Université Clermont Auvergne
BUT Info 1A
R207 - Graphes
Année 2021-2022

Florent Foucaud
Malika More
Thibault Ralet
Carine Simon
Thierry Trévisan

TD4 - Calculs des distances

Rappel de cours

0.1 Algorithme de Dijkstra

Cet algorithme permet de calculer les distances depuis un sommet source dans les *graphes valués* (avec des poids non-négatifs sur les arêtes). Dans un graphe valué, la *distance* entre deux sommets est la plus petite somme des poids des arêtes parmi les chaînes qui relient les deux sommets. Il ressemble au parcours en largeur (qui permet aussi de calculer les distances à la source mais seulement dans les graphes non-valués).

L'algorithme de Dijkstra fonctionne aussi pour les graphes valués *orientés*.

`Algorithme de Dijkstra pour le graphe G à partir du sommet source s`

- `L représentera la frontière. Contient initialement seulement s.`
- `La valeur de distance d est initialisée à d(s)=0 et à d(v)=`∞ `pour chaque autre sommet v`
- `Une liste T contient les sommets qui ont été complètement traités. Initialement T est vide.`
- `Tant que L n'est pas vide :`
  - ⋆ `choisir un sommet v dans L qui a une valeur de distance d(v) minimale`
  - ⋆ `pour tout voisin w de v qui n'est pas dans T :`
    - `si d(v) plus le poids p de l'arête (v,w) est inférieur à d(w), on fixe d(w)=d(v)+p`
    - `ajouter w à L`
  - ⋆ `enlever v de L, ajouter v à T`

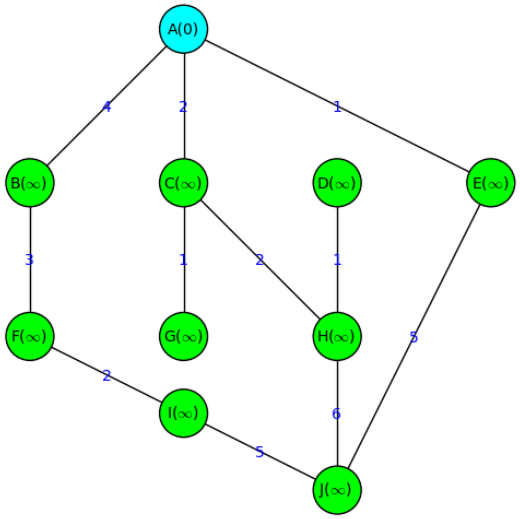
0.2 Dijkstra pour les humains en version tableau

Voici la méthode que l'on va appliquer pour dérouler Dijkstra sur feuille.

Etape 1 **(Initialisation)**

- Placer tous les sommets du graphe dans la 1$^{\text{ère}}$ ligne d'un tableau

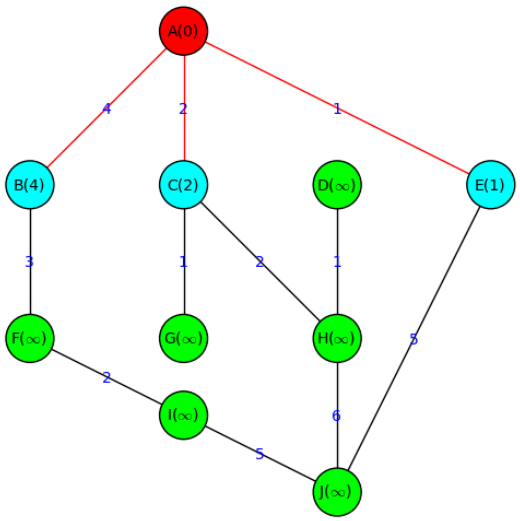
- Sur la 2ème ligne, écrire la valeur 0 pour le sommet de départ et les valeurs ∞ pour tous les autres sommets



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

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.



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	×
||	||	||	||	||	×	||	||	9	||
||	||	||	||	||	||	||	||	×	||

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

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

0.3 Algorithme de Floyd-Warshall

L'algorithme de Floyd-Warshall permet de calculer en une exécution, les distances entre *toutes* les paires de sommets d'un graphe valué (il peut être orienté), tant qu'il n'existe pas de cycle de poids total négatif.

