Calculs des distances dans les graphes valués

Florent Foucaud - Malika More - Thibault Ralet
Carine Simon - Thierry Trévisan

1A - BUT Info - UCA

R207 Graphes

Année 2021-2022

Sommaire

Trouver des chemins courts et calculer des distances

Problème du plus court chemin

Il s'agit d'un problème récurrent en informatique, rencontré très souvent :

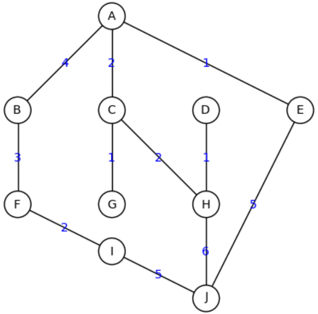
- En réseaux, pour le routage avec le protocole open shortest path first
- Calcul d'un itinéraire optimal dans une application GPS, ou dans un réseau de transports en commun
- En robotique, pour décider de la trajectoire d'un robot
- Dans l'industrie des jeux, comme sous-routine de l'IA gérant un bot

Plusieurs problèmes semblables

Il y a plusieurs variantes :

- Les distances depuis un sommet vers tous les autres
- Les chemins les plus courts depuis un sommet vers tous les autres
- Toutes les distances / tous les chemins plus courts entre toutes les paires de sommets
- Plusieurs chemins (les 10 chemins les plus courts par exemple, en cas de panne)
- etc.

Le problème des distances à un sommet



- On se donne un graphe avec des valeurs sur les arêtes (représentant une distance ou un temps de parcours)
- On veut calculer la distance de A à tous les autres sommets

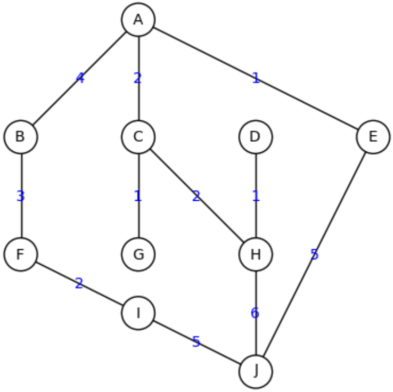
Comment faire ?

- Entrée : un graphe avec des distances sur les arêtes, et un sommet
- Question : calculer la distance de A à tous les autres sommets

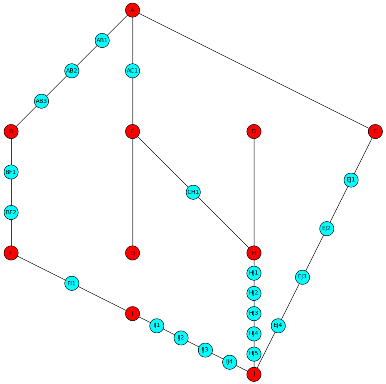
Sommaire

Avec un parcours en largeur ?

Graphe avec des distances

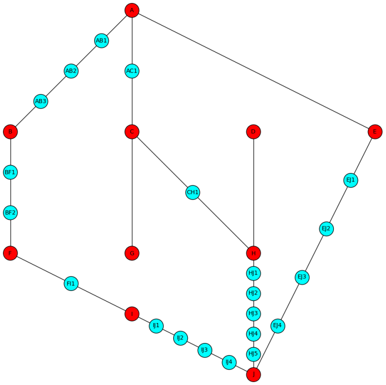


Graphe (usuel = sans distances)

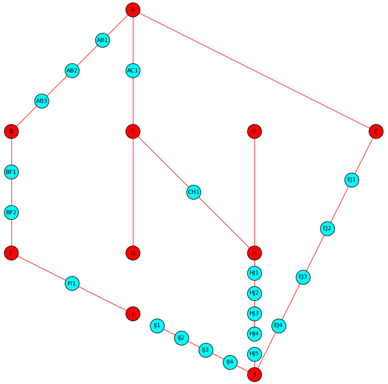


Avec un parcours en largeur ?

Graphe (usuel = sans distances)



Parcours en largeur



Inconvénients

- Il faut distinguer entre les anciens sommets (qui nous intéressent) et les nouveaux sommets (uniquement là pour calculer les distances vers les autres)
- Si les distances sont grandes, le graphe devient grand : pour notre petit graphe à 10 sommets, quelques distances de l'ordre de 100 sur certaines arêtes donnent un graphe à plusieurs centaines de sommets !
- Or le temps nécessaire au parcours en largeur dépend du nombre de sommets : il est proportionnel à $v + e$ si v est le nombre de sommets et e le nombre d'arêtes

Méthode sans ces inconvénients

- On simule le principe du parcours en largeur
- Mais on travaille directement sur le graphe avec distance
- Le temps dépend du nombre de sommets

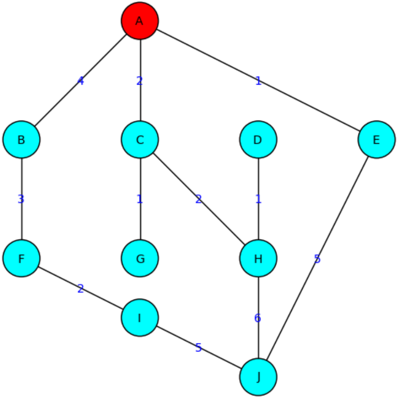
C'est le célèbre algorithme de Dijkstra de 1959.

