Licence 3 Informatique : cours Graphes I6S3 Chapitre II : Parcours, Arbres, Plus courts chemins et Flots

Olivier Togni,
IEM/LE2I
olivier.togni@u-bourgogne.fr

Modifié le 17 janvier 2024

Plan

Parcours en largeur

L'algorithme de parcours en largeur (BFS, *Breadth First Search*) : parcours d'un graphe par niveaux, à partir d'un sommet (racine) $\Rightarrow$ de manière itérative, en utilisant une file

Utilisé par exemple pour déterminer la connexité d'un graphe ou calculer les distances depuis un sommet

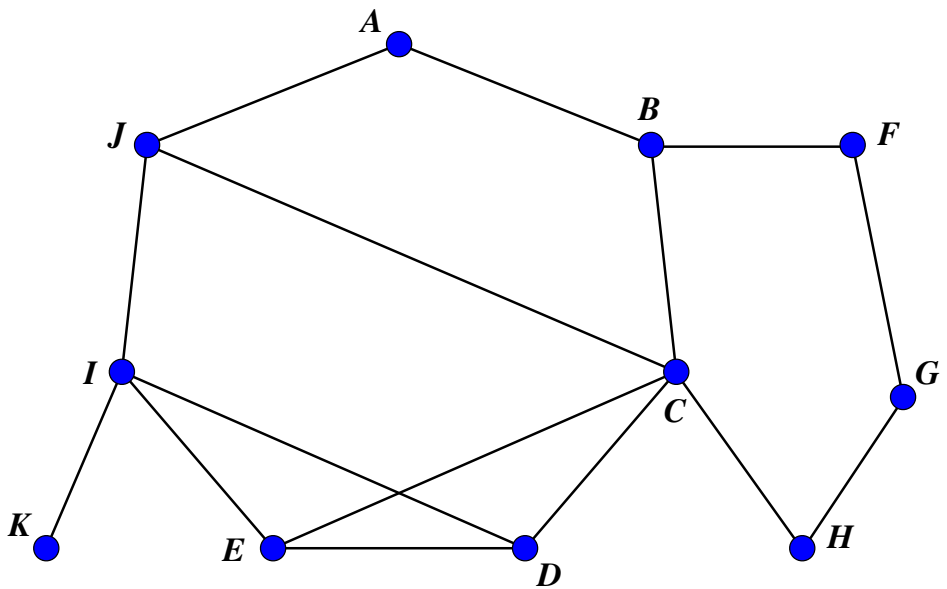
Entrée : un graphe G orienté ou non
Sortie : un arbre BFS T avec pour chaque sommet
p fonction prédécesseur (indique pour chaque sommet qui est son père dans l'arbre BFS),
ℓ niveau (distance de chaque sommet à la racine de l'arbre BFS),
t temps (indique l'ordre de visite des sommets)

Algorithme BFS

```
ParcoursLargeur(Sommet r):
{
  F = CreerFile(); F.enfiler(r);
  marquer(r);
  i = 0; t(r) = i++; l(r) = 0; p(r) = -1;
  TANT QUE NON F.vide() FAIRE
      x = F.defiler();
      traiter(x);
      POUR TOUT voisin y de x FAIRE
           SI y non marqué FAIRE
                p(y) = x; l(y) = l(x)+1; t(y) = i++;
                F.enfiler(y);
                marquer(y);
           FIN SI
      FIN POUR TOUT
  FIN TANT QUE
 }
```

Exemple BFS

```
ParcoursLargeur(Sommet r):
{
  F = CreerFile(); F.enfiler(r);
  marquer(r);
  i = 0; t(r) = i++; l(r) = 0; p(r) = -1;
  TANT QUE NON F.vide() FAIRE
      x = F.defiler();
      traiter(x);
      POUR TOUT voisin y de x FAIRE
           SI y non marqué FAIRE
                p(y) = x; l(y) = l(x)+1;
                t(y) = i++;
                F.enfiler(y);
                marquer(y);
           FIN SI
      FIN POUR TOUT
  FIN TANT QUE
 }
```



Ordre BFS : A, B, J, C, F, I, D, E, H, G, K

Propriétés algorithme BFS

Proposition

Pour tout sommet x de G, $\ell(x) = d_T(r, x) = d_G(r, x)$

Proposition

Toute arête de G joint deux sommets sur le même niveau ou sur deux niveaux consécutifs.

Tout sommet ne peut être ajouté et extrait plus d'une fois $\Rightarrow$ l'algorithme termine !
Initialisation : marquage de tous les sommets $O(|V|)$ opérations
Boucle principale : passage en revue de toutes les arêtes $O(|E|)$ (si listes d'incidence)
Complexité globale : $O(|V| + |E|)$

Parcours en profondeur

L'algorithme de parcours en profondeur (DFS, *Depth First Search*) : parcours de graphe de manière récursive (ou itérative avec un pile)
Permet de déterminer :

- la connexité (d'un graphe non orienté, comme BFS),
- un tri topologique,
- les composante fortement connexes (lancer DFS sur G puis sur G^T)

