

3.36pt

AGP: Algorithmique et programmation

tanguy.risset@insa-lyon.fr

Lab CITI, INSA de Lyon

Version du March 22, 2020

Tanguy Risset

March 22, 2020

Table of Contents

3.36pt

1 Pointeur de fonctions

2 Erreurs courante en C

3 Coder proprement

Présentation de la partie Graphe

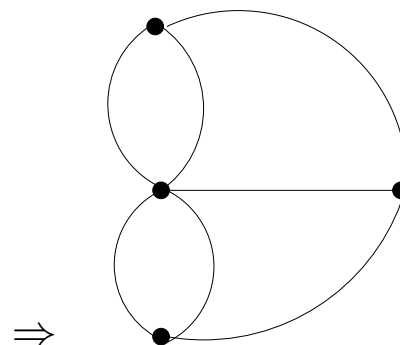
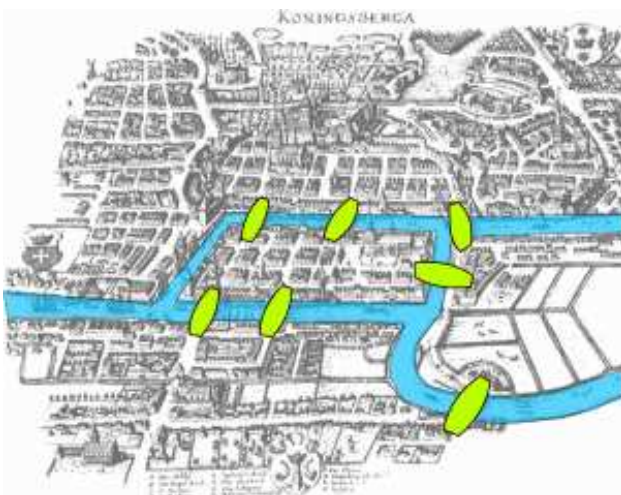
- Objectif:
 - Une bonne base des algorithmes sur les graphes
 - Notions réutilisées en réseaux (algorithmes de routage, optimisation de trafic, etc...)
- Déroulement du cours:
 - Graphes et Algorithmes sur les graphes (2 cours)
 - 2 TD d'algorithmique sur les graphes

Références utiles

- Calculateurs, Calcul et Calculabilité: O. Ridoux et G. Lesventes
Introduction très bien faite à ce domaine (utilisé pour ce cours)
- Introduction à l'algorithmique, T. Cormen, C. Leiserson, R. Rivest, 1994, Dunod
Approche complète et pédagogique de l'algorithmique
- Graphes et algorithmes (Gontran et Minoux) ou plutôt Graphes (C. Berge) plus complet et sans bug.
- Introduction à la théorie des graphes Sigward (université de Nancy-Metz) disponible sur Moodle.

Historique

- Euler résoud le problème des ponts de Königsberg au XVIII^{ème} siècle.



Graphe

Definition

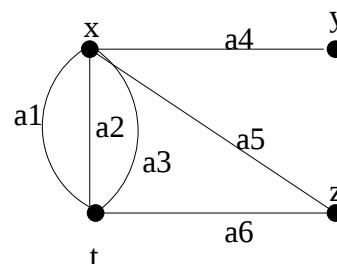
Graphe simple Un graphe simple $G = (X, E)$ est un couple formé de deux ensembles : un ensemble $X = \{x_1, x_2, \dots, x_n\}$ dont les éléments sont appelés *sommets*, et un ensemble $E = \{a_1, a_2, \dots, a_m\}$, partie de l'ensemble $P_2(X)$ des parties à deux éléments de X , dont les éléments sont appelés *arêtes*.

- Soit $a = \{x, y\} \in E$
 - a est l'arête de G d'extrémités x et y .
 - Parce que a existe, x et y sont des sommets *adjacents*.
 - On note $G = (X, E)$ ou $G(X, E)$

Multigraphe non orienté

- Un multigraphe $G = (X, A, f)$: est déterminé par:
 - X ensemble de sommets
 - A ensembles des arêtes
 - f un fonction de A dans $P_2(X)$
- Exemple:

- $X = \{x, y, z, t\}$
- $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$
- $f(a_1) = f(a_2) = f(a_3) = \{x, t\}$



Multigraphe avec boucle

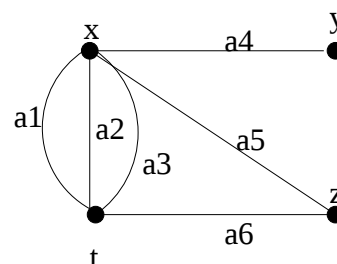
- La fonction f est de A dans $P_1(X) \times P_1(X)$
- $f(a) = \{x, x\}$ est possible

degré d'un sommet

Definition

degré d'un nœud Soit $G = (X, A)$ un graphe simple, et x un sommet de ce graphe. le *degré* de x est le nombre d'arêtes incidentes à x (c'est à dire contenant x).

- On étend cette définition aux multigraphes.
- Pour une boucle $\{x, x\}$ on compte deux!
- $d(x) = 0$ sommet isolé
- $d(x) = 1$ sommet pendant
- exemple: $d(x) = 5$, $d(y) = 1$,
 $d(z) = 3$, $d(t) = 4$



Definition

Lemme de la poignée de main

Lemme

Lemme de la poignée de main Soit $G = (X, A)$ un graphe simple, on a toujours:

$$\sum_{x \in X} d(x) = 2|A|$$



Graphe orienté

Definition

Graphe orienté un graphe orienté $G = (X, A)$ est la donnée de deux ensembles: un ensemble $X = \{x_1, \dots, x_n\}$ dont les éléments sont les *sommets*, un ensemble $A = \{a_1, \dots, a_m\}$ partie du produit cartésien $X \times X$ dont les éléments sont des arcs.



- On parle d'*arêtes* pour les graphes non orientés (a est la *paire* $\{x, y\}$) et d'*arc* pour les graphes orientés (a est le *couple* (x, y))
- si $a = (x, y)$
 - a est l'arc de G d'*extrémité initiale* x et d'*extrémité finale* y
 - On parle d'*incidence* d'un sommets à un arc

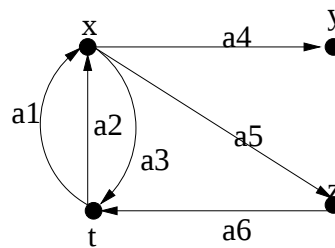
Degrés dans un graphe orienté

- $d(x)$ est toujours le nombre d'arcs incidents au sommet x
- On définit également le *degré entrant/sortant* pour le nombre d'arc dont x est l'extrémité initiale/extrémité finale. Notation : $\Gamma^+(x)$ et $\Gamma^-(x)$
- On a donc $d(x) = \Gamma^+(x) + \Gamma^-(x)$.
- et

$$\sum_{x \in X} \Gamma^+(x) = \sum_{x \in X} \Gamma^-(x) = |A|$$

- Exemple:

- $\Gamma^+(x) = 2, \Gamma^-(x) = 3,$
- $\Gamma^+(t) = 2, \Gamma^-(t) = 2,$



Sous-Graphe et Graphe Partiel

Definition

Sous-graphe $H = (Y, B)$ est un sous-graphe de $G = (X, A)$ si $Y \subset X$ et $B \subset A$

-

Definition

Graphe partiel $H = (Y, B)$ est un graphe partiel de $G = (X, A)$ si $Y = X$ et $B \subset A$

-

Chaînes dans les graphes non orientés

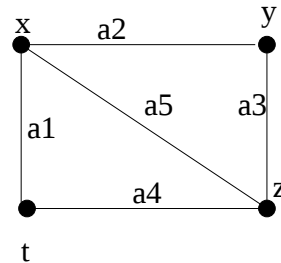
- *Chaîne* : suite finie de sommets reliés entre eux par une arête.
- Chaîne *simple* : chaîne qui n'utilise pas deux fois la même arête.
- Chaîne *eulérienne* : chaîne simple passant par toutes les arêtes d'un graphe.
- Chaîne *hamiltonienne* : chaîne simple passant par tous les sommets d'un graphe une et une seule fois.
- Notions définies pour un graphe non orienté, étendues facilement aux graphes orientés.

Définitions: Cycles/chemin

- *Cycle* : chaîne qui revient à son point de départ.
- Cycle *eulérien* : cycle simple passant par toutes les arêtes d'un graphe une et une seule fois.
- Cycle *hamiltonien* : cycle simple passant par tous les sommets d'un graphe une et une seule fois.
- Graphes orientés:
 - *Chemin* : suite de sommets reliés par des arcs dans un graphe orienté.
 - *Circuit* : Chemin qui revient à son point de départ (i.e. cycle pour les graphes orientés).

Exemple

- $\{a_1, a_2, a_3\}$: chaîne, chaîne simple, chaîne hamiltonienne.
- $\{a_1, a_2, a_3, a_4\}$: cycle.
- Existe-t'il un cycle eulerien?



Graphe connexe

Definition

Graphe connexe Un graphe G est dit connexe si pour toute paire de sommets $\{x, y\}$ de G , il existe une chaîne de premier terme x et de dernier terme y .

•

Arbres

Un arbre est un graphe $G(V, E)$ peut être défini par une des propriétés suivantes:

- C'est graphe $G(V, E)$ connexe et sans cycle
- $|V| = |E| + 1$ et il est connexe
- $|V| = |E| + 1$ et il est sans cycle
- Il existe une chaîne unique entre toute paire de noeuds
- Il est connexe et minimal pour cette propriété
- Il est sans cycle et maximal pour cette propriété

Graphes spéciaux

- *Graphe eulérien* : graphe qui possède un cycle eulérien.
- *Graphe semi-eulérien* : graphe qui possède une chaîne eulérienne.
- *Graphe hamiltonien* : graphe qui possède un cycle hamiltonien.
- *Graphe semi-hamiltonien* : graphe qui possède une chaîne hamiltonienne.

Graphes valués

Definition

Graphe valué Un graphe valué est un graphe où des valeurs sont associés aux arêtes ou aux sommets (on dit également graphe pondéré) on note $G = (V, E, P_V, P_E)$, P_E étant la pondération des arêtes et P_V la pondération des sommets.