Edsger W. Dijkstra
(1930-2002)

Sommaire

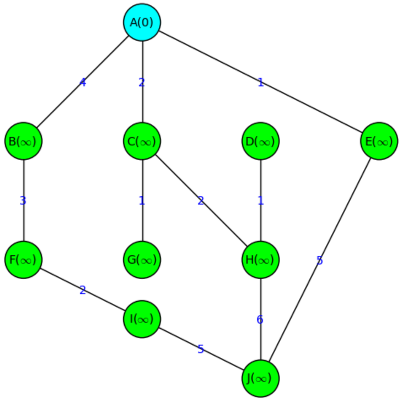
Sommaire

Exemple de l'algorithme de Dijkstra



Distances depuis A

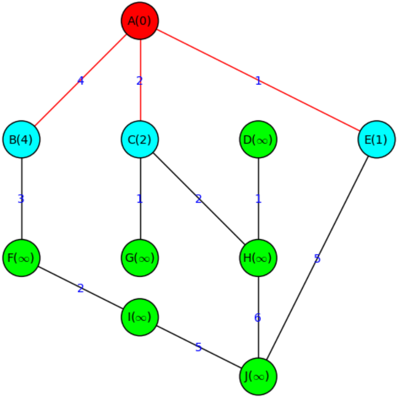
Exemple de l'algorithme de Dijkstra



Distances depuis A

Au début on ne connaît que le sommet A de distance 0 à lui-même, et les autres sommets sont à distance ∞

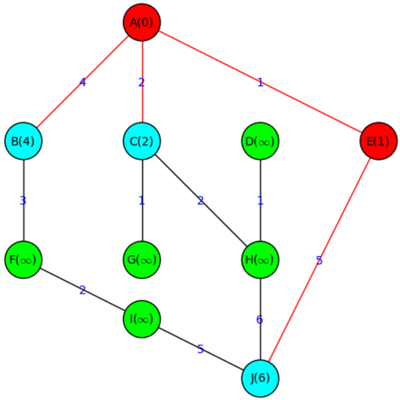
Exemple de l'algorithme de Dijkstra



Distances depuis A

On met à jour les distances vers les sommets B, C et E voisins de A, qu'on ajoute à la frontière et on marque A comme traité

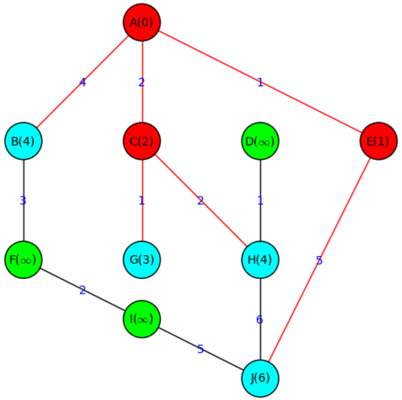
Exemple de l'algorithme de Dijkstra



Distances depuis A

- On choisit comme nouveau sommet courant celui dont la distance est la plus proche de A ici le sommet E.
  (cela correspond à travailler par niveau dans un parcours en largeur)
- On découvre un voisin de E jusque là inconnu (sommet J) et on met à jour sa distance à A
- On marque E comme traité

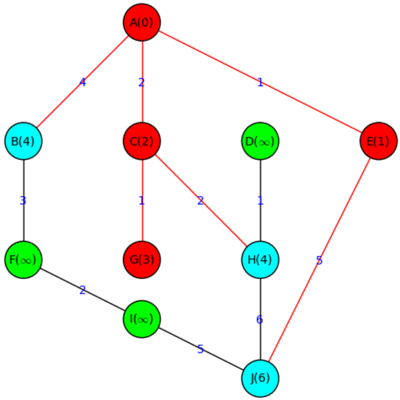
Exemple de l'algorithme de Dijkstra



Distances depuis A

On continue sur le même principe avec le prochain sommet non marqué (c-à-d. de la frontière) le plus proche de A , ici C, qui permet de découvrir G et H et de leur attribuer une distance à A

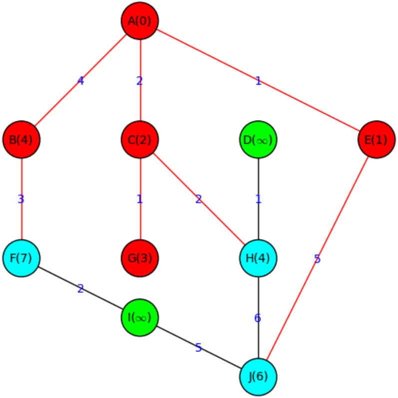
Exemple de l'algorithme de Dijkstra



Distances depuis A

Puis G

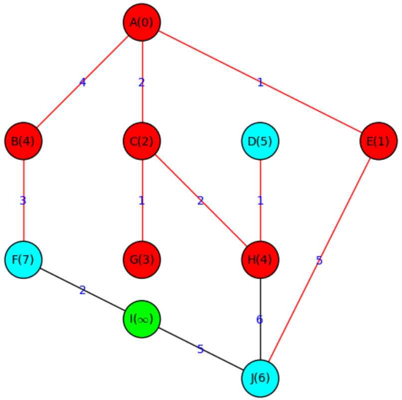
Exemple de l'algorithme de Dijkstra



Distances depuis A

Puis B, qui permet de découvrir F

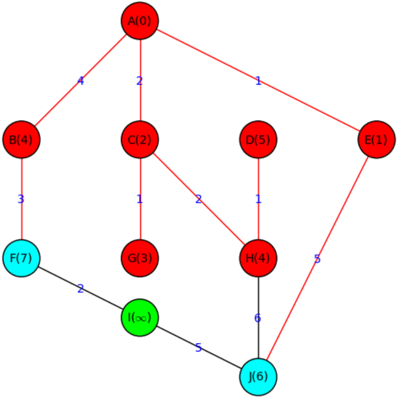
Exemple de l'algorithme de Dijkstra



Distances depuis A

Puis H, qui permet de découvrir D (notons qu'on aurait pu aussi faire H puis B puisqu'ils étaient tous les deux à distance 4)

