Parcours de graphe et recherche de chemin

Alexey Muranov

17 avril 2025

Table des matières

1 Généralités sur les graphes

Un *graphe orienté* est composé d'un ensemble de ses *sommets* (ou *nœuds*) et d'un ensemble des ses *arêtes orientées* (ou *liens orientés*, ou *arcs orientées*), où chaque arête orientée possède un *sommet initial* et un *sommet terminal*.

On va supposer que le sommet initial et le sommet terminal de chaque arête orientée sont implicitement déterminés, et on va définir un *graphe orienté* G comme un couple (V, E), où V est l'ensemble des *sommets* de G et E est l'ensemble des *arêtes orientées* de G.

Dire qu'une arête orientée e est *de u vers v* signifie la même chose que dire que le sommet initial de e est u, et que son sommet terminal est v.

Tant que cela ne provoque pas de confusion, on peut dire « *arête* » au lieu d'« *arête orientée* ».

Pour dire que e est une arête (orientée) de u vers v, on peut utiliser la notation « $u \xrightarrow{e} v$ ». Cette expression peut aussi être utilisée pour designer l'arête e en précisant au même temps son sommet initial et son sommet terminal.

Dire que dans un graphe orienté $G = (V, E)$, il y a des *arêtes multiples* de $u \in V$ vers $v \in V$ signifie que dans E il y a au moins deux arêtes orientées de u vers v.

Dire qu'un graphe orienté $G = (V, E)$ est *sans arêtes multiples* signifie que pour tous $u, v \in V$, dans E il n'y a qu'au plus une arête orientée de u vers v.

Dans un graphe orienté sans arêtes multiples, l'unique arête orientée de u vers v (s'il y en a) peut être notée « $u \to v$ ».

Si $G = (V, E)$ est un graphe orienté sans arête multiples, on peut identifier chaque arête orientée de E avec le couple ordonné de son sommet initial et de son sommet terminal : si $u \xrightarrow{e} v$, alors on peut considérer que $e = (u \to v) = (u, v)$.

On peut donc définir un *graphe orienté sans arêtes multiples* comme un couple d'ensembles $G = (V, E)$ tel que $E \subset V \times V$.

Dans un graphe orienté G, si u et v sont deux sommets reliés par une arête $u \xrightarrow{e} v$, on va dire que u et v sont *adjacents*, que v est un *voisin successeur* de u (ou un *successeur* de u tout court), et que u est un *voisin prédécesseur* de v (ou un *prédécesseur* de v). L'ensemble des voisins successeurs de u dans G sera noté $N^+(u)$ ou $N^+_G(u)$, et l'ensemble des voisins prédécesseurs de u dans G sera noté $N^-(u)$ ou $N^-_G(u)$.[1]

On peut aussi définir et considérer les *graphes non orientés*, où il n'y a pas de différence entre les voisins successeurs et les voisins prédécesseurs. On note $N(u)$ ou $N_G(u)$ l'ensemble des voisins de u dans un graphe non orienté G.

Un *graphe pondéré* est un graphe G muni d'une fonction qui à chaque arête de G associe un nombre réel (souvent strictement positif). Ce nombre associé à une arête peut être interprété comme le *poids*, ou la *longueur*, ou le *coût* de cette arête.

On va appeler un *chemin*[2] dans un graphe orientée une suite finie alternée non vide de sommets et d'arêtes orientées dans laquelle toute arête orientée est immédiatement précédée de son sommet initial et est immédiatement suivie de son sommet terminal.

[1] *Voisin* en anglais est *neighbour*.

[2] En *théorie de graphes*, cela s'appelle plutôt une *marche*, et un *chemin* est définie comme une marche où aucun sommet, sauf éventuellement le premier et le dernier, n'apparaît plus qu'une fois.

Un chemin $(v_0, e_1, v_1, e_2, \ldots, e_n, v_n)$ peut aussi être noté « $v_0 \overset{e_1}{\to} v_1 \overset{e_2}{\to} \cdots \overset{e_n}{\to} v_n$ ».

Dans un graphe orienté G sans arêtes multiples, un *chemin* peut être défini comme une suite finie non vide de sommets $(v_0, v_1, \ldots, v_n)$ telle que dans G il y a les arêtes orientées $(v_0 \to v_1), \ldots, (v_{n-1} \to v_n)$. Ce chemin peut aussi être noté « $v_0 \to v_1 \to \cdots \to v_n$ ».

Un chemin $(v_0, v_1, \ldots, v_n)$ dans un graphe non orienté sans arêtes multiples peut être noté « $v_0 - v_1 - \cdots - v_n$ ».

2 Représentations des graphes en informatique

En informatique, un graphe peut être représenté d'une façon *explicite* ou d'une façon *implicite*.

Dans une *représentation explicite* d'un graphe, l'ensemble des ses sommets et l'ensemble de ses arêtes sont stockés dans des *structures de données* (dans la mémoire de l'ordinateur).

Deux formats courants de représentation explicite sont une *liste d'adjacence* et une *matrice d'adjacence*. (Une liste d'adjacence n'est pas en fait une liste, mais c'est une structure de données qui contient les listes des voisins de tous les sommet du graphe.)

La bibliothèque NetworkX pour Python permet de travailler avec de graphes finis explicites.

