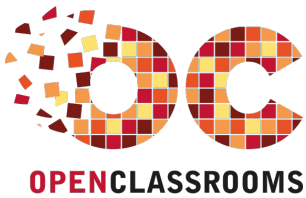
Le pathfinding avec Dijkstra

Par clemherreman



www.openclassrooms.com

Dernière mise à jour le 27/06/2011

Sommaire

Sommaire 2
Le pathfinding avec Dijkstra 3
Qu'est-ce que le pathfinding ? 3
Histoire et principe de l'algorithme de Dijkstra 3
Histoire de l'algorithme et de son inventeur 3
Principe de l'algorithme de Dijkstra 3
Étapes de l'algorithme de Dijkstra 5
Application de l'algorithme à travers un exemple (non codé) 8
On initialise les tableaux 8
On recherche le noeud avec le plus petit poids 9
On liste les fils de Bordeaux 9
Aperçu des tableaux 9
Et on recommence ! On recherche le noeud non entouré avec le poids le plus faible 10
On liste ses fils 10
Aperçu des tableaux 11
Et on recommence ! On recherche le noeud non entouré avec le poids le plus faible 11
On liste ses fils 12
Aperçu des tableaux 12
Limites et avantages de cet algorithme 14
Partager 14

Le pathfinding avec Dijkstra

Par clemherreman

Mise à jour : 27/06/2011

Bonjour !
Aujourd'hui nous allons étudier un algorithme assez connu (enfin, connu, c'est relatif bien sûr) : **l'algorithme de Dijkstra**.
Cet algorithme sert à trouver le chemin le plus court d'un point à un autre. Pour vous c'est facile, mais votre ordinateur, lui, est bête et ne sait pas aller de chez vous à chez Mamie par le chemin le plus court (eh non...).

Pour la suite, c'est par ici !

Sommaire du tutoriel :

- Qu'est-ce que le pathfinding ?
- Histoire et principe de l'algorithme de Dijkstra
- Application de l'algorithme à travers un exemple (non codé)
- Limites et avantages de cet algorithme

Qu'est-ce que le pathfinding ?

C'est l'histoire du livreur de pizzas, contraint de livrer ses commandes en moins d'une demi-heure avec pour seule arme un scooter faiblard et poussif. C'est aussi l'histoire de Monsieur Dupont qui, valise à la main, se rend à la gare de Strasbourg, direction Biarritz, sans bien savoir par où passer. C'est encore l'histoire du petit Epsien, synchronisant ses courses au Leclerc du coin avec le feu tricolore et qui cherche à épargner ses petits mollets. C'est enfin l'histoire d'Internet qui permet le transfert de tous types de données dans le monde entier via... quelle route, au fait ?

Le pathfinding est le fait de chercher à aller d'un point à un autre en trouvant le chemin le plus adapté (chemin le plus court, le plus rapide, uniquement sur autoroute, etc.).
Je vous invite à lire la définition du pathfinding sur Wikipédia qui est, je trouve, très complète.

J'ajoute que contrairement à l'algorithme A*, l'algorithme de Dijkstra est moins rapide et nécessite souvent plus de traitement, mais trouve le chemin **le plus court**.
Passons maintenant aux choses sérieuses .