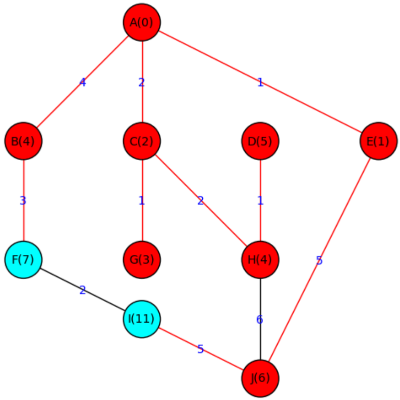
Exemple de l'algorithme de Dijkstra



Distances depuis A

Puis *D*

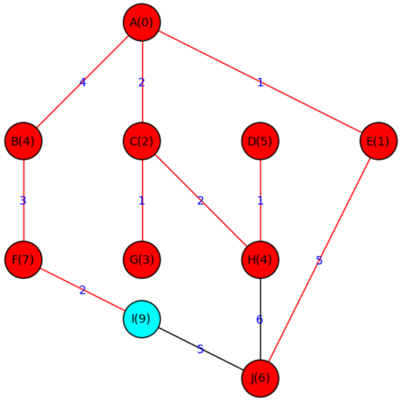
Exemple de l'algorithme de Dijkstra



Distances depuis A

Puis J : on découvre depuis J le sommet I à distance 11

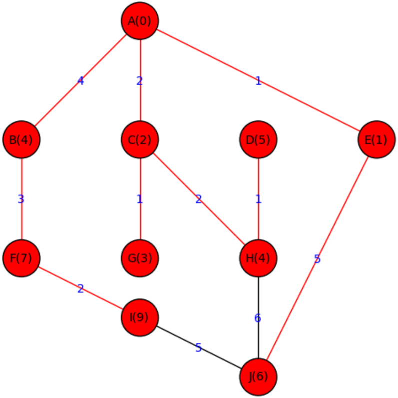
Exemple de l'algorithme de Dijkstra



Distances depuis A

Puis F (notez la mise à jour de la distance de I qu'on redécouvre depuis F, pour un chemin de longueur 9)

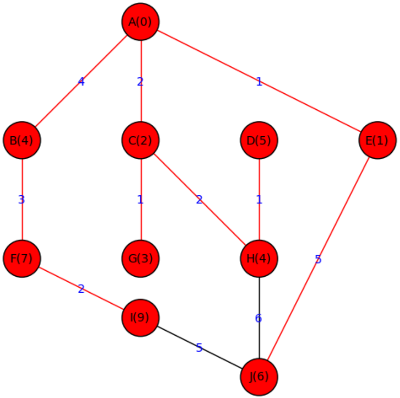
Exemple de l'algorithme de Dijkstra



Distances depuis A

Et enfin I

Arbre couvrant et distances



- Cet algorithme simule le principe d'un parcours en largeur, on obtient donc un arbre couvrant apparenté à celui d'un parcours en largeur
- Un plus court chemin de A à un sommet X, correspond à l'unique chemin dans l'arbre couvrant de A à X.

L'algorithme de Dijkstra point par point

Initialisation

0 pour le premier sommet, ∞ pour les autres

Traitement

On procède par marquage progressif des sommets :

- sommet courant = choisir X non marqué de valeur minimale
- pour tout Y non marqué adjacent à X, mettre à jour la distance / le prédécesseur Y si plus court via X
- marquer X

Fin

- quand tous les sommets sont marqués
- distance à la source = valeur trouvée pour chaque sommet
- pour connaître le chemin correspondant à cette distance, il faut noter le prédécesseur

Sommaire

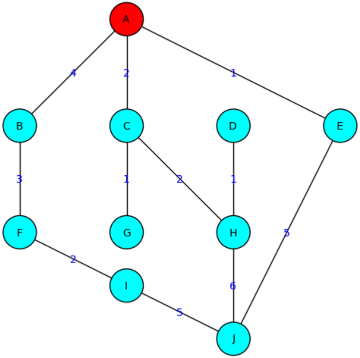
Algorithme de Djikstra

pour les humains en version tableau

Etape 1 (Initialisation)

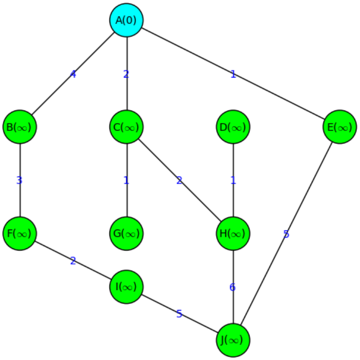
- Placer tous les sommets du graphe dans la 1ère ligne d'un tableau
- Sur la 2ème ligne, écrire la valeur 0 pour le sommet de départ et les valeurs ∞ pour tous les autres sommets

Présenter l'exécution de l'algo sous forme de tableau



A	B	C	D	E	F	G	H	I	J

Présenter l'exécution de l'algo sous forme de tableau



A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞

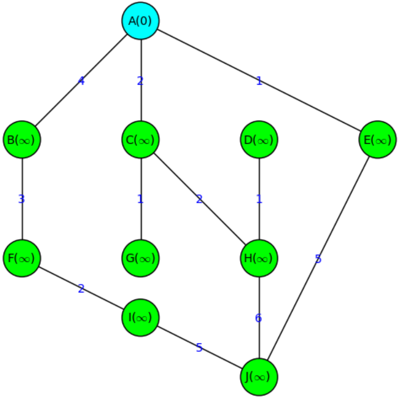
Algorithme de Dijkstra

pour les humains en version tableau

Etape 2 (Traitement)

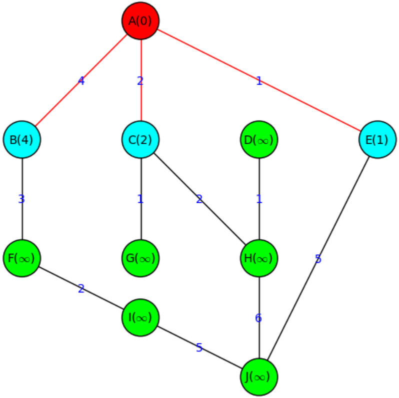
- Repérer le sommet X de valeur minimale : rayer la colonne sous ce sommet
- Pour tous les sommets Y non rayés :
  - Si Y est adjacent à X alors :
    - p = valeur(X)+poids(arête X-Y)
    - Si p < valeur(Y) alors remplacer la valeur de Y par p ; sinon recopier la valeur(Y).
  - Si Y n'est pas adjacent à X alors recopier la valeur(Y)
- Tant qu'il reste des colonnes non rayées, reprendre l'étape 2.