-
- Exemples de pondération possible:
 - La distance géographique
 - Le délai de transit
 - Le temps d'attente
 - Le coût de la communication
 - Le débit

Distance

- *Longueur* d'une chaîne : nombre des arêtes qui composent la chaîne.
- *Valeur* d'une chaîne : somme des valeurs des arêtes (arcs) d'une chaîne d'un graphe valué.
- *Distance* entre deux sommets : longueur de la plus courte chaîne joignant ces deux sommets.
- *Diamètre* d'un graphe : maximum des distances entre les sommets d'un graphe.

Nombre/Indice chromatique

- *Indice chromatique* d'un graphe : nombre minimal de couleurs permettant de colorier les arêtes d'un graphe, de telle sorte que deux arêtes adjacentes n'aient pas la même couleur.
- *Nombre chromatique* d'un graphe: nombre minimal de couleurs permettant de colorier les sommets d'un graphe, de telle sorte que deux sommets adjacents n'aient pas la même couleur.

Théorème d'Euler

Theorem

Un graphe simple connexe G est eulérien si et seulement si pour tout sommet x de G , $d(x)$ est pair.

-
- Démonstration....

Représentation machine

- Représentations statique:
 - Matrice d'incidence: une ligne par sommet, une colonne par arête
 - Matrice d'adjacence: une ligne par sommet, une colonne par sommet (matrice symétrique pour les graphes non orientés)
 - Attention, en général $|X|^2 \gg |V|$
- Représentations dynamique:
 - Structure auto-référencée (généralisation des structures d'arbres), assez difficile à manipuler.
 - Liste de sommets suivants

Table of Contents

3.36pt

1 Pointeur de fonctions

2 Erreurs courante en C

3 Coder proprement

Graphe: partie 2

Algorithmes sur les graphes

Les bases des protocoles de communications

- Fermeture transitive
- Transversal (parcours)
- Arbre de recouvrement
- Plus court chemin

Fermeture transitive

- Rappel: la fermeture transitive R^* d'une relation R est telle que

$$xR^*y \Leftrightarrow \exists z_1, \dots, z_n / xRz_1, z_1Rz_2, \dots, z_nRy$$

- La relation considérée est "l'adjacence" dans un graphe G
- Le calcul de la fermeture transitive permet de répondre aux questions concernant l'existence de chemins entre x et y dans un graphe G et ceci pour tout couple de sommets (x, y) .
- La fermeture transitive de $G(V, E)$, noté $G^*(V, E^*)$ est le plus petit graphe tel que $(x, y) \in E^*$ si et seulement s'il existe un chemin de x à y dans G

Algorithme fermeture transitive

- On part de la matrice d'adjacence de G (M) et on produit la matrice d'incidence de G^* : M'
- Répéter la procédure `iterationFT` jusqu'à stabilisation.
- `iterationFT(MATRICE M, MATRICE M')`

DEBUT

POUR i VARIANT de 1 à N

POUR j VARIANT de 1 à N

SI $M[i,j]=1$ ALORS

POUR k VARIANT de 1 à N

SI $M[j,k]=1$ ALORS $M'[i,k]=1$

FINPOUR

FINSI

FINPOUR

FINPOUR

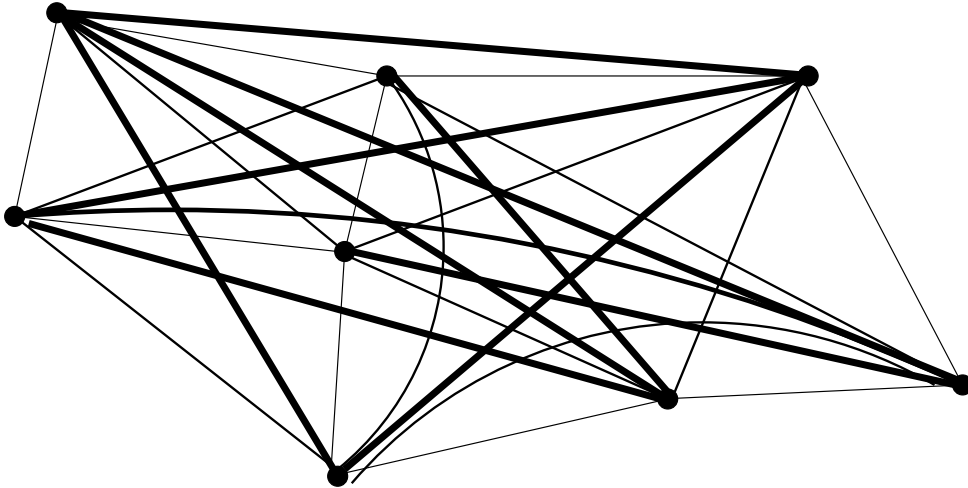
FIN

The graph consists of 8 vertices and 10 edges. It can be described as a central square with two triangles attached to its left and right sides. The vertices are labeled as follows: the top-left vertex of the square is v_1 , the top-right is v_2 , the bottom-right is v_3 , and the bottom-left is v_4 . The triangle on the left has vertices v_1 , v_4 , and v_5 (top-left). The triangle on the right has vertices v_2 , v_3 , and v_6 (bottom-right). The edges are (v_1, v_2) , (v_2, v_3) , (v_3, v_4) , (v_4, v_1) , (v_1, v_5) , (v_4, v_5) , (v_2, v_6) , (v_3, v_6) , (v_5, v_7) , and (v_6, v_8) , where v_7 and v_8 are the bottom-most vertices of the left and right triangles respectively.

A graph with 8 vertices and 15 edges. The vertices are arranged in a non-regular grid. The edges include both straight and curved lines, forming a complex network structure. The graph is connected and contains several cycles.

Exemple d'application

- Itération 2 (état stable)



Propriété de l'algorithme

- Au bout d'une itération, les sommets à distance 2 sont connectés.
- Au bout de deux itérations, les sommets à distance 4 sont connectés.
-
- $\Rightarrow$ L'algorithme effectue au plus $\log(N)$ itérations (N étant le nombre de sommets).

Algorithme de parcours

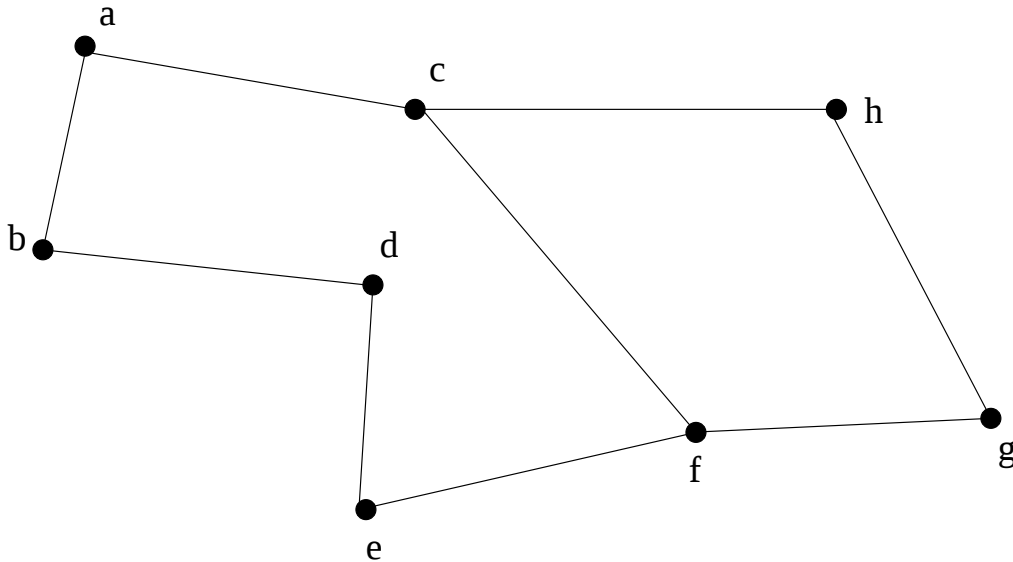
- On connaît déjà les parcours d'arbre (Profondeur uniquement)
- Comment parcourir un graphe ?
- Pareil... sauf qu'on peut s'autoriser à passer plusieurs fois par la même arrête ou le même noeud si on ne le *traite* qu'une seule fois !
- On va voir le parcours en profondeur d'abord et en largeur d'abord. On visitera les noeuds plusieurs fois, mais on les traitera une seule fois.
- un parcours permet de définir un *arbre de recouvrement*

Profondeur d'abord (récursif)

- Profondeur(Graphe G , noeud x)
 - Marquer(x)
 - Afficher(x)
 - POUR TOUT y du voisinage de x FAIRE
 - Si y non marqué ALORS Profondeur(G, y)
 - FINPOUR

