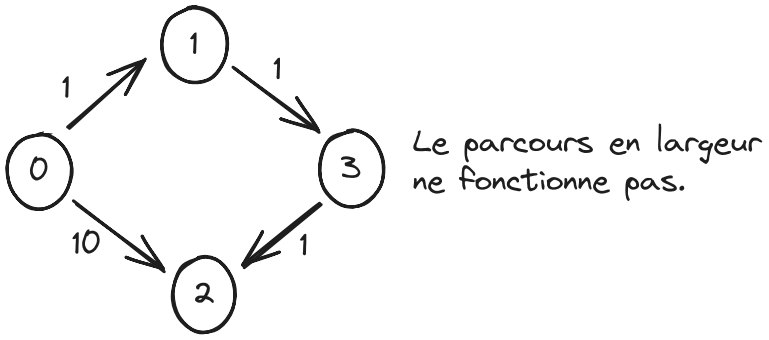
Chapitre 17 - Plus court chemin dans un graphe pondéré

Rappel : Si l'arbre est non pondéré, un parcours en largeur suffit.



I. Notion de plus court chemin

Graphe pondéré :

$G = (S, A, \omega)$ avec ω une fonction de pondération qui à une arête/arc associe son poids, étendue à tout couple de sommet en posant que $\omega(x, y) = +\infty$ si $\{x, y\} \notin A$.

Le poids d'un chemin est la somme des poids des arêtes/arcs qui le compose. On utilise aussi la notation ω.

Définition :

La distance d'un sommet x à un sommet y dans un graphe pondéré, notée $\delta(x, y) = \inf\{\omega(c) \mid c \text{ chemin de } x \text{ à } y\}$

Remarque :

La distance peut valoir $+\infty$ si y est non accessible depuis x et $-\infty$ si on a un cycle de poids négatif. On supposera pour tous nos algorithmes, qu'il n'y a pas de cycle de poids négatif.

Le chemin pour lequel la distance est absolue est appelé "plus court chemin" de x à y.

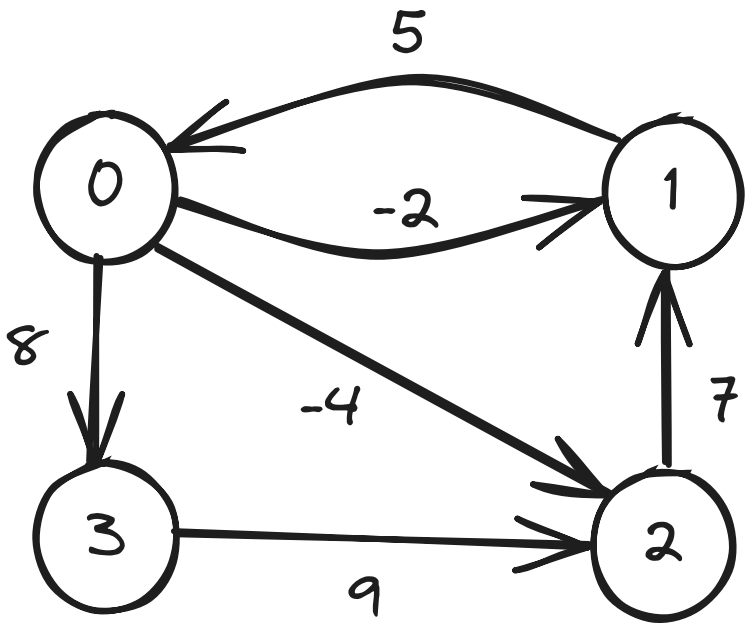
Propriété :

Soit $x \sim \ldots > c \sim \ldots > y$ un plus court chemin de x à y passant par un certain sommet c. Alors $x \sim \ldots > c$ et $c \sim \ldots > y$ sont respectivement des plus courts chemins de x à c

et de c à y.

On le prouve par l'absurde.