Présenter l'exécution de l'algo sous forme de tableau



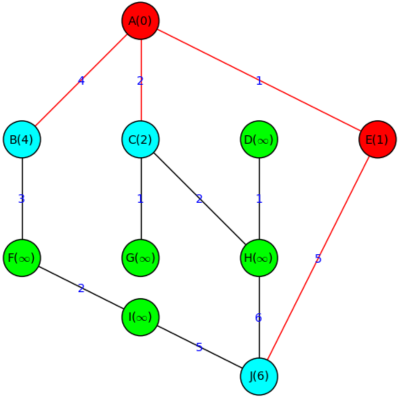
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞

Présenter l'exécution de l'algo sous forme de tableau



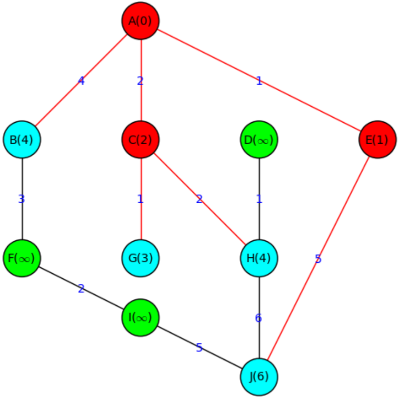
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞

Présenter l'exécution de l'algo sous forme de tableau



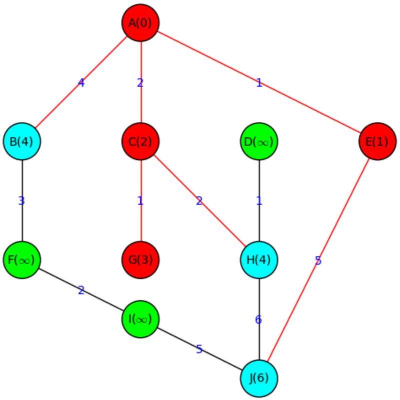
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6

Présenter l'exécution de l'algo sous forme de tableau



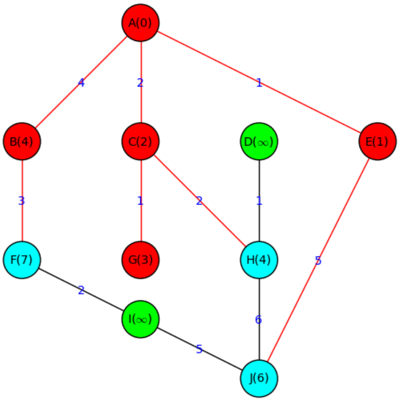
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6

Présenter l'exécution de l'algo sous forme de tableau



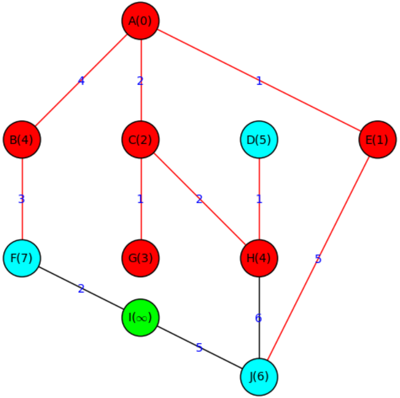
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6
$\|\|$	4	$\|\|$	∞	$\|\|$	∞	$\times$	4	∞	6

Présenter l'exécution de l'algo sous forme de tableau



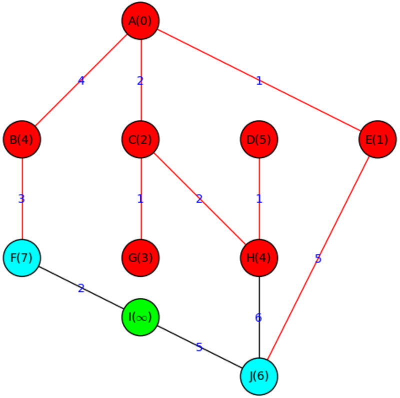
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6
$\|\|$	4	$\|\|$	∞	$\|\|$	∞	$\times$	4	∞	6
$\|\|$	$\times$	$\|\|$	∞	$\|\|$	7	$\|\|$	4	∞	6

Présenter l'exécution de l'algo sous forme de tableau



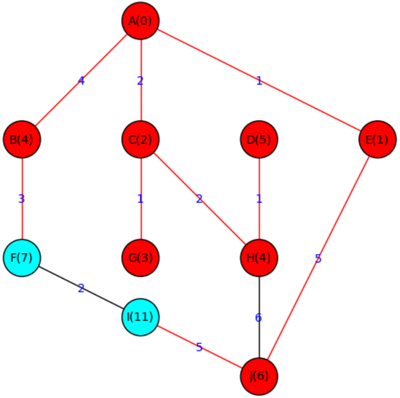
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6
$\|\|$	4	$\|\|$	∞	$\|\|$	∞	$\times$	4	∞	6
$\|\|$	$\times$	$\|\|$	∞	$\|\|$	7	$\|\|$	4	∞	6
$\|\|$	$\|\|$	$\|\|$	5	$\|\|$	7	$\|\|$	$\times$	∞	6

