Autres algorithmes de plus court chemin que Dijkstra

- Si on a des poids négatifs sur certains arcs et qu'on recherche toujours les plus courts chemins depuis une source s vers les autres sommets: **algorithme de Bellman-Ford**
- Si on recherche les plus courts chemins entre toutes les paires de sommets (all to all) **algorithme de Floyd-Warshall**
- S'il n'y a aucun circuit dans le graphe (DAG), il est possible de faire très simple en traitant les sommets dans l'ordre donné par un **tri topologique**

Algorithme de Ford-Bellman (1)

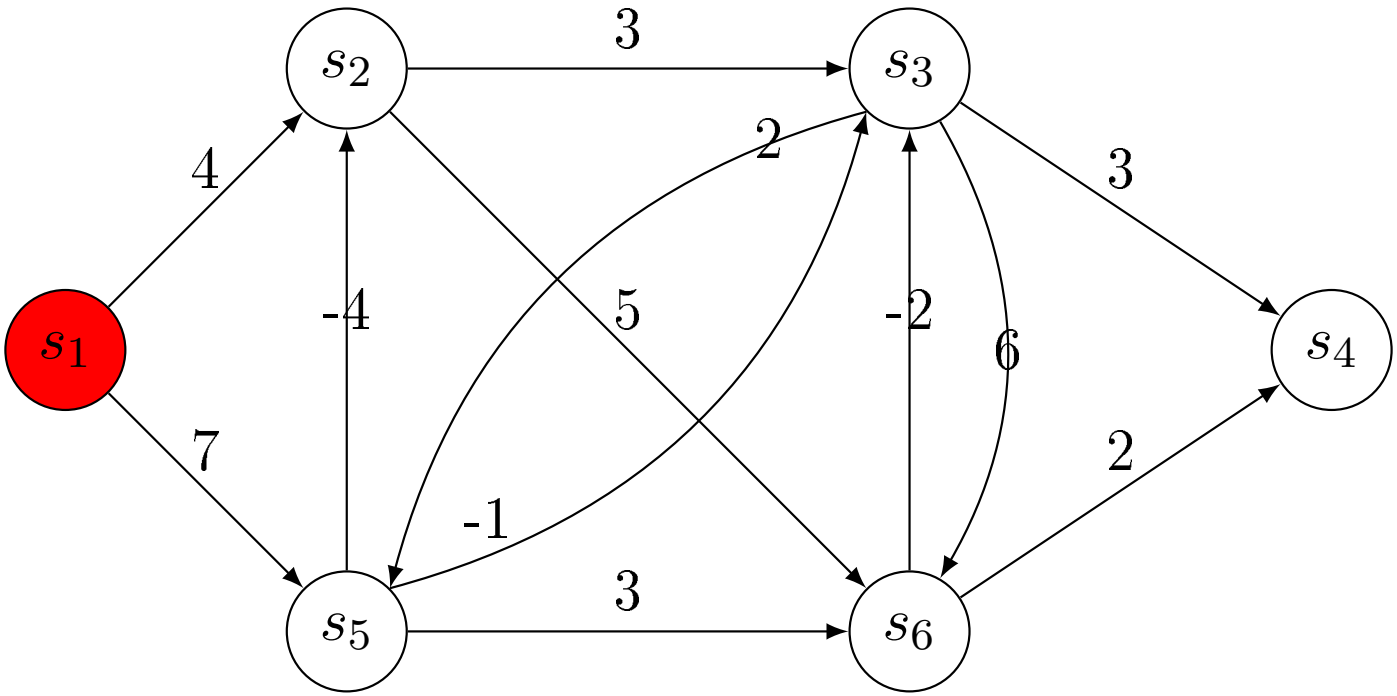
La méthode est assez proche dans sa forme générale de celle de Dijkstra. Mais la valeur d d'un sommet déjà traité pouvant être améliorée, il est nécessaire de revenir sur le traitement de ce sommet avec sa nouvelle valeur. Un sommet peut donc être traité plusieurs fois.

Par ailleurs, on ne traite pas forcément le sommet ayant la valeur d la plus petite (donc pas de recherche du sommet avec la plus petite valeur de d) mais un sommet quelconque dans la liste des sommets à traiter.

Algorithme de Ford-Bellman (2)

- Initialisations:
  - $d_s = 0$; $pere_s = Nil$;
  - $\forall i \neq s$, $d_i = \infty$; $pere_i = Nil$
  - $L = \{s\}$;
- Boucle principale: Tant que $L \neq \emptyset$ faire
  - Choisir $t \in L$
  - $L = L - \{t\}$
  - $\forall k \in \Gamma_t^+$ faire
    - ★ Si $(d_k > d_t + w(t,k))$ alors $d_k = d_t + w(t,k)$; $pere_k = t$; $L = L \cup \{k\}$;

Algo de BF (3)