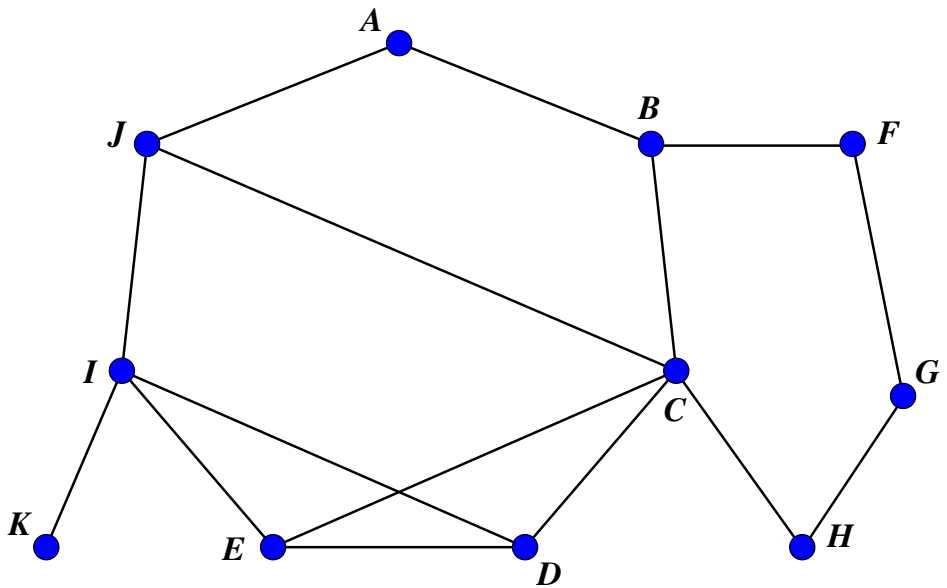
Arbre DFS T représenté par fonction prédécesseur p

Algorithme DFS

```
 DFS (graphe G, sommet x)
{
  Marquer(x);
  POUR CHAQUE voisin y de x FAIRE
     SI y non marqué ALORS
       p(y)=x;
       traiter(y);
       DFS(G,y);
     FIN-SI
  FIN-POUR
}
```

Exemple DFS

```
 DFS (graphe G, sommet x)
{
  Marquer(x);
  POUR CHAQUE voisin y de x FAIRE
     SI y non marqué ALORS
       p(y)=x;
       traiter(y);
       DFS(G,y);
     FIN-SI
  FIN-POUR
}
```



Ordre DFS : A, B, C, D, E, I, J, K, H, G, F

Propriétés algorithme DFS

Proposition

Pour chaque appel récursif de DFS(x), tous les sommets qui sont marqués entre l'invocation de l'appel et son retour sont des descendants de x dans T.

Proposition

Toute arête xy de G non dans T est telle que soit x est ancêtre de y, soit le contraire.

Tout sommet ne peut être ajouté et extrait plus d'une fois $\Rightarrow$ l'algorithme termine !
Initialisation : marquage de tous les sommets $O(|V|)$ opérations
Boucle principale : passage en revue de toutes les arêtes $O(|E|)$ (si listes d'incidence)
Complexité globale : $O(|V| + |E|)$

Composantes fortement connexes

Il existe des algorithmes efficaces pour trouver les composantes fortement connexes d'un graphe orienté :

- Tarjan (1972) : un seul parcours (en profondeur) du graphe + pile et marquage pour distinguer les sommets racine de chaque composante
- Kosaraju (1978) et Sharir (1981) : 2 parcours DFS
  1. DFS sur G à partir de sommet arbitraire + empiler les sommets visités en post-ordre
  2. DFS sur G^T (graphe dans lequel on a inversé le sens des arcs) à partir des sommets en haut de pile

Remarques :

- on peut utiliser BFS du moment que remplissage post-ordre
- utilise la propriété que G et G^T ont les mêmes composantes fortement connexes

Chemins/cycles Eulériens

Définition

Un chemin (cycle) Eulérien dans un graphe G est un chemin passant par chaque arête de G une fois et une seule.
Un graphe est Eulérien s'il possède un cycle Eulérien.

Théorème (Euler (1732)

Un graphe G connexe est Eulérien si et seulement si tous ses sommets sont de degré pair.

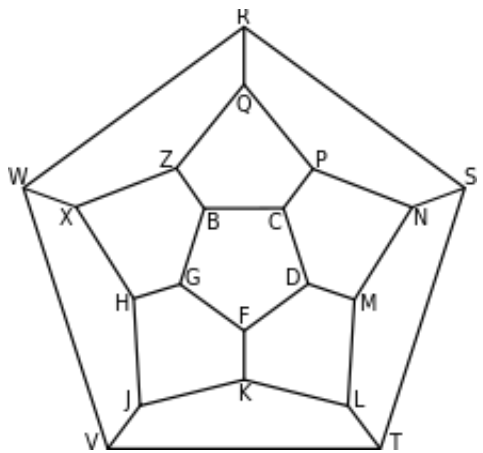
Egalement, G possède un chemin Eulérien s'il a au plus deux sommets de degré impair. $\Rightarrow$ Problème facile !

Chemins/cycles Hamiltoniens

Définition

Un chemin (cycle) Hamiltonien dans un graphe G est un chemin passant par chaque sommet de G une fois et une seule.
Un graphe est Hamiltonien s'il possède un cycle Hamiltonien.

