Plus courts chemins

Plan

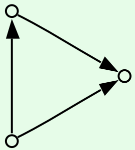
Plan

Graphe orienté

Définition

Un graphe orienté est un couple $G = (V, A)$ où

- V est un ensemble fini
- A est un ensemble de couples d'éléments de V



couple ordonné : $uv \neq vu$[1]

A est l'ensemble des **arcs** du graphe

1. Formellement l'arc uv devrait s'écrire (u, v)

Graphe orienté

Exercice

Soit $G = (V, E)$ un graphe non orienté. En orientant toutes les arêtes de G, combien de graphes différents peut-on créer ?

Graphe orienté

Définitions

Pour un sommet $u \in V$

- $d^-(u) = |\{v / vu \in A\}|$ est le **degré entrant** de u [*in-degree*]
- $d^+(u) = |\{v / uv \in A\}|$ est le **degré sortant** de u [*out-degree*]

Graphe orienté

Définitions

Pour un sommet $u \in V$

- $d^-(u) = |\{v / vu \in A\}|$ est le **degré entrant** de u [*in-degree*]
- $d^+(u) = |\{v / uv \in A\}|$ est le **degré sortant** de u [*out-degree*]

- si $d^-(u) = 0$ alors u est une **source** [*source*]
- si $d^+(u) = 0$ alors u est un **puits** [*sink*]

Graphe orienté

Définitions

Pour un sommet $u \in V$

- $d^-(u) = |\{v / vu \in A\}|$ est le **degré entrant** de u [*in-degree*]
- $d^+(u) = |\{v / uv \in A\}|$ est le **degré sortant** de u [*out-degree*]

- si $d^-(u) = 0$ alors u est une **source** [*source*]
- si $d^+(u) = 0$ alors u est un **puits** [*sink*]

$$\sum_{i \in V} d^-(i) = \sum_{i \in V} d^+(i) = |A|$$

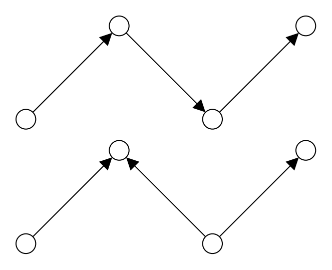
Graphe orienté

Définition

Un chemin d'un graphe orienté $G = (V, A)$ est une séquence alternée de sommets et d'arcs de la forme $(x_0, a_1, x_1, \ldots, a_k, x_k)$ où

- $x_i \in V$
- $a_i \in A$
- $a_i = x_{i-1}x_i$ pour $i = 1, \ldots, k$.

- x_0x_k-chemin (chemin de x_0 à x_k)
- si pas d'ambiguïté, on note : $(x_0, x_1, \ldots x_{k-1}, x_k)$
- les x_i ne sont pas nécessairement distincts.
- **longueur d'un chemin** = nombre d'arcs = k
- un chemin est orienté



✔ chemin

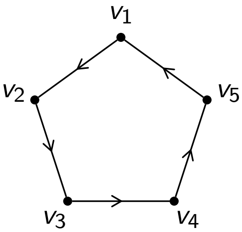
✘ pas un chemin

Graphe orienté

Définition

Un <u>circuit</u> dans un graphe orienté $G = (V, A)$ est une suite de la forme $(x_0, a_1, x_1, \ldots, a_k, x_0)$ où

- $x_i \in V$
- $a_i \in A$
- $a_i = (x_{i-1}, x_i)$ pour $i = 1, \ldots, k$.



- L'entier k est la *longueur* du circuit.

Graphe orienté

On note $x \rightsquigarrow y$ s'il existe un chemin de x à y.

La relation $\rightsquigarrow$ n'est pas symétrique.

$\exists x, y \in V$ tels que $x \rightsquigarrow y$ et $y \rightsquigarrow x$
$\Leftrightarrow$
G contient un circuit

Définition

Un graphe est <u>fortement connexe</u> si et seulement s'il existe un chemin entre chaque paire de sommets : $\forall x, y \in V, x \rightsquigarrow y$

Graphe orienté pondéré

Définition

Graphe orienté pondéré $G = (V, A, w)$:

- graphe orienté $G = (V, A)$
- muni d'une fonction de poids sur les arcs : $w : A \to \mathbb{R}$

[*weighted directed graph*]

Définition

Soit $G = (V, A, w)$ un graphe orienté pondéré.
La longueur (ou poids) d'un chemin est définie comme la somme des poids des arcs du chemin.

