Graphes – Base

Un graphe est un couple G = (V, E) formé par un ensemble fini V et un sous-ensemble E de $\binom{X}{2}$

- V est l'ensemble des sommets de G (on le note aussi V (G)).
- E est l'ensemble des arêtes de G (on le note aussi E(G)).

Définition

- u et v sont **adjacents** si uv ∈ E(G);
- E est **incidente** à u si u ∈ e
- Le voisinage de u dans G est l'ensemble NG(u) des sommets adjacents à u ;
- L'ensemble des arêtes incidentes à u est noté δG(u).

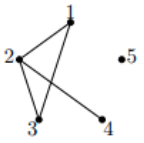
Sous-graphes : Soient G = (V, E) et H = (W, F) deux graphes.

- H est un sous-graphe de G si W ⊆ V et F ⊆ E.
- H est un sous-graphe couvrant de G si W = V et F ⊆ E. (Au moins mêmes sommets)
- H est un sous-graphe induit de G si W ⊆ V et F contient toutes les arêtes uv ∈ E où u, v ∈ W. On le note G[W].

Graphes complémentaires : les arêtes de G sont les non-arêtes de $\overline{G}$, et vice versa.

Graphes complets : tous les sommets sont liés à tous les autres.

Représentations



Graphique

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$
Matrice d'adjacence (V*V)

$$N = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$
Matrice d'incidence (E*V)

1 : [2,3]
2 : [1,3,4]
3 : [1,2]
4 : [2]
5 : []

Liste d'adjacence

Rappel complexité

f ∈ O(g)	f ∈ Θ(g)	f ∈ Ω(g)
f ≤ g	f ∈ O(g) et f ∈ Ω(g)	f ≥ g

Chaîne : suite de la forme (v0, e1, v1, . . ., ek, vk), k est la longueur, est élémentaire si ses sommets sont deux à deux distincts notée Pn

Cycle : Un cycle est une chaîne de longueur supérieure ou égale à 1 simple et fermée

Forêts : graphe acyclique

Connexité : Un graphe G est connexe s'il existe une chaîne entre u et v, pour toute paire de sommets

Relations d'équivalence : G = (V, E) un graphe et mettons u ~ v ssi il existe une chaîne entre u et v.

Arbres : graphe connexe et acyclique.

Arbres couvrants : sous-graphe couvrant de G qui est un arbre

graphe connexe

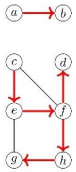

graphe non connexe

Algorithmes de parcours

Entrées : graphe $G = (V, E)$ et sommet $s \in V$
début
  créer file(Q) ; FIFO
  marquer(s) ;
  enfiler(Q, s) ;
  tant que $Q \neq \emptyset$ **faire**
    $u \leftarrow$ défiler(Q) ;
    pour tous les $uv \in E$ **faire**
      si v non marqué **alors**
        marquer(v) ;
        enfiler(Q, v)

Entrées : graphe $G = (V, E)$ et sommet $s \in V$
début
  pour tous les $u \in V \setminus \{s\}$ **faire** initialisa° dist
    $d(u) \leftarrow \infty$;
  dist(s) $\leftarrow 0$;
  $Q \leftarrow [s]$; → FIFO
  tant que $Q \neq \emptyset$ **faire**
    $u \leftarrow$ défiler(Q) ;
    pour tous les $uv \in E$ **faire**
      si $d(v) = \infty$ **alors**
        $d(v) = d(u) + 1$;
        enfiler(Q, v) ;
        parent(v) $\leftarrow u$

Complexité : O(n+m)



Procédure explorer (G, u) :
  marqué[u] $\leftarrow$ Vrai
  pour tous les $(u, v) \in E(G)$ **faire**
    si marqué[v] = Faux **alors**
      explorer (G, v)

Procédure DFS (G) :
  pour tous les $u \in V(G)$ **faire**
    marqué[u] $\leftarrow$ Faux
  pour tous les $u \in V(G)$ **faire**
    si marqué[u] = Faux **alors**
      explorer (G, u)

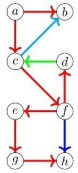
Procédure prévisite (u) :
  pre[s] $\leftarrow t$
  $t \leftarrow t + 1$

Procédure explorer (G, u) :
  marqué[u] $\leftarrow$ Vrai
  prévisite(u)
  pour tous les $(u, v) \in E(G)$ **faire**
    si v non marqué **alors**
      explorer (G, v)
  postvisite(u)

