Plus court chemins dans un graphe

à la recherche du point fixe

Jean-Marc.Vincent@univ-grenoble-alpes.fr[1]

[1] Laboratoire LIG
Équipe-Projet INRIA POLARIS
Université Grenoble-Alpes

Algorithmique et Modélisation
L3-INFO

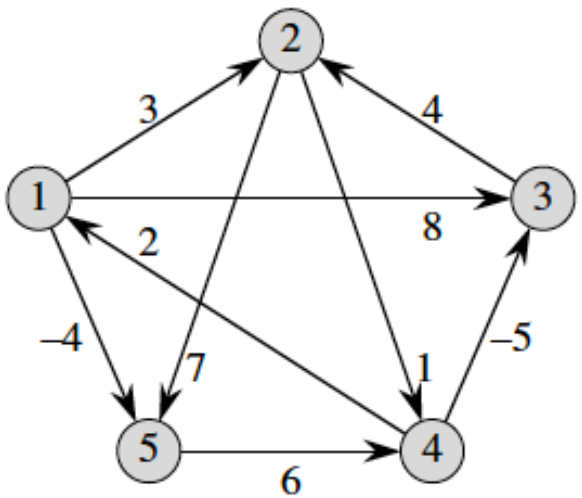
Le contexte

Soit $\mathcal{G} = (\mathcal{X}, \mathcal{A})$ un graphe orienté.
$\mathcal{X} = \{x_1, \cdots, x_n\}$ ensemble de n sommets, $|\mathcal{X}| = n$
$\mathcal{A}$ l'ensemble des arcs du graphe, on notera $|\mathcal{A}| = m$
$\mathcal{P}$ une matrice de pondération des arcs $p(x, y)$ est le poids de l'arc (x, y), s'il existe.
(On peut avoir des pondérations négatives)

Problèmes

- Pour tous couple x et y sommets du graphe, déterminer le poids minimal des chemins de ex à y.
- Donner un algorithme de construction d'un chemin de poids minimal de x à y.

Exemple (extrait du Cormen et al.)



Questions

Pour tout couple de sommet exhiber un chemin de poids minimal.

Algorithme de Floyd-Warshall

L'idée (voir les détails sur le TD) est d'utiliser une décomposition de l'ensemble des chemins en regardant les sommets intermédiaires.
Notons $a^k_{i,j}$ le poids minimal d'un chemin entre x_i et x_j dont les sommets intermédiaires sont dans le sous-ensemble de sommets $\{x_1, x_2, \cdots, x_k\}$. On a l'équation

$$a^k_{x,y} = a^{k-1}_{x,y} \oplus a^{k-1}_{x,k} \otimes a^{k-1}_{k,y}$$

avec $\oplus$ l'opérateur $\min$ et $\otimes$ l'opérateur $+$.
Cette équation est juste la formalisation du fait que le poids minimal d'un chemin dont les sommets intermédiaires sont dans le sous-ensemble de sommets $\{x_1, x_2, \cdots, x_k\}$ est le minimum entre

(a) le poids minimal d'un chemin de x_i à x_j dont les sommets intermédiaires sont dans $\{x_1, x_2, \cdots, x_{k-1}\}$

et

(b) le poids minimal d'un chemin de x_i à x_k plus le poids minimal d'un chemin de x_k à x_j, chemins ne passant que par des sommets dans $\{x_1, x_2, \cdots, x_{k-1}\}$.

Questions

Traduire cette idée en un algorithme, on pourra réutiliser la même matrice pour calculer les coefficients $a^k_{x,y}$ (expliquer pourquoi)

```
Algorithme_de_Floyd-Warshall (A)
  Données: A matrice d'adjacence des coûts n × n
  Résultat: matrice des coûts minimum A*
  A* matrice n × n initialisée à A et des 0 sur la diagonale
  for k = 1 to n
      foreach (i, j) ∈ {1, ··· , n}^2 do
          a*_{i,j} = a*_{i,j} ⊕ (a*_{i,k} ⊗ a*_{k,j})
  Return A*
```