Histoire et principe de l'algorithme de Dijkstra

Histoire de l'algorithme et de son inventeur

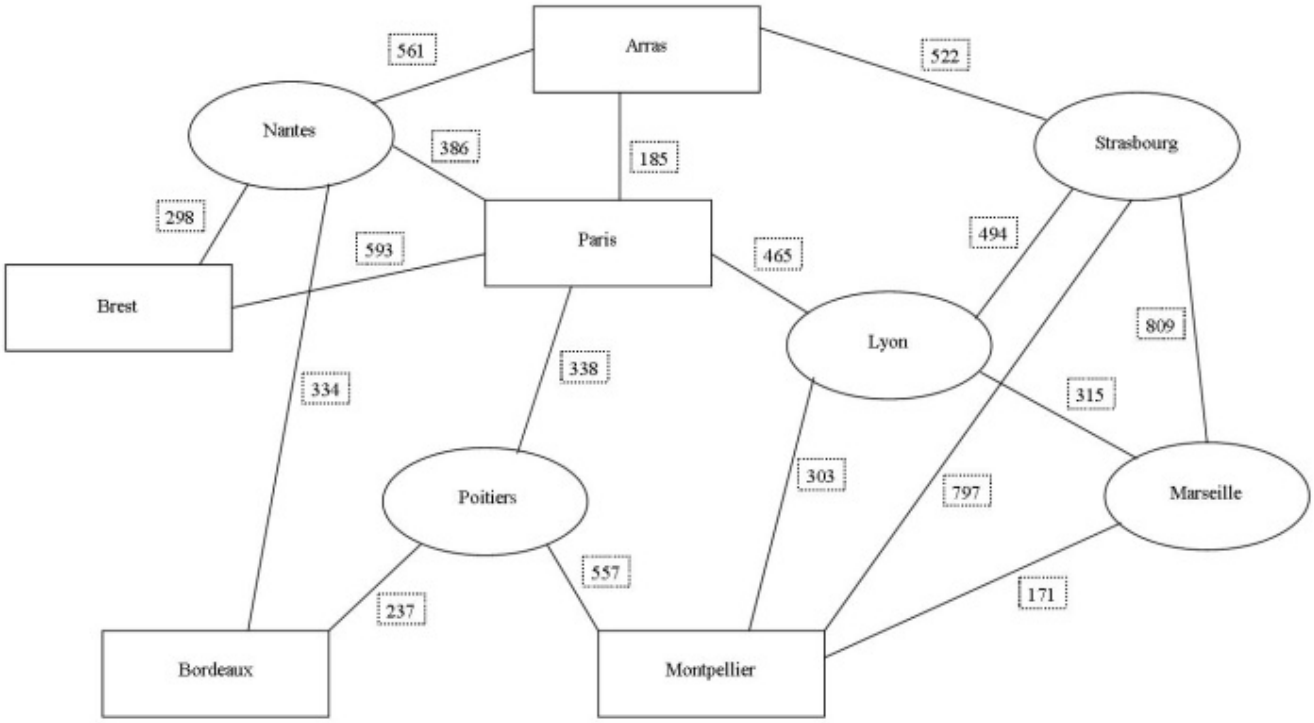
L'algorithme de Dijkstra a été trouvé par M. Dijkstra (si, si, je vous jure), Edsger de son prénom. Pour plus d'informations, vous pouvez consulter sa fiche sur Wikipédia, mais pour faire court, il a notamment trouvé cet algorithme du chemin le plus court, et a beaucoup participé au développement de langages tels l'ALGOL ou fait avancer les méthodes de programmation en se battant contre l'usage du **GOTO** en faveur d'une structure **If - Then - Else** à travers un article qu'il nomma *"A case against the GOTO statement"*.

Principe de l'algorithme de Dijkstra

L'algorithme de Dijkstra se découpe en plusieurs parties.

Prérequis

- Savoir ce qu'est un graphe et l'interpréter.
  Ceci est un graphe de liaison représentant ~~vaguement~~ la France, nous l'utiliserons afin d'illustrer ce tutoriel .



  Ce graphe est composé de 2 types d'objets : les noeuds, représentant des villes (ici dans des rectangles ou des ellipses), et les arêtes (les jointures entre différents noeuds), étant chacune affectée d'un poids (ici le nombre de kilomètres séparant chaque ville l'une de l'autre).
- Chaque noeud doit avoir au minimum une liaison.

Le principe

Le principe de l'algorithme de Dijkstra est de trouver le chemin ayant le poids le plus faible entre 2 noeuds, sachant que le poids d'un chemin est la somme des poids des arêtes qui le composent.

Euh...

En clair, sur ce graphe, l'algorithme va trouver le chemin le plus court en distance *(poids du chemin = sa longueur en km)*, sachant que la longueur du chemin est égale à la somme des longueurs de chaque arête le composant *(poids d'une arête = sa longueur)*.
Compris ?

Compris ! Mais comment l'ordinateur va-t-il faire ?

La démarche algorithmique n'est pas évidente à trouver, même pour les humains. Pour l'appliquer il va nous falloir deux tableaux.

