Une introduction à l'algorithme de Dijkstra

Quelques éléments de théorie des graphes apparaissent dans le programme de terminale ES, spécialité mathématiques. L'intérêt de ceux-ci est multiple[1] : mettre les élèves au contact de nouveaux objets et de nouveaux types de raisonnements, travailler sur des situations concrètes[2], faire le lien avec d'autres parties du programme de terminale ES [3], etc.

Parmi ces éléments de théorie des graphes figure la recherche d'un plus court chemin dans un graphe pondéré, à l'aide de l'algorithme de Dijkstra. Cette partie du cours satisfait *a priori* les avantages évoqués ci-dessus : résolution de problèmes naturels, lien avec l'enseignement d'informatique, et un type de calcul mis en jeu incontestablement nouveau pour les élèves.

Cependant, si l'on peut apprécier l'introduction de l'algorithme de Dijkstra en terminale ES, l'avis de l'auteur est que l'implémentation effective de cet algorithme[4] est douteuse : elle réduit la recherche d'un plus court chemin au remplissage d'un tableau suivant des règles arbitraires. Dès lors, le risque est qu'au lieu de faire raisonner les élèves, on leur fasse seulement retenir ces règles jusqu'à leur baccalauréat.

Cette note a deux objets. Le premier est de présenter une version simplifiée de l'algorithme de Dijkstra, à notre sens plus adaptée au public visé, puis de montrer comment retrouver le vrai algorithme de Dijkstra à partir de cette version simplifiée. L'autre but est de discuter de l'intérêt de la version simplifiée dans l'enseignement de théorie des graphes en terminale ES ; cette discussion a lieu en Sous-partie 2.3.

1 Un algorithme de Dijkstra moins efficace

Le but de l'algorithme de Dijkstra est de trouver un chemin le plus court entre deux sommets dans un graphe pondéré. Ses applications sont évidentes ; par exemple, il permet en théorie de trouver l'itinéraire, à pied ou en voiture, le plus rapide entre deux points du globe[5]. Pour faciliter notre présentation, nous commençons par en donner une version qui ne fait que calculer la distance entre deux sommets.

On se donne donc :

- un graphe fini $G = (V, A)$, simple et non orienté[6] ;
- pour chaque arête $e \in A$, un poids réel strictement positif $\ell(e) > 0$;
- deux sommets distincts de G, que nous noterons E et S pour entrée et sortie.

Les poids sur les arêtes représentent leur longueur. Le problème à résoudre est le suivant :

Trouver le chemin le plus court de E à S.

Notre version simplifiée de l'algorithme de Dijkstra offre une solution élégante, dont on peut ainsi résumer le principe :

Pour tout $n \geq 1$, on calcule récursivement les n sommets les plus proches de E, leur distance à E, et un chemin optimal les reliant à E.

1.1 Formalisation de l'algorithme

Nous allons maintenant formaliser cet algorithme. Il peut être utile de suivre l'exemple de la partie 1.2 pour mieux comprendre son fonctionnement.

DONNÉES DU PROBLÈME

Comme mentionné ci-dessus, il s'agit d'un graphe pondéré, d'un sommet d'entrée E, et d'un sommet de sortie S.

Pour tout $n \geq 1$, l'ensemble des données à calculer est composé :

- d'un sous-ensemble de sommets $L_n = \{v_1, \dots, v_n\} \subset V$;
- pour chacun de ces sommets v_k, d'un réel $\delta(v_k) \geq 0$;

tels que L_n est un ensemble[7] de n sommets les plus proches de E, et $\delta(v_k) = d(E, v_k)$.

INITIALISATION

Pour $n = 1$, on veut le sommet du graphe le plus proche de E. Il s'agit de E lui-même. On a donc :

[1] Voir par exemple : Eduscol, *Accompagnement de la mise en oeuvre des programmes en classe terminale de la série ES*. `http://eduscol.education.fr/cid45765/accompagnement-des-programmes-de-terminale-es.html`, août 2018.

[2] L'enseignement de théorie des graphes se fonde ainsi sur la résolution de problèmes plutôt que sur le cours magistral.

[3] Voir par exemple le dénombrement de chemins sur des graphes, qui fait intervenir de l'algèbre linéaire.

[4] Voir par exemple : `http://media.eduscol.education.fr/file/Programmes/16/8/graphes_109168.pdf`, page 15.