Dans une *représentation implicite* d'un graphe, le graphe est décrit à l'aide de procédures informatiques (« fonctions ») qui permettent de générer ses sommet ou ses arêtes à partir de donnés de certain type, et qui permettent de déterminer si deux sommets sont adjacents (s'ils sont reliés par une arête). Certains usages de graphes implicites nécessitent avoir une procédure qui à partir d'un sommet u peut générer l'ensemble de ses voisins successeurs $N^+(u)$.

Les graphes explicites sont souvent variables (modifiables), mais les graphes implicites sont d'habitude immuables, car les procédures informatiques qui les décrivent d'habitude ne peuvent pas être modifiées de façon utile en pratique.

Les nombres des sommets et des arêtes dans un graphe implicite ne sont pas limités par la capacité de la mémoire disponible, contrairement aux graphes explicites, et ils peuvent même être infinis. Par exemple, il n'est pas difficile de définir un graphe implicite dont les sommets représentent tous les configurations possibles du jeu d'échecs et les arêtes représentent tous les coups légaux.

3 Parcours de graphe

Dans un algorithme de *parcours de graphe*, les sommets d'un graphe donné sont découverts et « visités » dans un certain ordre en suivant les arêtes. Plus précisément, un tel algorithme commence avec un *sommet de départ* (*sommet initial*) s et découvre et « visite » l'ensemble des sommets atteignable depuis s, en arrivant à chaque nouveau sommet par une arête orientée depuis un sommet « visité » antérieurement. En général,

chaque sommet atteignable depuis s sera « visité » ainsi une fois, mais il peut y avoir des exceptions.

Un algorithme de parcours de graphe construit implicitement ou explicitement un *arbre de chemins* qui commencent dans le sommet initial. Dans certains cas, cet arbre de chemins peut produire un *arbre couvrant* pour le graphe.

On va considérer maintenant un schéma général d'une famille d'algorithmes de parcours de graphe.

À chaque étape d'exécution de ces algorithmes on associe un ensemble des sommets *« visités »*, une file d'attente Q de chemins en attente de *développement* (ou de *prolongement*, d'*expansion*, d'*extension*), et un tableau associatif N indexé par des chemins, où $N[p]$ est l'ensemble des prolongements du chemin p par une seule arête orientée qu'il reste à explorer à l'étape actuelle.

Pour simplifier la présentation de l'algorithme, on va adopter les notations et les conventions suivantes :

(1) pour tout chemin p dans un graphe orienté $G = (V, E)$, on va noter $N_G(p)$ l'ensemble des prolongements $q = pe$ de p par une arête orientée $e \in E$,

(2) on va identifier tout sommet v avec le chemin trivial (de longueur 0) de v vers v.

```
PARCOURS DE GRAPHE
Entrée :
    un graphe orienté G = (V, E), s ∈ V, ...
début
    « visiter » le sommet s par le chemin trivial s
    N ← un nouveau tableau associatif vide
    N[s] ← N_G(s)
    Q ← {s}
    répéter :
        p ← un élément de Q
        si N[p] = ∅ :
            Q ← Q \ {p}
            si Q = ∅ :
                terminer
        sinon :
            q ← un élément de N[p] ; N[p] ← N[p] \ {q}
            si le sommet terminal de q n'a pas été « visité » :
                « visiter » le sommet terminal de q par q
                N[q] ← N_G(q)
                Q ← Q ∪ {q}
```

La file d'attente Q joue le rôle central dans ce schéma d'algorithmes. En effet : l'ordre dans lequel les sommets de G sont « visités » est déterminé par l'ordre dans lequel les chemins en attente de prolongement sont extraits de Q et, en moindre mesure, par l'ordre dans lequel les éléments de $N_G(p)$ sont considérés (extraits de $N[p]$).

Le schéma suivant est moins général mais plus simple :

PARCOURS PAR EXPANSIONS
Entrée :
un graphe orienté $G = (V, E)$, $s \in V$, ...
début
« visiter » le sommet s par le chemin trivial s
$Q \leftarrow \{s\}$
tant que $Q \neq \varnothing$ **:**
$p \leftarrow$ un élément de Q **;** $Q \leftarrow Q \setminus \{p\}$
pour chaque $q \in N_G(p)$ **:**
si le sommet terminal de q n'a pas été « visité » **:**
« visiter » le sommet terminal de q par q
$Q \leftarrow Q \cup \{q\}$

L'exécution de ce schéma peut être vue comme une succession d'*expansion* (ou de *développements*) de chemins dont les sommets terminaux on été « visités » : chaque fois un tel chemin est extrait de la file d'attente, on l'*étend* (ou *développe*) en trouvant l'ensemble de ses prolongements par une arête orientée et en plaçant dans la file d'attente ceux de ces prolongements dont les sommets terminaux ont été « visités » pour la première fois.

Par contre, dans le schéma général, après qu'un prolongement d'un chemin est placé dans la file d'attente, on prend le chemin suivant à prolonger dans la file d'attente. On peut appeler le schéma général « parcours par extensions ».

La version classique de l'algorithme de *parcours en profondeur* (« *depth-first search* » en anglais) peut être réalisée suivant le schéma général, mais pas suivant le schéma de « par expansions ».

Si l'objectif n'est pas d'atteindre tous les sommets atteignables depuis le sommet de départ, mais seulement de découvrir un chemin « convenable » vers un des sommets « convenables », on peut ajouter des commandes de terminaison conditionnelle pour terminer l'algorithme dès que on trouve qu'un des chemins examinés est « convenable ».