Plan

1. Graphes orientés
2. Plus court chemin
3. Poids unitaires : algorithme BFS
4. Poids positifs : l'algorithme de Dijkstra
5. Compléments : algorithme de Bellman (pas au programme de l'UE)

Plus courts chemins

Les problèmes de plus courts chemins $G = (V, A, w)$

(P1) Plus courts chemins entre deux sommets donnés

Soient s et t deux sommets de G. Trouver un chemin de longueur minimum entre s et t dans G.

(P2) Plus courts chemins à partir d'un sommet

Soit s un sommet de G. Pour tous les sommets v de G, trouver un chemin de longueur minimum entre s et v dans G.

(P3) Plus court chemin entre chaque paire de sommets

Pour chaque paire de sommets u, v de G, trouver un chemin de longueur minimum entre u et v dans G.

Plus courts chemins

(P2) permet de résoudre **(P1)** et **(P3)**

Pour **(P3)**, voir l'algorithme de Floyd [2]

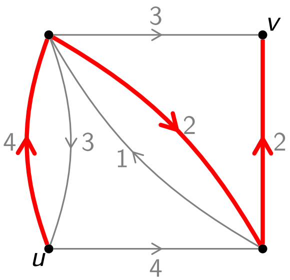
2. non étudié dans ce cours, mais on vous le fera programmer en TP

Distance

Définition

Soient u, v deux sommets dans un graphe orienté pondéré $G = (V, A, w)$. La <u>distance</u> de u à v est définie comme

$$\mathrm{dist}(u,v) = \min\{w(P) : P \text{ est un chemin de } u \text{ à } v\}$$



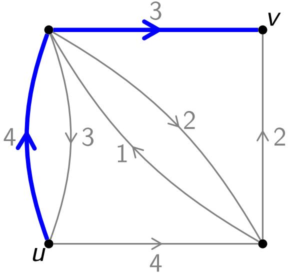
$\mathrm{dist}(u,v) \leq 8$

Distance

Définition

Soient u, v deux sommets dans un graphe orienté pondéré $G = (V, A, w)$. La distance de u à v est définie comme

$$\mathrm{dist}(u, v) = \min\{w(P) : P \text{ est un chemin de } u \text{ à } v\}$$



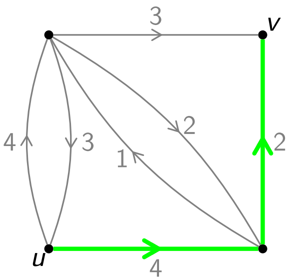
$\mathrm{dist}(u, v) \leq 7$

Distance

Définition

Soient u, v deux sommets dans un graphe orienté pondéré $G = (V, A, w)$. La <u>distance</u> de u à v est définie comme

$$\mathrm{dist}(u, v) = \min\{w(P) : P \text{ est un chemin de } u \text{ à } v\}$$



$\mathrm{dist}(u, v) \leq 6$

· et en fait $\mathrm{dist}(u, v) = 6$

Plus courts chemins

Les problèmes de plus courts chemins (reformulation) [3]

$G = (V, A, w)$

(P1) Plus courts chemins entre deux sommets donnés

Trouver la distance (et le chemin correspondant) entre s et t dans G : $\text{dist}(s, t)$.

(P2) Plus courts chemins à partir d'un sommet

Trouver la distance (et le chemin correspondant) entre s et tous les sommets de G : $\text{dist}(s, v), \quad \forall v \in V$.

(P3) Plus court chemin entre chaque paire de sommets

Trouver la distance (et le chemin correspondant) entre chaque paire de sommets de G : $\text{dist}(u, v), \quad \forall u, v \in V$.

3. "Le chemin le plus court d'un point à un autre est la ligne droite, à condition que les deux points soient bien en face l'un de l'autre." Pierre Dac

Plus courts chemins

Y a-t-il toujours un plus court chemin ?

Donnez des exemples de graphes pour lesquels, il n'y a pas de plus court chemin entre deux sommets donnés.

Plus courts chemins

Y a-t-il toujours un plus court chemin ?

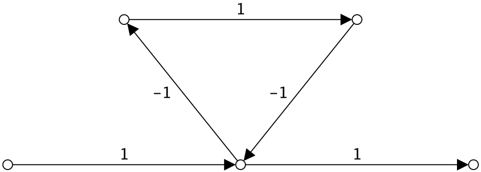
Donnez des exemples de graphes pour lesquels, il n'y a pas de plus court chemin entre deux sommets donnés.