[5] En pratique, on utilise pour cela des algorithmes donnant des solutions potentiellement un peu moins bonnes, mais beaucoup plus rapidement.

[6] Ces deux hypothèses ne sont pas nécessaires pour concevoir l'algorithme de Dijkstra. Elles ne servent ici qu'à simplifier l'exposition. Voir la Sous-partie 1.4 pour une discussion.

[7] Non nécessairement unique, si plusieurs sommets sont à la même distance de E.

- $L_1 = \{E\}$;
- $\delta(E) = d(E, E) = 0$.

CALCUL RÉCURSIF

Il faut maintenant comprendre comment calculer les ensembles L_n récursivement, c'est-à-dire comment calculer L_{n+1} à partir de L_n. Cela revient à trouver un sommet B qui est un des $n+1$-èmes sommets les plus proche de E, et sa distance $\delta(B)$ à E. Cela se fait grâce au lemme suivant :

Lemme 1.1. *Soit (V, A) un graphe pondéré et $n \geq 1$. Soient L_n et δ définies comme ci-dessus. Posons :*

$$d := \min_{\substack{[C'B'] \in A \\ B' \in V \setminus L_n,\ C' \in L_n}} \{\delta(C') + \ell([C'B'])\}.$$

Soit $B \in V \setminus L_n$ réalisant le minimum ci-dessus, c'est-à-dire tel que :

$$d = \min_{\substack{[C'B] \in A \\ C' \in L_n}} \{\delta(C') + \ell([C'B])\}.$$

Alors on peut choisir $L_{n+1} = L_n \cup \{B\}$, et $\delta(B) = d$.

Proof. Le nombre d est infini si et seulement si aucune arête ne relie un sommet de L_n à un sommet de $V \setminus L_n$; dans ce cas, il n'y a rien à dire. Supposons d fini, et soit B un sommet réalisant le minimum.

Soit $C \in L_n$ tel que $d = \delta(C) + \ell([CB])$. Soit γ' un chemin le plus court de E à C, et soit γ le chemin obtenu en parcourant le chemin γ' puis l'arête $[CB]$. Alors :

$$\begin{aligned}\ell(\gamma) &= \ell(\gamma') + \ell([CB]) \\ &= d(E, C) + \ell([CB]) \\ &= \delta(C) + \ell([CB]) \\ &= d,\end{aligned}$$

et donc $d(E, B) \leq d$.

Maintenant, soit $B' \in V \setminus L_n$ un des $n+1$ sommets (de V) les plus proches de E, et soit $\Gamma = (\Gamma(0), \dots, \Gamma(m))$ un chemin le plus court de E à B'. Remarquons que B' est un des sommets de $V \setminus L_n$ les plus proches de E, donc Γ est un chemin le plus court reliant E à un sommet de $V \setminus L_n$.

Le sommet d'arrivée de Γ est $B' \in V \setminus L_n$. Soit C' le sommet précédent sur ce chemin, et posons $\Gamma' = (\Gamma(0), \dots, \Gamma(m-1))$, qui est un chemin de E à C'. Le chemin Γ' est strictement plus court que Γ, car on a enlevé une arête. Comme Γ est un chemin le plus court reliant un élément de E à $V \setminus L_n$, cela implique que le point d'arrivée de Γ' n'est pas dans $V \setminus L_n$. Donc $C' \in L_n$.

De plus, Γ' est lui-même un chemin le plus court reliant E à C. En effet, s'il existait un autre chemin strictement plus court Γ'' de E à C, on pourrait relier E à B en empruntant Γ'' puis $[CB']$, ce qui serait plus court que d'emprunter Γ et contredirait l'hypothèse de minimalité sur Γ. Par conséquent,

$$\begin{aligned}d(E, B') &= \ell(\Gamma) \\ &= \ell(\Gamma') + \ell([C'B']) \\ &= d(E, C') + \ell([C'B']) \\ &= \delta(C') + \ell([C'B']) \\ &\geq d.\end{aligned}$$

Finalement, $d(E, C') \geq d \geq d(E, B)$ par les calculs précédents, et $d(E, B) \geq d(E, B')$ car B' est l'un des sommets de $V \setminus L_n$ les plus proches de E. On a donc bien $d(E, B) = d$, et B est l'un des sommets de $V \setminus L_n$ les plus proches de E. □

Pour trouver un sommet B que l'on peut ajouter à L_n pour obtenir L_{n+1}, il suffit donc de considérer toutes les arêtes reliant un sommet $B' \in L_n$ et un sommet $C' \in V \setminus L_n$, de calculer la quantité $\delta(C') + \ell([C'B'])$ pour chacune d'entre elles, et de choisir un des sommets réalisant le minimum. La quantité $\delta(B)$ est alors ce minimum.

