

Algorithme de Dijkstra

Fiche de synthèse

Important. L'algorithme de Dijkstra est hors programme : il prolonge le chapitre sur les parcours de graphes.

① PRINCIPE

Dans un graphe pondéré dont tous les poids sont positifs ou nuls, l'algorithme de Dijkstra, publié en 1959, calcule les plus courts chemins d'un sommet de départ vers tous les autres sommets. Il repose sur une propriété : un morceau d'un plus court chemin est lui-même un plus court chemin.

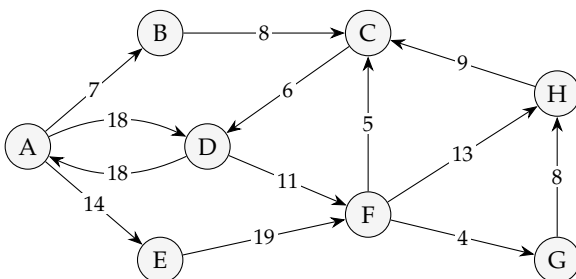
≡ Algorithme de Dijkstra.

1. Donner la distance 0 au sommet de départ et la distance $+\infty$ à tous les autres.
2. Choisir, parmi les sommets non encore traités, celui dont la distance provisoire est la plus petite.
3. Pour chacun de ses voisins, si passer par ce sommet donne une distance plus petite que celle connue, remplacer la distance du voisin et noter ce sommet comme son **prédécesseur**.
4. Recommencer à l'étape 2 jusqu'à ce que tous les sommets soient traités.

Un poids négatif fausserait les distances des sommets déjà traités : l'algorithme ne s'applique qu'aux poids positifs ou nuls. Il est utilisé par les calculateurs d'itinéraires et par le protocole de routage OSPF.

② EXEMPLE

On cherche le plus court chemin de A à H dans ce graphe orienté, dont les poids sont des distances en kilomètres.



La notation 7_A signifie « distance 7, en venant de A ». Chaque ligne indique le sommet choisi, puis les distances mises à jour.

Choisi	B	C	D	E	F	G	H
A (0)	7_A	∞	18_A	14_A	∞	∞	∞
B (7)		15_B	18_A	14_A	∞	∞	∞
E (14)		15_B	18_A		33_E	∞	∞
C (15)			18_A		33_E	∞	∞
D (18)					29_D	∞	∞
F (29)						33_F	42_F
G (33)							41_G

Le sommet H est enfin choisi avec la distance 41. En traitant D, la distance de F passe de 33 à $18 + 11 = 29$. En remontant les prédécesseurs depuis H, on obtient $H \leftarrow G \leftarrow F \leftarrow D \leftarrow A$: le plus court chemin est A, D, F, G, H, de longueur 41 km.

③ IMPLÉMENTATION EN PYTHON

Le graphe est un dictionnaire de dictionnaires : `graphe["A"]["B"]` est le poids de l'arc de A vers B.

```
graphe = {"A": {"B": 7, "E": 14, "D": 18},
          "B": {"C": 8, "C": {"D": 6}},
          "D": {"A": 18, "F": 11},
          "E": {"F": 19},
          "F": {"G": 4, "H": 13, "C": 5},
          "G": {"H": 8, "H": {"C": 9}}

def dijkstra(graphe, depart):
    dist = {s: float("inf") for s in graphe}
    dist[depart] = 0
    pred = {s: None for s in graphe}
    non_traites = list(graphe)
    while non_traites != []:
        s = min(non_traites,
                key=lambda x: dist[x])
        non_traites.remove(s)
        for voisin, poids in graphe[s].items():
            if dist[s] + poids < dist[voisin]:
                dist[voisin] = dist[s] + poids
                pred[voisin] = s
    return dist, pred

def plus_court_chemin(graphe, depart, arrivee):
    dist, pred = dijkstra(graphe, depart)
    chemin = []
    s = arrivee
    while s != None:
        chemin.append(s)
        s = pred[s]
    chemin.reverse()
    return chemin, dist[arrivee]
```

`float("inf")` est un flottant plus grand que tout nombre : il représente une distance infinie. `plus_court_chemin(graphe, "A", "H")` renvoie `(['A', 'D', 'F', 'G', 'H'], 41)`.

④ COMPLEXITÉ

Pour n sommets, chaque étape cherche le minimum en parcourant les sommets non traités : la complexité est en $O(n^2)$. Avec un tas, structure qui donne rapidement le minimum, elle descend à $O((n + m) \log n)$, où m est le nombre d'arêtes : l'algorithme traite alors des réseaux routiers de millions de sommets.

À retenir.

- Dijkstra calcule les plus courts chemins depuis un sommet, dans un graphe à poids positifs ou nuls.
- À chaque étape, il traite le sommet non traité de plus petite distance, puis met à jour ses voisins.
- On reconstitue le chemin en remontant les prédécesseurs depuis l'arrivée.
- La complexité est en $O(n^2)$ avec une recherche simple du minimum.