La présentation des algorithmes de parcours de graphes en termes des ensemble de chemins est bien adaptée à des raisonnements et à des représentations graphiques. Cependant, en pratique, dans le code, les chemins d'habitude n'apparaissent qu'implicitement.

Une façon assez simple de stocker un arbre de chemins dans la mémoire est sous la forme d'un tableau associatif (dictionnaire) qui à tout sommet de tout chemin associe l'arrête orientée précédente sur ce chemin. Cette forme de stockage est possible autant qu'aucuns deux chemins différents de l'arbre n'ont le même sommet terminal. Si le graphe en question n'a pas d'arêtes multiples, alors au lieu de l'arête orientée précédente, on peut stocker le sommet précédent (qui est un des prédécesseurs du sommet donné).

4 Parcours en profondeur

Si la file d'attente Q dans le schéma général de la section 3 fonctionne comme une *pile*, c'est-à-dire, selon la règle « dernier arrivé – premier sorti » (« *LIFO* » en anglais pour « *Last In – First Out* »), alors l'algorithme obtenu est dit *parcours en profondeur* (« *depth-first search* » en anglais).

Parcours en profondeur admet une réalisation récursive particulièrement simple. En voici un schéma, en utilisant les notations et les conventions de la section 3 :

PARCOURS EN PROFONDEUR SOUS UNE FORME RÉCURSIVE
Entrée :
un graphe orienté $G = (V, E)$, $s \in V$, ...
début
définition d'une procédure récursive RDFS(p) **:**
« visiter » le sommet terminal de p par p
pour chaque $q \in N_G(p)$ **:**
si le sommet terminal de q n'a pas été « visité » **:**
exécuter RDFS(q)
exécuter RDFS(s)

Ici la pile Q n'apparaît qu'implicitement, cachée dans la *pile d'exécution*.

5 Parcours en largeur

Si la file d'attente Q dans le schéma général de la section 3 fonctionne comme une simple *file d'attente*, c'est-à-dire, selon la règle « premier arrivé – premier sorti » (« *FIFO* » en anglais pour « *First In – First Out* »), alors l'algorithme obtenu est dit *parcours en largeur* (« *breadth-first search* » en anglais).

Proposition. *Parcours en largeur visite chaque sommet atteignable depuis le sommet de départ par un chemin le plus court possible, si la longueur du chemin est entendue comme son nombre d'arêtes orientées.*

Remarque. L'algorithme de parcours en largeur trouve les plus court chemins depuis le sommet de départ vers les sommets atteignables *si la longueur du chemin est entendue comme son nombre d'arêtes*. Or, le *problème de plus court chemin* se pose souvent dans des graphes pondérés, ou différents arêtes orientées peuvent avoir différentes longueurs. Pour résoudre ce problème, des algorithmes un peu plus complexes sont utilisés, tels que l'*algorithme de Dijkstra* ou l'*algorithme A**.

6 Recherche de chemin

L'objectif d'un algorithme de *recherche de chemin* dans un graphe orienté est de trouver certains chemins entre certains sommets du graphe. Il y a différents types d'algorithmes de recherche de chemin, dont certains peuvent être considérés des cas particuliers des algorithmes de parcours de graphe.

Voici quelques exemples de problèmes qui peuvent être résolus par différents algorithmes de recherche de chemin :

(1) trouver un chemin « convenable » d'un *sommet de départ* vers chaque sommet atteignable depuis le sommet de départ,

(2) trouver un chemin « convenable » d'un *sommet de départ* vers chaque *sommet d'arrivée*,

(3) trouver un chemin « convenable » d'un *sommet de départ* vers un *sommets d'arrivée* « convenable ».

Pour de certains problèmes, un chemin « convenable » est tout simplement le premier chemin découvert. Dans le *problème de plus court chemin*, un chemin « convenable » est un chemin le plus court.

Plusieurs algorithmes de recherche de chemin au cours d'exécution construisent un arbre de chemins commençant dans le sommet de départ jusqu'à ce qu'ils trouvent les chemins désirés.

7 Construction d'un arbre de chemins

Voici un schéma de parcours de graphe pour construire un arbre de chemins d'un sommet de départ s vers tous les sommets atteignables depuis s dans un graphe orienté G sans arêtes multiples. Le résultat est présenté sous la forme d'un tableau associatif p indexé par les sommets atteignables depuis s, où $p[v]$ représente le sommet qui **p**récède v sur un chemin trouvé de s à v.

```
RECHERCHE DE CHEMINS
Entrée : N_G^+ : V → V, s ∈ V
Sortie :
    un arbre de chemins de s vers tous les sommets atteignables depuis s
    sous la forme d'un tableau des prédécesseurs
début
    p ← un nouveau tableau associatif vide
    p[s] ← une valeur sentinelle (•)
    N ← un nouveau tableau associatif vide
    N[s] ← N_G^+(s)
    Q ← {s}
    répéter :
        u ← un élément de Q
        si N[u] = ∅ :
            Q ← Q \ {u}
            si Q = ∅ :
                renvoyer p
        sinon :
            v ← un élément de N[u] ; N[u] ← N[u] \ {v}
            si « p[v] » n'est pas défini :
                p[v] ← u
                N[v] ← N_G^+(v)
                Q ← Q ∪ {v}
```

Voici un schéma moins général mais plus simple :