Présenter l'exécution de l'algo sous forme de tableau



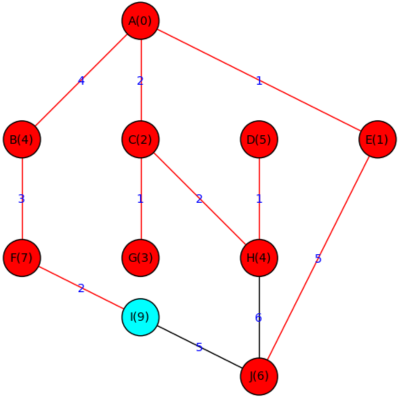
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6
$\|\|$	4	$\|\|$	∞	$\|\|$	∞	$\times$	4	∞	6
$\|\|$	$\times$	$\|\|$	∞	$\|\|$	7	$\|\|$	4	∞	6
$\|\|$	$\|\|$	$\|\|$	5	$\|\|$	7	$\|\|$	$\times$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	7	$\|\|$	$\|\|$	∞	6

Présenter l'exécution de l'algo sous forme de tableau



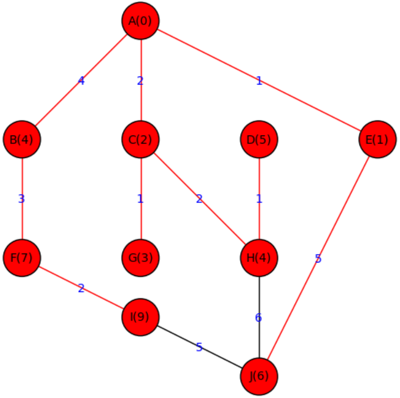
A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
×	4	2	∞	1	∞	∞	∞	∞	∞
||	4	2	∞	×	∞	∞	∞	∞	6
||	4	×	∞	||	∞	3	4	∞	6
||	4	||	∞	||	∞	×	4	∞	6
||	×	||	∞	||	7	||	4	∞	6
||	||	||	5	||	7	||	×	∞	6
||	||	||	×	||	7	||	||	∞	6
||	||	||	||	||	7	||	||	11	×

Présenter l'exécution de l'algo sous forme de tableau



A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\Vert$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\Vert$	4	$\times$	∞	$\Vert$	∞	3	4	∞	6
$\Vert$	4	$\Vert$	∞	$\Vert$	∞	$\times$	4	∞	6
$\Vert$	$\times$	$\Vert$	∞	$\Vert$	7	$\Vert$	4	∞	6
$\Vert$	$\Vert$	$\Vert$	5	$\Vert$	7	$\Vert$	$\times$	∞	6
$\Vert$	$\Vert$	$\Vert$	$\times$	$\Vert$	7	$\Vert$	$\Vert$	∞	6
$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	7	$\Vert$	$\Vert$	11	$\times$
$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\times$	$\Vert$	$\Vert$	9	$\Vert$

Présenter l'exécution de l'algo sous forme de tableau



A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\Vert$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\Vert$	4	$\times$	∞	$\Vert$	∞	3	4	∞	6
$\Vert$	4	$\Vert$	∞	$\Vert$	∞	$\times$	4	∞	6
$\Vert$	$\times$	$\Vert$	∞	$\Vert$	7	$\Vert$	4	∞	6
$\Vert$	$\Vert$	$\Vert$	5	$\Vert$	7	$\Vert$	$\times$	∞	6
$\Vert$	$\Vert$	$\Vert$	$\times$	$\Vert$	7	$\Vert$	$\Vert$	∞	6
$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	7	$\Vert$	$\Vert$	11	$\times$
$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\times$	$\Vert$	$\Vert$	9	$\Vert$
$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\times$	$\Vert$

Algorithme de Djikstra

pour les humains en version tableau

Etape 3 (Fin)

- Le poids de la chaîne la plus courte depuis le sommet de départ est le dernier nombre dans la colonne du sommet d'arrivée
- Pour connaître cette chaîne, on remonte à l'envers : partir du dernier nombre de la colonne du sommet d'arrivée. Rechercher l'étape de sa modification : prendre le sommet qui a été barré à cette étape. Procéder de même jusqu'à remonter à la valeur 0

Retrouver un plus court chemin avec le tableau

Exemple

Distance de A à I = 9

Chemin le plus court de A à I (à l'envers) : I

A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\Vert$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\Vert$	4	$\times$	∞	$\Vert$	∞	3	4	∞	6
$\Vert$	4	$\Vert$	∞	$\Vert$	∞	$\times$	4	∞	6
$\Vert$	$\times$	$\Vert$	∞	$\Vert$	7	$\Vert$	4	∞	6
$\Vert$	$\Vert$	$\Vert$	5	$\Vert$	7	$\Vert$	$\times$	∞	6
$\Vert$	$\Vert$	$\Vert$	$\times$	$\Vert$	7	$\Vert$	$\Vert$	∞	6
$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	7	$\Vert$	$\Vert$	11	$\times$
$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\times$	$\Vert$	$\Vert$	9	$\Vert$
$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\Vert$	$\times$	$\Vert$

Retrouver un plus court chemin avec le tableau

Exemple

Distance de A à $I = 9$

Chemin le plus court de A à I (à l'envers) : I–F

A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6
$\|\|$	4	$\|\|$	∞	$\|\|$	∞	$\times$	4	∞	6
$\|\|$	$\times$	$\|\|$	∞	$\|\|$	7	$\|\|$	4	∞	6
$\|\|$	$\|\|$	$\|\|$	5	$\|\|$	7	$\|\|$	$\times$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	7	$\|\|$	$\|\|$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	7	$\|\|$	$\|\|$	11	$\times$
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	$\|\|$	9	$\|\|$
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$

Retrouver un plus court chemin avec le tableau

Exemple

Distance de A à $I = 9$

Chemin le plus court de A à I (à l'envers) : I–F

A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6
$\|\|$	4	$\|\|$	∞	$\|\|$	∞	$\times$	4	∞	6
$\|\|$	$\times$	$\|\|$	∞	$\|\|$	7	$\|\|$	4	∞	6
$\|\|$	$\|\|$	$\|\|$	5	$\|\|$	7	$\|\|$	$\times$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	7	$\|\|$	$\|\|$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	7	$\|\|$	$\|\|$	11	$\times$
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	$\|\|$	9	$\|\|$
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$