Remarque

Il peut ne pas exister de plus court chemin de s à t :

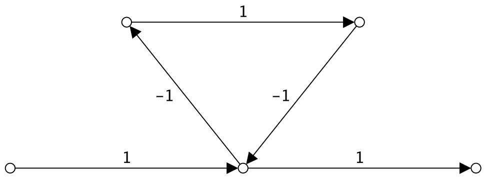
- S'il n'y a aucun chemin de s à t : $\text{dist}(s, t) = +\infty$
- S'il y a un circuit de longueur strictement négative (circuit absorbant) : $\text{dist}(s, t) = -\infty$

Plus courts chemins



Exemple de circuit absorbant

Plus courts chemins



Exemple de circuit absorbant

S'il existe un *st*-chemin, alors, si on considère seulement les chemins élémentaires entre s et t, il existe un plus court *st*-chemin élémentaire.

Plus courts chemins

Le problème des plus courts chemins est bien défini si :

- soit on cherche un chemin élémentaire
- soit les poids sont positifs
- soit le graphe ne contient pas de circuit

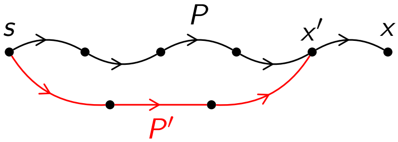
(P2) Plus courts chemins à partir d'un sommet

Trouver la distance entre s et tous les sommets de G.

Plus courts chemins

Principe de sous-optimalité

Dans un graphe orienté pondéré $G = (V, A, w)$, soit P un plus court chemin de s vers x. Alors, en notant x' le prédécesseur de x dans ce chemin, le sous-chemin de P qui va de s vers x', noté $P[s, x']$ est un plus court chemin de s vers x'.



Démonstration par l'absurde

S'il existe P' de s vers x' de poids strictement inférieur au sous-chemin de P de s vers x' alors en concaténant P' et xx' on aurait un chemin de s vers x de poids strictement inférieur à celui de P. Contradiction.

Plus courts chemins

Principe de sous-optimalité

- Plus généralement : les sous-chemins des plus courts chemins sont des plus courts chemins
- Conséquence : structure d'arborescence (arbre enraciné) des plus courts chemins

Plus courts chemins : Algorithmes

Intuition de l'algorithme

- On maintient des distances provisoires, $\lambda(v)$ (poids d'un chemin de s à v trouvé à un certain stade de l'exécution de l'algorithme)

- lorsque $\lambda(u) + w(uv) < \lambda(v)$ pour un arc uv, alors on a trouvé un meilleur chemin jusqu'à v et donc on met à jour $\lambda(v)$

Plus courts chemins : Algorithmes

Intuition de l'algorithme

- On maintient des distances provisoires, $\lambda(v)$ (poids d'un chemin de s à v trouvé à un certain stade de l'exécution de l'algorithme)
  et une arborescence π décrivant ces chemins
- lorsque $\lambda(u) + w(uv) < \lambda(v)$ pour un arc uv, alors on a trouvé un meilleur chemin jusqu'à v et donc on met à jour $\lambda(v)$ et π

Plus courts chemins : Algorithmes

- π n'est modifié que s'il y a une amélioration stricte, jamais pour changer et trouver un autre chemin de même poids
- à tout moment dans l'algorithme, $\lambda(v)$ est la longueur d'un sv-chemin existant

Plus courts chemins : Algorithmes

- π n'est modifié que s'il y a une amélioration stricte, jamais pour changer et trouver un autre chemin de même poids
- à tout moment dans l'algorithme, $\lambda(v)$ est la longueur d'un *sv*-chemin existant

$\lambda(v) \geq \mathrm{dist}(s, v)$ pour tout sommet v.

Plan

Poids unitaires

Plus courts chemins dans un graphe non pondéré

Poids unitaires : $w(a) = 1 \qquad \forall a \in A$

- L'algorithme BFS (parcours en largeur) trouve les plus courts chemins
- c'est le meilleur et le plus simple algorithme

Poids unitaires

Plus courts chemins dans un graphe non pondéré
Certificat $\text{dist}(s, t) = k$

- $\text{dist}(s, t) \leq k$:

Poids unitaires

Plus courts chemins dans un graphe non pondéré

Certificat $\mathrm{dist}(s,t) = k$

- $\mathrm{dist}(s,t) \leq k$: exhiber un *st*-chemin de longueur k
- $\mathrm{dist}(s,t) \geq k$?

