Introduction à la Recherche Opérationnelle

Rodéric Moitié, Christophe Osswald, Jordan Ninin

ENSTA Bretagne

2016

Sommaire

Sommaire

Sommaire

Complexités

Complexité en temps

- Donald Knuth fut un des premiers à l'appliquer ;
- évaluation de l'efficacité des algorithmes ;

Complexités

Complexité en temps

- Donald Knuth fut un des premiers à l'appliquer ;
- évaluation de l'efficacité des algorithmes ;
- nombre d'opérations par rapport à la taille des entrées ;

Complexités

Complexité en temps

- Donald Knuth fut un des premiers à l'appliquer ;
- évaluation de l'efficacité des algorithmes ;
- nombre d'opérations par rapport à la taille des entrées ;
- évaluation dans le pire des cas, en moyenne.

Complexités

Complexité en temps

- Donald Knuth fut un des premiers à l'appliquer ;
- évaluation de l'efficacité des algorithmes ;
- nombre d'opérations par rapport à la taille des entrées ;
- évaluation dans le pire des cas, en moyenne.

Complexité en espace

- mesure de l'espace mémoire utilisé

Bons algorithmes

	20	50	100	200	500
$10^3 n$	0.02 s	0.05 s	0.1 s	0.2 s	0.5 s
$10^3 n \log n$	0.09 s	0.3 s	0.6 s	1.5 s	4.5 s
$100n^2$	0.04 s	0.25 s	1 s	4 s	25 s
$10n^3$	0.02 s	1 s	10 s	1 mn	21 mn
$n^{\log n}$	0.4 s	1.1 h	220 j	125 s	5.10^8 s
$2^{\frac{n}{3}}$	10^{-4} s	0.1 s	2.7 h	3.10^4 s	
2^n	1 s	0.36 s			
3^n	58 mn	2.10^9 s			
$n!$	771.00 s				

Sommaire

Machines de Turing

- Décrites en 1936 par Alan Turing
- Modèle d'ordinateur
- Composition :
  - bande infinie des deux côtés (mémoire)
  - tête de lecture
  - un état interne
  - une fonction de transition

Théorie de la complexité

Objectif : déterminer les problèmes intrinsèquement difficiles.

Théorie de la complexité

Objectif : déterminer les problèmes intrinsèquement difficiles.

Définition (Problème décidable/indécidable)

Un problème de décision est dit décidable s'il existe un algorithme (ou une machine de Turing) qui le décide. Sinon il est indécidable.

Théorie de la complexité

Objectif : déterminer les problèmes intrinsèquement difficiles.

Définition (Problème décidable/indécidable)

Un problème de décision est dit décidable s'il existe un algorithme (ou une machine de Turing) qui le décide. Sinon il est indécidable.

- Problème indécidable : difficile
  - Exemple : problème de l'arrêt

Théorie de la complexité

Objectif : déterminer les problèmes intrinsèquement difficiles.

Définition (Problème décidable/indécidable)

Un problème de décision est dit décidable s'il existe un algorithme (ou une machine de Turing) qui le décide. Sinon il est indécidable.

- Problème indécidable : difficile
  - Exemple : problème de l'arrêt
- Autre problèmes difficiles ?

Théorie de la complexité

- données en entrée
- question sur ces données
- réponse oui/non

Théorie de la complexité

- données en entrée
- question sur ces données
- réponse oui/non

Exemple (Problème SAT)

- $X = \{x_1, ..., x_n\}$ variables booléennes.
- $E = C_1 \wedge ... \wedge C_m$ expression
- $\forall i, C_i = u_{i1} \vee ... \vee u_{ik}$
- u_{ij} : variable de X complémentée ou non
- Problème : déterminer x_i tel que $E = vrai$

Théorie de la complexité

- données en entrée
- question sur ces données
- réponse oui/non

Exemple (Problème SAT)

Ex :
$E = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_2) \wedge (x_3)$
réponse *VRAI* pour $x_1 = 0$ ou 1, $x_2 = 1$, $x_3 = 1$.

Classes P/NP

Définition (Classe P)

$\mathcal{P} \in P$: il existe un algorithme polynomial permettant de le résoudre

Classes P/NP

Définition (Classe P)

$\mathcal{P} \in P$: il existe un algorithme polynomial permettant de le résoudre

Exemple (Classe P)

Problèmes de tris

Classes P/NP

Définition (Classe P)

$\mathcal{P} \in P$: il existe un algorithme polynomial permettant de le résoudre

Exemple (Classe P)

Problèmes de tris

Algorithme polynomial = « bon » algorithme

Classes P/NP

Définition (Classe NP)

$\mathcal{P} \in NP$: il existe un algorithme polynomial sur une machine de Turing non déterministe permettant de le résoudre.

Machine non déterministe $=$ machine déterministe $+$ *oracle*

Classes P/NP

Définition (Classe NP)

$\mathcal{P} \in NP$: il existe un algorithme polynomial sur une machine de Turing non déterministe permettant de le résoudre.

Machine non déterministe $=$ machine déterministe $+$ *oracle*

Définition (Classe NP)

$\mathcal{P} \in NP$: il existe un algorithme polynomial sur une machine de Turing déterministe permettant de vérifier une solution de $\mathcal{P}$.