On va faire n itérations (où n est le nombre de sommets). A l'itération numéro k, pour toute paire de sommets, on considère les chemins possibles entre ces deux sommets qui peuvent emprunter les sommets $s_1, \ldots, s_k$ comme *sommets internes* du chemin. Pour cela on utilise la formule récursive suivante (où $d_k(i,j)$ désigne la longueur d'un plus court chemin entre s_i et s_j qui est autorisé à emprunter les sommets $s_1, \ldots, s_k$ comme sommets internes) :

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

En effet, un chemin de s_i à s_j qui peut emprunter les sommets $s_1, \ldots, s_k$ comme sommets internes utilise soit uniquement les sommets $s_1, \ldots, s_{k-1}$ comme sommets internes, soit il passe par s_k et dans ce cas c'est un chemin le plus court entre s_i et s_k qui utilise $s_1, \ldots, s_{k-1}$ comme sommets internes, suivi d'un chemin le plus court entre s_k et s_j qui utilise aussi $s_1, \ldots, s_{k-1}$ comme sommets internes.

Cette formule permet donc de calculer itérativement tous les plus courts chemins, puisque la distance entre s_i et s_j est égale à la valeur $d_n(i,j)$ (on a le droit d'utiliser n'importe quels sommets comme sommets internes du plus court chemin).

Remarque : À l'étape k, on a $d_k(k,j) = d_{k-1}(k,j)$ et $d_k(i,k) = d_{k-1}(i,k)$.

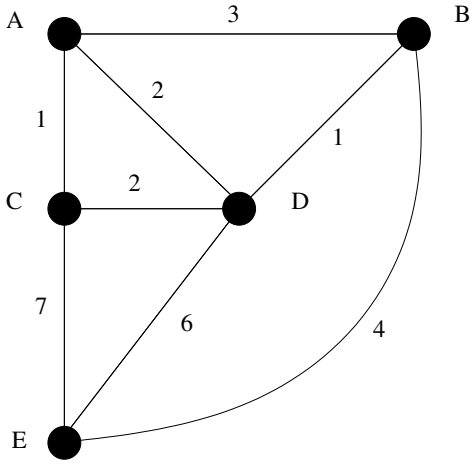
L'algorithme peut s'écrire de la façon suivante :

`Algorithme de Floyd-Warshall pour le graphe G`

- `Les sommets sont ordonnés :` s_1 `...` s_n `avec` n `le nombre de sommets`
- `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

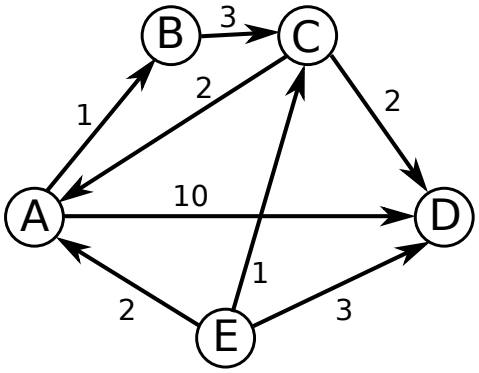
Exercice 1 (Dijkstra)**.** Exécutez l'algorithme de Dijkstra sur le graphe valué G ci-dessous en partant du sommet A. Il faudra rédiger votre réponse en détaillant dans un tableau toutes les étapes.



A	B	C	D	E

Exercice 2 (Dijkstra orienté)**.** Exécutez l'algorithme de Dijkstra sur le graphe orienté valué G ci-dessous en partant du sommet A. Il faudra rédiger votre réponse en détaillant dans un tableau toutes les étapes.

Qu'observe-t-on pour le sommet E ?



A	B	C	D	E

Exercice 3. Montrer que dans un graphe où toutes les arêtes ont le même poids, l'algorithme de Dijkstra effectue un parcours en largeur.

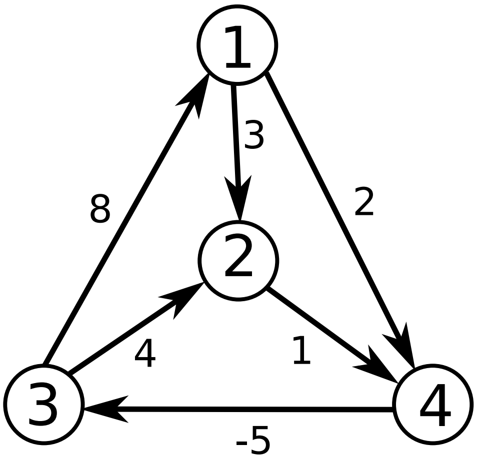
Exercice 4.