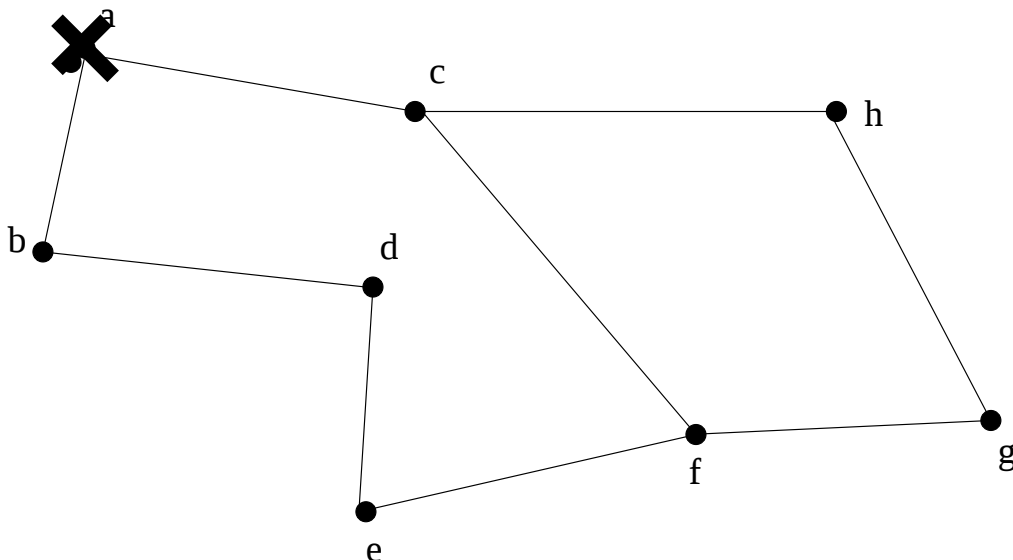
Exemple d'application

- Graphe initial
- Noeuds traités: $\emptyset$



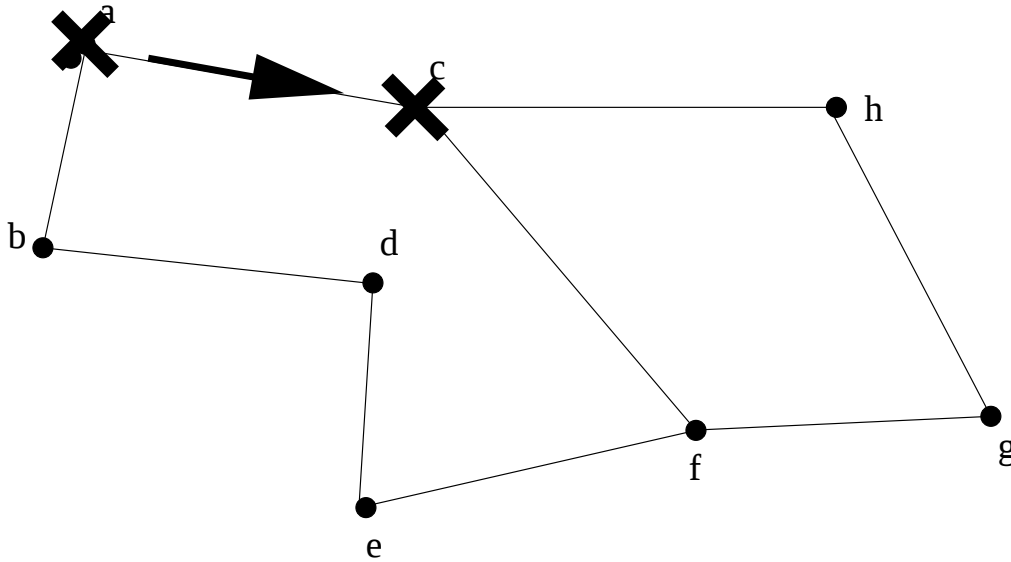
Exemple d'application

- Appel $Prof(G, a)$
- Noeuds traités: a



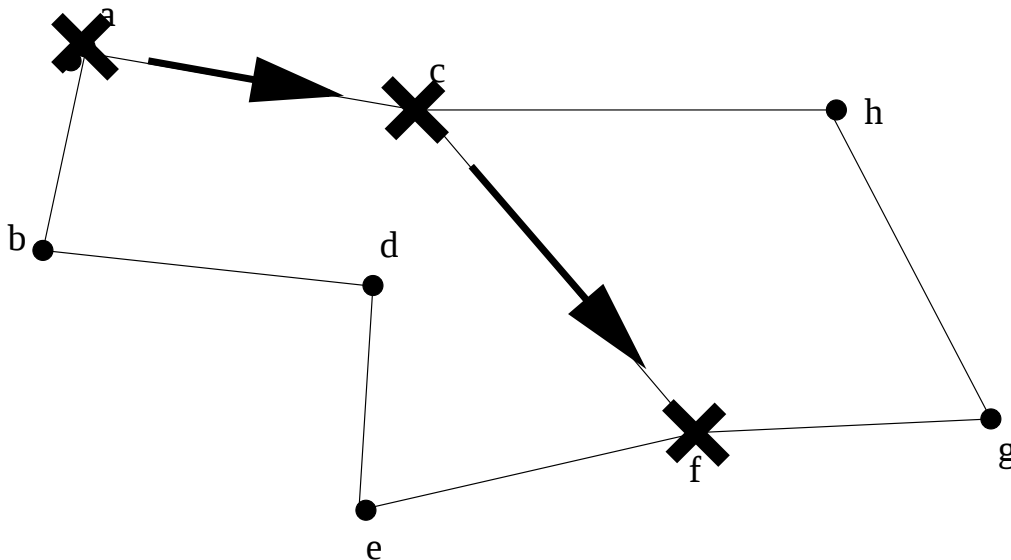
Exemple d'application

- Lance appel $Prof(G, c)$
- Noeuds traités: a, c



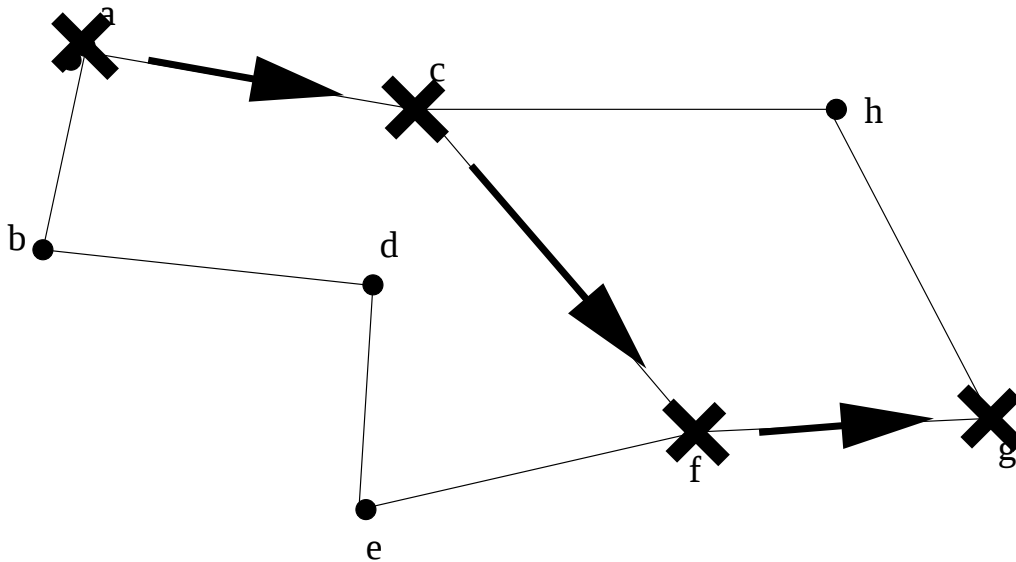
Exemple d'application

- Lance Appel $Prof(G, f)$
- Noeuds traités: a, c, f



Exemple d'application

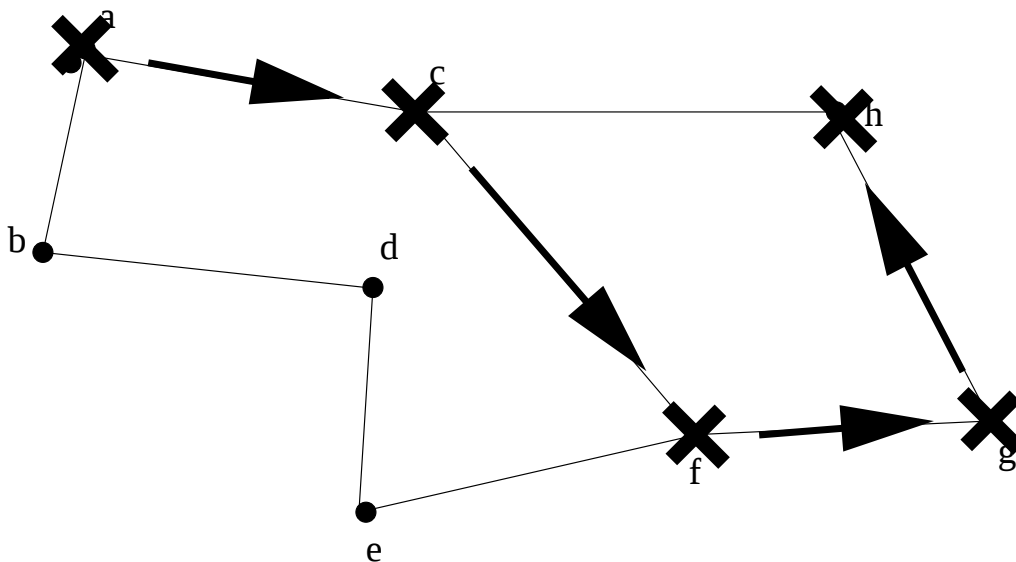
- Lance Appel $Prof(G, g)$
- Noeuds traités: a, c, f, g



Navigation icons: back, forward, search, etc.

Exemple d'application

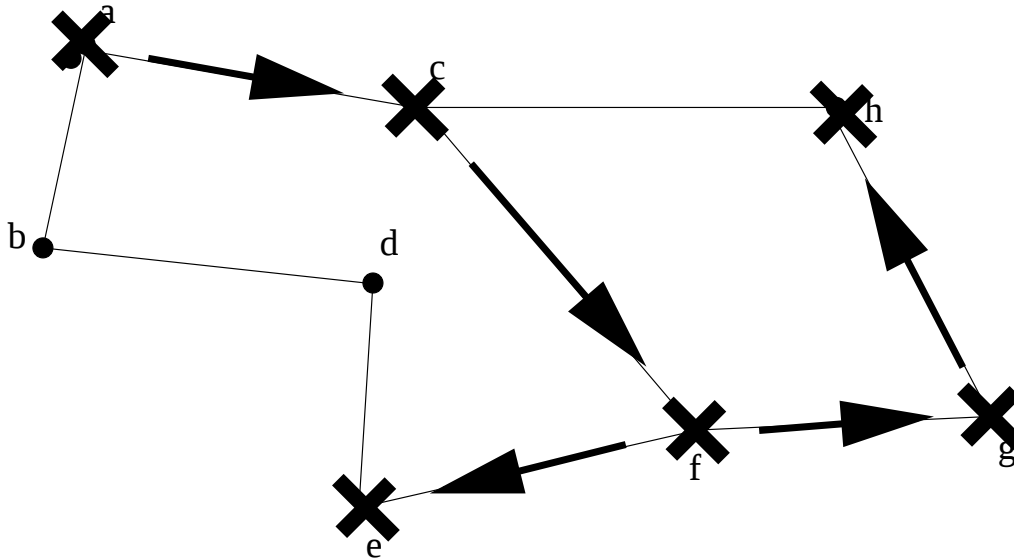
- Lance Appel $Prof(G, h)$
- Noeuds traités: a, c, f, g, h



Navigation icons: back, forward, search, etc.