Questions

Exécuter cet algorithme sur l'exemple

Exécution de l'algorithme

Algorithme

```
Algorithme_de_Floyd-Warshall (A)
  Données: A matrice d'adjacence des coûts n × n
  Résultat: matrice des coûts minimum A*
  A* matrice n × n initialisée à A et des 0 sur la diagonale
  for k = 1 to n
      // point d'observation de A* et k
      foreach (i, j) ∈ {1, ··· , n}² do
          A*_{i,j} = A*_{i,j} ⊕ (A*_{i,k} ⊗ A*_{k,j})
  Return A*
```

On observe le déroulement du programme au point d'observation
Initialement

$$A = \begin{bmatrix} \infty & 3 & 8 & \infty & -4 \\ \infty & \infty & \infty & 1 & 7 \\ \infty & 4 & \infty & \infty & \infty \\ 2 & \infty & -5 & \infty & \infty \\ \infty & \infty & \infty & 6 & \infty \end{bmatrix}$$

$$A(1) = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$A(2) = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

Trace d'exécution

$$A(3) = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$A(4) = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$A(5) = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

$$A^* = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

UGA Université Grenoble Alpes

Extensions

Questions

- Comment détecter les éventuels circuits absorbants (de poids négatif) ?
- Que modifier dans l'algorithme pour obtenir l'existence d'un chemin ?
- Adapter l'algorithme pour pouvoir associer à chaque couple de sommet un chemin de poids minimum (construction de la matrice de routage)

EXTENSIONS

- Comment détecter les éventuels circuits absorbants (de poids négatif) ?

  En fait dès que l'on boucle un circuit absorbant, le sommet x qui "ferme" la boucle aura une valeur $a^*_{x,x}$ strictement négative, donc le graphe ne contiendra aucun circuit absorbant lorsque tous les éléments diagonaux de la matrice sont 0.

- Que modifier dans l'algorithme pour obtenir l'existence d'un chemin ?

  Facile, on remplace la matrice des poids par la matrice booléenne d'adjacence, au lieu de faire un $\min$ on fait un *ou* et au lieu du $+$ on fait un et.

- Adapter l'algorithme pour pouvoir associer à chaque couple de sommet un chemin de poids minimum (construction de la matrice de routage)

  Comme dans l'algorithme de Dantzig, on construit la matrice de routage dans l'algorithme, lorsque l'on actualise $A^*_{i,j}$.

LIMITES DES ALGORITHMES

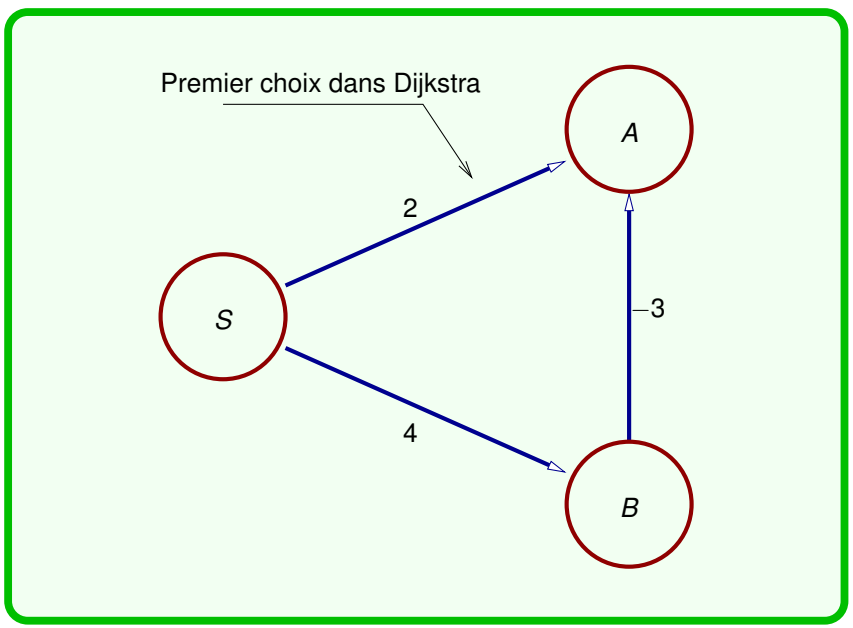
On considère le problème restreint aux chemins d'origine fixée *s* avec des **pondérations** qui peuvent être **négatives**