- Un tableau à 3 lignes, que l'on va appeler "tableau des poids" contenant :
  - dans la première ligne, la liste des noeuds (désignés soit par leur nom, soit plutôt par un numéro) ;
  - dans la seconde ligne, un poids affecté à chaque noeud ;
  - dans la troisième ligne, une variable *vrai* ou *faux*, qui servira à savoir si l'on est déjà passé par ce noeud.
- Un autre tableau à une dimension, que l'on nommera "tableau des prédécesseurs" contenant ce que nous allons appeler les prédécesseurs de chaque noeud, c'est-à-dire le "noeud-père" qui précède le noeud dans le chemin que l'on prend pour y aller.
  Exemple : si pour aller de Nantes à Strasbourg je suis ce trajet *(cf. : graphe)* : Strasbourg => Arras => Nantes, le

prédécesseur de Nantes sera Arras, le prédécesseur de Arras sera Strasbourg, et Strasbourg lui, n'aura pas de prédécesseur (le pauvre...).

Cela peut sembler un peu confus, mais accrochez-vous, je vais expliciter un peu.
Prenons un exemple, reprenant le graphe que je vous ai donné ci-dessus.
Supposons que l'on veuille aller de Brest à Montpellier, en prenant le chemin le plus court en distance.

Étapes de l'algorithme de Dijkstra

0) Tout d'abord on initialise les tableaux

Concernant le tableau des poids, on va mettre tous les poids à -1, montrant par là qu'aucun poids n'a encore été affecté à un noeud, et on va mettre toutes les variables oui ou non à non, car on n'est passé par aucun noeud (ce qui est logique vu que l'algorithme n'a pas encore commencé son travail).
Ensuite, on met le poids du point de départ à 0 (bah oui, de Brest à Brest, il y a bien 0 km...).

Tableau des poids

Nom du noeud	Poids	Déjà parcouru ?
Arras	-1	Non
Bordeaux	-1	Non
Brest	0	Non
Lyon	-1	Non
Marseille	-1	Non
Montpellier	-1	Non
Nantes	-1	Non
Paris	-1	Non
Poitiers	-1	Non
Strasbourg	-1	Non

Puis on met à 0 le tableau des antécédents.

Tableau des antécédents

Noeud	Antécédent du noeud
Arras	Aucun
Bordeaux	Aucun
Brest	Aucun
Lyon	Aucun
Marseille	Aucun
Montpellier	Aucun
Nantes	Aucun
Paris	Aucun
Poitiers	Aucun
Strasbourg	Aucun

1) On recherche le noeud non parcouru ayant le poids le plus faible et on indique donc qu'on l'a parcouru

La première fois, il s'agit forcément du noeud de départ. On met donc la variable oui ou non de Brest à oui.

2)On va rechercher ce que l'on appelle "les fils" du noeud où l'on se trouve

Si l'on découvre des fils au noeud, on va effectuer une condition sur chaque fils que l'on a trouvé :

Code : Autre

```
SI ( le noeud-fils n'a pas encore été parcouru )

ET QUE ( Poids(Noeud-père) + Poids(Liaison Noeud-père/Noeud-
fils) < Poids(Noeud-fils) ) OU Poids(Noeud-fils) = -1

{

    Poids(Noeud-fils) = Poids(Noeud-père) + Poids(Liaison Noeud-
père/Noeud-fils)

    Antecedent(Noeud-fils) = Noeud-Père

}
```

Pourquoi on fait ça ? Et pourquoi on change les poids et les antécédents ? J'aimais bien les anciens, moi...

Décortiquons un peu ce code .

Code : Autre

```
SI ( le noeud-fils n'a pas encore été parcouru )

ET QUE ( Poids(Noeud-père) + Poids(Liaison Noeud-père/Noeud-
fils) < Poids(Noeud-fils) ) OU Poids(Noeud-fils) = -1
```

Que veux dire cette condition ?

Code : Autre

```
SI ( le noeud-fils n'a pas encore été parcouru )
```

Cette ligne vérifie que l'on n'a pas encore parcouru le noeud-fils (si, si, je vous jure).
En effet, en réfléchissant un peu, si on a déjà parcouru le noeud-fils, c'est que la distance (Point de départ-Noeud fils) est inférieure à la distance (Point de départ-Noeud père) et que donc le Noeud-fils ne nous intéresse plus.

Code : Autre

```
Poids(Noeud-père) + Poids(Liaison Noeud-père/Noeud-
fils) < Poids(Noeud-fils)
```

- Poids(Noeud-père) est le poids pour aller jusqu'au Noeud-père. Ici cela représente la distance parcourue du noeud de

départ jusqu'au noeud où l'on se trouve ;
- Poids(Liaison Noeud-père/Noeud-fils) représente ici la distance entre le Noeud-père où l'on se trouve, et le Noeud-fils ;
- Poids(Noeud-fils) représente la distance à parcourir pour aller du noeud de départ au Noeud-fils.