Retrouver un plus court chemin avec le tableau

Exemple

Distance de A à $I = 9$

Chemin le plus court de A à I (à l'envers) : I–F–B

A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6
$\|\|$	4	$\|\|$	∞	$\|\|$	∞	$\times$	4	∞	6
$\|\|$	$\times$	$\|\|$	∞	$\|\|$	7	$\|\|$	4	∞	6
$\|\|$	$\|\|$	$\|\|$	5	$\|\|$	7	$\|\|$	$\times$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	7	$\|\|$	$\|\|$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	7	$\|\|$	$\|\|$	11	$\times$
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	$\|\|$	9	$\|\|$
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$

Retrouver un plus court chemin avec le tableau

Exemple

Distance de A à $I = 9$

Chemin le plus court de A à I (à l'envers) : I–F–B

A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|\|$	4	$\times$	∞	$\|\|$	∞	3	4	∞	6
$\|\|$	4	$\|\|$	∞	$\|\|$	∞	$\times$	4	∞	6
$\|\|$	$\times$	$\|\|$	∞	$\|\|$	7	$\|\|$	4	∞	6
$\|\|$	$\|\|$	$\|\|$	5	$\|\|$	7	$\|\|$	$\times$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	7	$\|\|$	$\|\|$	∞	6
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	7	$\|\|$	$\|\|$	11	$\times$
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$	$\|\|$	9	$\|\|$
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\times$	$\|\|$

Retrouver un plus court chemin avec le tableau

Exemple

Distance de A à $I = 9$
Chemin le plus court de A à I (à l'envers) : I–F–B–A

A	B	C	D	E	F	G	H	I	J
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\times$	4	2	∞	1	∞	∞	∞	∞	∞
$\|$	4	2	∞	$\times$	∞	∞	∞	∞	6
$\|$	4	$\times$	∞	$\|$	∞	3	4	∞	6
$\|$	4	$\|$	∞	$\|$	∞	$\times$	4	∞	6
$\|$	$\times$	$\|$	∞	$\|$	7	$\|$	4	∞	6
$\|$	$\|$	$\|$	5	$\|$	7	$\|$	$\times$	∞	6
$\|$	$\|$	$\|$	$\times$	$\|$	7	$\|$	$\|$	∞	6
$\|$	$\|$	$\|$	$\|$	$\|$	7	$\|$	$\|$	11	$\times$
$\|$	$\|$	$\|$	$\|$	$\|$	$\times$	$\|$	$\|$	9	$\|$
$\|$	$\|$	$\|$	$\|$	$\|$	$\|$	$\|$	$\|$	$\times$	$\|$

Conclusion

Nous avons vu :

- L'algorithme de Dijkstra pour calculer des plus courts chemins depuis un sommet source vers tous les autres sommets d'un graphe (dont les arêtes comportent des poids positifs modélisant des distances)
- Une manière simple (pour un humain) de présenter l'exécution de Dijkstra avec un tableau

Sommaire

Pourquoi l'algorithme de Dijkstra calcule-t-il les distances avec le sommet de départ ?

- L'algorithme de Dijkstra s'arrête toujours

À chaque étape, on marque un sommet non marqué.
On ne démarque jamais un sommet marqué.
On s'arrête quand tous les sommets sont marqués, ce qui finit forcément par se produire.

Pourquoi l'algorithme de Dijkstra calcule-t-il les distances avec le sommet de départ ?

- La distance des sommets marqués ne change plus

On met à jour les valeurs uniquement pour des sommets non marqués.
On ne démarque jamais un sommet marqué.

Pourquoi l'algorithme de Dijkstra calcule-t-il les distances avec le sommet de départ ?

- La distance des sommets marqués est correcte

À chaque étape, la valeur ($< \infty$) d'un sommet Y non marqué représente le poids d'une chaîne entre le sommet de départ et Y.
Quand on met à jour les valeurs, on essaye toujours de les faire diminuer, c-à-d. de trouver une chaîne de poids plus faible.
Quand on marque un sommet, on décide qu'une chaîne de poids minimum a été trouvée. C'est ce qu'il reste à vérifier.

Pourquoi l'algorithme de Dijkstra calcule-t-il les distances avec le sommet de départ ?

Prouvons la propriété suivante, par récurrence :

Hypothèse de récurrence

La valeur des sommets déjà marqués est la distance minimum au sommet de départ A. De plus, les sommets non marqués sont tous "plus loin" de A que les sommets déjà marqués.

Pourquoi l'algorithme de Dijkstra calcule-t-il les distances avec le sommet de départ ?

Prouvons la propriété suivante, par récurrence :

Hypothèse de récurrence

La valeur des sommets déjà marqués est la distance minimum au sommet de départ A. De plus, les sommets non marqués sont tous "plus loin" de A que les sommets déjà marqués.

Déjà, la propriété est vraie à l'initialisation de la méthode :

(0) est bien la distance entre le sommet de départ et lui-même, et tous les autres sommets sont à distance ≥ 0.

Pourquoi l'algorithme de Dijkstra calcule-t-il les distances avec le sommet de départ ?

Prouvons la propriété suivante, par récurrence :

Hypothèse de récurrence

La valeur des sommets déjà marqués est la distance minimum au sommet de départ A. De plus, les sommets non marqués sont tous "plus loin" de A que les sommets déjà marqués.