(a) Cela a-t-il du sens de calculer les distances dans un graphe qui a un cycle de poids total négatif ?

(b) Pourquoi l'algorithme de Dijkstra ne fonctionne-t-il pas lorsque les arcs peuvent avoir des poids négatifs ? Trouver un contre-exemple avec trois sommets.

Exercice 5 (Floyd-Warshall orienté avec poids négatif)**.**

1. Exécutez l'algorithme de Floyd-Warshall sur le graphe orienté valué G ci-dessous. Remplissez pour cela les différents états de la matrice.



	1	2	3	4
1				
2				
3				
4				

	1	2	3	4
1				
2				
3				
4				

	1	2	3	4
1				
2				
3				
4				

	1	2	3	4
1				
2				
3				
4				

	1	2	3	4
1				
2				
3				
4				

2. Si l'on programme l'algorithme, pourquoi peut-on éviter de faire une copie de la matrice à chaque itération ? Pourquoi faire une telle copie peut quand même être utile ?

Exercice 6 (Algorithme A^* - Hart-Nilsson-Raphael, 1968)**.**

L'algorithme A^* est une modification de l'algorithme de Dijkstra, dans le cas où on veut trouver un plus court chemin entre une unique paire de sommets. Le but est d'obtenir une solution plus rapidement. C'est un *algorithme informé* car il a besoin d'informations supplémentaires : une estimation préalable de la distance, appelée *heuristique*.

Par exemple, dans le problème du navigateur GPS dans un réseau routier, on peut prendre en compte la distance Euclidienne, pour éviter de passer par des chemins de traverse (qui ont peu de chances de donner un plus court chemin).

Cependant, cette amélioration de performances à un prix : il est possible que le chemin trouvé ne soit pas optimal, si les estimations au départ ne sont pas bonnes.

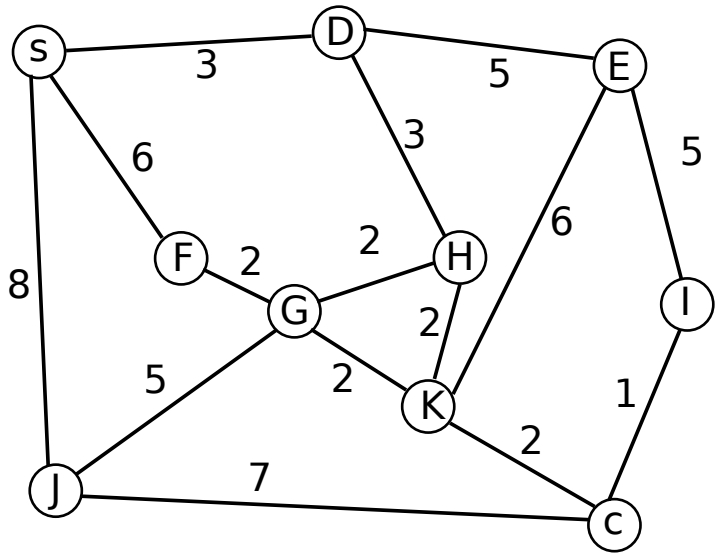
Voici l'algorithme :

Algorithme A^* pour le graphe G à partir du sommet source s, avec le sommet c comme cible

- L représentera la frontière. Contient initialement seulement s.
- La valeur de distance d est initialisée à d(s)=0 et à d(v)=∞ pour chaque autre sommet v
- pred[v] représente le prédecesseur d'un sommet v sur un plus court chemin depuis s. Initialement, pred[v]=None pour tout sommet v.
- Une liste T contient les sommets qui ont été complètement traités. Initialement T est vide.
- Pour un sommet v, h(v) représente l'estimation de la distance de v à la cible c.
- Tant que L n'est pas vide :
  - ⋆ choisir un sommet de L qui minimise h(v)+d(v)
  - ⋆ Si v=c, on arrête l'algorithme et on reconstruit le chemin de s à c en utilisant les valeurs dans pred
  - ⋆ pour tout voisin w de v qui n'est pas dans T :
    - si d(v) plus le poids p de l'arête (v,w) est inférieur à d(w), on fixe d(w)=d(v)+p et on fixe pred[w]=v.
    - ajouter w à L
  - ⋆ ajouter v à T et enlever v de L
- Afficher "il n'existe aucun chemin de s à c".