Matrice d'adjacence d'un graphe pondéré :



On fait comme la matrice d'ajcence sauf qu'on remplace les 1 par les poids associés et on met $+\infty$ au lieu de 0 pour signifier que deux sommets ne sont pas reliés.

On s'intéresse ici à la recherche du plus court chemin dans un arbre pondéré :

- De tous les sommets à tous les autres (départ et arrivé non fixés).
- D'un sommet de départ fixé à tous els autres.

C'est un problème d'optimisation.

II. Algorithme de Floyd-Warshall

Il est basé sur la programmation dynamique, il n'y a pas de cycle de poids strictement négatif et il résout la première variante du problème.

Principe :

On calcule, tout d'abord, les plus courts chemins de x à y sans sommet intermédiaire. Ensuite, on regarde si on peut faire mieux avec 0 comme sommet intermédiaire. On

regarde si on peut faire mieux en considérant 1 dans le chemin et ainsi de suite jusqu'à avoir regardé pour tous les sommets intermédiaires possibles (jusqu'à $|S|-1$).

On note $\delta_k(x,y)$ la distance de x à y en passant éventuellement par les sommets intermédiaires de 0 à $k-1$.

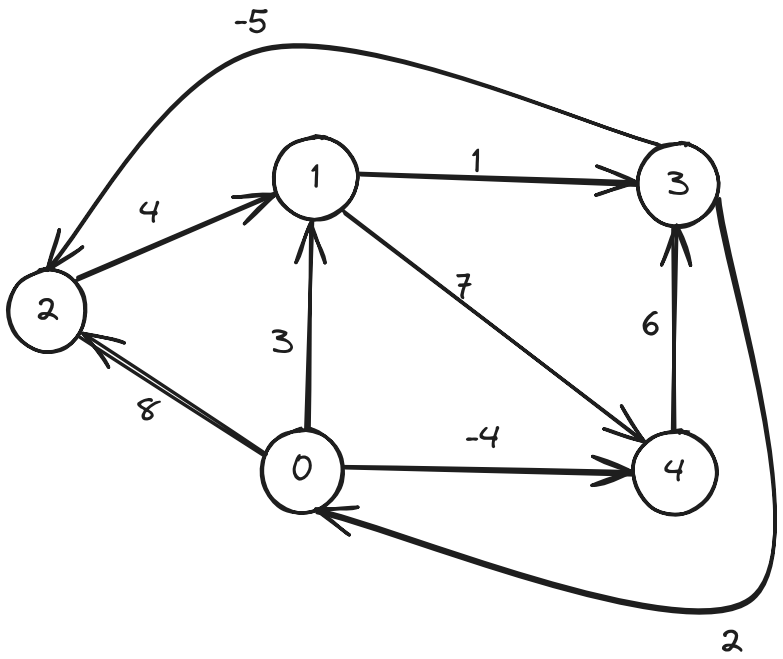
$$\begin{cases}\delta_0(x,y) = \omega(x,y) \\ \delta_{k+1}(x,y) = \min(\delta_k(x,y), \delta_k(x,k)) + \delta_k(k,y)\end{cases}$$

On implémente cette relation de récurrence avec une approche bottom-up, les $\delta_k(i,j)$ sont stockés dans les matrices M_k de taille $|S| \times |S|$.

On stocke aussi les informations nécessaires pour reconstruire les plus courts chemins. Dans une matrice P_k prédécesseur de j dans tous les plus courts chemins de i à j.

- Si on est dans le cas $\delta_{k+1}(i,j) = \delta_k(i,j)$ alors $P_{k+1}(i,j) = P_k(i,j)$.
- Sinon, dans le cas $\delta_{k+1}(i,j) = \delta_k(i,k) + \delta_k(k,j)$ alors le plus court chemin est de la forme $i \to \ldots \to k \to \ldots \to j$ donc $P_{k+1}(i,j) = P_k(k,j)$.