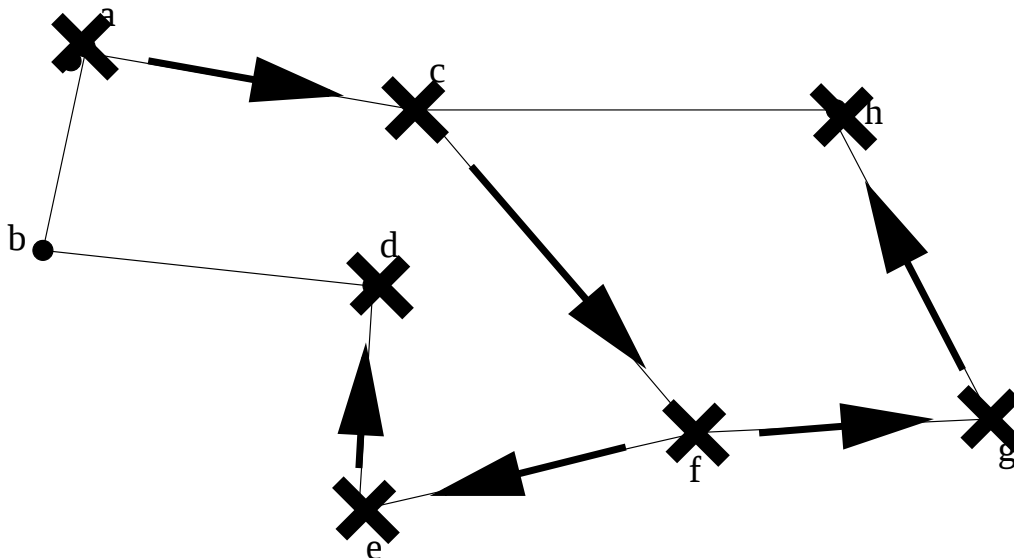
Exemple d'application

- Retour h, g , jusqu'à appel $Prof(G, f)$
- Lance appel $Prof(G, e)$
- Noeuds traités: a, c, f, g, h, e



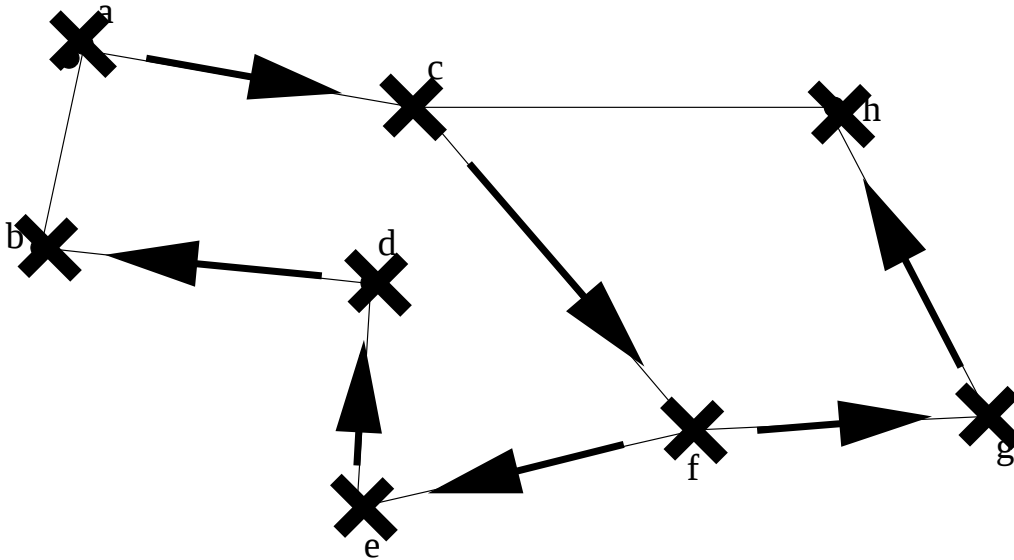
Exemple d'application

- lance Appel $Prof(G, d)$
- Noeuds traités: a, c, f, g, h, e, d



Exemple d'application

- Appel $Prof(G, d)$, fin de l'algorithme
- Noeuds traités: a, c, f, g, h, e, d, b



Parcours en Largeur d'abord

- Plus complexe
- Parcours des sommets "comme des cercles concentriques" autour du point de départ
- Nécessite une file d'attente

Algorithmme du parcours en largeur (itératif)

```

LARGEUR(Graphe G, noeud x)
  Marquer(x)
  Traiter(x)
  AjouterFile(x)
  TANT QUE FileNonVide() FAIRE
    y=RetirerFile()
    POUR TOUT z voisin de y FAIRE
      SI NonMarqué(z) ALORS
        Marquer(z)
        Traiter(z)
        AjouterFile(z)
      FINSI
    FINPOUR
  FIN TANTQUE
  
```

Navigation icons: back, forward, search, etc.

Tanguy Risset

AGP: Algorithmique et programmation

46

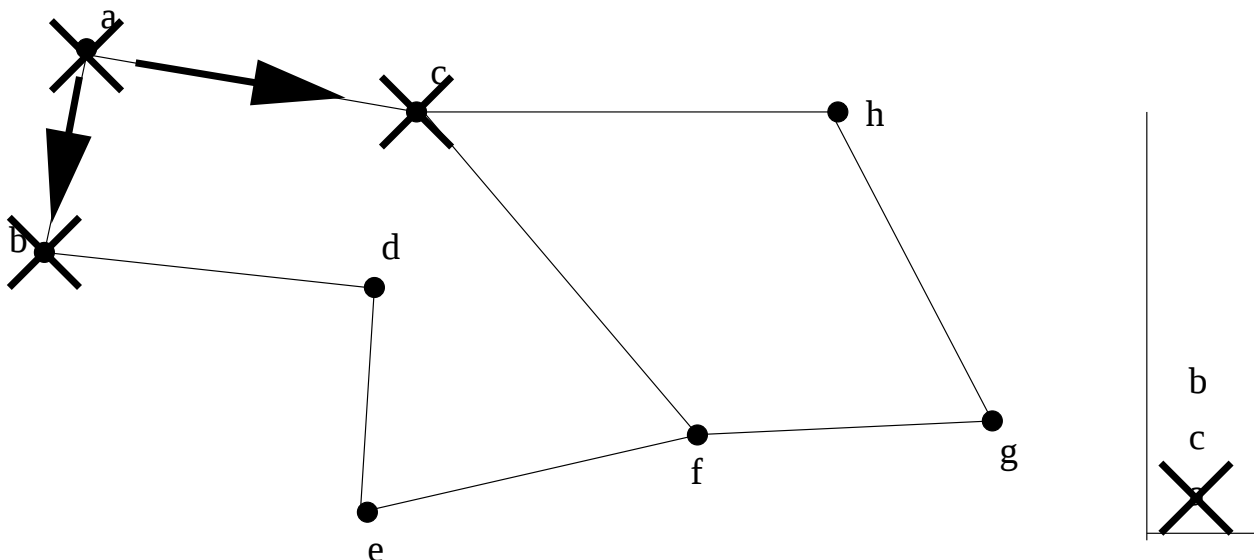
Pointeur de fonctions
oooooooooooo

Erreurs courante en C
oooooooo

Coder proprement
oooooooooooo

Exemple d'application

- Itération 1, ($y = a$)
- Noeuds traités: a, c, b



Navigation icons: back, forward, search, etc.

