

Cours

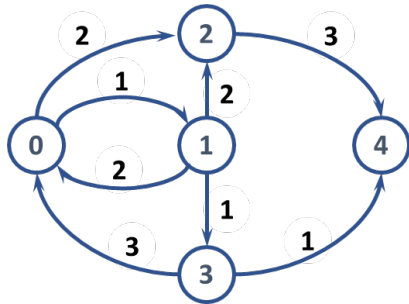
Ch. 8
Parcours de graphes

L'algorithme de Dijkstra est un parcours en largeur d'un graphe **pondéré** et orienté. Il permet de calculer l'ensemble des plus courts chemins entre un sommet vers tous les autres sommets du graphe.

Pour modéliser le graphe, on utilisera une matrice d'adjacence M pour laquelle $M_{ij} = w(i, j)$ et $w(i, j)$ représente le poids de l'arête de i vers j . Lorsqu'il n'y a pas d'arc entre deux sommets, on aura $M_{ij} = \infty$.

■ Exemple

JulesSvartz



	Colonne j , sommet d'arrivée				
Ligne i , sommet de départ	0	1	2	∞	∞
2	0	1	2	∞	∞
1	∞	∞	0	∞	3
3	3	∞	∞	0	1
∞	∞	∞	∞	∞	0

Définition Poids d'un chemin Soit un un graphe pondéré $G = (V, E, w)$ où V désigne l'ensemble des sommets, E l'ensemble des arêtes et w , la fonction poids définie par $w : E \rightarrow \mathbb{R}$ ($w(u, v)$ est le poids de l'arête de u vers v).

On appelle poids du chemin C et on note $w(C)$ la somme des poids des arêtes du chemin.

Un chemin de $u \in V$ à $v \in V$ est un plus court chemin s'il n'existe pas de chemin de poids plus petit.

Définition Distance La distance $d(u, v)$ est le poids d'un plus court chemin de u à v . On peut alors noter $d(u, v) = \inf\{w(C) | C \text{ est un chemin de } u \text{ à } v\}$.

Si v n'est pas atteignable depuis u on pose $d(u, v) = \infty$.

Propriété Sous-optimalité – Soit C un plus court chemin de u à v ainsi que u' et v' deux sommets de C . Alors le sous-chemin de C de u' à v' est aussi un plus court chemin.

Objectif Soit un graphe pondéré $G = (V, E, w)$ où V désigne l'ensemble des sommets, E l'ensemble des arêtes et w , la fonction poids.

Soit s un sommet de V . L'objectif est de déterminer la liste de l'ensemble des distances entre s et l'ensemble des sommets de V .

Pour répondre à l'objectif, on peut formuler l'algorithme de Dijkstra ainsi.

Entrées : un graphe pondéré donné par liste ou marrice d'adjacence, un sommet s du graphe

Sortie : D liste des distances entre s et chacun des sommets

Initialisation de $D : D[i] = \infty$

Initialisation de $D : D[s] = 0$

Initialisation de $T : D[i] = \text{False}$ liste des sommets traités

Initialisation d'une file de priorité avec le sommet de départ $F = \{s\}$

Tant que F n'est pas vide :

 Recherche du sommet u tel que $d[u]$ minimal parmi les sommets de F

Pour tout voisin v de u **faire** :