Questions

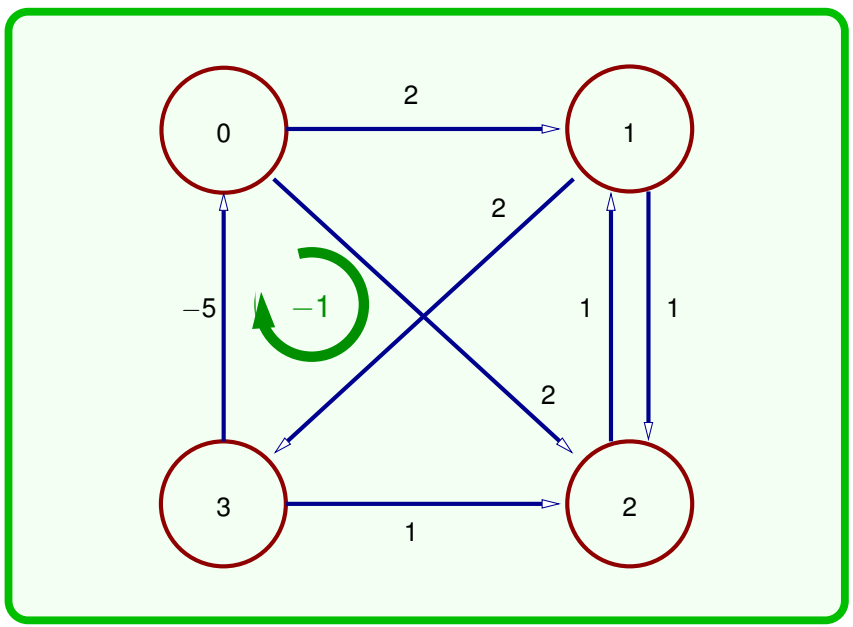
- Donner un exemple de graphe pondéré (avec certaines pondérations négatives) tel que l'exécution de l'algorithme de Dijkstra sur ce graphe ne donne pas un résultat correct.
- Donner un exemple de graphe pondéré (avec certaines pondérations négatives) pour lequel il n'existe pas de chemin de poids minimal entre *s* et un autre sommet accessible à partir de *s*.

LIMITES DES ALGORITHMES

- Donner un exemple de graphe pondéré (avec certaines pondérations négatives) tel que l'exécution de l'algorithme de Dijkstra sur ce graphe ne donne pas un résultat correct.



- Donner un exemple de graphe pondéré (avec certaines pondérations négatives) pour lequel il n'existe pas de chemin de poids minimal entre *s* et un autre sommet accessible à partir de *s*.



POINT FIXE

On suppose qu'il existe un chemin de poids minimal entre s à chacun des autres sommets du graphe (il n'y a pas de circuit de poids négatif, un tel circuit est dit **absorbant**), on note $d^{\min}(x)$ le poids minimal d'un chemin de s à x.

Questions

Démontrer que les variables $d^{\min}(.)$ vérifient l'équation de point fixe

$$d^{\min}(y) = \min_{(x,y)\in\mathcal{A}} \left\{ d^{\min}(x) + p(x,y) \right\} \text{ pour tout } y \in \mathcal{X}\backslash\{s\} \text{ avec } d^{\min}(s) = 0. \tag{1}$$

Indication : on pourra d'abord démontrer que $d^{\min}(y) \leqslant d^{\min}(x) + p(x,y)$ pour tout arc (x,y), puis montrer qu'il existe un arc (z,y) tel que $d^{\min}(y) = d^{\min}(z) + p(z,y)$.

- D'abord, s'il n'y a pas de circuit absorbant, alors s'il existe un chemin entre s et x, il en existe un de poids minimum, de plus ce chemin est élémentaire (un chemin contenant un circuit est de poids plus grand que le même chemin auquel on a enlevé ce circuit, donc un chemin de poids minimal sera sans circuit c'est-à-dire élémentaire, les chemins élémentaires de s à x sont en nombre fini donc il en existe un plus petit)
- Soit un chemin $s \rightsquigarrow x$ de poids minimum $d^{min}(x)$ et soit l'arête $x \to y$, alors $s \rightsquigarrow x \to y$ est un chemin de s à y passant par x de poids $d^{min}(x) + p(x, y)$ d'où

  $$d^{\min}(y) \leqslant d^{\min}(x) + p(x, y) \text{ car } d^{min}(y) \text{ est le poids min des chemins } s \rightsquigarrow y$$