Concrètement, cela veut dire que la distance pour aller du départ au Noeud-fils, en passant par le Noeud-père où l'on se trouve, est plus petite que la distance pour aller du départ au Noeud-fils que l'on avait déjà trouvée en passant par un autre chemin.

Si on se trouve dans ce cas précis **alors** notre condition est validée, on exécute ce code :

Code : Autre

```
Poids(Noeud-fils) = Poids(Noeud-père) + Poids(Liaison Noeud-
père/Noeud-fils)

Antecedent(Noeud-fils) = Noeud-Père
```

On change donc le poids du Noeud-fils, en mettant celui du chemin passant par le Noeud-père, et on indique dans le tableau d'antécédents, que l'étape précédente du Noeud-fils est le Noeud-père.

3) On boucle en retournant à l'étape 1) tant que...

Tant que le noeud ayant le poids le plus faible n'est pas le noeud d'arrivée.
En effet, quand le noeud ayant le poids le plus faible sera le noeud d'arrivée, cela voudra dire que l'on a trouvé le chemin le plus court pour y aller en partant du noeud de départ.

Le tour est joué  !

 Oui mais... comment je retrouve ce chemin ??

C'est là que le tableau des antécédents est utile !
On va prendre l'antécédent du noeud d'arrivée (dans notre exemple, Montpellier), puis l'antécédent de l'antécédent de Montpellier, puis l'antécédent de l'antécédent de l'antécédent de Montpellier puis l'antécédent de l'antécédent de...
Enfin bref, vous avez compris  .

Exemple : supposons qu'à la fin de l'exécution de l'algorithme, vous avez un tableau d'antécédents qui ressemble à ça, et que vous cherchiez à aller de Nantes à Lyon.

Noeud	Antécédent du noeud
Arras	Nantes
Bordeaux	Nantes
Brest	Nantes
Lyon	Paris
Marseille	*Aucun*
Montpellier	Poitiers
Nantes	*Aucun*
Paris	Nantes
Poitiers	Bordeaux
Strasbourg	Arras

Vous allez regarder l'antécédent de Lyon : Paris, l'antécédent de Paris : Nantes, hop ! Vous êtes arrivés à Nantes en marche arrière !! (OK, je sors...)
Vous remettez le tout dans l'ordre, et vous avez votre chemin le plus court :

Nantes => Paris => Lyon

Évidemment, en regardant le graphe, cela semble évident et inutile d'utiliser un algorithme pour arriver à un résultat si trivial, mais imaginez un peu que vous modélisiez toutes les villes et toutes les routes de la France ? Ah ! ça devient intéressant là, non ? Imaginez qu'en plus de ça, vous ajoutiez toutes les rues de chaque ville , ça deviendrait carrément indispensable...