Poids unitaires

Algorithme 1 : BFS

Données : Un graphe orienté $G = (V, A)$ et un sommet s de G

Résultat :

- une arborescence de plus courts chemins d'origine s
- Les distances $\mathrm{dist}(s, v)$ de s à tous les sommets v de G

$\lambda(v) \leftarrow +\infty \quad \forall v$ sommet de G $\qquad F$ une file vide

Ajouter s à F $\qquad \lambda(s) \leftarrow 0$

tant que *F est non vide* **faire**

  Défiler un sommet de F : v

  pour *chaque sommet w tel que $vw \in A$* **faire**

    si $\lambda(w) = +\infty$ **alors**

      Enfiler w dans F

      $\pi(w) \leftarrow v$

      $\lambda(w) \leftarrow \lambda(v) + 1$

retourner λ, π

Poids unitaires

Définition

Soient s et t deux sommets. $S \subset V$ est une *st*-**coupe** si $s \in S$ et $t \notin S$

Certificat $\mathrm{dist}(s,t) \geq k$

- Si S est une *st*-coupe alors chaque chemin de s à t contient au moins un arc sortant de S ($uv \in A$ avec $u \in S$ et $v \notin S$)

Poids unitaires

Définition

Soient s et t deux sommets. $S \subset V$ est une *st*-**coupe** si $s \in S$ et $t \notin S$

Certificat $\mathrm{dist}(s,t) \geq k$

- Si S est une *st*-coupe alors chaque chemin de s à t contient au moins un arc sortant de S ($uv \in A$ avec $u \in S$ et $v \notin S$)
- Si on a une famille de k *st*-coupes dont les ensembles d'arcs sortants sont disjoints deux-à-deux, alors, $\mathrm{dist}(s,t) \geq k$

Poids unitaires

Définition

Soient s et t deux sommets. $S \subset V$ est une st-**coupe** si $s \in S$ et $t \notin S$

Certificat $\text{dist}(s, t) \geq k$

- Si S est une st-coupe alors chaque chemin de s à t contient au moins un arc sortant de S ($uv \in A$ avec $u \in S$ et $v \notin S$)
- Si on a une famille de k st-coupes dont les ensembles d'arcs sortants sont disjoints deux-à-deux, alors, $\text{dist}(s, t) \geq k$
- BFS fournit une telle famille :

Poids unitaires

Définition

Soient s et t deux sommets. $S \subset V$ est une st-**coupe** si $s \in S$ et $t \notin S$

Certificat $\mathrm{dist}(s,t) \geq k$

- Si S est une st-coupe alors chaque chemin de s à t contient au moins un arc sortant de S ($uv \in A$ avec $u \in S$ et $v \notin S$)
- Si on a une famille de k st-coupes dont les ensembles d'arcs sortants sont disjoints deux-à-deux, alors, $\mathrm{dist}(s,t) \geq k$
- BFS fournit une telle famille :
  $S_i = \{v \in V / \lambda(v) \leq i\}$ pour $i = 0, 1 \ldots k-1$

Poids unitaires

Définition

Soient s et t deux sommets. $S \subset V$ est une *st*-**coupe** si $s \in S$ et $t \notin S$

Certificat $\mathrm{dist}(s, t) \geq k$

- Si S est une *st*-coupe alors chaque chemin de s à t contient au moins un arc sortant de S ($uv \in A$ avec $u \in S$ et $v \notin S$)
- Si on a une famille de k *st*-coupes dont les ensembles d'arcs sortants sont disjoints deux-à-deux, alors, $\mathrm{dist}(s, t) \geq k$
- BFS fournit une telle famille :
  $S_i = \{v \in V / \lambda(v) \leq i\}$ pour $i = 0, 1 \ldots k - 1$
  - *st*-coupes : $\lambda(s) = 0$ donc $s \in S_i$, $\lambda(t) \geq k$ donc $t \notin S_i$,

Poids unitaires

Définition

Soient s et t deux sommets. $S \subset V$ est une st-**coupe** si $s \in S$ et $t \notin S$

Certificat $\text{dist}(s,t) \geq k$

- Si S est une st-coupe alors chaque chemin de s à t contient au moins un arc sortant de S ($uv \in A$ avec $u \in S$ et $v \notin S$)
- Si on a une famille de k st-coupes dont les ensembles d'arcs sortants sont disjoints deux-à-deux, alors, $\text{dist}(s,t) \geq k$
- BFS fournit une telle famille :
  $S_i = \{v \in V / \lambda(v) \leq i\}$ pour $i = 0, 1 \ldots k-1$
  - st-coupes : $\lambda(s) = 0$ donc $s \in S_i$, $\lambda(t) \geq k$ donc $t \notin S_i$,
  - ens. arcs sortants disjoints : Si $uv \in A$ alors $\lambda(v) \leq \lambda(u) + 1$

Poids unitaires

Généralisation

- Idée pour encoder un poids entier positif p :

Poids unitaires

Généralisation

- Idée pour encoder un poids entier positif p : subdiviser un arc en p arcs (bonne idée ?)
- Idée pour encoder un poids négatif : personne ne l'a trouvée