Remarques classe NP

- $NP \neq$ problèmes non polynomiaux

Remarques classe NP

- $NP \neq$ problèmes non polynomiaux
- $NP =$ problèmes vérifiables en temps polynomial

Remarques classe NP

- $NP \neq$ problèmes non polynomiaux
- $NP =$ problèmes vérifiables en temps polynomial
- $P \subseteq NP$

Remarques classe NP

- $NP \neq$ problèmes non polynomiaux
- $NP =$ problèmes vérifiables en temps polynomial
- $P \subseteq NP$
- autre inclusion ?

Remarques classe NP

- $NP \neq$ problèmes non polynomiaux
- $NP =$ problèmes vérifiables en temps polynomial
- $P \subseteq NP$
- autre inclusion ?
- problème à \$1.000.000

Classes

Quelques classes de complexité :

- L/NL : algorithme logarithmique en espace
- P/NP/Co-NP : algorithme polynomial en temps
- PSPACE/NPSPACE : algorithme polynomial en espace
- EXPTIME : algorithme exponentiel en temps

Problèmes NP-Complets

Définition

Un problème est NP-complet si tout problème de NP se réduit polynomialement à lui.

Problèmes NP-Complets

Définition

Un problème est NP-complet si tout problème de NP se réduit polynomialement à lui.

Théorème (Cook)

Il existe des problèmes NP-complets.

Problèmes NP-Complets

Définition

Un problème est NP-complet si tout problème de NP se réduit polynomialement à lui.

Théorème (Cook)

Il existe des problèmes NP-complets.

- beaucoup de problèmes dans cette classe
- impossibilité de les résoudre efficacement (non déterminisme)

Sommaire

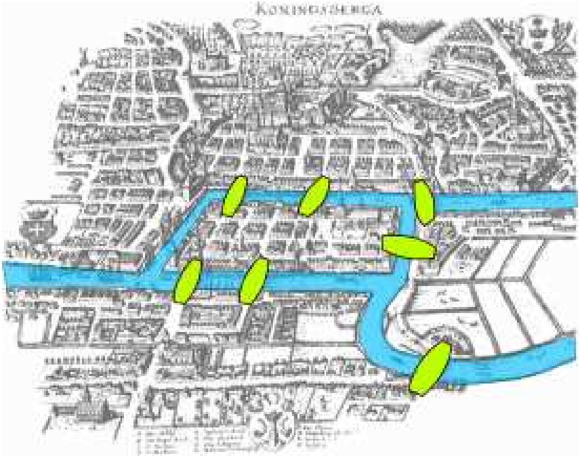
Sommaire

Sommaire

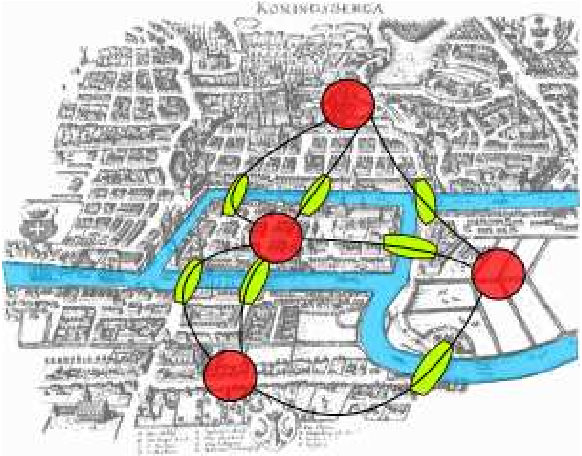
Historique

- 1736 Euler : les 7 ponts de Königsberg

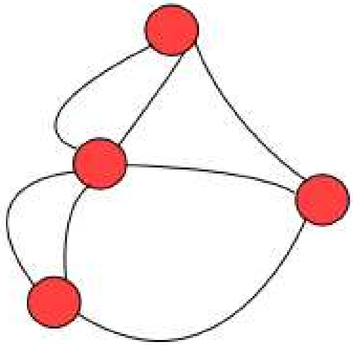
Historique



Historique



Historique



Historique

- 1736 Euler : les 7 ponts de Königsberg
- 1845 : loi de Kirchhoff dans les circuits électriques

Historique

- 1736 Euler : les 7 ponts de Königsberg
- 1845 : loi de Kirchhoff dans les circuits électriques
- 1852 : Guthrie pose le problème des 4 couleurs (résolu en 1976)

Sommaire

Définitions

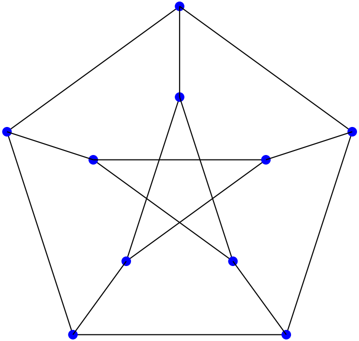
Définition (Graphe)

Un graphe G est un couple (S, A) où S est un ensemble fini et $A \subseteq S \times S$ une relation binaire sur S. S est appelé ensemble des sommets, et A ensemble des arcs. Un graphe est dit pondéré si un poids est associé à chacun de ses arcs.

