Chapître XIII.
Plus courts chemins à origine unique

1. Le problème

-) donné:

graphe orienté $G = (V, A)$

pour chaque arc $(u, v) \in A$,
un poids $w(u, v)$ à valeur réelle

-) définitions:

Le **poids du chemin** $p = (v_0, v_1, \dots, v_k)$ est défini par $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$.

On définit le **poids du plus court chemin** entre u et v par
$\delta(u, v) = \min\{w(p) : p \text{ chemin entre } u \text{ et } v\}$,
s'il existe un chemin entre u et v,
et $\delta(u, v) = \infty$ sinon.

-) problème:

On souhaite trouver un plus court chemin depuis un sommet origine donné $s \in V$ vers n'importe quel sommet $v \in V$.

Remarques:

On se sert des poids pour représenter une distance, un temps, un coût, des pénalités, une déperdition ...

Il y a plusieurs autres variantes du problème des plus courts chemins, en particulier le **problème des plus courts chemins pour tout couple de sommets**.

2. Représentation des plus courts chemins

Étant donné un graphe orienté $G = (V, A)$, on maintient à jour pour chaque sommet $v \in V$ un **prédécesseur** $\pi[v]$ qui est soit un autre sommet, soit NIL.

L'algorithme de Dijkstra met à jour π de manière que le chemin des prédécesseurs partant d'un sommet v remonte à l'envers un plus court chemin entre s et v:

$$(s, \ldots, \pi[\pi[\pi[v]]], \pi[\pi[v]], \pi[v], v).$$

Le **sous-graphe de liaison**
$G_\pi = (V_\pi, A_\pi)$ induit par les valeurs π
est défini par
$V_\pi = \{v \in V : \ \pi[v] \neq \text{ NIL}\} \cup \{s\}$, et
$A_\pi = \{(\pi[v], v) \in A : \ v \in V_\pi \setminus \{s\}\}$.

Exemple:

Nous allons constater que les valeurs de π obtenues par l'algorithme de Dijkstra ont pour propriété que: après termination de l'algorithme, G_π est une arborescence des plus courts chemins.

3. Arborescence des plus courts chemin

Un arborescence des plus courts chemins enracinée en s est un sous-graphe orienté $G' = (V', A')$, où $V' \subseteq V$ et $A' \subseteq A$, tel que

1. S' est l'ensemble des sommets accessibles de s dans G,

2. G' forme une arborescence de racine s, et

3. pour tout $v \in V'$, le chemin (élémentaire) de s à v dans G' est un plus court chemin de s vers v dans G.

Remarques:

Les plus courts chemins ne sont pas obligatoirement uniques, pas plus que les arborescences de plus courts chemins.

4. Relâchement

L'algorithme de Dijkstra et
l'algorithme de Bellman-Ford
utilisent une technique principale qu'on appelle relâchement pour resoudre le problème des plus courts chemins à origine unique.

méthode:

L'algorithme diminue progressivement une borne supérieure sur le poids d'un plus court chemin pour chaque sommet,

jusqu'à ce que la borne supérieure soient égale au poids d'un plus court chemin.

5. Propriétés de bases

Lemme. *Soit $G = (V, A)$ un graphe orienté pondéré de fonction de pondération $w : A \to \mathbb{R}$. Soit $p = (v_1, v_2, \ldots, v_k)$ un plus court chemin d'un sommet v_1 à un sommet v_k et, pour tout couple d'entiers i, j quelconques tels que $1 \leq i \leq j \leq k$, soit $p_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ un sous-chemin de p allant de v_i à v_j. Alors, p_{ij} est un plus court chemin de G.*

Lemme. *Soit $G = (V, A)$ un graphe orienté pondéré de fonction de pondération $w : A \to \mathbb{R}$ et un sommet origine s. Alors, pour tous les arcs $(u, v) \in A$, on a $\delta(u, v) \leq \delta(s, u) + w(u, v)$.*

Démonstration Un plus court chemin p entre l'origine s et un sommet v a un poids inférieur ou egal à celui de n'importe quel autre chemin de s vers v. En particulier, le chemin p a un poids inférieur ou egal au chemin constitué d'un plus court chemin de l'origine s au sommet u, et de l'arc (u, v). $\square$

6. Initialisation et Relâchement

On maintient à jour un attribut $d[v]$, qui est la borne supérieure sur le poids d'un plus court chemin de l'origine s au sommet v. On initialise $d[v]$ et $\pi[v]$ pour chaque sommet v de G.

$SOURCE{-}UNIQUE{-}INITIALISATION(G, s)$

pour chaque sommet $v \in V$
  faire $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

La procédure suivante effectue une étape de relâchement sur l'arc (u, v).