Déterminer si un graphe est Hamiltonien est un problème difficile (NP-complet)



Jeu *Icosian game* de W. R. Hamilton (image de Wikipédia)

Arbre couvrant de poids minimum

Définition

Un arbre couvrant d'un graphe $G = (V, E)$ est un sous-graphe $T = (V_T, E_T)$ de G qui est un arbre et tel que $V_T = V$

Problème ACPM (MWST) :
Entrée Graphe G non orienté, connexe, avec pondération des arêtes (coûts)
Sortie un arbre couvrant T de G dont le poids $w(T)$ (la somme des poids des arêtes) est minimal
Utilité : optimiser les ressources (ex. protocole STP dans les réseaux Ethernet commutés)

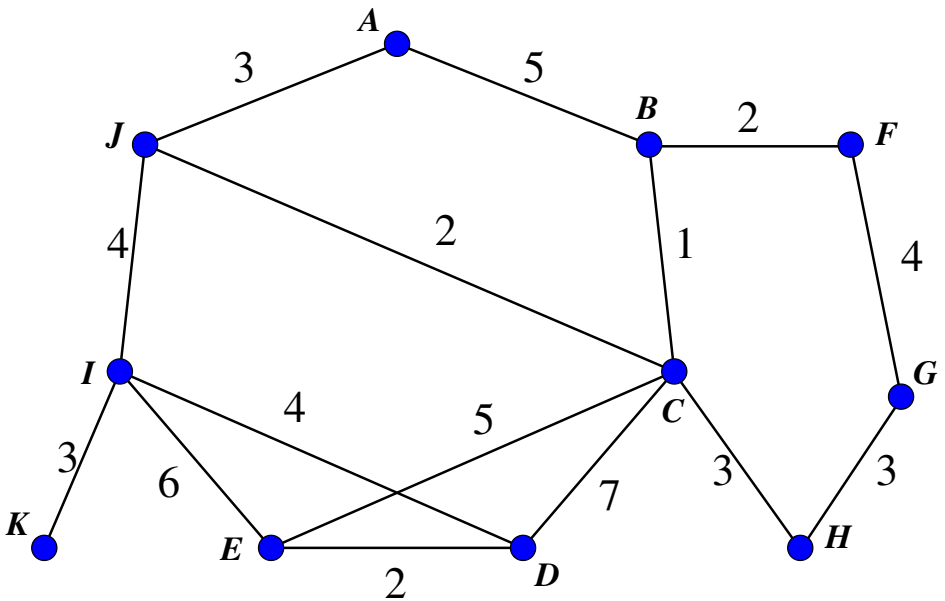
Algorithme de Prim (1956)

```
PRIM (graphe G, poids w)
{
  T=({r},0); // r arbitraire
  TANT QUE il existe un sommet x NON dans T FAIRE
      choisir arête xy avec y dans T et w(xy) minimum
  FIN-TANT QUE
}
```

Complexité : $O(|V| \times |E|)$ dans le pire cas avec listes d'incidence
Optimisation : utiliser une file de priorité pour passer à
$O(\log |V| \times |E|)$

Exemple PRIM

```
PRIM (graphe G, poids w)
{
  T=({r},0);
  TANT QUE T non couvrant
       choisir xy tq y dans T
       et w(xy) min
  FIN-TANT QUE
}
```



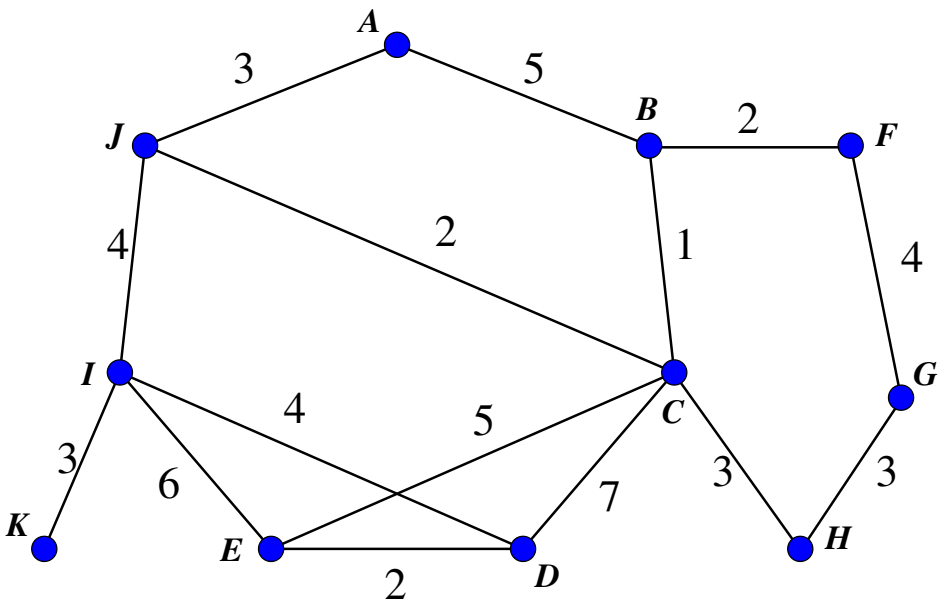
Algorithme de Kruskal (1957)

```
KRUSKAL(graphe G, poids w)
{
  T=(V,0);
  TANT QUE T n'est pas connexe FAIRE
      ajouter à T une arête de poids minimal
      ne formant pas de cycle
  FIN-TANT QUE
}
```