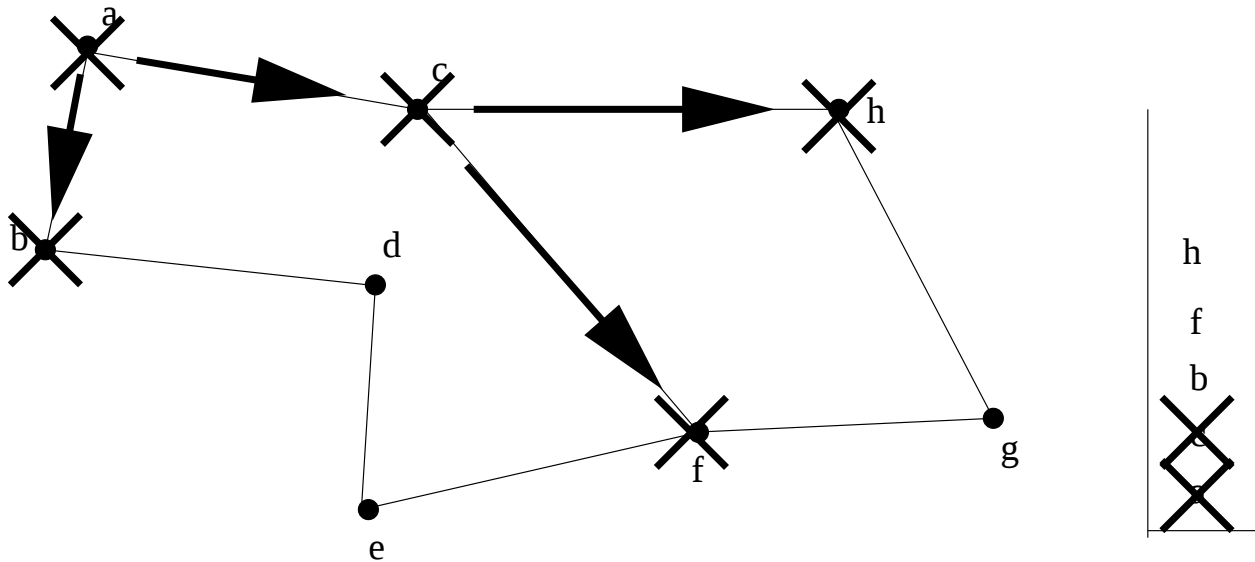
Tanguy Risset

AGP: Algorithmique et programmation

47

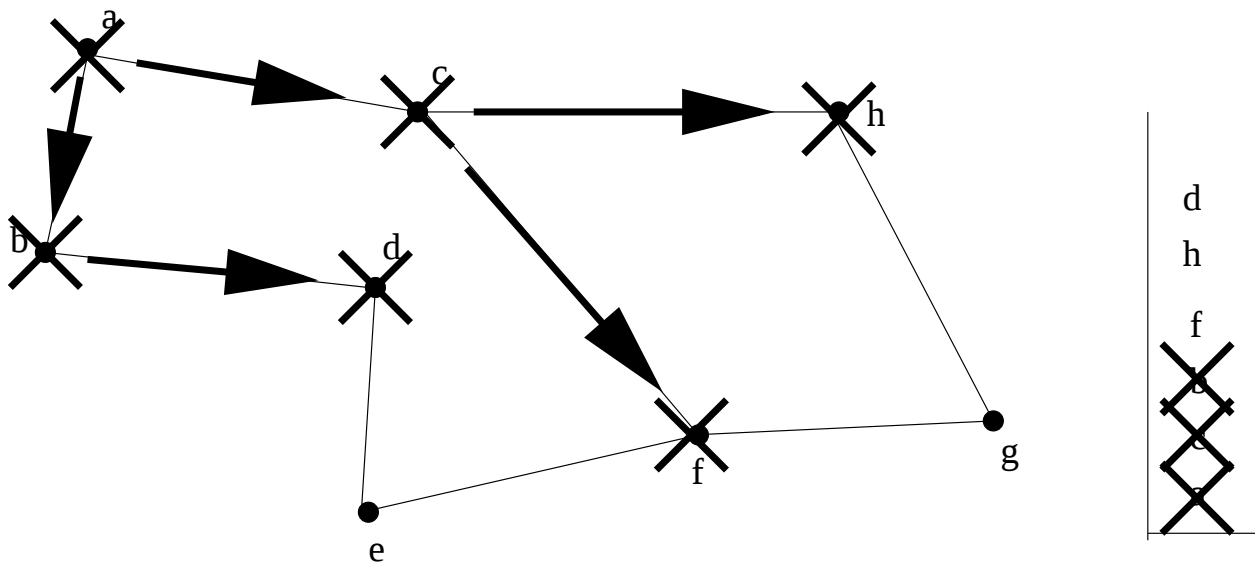
Exemple d'application

- Itération 2, ($y = c$)
- Noeuds traités: a, c, b, f, h



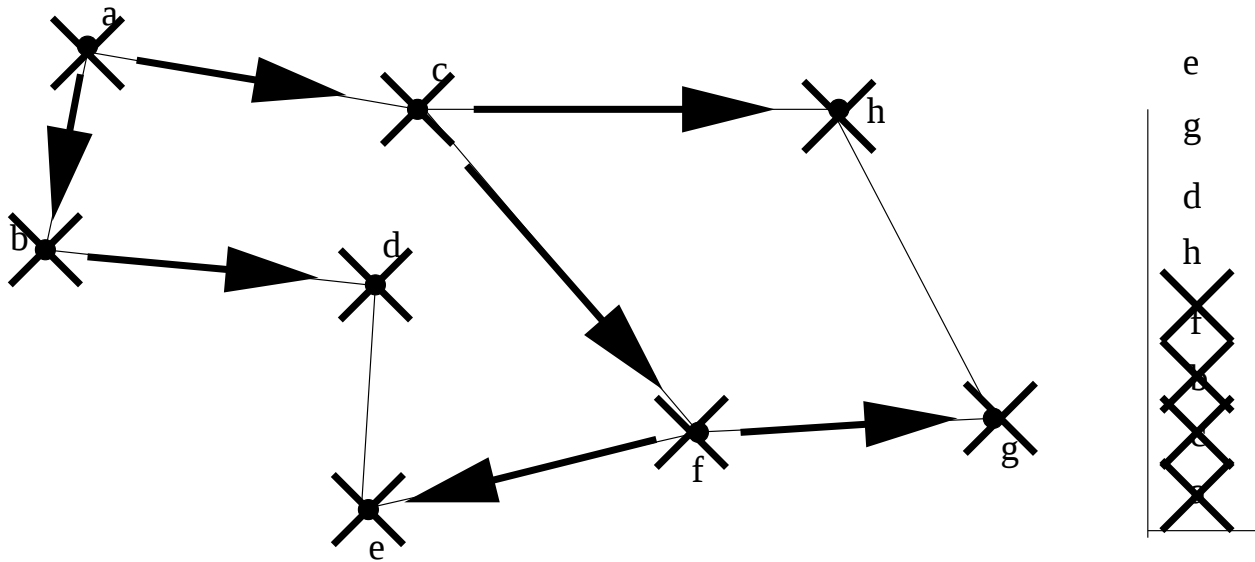
Exemple d'application

- Itération 3, ($y = b$)
- Noeuds traités: a, c, b, f, h, d



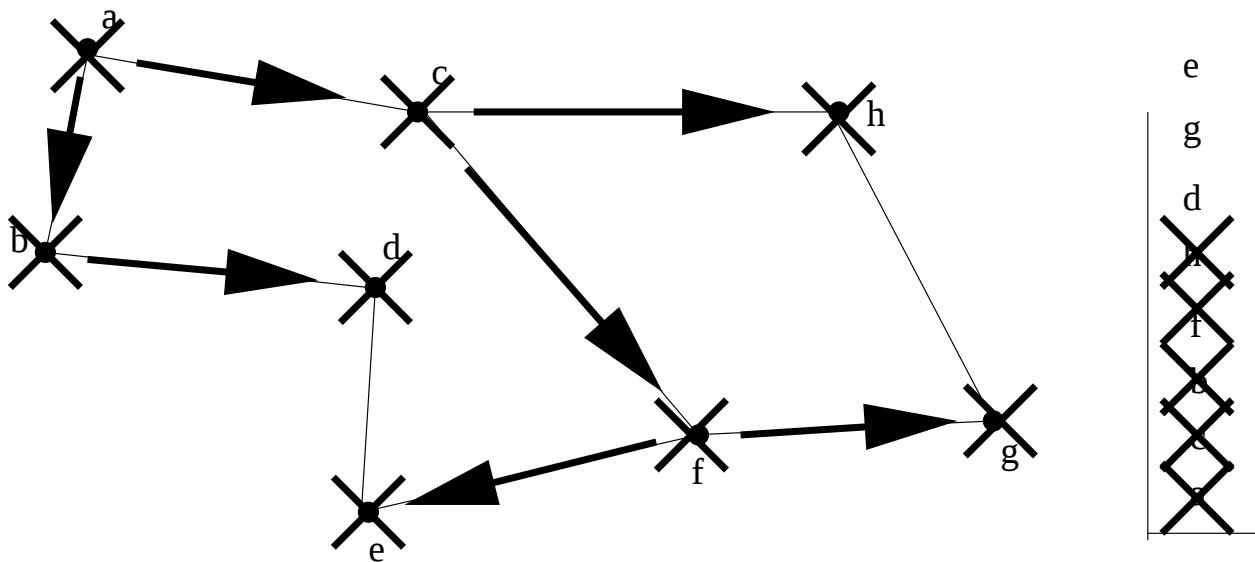
Exemple d'application

- Itération 4, ($y = f$)
- Noeuds traités: a, c, b, f, h, d, e, g



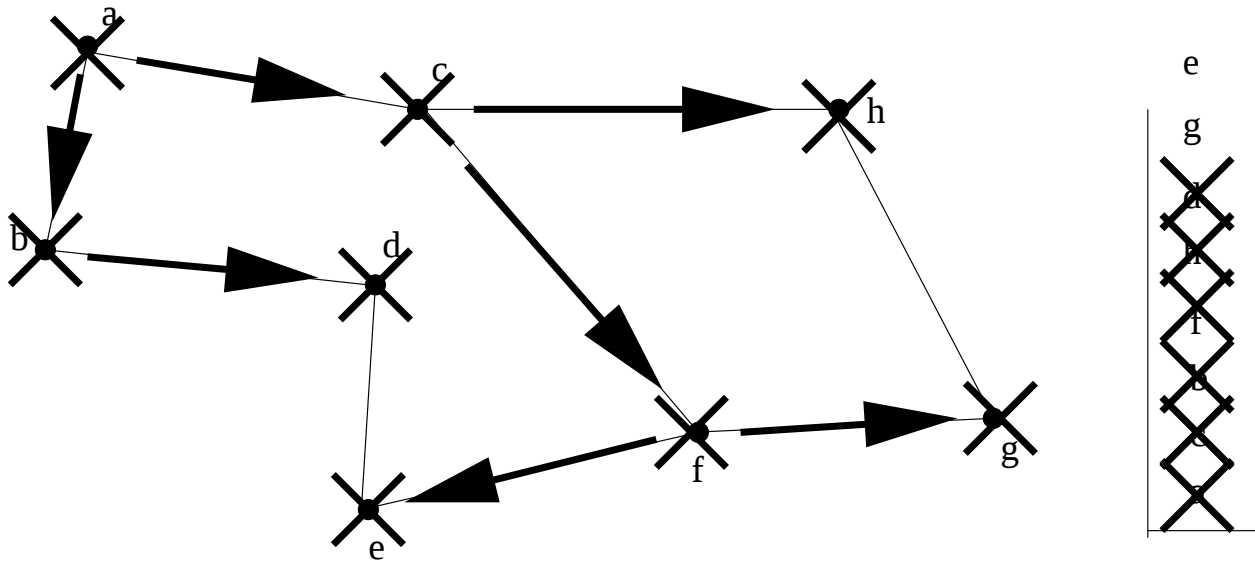
Exemple d'application

- Itération 5, ($y = h$)
- Noeuds traités: a, c, b, f, h, d, e, g



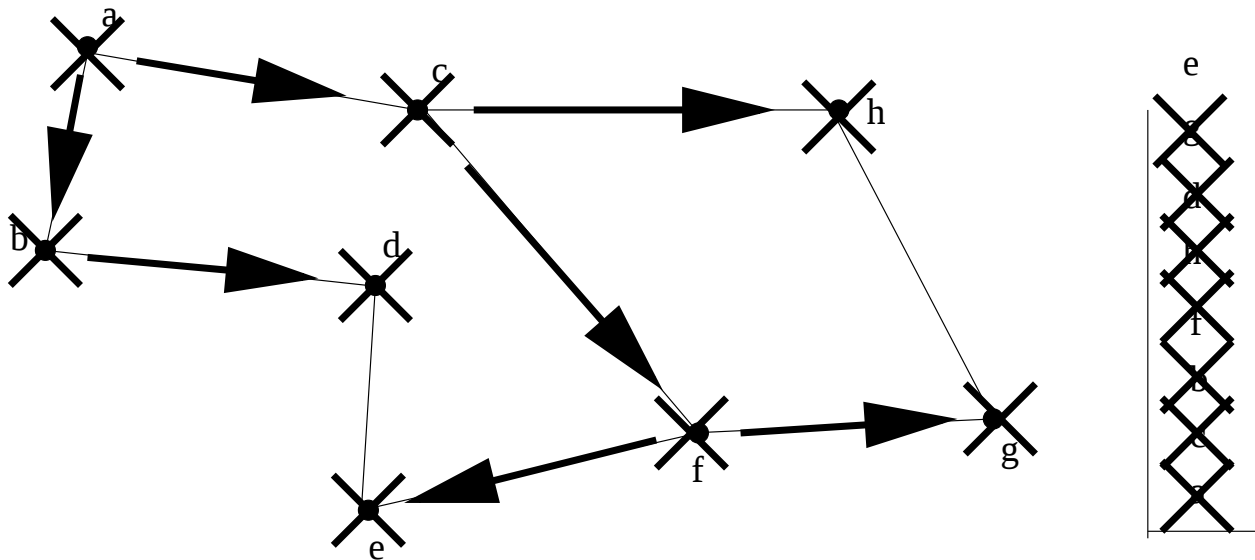
Exemple d'application

- Itération 6, ($y = d$)
- Noeuds traités: a, c, b, f, h, d, e, g



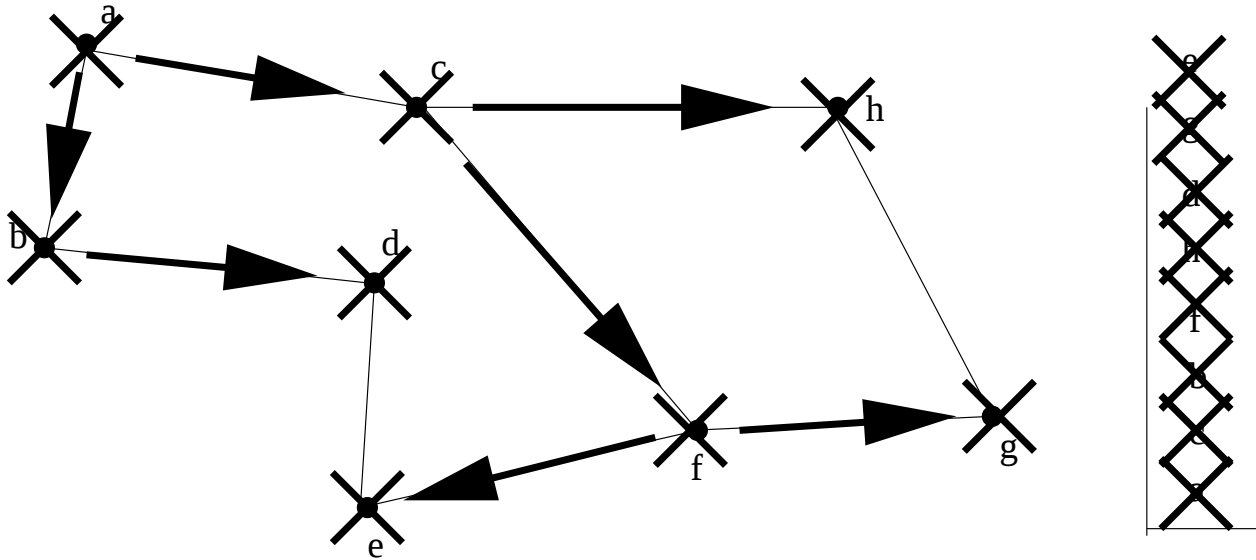
Exemple d'application

- Itération 7, ($y = g$)
- Noeuds traités: a, c, b, f, h, d, e, g



Exemple d'application

- Itération 8, ($y = e$)
- Noeuds traités: a, c, b, f, h, d, e, g



Arbre de poid minimum

- Soit $G = (V, E, P)$ un graphe pondéré sur les arêtes
- On désire trouver un sous ensemble T d'arêtes formant un arbre, tel que $P(T)$ soit minimum
- On va ordonner l'ensemble des arêtes E par poids croissant:
 $E = \{E_1, \dots, E_n\} / P(E_i) \leq P(E_{i+1})$

Algorithme

ARBRE_POID_MIN(Graphe G)

$T = \emptyset$

$i = 1$

TANT QUE $|T| \neq |V| - 1$ FAIRE

SI $T \cup \{E_i\}$ ne forme pas un cycle ALORS

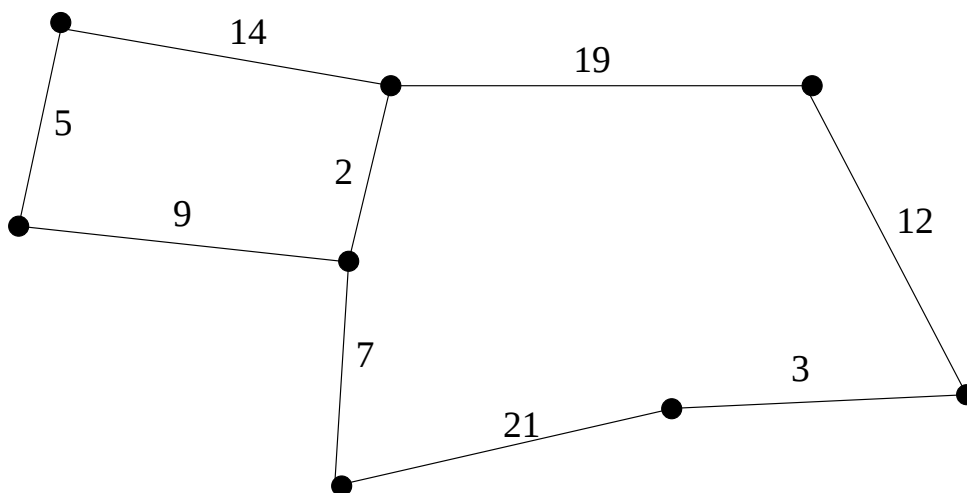
$T \leftarrow T \cup \{E_i\}$

FINSI

$i = i + 1$

FIN TANTQUE

Exemple



Chemin minimum

- Propriété: Si le plus court chemin $C_{x,y}$ de x à y passe par z ($C_{x,y} = C_{x,z} \cup C_{z,y}$) alors le plus court chemin entre z et y est $C_{z,y}$.
- Preuve : Si $\exists C'_{z,y} / P(C'_{z,y}) < P(C_{z,y})$ alors $P(C'_{x,y} = C_{x,z} \cup C'_{y,z}) < P(C_{x,y})$
- $\Rightarrow$ L'ensemble des plus court chemins entre un noeud s et l'ensemble des autres noeuds du graphe est une arborescence de racine s

- Algorithme du plus court chemin
