Problème de plus court chemin

Problème de plus court chemin

- **Entrée :** un graphe $G = (V, E)$, des longueurs $l(u, v)$ pour tout $(u, v) \in E$, deux sommets $s, t \in V$.
- **Sortie :** $\delta(s, t)$ la longueur du plus court chemin entre s et t.

Plusieurs cas

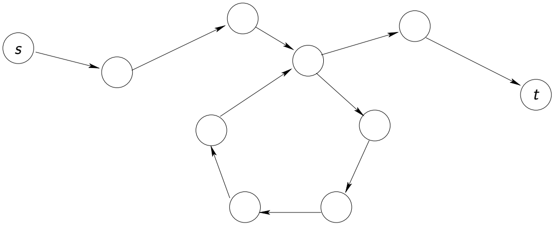
- **Arcs de longueurs positives :** l'algorithme de Dijkstra résout le problème en $|E| \log |V|$.
- **Arcs de longueurs quelconques :**
  - **Cycle de longueur négative :** pas de plus court chemin
  - **Pas de cycle de longueur négative :** on peut résoudre le problème en utilisant la *programmation dynamique :*

algorithme de Bellman-Ford

Idée de base

Idée de base

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin **élémentaire** entre s et t.



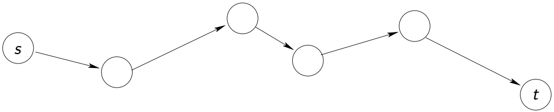
Sous-problème

- $OPT(i, v)$ la longueur minimale d'un chemin de v à t contenant au maximum i arcs (**objectif** : calculer $OPT(n-1, s)$).

Idée de base

Idée de base

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin **élémentaire** entre s et t.



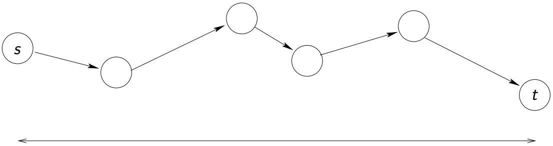
Sous-problème

- $OPT(i, v)$ la longueur minimale d'un chemin de v à t contenant au maximum i arcs (**objectif** : calculer $OPT(n-1, s)$).

Idée de base

Idée de base

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin **élémentaire** entre s et t.



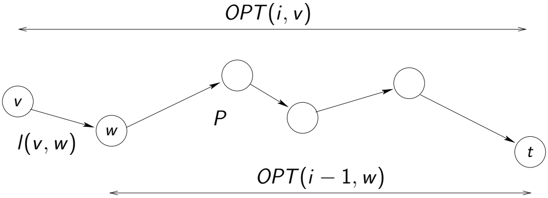
$\leq n - 1$ arcs

Sous-problème

- $OPT(i, v)$ la longueur minimale d'un chemin de v à t contenant au maximum i arcs (**objectif** : calculer $OPT(n-1, s)$).

Formule de récurrence

Soit P un chemin optimal pour le sous-problème $OPT(i, v)$



Formule de récurrence

Deux cas :

- si P utilise au plus $i-1$ arcs, $OPT(i, v) = OPT(i-1, v)$;
- si P utilise exactement i arcs, $OPT(i, v) = l(v, w) + OPT(i-1, w)$

On déduit la formule de récurrence suivante, pour tout $i > 0$, $v \in V - \{t\}$,

$$OPT(i, v) = \min(OPT(i-1, v), \min_{w \in V}(l(v, w) + OPT(i-1, w))). \qquad (1)$$

Algorithme de Bellman-Ford

En utilisant, la formule de récurrence (1), on obtient l'algorithme de programmation dynamique suivant pour calculer $OPT(n-1, s)$.