$RELACHER(u, v, w)$

si $d[v] > d[u] + w(u, v)$
  alors $d[v] \leftarrow d[u] + w(u, v)$
    $\pi[v] \leftarrow u$

7. L'algorithme de Dijkstra

L'algorithme de Dijkstra résout le problème de la recherche d'un plus court chemin à origine unique pour un graphe orienté pondéré $G = (V, A)$ dans le cas où **tous les arcs ont des poids positifs ou nuls**.
On supposera donc que $w(u, v) \geq 0$ pour chaque arc $(u, v) \in A$.

$DIJKSTRA(G, w, s)$

$SOURCE\text{–}UNIQUE\text{–}INITIALISATION(G, s)$
$E \leftarrow \emptyset$
$F \leftarrow V$
tant que $F \neq \emptyset$
    faire $u \leftarrow EXTRAIRE - MIN(F)$
            $E \leftarrow E \cup \{u\}$
            pour chaque sommet $v \in Adj[u]$
                    faire $RELACHER(u, v, w)$

8. L'exécution de l'algorithme de Dijkstra

9. Validité de l'algorithme de Dijkstra

La validité de l'algorithme de Dijkstra n'est pas évidente.

Il faut se demander pourquoi chaque fois qu'un sommet u est inséré dans l'ensemble E, on a $d[u] = \delta(s, u)$.

Théorème. *Si l'on exécute l'algorithme de Dijkstra sur un graphe orienté pondéré $G = (V, A)$, avec une fonction de pondération non négative w et une origine s, alors après l'exécution, on a $d[u] = \delta(s, u)$ pour tous les sommets $u \in V$.*

Démonstration (Cormen, p. 518-520)

10. L'algorithme de Dijkstra et des poids négatifs

Contre-exemple:

Donner un graphe orienté pondéré $G = (V, A)$, avec une fonction de pondération w et une origine s, tel que il y a des arcs de poids négatifs, il n'y a pas de circuit négatif dans G et,

l'algorithme de Dijkstra ne trouve pas le poids d'un plus court chemin entre s et v, pour chaque sommet v de G.

11. Analyse du temps de l'algorithme de Dijkstra

Quelle est la rapidité de l'algorithme de Dijkstra?

analyse au pire des cas:

L'initialisation et toutes les opérations excepté $EXTRAIRE - MIN$ et $RELACHER$ s'exécutent en temps $O(n)$ au pire des cas.

L'algorithme appelle la procédure $RELACHER(u, v, w)$ au plus m fois, c'est-à-dire une fois par arc.

L'algorithme appelle $EXTRAIRE - MIN(F)$ au plus n fois.

Pour une implémentation puissante de l'algorithme de Dijkstra il nous faut une **structure de données** qui supporte les deux opérations:

-) EXTRAIRE-MIN

-) DIMINUER-CLE (l'opération RELACHER)

Comme structure de données on utilise une **file de priorité**.

-) tas binaire

-) tas binomial (Cormen, Chapître 20)

-) tas de Fibonacci (Cormen, Chapître 21)

Dans la file de priorité F la clé d'un sommet v a la valeur $d[v]$.

Théorème. *L'algorithme de Dijkstra s'exécute en temps* $O(m + n \log n)$ *au pire des cas, si la file de priorité* F *est implémenté à l'aide d'un tas de Fibonacci.*

12. L'algorithme de Bellman-Ford

L'algorithme de Bellman-Ford résout le problème de la recherche d'un plus court chemin à origine unique pour un graphe orienté pondéré $G = (V, A)$ dans le cas plus général où **les poids d'arc peuvent avoir des valeurs négatives**.

Mais s'il y a un circuit de poids négatif accessible de s dans G, le poids d'un plus court chemin n'est plus correctement défini.

L'algorithme de Bellman-Ford retourne une valeur booléenne indiquant s'il existe ou non un circuit de poids négatif accessible à partir de s. Si un tel circuit n'existe pas l'algorithme donne les plus courts chemins ainsi que leurs poids.

$BELLMAN - FORD(G, w, s)$

$SOURCE–UNIQUE–INITIALISATION(G, s)$
pour $i \leftarrow 1$ **à** $|V| - 1$
    faire pour chaque arc $(u, v) \in A$
            faire $RELACHER(u, v, w)$
pour chaque arc $(u, v) \in A$
    faire si $d[v] > d[u] + w(u, v)$
            alors retourner $FAUX$
retourner $VRAI$

13. L’exécution
de l’algorithme de Bellman-Ford