Complexité : $O(|E| \times \log |E|)$ dans le pire cas

Exemple KRUSKAL

```
KRUSKAL(graphe G, poids w)
{
  T=(V,O);
  TANT QUE T non connexe FAIRE
      ajouter à T une arête
      de poids minimal
      ne formant pas de cycle
  FIN-TANT QUE
}
```



Plus courts chemins

Problème PCC (plus courts chemins) :
Entrée Graphe G orienté ou non, (fortement) connexe, avec pondération w des arêtes (coûts)
Sortie les distances entre un sommet source s et tous les sommets avec un arbre des plus courts chemins
Plusieurs algorithmes existent :

- Dijsktra : PPC depuis un sommet vers tous les autres (pas de cycles de poids négatif)
- Bellman-Ford : PPC depuis un sommet vers tous les autres, détecte les cycles de poids négatif

Basés sur la découverte de raccourcis : depuis le sommet r, s'il existe un voisin u d'un sommet v tel que
$dist(r,u) + w(uv) < dist(r,v)$ alors $dist(r,v) = dist(r,u) + w(uv)$

Algorithme de Dijkstra (1959)

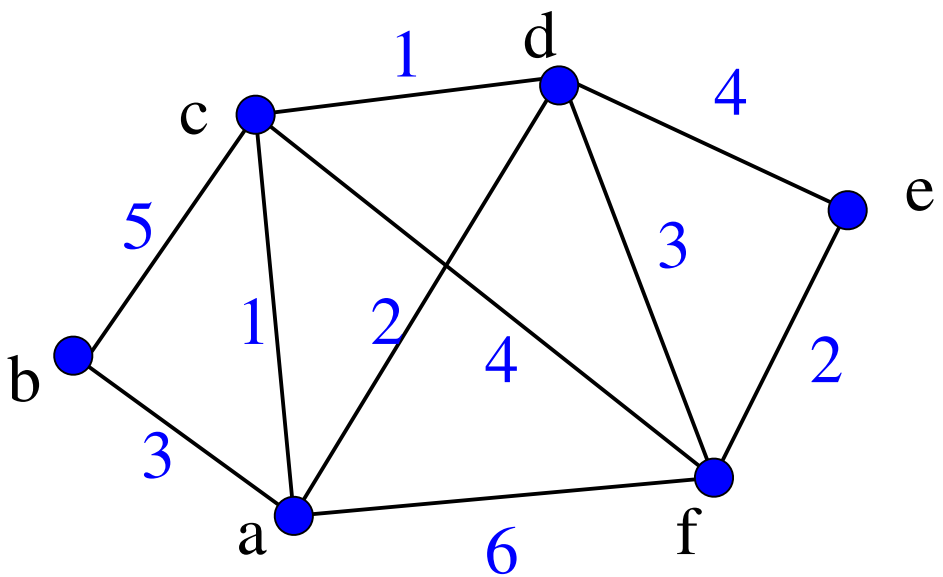
p : prédécesseur dans l'arbre des plus courts chemins

ℓ : distance depuis la source s

```
Dijkstra (graphe G, poids w, source s)
{
  l(s)=0; p(s)=-1; l(v)= 1000000 pour tout autre sommet v;
  TANT QUE il existe un sommet non marqué FAIRE
      choisir le sommet x non marqué avec l(x) minimal;
      marquer x;
      POUR CHAQUE voisin y de x non marqué avec
      l(y) > l(x) + w(xy) FAIRE
        p(y)=x; l(y) = l(x) + w(xy);
      FIN POUR
  FIN-TANT QUE
}
```

Complexité : $O((|E| + |V|) \times \ln |V|)$ dans le pire cas avec utilisation de file de priorité en tas binaire

