

Partie V – Algorithme de Dijkstra

Graphes et Algorithmes – GRAAL

Gaëtan Staquet
gaetan.staquet@ec-nantes.fr

École Centrale de Nantes – LS2N
S508

Janvier à mars 2026

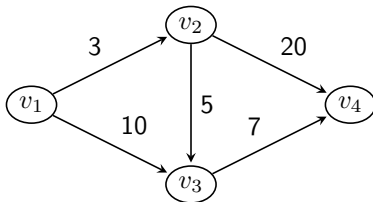
1. Algorithme de Dijkstra
2. Correction et complexité
3. Pour aller plus loin

Graphe pondéré

Définition V.1. Un **graphe pondéré** est un tuple $\mathcal{G} = (V, E, p)$ où

- ▶ (V, E) est un graphe (orienté ou non) et
- ▶ $p : E \rightarrow \mathbb{R}$ est une fonction qui associe un poids à chaque arc.

Le poids d'un chemin (v_1, \dots, v_n) est la somme des $p((v_i, v_{i+1}))$.

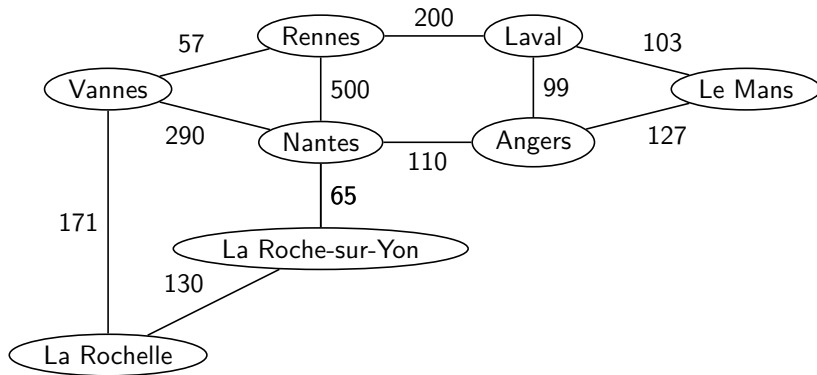


(v_1, v_3, v_4) est un chemin de longueur minimale, mais a un poids de $10 + 7 = 17$. En prenant (v_1, v_2, v_3, v_4) , on obtient un poids de $3 + 5 + 7 = 15$.

Regarder le nombre d'arcs ne suffit plus !

1. Algorithme de Dijkstra
 1. Dijkstra
2. Correction et complexité
3. Pour aller plus loin

Atteindre sa distance avec un coût minimal



Les poids sur les arcs ne sont pas nécessairement les distances. On peut prendre en compte les préférences. Par exemple, en associant un poids plus faible s'il faut prendre le bateau que de rouler dans des bouchons.

Quel est le chemin optimal de Rennes à La Roche-sur-Yon ?

Algorithme de Dijkstra – Contraintes et idées

L'**algorithme de Dijkstra** calcule le poids du chemin optimal depuis **un** sommet de départ vers **tous** les autres sommets.

Il requiert que toutes les valeurs sur le graphe pondéré soient **positives**, *i. e.*, $p : E \rightarrow \mathbb{R}^{\geq 0}$.

Algorithme de Dijkstra – Contraintes et idées

L'**algorithme de Dijkstra** calcule le poids du chemin optimal depuis **un** sommet de départ vers **tous** les autres sommets.

Il requiert que toutes les valeurs sur le graphe pondéré soient **positives**, *i. e.*, $p : E \rightarrow \mathbb{R}^{\geq 0}$.

Dans les grandes lignes :

- ▶ Notons s le sommet de départ.
- ▶ On va calculer, pour chaque sommet v , le poids du meilleur chemin de s à v .
- ▶ On explore le graphe en s'inspirant du BFS : on prend un sommet avec le plus petit poids calculé et on regarde ses successeurs.

Algorithme de Dijkstra

Nécessite : Pour tout $v \in V$, il existe un chemin qui relie s à v , $p : E \rightarrow \mathbb{R}^{\geq 0}$, $s \in V$

```
1 : procédure DIJKSTRA( $\mathcal{G} = (V, E, p), s$ )
2 :   Distances  $\leftarrow$  une fonction vide  $V \rightarrow \mathbb{R}^{\geq 0} \cup \{\infty\}$ 
3 :   pour chaque  $v \in V$  faire
4 :      $\lfloor$  Distances( $v$ )  $\leftarrow \infty$ 
5 :   Distances( $s$ ) = 0
6 :   pour chaque successeur  $t$  de  $s$  faire
7 :      $\lfloor$  Distances( $s$ )  $\leftarrow p((s, t))$ 
8 :    $S = \{s\}$ 
9 :   tant que  $|S| < |V|$  faire
10 :     $v \leftarrow$  un sommet de  $V \setminus S$  tel que Distances( $v$ ) est minimum
11 :    pour chaque successeur  $t$  de  $v$  faire
12 :       $\lfloor$  Distances( $t$ )  $\leftarrow \min\{\text{Distances}(t), \text{Distances}(v) + p((v, t))\}$ 
13 :       $S \leftarrow S \cup \{v\}$ 
14 :    retourner Distances
```

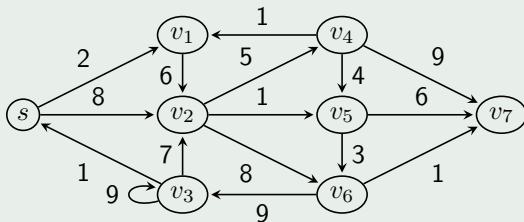
▷ Les sommets déjà explorés

Exemple au tableau.