Procédure postvisite (u) :
  post[s] $\leftarrow t$
  $t \leftarrow t + 1$

Procédure DFS (G) :
  $t = 1$
  pour tous les $u \in V(G)$ **faire**
    marqué[u] $\leftarrow$ Faux
  pour tous les $u \in V(G)$ **faire**
    si marqué[u] = Faux **alors**
      explorer (G, u)

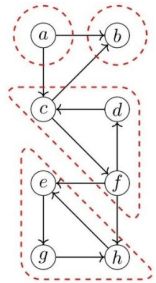
Classification des arêtes et arcs

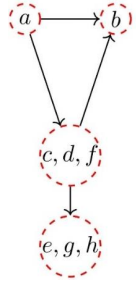


Un parcours en profondeur dans un graphe orienté G donne lieu a 4 types d'arcs de G.
On dit que l'arc (u, v) est :

1. un arc *de l'arbre* si u est un parent de v. [u [v]v]u
2. *avant* si u est un ancêtre (non parent) de v
3. *retour* si v est un ancêtre de u [v [u]u]v
4. *transverse* dans les autres cas [u]u [v]v

Composantes fortement connexes





DAG

Un algorithme de composantes fortement connexes

1. Exécuter un parcours en profondeur sur GR.
2. Exécuter un parcours en profondeur sur le graphe non-orienté sous-jacent de G, en traitant les sommets par nombre post décroissant.

PCC

Il peut ne pas exister de plus court chemin de u à v :

- S'il n'y a aucun chemin : dist(u, v) = ∞
- S'il y a un circuit négatif sur le chemin : dist(u, v) = −∞

Algorithme de Dijkstra et files de priorités

pour tous les $u \in V$ **faire**
  $D[u] \leftarrow +\infty$
  prev[u] $\leftarrow \emptyset$
$D[s] \leftarrow 0$
$H \leftarrow$ makequeue(V)
tant que $H \neq \emptyset$ **faire**
  $u \leftarrow$ deletemin(H)
  pour tous les $(u, v) \in E$ **faire**
    si $D[v] > D[u] + w(u, v)$ **alors**
      $D[v] = D[u] + w(u, v)$
      prev[v] $\leftarrow u$
      decreasekey(H, v)

Principe :
On regarde à chaque tour le sommet ayant la plus petite valeur.

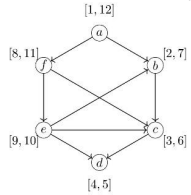
On fait une MAJ des valeurs son voisinage.

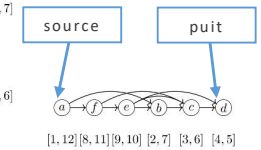
On l'enlève de la liste des sommets

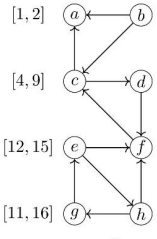
Chemins : équivalent des chaînes dans un graphe orienté

Circuits : équivalent d'un cycle dans un graphe orienté

Tri topologique : ordre total ≺ sur V tel que, pour tout arc (u, v) ∈ E, on a u ≺ v. **Il ne doit pas y avoir de cycle.** Il suffit alors de faire un DFS, et trier les sommets de G par ordre décroissant de post(·).

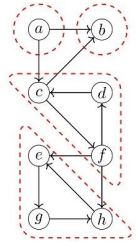






G^R

g, e, h, b, c, d, f, a


G

$\{g, e, h\}, \{b\}, \{c, d, f\}, \{a\}$

Algorithme de Bellman–Ford

$D[s] \leftarrow 0$
prev[s] $\leftarrow s$
pour tous les $u \in V \setminus \{s\}$ **faire**
  $D[u] \leftarrow +\infty$
  prev[u] $\leftarrow \emptyset$
repéter $|V| - 1$ **fois**
  pour tous les $e \in E$ **faire**
    maj(e)
retourner D, prev

Principe :
On fait la MAJ de chaque arête |V|-1 fois

Si cela change au bout d'un Ve tour => cycle négatif

Algorithme de Floyd–Warshall

pour tous les $i \in \{1, \ldots, n\}$ **faire**
  pour tous les $j \in \{1, \ldots, n\}$ **faire**
    dist$(i, j, 0) \leftarrow \infty$
pour tous les $(i, j) \in E$ **faire**
  dist$(i, j, 0) \leftarrow \ell(i, j)$
pour tous les $k \in \{1, \ldots, n\}$ **faire**
  pour tous les $i \in \{1, \ldots, n\}$ **faire**
    pour tous les $j \in \{1, \ldots, n\}$ **faire**
      dist$(i, j, k) = \min\{\text{dist}(i, k, k-1) + \text{dist}(k, j, k-1), \text{dist}(i, j, k-1)\}$