  Comme c'est vérifié pour tout x tel que l'arête (x, y) existe

  $$d^{\min}(y) \leqslant \min_{(x,y)\in\mathcal{A}} \left\{ d^{\min}(x) + p(x, y) \right\}$$

- Soit un chemin $s \rightsquigarrow y$ de poids minimum $d_{min}(y)$. Soit z le dernier sommet sur ce chemin avant d'arriver à y et soit $d(z)$ la longueur de ce chemin de s à z. on obtient ainsi

  $$\begin{aligned} d^{min}(y) &= d(z) + p(z, y) \\ &\geqslant d^{min}(z) + p(z, y) \text{ car } d(z) \geqslant d^{min}(z) \\ &\geqslant \min_{(x,y)\in\mathcal{A}} \left\{ d^{\min}(x) + p(x, y) \right\} \end{aligned}$$

  On prouve ainsi l'équation de point fixe et l'existence d'un z tel que $d^{min}(y) = d^{min}(z) + p(z, y)$

ALGORITHME

Algorithme `CalculePoidsMin` ($\mathcal{G}$,s)

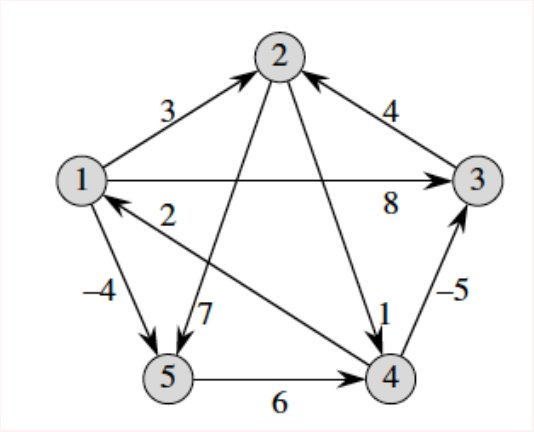
Données: Graphe pondéré $\mathcal{G} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$,
$p(x, y)$ étant le poids de l'arc (x, y)

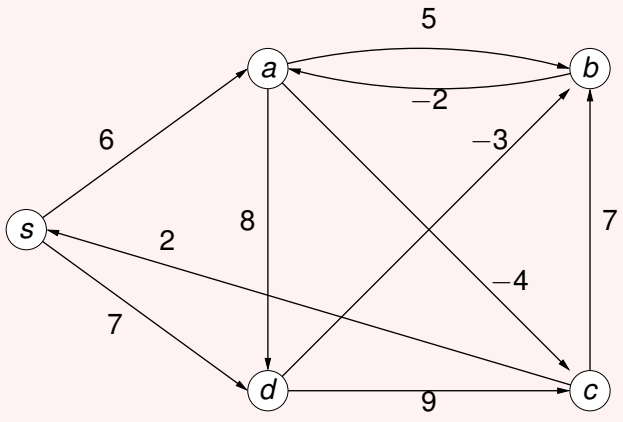
Résultat: Renvoie $d^{\min}(.)$

foreach $x \in \mathcal{X} \backslash \{s\}$ **do** $d(x) = +\infty$
$d(s) = 0$
while **il existe un arc** (x, y) **tel que** $d(y) > d(x) + p(x, y)$ **do**
  $d(y) = d(x) + p(x, y)$
`Retourne` d

Questions

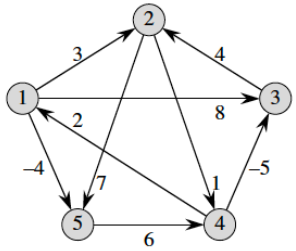
Exécuter l'algorithme sur le graphe d'exemple ($s = 1$) puis sur le graphe ci-dessous, on notera la séquence des arcs choisis et la suite des vecteurs d obtenus à la fin de chaque itération.