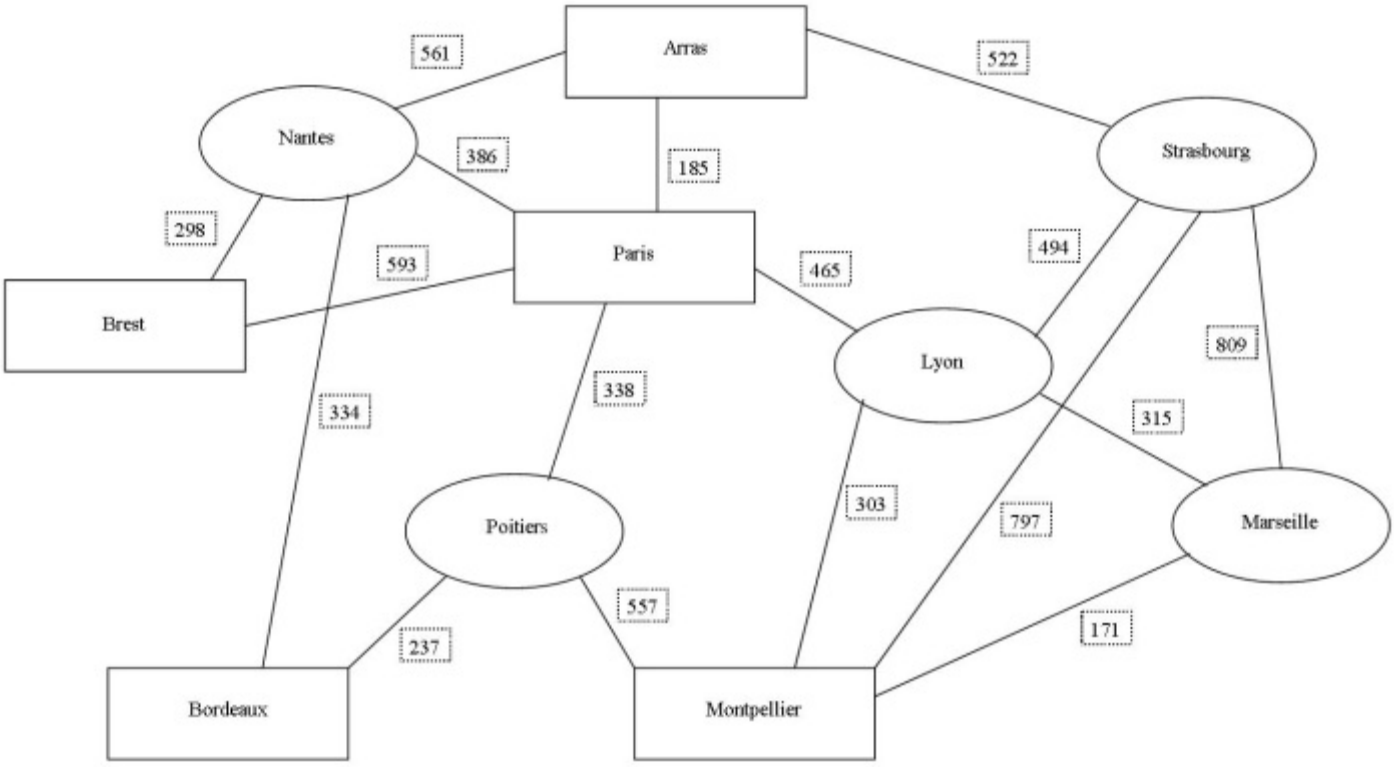
Application de l'algorithme à travers un exemple (non codé)

Vous avez tout compris ?

Euh oui, mais c'est peut-être encore un peu confus pour moi...

J'imagine , c'est pour ça que nous allons voir un exemple concret pas à pas. Ne vous inquiétez pas, par souci d'ouverture, c'est un exemple n'utilisant aucun langage, que ce soit le C, le PHP, etc. (je suis gentil, hein ?).

Reprenons le graphe de tout à l'heure.



Supposons que nous soyons à Bordeaux et que nous voulions aller à Strasbourg.

On initialise les tableaux

Tableau des poids

Nom du noeud	Poids	Déjà parcouru ?
Arras	-1	Non
Bordeaux	0	Non
Brest	-1	Non
Lyon	-1	Non

Marseille	-1	Non
Montpellier	-1	Non
Nantes	-1	Non
Paris	-1	Non
Poitiers	-1	Non
Strasbourg	-1	Non

Tableau des antécédents

Noeud	Antécédent du noeud
Arras	*Aucun*
Bordeaux	*Aucun*
Brest	*Aucun*
Lyon	*Aucun*
Marseille	*Aucun*
Montpellier	*Aucun*
Nantes	*Aucun*
Paris	*Aucun*
Poitiers	*Aucun*
Strasbourg	*Aucun*

On recherche le noeud avec le plus petit poids

Il s'agit de **Bordeaux**.

On liste les fils de Bordeaux

- Nantes :
  - est-on déjà passé par Nantes ?
    => Non ;
  - kilométrage de Nantes > kilométrage de Bordeaux + distance Bordeaux-Nantes ?
    OU
    kilométrage de Nantes non défini (= -1) ?
    => Oui :
    - kilométrage de Nantes = kilométrage de Bordeaux + distance Bordeaux-Nantes ;
    - kilométrage de Nantes = 0 + 334 = 334 ;
    - antécédent de Nantes = Bordeaux.
- Poitiers :
  - est-on déjà passé par Poitiers ?
    => Non ;
  - kilométrage de Poitiers > kilométrage de Bordeaux + distance Bordeaux-Poitiers ?
    OU
    kilométrage de Poitiers non défini (= -1) ?
    => Oui :
    - kilométrage de Poitiers = kilométrage de Bordeaux + distance Bordeaux-Poitiers ;
    - kilométrage de Poitiers = 0 + 237 = 237 ;
    - antécédent de Poitiers = Bordeaux.