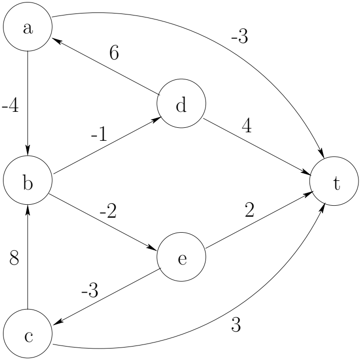
Algorithme de Bellman-Ford (V1)

Bellman-Ford-V1(G, s, t**)**
`Fixer` $M[0, v] = \infty$ `pour tout` $v \in V - \{t\}$ `et` $M[0, t] = 0$
`Pour` $i = 1, \ldots, n-1$ `faire`
  `Pour` $v \in V$ `faire`
    $M[i, v] = \min(M[i-1, v], \min_{w \in V}(l(v, w) + M[i-1, w]))$
  `Fin-faire`
`Fin-faire`
`Renvoyer` $M[n-1, s]$

Complexité

Chacune des n^2 entrées de la table M est calculée en temps $O(n)$, la complexité de cet algorithme est donc $O(n^3)$.

Exemple



(a)

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4	-6	-6
b	∞	∞	0	-2	-2	-2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

(b)

Amélioration du temps de calcul

Amélioration du temps de calcul

L'algorithme de Bellman-Ford peut-être implémenté en $O(mn)$ avec $n := |V|$ et $m = |E|$.

Justification

- En effet, lors de l'évaluation de l'expression

  $$\min_{w \in V}(l(v,w) + M[i-1,w])$$

  il suffit de considérer les sommets w qui sont des voisins de v

  $$\min_{w \in \mathrm{Adj}(v)}(l(v,w) + M[i-1,w])$$

- Le coût d'évaluation de chacune des entrées $M[i,v]$ est donc en $O(n_v)$, où n_v est le nombre de voisins de v.
- Le coût global de l'algorithme est $O(n \sum_{v \in V} n_v)$, i.e. $O(nm)$.

Algorithme de Bellman-Ford

L'amélioration conduit à cette deuxième version de l'algorithme :

Algorithme de Bellman-Ford (V2)

Bellman-Ford-V2(G, s, t)
Fixer $M[0, v] = \infty$ pour tout $v \in V - \{t\}$ et $M[0, t] = 0$
Pour $i = 1, \ldots, n-1$ faire
  Pour $v \in V$ faire
    $M[i, v] = \min(M[i-1, v], \min_{w \in \mathtt{Adj}[v]}(l(v, w) + M[i-1, w]))$
  Fin-faire
Fin-faire
Renvoyer $M[n-1, s]$

Complexité

Chacune des entrées de la table $M[i, v]$ est calculée en temps $O(n_v)$, la complexité de cet algorithme est donc $O(n \sum_{v \in V} n_v) = O(nm)$.

Amélioration de l'espace mémoire

Espace Mémoire

- Pour diminuer l'espace mémoire, on peut garder une seule valeur $M[v]$ pour chaque sommet v.
- Pour $i = 1, \ldots, n-1$, on effectue la mise à jour suivante

$$M[v] = \min(M[v], \min_{w \in \mathtt{Adj}(v)} (l(v, w) + M[w])).$$

Observation

Tout au long de l'exécution de l'algorithme, $M[v]$ représente la longueur d'un chemin allant de v à t, et après i mises à jour la valeur $M[v]$ est **au plus** *la longueur d'un plus court chemin entre v et t contenant au maximum i arcs.*

Amélioration de l'espace mémoire

Algorithme de Bellman-Ford (V3)

Bellman-Ford-V3(G, s, t)

```
Fixer M[v] = ∞ pour tout v ∈ V − {t} et M[t] = 0
Pour i = 1, ..., n − 1 faire
   Pour v ∈ V faire
      M[v] = min(M[v], min_{w∈Adj[v]}(l(v, w) + M[w]))
   Fin-faire
Fin-faire
Renvoyer M[s]
```