Recherche de chemins par expansions
Entrée : $N_G^+ : V \to V$, $s \in V$
Sortie :
un arbre de chemins de s vers tous les sommets atteignables depuis s sous la forme d'un tableau des prédécesseurs
début
$p \leftarrow$ un nouveau tableau associatif vide
$p[s] \leftarrow$ une *valeur sentinelle* ($\bullet$)
$Q \leftarrow \{s\}$
tant que $Q \neq \varnothing$ **:**
$u \leftarrow$ un élément de Q **;** $Q \leftarrow Q \setminus \{u\}$
pour chaque $v \in N_G^+(u)$ **:**
si « $p[v]$ » n'est pas défini **:**
$p[v] \leftarrow u$
$Q \leftarrow Q \cup \{v\}$
renvoyer p

8 Recherche « le meilleur en premier »

Soient $G = (V, E)$ un graphe orienté, $s \in V$ un sommet de départ et $T \subset V$ un ensemble de sommets d'arrivée.

Voici un schéma général d'algorithmes dits *le meilleur en premier* (« *best-first* » en anglais) qui cherchent un chemin de s à un sommet dans T. Ici p est un tableau indexé par des sommets du graphe G, où $p[v]$ représente le sommet qui **p**récède v sur un chemin connu de s à v.

Recherche « le meilleur en premier »
Entrée : $N_G^+ : V \to V$, $s \in V$, $T \subset V$, ...
début
$p \leftarrow$ un nouveau tableau associatif vide
$p[s] \leftarrow$ une *valeur sentinelle* ($\bullet$)
$Q \leftarrow \{s\}$
tant que $Q \neq \varnothing$ **:**
$u \leftarrow$ un élément le *plus prometteur* de Q **;** $Q \leftarrow Q \setminus \{u\}$
si $u \in T$ **:**
renvoyer (p, u, Q)
pour chaque $v \in N_G^+(u)$ **:**
si « $p[v]$ » n'est pas défini ou qu'il y a une *raison suffisante* pour changer **:**
$p[v] \leftarrow u$
$Q \leftarrow Q \cup \{v\}$
renvoyer $(p, -, \varnothing)$

9 Recherche purement heuristique

Supposons qu'on dispose d'une fonction $h\colon V \to \mathbf{R}_+$ qui donne, pour chaque sommet v, une estimation du travail minimal suffisant pour construire (algorithmiquement) un chemin depuis v jusqu'à un sommet dans T. Par exemple, $h(v)$ peut être une estimation de la longueur d'un le plus courte chemin de v à un des sommets dans T, où la *longueur* est entendue comme le nombre d'arêtes orientées.

Dans ce cas on peut utiliser h pour tenter de trouver un chemin de $s \in V$ à un sommet de $T \subset V$ plus rapidement qu'avec un parcours de graphe « en aveugle ». Il suffit d'effectuer la recherche « le meilleur en premier », où on considère les éléments avec la plus petite valeur de h comme les plus prometteurs.

Une telle fonction h qu'on utilise en espérant qu'elle rendra l'algorithme plus efficace (sans en avoir certitude et parfois sans être sûr que h possède les propriétés requises) est dite *heuristique*.

Voici le schéma de recherche de chemin guidée par une heuristique h :

```
Recherche purement heuristique
Entrée : N_G^+ : V → V, s ∈ V, T ⊂ V, h : V → R_+
début
    p ← un nouveau tableau associatif vide
    p[s] ← •
    Q ← {s}
    tant que Q ≠ ∅ :
        u ← un élément de Q avec la plus petite valeur de h ; Q ← Q \ {u}
        si u ∈ T :
            renvoyer (p, u, Q)
        pour chaque v ∈ N_G^+(u) :
            si « p[v] » n'est pas défini :
                p[v] ← u
                Q ← Q ∪ {v}
    renvoyer (p, −, ∅)
```

Remarque. Ce schéma s'applique aussi lorsque h prend les valeurs dans un n'importe quel ensemble totalement ordonné (à la place de $\mathbf{R}_+$).

Remarque. Cet algorithme de recherche de chemin est parfois appelé *le meilleur en premier* tout simplement ou *le meilleur en premier glouton* (« *greedy best-first* » en anglais), mais les deux appellations semblent inopportunes. D'autres algorithmes de recherche de chemin peuvent être vus comme *le meilleur en premier*, incluant les algorithmes de Dijkstra et A*, et on peut argumenter que tout algorithme *le meilleur en premier* est *glouton* par définition.

Peut-être on peut appeler cet algorithme *impatient*, pour le distinguer de ceux qui utilisent des estimations heuristiques en essayant d'« optimiser » les chemins construits.

10 Problème de plus court chemin

Soit $G = (V, E)$ un graphe orienté *pondéré* par une fonction $c\colon E \to \mathbf{R}_+$, qui à chaque arête orientée e associe son *poids* $c(e) \geqslant 0$. La valeur $c(e)$ peut être interprétée comme la *longueur* de e ou comme le *coût* d'utilisation de e.

Pour simplifier la notation, on va utiliser l'écriture « $c(x, y)$ » comme synonyme de « $c(x \to y)$ ».