 - Dijkstra (1959)
 - Ford Fulkerson (1962) ou Bellman (1957)
- Ces algorithmes sont aujourd'hui utilisés pour le routage dans les réseaux

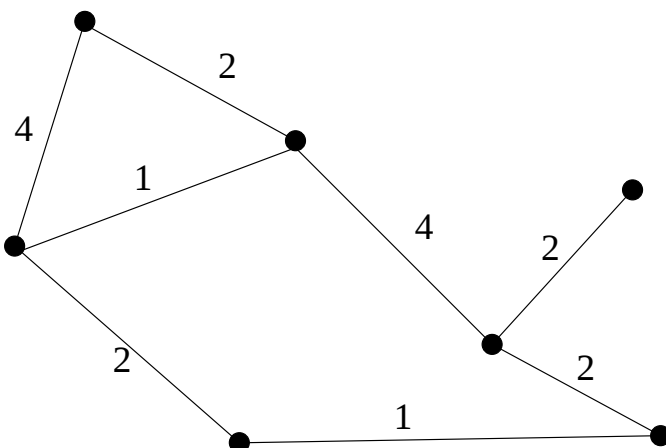
Algorithme de Bellman

- Pondération quelconque (positive ou négative)
- À partir d'un sommet source S
- Chaque noeud x est initialisé à une distance de $S \forall x \in V, x \neq s$
 $dist_0(x) = \infty$, et $dist_0(s) = 0$.
- À chaque étape i on réévalue $dist$:

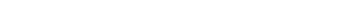
$$dist_i(x) = \text{Min}(dist_{i-1}(x), \text{Min}_{y \in \Gamma^-(x)}(dist_{i-1}(y) + P(xy)))$$

- À l'étape i on obtient le plus court chemin ayant au plus i arcs entre s et x
- On s'arrête au bout de $|V| - 1$ itérations : Pourquoi ?

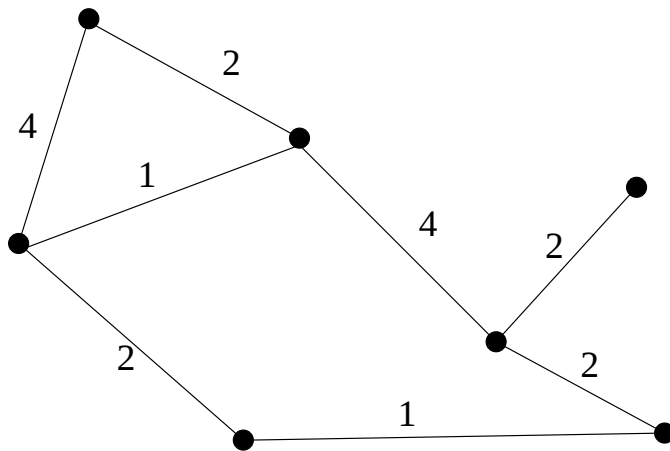
Exemple





- 

Exemple



Analyse des algorithmes

- Bellman $O(|V|^3)$
 - Nombre d'étapes $O(|V|)$ car le diametre peut avoir une longueur de $O(|V|)$
 - Coût d'une étape $O(|E|)$ car il faut évaluer, à chaque étape, toutes les arêtes
- Dijkstra $O(|V|^3)$
 - Nombre d'étapes $O(|V|)$ car à chaque étape on marque un sommet
 - Coût d'une étape $O(|E|)$ Car il faut évaluer toutes les arêtes
- Avec des structures de données adaptées
 - Bellman $O(|V|^3)$
 - Dijkstra $O(|V|^2 \log |V|)$

Table of Contents

3.36pt

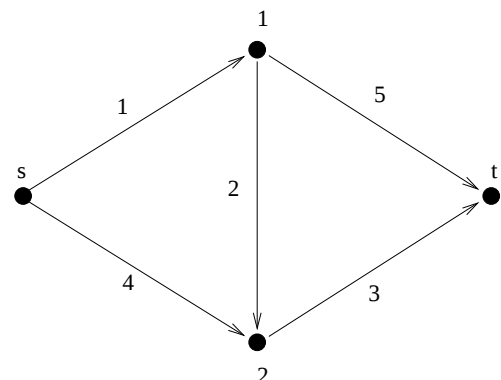
1 Pointeur de fonctions

2 Erreurs courante en C

3 Coder proprement

Problème du flot maximum

- Problème posé:
 - On dispose d'un graphe $G(V, A)$ orienté
 - Deux sommets particuliers sont identifiés: s appelé *source* et t appelé *puits*.
 - Les arcs de G sont étiquetés par une valeur entière positive appelée *capacité*, $c : A \mapsto \mathbb{N}^+$
- Trouver une fonction $f : A \mapsto \mathbb{N}^+$ qui soit un *flot admissible conservatif maximal*
 - admissible: respecte la capacité des arcs.
 - Conservatif: pour tous les nœuds, sauf la source le puits, le flot entrant est égal au flot sortant.
 - Maximal: qui maximise la capacité sortant de s .