Ensuite, à chaque nouvelle étape, on compare des valeurs du type $Val(Y) = d(A, X) + Poids(X, Y)$ pour X un sommet marqué et Y un sommet non marqué, et on marque le sommet Y^* de plus petite valeur. Alors, le plus court chemin de A jusqu'à Y^* passe uniquement par des sommets X déjà marqués (sinon contradiction de l'hypothèse de récurrence). La distance de A à Y^* est donc bien donnée par le nombre $Val(Y^*)$ qu'on vient de calculer. La propriété de récurrence reste donc vraie pour le nouveau sommet marqué, Y^*.

Pourquoi l'algorithme de Dijkstra calcule-t-il les distances avec le sommet de départ ?

Prouvons la propriété suivante, par récurrence :

Hypothèse de récurrence

La valeur des sommets déjà marqués est la distance minimum au sommet de départ A. De plus, les sommets non marqués sont tous "plus loin" de A que les sommets déjà marqués.

Par récurrence, la propriété est donc vraie pour tous les sommets du graphe.

Efficacité ?

Analyse de notre version

- Autant de passage dans la boucle de marquage que de sommets (v)
  - recherche du sommet courant (plus proche) on scanne tous les sommets restants (au plus v)
  - découverte de nouveaux sommets/chemins différents (au plus v si on scanne une matrice d'adjacence)
  - $+$ coût vérification /calcul nouveau chemin (somme de distances)
- Le nombre d'étapes est donc proportionnel à v^2.
- Le temps de calcul à chaque étape est le temps requis pour calculer la somme des distances. On peut le considérer comme une constante (si toutes les distances sont codées sur 64 bits).

Conclusion Efficacité

Temps de calcul de l'algorithme de Dijkstra de l'ordre de v^2

Version améliorée de Dijkstra

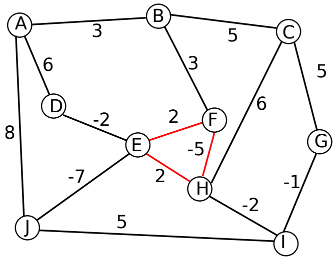
- Il est possible d'améliorer le temps de calcul, dans le cas particulier où le graphe n'est pas très dense (nombre d'arêtes e beaucoup plus petit que v^2).
- L'idée est d'utiliser une structure de données particulière appelée le tas de Fibonacci, qui permet de gérer efficacement la recherche du sommet courant (sommet min non marqué), l'insertion de nouveaux sommets, et la mise à jour des distances.
- Dans ce cas, nombre d'étapes : $e + v\log(v)$ où v est le nombre de sommets et e le nombre d'arêtes. Ce qui est mieux (plus rapide) que notre complexité précédente en v^2.
- Dans le cas du routage, les graphes sont en effet peu denses, ce qui explique le choix de l'algorithme de Dijkstra (dans sa version optimisée).

Sommaire

Inconvénient : L'algo de Dijkstra ne marche pas si on a des poids négatifs.

Les poids négatifs peuvent être utiles, par exemple :
poids négatifs = dépenses, poids positifs = gains

L'algorithme de Bellman-Ford (moins efficace que Dijkstra) permet de gérer les poids négatifs dans les graphes sans cycles négatifs.
(pas au programme !)



Si on a un cycle négatif, la notion de distance n'a pas de sens !

Sommaire

Variante : A^* (en anglais on dit A-star)

Peter E. Hart, Nils John Nilsson et Bertram Raphael (1968)

Question

Calculer une distance depuis une position vers une autre

Conditions spéciales

Variante : A^* (en anglais on dit *A*-star)

Peter E. Hart, Nils John Nilsson et Bertram Raphael (1968)

Question

Calculer une distance depuis une position vers une autre

Conditions spéciales

- Graphe très grand (robot sur mars) $\Longrightarrow$ On ne peut pas tout regarder

Variante : A^* (en anglais on dit A-star)

Peter E. Hart, Nils John Nilsson et Bertram Raphael (1968)

Question

Calculer une distance depuis une position vers une autre

Conditions spéciales

- Graphe très grand (robot sur mars) $\Longrightarrow$ On ne peut pas tout regarder
- Les calculs doivent être rapides et ne pas coûter cher (le robot est tout le temps en train de calculer des chemins et il faut économiser sa batterie)

Variante : A^* (en anglais on dit A-star)

Peter E. Hart, Nils John Nilsson et Bertram Raphael (1968)

Question

Calculer une distance depuis une position vers une autre

Conditions spéciales

- Graphe très grand (robot sur mars) $\Longrightarrow$ On ne peut pas tout regarder
- Les calculs doivent être rapides et ne pas coûter cher (le robot est tout le temps en train de calculer des chemins et il faut économiser sa batterie)
- Pas n'importe quel graphe (surface de mars : la ligne droite c'est pas mal, sauf si il y a un trou, une crevasse)

Variante : A^* (en anglais on dit A-star)

Peter E. Hart, Nils John Nilsson et Bertram Raphael (1968)

Question

Calculer une distance depuis une position vers une autre

Conditions spéciales

- Graphe très grand (robot sur mars) $\Longrightarrow$ On ne peut pas tout regarder
- Les calculs doivent être rapides et ne pas coûter cher (le robot est tout le temps en train de calculer des chemins et il faut économiser sa batterie)
- Pas n'importe quel graphe (surface de mars : la ligne droite c'est pas mal, sauf si il y a un trou, une crevasse)
- On peut utiliser la distance à vol d'oiseau pour guider la recherche du chemin et ne pas tout explorer.

Sommaire

Sommaire

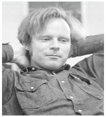
Algorithme de Floyd–Warshall : généralités

Question

Calculer les distances entre toutes les paires de sommets (plus efficacement que de lancer Dijkstra sur chaque sommet).

Remarques

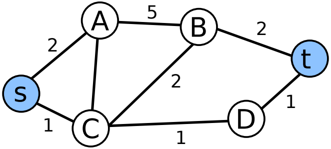
- conçu en 1962
- Algorithme conceptuellement plus simple que celui de Dijkstra.
- Complexité : v^3 où v est le nombre de sommets.
- Fonctionne aussi pour les poids négatifs



Robert W. Floyd (1936-2001)

Stephen Warshall (1935-2006)

Algorithme de Floyd–Warshall : principe général



On ordonne les sommets : $s_1, \ldots, s_n$.