Pour un chemin $x_0 \to \cdots \to x_n$, on va définir son *coût* comme la somme des coûts de ses arêtes $(x_0 \to x_1), \ldots, (x_{n-1} \to x_n)$, et on va le noter « $c(x_0 \to \cdots \to x_n)$ » ou « $c(x_0, \ldots, x_n)$ » :

$$c(x_0, \ldots, x_n) = c(x_0, x_1) + \cdots + c(x_{n-1}, x_n).$$

En particulier, pour tout $x \in V$, le coût du chemin trivial de x à x est 0 : $c(x) = 0$.

Si le graphe G est infini (mais toujours localement fini), on va supposer que pour tout sommet x et pour tout réel positif r, il n'existe qu'un nombre fini de sommets atteignable depuis x par un chemin de coût $\leqslant r$. Vu que G est localement fini, cette condition est équivalente à ce qu'il n'y ait pas de chemins infinis de coût fini.

Étant donner un sommet de départ $s \in V$ (sommet source, sommet origine, sommet initial) et un ensemble de sommets d'arrivée $T \subset V$ (sommets cibles, sommets terminaux, sommets finaux), on peut considérer le problème de trouver un chemin optimale de s à un $t \in T$, c'est-à-dire un chemin du moindre coût parmi tous les chemins de s à un sommet de T. Comme le coût d'un chemin peut être interprété comme sa longueur, ce problème est parfois appelé le *problème de plus court chemin*.

Le problème de plus court chemin peut être résolu en utilisant des techniques de *parcours de graphe*, de *programmation par contraintes*, ou d'*optimisation linéaire*. Ces techniques différentes donnent lieu à des algorithmes similaires.

11 Construction d'un arbre de chemins des moindres coûts

Soient $G = (V, E)$ un graphe orienté sans arêtes multiples, $c\colon E \to \mathbf{R}$ une fonction qui définit les coûts des arêtes de G, et $s \in V$ un sommet de départ.

On va identifier chaque arête orienté avec avec le couple ordonné de son sommet initial et de son sommet terminal.

Voici un schéma d'algorithme pour construire un arbre de chemins des moindres coûts d'un sommet de départ s vers tous les sommets atteignables depuis s dans G, à la condition *qu'il n'y a pas de chemins en boucle de coût strictement négatif*. Ici d et p sont deux tableaux indexés par des sommets du graphe G, où $d[v]$ représente le plus petit coût parmi les chemins connus de s à v (la **d**istance accumulée sur ce chemin si le coût est la longueur), et $p[v]$ représente le sommet qui **p**récède v sur un chemin connu de s à v du moindre coût.

Recherche de chemins des moindres coûts
Entrée : $N_G^+ : V \to V$, $s \in V$, $c : E \to \mathbf{R}_+$
Sortie :
un arbre de chemins les plus courts de s vers tous les sommets atteignables sous la forme d'un tableau des prédécesseurs
et le tableau des distances des s vers les sommets atteignables
début
$p, d \leftarrow$ deux nouveaux tableaux associatifs vides
$p[s],\ d[s] \leftarrow \bullet,\ 0$
$Q \leftarrow \{s\}$
tant que $Q \neq \varnothing$ **:**
$u \leftarrow$ un élément de Q **;** $Q \leftarrow Q \setminus \{u\}$
pour chaque $v \in N_G^+(u)$ **:**
si « $d[v]$ » n'est pas défini ou $d[v] > d[u] + c(u, v)$ **:**
$p[v],\ d[v] \leftarrow u,\ d[u] + c(u, v)$
$Q \leftarrow Q \cup \{v\}$
renvoyer (p, d)

Si l'algorithme se termine et renvoie un couple (p, d), alors pour tout v atteignable depuis s, $d[v]$ est le coût d'un chemin optimal de s à v, et ce chemin peut être extrait de p comme $(s, \ldots, p[p[v]], p[v], v)$.

Théorème. *Si le graphe G est fini et n'a pas de chemins en boucle de coût strictement négatif, alors cet algorithme trouve des chemins optimaux vers tous les sommets atteignables depuis s.*

12 Algorithme de Dijkstra

On va confondre l'*algorithme de Dijkstra* classique avec sa version qui s'appelle en anglais « *uniform-cost search* » (*recherche en coût uniforme*). La version classique de l'algorithme de Dijkstra suppose que le graphe est fini et donné de manière explicite, alors que la version UCS peut être appliquée à des graphes implicites, éventuellement infinis.

Soient $G = (V, E)$ un graphe orienté sans arêtes multiples, $c : E \to \mathbf{R}_+$ une fonction qui définit les coûts des arêtes de G, $s \in V$ un sommet de départ.

On va identifier chaque arête orienté avec avec le couple ordonné de son sommet initial et de son sommet terminal.

Voici une version de l'algorithme de Dijkstra qui cherche des chemins optimaux de s à tous les sommet atteignables depuis s. Ici d et p sont deux tableaux indexés par des sommets du graphe G, où $d[v]$ représente le plus petit coût parmi les chemins connus de s à v (la **d**istance accumulée sur ce chemin si le coût est la longueur), et $p[v]$ représente le sommet qui **p**récède v sur un chemin connu de s à v du moindre coût.