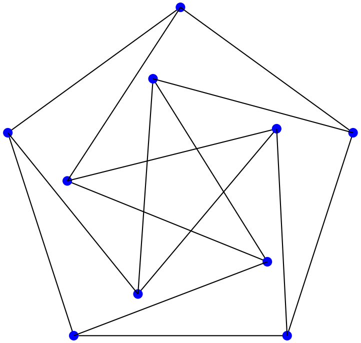
On distingue deux types de graphes :

- les graphes orientés
- les graphes non orientés

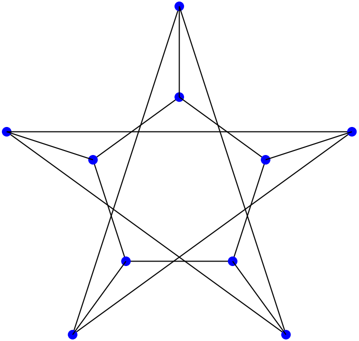
Exemple : le graphe de Petersen



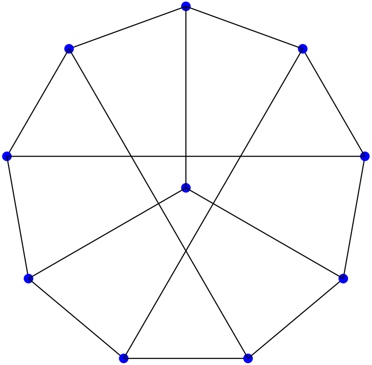
Exemple : le graphe de Petersen



Exemple : le graphe de Petersen



Exemple : le graphe de Petersen



Définition (Chemin, chaîne)

Dans un graphe orienté, un chemin entre deux sommets a et b est une suite finie de n sommets (s_i) tels que $a = s_1$, $b = s_n$ et $\forall i \in [1, n-1], (s_i, s_{i+1}) \in A$. Un chemin est dit élémentaire si chaque sommet est présent au plus une fois.
Dans le cas d'un graphe non orienté, on parle de chaîne.
S'il existe un chemin entre a et b, on notera $a \rightsquigarrow b$.

Définition (Circuit, cycle, boucle)

Un circuit est un chemin dont le sommet initial coïncide avec le sommet final. Une boucle est un circuit de longueur 1. Dans le cas d'un graphe non orienté, on parle de cycle.

Définition (Degré)

Dans un graphe non orienté, le degré d'un sommet est le nombre d'arêtes issues de ce sommet.

Définition (Connexité)

Un graphe non orienté est dit connexe si pour tout couple de sommets a et b du graphe, il existe une chaîne entre a et b.

Définition (Connexité forte)

Un graphe orienté est dit fortement connexe si pour tout couple de sommets a et b du graphe, $a \rightsquigarrow b$ et $b \rightsquigarrow a$.

Définition (Planarité)

Un graphe est dit planaire s'il existe au moins une façon de le dessiner dans le plan sans que deux arêtes ne se croisent.

Définition (Graphe complet)

Un graphe complet est un graphe pour lequel tous les sommets sont reliés entre eux deux à deux. Le graphe complet à n sommets est noté K_n.

Sommaire

Cycle Eulérien

Définition (Cycle Eulérien)

Cycle Eulérien : cycle qui inclut tous les arcs du graphe une et une seule fois.

Cycle Eulérien

Définition (Cycle Eulérien)

Cycle Eulérien : cycle qui inclut tous les arcs du graphe une et une seule fois.

Définition (Chaîne Eulérienne)

Chaîne Eulérienne : chaîne qui inclut tous les arcs du graphe une et une seule fois.

Cycle Eulérien

Définition (Cycle Eulérien)

Cycle Eulérien : cycle qui inclut tous les arcs du graphe une et une seule fois.

Définition (Chaîne Eulérienne)

Chaîne Eulérienne : chaîne qui inclut tous les arcs du graphe une et une seule fois.

Théorème

Un graphe est Eulérien ssi tous ses sommets sont de degré pair.

Cycle Hamiltonien

Définition (Cycle Hamiltonien)

Cycle Hamiltonien : cycle qui inclut tous les sommets du graphe une et une seule fois.

Cycle Hamiltonien

Définition (Cycle Hamiltonien)

Cycle Hamiltonien : cycle qui inclut tous les sommets du graphe une et une seule fois.

Définition (Chaîne Hamiltonienne)

Chaîne Hamiltonienne : chaîne qui inclut tous les sommets du graphe une et une seule fois.

Cycle Hamiltonien

Définition (Cycle Hamiltonien)

Cycle Hamiltonien : cycle qui inclut tous les sommets du graphe une et une seule fois.

Définition (Chaîne Hamiltonienne)

Chaîne Hamiltonienne : chaîne qui inclut tous les sommets du graphe une et une seule fois.

Théorème

Déterminer si un graphe est Hamiltonien est un problème NP-Complet.