Construction du chemin

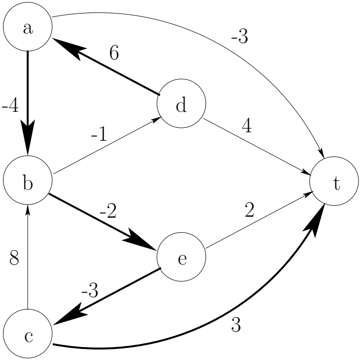
Information supplémentaire pour la reconstruction du chemin

- $\texttt{premier}[v]$, le premier sommet après v dans le chemin de longueur $M[v]$ entre v et t.
- Lorsque $M[v]$ est remplacée par $\min_{w \in V}(l(v, w) + M[w])$, $\texttt{premier}[v]$ devient le sommet w qui permet d'atteindre ce minimum.

Construction

- Soit P le graphe avec V comme ensemble de sommets et $\{(v, \texttt{premier}(v)) : v \in V\}$ comme ensemble d'arcs.
- Ce graphe est acyclique et pour tout sommet $v \in V - \{t\}$, l'unique chemin entre v et t dans P est un plus court chemin dans G.

Exemple



(a)

	a	b	c	d	e
M	-6	-2	3	0	0
`premier`	b	e	t	a	c

(b)

Routage

Une application importante des algorithmes de plus courts chemins :

Routage dans les réseaux de communications

Déterminer le chemin le plus efficace pour router les messages jusqu'à leurs destinations.

Graphe	Réseau de Communications
Sommets	Routeurs
Arcs	Lignes de communications
Longueurs	Délais

Les longueurs (délais) sont positives mais l'algorithme de Dijkstra a l'inconvénient de fonctionner **globalement** (choix de la marque minimal).

Au contraire, l'algorithme de Bellman-Ford est plutôt **local**. chaque noeud a besoin seulement de connaître les marques de ses voisins.

Amélioration

Amélioration de l'algorithme de Bellman-Ford pour le rendre

- plus efficace
- mieux adapté au problème du routage

Modification

- **Versions précédentes** : pour recalculer sa marque $M[v]$ un sommet v doit demander à tous ses voisins leurs marques.

  Si aucun voisin n'a changé de marque $\rightarrow$ calculs superflux.
- **Nouvelle version** : chaque noeud dont la marque change en informe ses voisins.
- Permet de **stopper l'algorithme plus tôt** si aucune marque ne change pendant une itération.

Amélioration

Voilà une description de cette nouvelle implémentation :

Algorithme de Bellman-Ford (V4)

Bellman-Ford-V4(G, s, t)
Fixer $M[v] = \infty$ pour tout $v \in V - \{t\}$ et $M[t] = 0$
Pour $i = 1, \ldots, n-1$ faire
    Pour $w \in V$ faire
        Si $M[w]$ a été modifié à l'itération précédente alors
            Pour tous les arcs (v, w) faire
                $M[v] = \min(M[v], l(v, w) + M[w])$
                Si la valeur de $M[v]$ change, premier$[v] = w$
            Fin-faire
    Fin-faire
    Si aucune valeur n'a changé, arrêter l'algorithme.
Fin-faire
Renvoyer $M[s]$

Problème de synchronisation

Problème de synchronisation

- En réalité, il se peut que certains noeuds envoient les notifications de changement beaucoup plus vite que d'autres.
- On doit donc utiliser une version asynchrone de l'algorithme : chaque noeud dont la marque change devient "actif" et notifie ce changement à ces voisins.

Le comportement global sera alors le suivant :

Algorithme de Bellman-Ford (V5, version asynchrone)

```
Bellman-Ford-V5(G, s, t)
Fixer M[v] = ∞ pour tout v ∈ V − {t} et M[t] = 0
Déclarer t actif et tous les autres noeuds inactifs.
Tant qu'il existe un noeud actif faire
   Choisir un noeud actif w
   Pour tous les arcs (v, w) faire
      M[v] = min(M[v], l(v, w) + M[w])
      Si cela change la valeur de M[v], alors
         premier[v] = w
         v devient actif
   Fin-faire
   w devient inactif
Fin-faire
```