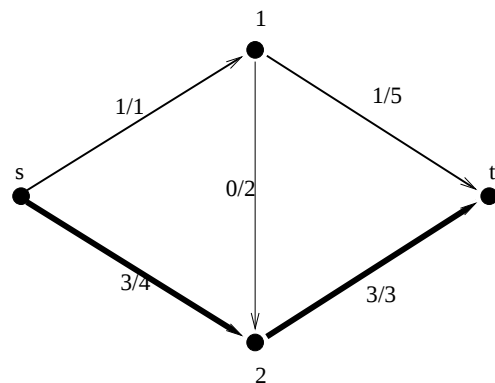
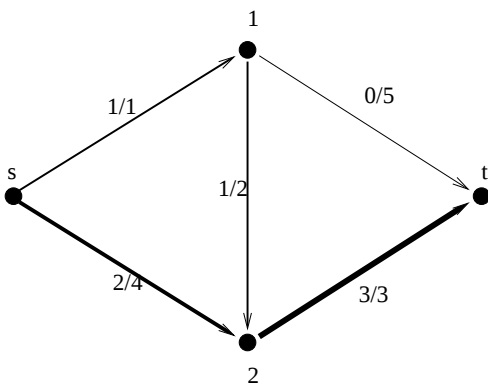
Application

Ce problème trouve des applications dans de nombreux domaines:

- Réseau de tuyau d'évacuation (égout, pipeline)
- Trafic internet
- Circulation automobile

Un exemple simple

- Deux flots



- Pourquoi sont-ils “admissibles”, “conservatifs”?
- Quel est le meilleur?
- Comment passer de l’un à l’autre?

Définitions formelles

- Un flot f sur un graphe $G(V, A, s, t, c)$ est **admissible** si

$$\forall a \in A, 0 \leq f(a) \leq c(a)$$

Le flot nul est toujours admissible.

- Le flot f est **conservatif** si pour tout les sommets (hormis la source et le puits), le flot entrant est égal au flot sortant:

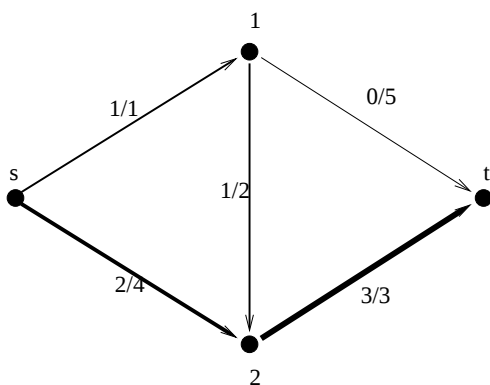
$$\forall x \in V, x \neq s, x \neq t, \sum_{a^- \in \Gamma^-(x)} (f(x, a^-)) - \sum_{a^+ \in \Gamma^+(x)} (f(a^+, x)) = 0$$

- $\Leftrightarrow$ il n'y a pas de fuite dans le réseau.
- La **valeur** du flot $v(f)$ est définie par le flot sortant de la source ou entrant dans le puits:

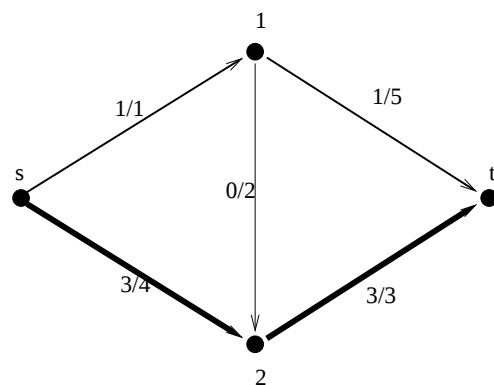
$$v(f) = \sum_{a^- \in \Gamma^-(s)} (f(s, a^-)) = \sum_{a^+ \in \Gamma^+(t)} (f(a^+, t))$$

Navigation icons: back, forward, search, etc.

Vérification



$$v(f) = 3$$



$$v(f) = 4$$

- Les deux flots sont admissibles.
- Les deux flots sont conservatifs.
- Existe-t'il un meilleur flot que celui de droite?

Navigation icons: back, forward, search, etc.

Algorithme de Ford-Fulkerson

- Trouver un flot admissible (par exemple le flot nul)
- Essayer d'améliorer ce flot tant que c'est possible
- Pour améliorer ce flot, on trouve une *chaîne améliorante*: une chaîne entre s à t sur laquelle la quantité de flot peut être augmentée.
 - On marque progressivement les sommets en commençant par s
 - On marque les successeurs y des sommets x marqués si $f(x, y) < c(x, y)$ (i.e. on peut augmenter le flot de x vers y)
 - On marque les prédécesseurs y des sommets x marqués si $f(y, x) > 0$ (i.e. on peut augmenter le flot de x vers y)
 - La marque identifie par quels successeur (ou quel prédécesseur) la chaîne améliorante passe.

Algorithme de marquage

On marque l'entrée du réseau par *

TANT QUE non stable

POUR CHAQUE x marqué

POUR CHAQUE y successeur de x non marqué

SI $f(x, y) < c(x, y)$ ALORS marquer(y) par $(+x)$

FINPOUR

POUR CHAQUE y prédécesseur de x non marqué

SI $f(y, x) > 0$ ALORS marquer(y) par $(-x)$

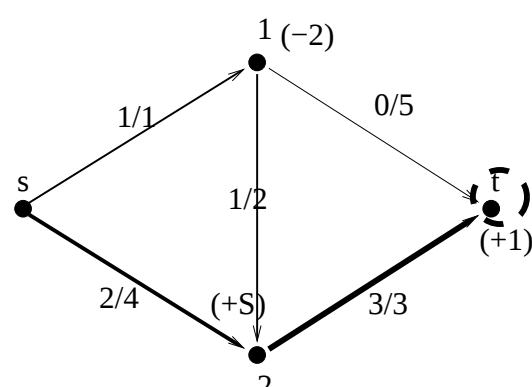
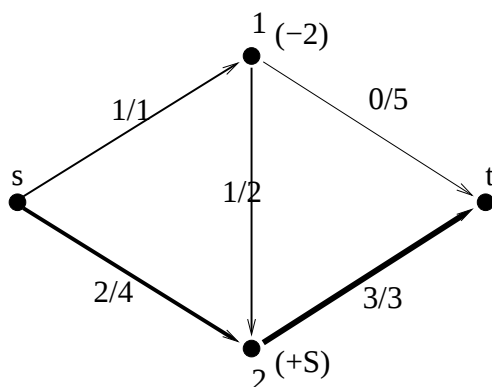
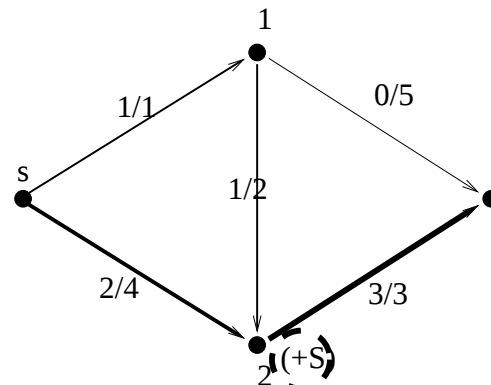
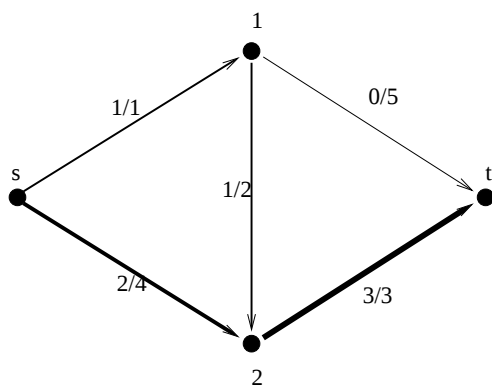
FINPOUR

FINPOUR

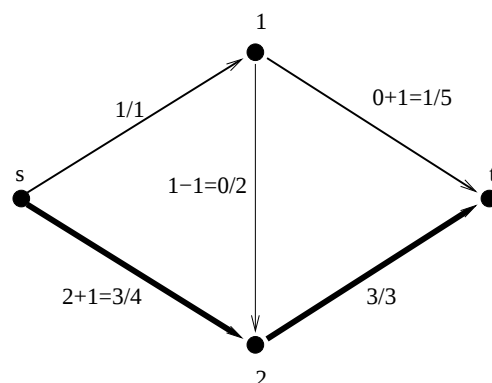
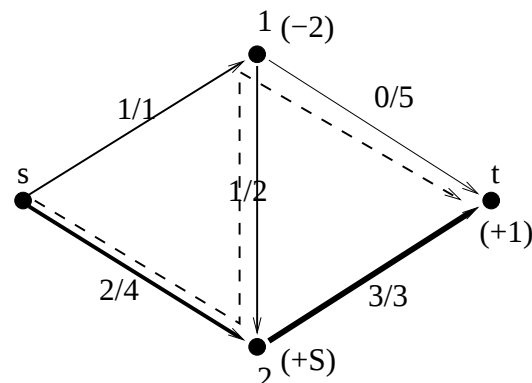
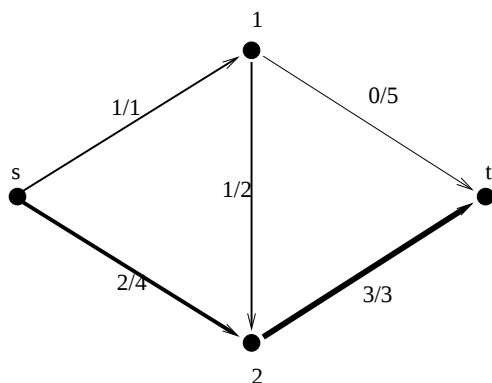
SI t est marqué ALORS

on a trouvé une chaîne améliorante

FIN TANTQUE



Navigation icons: back, forward, search, etc.



Navigation icons: back, forward, search, etc.