Dans la suite, nous parlerons de sommets *visités* au temps n pour ceux appartenant à L_n, et de sommets non visités pour les autres.

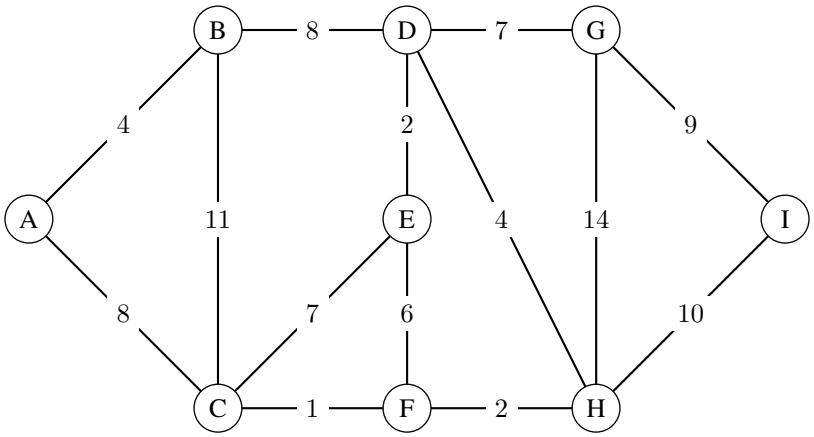
CONDITIONS D'ARRÊT

Rappelons que le but est de calculer $d(E, S)$, où S est un sommet donné. Il y a deux possibilités :

- à une étape, le sommet ajouté à l'ensemble L_n est S. Dans ce cas, on calcule au passage $\delta(S) = d(E, S)$, et l'on peut arrêter l'algorithme.
- à une étape, on ne peut pas agrandir L_n, parce qu'il n'existe pas d'arête reliant un sommet de L_n et un sommet de $V \setminus L_n$. Dans ce cas, L_n est la composante connexe de E, et S n'y appartient pas : il n'existe pas de chemin de S à E. On arrête l'algorithme, et on écrit $d(E, S) = +\infty$.

1.2 Exemple

Un dessin valant mieux qu'un long discours, voici un exemple d'application de l'algorithme ci-dessus. Dans le graphe pondéré suivant, on veut trouver le chemin le plus court du sommet A au sommet I.

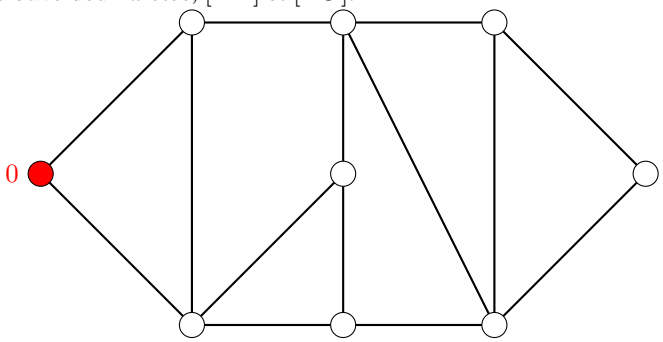


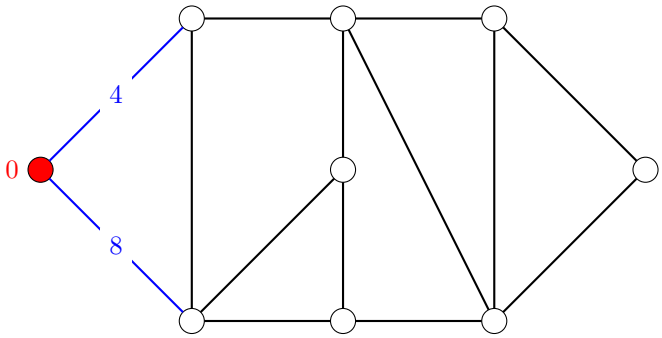
Dans ce qui suit, nous affichons :

- dans la colonne de gauche : les sommets visités à l'étape n sont en rouge, et la valeur de la fonction δ pour chacun de ces sommets est affichée en rouge.
- dans la colonne de droite : les arêtes ayant une extrémité dans L_n et l'autre dans $V \setminus L_n$ sont affichées en bleu.