Exemple DISJKSTRA

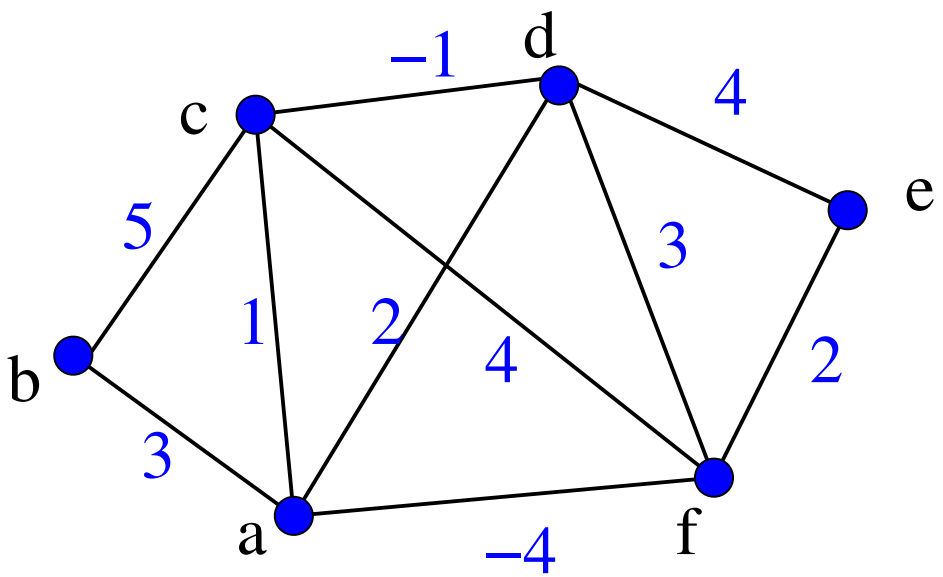


Algorithme de Bellman-Ford (1959)

```
Bellman-Ford (graphe G d'ordre n, poids w, source s)
{
  l(s)=0; p(s)=-1; l(v)= 1000000 pour tout autre sommet v;
  POUR i de 1 à n-1
    POUR TOUT sommet x FAIRE
      POUR CHAQUE voisin y de x avec l(y) > l(x) + w(xy) FAIRE
        l(y)=l(x) + w(x,y);
        p(y)=x;
      FIN POUR
    FIN POUR
  FIN POUR
}
```

Complexité : $O(|E| \times |V|)$

Exemple BELLMAN-FORD avec poids négatifs



Plus courts chemins entre toutes les paires de sommets

Algorithme de Floyd-Warshall : simple à coder (3 boucles imbriquées, complexité en $O(n^3)$)

```
Floyd-Warshall (graphe G d'ordre n, poids w)
{
  POUR i,j entre 0 et n-1 FAIRE
    dist[i,j]=w(i,j) si ij est une arête; dist[i,i]=0
      et dist[i,j]=MAX-INT sinon
  POUR i de 0 à n-1 FAIRE
    POUR j de 0 à n-1 FAIRE
      POUR k de 0 à n-1 FAIRE
dist[i,j]=min(dist[i,j],dist[i,k]+dist[k][j])
      FIN POUR
    FIN POUR
  FIN POUR
}
```

Introduction aux flots

- Problème du plus court chemin : acheminer une unité d'une source vers une destination
- Problème de flot : acheminer une quantité de marchandises (divisibles : on peut acheminer nos marchandises par des routes différentes) de la source vers la destination

Applications :

- logistique : transport de marchandises (train, camion , ...)
- distribution de liquide (eau, pétrole, ...) par canalisations
- électricité : réseau ERDF, ...
- information : internet, réseaux sociaux, ...

Réseau

Définition

Un réseau (de distribution) est un quadruplet (G, c, s, t) avec

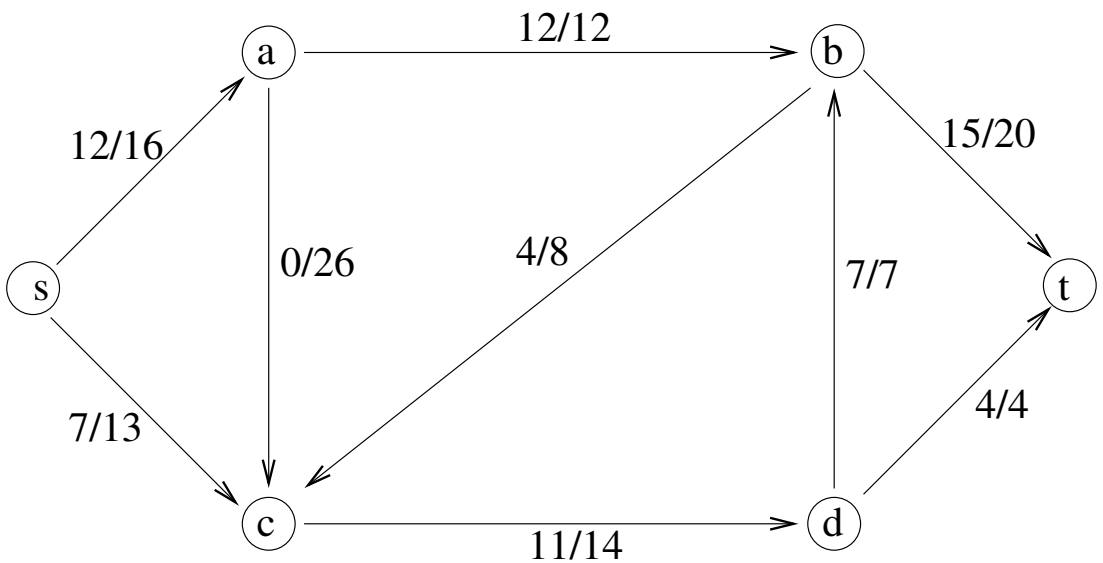
- $G = (V, E)$ graphe orienté connexe et sans boucle
- capacité $c(x, y)$ sur chaque arc
- sommet s de degré entrant nul (source)
- sommet t de degré sortant nul (puit)

Définition

Un flot sur un réseau (G, c, s, t) est une fonction $f : E(G) \to \mathbb{R}$ qui vérifie la loi des nœuds (loi de Kirchoff, 1847) :
ce qui entre = ce qui sort de chaque nœud (différent de s, t).