Sommaire

Représentation de graphes

Deux méthodes principales :

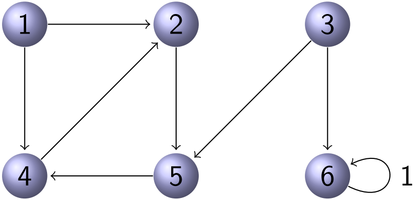
- listes d'adjacences
  - graphe peu dense
  - économe en mémoire

Représentation de graphes

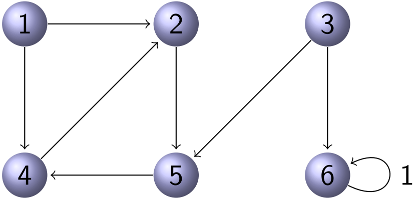
Deux méthodes principales :

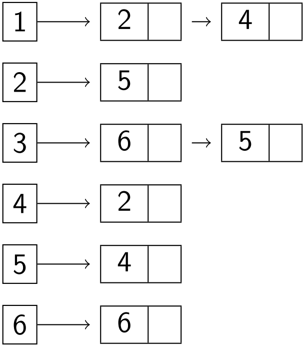
- listes d'adjacences
  - graphe peu dense
  - économe en mémoire
- matrice d'adjacences
  - graphe dense
  - pratique pour tester si $(a, b) \in A$

Exemple

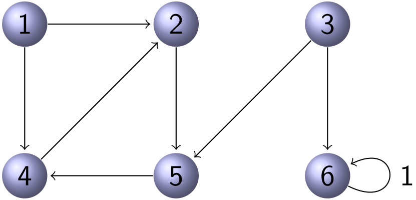


Exemple





Exemple



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Sommaire

1. Recherche opérationnelle
   - Notions sur la complexité
   - Classes de complexité
2. Graphes : généralités
   - Historique
   - Définitions
   - Cycle dans un graphe
   - Représentation
   - Composantes fortement connexes
3. Plus court chemin dans un graphe
   - Plus court chemin à origine unique

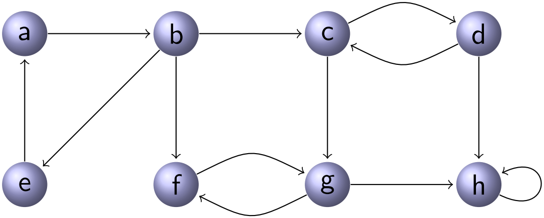
Composantes fortement connexes

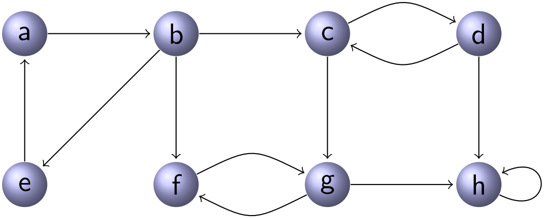
Définition

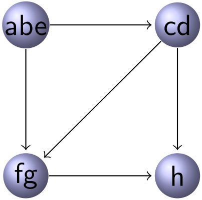
Soit R relation d'équivalence définie par xRy ssi $x \rightsquigarrow y$ et $y \rightsquigarrow x$. On appelle alors composante fortement connexe d'un graphe les classes d'équivalence de R. Un graphe fortement connexe est un graphe possédant une seule composante fortement connexe.

Utilité :

- diviser un problème en sous-problèmes


a
b
c
d
e
f
g
h


a
b
c
d
e
f
g
h


abe
cd
fg
h

Recherche de composantes fortement connexes

Définition (transposée d'un graphe)

On appelle transposée d'un graphe $G = (S, A)$, et on note $G^T = (S, A^T)$. $A^T = \{(u, v)/(v, u) \in A\}$

Pour l'algorithme de recherche, on note :

- *ncfc* : nombre de composantes fortement connexes
- *nc* : numero de composante fortement connexe de chaque sommet

Remarque

- algorithme peu efficace : $\Theta(|S| \times |A|)$
- algorithme de Tarjan : $\Theta(\max(|S|, |A|)$

Algorithme

Algorithme : Composantes fortement connexes : cfc(Graphe G)

Données : *ncfc* entier : nb composantes fortement connexes
Données : *nc* tableau d'entiers : n° de CFC

$ncfc \leftarrow 0$;

pour tous $i \in [1, |S|]$ **faire**
  $nc[i] \leftarrow 0$;

Algorithme

Algorithme : Composantes fortement connexes : cfc(Graphe G)

Données : *ncfc* entier : nb composantes fortement connexes
Données : *nc* tableau d'entiers : n° de CFC

$ncfc \leftarrow 0$;
pour tous $i \in [1, |S|]$ **faire**
  $nc[i] \leftarrow 0$;

pour tous $i \in [1, |S|]$ **faire**
  si $nc[i] = 0$ **alors**

Algorithme

Algorithme : Composantes fortement connexes : cfc(Graphe G)

Données : *ncfc* entier : nb composantes fortement connexes
Données : *nc* tableau d'entiers : n° de CFC

$ncfc \leftarrow 0$;
pour tous $i \in [1, |S|]$ **faire**
    $nc[i] \leftarrow 0$;
pour tous $i \in [1, |S|]$ **faire**
    si $nc[i] = 0$ **alors**
        dfs(G, s_i) ;
        dfs(G^T, s_i) ;
        $ncfc \leftarrow ncfc + 1$;