Les itérations

Sommet	s_1	s_2	s_3	s_4	s_5	s_6	L
Init	**0**	∞	∞	∞	∞	∞	$\{s_1\}$
s_1	0	**4** (s_1)	∞	∞	**7** (s_1)	∞	$\{s_2, s_5\}$
s_2	0	4 (s_1)	**7** (s_2)	∞	7 (s_1)	**9** (s_2)	$\{s_5, s_3, s_6\}$
s_5	0	**3** (s_5)	**6** (s_5)	∞	7 (s_1)	9 (s_2)	$\{s_3, s_6, s_2\}$
s_3	0	3 (s_5)	6 (s_5)	**9** (s_3)	7 (s_1)	9 (s_2)	$\{s_6, s_2, s_4\}$
s_6	0	3 (s_5)	6 (s_5)	9 (s_3)	7 (s_1)	9 (s_2)	$\{s_2, s_4\}$
s_2	0	3 (s_5)	6 (s_5)	9 (s_3)	7 (s_1)	**8** (s_2)	$\{s_4, s_6\}$
s_4	0	3 (s_5)	6 (s_5)	9 (s_3)	7 (s_1)	8 (s_2)	$\{s_6\}$
s_6	0	3 (s_5)	6 (s_5)	9 (s_3)	7 (s_1)	8 (s_2)	$\emptyset$

Algo FB - Preuve de l'algorithme

Si le plus court chemin élémentaire de s à un sommet v comporte k arcs, alors après avoir traité au plus k fois ce sommet, on a d_v à sa juste valeur: $d_v = d(s, v)$.
Preuve par récurrence (assez évidente):

- C'est vrai pour $k = 1$: les sommets reliés directement à s dans l'arborescence des + courts chemins sont directement à leur bonne valeur dès le traitement initial de s. Ils ne sont traités qu'une fois.
- A $k - 1$, la propriété est vraie pour le prédecesseur p de v sur le plus court chemin de s à v (le plus court chemin de s à p comportant $k - 1$ arcs).

Circuits absorbants

Du fait de la propriété précédente, l'algorithme nécessite dans le pire des cas: $1+1+2+...+n-1 = \frac{n.(n-1)}{2}+1$ itérations pour converger (en tout cas $O(n^2)$).

Définition: Un circuit absorbant est un circuit dont la somme des poids est négative.

En cas de circuit absorbant, il n'y a pas de solution au plus court chemin pour les sommets concernés: on a intérêt à parcourir indéfiniment le circuit absorbant.

Détection de l'existence d'un circuit absorbant par l'algorithme de BF: si le nombre d'itérations dépasse le nombre maximum possible indiqué ci-dessus, c'est qu'il y a un circuit absorbant: stop.

Complexité

Une autre version de l'algo (bien plus bourrine et que j'aime moins), basée sur les propriétés précédentes consiste à répéter $n-1$ fois le parcours de chaque arc pour voir s'il améliore le d du sommet final de l'arc:

- Boucle principale: Pour i allant de 1 à n-1 faire
  - Pour chaque arc (u, v) faire
    - Si $(d_v > d_u + w(u, v))$ alors $d_v = d_u + w(u, v)$; $pere_v = u$;

Il a toutefois l'intérêt de bien montrer que la complexité de l'algo est en $O(n.m)$. Donc, en cas de graphe dense, en $O(n^3)$.

Algorithme de Floyd- Warshall

Le graphe est représenté sous forme d'une matrice d'adjacence avec:

- $M[i,j]$ = la valeur de l'arc (i,j) si cet arc existe.
- $M[i,j] = +\infty$ sinon.

Les arcs du graphe peuvent avoir des poids négatifs. L'algo, très simple est le suivant:

- $W = M$ (pour garder la matrice d'adjacence intacte)
- Pour k allant de 1 à n faire
  - Pour i allant de 1 à n faire
    - Pour j allant de 1 à n faire
    - $W[i,j] = Min(W[i,j], W[i,k] + W[k,j])$

A l'étape k, on améliore tous les chemins passant par le sommet k.

Remarques

- La complexité est de manière évidente en $O(n^3)$
- Cet algo ne donne que les valeurs des plus courts chemins de chaque sommet i à chaque sommet j.
- Mais c'est bien plus compliqué d'ajouter une structure qui permet de retrouver la composition de chaque plus court chemin, car en chaque sommet il faut un père pour chaque origine: $pred_u[i]$: le père de u dans l'arborescence des plus courts chemins depuis le sommet i.
- Pour les tables de routage, on fait plutôt l'inverse: $succ_u[j]$: à qui faut-il faire suivre le message quand le destinataire final est j.

Reprendre l'exemple du slide 4.

Pour les DAG, c'est bien plus facile !

Dans ce cas, il suffit en effet d'effectuer en premier lieu un tri topologique des sommets (ordre où chaque sommet est plaçé avant ses successeurs).
Puis, tout simplement de traiter les sommets dans cet ordre:
$d_i = Min_{k \in \Gamma_i^-} (d_k + w(k,i))$ (+pere determiné ainsi).