Plan

Algorithme de Dijkstra

Algorithme de Dijkstra

- Graphe avec des poids **positifs**
- garantit qu'il n'y a pas de circuit négatif

Algorithme de Dijkstra

Algorithme de Dijkstra : idées

À chaque étape :

- $V = S \cup V \backslash S$
- Si $v \in S$, $\lambda(v) = \mathrm{dist}(s, v)$
- Si $v \notin S$, $\lambda(v) \geq \mathrm{dist}(s, v)$
  $\lambda(v) =$ longueur d'un plus court sv-chemin qui n'utilise que des sommets de S.
- Le sommet suivant t qui rentre dans S : valeur de λ minimale.
- Puis mise à jour des voisins de t si on peut améliorer

Algorithme de Dijkstra

Dijkstra(s)

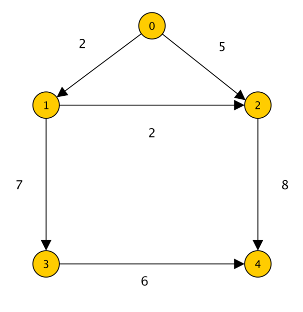
Données : un graphe $G = (V, A, w)$ avec des poids positifs et un sommet s de G

Résultat :

- une arborescence de plus courts chemins d'origine s
- Les distances $\mathrm{dist}(s, v)$ de s à tous les sommets v de G

pour *v sommet de G* **faire**
  $\lambda(v) \leftarrow +\infty$
$S \leftarrow \emptyset \quad \lambda(s) \leftarrow 0$
tant que $S \neq V$ **faire**
  $t \leftarrow$ un sommet de $V \setminus S$ tel que $\lambda(t)$ soit minimum
  $S \leftarrow S \cup \{t\}$
  tant que *il existe $v \notin S$ avec $tv \in A$* **faire**
    si $\lambda(v) > \lambda(t) + w(tv)$ **alors**
      $\lambda(v) \leftarrow \lambda(t) + w(tv)$
      $\pi(v) \leftarrow t$
retourner λ, π

Algorithme de Dijkstra



valeurs de λ $s = 0$

0	1	2	3	4
0	∞	∞	∞	∞
0	2 (0)	5 (0)	∞	∞
0	2 (0)	4 (1)	9 (1)	∞
0	2 (0)	4 (1)	9 (1)	12 (2)
0	2 (0)	4 (1)	9 (1)	12 (2)

Algorithme de Dijkstra

1. *tout s'exécute correctement*

Algorithme de Dijkstra

1. *tout s'exécute correctement*
2. *en un nombre fini d'étapes*

Algorithme de Dijkstra

1. *tout s'exécute correctement*
2. *en un nombre fini d'étapes*
   - $S \subset V$ et à chaque étape $|S|$ augmente de 1
3. *en cas d'arrêt, on obtient l'objet souhaité*

Algorithme de Dijkstra

1. *tout s'exécute correctement*
2. *en un nombre fini d'étapes*
   - $S \subset V$ et à chaque étape $|S|$ augmente de 1
3. *en cas d'arrêt, on obtient l'objet souhaité*
   - A démontrer : $\lambda(v) = \text{dist}(s, v)$ pour tout $v \in S$

Algorithme de Dijkstra

Preuve par récurrence que $\lambda(v) = \text{dist}(s, v)$ pour tout $v \in S$

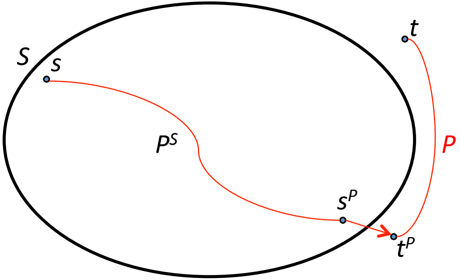
Soit $P(k)$ la propriété : à l'itération k, pour tout sommet $v \in S$, on a : $\lambda(v) = \text{dist}(s, v)$

- Vrai pour $k = 0$: pour le sommet s, $\lambda(s) = 0$ et $\text{dist}(s, v) = 0$. (trivial)
- On suppose que c'est vrai à l'itération $k - 1$.
- On démontre que la propriété est maintenue à l'itération suivante
  c'est-à-dire, $\lambda(t) = \text{dist}(s, t)$ lorsque t est ajouté dans S