DIJKSTRA – RECHERCHE DE CHEMINS LES PLUS COURTS
Entrée : $N_G^+ \colon V \to V$, $s \in V$, $c \colon E \to \mathbf{R}_+$
début
    $p, d \leftarrow$ deux nouveaux tableaux associatifs vides
    $p[s],\ d[s] \leftarrow \bullet,\ 0$
    $Q \leftarrow \{s\}$
    tant que $Q \neq \varnothing$ **:**
        $u \leftarrow$ un élément de Q avec la plus petite valeur de d **;** $Q \leftarrow Q \setminus \{u\}$
        pour chaque $v \in N_G^+(u)$ **:**
            si « $d[v]$ » n'est pas défini ou $d[v] > d[u] + c(u, v)$ **:**
                $p[v],\ d[v] \leftarrow u,\ d[u] + c(u, v)$
                $Q \leftarrow Q \cup \{v\}$
    renvoyer (p, d)

Soit $T \subset V$ un ensemble de sommets d'arrivée. Voici une version de l'algorithme de Dijkstra qui cherche un chemin optimal de s à un des sommets dans T les plus proches depuis s :

DIJKSTRA – RECHERCHE D'UN SOMMET LE PLUS PROCHE
Entrée : $N_G^+ \colon V \to V$, $s \in V$, $T \subset V$, $c \colon E \to \mathbf{R}_+$
début
    $p, d \leftarrow$ deux nouveaux tableaux associatifs vides
    $p[s],\ d[s] \leftarrow \bullet,\ 0$
    $Q \leftarrow \{s\}$
    tant que $Q \neq \varnothing$ **:**
        $u \leftarrow \operatorname{best} Q$ **;** $Q \leftarrow Q \setminus \{u\}$
        si $u \in T$ **:**
            renvoyer (p, d, u, Q)
        pour chaque $v \in N_G^+(u)$ **:**
            si « $d[v]$ » n'est pas défini ou $d[v] > d[u] + c(u, v)$ **:**
                $p[v],\ d[v] \leftarrow u,\ d[u] + c(u, v)$
                $Q \leftarrow Q \cup \{v\}$
    renvoyer $(p, d, -, \varnothing)$

Ici $\operatorname{best} Q$ est un élément $u \in Q$ tel que :

(1) la valeur $d[u]$ est la plus petite possible, et

(2) $u \in T$ si possible avec la condition précédente.

Théorème. *L'algorithme de Dijkstra est correct : le chemin trouvé est optimal.*

13 Algorithme A*

D'habitude l'*algorithme A** est présenté dans le contexte des graphes pondérés avec les coûts d'arêtes positifs. Ici on ne va pas supposer que les coût soient positifs car cette restriction ne semble pas être particulièrement utile pour l'analyse de l'algorithme.

Soient donc $G = (V, E)$ un graphe orienté sans arêtes multiples, $c\colon E \to \mathbf{R}$ une fonction qui définit les coûts des arêtes de G, $s \in V$ un sommet de départ et $T \subset V$ un ensemble de sommets d'arrivée.

On va identifier chaque arête orienté avec avec le couple ordonné de son sommet initial et de son sommet terminal.

Pour deux sommets $x, y \in V$, un *chemin optimal* de x à y est un chemin de x à y du coût minimal.

S'il existe un chemin de x à y, alors certaines conditions sur le graphe et sur la fonction coût c peuvent garantir l'existence d'un chemin optimal de x à y. Cependant, en toute généralité, on peut trouver des exemples où il y a des chemins de x à y mais il n'y en a aucun qui soit optimal. Par exemple, lorsqu'il y a des arête à coût négatif, et qu'il y a un chemin en boucle de coût négatif, alors pour les coûts des chemins entre certains sommets ne seront minorés par aucun nombre réel (négatif).

Dans ce qui suit, on va supposer que pour tout couple de sommets $x, y \in \mathbf{R}$ et pour tout $C \in \mathbf{R}$, le nombre des chemins de x à y du coût inférieur à C est fini. Ainsi, en particulier, s'il existe un chemin de x à y, alors il existe un chemin optimal de x à y.

Pour tout couple de sommets $x, y \in V$, posons $\bar{c}(x, y)$ le coût d'un chemin optimal de x à y si un tel chemin existe. S'il n'y a aucun chemin de x à y, on va déclarer la valeur de « $\bar{c}(x, y)$ » non définie. (Il peut aussi être pratique de poser $\bar{c}(x, y) = \infty$ s'il n'y a pas de chemins de x à y.)

Posons aussi

$$\begin{aligned}
\bar{c}(x, Y) &= \min\{\, \bar{c}(x, y) \mid y \in Y \,\} && \text{pour } x \in V,\ Y \subset V, \\
\bar{c}(X, y) &= \min\{\, \bar{c}(x, y) \mid x \in X \,\} && \text{pour } X \subset V,\ y \in V, \\
\bar{c}(X, Y) &= \min\{\, \bar{c}(x, y) \mid x \in X,\ y \in Y \,\} && \text{pour } X \subset V,\ Y \subset V.
\end{aligned}$$

Considérons la fonction $h\colon V \to \mathbf{R}$ définie ainsi :

$$h(x) = \bar{c}(x, T).$$

Supposons que toute valeur $h(x)$ peut être calculée de manière assez efficace en connaissant le sommet x, sans être obligé à trouver d'abord un chemin optimal de x à T. Alors on peut se servir de cette h pour trouver efficacement un chemin optimal de s à T : si $s \notin T$, on choisie un successeur v de s tel que $h(s) = c(s, v) + h(v)$ (s'il y en a plusieurs, on peut choisir celui avec la plus petite valeur de « $h(v)$ »), ce v sera le successeur de s sur un chemin optimal de s à T, et on continue ainsi en cherchant un chemin optimal de v à T.