Exemple de flot

Sur chaque arc : flot/capacité



Flot réalisable

Définition

Le flot f est réalisable si pour tout arc (x, y) on a :

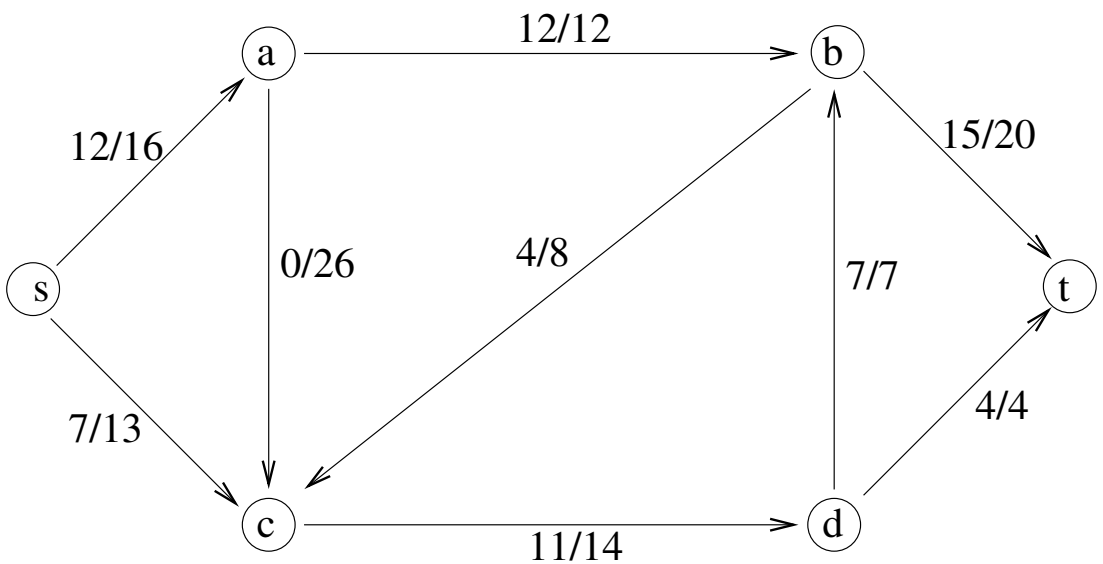
$$0 \leq f(x, y) \leq c(x, y)$$

Définition

La valeur du flot f de s à t est
$v(f) = \sum_{(x,t)\in E(G)} f(x, t) = \sum_{(s,y)\in E(G)} f(s, y)$.

Exemple de flot réalisable

Valeur du flot en rouge : $v = 19$



Flot maximum

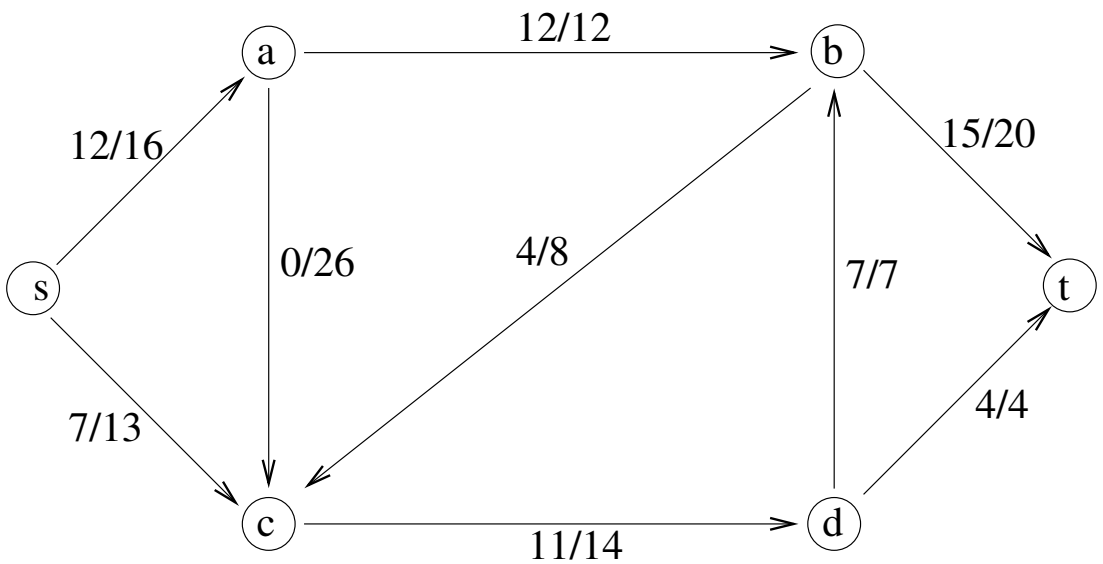
Problème Flot maximum (FM) :
Entrée réseau (G, c, s, t)
Sortie flot f réalisable de valeur maximum
Algorithme de Ford-Fulkerson (1956) :