1. On peut montrer que si pour tout sommet v, $h(v)$ est toujours inférieur ou égal à la vraie distance à la cible c, alors l'algorithme A^* trouve le bon résultat. On suppose qu'il n'y a pas de poids négatifs. Que se passe-t-il si on fixe $h(v) = 0$ pour tout sommet v ? Comparer l'algorithme A^* à l'algorithme de Dijkstra dans ce cas particulier.
2. On veut trouver un chemin le plus court entre s et c dans le graphe suivant, qui représente un réseau routier. (On peut voir que certaines routes sont plus ou moins rapides par rapport à leur longueur, en fonction des conditions de circulation et des limitations de vitesse.)

Mesurer la distance Euclidienne approximative entre chaque sommet et c, avec une règle (ou une autre mesure que vous avez sous la main), et la noter à côté du sommet. Ce seront les valeurs de l'heuristique h.



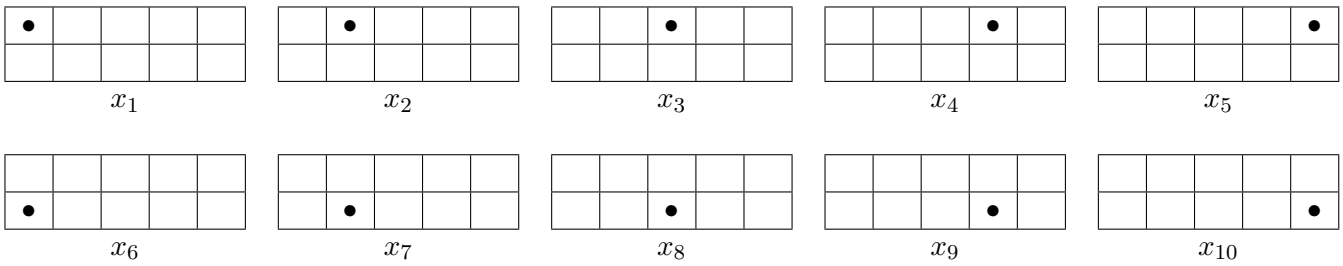
3. Appliquer l'algorithme A^* à ce graphe pour trouver un chemin de s à c. Utiliser la structure du tableau, comme pour Dijkstra. Pour faciliter les calculs, écrivez la valeur de h au-dessus du nom de chaque sommet.

s	D	E	F	G	H	I	J	K	c

4. Un accident se produit en haut du sommet K, et les arêtes $\{H, K\}$ et $\{G, K\}$ ont maintenant un poids de 10 chacune. Les estimations restent cependant les mêmes. Réappliquer l'algorithme A^* au nouveau graphe. Que constate-t-on sur le nombre d'étapes de l'algorithme et la qualité de la solution ?

s	D	E	F	G	H	I	J	K	c

Exercice 7. On dispose d'une grille délimitant 10 cases. Un point se trouvant sur l'une des cases donne lieu à 10 dispositions représentées et indexées ci-dessous :

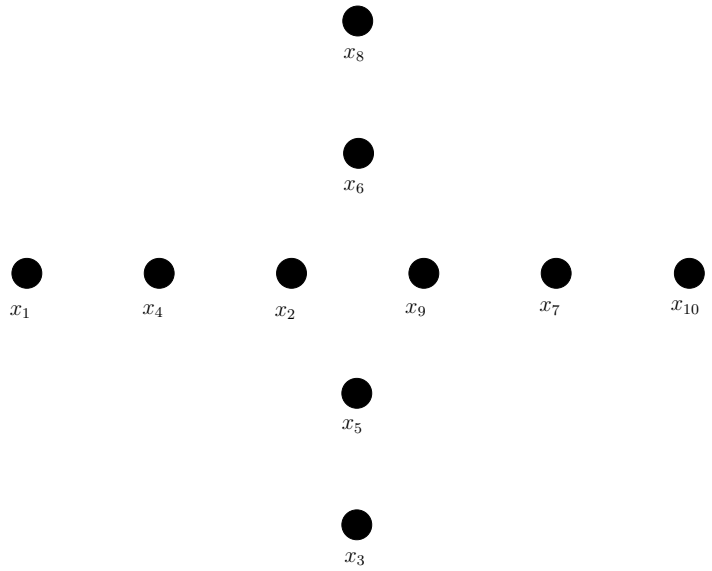


Au départ du jeu, la grille est disposée selon x_1. Le but du jeu est de déplacer le point pour aboutir à la disposition x_6. Pour cela, les déplacements autorisés sont :