Soit $d_k(i,j)$ la longueur d'un plus court chemin de s_i à s_j qui a le droit d'utiliser les sommets $s_1, \ldots, s_k$ comme *sommets internes*.

On peut écrire la formule récursive :

$$d_k(i,j) = \min\{d_{k-1}(i,j), d_{k-1}(i,k) + d_{k-1}(k,j)\}.$$

Algorithme de Floyd–Warshall : pseudo-code

On utilise la formule récursive :

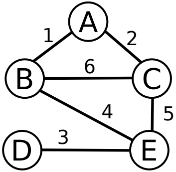
$$d_k(i,j) = \min\{d_{k-1}(i,j), d_{k-1}(i,k) + d_{k-1}(k,j)\}.$$

Algorithme de Floyd-Warshall pour le graphe G

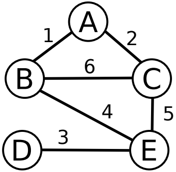
- Les sommets sont ordonnés : s_1 ... s_n
- On initialise une matrice D de taille $n \times n$, où $D(i,j)$ devra contenir la distance entre le sommet s_i et le sommet s_j. Pour toute arête entre s_i et s_j de poids p_{ij}, on fixe $D(i,j) = p_{ij}$, et pour tout i on fixe $D(i,i) = 0$. Dans les autres cas, on fixe $D(i,j) = \infty$.
- Pour k allant de 1 à n :
  - Pour i allant de 1 à n :
    - Pour j allant de 1 à n : $D(i,j) = \min\{D(i,j), D(i,k) + D(k,j)\}$
- Renvoyer D

Sommaire

Algorithme de Floyd–Warshall : exemple

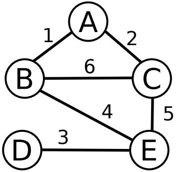


Algorithme de Floyd–Warshall : exemple



D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

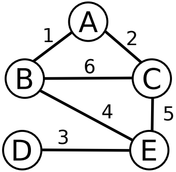
Algorithme de Floyd–Warshall : exemple



D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

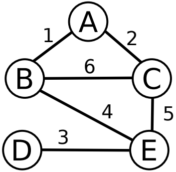
Algorithme de Floyd–Warshall : exemple



D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

Algorithme de Floyd–Warshall : exemple

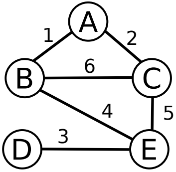


D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

Algorithme de Floyd–Warshall : exemple

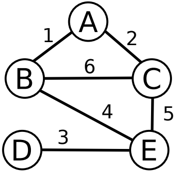


D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_2	A	B	C	D	E
A	0	1	2	∞	**5**
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	**5**	4	5	3	0

Algorithme de Floyd–Warshall : exemple



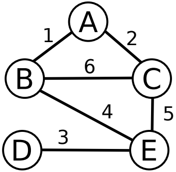
D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_2	A	B	C	D	E
A	0	1	2	∞	**5**
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	**5**	4	5	3	0

D_2	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

Algorithme de Floyd–Warshall : exemple



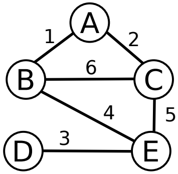
D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_2	A	B	C	D	E
A	0	1	2	∞	**5**
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	**5**	4	5	3	0

D_3	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

Algorithme de Floyd–Warshall : exemple



D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

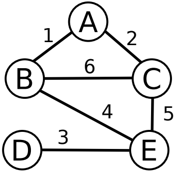
D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_2	A	B	C	D	E
A	0	1	2	∞	**5**
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	**5**	4	5	3	0

D_3	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

D_3	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

Algorithme de Floyd–Warshall : exemple



D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

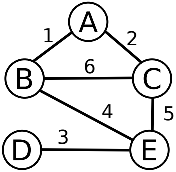
D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_2	A	B	C	D	E
A	0	1	2	∞	**5**
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	**5**	4	5	3	0

D_3	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

D_4	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

Algorithme de Floyd–Warshall : exemple



D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

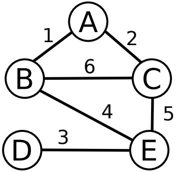
D_2	A	B	C	D	E
A	0	1	2	∞	**5**
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	**5**	4	5	3	0

D_3	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

D_4	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

D_4	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

Algorithme de Floyd–Warshall : exemple



D_0	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	6	∞	4
C	2	6	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_1	A	B	C	D	E
A	0	1	2	∞	∞
B	1	0	**3**	∞	4
C	2	**3**	0	∞	5
D	∞	∞	∞	0	3
E	∞	4	5	3	0

D_2	A	B	C	D	E
A	0	1	2	∞	**5**
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	**5**	4	5	3	0

D_3	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

D_4	A	B	C	D	E
A	0	1	2	∞	5
B	1	0	3	∞	4
C	2	3	0	∞	5
D	∞	∞	∞	0	3
E	5	4	5	3	0

D_5	A	B	C	D	E
A	0	1	2	**8**	5
B	1	0	3	**7**	4
C	2	3	0	**8**	5
D	**8**	**7**	**8**	0	3
E	5	4	5	3	0

Sommaire

Conclusion

Aujourd'hui nous avons vu :

- L'algorithme de Dijkstra pour calculer des plus courts chemins depuis un sommet source vers tous les autres sommets d'un graphe valué (dont les arêtes comportent des poids positifs)
- Une manière simple (pour un humain) de présenter l'exécution de Dijkstra avec un tableau
- L'algorithme de Floyd-Warshall pour calculer toutes les paires de distances.