Algorithme Ford-Fulkerson(G, s, t, c)

```

marquer + le sommet source s
TANT QUE le flot n'est pas maximal FAIRE
  TANT QUE l'on marque des sommets FAIRE
    POUR CHAQUE sommet marqué  $x$  FAIRE
      POUR CHAQUE arc  $(x, y)$  FAIRE
        Si  $y$  n'est pas marqué et  $(x, y)$  non saturé ALORS
          marquer  $y$  par  $(+x)$ 
      FINPOUR
    POUR CHAQUE arc  $(y, x)$  FAIRE
      Si  $y$  n'est pas marqué et  $(y, x)$  à un flot non nul
        marquer  $y$  par  $(-x)$ 
    FINPOUR
  FINPOUR
  FINTANTQUE
  Si le puits  $t$  n'est pas marqué ALORS
    le flot est maximal (arrêt)

```

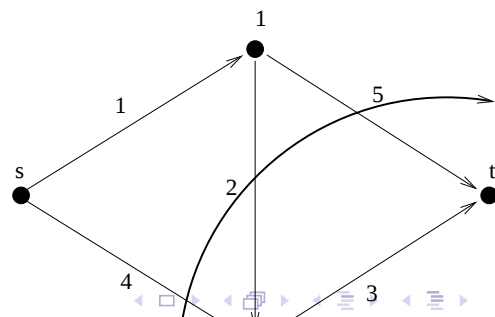
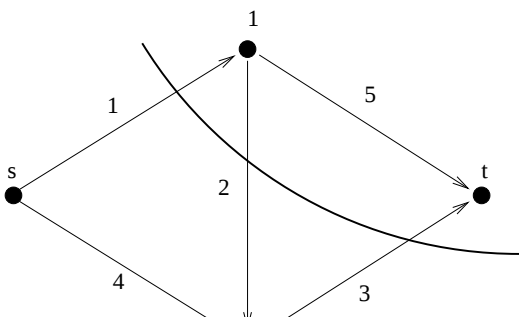
Tanguy Risset

AGP: Algorithmique et programmation

76

Max-flow, Min-cut

- On peut montrer que l'algorithme de Ford-Fulkerson est en $O(M * n^2)$ ou M est la capacité maximale d'une arête.
- L'algorithme utilise un résultat important: le théorème *max-flo*, *min-cut*: la valeur d'un flot maximum dans un graphe est égale à la valeur de la coupe minimale: on a ce qu'on appelle deux problèmes duaux.
- Une coupe est une partition des sommets en deux ensembles, l'un contenant la source, l'autre le puits.
- La valeur d'une coupe est le flot traversant la coupe en ne comptant que les arcs allant de s vers t .

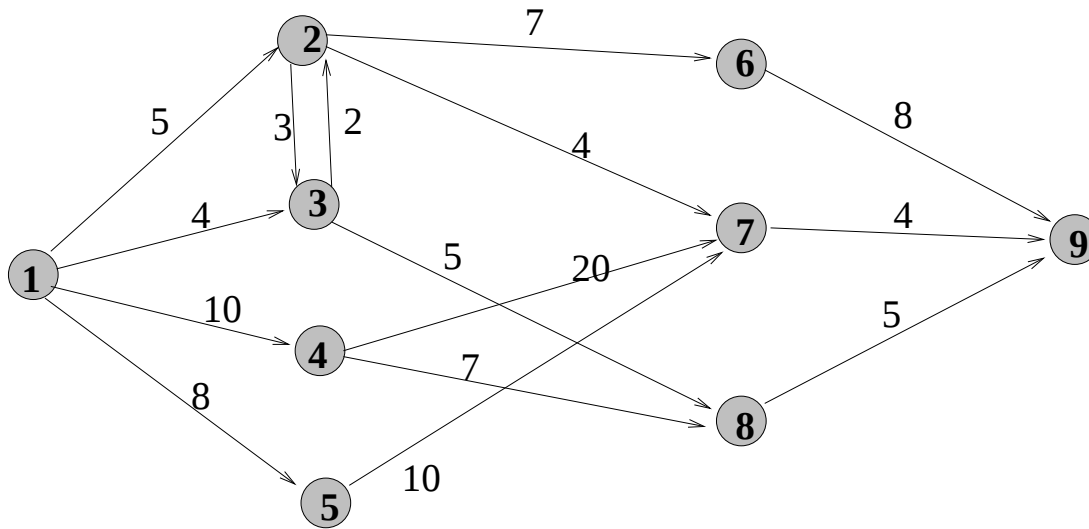


Tanguy Risset

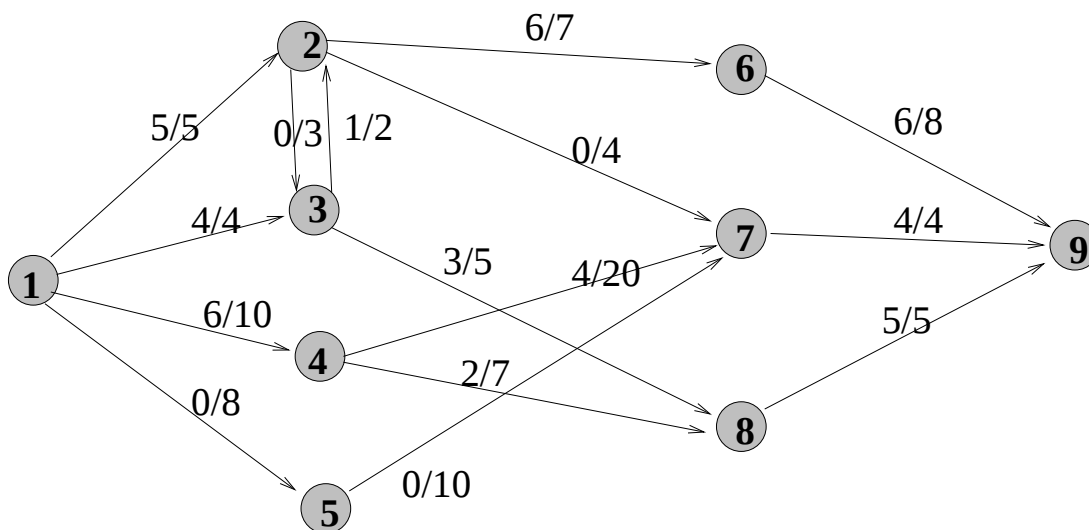
AGP: Algorithmique et programmation

77

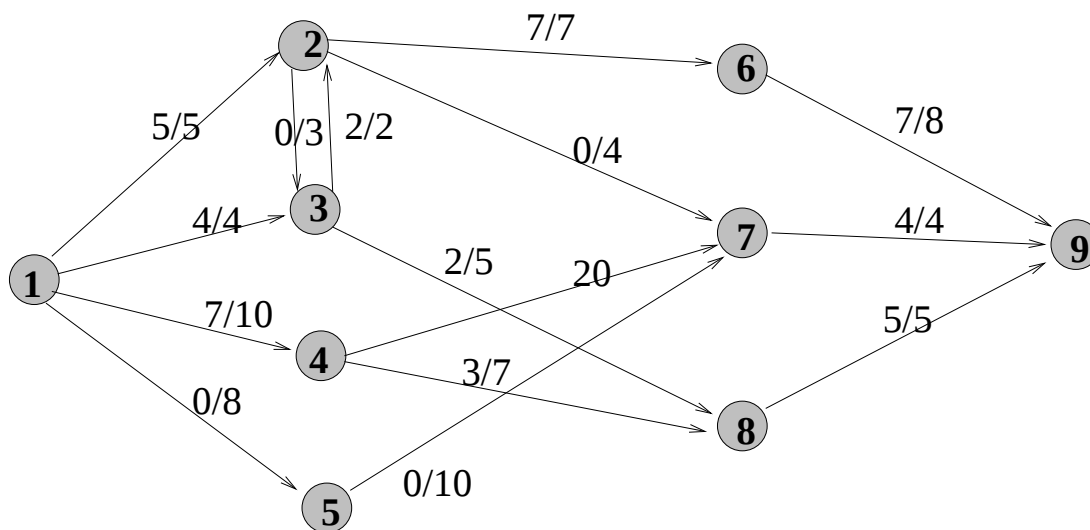
Exemple



Un flot admissible

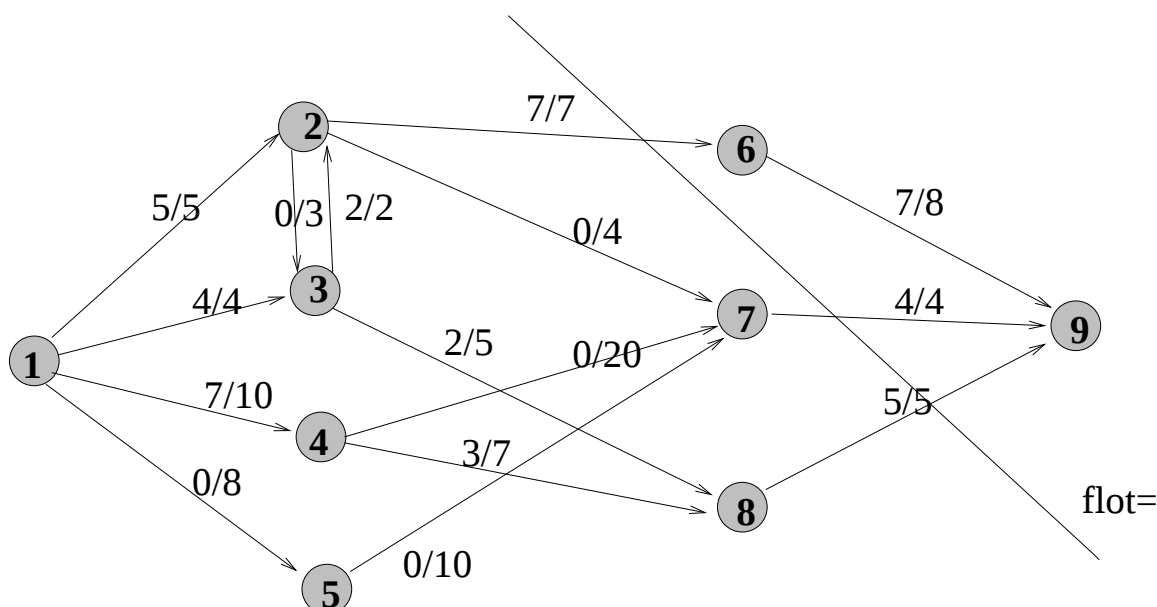


Amélioration



Navigation icons: back, forward, search, etc.

flot maximum



Navigation icons: back, forward, search, etc.