Principe :
A chaque tour on vérifie si
dist(i,j,k-1) > dist(i,k,k-1)+dist(k,j,k-1)

Algorithme de plus court chemin dans les DAG

On met à jour les arcs du voisinages sommets dans l'ordre topologique

$D[s] \leftarrow 0$
$\texttt{prev}[s] \leftarrow s$
pour tous les $u \in V \setminus \{s\}$ **faire**
  $D[u] \leftarrow +\infty$
  $\texttt{prev}[u] \leftarrow \emptyset$
Tri topologique de G
pour tous les $u \in V$ *dans l'ordre topologique* **faire**
  pour tous les $(u, v) \in E$ **faire**
    $\texttt{maj}(u, v)$
retourner $D, \texttt{prev}$

Arbre couvrant de poids minimum

L'algorithme de Kruskal

Trier les arêtes E par poids croissant
pour tous les $e \in E$ **faire**
  si $(V, X \cup \{e\})$ *est acyclique* **alors**
    $X \leftarrow X \cup \{e\}$

A chaque tour on ajoute l'arête avec la plus petite valeur, sauf si on forme un cycle

Fonctions sur les sets

Rank(x) : hauteur de la sous arborescence ayant la racine x
Makeset(x) : x est le parent du set, de rang 0
Find(x) : on retourne le parent du set
Union(x) : fusion entre deux set A et B

- Si rA est la racine de A et rB est la racine de B, il suffit de définir π(rA) = rB ou π(rB) = rA.
- Si la hauteur de A est supérieure a celle de B, et on définit π(rA) = rB, alors la hauteur du nouveau arbre augmente de 1.
- Par contre, si on définit π(rA) = rB, alors la hauteur n'augmente pas.

Algorithme de Johnson

1. Calculer G'.
2. Appliquer Bellman–Ford à G', avec source s, pour calculer $h_v := \text{dist}(s, v)$ pour tout $v \in V(G)$ (ou trouver un cycle négatif)
3. Repondérer chaque arc $(u, v) \in E(G)$ par $\ell'_{(u,v)} = \ell_{(u,v)} + h(u) - h(v)$.
4. Pour chaque $u \in V(G)$, exécuter Dijkstra pour calculer $\text{dist}_{\ell'}(u, v)$ pour tout $v \in V(G)$.
5. Pour chaque couple u, v, on a $\text{dist}_\ell(u, v) = \text{dist}_{\ell'}(u, v) + h(v) - h(u)$.

L'algorithme de Prim

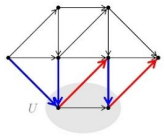
pour tous les $u \in V$ **faire**
  makeset(u)
$X \leftarrow \varnothing$
Trier les arêtes E par poids croissant
pour tous les $uv \in E$, *dans l'ordre croissant de poids* **faire**
  si *find*$(u) \neq$ *find*(v) **alors**
    $X \leftarrow X \cup \{uv\}$
    union(u, v)

Principe :
Choisir arbitrairement un sommet s et le marquer

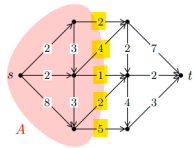
Tant que ∃ un sommet v non marqué adjacent à un sommet marqué u, on prend un sommet v non marqué minimisant le poids de l'arête uv

Flots

Arcs entrants et sortants

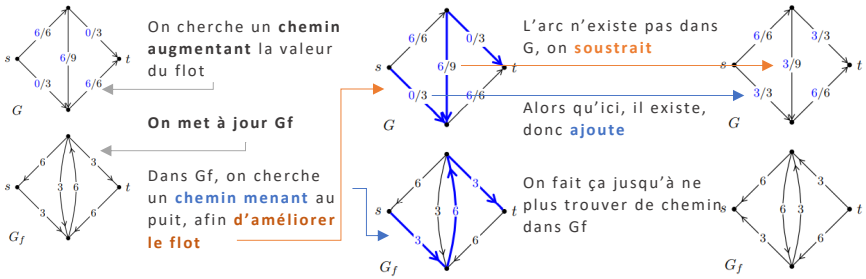


Arcs entrants et sortants
δ +(U) : arcs sortant de U
δ −(U) : arcs rentrant dans U
Cap(A,B) = 2 + 4 + 1 + 2 + 5 = 14



Algorithme de Ford–Fulkerson (flot max) – complexité O(mC)