En pratique, pour calculer exactement la valeur $\bar{c}(x, T)$, on peut être obligé à trouver d'abord un chemin optimal de x à T.

Supposons maintenant qu'on peut au moins calculer efficacement les valeurs d'une fonction $h\colon V \to \mathbf{R}$ qui satisfait les conditions suivantes :

(1) $h(t) = 0$ pour tout $t \in T$, et

(2) $h(x) \leqslant \bar{c}(x, T)$ pour tout $x \in V$.

Alors on peut modifier l'algorithme de Dijkstra pour profiter de cette information supplémentaire. Ainsi on obtient l'*algorithme A**, qui est un algorithme de recherche de chemin « guidé » par une fonction $h\colon V \to \mathbf{R}_+$ qui, pour chaque sommet dans V, donne un minorant du coût des chemins depuis ce somment vers les sommet d'arrivée.

Remarque. Une telle fonction h utilisée dans l'algorithme A* est couramment appelée une *fonction heuristique*, ce qui laisse entendre que le choix de h est en une partie arbitraire ou que ses propriétés ont été admises sans démonstration. Pourtant, en général on considère des fonctions avec des propriétés bien établies, ou on parle de fonctions inconnues ou arbitraires.

```
A*
Entrée : N_G^+ : V → V, s ∈ V, T ⊂ V, c : E → R, h : V → R
début
    p, d ← deux nouveaux tableaux associatifs vides
    p[s], d[s] ← •, 0
    Q ← {s}
    tant que Q ≠ ∅ :
        u ← best Q ; Q ← Q \ {u}
        si u ∈ T :
            renvoyer (p, d, u, Q)
        pour chaque v ∈ N_G^+(u) :
            si « d[v] » n'est pas défini ou d[v] > d[u] + c(u, v) :
                p[v], d[v] ← u, d[u] + c(u, v)
                Q ← Q ∪ {v}
    renvoyer (p, d, −, ∅)
```

Ici best Q est un élément $u \in Q$ tel que :

(1) la valeur $d[u] + h(u)$ est la plus petite possible, et

(2) $u \in T$ si possible avec la condition précédente.

On dit que l'algorithme A* est un algorithme de recherche de chemin *informé*, alors que l'algorithme de Dijkstra est un algorithme *non informé*. L'algorithme de recherche heuristique est également informé.

L'algorithme de Dijkstra est le cas particulier de l'algorithme A* avec la fonction « guide » nulle ($h(x) = 0$ pour tout $x \in V$).

Notons que rien n'empêche d'exécuter l'algorithme A* avec une fonction $h\colon V \to \mathbf{R}$ quelconque.

Définition. Une fonction « guide » $h\colon V \to \mathbf{R}$ est dite *admissible* pour l'algorithme A* si et seulement si

(1) h est constante sur T, et

(2) $h(x) \leqslant \bar{c}(x, t) + h(t)$ pour tous $x \in V$ et $t \in T$.

Théorème. *Si h est admissible, alors le chemin trouvé par l'algorithme A* est optimal.*

Définition. Une fonction « guide » $h\colon V \to \mathbf{R}_+$ est dite *cohérente* si et seulement si

(1) h est constante sur T, et

(2) $h(x) \leqslant c(x,y) + h(y)$ pour toute arête orientée $(x,y) \in E$.[3]

Proposition. *Si h est cohérente, alors elle est admissible.*

Proposition. *Si h est cohérente, alors l'algorithme A* n'appelle jamais la fonction N_G^+ plus d'une fois pour un même sommet.*

Remarque. Les définitions habituelles de l'*admissibilité* et de la *cohérence* de h sont plus restrictives que celles données ici : on suppose d'habitude que c et h sont à valeurs positives et que $h(t) = 0$ pour tout $t \in T$.

Supposons maintenant que la fonctions coût c est à valeurs strictement positives. Dans ce cas l'algorithme de Dijkstra trouve un chemin optimal de s à un sommet de T après avoir appliqué la fonction N_G^+ à chaque sommet x tel que $\bar{c}(s,x) < \bar{c}(s,T)$.

Soit $h\colon V \to \mathbf{R}_+$ une fonction « guide » à valeurs positives et telle que $h(t) = 0$ pour tout $t \in T$.

Si une telle h est cohérente, alors l'exécution de l'algorithme A* sera au moins aussi efficace que l'exécution de l'algorithme de Dijkstra dans le sens qu'à chaque appel de la fonction N_G^+ dans l'algorithme A* corresponde un appel de N_G^+ dans l'algorithme de Dijkstra.

Par contre, on peut fabriquer un graphe avec un choix d'un sommet de départ et d'un sommet d'arrivée et une fonction h admissible (mais incohérente) où l'algorithme de Dijkstra sera plus efficace que A*. En effet, pour une estimation h incohérent, avant de trouver le chemin optimal, l'algorithme A* peut « se perdre » en repassant par les mêmes sommets plusieurs fois.