- procéder par marquage successifs des sommets depuis la source vers le puit
- on traite chaque sommet x marqué successivement
- on marque tout successeur y positivement si l'arc n'est pas saturé $(c(x, y) > f(x, y))$
- on marque tout prédécesseur z négativement si l'arc (z, x) a un flot non nul
- ceci permet de trouver des chaînes augmentantes

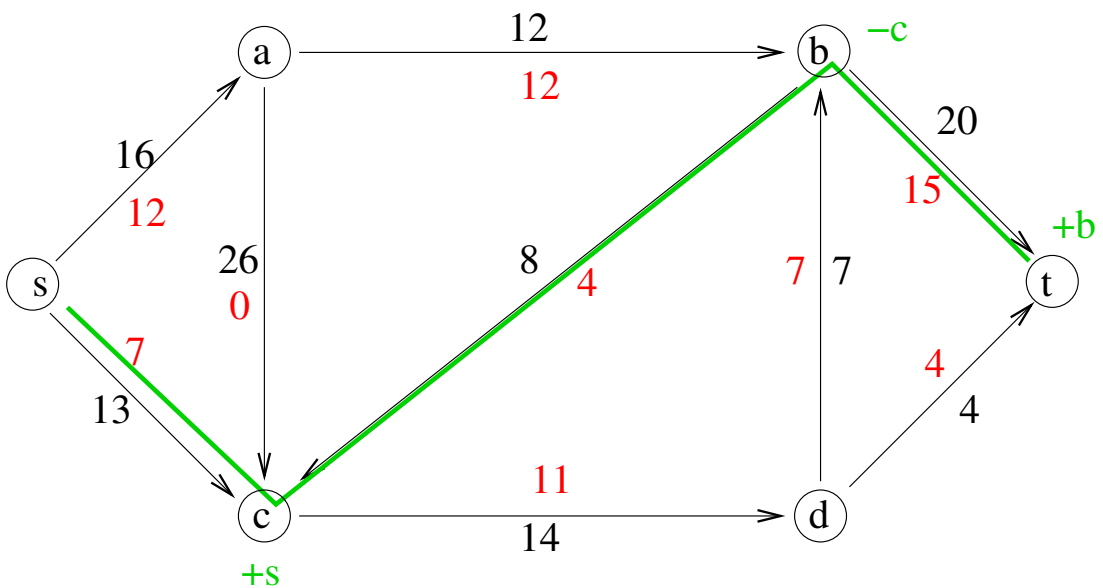
Algorithme FF (Ford-Fulkerson)

```
FF(G, c, s, t)
{
  marque(s)=+;
  TANT QUE le flot n'est pas maximal FAIRE
    TANT QUE on marque des sommets FAIRE
      POUR CHAQUE sommet marqué x non encore traité FAIRE
        POUR CHAQUE arc (x,y) FAIRE
          SI y n'est pas marqué et c(x,y) > f(x,y) ALORS
            marque(y)=(+,x)
          FINSI
        FINPRCH
        POUR CHAQUE arc (y,x) FAIRE
          SI y n'est pas marqué et f(y,x) > 0 ALORS
            marque(y)=(-,x)
          FINSI
        FINPRCH
      FINPRCH
      SI le puit t n'est pas marqué ALORS
        le flot est maximum (on s'arrête)
      SINON
        augmenter le flot et continuer
      FINSI
    FINTQ
  FINTQ
}
```

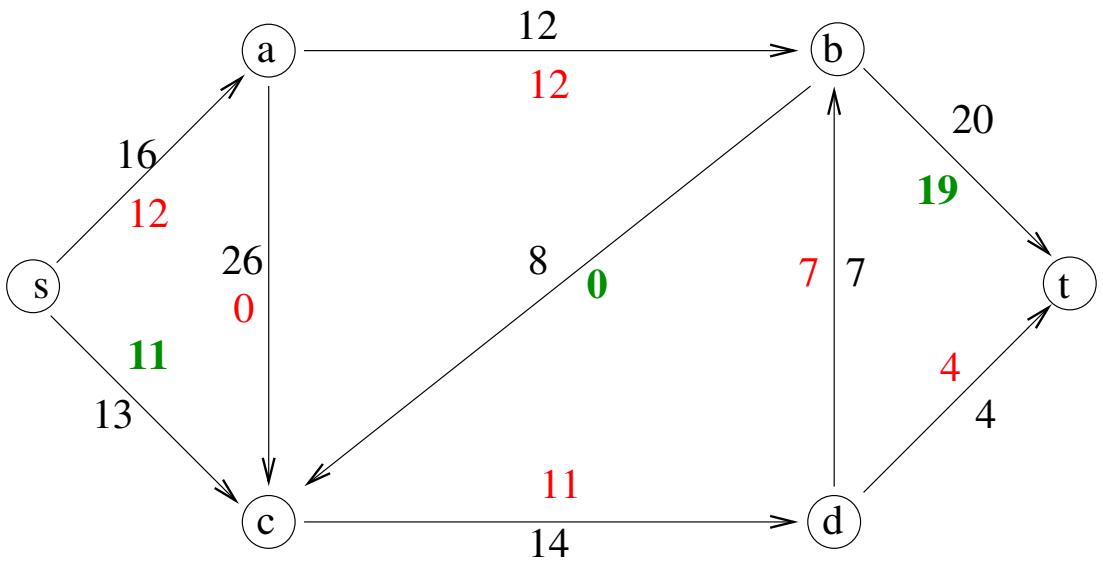
Exemple



Chaîne augmentante



Flot augmenté



Propriétés algorithme FF

Définition

Une coupe dans un réseau (G, c, s, t) est une partition de $V(G)$ en deux ensemble X et $\overline{X}$ telle que :

