Plus court chemin

Problème très classique de théorie des graphes avec de multiples applications (chemin le plus rapide ou de distance minimum entre deux points : cf itinéraire sous google-map, waze, viamichelin, tables de routage en informatique, ...). Dans sa forme la plus simple, le problème est bien résolu avec plusieurs algorithmes polynomiaux de résolution proposés il y a une cinquantaine d'années.
Mais, de nombreuses variations peuvent être introduites, certaines parvenant même à rendre le problème bien plus difficile (NP dur).

- Contraintes additionnelles : passage obligatoire en certains points, précédence entre sommets, législation du travail (par ex points de pause), plusieurs chemins arcs-disjoints pour sécurisation, contrainte sur la capacité du véhicule, fenêtres de temps de passage en un point, ...
- Autre élément de variation : One to one (une origine, une destination), One to all (broadcast), All to all.
- Existence ou non d'arcs de coûts négatifs et/ou de circuits.

Définitions de base

Données

- Un graphe $G = (V, E)$ orienté
- Une fonction de pondération (ou de distance) $w : E \to \mathbb{R}$

Poids d'un chemin

Le poids du chemin $p =< v_0, v_1, v_2, \ldots, v_k >$ est la somme des poids des arcs qui le constituent.

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Algorithme de Dijkstra

Cadre :

- Une seule origine (source) : sommet s
- Un graphe avec des arcs valués positivement, symétrique ou non

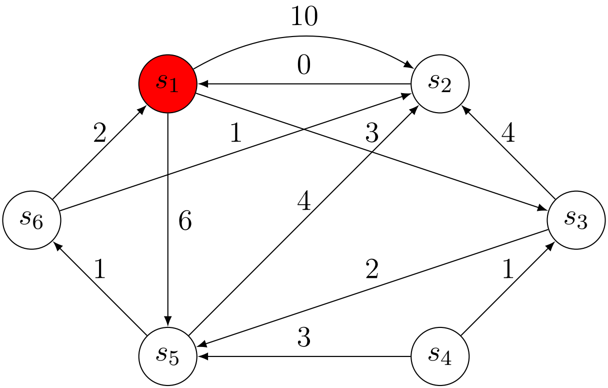
Idée :

- On calcule progressivement les plus courtes distances de s vers chaque sommet x_i (d_i).
- A chaque itération, on traite le sommet non encore traité de plus petite valeur d_i (car on est sûr avec les valuations positives qu'il ne pourra plus être amélioré).
- on crée progressivement une arborescence des plus courts chemins de racine s par la structure *pere*.

Détail de l'algorithme de Dijkstra

- Initialisations :
  - $T = \{s\}$; $d_s = 0$;
  - $\forall i \neq s$, si l'arc (s,i) existe alors $d_i = w(s,i)$ (∞ sinon), $pere(i) = s$
- Boucle principale : Tant que $T \neq V$ faire
  - Trouver un noeud t de $V - T$ tq $d_t = Min(d_i, i \in V - T)$,
  - $T = T \cup \{t\}$
  - $\forall k \in \Gamma_t^+$
    - Si $(d_k > d_t + w(t,k))$ alors $d_k = d_t + w(t,k)$, $pere(k) = t$

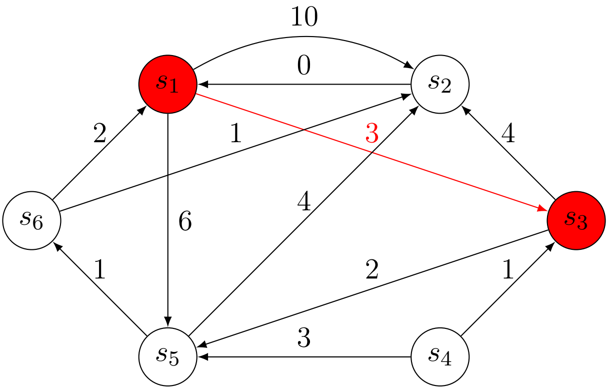
Exemple : pcc à partir de s_1



$V = \{s_1\}$

Sommet	$\mathbf{s_1}$	s_2	s_3	s_4	s_5	s_6
d	**0**	10	3	∞	6	∞
pere	-	s_1	s_1	-	s_1	-