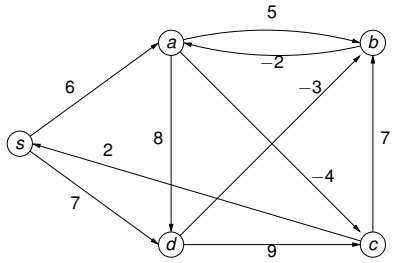




EXÉCUTION

Algorithme `CalculePoidsMin` ($\mathcal{G}$,s)
  foreach $x \in \mathcal{X} \backslash \{s\}$ **do** $d(x) = +\infty$
  $d(s) = 0$
  while **il existe un arc** (x, y) **tel que** $d(y) > d(x) + p(x, y)$ **do**
    $d(y) = d(x) + p(x, y)$
    `// point d'observation`
  `Retourne` d





	arc	d
	/	$[0, \infty, \infty, \infty, \infty]$
1	$(0, 1, 3)$	$[0, 3, \infty, \infty, \infty]$
2	$(0, 2, 8)$	$[0, 3, 8, \infty, \infty]$
3	$(1, 3, 1)$	$[0, 3, 8, 4, \infty]$
4	$(1, 4, 7)$	$[0, 3, 8, 4, 10]$
5	$(3, 2, -5)$	$[0, 3, -1, 4, 10]$
6	$(0, 4, -4)$	$[0, 3, -1, 4, -4]$
7	$(4, 3, 6)$	$[0, 3, -1, 2, -4]$
8	$(3, 2, -5)$	$[0, 3, -3, 2, -4]$
9	$(2, 1, 4)$	$[0, 1, -3, 2, -4]$

	arc	d
	/	$[0, \infty, \infty, \infty, \infty]$
1	$(0, 1, 6)$	$[0, 6, \infty, \infty, \infty]$
2	$(0, 4, 7)$	$[0, 6, \infty, \infty, 7]$
3	$(1, 2, 5)$	$[0, 6, 11, \infty, 7]$
4	$(1, 3, -4)$	$[0, 6, 11, 2, 7]$
5	$(3, 2, 7)$	$[0, 6, 9, 2, 7]$
6	$(4, 2, -3)$	$[0, 6, 4, 2, 7]$
7	$(2, 1, -2)$	$[0, 2, 4, 2, 7]$
8	$(1, 3, -4)$	$[0, 2, 4, -2, 7]$

PREUVE

- Si l'algorithme termine, démontrer que le vecteur d retourné est un point fixe de l'équation ci-dessus.

  Lorsque l'algorithme se termine alors il n'existe pas d'arc (x, y) tel que $d(y) > d(x) + p(x, y)$ (condition de sortie de boucle). Comme le vecteur d est actualisé, en décroissant, avec une affectation de type $d(z) = d(x) + p(x, z)$ il existe un z tel que l'on ait l'égalité.
  Donc d est point fixe.

- Démontrer que l'algorithme se termine. *Indication : trouver un variant de l'itération et utiliser le fait que le nombre de chemins "potentiellement de poids minimal" est fini*.

  En fait, il faut jouer sur 2 variants : le nombre de sommets non-atteints ($d(x) = \infty$) et la somme des $d(\dot{})$ pour les sommets atteints.
  Ce couple est strictement décroissant pour l'ordre lexicographique, il est dans un ensemble discret et fini donc l'algorithme s'arrête.

Preuve

- Conclure

L'algorithme s'arrête et le vecteur d est un point fixe avec $d(s) = 0$ il reste à montrer que $d = d^{\min}$. Comme d est construit par des choix successifs d'arêtes $d(x)$ est la longueur d'un chemin $s \rightsquigarrow x$. Donc pour tout x, $d(x) \geqslant d^{\min}(x)$.
Faisons un raisonnement par l'absurde. Supposons $d(x) > d^{\min}(x)$. Considérons un chemin de poids minimal $s =, x_1, \cdots, x_k = x$ le chemin de s à x de poids $d(x)$. Sur ce chemin $d^{\min}(x_i) - d^{\min}(x_{i-1}) = p(x_{i-1}, x_i)$. Il existe un premier sommet x_i tel que $d(x_i) > d^{\min}(x_i)$ comme l'arc a un poids $p(x, y)$ il existe un couple de sommet (x_{i-1}, x_i) tel que $d(x_i) > d^{\min}(x_{i-1}) + p((x_{i-1}, x_i) = d(x_{i-1}) + (x_{i-1}, x_i)$ ce qui contredit la condition de sortie de la boucle.
Donc d est le vecteur des poids minimum des chemins du graphe d'origine s

ROUTAGE

- Modifier l'algorithme afin d'avoir le routage, c'est à dire, une méthode qui, pour chaque sommet x, renvoie un chemin de poids minimal s à x

  Indication : utiliser une variable pere(v).