Si v n'est ni dans F ni dans H **alors**

 Ajouter v à F

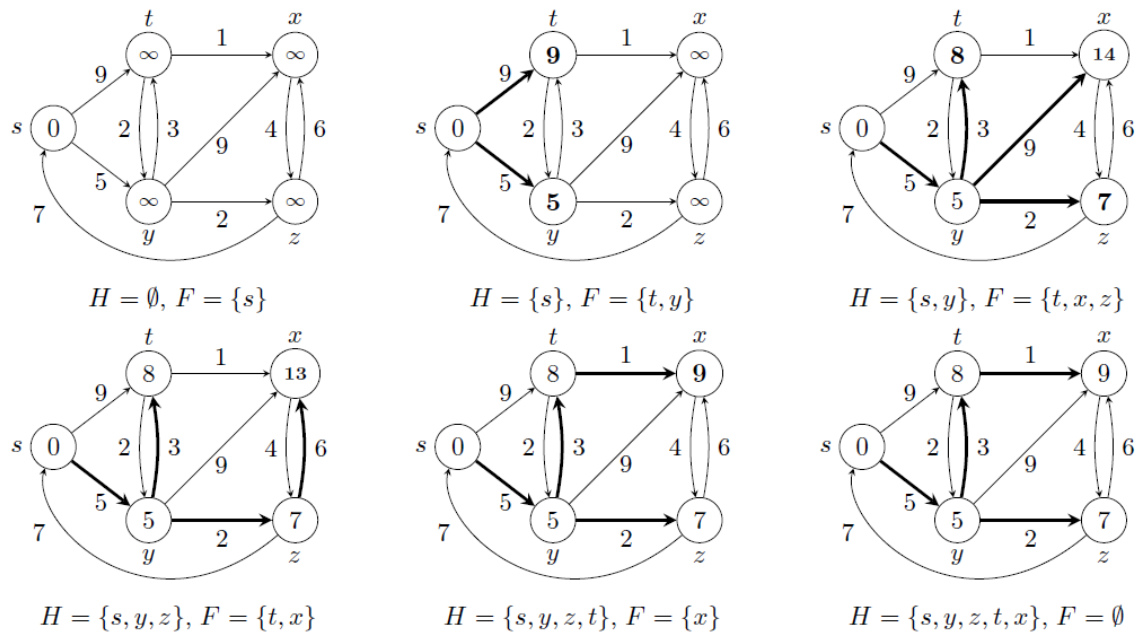
$D = \min(d[v], d[u] + w(u, v))$

 Ajouter u à H

Renvoyer D

Une des étapes qui diffère avec le parcours en largeur notamment, est l'utilisation d'une file de priorité et la recherche du sommet vérifiant $d[u]$ minimal. Cela signifie que lorsqu'on partira d'un sommet s . On déterminera alors l'ensemble des distances permettant d'atteindre les voisins de s . À l'itération suivante, on vistera alors le sommet ayant la distance la plus faible.

■ **Exemple** La figure suivante représente le déroulement de l'algorithme de Dijkstra sur un graphe à 5 sommets, depuis la source s . Pour chaque sommet u on a fait figurer la valeur $d[u]$ à l'intérieur du cercle. Les arcs en gras représentent l'évolution de la liste des prédecesseurs.



22

On peut donc commencer par implémenter une fonction `cherche_min` permettant de trouver le sommet i vérifiant $d[i]$ minimal parmi les sommets tels que `traites[i]` est faux (s'il existe un tel sommet vérifiant $d[i] < +\infty$, sinon -1 est renvoyé).

```
def cherche_min(d,traites):
    """ Renvoie le sommet i vérifiant d[i] minimal et traits[i] faux, s'il existe un tel sommet
    tel que d[i] != inf. Sinon, renvoie -1 """
    n=len(d)
    x=-1
    for i in range(n):
        if not traits[i] and d[i] != float('inf') and (x==-1 or d[x]>d[i]):
            x=i
    return x
```

On donne alors l'algorithme de Dijkstra.

```
def dijkstra_mat(G,s):
    """ G donné par matrice d'adjacence. Renvoie les poids chemins de plus petits poids depuis s/
    . """
    n=len(G)
    d = [float('inf')]*n
    d[s]=0
    traits = [False]*n
    while True:
        x=cherche_min(d,traits)
        if x==-1:
            return d
        for i in range(n):
            d[i]=min(d[i], d[x]+G[x][i])
        traits[x]=True
```

Sources

- Cours de Quentin Fortier.
- Cours de Jules Svartz, Lycée Masséna.