Algorithme

Algorithme : Composantes fortement connexes : cfc(Graphe G)

Données : *ncfc* entier : nb composantes fortement connexes

Données : *nc* tableau d'entiers : n° de CFC

$ncfc \leftarrow 0$;

pour tous $i \in [1, |S|]$ **faire**

  $nc[i] \leftarrow 0$;

pour tous $i \in [1, |S|]$ **faire**

  si $nc[i] = 0$ **alors**

    dfs(G, s_i) ;

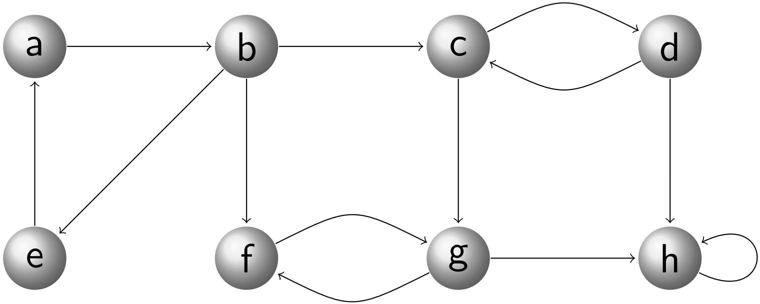
    dfs(G^T, s_i) ;

    $ncfc \leftarrow ncfc + 1$;

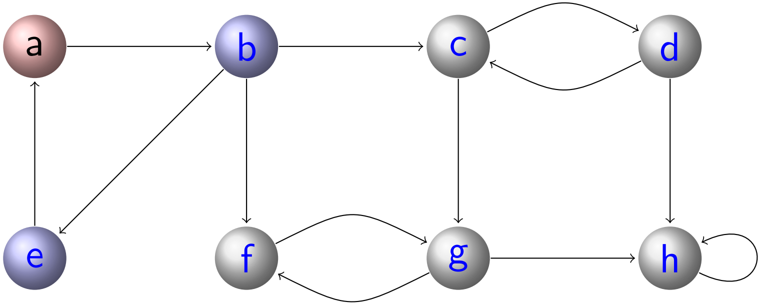
    pour tous i/s_i *marqué pour les deux explorations* **faire**

      $nc[i] \leftarrow ncfc$;

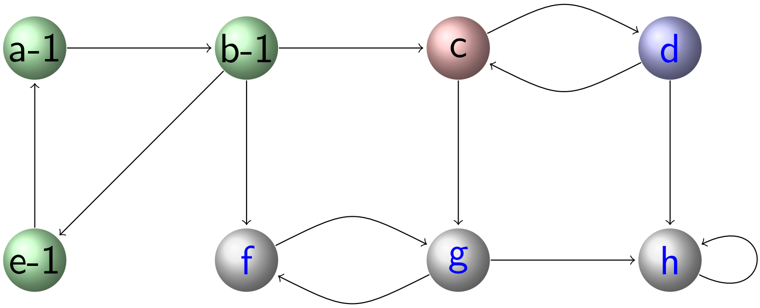
décomposition en composantes fortement connexes



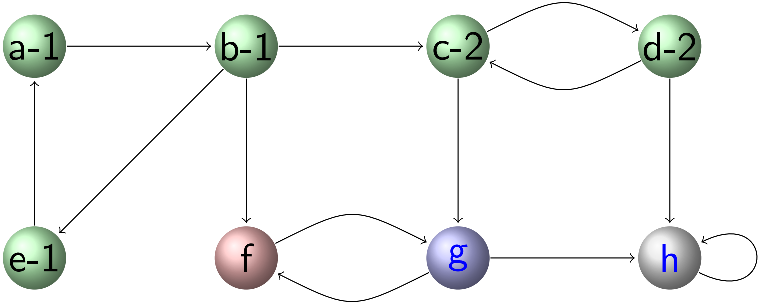
décomposition en composantes fortement connexes



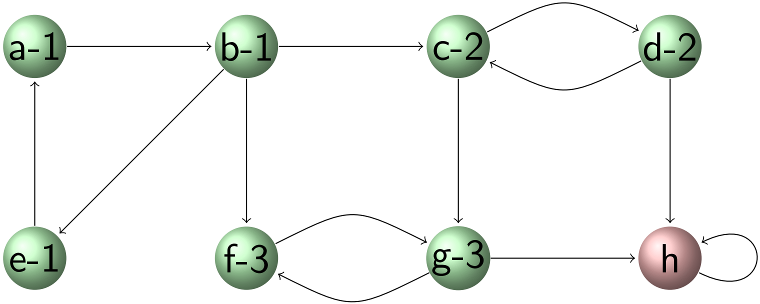
décomposition en composantes fortement connexes



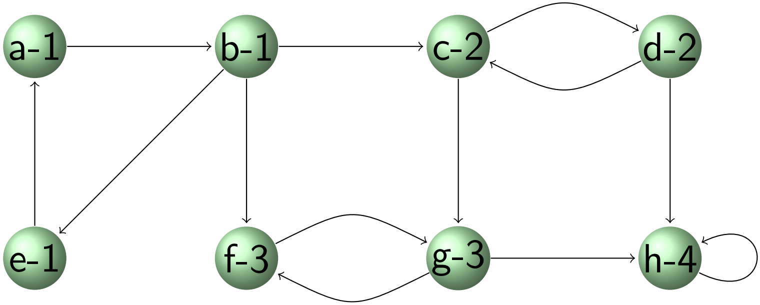
décomposition en composantes fortement connexes



décomposition en composantes fortement connexes



décomposition en composantes fortement connexes



Soit $G = (S, A)$ un graphe connexe.