Algorithme `CalculePoidsMin` ($\mathcal{G}$,s)

Données: Graphe pondéré $\mathcal{G} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$,
$p(x, y)$ étant le poids de l'arc (x, y)

Résultat: Renvoie $d^{\min}(.)$

foreach $x \in \mathcal{X} \setminus \{s\}$ **do**
  $d(x) = +\infty$
  $pere(x) = /$

$d(s) = 0$

while **il existe un arc** (x, y) **tel que** $d(y) > d(x) + p(x, y)$ **do**
  $d(y) = d(x) + p(x, y)$
  $pere(y) = x$

`Retourne` d

AMÉLIORATIONS

*On remplace les lignes 3 et 4 (boucle **While**) de l'algorithme par*

repeat $|X| - 1$ **fois**
  foreach $(x, y) \in \mathcal{A}$ **do**
    $d(y) = \min\{d(y), d(x) + p(x, y)\}$

Questions

- Justifier la correction de ce nouvel algorithme
- S'il n'y a pas de circuit absorbant, montrer que le coût de cet algorithme se réduit à $|\mathcal{X}|.|\mathcal{A}|$.
- Comparer cet algorithme avec l'algorithme de Dijkstra (on se donnera au préalable des critères de comparaison).
- Que se passe-t-il si le graphe possède un circuit absorbant ? Modifier l'algorithme pour qu'il détecte s'il existe un circuit absorbant ou renvoie le $d^{\min}(.)$.
- Transformer cet algorithme pour calculer les chemins de toute origine vers toute destination.

AMÉLIORATIONS

*On remplace les lignes 3 et 4 (boucle **While**) de l'algorithme par*

for $i = 1$ **to** $|X| - 1$
  foreach $(x, y) \in \mathcal{A}$ **do**
    $d(y) = \min\{d(y), d(x) + p(x, y)\}$

- Justifier la correction de ce nouvel algorithme

  À condition que le graphe n'ait pas de circuit absorbant, les chemins de poids min sont élémentaires. Comme un chemin élémentaire contient $|X| - 1$ arêtes, on a au maximum $|X| - 1$ actualisations si on s'y prend bien.
  Au vu de la forme de l'itération et en reprenant ce qui a été dit dans la première partie du cours en décomposant l'ensemble des chemins par leur longueur, on remarque que le vecteur d contient le poids min des chemins de longueur i issu de s.

- S'il n'y a pas de circuit absorbant, montrer que le coût de cet algorithme se réduit à $|\mathcal{X}|.|\mathcal{A}|$.

  Le coût de l'itération interne est le nombre d'arcs $|\mathcal{A}|$ et il est répété $|\mathcal{X}| - 1$, en ajoutant le coût de l'initialisation on a un coût de $|\mathcal{X}|.|\mathcal{A}|$, c'est à dire $\mathcal{O}(n^3)$ (au pire si le nombre des arc est de l'ordre de n^2).

- Comparer cet algorithme avec l'algorithme de Dijkstra (on se donnera au préalable des critères de comparaison).

  L'algorithme de Dijkstra utilise une structure auxiliaire (file à priorité) de taille n. Si l'on optimise la recherche d'un prioritaire on peut exécuter l'algorithme de Dijkstra en $\mathcal{O}(|\mathcal{X}| + |\mathcal{A}| \log |\mathcal{X}|)$.

- Que se passe-t-il si le graphe possède un circuit absorbant ? Modifier l'algorithme pour qu'il détecte s'il existe un circuit absorbant ou renvoie le $d^{\min}(.)$.

  Comme pour l'algorithme de Floyd-Warshall, Dantzig ou le produit de matrice un circuit absorbant apparait lorsqu'un terme diagonal devient strictement négatif.

- Transformer cet algorithme pour calculer les chemins de toute origine vers toute destination.

  Il suffit de construire la matrice des poids min $D = ((d_{i,j}))$ avec $d_{i,j}$ le poids min d'un chemin $i \rightsquigarrow j$. Cela revient à insérer une itération dans le bloc interne de l'algorithme.