Exercices

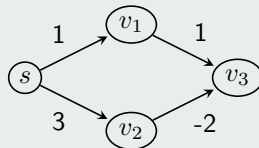
Exercice V.2.

Appliquez l'algorithme de Dijkstra sur ce graphe, en utilisant s comme sommet de départ. Déduisez-en un chemin optimal de s à v_7 .



Exercice V.3.

Appliquez l'algorithme de Dijkstra sur ce graphe, en partant de s . Pourquoi est-ce que l'algorithme ne donne pas le bon résultat ?



1. Algorithme de Dijkstra
2. Correction et complexité
3. Pour aller plus loin

Arrêt

Lemme V.4. Sur un graphe fini $\mathcal{G} = (V, E, p)$, l'algorithme de Dijkstra s'exécute en tant fini.

Ébauche de preuve. À chaque itération, on ajoute exactement un sommet dans S . Comme le graphe est fini, on finit par obtenir $|S| = |V|$. □

Correction

Exercice V.5. Fixons un graphe **fini** $\mathcal{G} = (V, E, p)$ et un sommet $s \in V$ tel qu'il existe un chemin depuis s vers n'importe quel sommet. Pour tous u et v , on note $d(u, v)$ pour le poids d'un chemin optimal de u à v . Prouvez les résultats suivants.

Proposition V.6 (Invariant de boucle). Supposons que, pour tout $v \in S$, on a $\text{Distances}(v) = d(s, v)$. Soit $t \in V$. Si $\text{Distances}(t) \neq +\infty$, alors il existe un chemin de s à t dont le poids est $\text{Distances}(t)$. De plus, $\text{Distances}(t) \geq d(s, t)$.

Proposition V.7 (Invariant de boucle). Supposons que, pour tout $v \in S$, on a $\text{Distances}(v) = d(s, v)$. Soit t le sommet choisi pendant l'itération actuelle, i.e., tel que $\text{Distances}(t)$ est minimum. Alors, $\text{Distances}(t) \leq d(s, t)$.

Lemme V.8 (Invariant de boucle). Pour tout $v \in S$, $\text{Distances}(v) = d(s, v)$.

Théorème V.9. À la fin de l'algorithme, pour tout $v \in V$, $\text{Distances}(v) = d(s, v)$.

Complexité

Avec une implémentation naïve, l'algorithme de Dijkstra est en $\mathcal{O}(|V|^2)$.

Lemme V.10. Il est possible d'implémenter l'algorithme de Dijkstra pour obtenir une complexité en $\mathcal{O}(|E| + |V| \log |V|)$.

1. Algorithme de Dijkstra
2. Correction et complexité
3. Pour aller plus loin

Pour aller plus loin

- ▶ L'algorithme de Bellman-Ford gère les poids négatifs, y compris les cycles de poids négatifs. Il est même capable de détecter ces cycles. Sa complexité est $\mathcal{O}(|V||E|)$.
- ▶ A^* n'explore pas tout le graphe. Cet algorithme se base sur une **heuristique** pour guider l'exploration. Sa complexité est $\mathcal{O}(|E| \log |V|)$, mais l'heuristique doit satisfaire certaines conditions.
- ▶ Les algorithmes précédents calculent la distance entre **UN** sommet de départ et tous les autres sommets. L'algorithme de Johnson permet de calculer les distances entre tous les sommets, à la condition qu'il n'y a pas de cycles de poids négatifs. Sa complexité est $\mathcal{O}(|V|^2 \log |V| + |V||E|)$.
- ▶ L'algorithme de Floyd (ou Floyd-Warshall) calcule les distances entre tous les sommets et gère les poids négatifs.

Pour aller plus loin

- ▶ L'algorithme de Bellman-Ford gère les poids négatifs, y compris les cycles de poids négatifs. Il est même capable de détecter ces cycles. Sa complexité est $\mathcal{O}(|V||E|)$.
- ▶ A^* n'explore pas tout le graphe. Cet algorithme se base sur une **heuristique** pour guider l'exploration. Sa complexité est $\mathcal{O}(|E| \log |V|)$, mais l'heuristique doit satisfaire certaines conditions.
- ▶ Les algorithmes précédents calculent la distance entre **UN** sommet de départ et tous les autres sommets. L'algorithme de Johnson permet de calculer les distances entre tous les sommets, à la condition qu'il n'y a pas de cycles de poids négatifs. Sa complexité est $\mathcal{O}(|V|^2 \log |V| + |V||E|)$.
- ▶ L'algorithme de Floyd (ou Floyd-Warshall) calcule les distances entre tous les sommets et gère les poids négatifs.

Dans le TP d'aujourd'hui :

- ▶ Algorithme de Floyd-Warshall pour trouver les poids des chemins optimaux.
- ▶ Chemin le plus large et élections plurinominales.