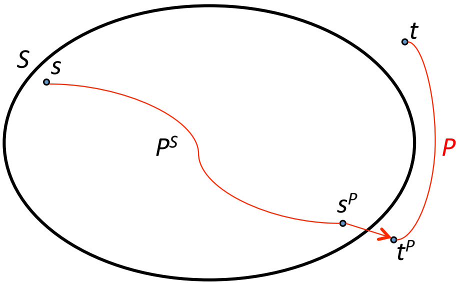
Algorithme de Dijkstra

Juste avant l'ajout de t dans S,

- Soit P un chemin quelconque de s à t
- Soit t^P le premier sommet de P non dans S (existe car $s \in S$ et $t \notin S$)
- Soit s^P le prédécesseur de t^P dans P
- Soit P^S le sous chemin de P de s à s^P

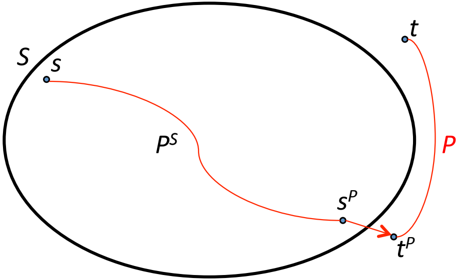


Algorithme de Dijkstra



$$w(P) \geq w(P^s) + w(s^P t^P)$$

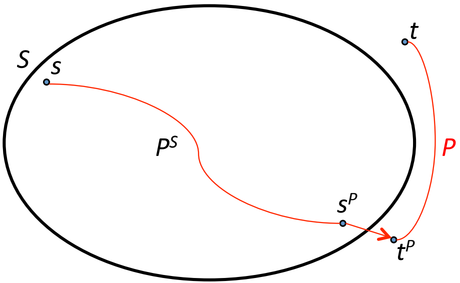
Algorithme de Dijkstra



$$w(P) \geq w(P^s) + w(s^P t^P)$$

P^S plus l'arc $s^P t^P$ sous chemin de P

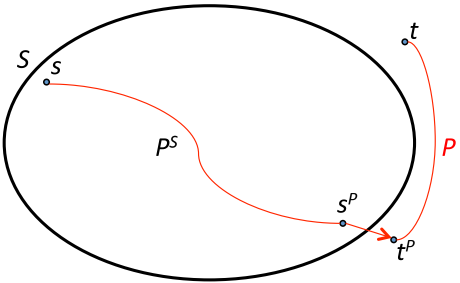
Algorithme de Dijkstra



$$w(P) \geq w(P^s) + w(s^P t^P) \ \geq \lambda(s^P) + w(s^P t^P)$$

$\lambda(s^P) = \text{dist}(s, s^P)$ par l'hypothèse de récurrence

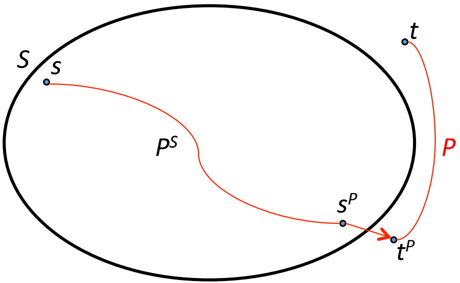
Algorithme de Dijkstra



$$w(P) \geq w(P^s) + w(s^P t^P) \ \geq \lambda(s^P) + w(s^P t^P) \ \geq \lambda(t^P)$$

mis à jour lorsque l'arc $s^P t^P$ a été vu en traitant s^P

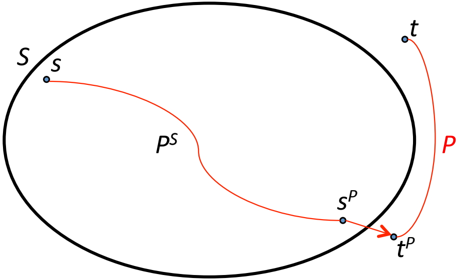
Algorithme de Dijkstra



$$w(P) \geq w(P^s) + w(s^P t^P) \ \geq \lambda(s^P) + w(s^P t^P) \ \geq \lambda(t^P) \ \geq \lambda(t)$$

choix du λ minimum hors de S

Algorithme de Dijkstra

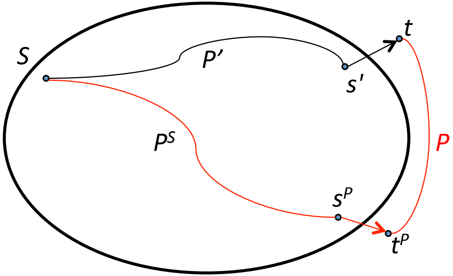


$$w(P) \geq w(P^s) + w(s^P t^P) \ \geq \lambda(s^P) + w(s^P t^P) \ \geq \lambda(t^P) \ \geq \lambda(t)$$

Donc quel que soit le chemin P de s à t, $w(P) \geq \lambda(t)$. Donc $\lambda(t) \leq \text{dist}(s, t)$