- Algorithme de calcul de la matrice des poids min et de leurs chemins

Algorithme `CalculeMatricePoidsMin` ($\mathcal{G}$,s)

Données: Graphe pondéré $\mathcal{G} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$,
$p_{x,y}$ étant le poids de l'arc (x, y) s'il existe

Résultat: Renvoie la matrice des poids minimaux $D^{\min}$ et la matrice *Pere*

`// Initialisation des matrices` D `et` *Pere*

foreach $(s, y) \in \mathcal{X}^2$ $s \neq y$ **do** $d^{min}_{s,y} = +\infty$; $Pere_{s,y} = /$

foreach $s \in \mathcal{X}$ **do** $d^{min}_{s,s} = 0$; $Pere_{s,s} = s$

Do $|X| - 1$ **fois** // Itération sur la longueur des chemins

  foreach $(x, y) \in \mathcal{A}$ **do** // Itération sur les arcs

    foreach $s \in \mathcal{X}$ **do** // Itération sur les sources

      if $d_{s,y} > d_{s,x} + p_{x,y}$

        $d_{s,y} = d_{s,x} + p_{x,y}$ // Mise à jour du poids min

        $Pere_{s,y} = x$ // Mise à jour du routage

`Retourne` D, *Pere*

SYNTHÈSE : UNE HISTOIRE DE CHEMINS

Données

$\mathcal{G} = (\mathcal{X}, \mathcal{A}, \mathcal{V})$ graphe orienté arc-valués

- Valeur d'un chemin
  $C = x_0 \to x_1 \to x_2 \to \cdots \to x_n$

  $v(C) = v(x_0, x_1) \otimes v(x_1, x_2) \otimes \cdots \otimes v(x_{n-1}, x_n)$

- Valeur d'un ensemble $\mathcal{C}$ de chemins

  $$v(\mathcal{C}) = \bigoplus_{C \in \mathcal{C}} v(C)$$

SYNTHÈSE : UNE HISTOIRE DE CHEMINS

Données

$\mathcal{G} = (\mathcal{X}, \mathcal{A}, \mathcal{V})$ graphe orienté arc-valués

- Valeur d'un chemin
  $C = x_0 \to x_1 \to x_2 \to \cdots \to x_n$

  $v(C) = v(x_0, x_1) \otimes v(x_1, x_2) \otimes \cdots \otimes v(x_{n-1}, x_n)$

- Valeur d'un ensemble $\mathcal{C}$ de chemins

  $$v(\mathcal{C}) = \bigoplus_{C \in \mathcal{C}} v(C)$$

Problèmes

i. chemins d'origine s vers la destination d

ii. chemins d'origine s vers toutes les destinations

iii. chemins de toute origine vers toute destination

Synthèse : une histoire de chemins

Données

$\mathcal{G} = (\mathcal{X}, \mathcal{A}, \mathcal{V})$ graphe orienté arc-valués

- Valeur d'un chemin
  $C = x_0 \to x_1 \to x_2 \to \cdots \to x_n$

  $v(C) = v(x_0, x_1) \otimes v(x_1, x_2) \otimes \cdots \otimes v(x_{n-1}, x_n)$
- Valeur d'un ensemble $\mathcal{C}$ de chemins

  $$v(\mathcal{C}) = \bigoplus_{C \in \mathcal{C}} v(C)$$

Problèmes

1. chemins d'origine s vers la destination d
2. chemins d'origine s vers toutes les destinations
3. chemins de toute origine vers toute destination

Algorithmes

- Algorithme de Dijkstra : (ii) puis (i)
- Algorithme $A^\star$: (i) avec une heuristique
- Algorithme de Dantzig : (iii) augmente $\mathcal{X}$
- Algorithme de Floyd-Warshall : (iii) augmente l'ensemble des sommets intermédiaires
- Algorithme de Bellman : (ii) et (iii) point fixe
- Diviser pour régner : (iii) diviser pour régner

Généralisation

v	$\oplus$	$\otimes$	Sémantique
booléen	ou	et	existence d'un chemin
$\{0, 1\}$	$+$	min	nombre de chemins
réel $\geqslant 0$	min	$+$	chemin de valeur minimale
réel $\geqslant 0$	max	$+$	chemin de valeur maximale (graphe sans circuit)
réel $\geqslant 0$	max	*min*	chemin de capacité maximale
réel $\geqslant 0$	$+$	$\times$	chaînes de Markov

UGA Université Grenoble Alpes