Exemple :



On fait alors M_0 la matrice d'adjacence du graphe en mettant ∞ quand il n'y a pas de prédécesseur et le poids du prédécesseur sinon, dans la matrice P_0, on met une croix là où il n'y a pas de prédécesseur dans la matrice d'adjacence et où il y a 0 et le numéro du prédécesseur pour le reste, lorsque l'on fait la matrice du noeud suivant, on ajoute en noeud intermédiaire le noeud précédent et adapte la matrice M_k en conséquence.

Plus court chemin de 4 à 1 :

- $\delta(4,1) = M_5(4,1) = 5$
- $4 \to 3 \to 2 \to 1$ avec $(4 = P_5(4,3), 3 = P_5(4,2), 2 = P_5(4,1))$.

Remarque :

Si la diagonale est modifiée, il y a un cycle de poids négatif, on arrête l'algorithme.

Algorithme :

Entrées : G une matrice d'adjacence d'un graphe pondéré sans cycle de poids négatif.

Code :

```
M <- copie de G
Pour k allant de 0 à |S| - 1 :
        Pour i allant de 0 à |S| - 1 :
                Pour j allant de 0 à |S| - 1 :
                        M(i)(j) <- min(M(i)(j), M(i)(k) + M(k)(j))
```

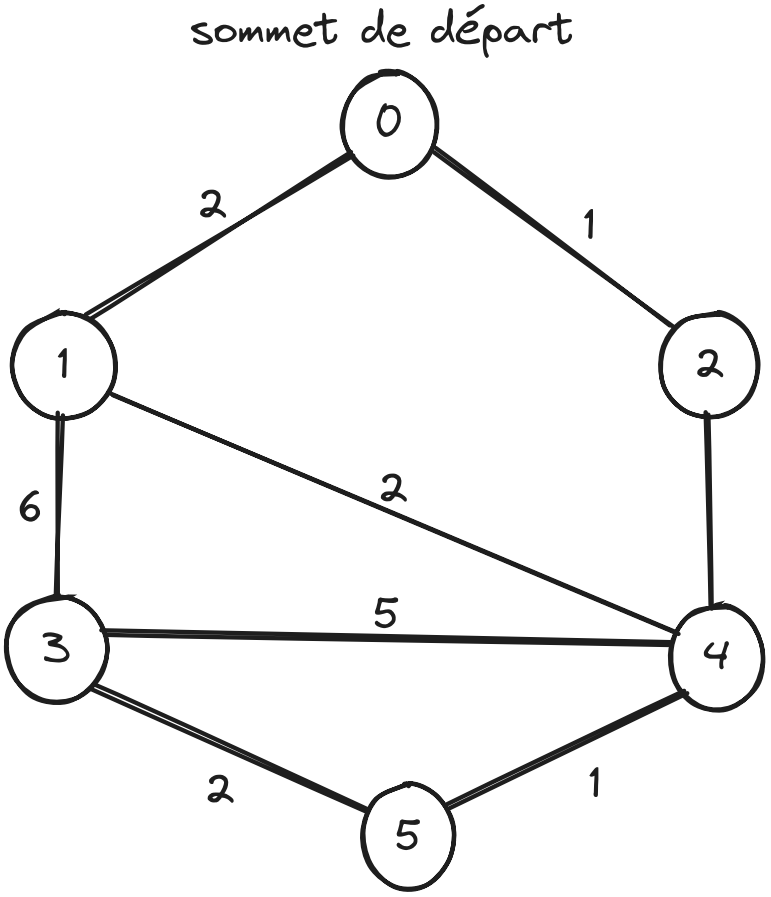
Complexité :

$$\mathcal{O}(|S|^2) + \sum_{k=0}^{|S|-1} \sum_{i=0}^{|S|-1} \sum_{j=0}^{|S|-1} \mathcal{O}(1) = \mathcal{O}(|S|^3).$$

III. Algorithme de Dijkstra

C'est un algorithme à approche gloutonne. Il ne faut que des poids positifs. Le sommet de départ est fixé.