Définition

On appelle sommet d'articulation de G tout sommet dont la suppression rend G non connexe.

Définition

On appelle pont ou isthme de G tout arc dont la suppression rend G non connexe.

Définition

Un graphe non orienté est k-connexe s'il reste connexe après suppression d'un ensemble quelconque de k-1 arêtes et s'il existe un ensemble de k arêtes qui déconnecte le graphe.

Sommaire

Objectif

- Déterminer la distance optimale pour aller d'un sommet à un autre dans un graphe
- Déterminer le chemin associé
- En général, notion intéressante : plus court chemin
- Graphe considéré :
  - Graphe orienté
  - Graphe pondéré

Plus court chemin

Définition

Soit $G = (S, A)$ un graphe orienté pondéré. On appelle circuit absorbant un circuit le long duquel la somme des poids des arcs est négative.

Définition

Soient $G = (S, A)$ un graphe pondéré avec une fonction de pondération $w : A \to \mathbb{R}$, et $(a, b) \in S \times S$. Soit $p = (a_0, ..., a_n)$ un chemin. On appelle coût du chemin p la somme des coûts des arcs le composant.

Sommaire

Plus court chemin à origine unique

- Distance et chemin d'un sommet à tous les autres

Plus court chemin à origine unique

- Distance et chemin d'un sommet à tous les autres
- représentation du graphe par listes d'adjacences

Plus court chemin à origine unique

- Distance et chemin d'un sommet à tous les autres
- représentation du graphe par listes d'adjacences
- sommet de départ s

Plus court chemin à origine unique

- Distance et chemin d'un sommet à tous les autres
- représentation du graphe par listes d'adjacences
- sommet de départ s
- distance de s à a : $d[a]$

Plus court chemin à origine unique

- Distance et chemin d'un sommet à tous les autres
- représentation du graphe par listes d'adjacences
- sommet de départ s
- distance de s à a : $d[a]$
- prédécesseur de a pour le plus court chemin : $\pi[a]$

Relâchement

Relâchement : mise à jour de d et π

Algorithme : Relacher(sommet a, sommet b)

si $d[b] > d[a] + \omega(a, b)$ **alors**
  $d[b] \leftarrow d[a] + \omega(a, b)$;
  $\pi[b] \leftarrow a$;

Propriétés

Inégalité triangulaire : $\forall (a,b) \in A,\ \delta(s,b) \leq \delta(s,a) + \omega(a,b)$.

Propriétés

Inégalité triangulaire : $\forall(a, b) \in A,\ \delta(s, b) \leq \delta(s, a) + \omega(a, b).$

Majorant : $\forall a \in S, d[a] \geq \delta(s, a).\ d[a] = \delta(s, a) \Rightarrow$ stable

Propriétés

Inégalité triangulaire : $\forall (a,b) \in A,\ \delta(s,b) \leq \delta(s,a) + \omega(a,b)$.

Majorant : $\forall a \in S, d[a] \geq \delta(s,a)$. $d[a] = \delta(s,a) \Rightarrow$ stable

Aucun-chemin : $s \not\rightsquigarrow a \Rightarrow d[a] = \delta(s,a) = +\infty$

Propriétés

Inégalité triangulaire : $\forall (a, b) \in A,\ \delta(s, b) \leq \delta(s, a) + \omega(a, b)$.

Majorant : $\forall a \in S, d[a] \geq \delta(s, a)$. $d[a] = \delta(s, a) \Rightarrow$ stable

Aucun-chemin : $s \not\rightsquigarrow a \Rightarrow d[a] = \delta(s, a) = +\infty$

Convergence : $(a, b) \in A$, $\Upsilon = (s, ..., a)$ PCC. Si $d[a] = \delta(s, a)$ avant de relâcher $(a, b) \Rightarrow d[b] = \delta(s, b)$ après.

Propriétés

Inégalité triangulaire : $\forall (a, b) \in A,\ \delta(s, b) \leq \delta(s, a) + \omega(a, b)$.

Majorant : $\forall a \in S, d[a] \geq \delta(s, a)$. $d[a] = \delta(s, a) \Rightarrow$ stable

Aucun-chemin : $s \not\rightsquigarrow a \Rightarrow d[a] = \delta(s, a) = +\infty$

Convergence : $(a, b) \in A$, $\Upsilon = (s, ..., a)$ PCC. Si $d[a] = \delta(s, a)$ avant de relâcher $(a, b) \Rightarrow d[b] = \delta(s, b)$ après.