- des déplacements horizontaux de trois cases (vers la droite ou la gauche), chaque déplacement coûtant 1 point. Par exemple, on peut ainsi passer de la position x_1 à la position x_4 (ou le contraire). Lorsqu'on rencontre une paroi du cadre, on revient sur ses pas. Par exemple, on peut ainsi passer de la position x_8 à la position x_9 (ou le contraire).
- des déplacements diagonaux d'une case dans la direction (sud-ouest)-(nord-est) (ou le contraire), chaque déplacement coûtant 2 points. Ce déplacement est impossible à partir de la disposition x_1 ou de la disposition x_{10}. Le déplacement suivant l'autre diagonale (direction (nord-ouest)-(sud-est)) est impossible également. Par exemple, on peut ainsi passer de la position x_4 à la position x_8 (ou le contraire), mais pas de la position x_3 à la position x_9.

On se propose de déterminer le coût minimal permettant de gagner à ce jeu.

1. Complétez le *graphe des états* du jeu : les sommets sont les dispositions x_i ($1 \leq i \leq 10$), et deux dispositions sont reliées dans le graphe si et seulement si un déplacement élémentaire permet de passer d'une disposition à l'autre. Les arêtes sont valuées par le coût (1 ou 2) du déplacement élémentaire correspondant.



2. Appliquez l'algorithme de Dijsktra à ce graphe, au départ du sommet x_1. Quel est le coût minimum permettant de gagner à ce jeu, et quels sont les déplacements à effectuer pour obtenir ce minimum ?

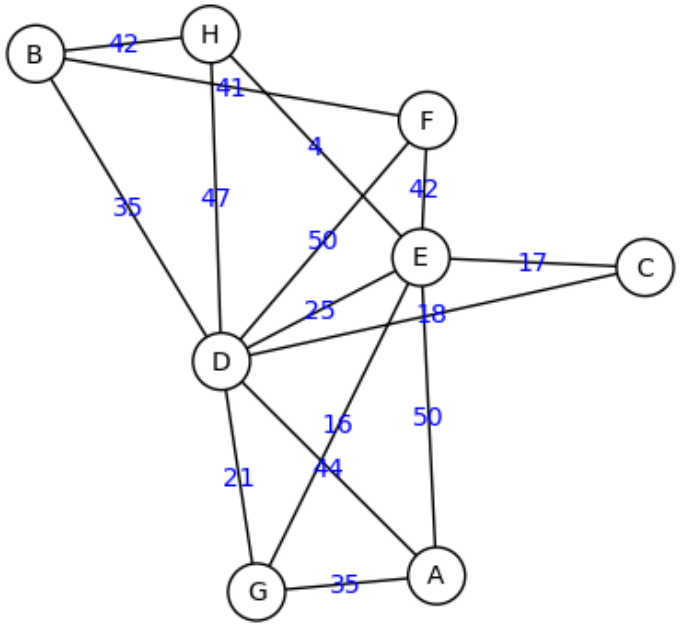
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}

Exercice 8 (Énigme du IXe siècle)**.** Un paysan doit passer une rivière dans une barque juste assez grande pour lui et son loup, ou lui et sa chèvre, ou lui et ses choux. Les choux seront

mangés s'il les laisse seuls avec la chèvre, et la chèvre sera mangée s'il la laisse seule avec le loup. Quelle est la solution la plus rapide pour faire passer tout ce monde sain et sauf ?

Modéliser la situation par un *graphe des états*. Un état est un couple (X, Y) où X est l'ensemble des protagonistes sur la rive de départ et Y ceux sur la rive d'arrivée. Chaque état (sécurisé) possible est un sommet. On a une arête entre deux états si on peut passer de l'un à l'autre par un voyage en barque. Avant de dessiner le graphe, essayer de compter le nombre de sommets qu'il aura.

Exercice 9 (Dijkstra, le retour)**.** En appliquant l'algorithme de Dijkstra, calculez la distance entre le sommet A et chaque autre sommet du graphe. Il faudra rédiger votre réponse en détaillant dans un tableau toutes les étapes.



A	B	C	D	E	F	G	H