Aperçu des tableaux

Tableau des poids

Nom du noeud	Poids	Déjà parcouru ?
Arras	-1	Non
Bordeaux	0	Oui
Brest	-1	Non
Lyon	-1	Non
Marseille	-1	Non
Montpellier	-1	Non
Nantes	334	Non
Paris	-1	Non
Poitiers	237	Non
Strasbourg	-1	Non

Tableau des antécédents

Noeud	Antécédent du noeud
Arras	*Aucun*
Bordeaux	*Aucun*
Brest	*Aucun*
Lyon	*Aucun*
Marseille	*Aucun*
Montpellier	*Aucun*
Nantes	Bordeaux
Paris	*Aucun*
Poitiers	Bordeaux
Strasbourg	*Aucun*

Et on recommence ! On recherche le noeud non entouré avec le poids le plus faible

Il s'agit de **Poitiers**.

On liste ses fils

- Bordeaux :
  - est-on déjà passé par Bordeaux ?
    => Oui.
- Montpellier :
  - est-on déjà passé par Montpellier ?
    => Non ;
  - kilométrage de Montpellier > kilométrage de Poitiers + distance Poitiers-Montpellier ?
    OU
    kilométrage de Montpellier non défini (= -1) ?
    => Oui :
    - kilométrage de Montpellier = kilométrage de Poitiers + distance Poitiers-Montpellier ;
    - kilométrage de Montpellier = 237 + 557 = 794 ;
    - antécédent de Montpellier = Poitiers.

- Paris :
  - est-on déjà passé par Paris ?
    => Non ;
  - kilométrage de Paris > kilométrage de Poitiers + distance Poitiers-Paris ?
    OU
    kilométrage de Paris non défini (= -1) ?
    => Oui :
    - kilométrage de Paris = kilométrage de Poitiers + distance Poitiers-Paris ;
    - kilométrage de Paris = 237 + 338 = 575 ;
    - antécédent de Paris = Poitiers.

Aperçu des tableaux

Tableau des poids

Nom du noeud	Poids	Déjà parcouru ?
Arras	-1	Non
Bordeaux	0	Oui
Brest	-1	Non
Lyon	-1	Non
Marseille	-1	Non
Montpellier	794	Non
Nantes	334	Non
Paris	575	Non
Poitiers	237	Oui
Strasbourg	-1	Non

Tableau des antécédents

Noeud	Antécédent du noeud
Arras	*Aucun*
Bordeaux	*Aucun*
Brest	*Aucun*
Lyon	*Aucun*
Marseille	*Aucun*
Montpellier	Poitiers
Nantes	Bordeaux
Paris	Poitiers
Poitiers	Bordeaux
Strasbourg	*Aucun*

Et on recommence ! On recherche le noeud non entouré avec le poids le plus faible

Il s'agit de **Nantes**.

On liste ses fils

- Arras :
  - est-on déjà passé par Arras ?
    => Non ;
  - kilométrage de Arras > kilométrage de Nantes + distance Nantes-Arras ?
    OU
    kilométrage de Arras non défini (= -1) ?
    => Oui :
    - kilométrage de Arras = kilométrage de Nantes + distance Nantes-Arras
    - kilométrage de Arras = 334 + 561 = 895 ;
    - antécédent de Arras = Nantes.
- Bordeaux :
  - est-on déjà passé par Bordeaux ?
    => Oui.
- Brest :
  - est-on déjà passé par Brest ?
    => Non ;
  - kilométrage de Brest > kilométrage de Nantes + distance Nantes-Brest ?
    OU
    kilométrage de Brest non défini (= -1) ?
    => Oui :
    - kilométrage de Brest = kilométrage de Nantes + distance Nantes-Brest ;
    - kilométrage de Brest = 334 + 298 = 632 ;
    - antécédent de Brest = Nantes.
- Paris :
  - est-on déjà passé par Paris ?
    => Non ;
  - kilométrage de Paris > kilométrage de Nantes + distance Nantes-Paris ?
    OU
    kilométrage de Paris non défini (= -1) ?
    => Non.
    - Remarque : donc on n'y touche pas.