Exemple :



0	1	2	3	4	5
0	∞	∞	∞	∞	∞
/	$2_{(0)}$	$1_{(0)}$	∞	∞	∞
/	$2_{(0)}$	/	∞	$6_{(2)}$	∞
/	/	/	$8_{(1)}$	$4_{(1)}$	∞
/	/	/	$8_{(1)}$	/	$5_{(4)}$
/	/	/	$7_{(5)}$	/	/

A chaque étape, on “fixe” le sommet pour lequel l’estimation actuelle de la distance est minimale parmi les non fixés.

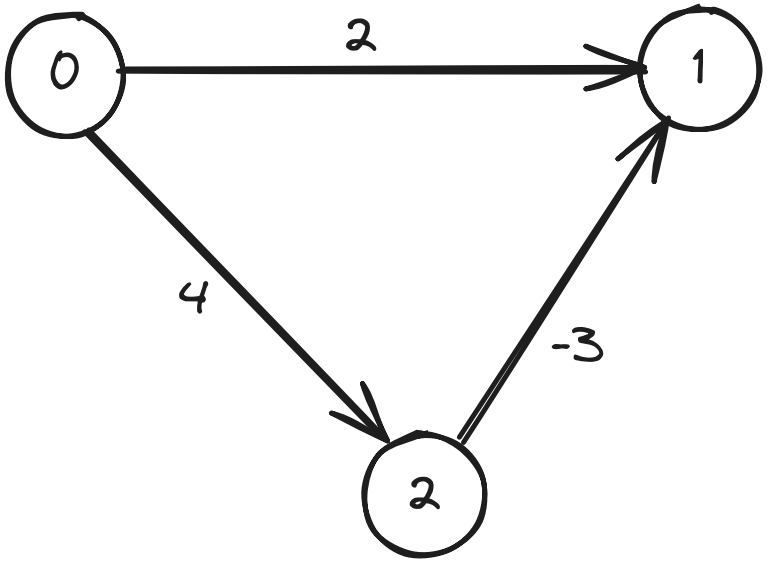
On regarde les voisins du sommet fixé, et on met à part l’estimation de la distance du voisin si prendre l’arête sommet fixé $\rightarrow$ voisin donne un meilleur chemin.

Les autres estimations restent inchangées.

Plus court chemin de 0 à 3.

- $\delta(0,3) = 7$ (estimation quand 3 est fixé)
- $0 \rightarrow 1 \rightarrow 4 \rightarrow 5 \rightarrow 3$.

Pourquoi pas de poids négatif :



1 est évaluée à 2 alors que $\delta(0,1) = 1$.

Preuve d'optimalité :

On note :

- dep le sommet de départ
- F l'ensemble des sommets fixés
- e(s) l'estimation de la distance de dep à s.

On montre l'invariant suivant :

(1) $\forall s \in F, e(s) = \delta(dep, s)$.

(2) $\forall s' \in S\backslash(F \cup \{dep\}), e(s') = \inf_{s \in F}\{S(dep, s) + \omega(s, s')\}$.

Preuve sur framagit

Algorithme :

Entrées : G, liste d'adjacence d'un graphe pondéré sans poids négatif, $dep \in S$.

Code :

```
fixes <-  {}
à_traiter <- file de priorité vide
à_traiter <- enfiler dep de priorité 0
distances <- tableau de taille |S| rempli de + ∞
distances[dep] <- 0
Tant que à_traiter non vide :
        s <- défiler un élément de à_traiter de priorité minimale
        Si s n'appartient pas à fixés :
                fixes <- fixes ∪ {s}
                Pour chaque voisin v de s :
                        Si distance[v] > distance[s] + poids de {s,v}
                                distances[v] = distance[s] + poids de
{s,v}
                                à_traiter <- enfiler v de priorité
distance[v]
```

Complexité :

En implémentant la file de priorité avec un tas-min, la complexité est optimale.

$\mathcal{O}((|S| + |A|) \times \log(|S|))$.