Algorithme de Dijkstra

Il faut encore montrer que $\lambda(t) \geq \text{dist}(s,t)$



- $\exists s' \in S$ tel que $\lambda(t) = \lambda(s') + w(s't)$
- Soit P' un plus court chemin de s à s' de longueur $\lambda(s')$ (par hypothèse de récurrence)
- $\lambda(t)$ est la longueur du chemin composé de P' augmentée de $w(s't)$. Donc $\lambda(t) \geq \text{dist}(s,t)$

Donc lorsque t est ajouté à S, on a $\lambda(t) = \text{dist}(s,t)$

Algorithme de Dijkstra

Remarques

- Si on tombe dans l'algo sur t avec $\lambda(t) = +\infty$ est-ce que ça vaut le coup de continuer ?
- extension graphe non orienté avec longueurs positives

Donnez un exemple de graphe sans circuit absorbant où l'algorithme de Dijkstra ne donne pas les plus courts chemins.

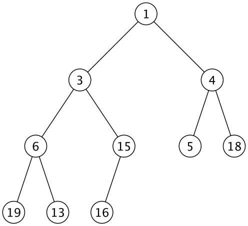
Comment coder **efficacement** l'algorithme de Dijkstra ?

- $=$ comment enlever rapidement l'élément de λ minimum ? $\Rightarrow$ on utilise une file de priorité
- $\Rightarrow$ Structure de données : **tas binaire** [*Binary heap*]
  - permet d'encoder des ensembles, d'ajouter des éléments, de retirer l'élément de plus petite étiquette...
  - ex : heap sort

Tas binaires

Tas binaire : arbre enraciné qui vérifie les propriétés suivantes

- c'est un arbre binaire parfait :
  - Les nœuds du dernier niveau n'ont pas de fils
  - Les nœuds de l'avant dernier niveau ont au plus deux fils
  - Tous les autres nœuds ont deux fils
  - si le dernier niveau n'est pas totalement rempli, alors il est rempli de gauche à droite
- c'est un tas :
  - chaque nœud contient une étiquette
  - l'étiquette du père est plus petite que les étiquettes de ses fils

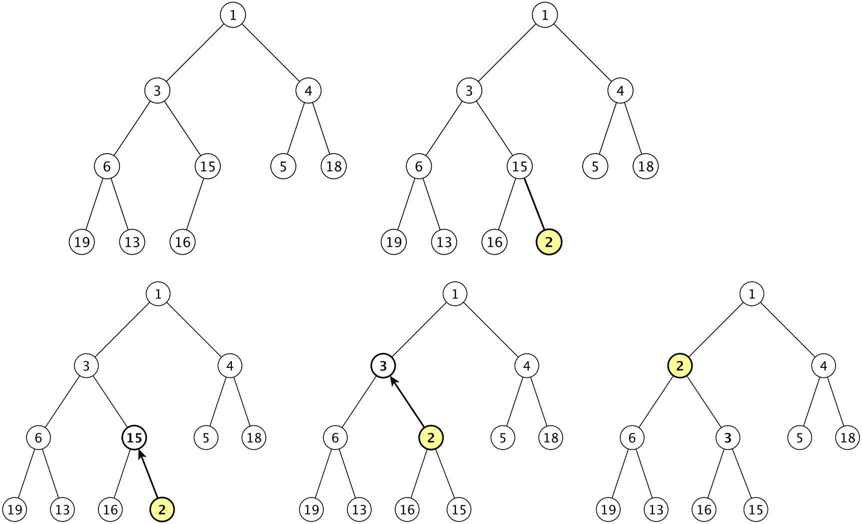


Tas binaires

Tas binaires : opérations

- **ajouter** un élément
  - L'élément *s* est d'abord rajouté en dernière position au tas. Puis, tant qu'il n'est pas la racine et qu'il est plus petit que son père, on échange les positions entre *s* et son père.