Relâchement de chemin : $\Upsilon = (s_0, ..., s_n)$ plus court chemin. Relâchement dans l'ordre $\Rightarrow d[b] = \delta(a, b)$.

Propriétés

Inégalité triangulaire : $\forall (a,b) \in A,\ \delta(s,b) \leq \delta(s,a) + \omega(a,b)$.

Majorant : $\forall a \in S, d[a] \geq \delta(s,a)$. $d[a] = \delta(s,a) \Rightarrow$ stable

Aucun-chemin : $s \not\leadsto a \Rightarrow d[a] = \delta(s,a) = +\infty$

Convergence : $(a,b) \in A$, $\Upsilon = (s, ..., a)$ PCC. Si $d[a] = \delta(s,a)$ avant de relâcher $(a,b) \Rightarrow d[b] = \delta(s,b)$ après.

Relâchement de chemin : $\Upsilon = (s_0, ..., s_n)$ plus court chemin. Relâchement dans l'ordre $\Rightarrow d[b] = \delta(a,b)$.

Sous graphe prédécesseur : $d[a] = \delta(s,a) \Rightarrow G_\pi$ arborescence de plus courts chemins.

Algorithme de Bellman-Ford

Algorithme : Bellmanford(graphe G, sommet s)

init(G,s) ;
pour $i \in [1, |S| - 1]$ **faire**
  pour tous $(a, b) \in A$ **faire**
    relacher(a, b) ;

Algorithme de Bellman-Ford

Algorithme : Bellmanford(graphe G, sommet s)

init(G,s) ;
pour $i \in [1, |S| - 1]$ **faire**
    pour tous $(a, b) \in A$ **faire**
        relacher(a, b) ;
pour tous $(a, b) \in A$ **faire**
    si $d[b] > d[a] + \omega(a, b)$ **alors**
        retourner FAUX ;
retourner VRAI ;

Algorithme de Bellman-Ford

- Utilisation : plus court chemin à origine unique dans le cas général où les arcs peuvent avoir des poids négatifs.
- Retourne VRAI si pas de cycle absorbant.
- Complexité : $\Theta(|S|.|A|)$.

Algorithme de Bellman-Ford

- Utilisation : plus court chemin à origine unique dans le cas général où les arcs peuvent avoir des poids négatifs.
- Retourne VRAI si pas de cycle absorbant.
- Complexité : $\Theta(|S|.|A|)$.

Théorème (Validité de Bellman-Ford)

Après l'algorithme,

- $\forall a \in S, d[a] = \delta(s, a)$
- G_π *arborescence de plus courts chemins*

Algorithme de Dijkstra

- Algorithme *Bellman-Ford* : très général

Algorithme de Dijkstra

- Algorithme *Bellman-Ford* : très général

$\Rightarrow$ possibilités d'optimisation

Algorithme de Dijkstra

- Algorithme *Bellman-Ford* : très général

$\Rightarrow$ possibilités d'optimisation

- Hypothèse supplémentaire : arcs de poids positifs

Algorithme de Dijkstra

- Algorithme *Bellman-Ford* : très général

$\Rightarrow$ possibilités d'optimisation

- Hypothèse supplémentaire : arcs de poids positifs
- Algorithme de *Edsger Dijkstra*

Algorithme de Dijkstra

- Algorithme *Bellman-Ford* : très général

$\Rightarrow$ possibilités d'optimisation

- Hypothèse supplémentaire : arcs de poids positifs
- Algorithme de *Edsger Dijkstra* [ˈɛtˌsxər ˈdɛɪkˌstra]

Algorithme de Dijkstra

- Algorithme *Bellman-Ford* : très général

$\Rightarrow$ possibilités d'optimisation

- Hypothèse supplémentaire : arcs de poids positifs
- Algorithme de *Edsger Dijkstra* [ˈɛtˌsxər ˈdɛɪkˌstra]
  - Mathématicien/informaticien hollandais
  - Travaux en théorie des graphes
  - Travaux sur la programmation concurrente
  - "A Case against the GO TO Statement"
  - Prix *Turing* en 1972

Algorithme de Dijkstra

Algorithme : Dijkstra(graphe G, sommet s)

init(G,s) ;
$F \leftarrow S$;

Algorithme de Dijkstra

**Algorithme : ** Dijkstra(graphe G, sommet s)

init(G,s) ;
$F \leftarrow S$;
tant que $F \neq \emptyset$ **faire**
    $a \leftarrow$ extraireMin(F) ;
    pour tous $b \in succ(a)$ **faire**
        relacher(a, b) ;

Étude de Dijkstra

- Utilise une file de priorité F.
- Ensemble de sommets traités E
- Invariant : $F = S - E$.
- Stratégie gloutonne : choix du sommet de $S - E$ qui a le coût le plus faible.

Étude de Dijkstra

- Utilise une file de priorité F.
- Ensemble de sommets traités E
- Invariant : $F = S - E$.
- Stratégie gloutonne : choix du sommet de $S - E$ qui a le coût le plus faible.
- Complexité dépend de l'implantation de F.

Étude de Dijkstra

- Utilise une file de priorité F.
- Ensemble de sommets traités E
- Invariant : $F = S - E$.
- Stratégie gloutonne : choix du sommet de $S - E$ qui a le coût le plus faible.
- Complexité dépend de l'implantation de F.
- $\Theta(|S|^2)$ si F est un tableau de n cases.