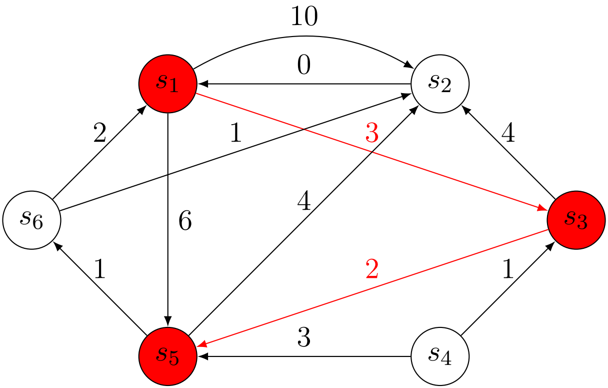
Exemple : pcc à partir de s_1



$V = \{s_1, s_3\}$

Sommet	$\mathbf{s_1}$	s_2	$\mathbf{s_3}$	s_4	s_5	s_6
d	**0**	7	**3**	∞	5	∞
pere		s_3	s_1	-	s_3	-

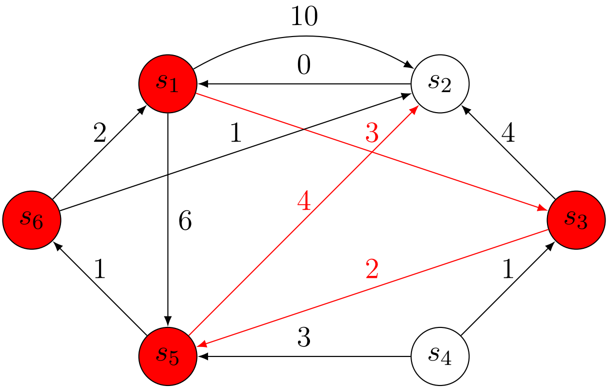
Exemple : pcc à partir de s_1



$V = \{s_1, s_3, s_5\}$

Sommet	$\mathbf{s_1}$	s_2	$\mathbf{s_3}$	s_4	s_5	s_6
d	**0**	7	**3**	∞	**5**	6
pere	-	s_3	s_1	-	s_3	s_5

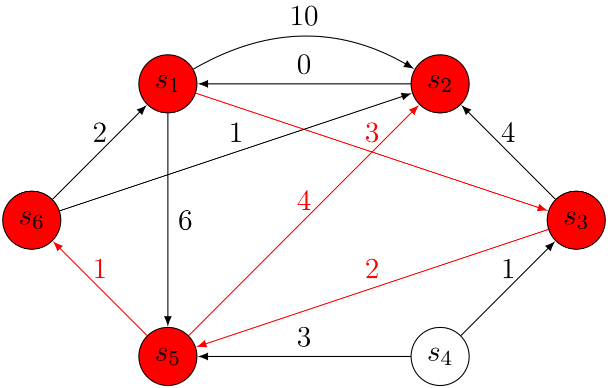
Exemple : pcc à partir de s_1



$V = \{s_1, s_3, s_5, s_6\}$

Sommet	$\mathbf{s_1}$	s_2	$\mathbf{s_3}$	s_4	s_5	s_6
d	**0**	7	**3**	∞	**5**	**6**
pere		s_3	s_1	-	s_3	s_5

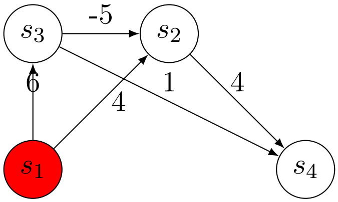
Exemple : pcc à partir de s_1



$V = \{s_1, s_2, s_3, s_5, s_6\}$

Sommet	$\mathbf{s_1}$	s_2	$\mathbf{s_3}$	s_4	s_5	s_6
d	**0**	**7**	**3**	∞	**5**	**6**
pere		s_3	s_1	-	s_3	s_5

Ca ne marche plus avec des coûts négatifs



Structure de données - Complexité

L'implémentation la plus simple consiste à faire un tableau pour d. On obtient alors une complexité en $O(n^2)$:

- A chaque itération, on traite un sommet ($n-1$ itérations)
- Lors du traitement d'un sommet, on parcourt le tableau d pour rechercher le sommet non traité de plus petite valeur ($O(n)$)
- Chaque successeur (en fait chaque arc) est traité une et une seule fois. Or $m \leq n.(n-1)$. D'où $O(n^2)$

On peut se battre pour obtenir une meilleure complexité. La meilleure complexité pouvant être obtenue pour cet algorithme est en $O(m + n.log_2 n)$.
On peut noter que si $m = O(n^2)$, c'est à dire si le graphe est complet ou tout du moins très dense, cela a donc peu d'intérêt.

Une complexité en O(n.log n)

Pour l'obtenir, il faut utiliser une structure de données un peu complexe : soit un tas binomial, soit un tas de Fibonacci. L'objectif est que toutes les opérations suivantes soient en $log_2 n$:

- Trouver l'élément de plus petite valeur
- Effacer du tas l'élément de plus petite valeur (en remontant les autres donc)
- Accéder à la valeur d'un élément donné
- Diminuer la valeur d'un élément donné (= Effacer un élément donné du tas + insérer un nouvel élément)