + 1 ;
            last := queue (file) ; {retourne le dernier élément de la file X}
        finsi
    jusqu’à (q = ∅ **ou** pass ≥ n)
{q = ∅ : terminaison normale, pass ≥ n : présence d'un circuit absorbant}
fin.

Algorithme de Floyd-Warshall (méthode matricielle)

Calcul des plus courts chemins (pcc) pour tout couple de sommets (x, y) dans un graphe G valué et décrit par sa matrice de valuation VAL; les sommets sont numérotés de 1 à n.

```
var V : tableau [1..n, 1..n] de réel ; P : tableau [1..n, 1..n] de entier ; i, j, k : entier ;
début
    début {initialisation}
        pour i := 1 à n faire
            pour j := 1 à n faire
                V[i,i] := 0 ;
                P[i,i] := i ;
                si i ≠ j alors
                    V[i,j] := VAL[i,j] ;
                    P[i,j] := i ;
                finsi
            finfaire
        finfaire
    fin
    début {calcul des pcc et de la matrice des prédécesseurs P}
        pour k := 1 à n faire
            pour i := 1 à n faire
                pour j := 1 à n faire
                        si ( i ≠ j et V[i,k] + V[k,j] < V[i,j]) alors
                            V[i,j] := V[i,k] + V[k,j] ;
                            P[i,j] := P[k,j] ;
                        finsi
                        si (i = j et V[i,k] + V[k,j] < 0) alors
                            imprimer « circuit absorbant» ;
                            arrêt ;
                        finsi
                finfaire
            finfaire
        finfaire
    fin
fin.
```

Flots

On considère un réseau avec s est le sommet source et p le sommet puit.
A chaque arc du réseau est associé un triplet (b, c, d) :

- b représente la borne inférieure de capacité (on considère ici b = 0)
- c représente la borne supérieure de capacité (c > 0).
- d représente le coût de transport d'une unité de matière sur cet arc (d positif).

Le premier problème consiste à trouver un flot f de s à p de valeur maximum, compte tenu des contraintes de capacité. Il sera résolu par l'algorithme de **Ford-Fulkerson**.

Le second problème consiste à déterminer un flot f de s à p de valeur v et de coût minimum. En particulier on pourra obtenir le flot de valeur maximum et de coût minimum. Ce problème sera résolu à l'aide de l'algorithme de **Busacker-Gowen (de Roy)**.

Algorithme de Ford-Fulkerson

La procédure principale partant d'un flot f_i de valeur v_i est la suivante.

Phase 1 :

Marquer le sommet s, $s \in A$.

Phase 2 :

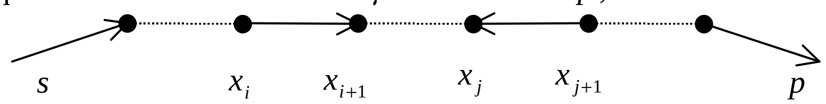
Soit A l'ensemble des sommets marqués et $x \in A$.
Marquer tous les successeurs y de x tels que $f(xy) < c(xy)$ (marquage avant).
Marquer tous les prédécesseurs y de x tels que $f(yx) > 0$ (marquage arrière).
Il s'agit d'un marquage en largeur.

Quand on ne peut plus rien marquer, deux éventualités se présentent :

- p est marqué : passer à la phase 3.
- p ne peut être marqué : passer à la phase 4.

Phase 3 :

p est marqué : on a obtenu une chaîne μ allant de s à p, où :



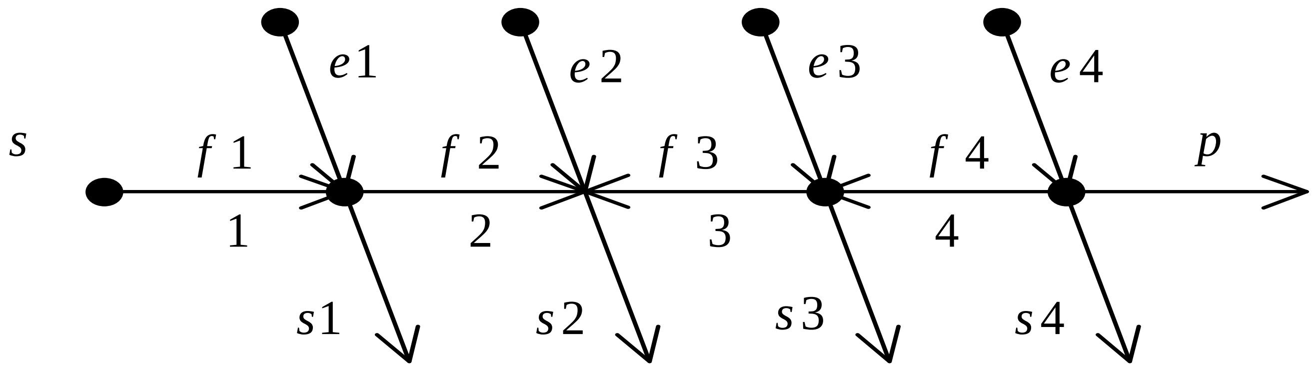
- tous les arcs $(x_i x_{i+1})$ parcourus vers l'avant $(\in \mu^+)$ sont non saturés :

  $f(x_i x_{i+1}) < c(x_i x_{i+1})$. Soit alors $m_1 = \min\{c(x_i x_{i+1}) - f(x_i x_{i+1})\}$ pour ces arcs "avant".
- tous les arcs $(x_j x_{j+1})$ parcourus vers l'arrière $(\in \mu^-)$ sont de flot non nul : $f(x_{j+1} x_j) > 0$.