Tas binaire : ajouter

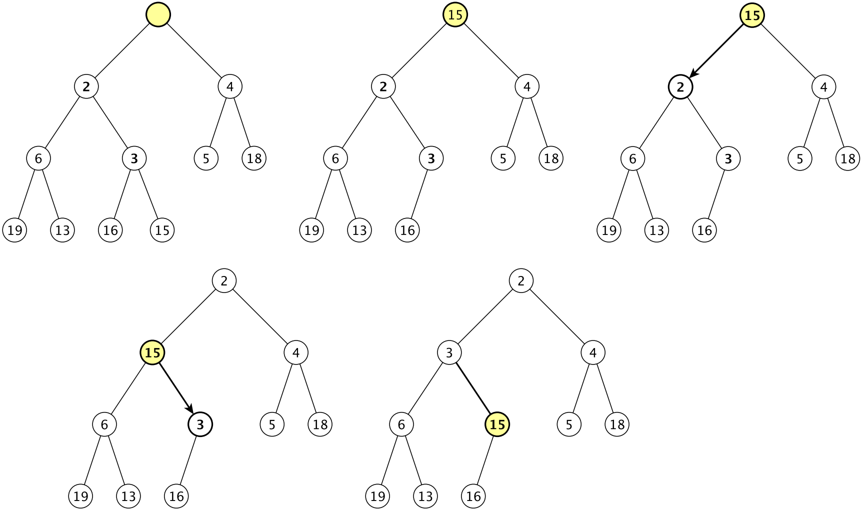


Tas binaires

Tas binaires : opérations

- **ajouter** un élément
  - L'élément *s* est d'abord rajouté en dernière position au tas. Puis, tant qu'il n'est pas la racine et qu'il est plus petit que son père, on échange les positions entre *s* et son père.
- **retirer**
  - renvoie la valeur du sommet racine du tas, qui correspond au sommet qui a la plus petite valeur.
  - met le tas à jour : remplace la valeur à la racine par celle du dernier sommet du tas puis tant que ce sommet a des fils et qu'il est strictement supérieur à ses fils, échanger sa position avec celle du plus petit de ses fils.

Tas binaire : retirer



Tas binaires

Tas binaires : opérations

- **diminuer** l'étiquette d'un nœud
  Il faut faire attention à maintenir la structure de tas en faisant éventuellement remonter le nœud qui a été modifié dans l'arbre comme lorsqu'on ajoute un sommet.

Remarque : en pratique, pour appliquer l'algorithme de Dijkstra, on stocke pour chaque sommet le numéro du sommet et l'étiquette correspondante sera la valeur de λ pour ce sommet.

Tas binaires

Tas binaires : complexité

Le nombre d'opérations de chaque fonction est borné par la hauteur de l'arbre.

Arbre binaire complet $\Rightarrow$
profondeur $\leq$ logarithme du nombre de nœuds.

Chaque opération sur le tas binaire a donc une complexité $O(\log n)$
L'algorithme de Dijkstra a donc une complexité [4]
$O((|V| + |E|) \log |V|)$

4. $|V|*$retirer$+(|V| + |E|)*$ajouter/diminuer

Conclusion

- poids unitaires : BFS
- coûts positifs : algorithme de Dijkstra
- pas de circuit : algorithme de Bellman (extension possible aux plus longs chemins)
- Algorithme de Bellman Ford : caractérisation des graphes orientés pondérés sans circuit absorbant.

Plan

Graphe sans circuit

Graphe sans circuit

- *Directed Acyclic Graph* ou DAG
- Utilisé en programmation dynamique (états associés à une formule de récurrence)

Décrivez un graphe qui n'a ni source, ni puits.

Un DAG a toujours une source et un puits [5]

preuve ? (algorithmique, par contre-exemple maximal)

5. rappel : le nombre de sommets est fini

Graphe sans circuit

Propriété

Un graphe est sans circuit si et seulement si il admet un ordre topologique.

- Un ordre $v_1 = s, \ldots, v_n$ des sommets est topologique si tous les arcs sont de la forme $v_i v_j$ avec $i < j$.
- l'algorithme DFS permet d'obtenir un ordre topologique en temps linéaire. (Fournier, page 216)
- autre algorithme ?

Algorithme de Bellman

Algorithme 3 : Plus court chemin dans un DAG

Données : un graphe orienté valué G sans circuit et un sommet s

Résultat : une arborescence de plus courts chemins d'origine s

Soit $v_1, v_2 \ldots, v_n$ un ordre topologique des sommets de G
Pour tout sommet v, $d(v) \leftarrow \infty$
$d(s) \leftarrow 0$
pour $k = 1$ **à** n **faire**
  Pour chaque sommet v tel que $v_k v \in A$
  si $d(v) > d(v_k) + w(v_k v)$ **alors**
    $d(v) \leftarrow d(v_k) + w(v_k v)$
    $\pi(v) \leftarrow v_k$

Algorithme de Bellman

Extensions

- Plus long chemin : chemin orienté dont le poids est maximum :

Algorithme de Bellman

Extensions

- Plus long chemin : chemin orienté dont le poids est maximum : opposé des poids.
- Plus sûrs chemins :

Algorithme de Bellman

Extensions

- Plus long chemin : chemin orienté dont le poids est maximum : opposé des poids.
- Plus sûrs chemins : logarithme des poids.

cf feuille d'exercice