14 Comparaison des algorithmes de Dijkstra et A* sur un exemple

Soit $G = (V, E)$ un graphe non orienté avec l'ensemble de sommets

$$V = \{a, b, c, d, e, f, g\},$$

où a, b, c, d, e, f, g sont deux-à-deux distincts, et avec l'ensemble d'*arêtes non orientées*

$$E = \{[a,d], [a,e], [a,f], [b,d], [b,e], [b,g], [c,d], [c,e], [c,f], [f,g]\}.$$

La notation « $[x,y]$ » ici signifie une arête non orientée entre x et y, et donc $[x,y]$ est la même arête non orientée que $[y,x]$, et elle corresponde à deux arêtes orientées (x,y) et (y,x).

[3] À comparer avec l'inégalité triangulaire dans les *espaces quasimétriques*.

Soit la fonction $c\colon E \to \mathbf{R}_+$ définie par le tableau suivant :

$[x,y]$	$[a,d]$	$[a,e]$	$[a,f]$	$[b,d]$	$[b,e]$	$[b,g]$	$[c,d]$	$[c,e]$	$[c,f]$	$[f,g]$
$c(x,y)$	2	1	4	1	3	4	5	5	1	2

Le graphe G muni de la fonction c est un *graphe pondéré*, où la fonction c définit les *poids* des arêtes. On va interpréter le poids d'une arête non orientée $[x,y]$ comme son coût, et ainsi comme le coût commun des arêtes orientées (x,y) et (y,x).

On va chercher un chemin optimal de a à g par l'algorithme de Dijkstra. Voici la trace de son exécution :

		a	b	c	d	e	f	$\boldsymbol{g}$
1	$a : (\bullet, 0)$	$\boxed{(\bullet,0)}$	–	–	–	–	–	–
2	$e : (a,1)$	⋮	–	–	$(a,2)$	$\boxed{(a,1)}$	$(a,4)$	–
3	$d : (a,2)$	⋮	$(e,4)$	$(e,6)$	$\boxed{(a,2)}$	⋮	⋮	–
4	$b : (d,3)$	⋮	~~$(e,4)$~~ $\boxed{(d,3)}$	$(e,6)$	⋮	⋮	⋮	–
5	$f : (a,4)$	⋮	⋮	⋮	⋮	$\boxed{(a,1)}$	$\boxed{(a,4)}$	$(b,7)$
6	$c : (f,5)$	⋮	⋮	~~$(e,6)$~~ $\boxed{(f,5)}$	⋮	⋮	⋮	~~$(b,7)$~~ $(f,6)$
7	$g : (f,6)$	⋮	⋮	⋮	$\boxed{(a,2)}$	$\boxed{(a,1)}$	⋮	$\boxed{(f,6)}$

Ainsi, $a \to f \to g$ est un chemin optimal de a à g, son coût est 6.

Soit la fonctions $h\colon V \to \mathbf{R}_+$ définie par le tableau suivant :

x	a	b	c	d	e	f	$\boldsymbol{g}$
$h(x)$	5	3	2	3	6	2	**0**

On peut vérifier que h est un minorant cohérente du coût pour atteindre g depuis le sommet donné. En particulier, elle est une estimation admissible pour l'algorithme A* (elle fournit une sous-estimation du coût suffisant pour atteindre g).

Voici la trace d'exécution de l'algorithme A* en utilisant cette fonction h :

		a (+5)	b (+3)	c (+2)	d (+3)	e (+6)	f (+2)	$\boldsymbol{g}$ (+0)
1	$a : (\bullet, 0)$	$\boxed{(\bullet, 0)}$	–	–	–	–	–	–
2	$d : (a, 2)$	⋮	–	–	$\boxed{(a, 2)}$	$(a, 1)$	$(a, 4)$	–
3	$b : (d, 3)$	⋮	$\boxed{(d, 3)}$	$(d, 7)$	⋮	⋮	⋮	–
4	$f : (a, 4)$	⋮	⋮	⋮	⋮	$(a, 1)$	$\boxed{(a, 4)}$	$(b, 7)$
5	$g : (f, 6)$	⋮	⋮	~~$(d, 7)$~~ $(f, 5)$	⋮	⋮	⋮	~~$(b, 7)$~~ $\boxed{(f, 6)}$

Ici l'algorithme A* trouve le même chemin $a \to f \to g$ que l'algorithme de Dijkstra.

15 Optimalité de A*

Lorsque on évalue l'efficacité de différents algorithmes de tri par comparaison, on compte le nombre des opérations de comparaison qu'ils effectuent. On peut comparer l'efficacité de différents algorithmes de recherche de chemin en comparant les nombres des fois qu'ils appellent la fonction N_G^+.

En se limitant au cas où la fonction h est cohérente, on peut montrer que l'algorithme A* est optimal parmi les algorithmes de recherche de chemin informés dans le sens suivant.

Théorème. *Soit A un algorithme qui reçoit le même type d'entrées (N_G^+, s, T, c, h) que l'algorithme A* et qui trouve un chemin optimal de s à un sommet de T dans G (à la condition qu'au moins un tel chemin existe). Supposons qu'il y a un tuple d'entrées $(N_{G_1}^+, s_1, T_1, c_1, h_1)$ avec h_1 cohérente, pour lequel l'algorithme A trouve un chemin optimal en effectuant strictement moins d'appels à la fonction $N_{G_1}^+$ que l'algorithme A*. Alors il existe un tuple d'entrées $(N_{G_2}^+, s_2, T_2, c_2, h_2)$ avec h_2 cohérente, où c'est l'inverse : A* trouve un chemin optimal en effectuant strictement moins d'appels à la fonction $N_{G_2}^+$ que l'algorithme A.*