  Soit alors $m_2 = \min f(x_{j+1} x_j)$ pour ces arcs "arrière".

Soit $m = \min(m_1, m_2) > 0$: on peut alors le long de cette chaîne faire passer une quantité de matière m supplémentaire de s à p ; il suffit pour cela de prendre le nouveau flot $f_{i+1}(xy)$ défini par : $\begin{cases} f_{i+1}(xy) = f(xy) \; si \; (xy) \notin \mu \\ f_{i+1}(xy) = f(xy) + m \; si \; (xy) \in \mu^+ \\ f_{i+1}(xy) = f(xy) - m \; si \; (xy) \in \mu^- \end{cases}$

On vérifie aisément que $v_{i+1} = v_i + m$, et que les nouveaux flots f_{i+1} satisferont également à la loi de conservation des flux en chaque sommet ; on peut en effet se trouver dans l'un des quatre cas possibles :



On revient à la phase 2.

Phase 4:

p ne peut être marqué.

On a ainsi construit une coupe (A, $\bar{A}$) avec A est l'ensemble des sommets marqués et $\bar{A}$ l'ensemble des non marqués $\bar{A}$ ($s \in A$, $p \in \bar{A}$).

Pour cette coupe, on a : $\left.\begin{array}{c} x \in A, f(xy) = c(xy) \\ y \in \bar{A}, f(yx) = 0 \end{array}\right\}$ sinon on aurait pu marquer y.

En sommant sur tous les $x \in A$, $y \in \bar{A}$: $f(A, \bar{A}) = c(A, \bar{A}) \quad et \quad f(\bar{A}, A) = 0$.

On trouve ainsi, que pour le flot et la coupe considérés : $f(A, \bar{A}) = f(\bar{A}, A) = c(A, \bar{A})$,

soit en vertu de la relation fondamentale flot/coupe : $v = c(A, \bar{A})$.

On est donc à l'optimum :

Le flot f est de valeur maximale et (A, $\bar{A}$) est une coupe de capacité minimale.

- En pratique, on vérifiera bien que :

- les arcs sortants de la coupe (du type $(A, \bar{A})$) sont saturés
- les arcs entrants dans la coupe (du type $(\bar{A}, A)$) sont de flot nul.

Algorithme de Roy (Busacker et Gowen)

On recherche un flot de valeur maximale (ou de valeur v donnée) et de coût minimal.

Etape 0

On part d'un flot initial de valeur $v_0 = 0$, ce flot est réalisable (tous les b sont nuls) et de coût minimal (tous les d sont positifs).

Etape i

Soit $f_{i-1}(xy)$ les flots obtenus à l'étape précédente de valeur globale v_{i-1} et de coût D_{i-1}.

- On construit le graphe d'écart G_E^i correspondant à ce flot :
    - Si $0 < f_{i-1}(xy) < c(xy)$

      $(xy) \in G_E^i$ avec $\begin{cases} c_i(xy) = c(xy) - f_{i-1}(xy) \\ d_i(xy) = d(xy) \end{cases}$ et $(yx) \in G_E^i$ avec $\begin{cases} c_i(yx) = f_{i-1}(xy) \\ d_i(yx) = -d(xy) \end{cases}$
    - Si $0 < f_{i-1}(xy) = c(xy)$

      $(xy) \notin G_E^i$ et $(yx) \in G_E^i$ avec $\begin{cases} c_i(yx) = c(xy) \\ d_i(yx) = -d(xy) \end{cases}$
    - Si $f_{i-1}(xy) = 0$

      $(xy) \in G_E^i$ avec $\begin{cases} c_i(xy) = c(xy) \\ d_i(xy) = d(xy) \end{cases}$ et $(yx) \notin G_E^i$
- On cherche sur G_E^i le chemin de coût minimal, allant de s à p.
- Si un tel chemin existe, soit m_i la capacité minimale des arcs constituant ce chemin et d_i le coût de passage de m_i unités de matière le long de ce chemin.
    - On obtient alors un nouveau flot de valeur $v_i = v_{i-1} + m_i$ et de coût minimal $D_i = D_{i-1} + d_i$.
    - Passer à l'étape $i + 1$. (on construit directement G_E^{i+1} à partir de G_E^i)
- Si ce chemin n'existe pas, on est à l'optimum ($f^*(x, y) = c_i(y, x)$).

Cet algorithme permet de fournir tous les flots de coût minimum de valeur $v \le v_{\max imum}$.