On initialise un flot nul pour G et un graphe résiduel de G, Gf

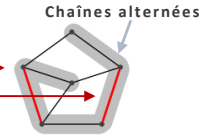


On cherche un **chemin augmentant** la valeur du flot
On met à jour Gf
Dans Gf, on cherche un **chemin menant** au puit, afin **d'améliorer le flot**
L'arc n'existe pas dans G, on **soustrait**
Alors qu'ici, il existe, donc **ajoute**
On fait ça jusqu'à ne plus trouver de chemin dans Gf

L'algorithme de Edmonds–Karp : on utilise un BFS pour trouver un chemin augmentant avec le nombre minimum d'arcs. Complexité : O(nm²)

Couplages

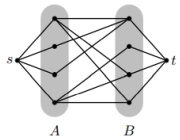
Couplages : sous ensemble d'arête qui ne partagent pas les mêmes sommets



Théorème de Berge : commencer par un couplage de taille 1. S'il existe une chaîne augmentante, augmenter le couplage. Répéter jusqu'à ce qu'il n'y ait aucune chaîne augmentante.

Graphes bipartis

Des couplages aux flots



Algorithme couplages maximaux dans les graphes bipartis

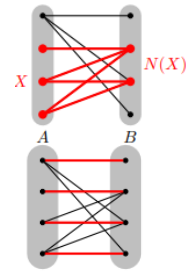
- Ajouter deux nouveaux sommets s et t au graphe biparti
- Ajouter une arête entre s et chaque sommet de A
- Ajouter une arête entre t et chaque sommet de B
- Orienter toutes les arêtes de gauche à droite
- Donner une capacité de 1 à chaque arc.
- Appliquer l'algorithme de Ford–Fulkerson

Transversal : ensemble de sommets qui ont accès à toutes les arêtes

Théorème de König : Si G est biparti, alors τ (G) (taille min transversal) = ν(G) (taille max couplage).

Théorème de Hall

Soit G = (A, B) un graphe biparti. Alors G a un couplage couvrant tous les sommets de A ssi |N(X)| ≥ |X| pour tous X ⊆ A.



Conséquences du théorème de Hall

Soit G un graphe biparti. G a un couplage parfait ssi |A| = |B| et |N(X)| ≥ |X| pour tout X ⊆ A

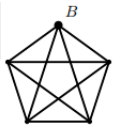
Corollaire

Soit G = (A, B) un graphe biparti, avec tous les sommets de degré k, pour un entier k ≥ 1. Alors G a un couplage parfait.

Algorithmes d'approximation

Lorsque l'on n'arrive pas à trouver un algorithme polynomial pour un problème d'optimisation, on peut espérer de trouver un algorithme d'approximation.

$$\rho = \max_I \frac{\mathcal{A}(I)}{\text{OPT}(I)}.$$



- OPT(I) la solution optimale de l'instance I.
- Soit A un algorithme (polynomial) qui retourne une solution A(I) de l'instance I.

Algorithme : Vertex-Cover-Approx
Entrées : un graphe $G = (V, E)$
Sorties : un transversal (vertex cover) T de G
début
  $T \leftarrow \emptyset$
  $F \leftarrow E$
  tant que $F \neq \emptyset$ **faire**
    Soit uv une arête de F
    $T \leftarrow T \cup \{u, v\}$
    $F \leftarrow F \setminus \{\delta(u) \cup \delta(v)\}$
  retourner T

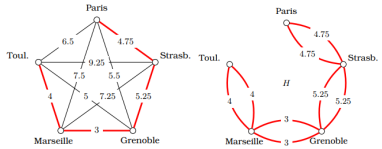
Principe :

Graphes eulériens

Un cycle C dans un graphe G est eulérien si C **traverse chaque arête** de G une et **une seule fois**.

Un graphe G est eulérien si et seulement si **G est connexe**, et **tout sommet de G est de degré pair**.

Algorithme Double-Tree (2-approximation problème du voyageur)



Principe :
(1) Trouver un ACM
(2) Construire le multigraphe H
(3) Trouver un cycle eulérien dans H.
(4) Supprimer tout sauf la 1ère occurrence de chaque sommet (+ dernière occurrence de v1).
Complexité : O(n 2 log n).

L'algorithme de Christofides (cycle hamilt.)

Cycle hamiltonien : cycle contenant tous les sommets du graphe

Trouver un arbre couvrant T de poids minimum dans G
Trouver l'ensemble U ⊆ V de sommets de degré impair dans T
Trouver un couplage parfait M de poids minimum dans G[U]
Construire un graphe eulérien H en ajoutant les arêtes de M à T
Trouver un cycle eulérien C'' de H
Faire des "raccourcis" pour obtenir un cycle hamiltonien C ⊆ G (comme l'algorithme Double-Tree)