Aperçu des tableaux

Tableau des poids

Nom du noeud	Poids	Déjà parcouru ?
Arras	895	Non
Bordeaux	0	Oui
Brest	632	Non
Lyon	-1	Non
Marseille	-1	Non
Montpellier	794	Non
Nantes	334	Oui
Paris	575	Non
Poitiers	237	Oui
Strasbourg	-1	Non

Tableau des antécédents

Noeud	Antécédent du noeud
Arras	Nantes

Bordeaux	*Aucun*
Brest	Nantes
Lyon	*Aucun*
Marseille	*Aucun*
Montpellier	Poitiers
Nantes	Bordeaux
Paris	Poitiers
Poitiers	Bordeaux
Strasbourg	*Aucun*

Je ne continue pas afin d'épargner à ce tutoriel un alourdissement énorme et d'éviter qu'il ne fasse cinq kilomètres de long (et aussi parce que c'est assez long à mettre en forme pour moi).
Toutes les différentes situations conditionnelles concernant les noeuds-fils ont été rencontrées, et il suffit de continuer en boucle, jusqu'à ce que le noeud de poids le plus faible soit le noeud d'arrivée (Strasbourg).

Je vous invite à finir de trouver le chemin, et je vous laisse les états des tableaux des poids et des antécédents quand on est arrivé à Strasbourg, comparez-les avec les vôtres .

Secret (cliquez pour afficher)

Tableau des antécédents

Noeud	Antécédents du noeud
Arras	Paris
Bordeaux	*Aucun*
Brest	Nantes
Lyon	Paris
Marseille	Montpellier
Montpellier	Poitiers
Nantes	Bordeaux
Paris	Poitiers
Poitiers	Bordeaux
Strasbourg	Arras

Tableau des poids

Nom du noeud	Poids	Déjà parcouru ?
Arras	760	Oui
Bordeaux	0	Oui
Brest	632	Oui
Lyon	1040	Oui
Marseille	965	Oui
Montpellier	794	Oui

Nantes	334	Oui
Paris	575	Oui
Poitiers	237	Oui
Strasbourg	1282	Oui

Dans cet exemple, on est passé par toutes les villes mais ce n'est pas toujours le cas.
Exemple : aller d'Arras à Brest.

Chemin le plus court (roulement de tambours...) :
1282 kilomètres parcourus !
Pour un total de

Bordeaux -> Poitiers -> Paris -> Arras -> Strasbourg

Limites et avantages de cet algorithme

L'algorithme de Dijkstra, très puissant, admet quand même quelques limites :

- il demande une certaine masse de traitements quand le graphe devient grand. Par exemple, comme l'explique Haveo dans son tutoriel "Le pathfinding avec A*", quand la puissance matérielle est limitée, où que l'on souhaite traiter rapidement le problème du chemin le plus court, on peut utiliser d'autres algorithmes, tel A*, qui sont certes moins précis parfois, mais qui sont plus rapides et moins gourmands en ressources ;
- on ne peut pas affecter aux liaisons un poids négatif, sinon l'algorithme est faussé et retournera des résultats faux.

Cependant il a plusieurs avantages :

- contrairement à l'algorithme A* par exemple, il trouve toujours le chemin **ayant le poids le plus faible** ;
- il n'est pas si dur à comprendre et à mettre en place (surtout avec quelqu'un qui explique aussi bien que moi) ;
- il peut être utilisé pour d'autres applications que la distance la plus courte d'un point à un autre.
  On peut remplacer les villes par d'autres choses. J'avais par exemple fait un graphe où les liaisons étaient des pertes d'argent et les noeuds étaient des choix économiques d'entreprise.

Voilà, c'est fini !
J'espère avoir éclairé votre lanterne sur le fonctionnement de Mappy .
À noter cependant que ma manière d'implémenter peut probablement être améliorée. J'utilise pour ma part une boucle qui se répète jusqu'au noeud d'arrivée, on peut utiliser à la place une approche récursive, chacun ses goûts.

J'ai implémenté cet algorithme en langage C dans un de mes projets scolaires : si cela intéresse quelqu'un, je peux probablement extraire le module du chemin le plus court et partager les sources. Contactez-moi par message privé .

Ce tutoriel a été corrigé par les zCorrecteurs.