- $V(G) = X \cup \overline{X}$;
- $X \cap \overline{X} = \emptyset$;
- $s \in X$ et $t \in \overline{X}$

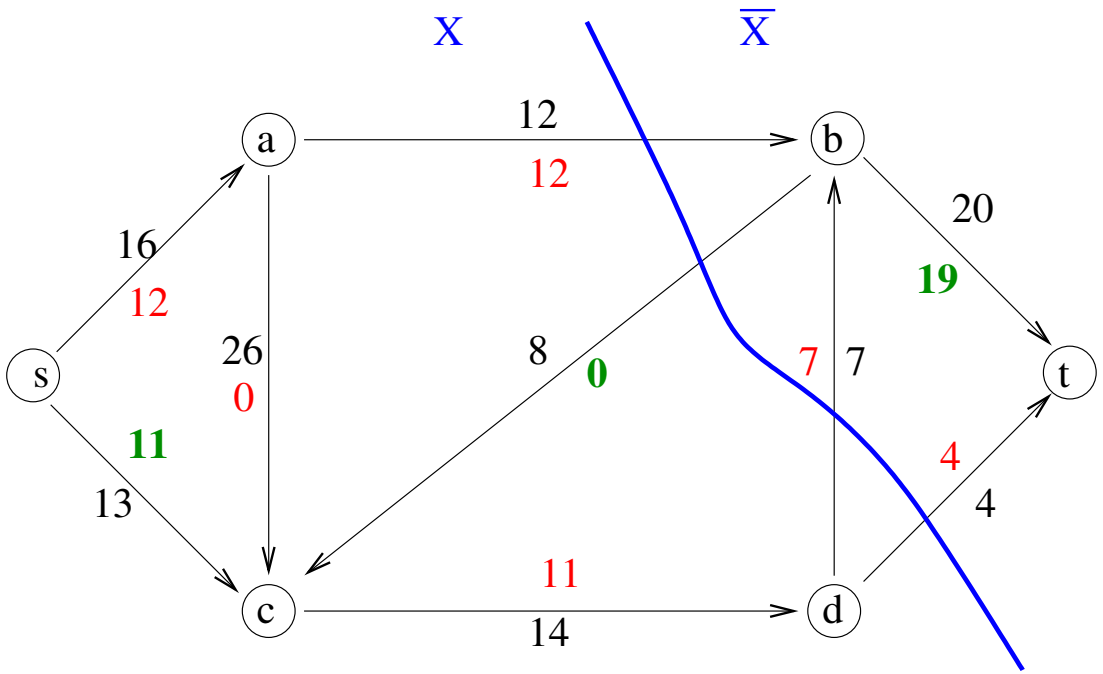
La capacité de la coupe est définie par $c(X, \overline{X}) = \sum_{x \in X, y \in \overline{X}} c(x, y)$

Théorème (Flot Max Coupe Min)

La valeur d'un flot maximum est égale à la capacité minimum d'une coupe.

Le flot est maximum

car sa valeur est de 23 qui est la capacité de la coupe $(X, \overline{X})$



Réseau des écarts ou réseau résiduel

Soit $G = (V, E)$ un graphe orienté et f un flot sur le réseau (G, c, s, t). Le réseau résiduel (G_f, c', s, t) est obtenu ainsi :

- $G_f = (V, E')$ avec
- si $f(x, y) < c(x, y)$ alors $(x, y) \in E'$ et $c'(x, y) = c(x, y) - f(x, y)$;
- si $f(x, y) > 0$ alors $(y, x) \in E'$ et $c'(y, x) = f(x, y)$.

Propriété

Il existe un chaîne augmentante entre s et t dans G pour f si et seulement si il existe un chemin de s à t dans G_f

Finitude et complexité

Si capacités à valeurs entières, l'algorithme FF converge en un nombre fini d'opérations :

- $O(m)$ opérations pour recherche de chaîne augmentante et amélioration du flot
- capacité d'une coupe en $O(n \times c_{\max})$ où $c_{\max}$ est le maximum des capacités des arcs
- dans le pire des cas, le flot augmente d'une seule unité à chaque fois

$\Rightarrow O(nmc_{\max})$ opérations pour l'algorithme de Ford-Fulkerson

Variantes et applications

Algorithme d'Edmonds-Karp (1972)

Implantation particulière de l'algorithme FF avec parcours en largeur (BFS) et qui consiste toujours à choisir une chaîne augmentante de plus court chemin de s à t, c'est à dire celle avec le moins d'arcs possible
Cet algorithme se termine toujours (même pour des capacités non entières, avec une complexité en $O(nm^2)$

Théorème (Menger (1927))

Dans un graphe non orienté, le nombre maximum de chemins arête-disjoints entre deux sommets s et t est égal à la capacité minimum d'une coupe entre s et t