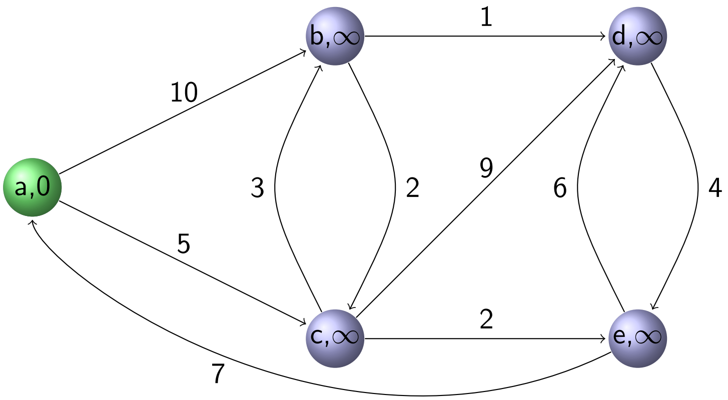
Étude de Dijkstra

- Utilise une file de priorité F.
- Ensemble de sommets traités E
- Invariant : $F = S - E$.
- Stratégie gloutonne : choix du sommet de $S - E$ qui a le coût le plus faible.
- Complexité dépend de l'implantation de F.
- $\Theta(|S|^2)$ si F est un tableau de n cases.
- $\Theta((|S| + |A|) \log |S|)$ avec un tas binomial.

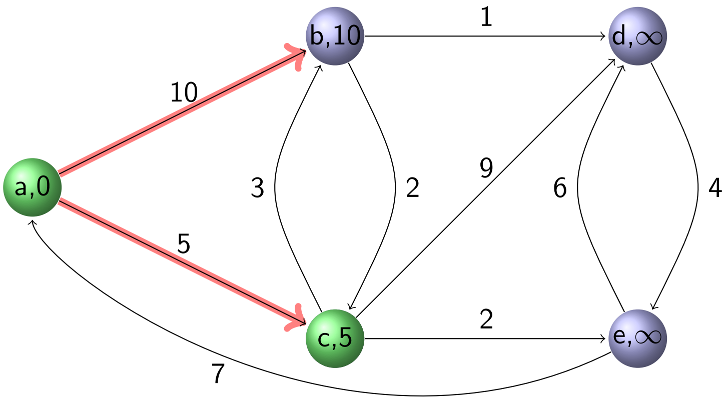
Étude de Dijkstra

- Utilise une file de priorité F.
- Ensemble de sommets traités E
- Invariant : $F = S - E$.
- Stratégie gloutonne : choix du sommet de $S - E$ qui a le coût le plus faible.
- Complexité dépend de l'implantation de F.
- $\Theta(|S|^2)$ si F est un tableau de n cases.
- $\Theta((|S| + |A|) \log |S|)$ avec un tas binomial.
- $\Theta(|S| \log |S| + |A|)$ avec un tas de Fibonacci.

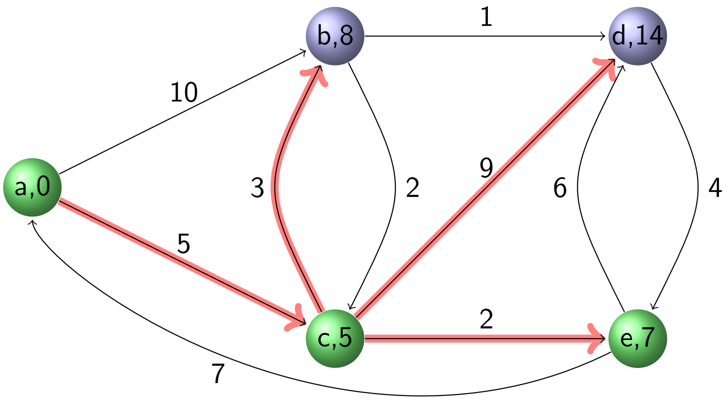
Dijkstra



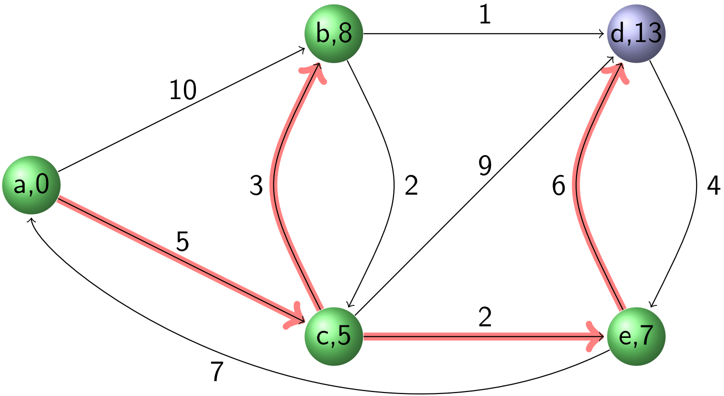
Dijkstra



Dijkstra



Dijkstra



Dijkstra