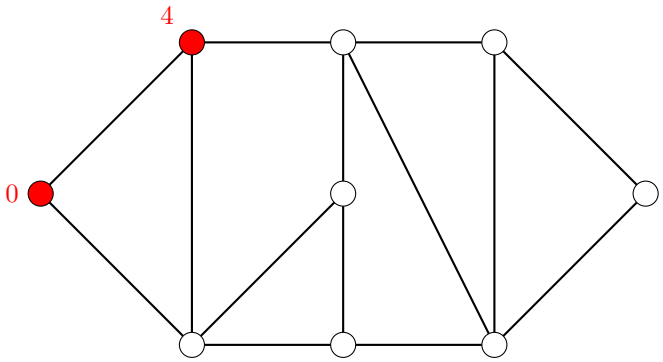
Afin de ne pas surcharger le graphe, nous ne reproduisons pas le nom des sommets, et seul le poids des arêtes considérées à l'étape courante est affiché.

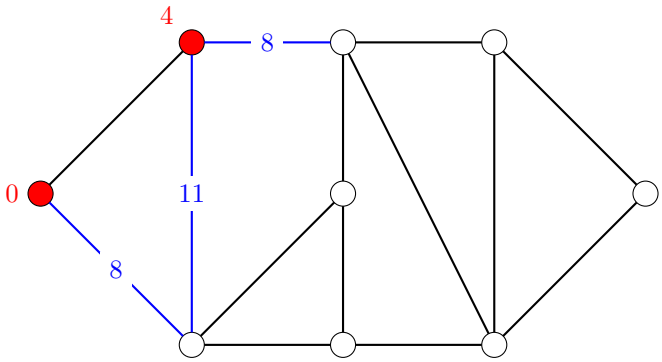
Le sommet de départ est A. À gauche, l'initialisation : seul le sommet A est en rouge, à une distance de 0 de A. À droite, le premier pas : on considère toutes les arêtes e dont A est une extrémité, et on recherche celle qui minimise la quantité $\delta(A) + \ell(e) = \ell(e)$. On trouve deux arêtes, $[AB]$ et $[AC]$.



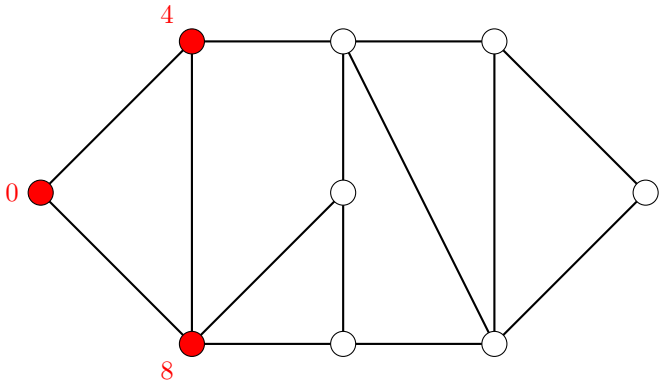


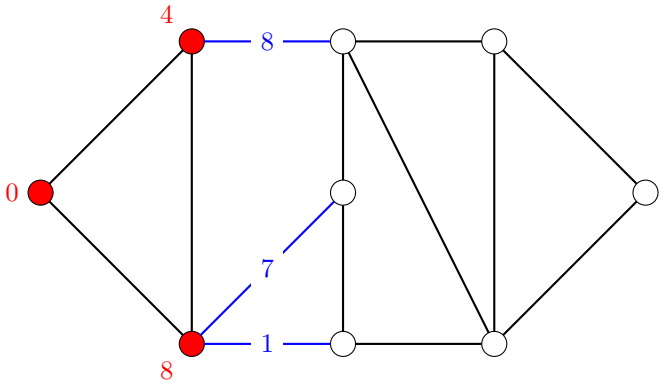
L'arête dont la longueur est minimale est $[AB]$, qui est de longueur 4. On rajoute le sommet B, et donc $L_2 = \{A, B\}$. Ensuite, il faut considérer toutes les arêtes qui partent de A ou de B, et qui mènent à un sommet qui n'est ni A ni B. Il y en a 3 : $[AC]$, $[BC]$ et $[BD]$.



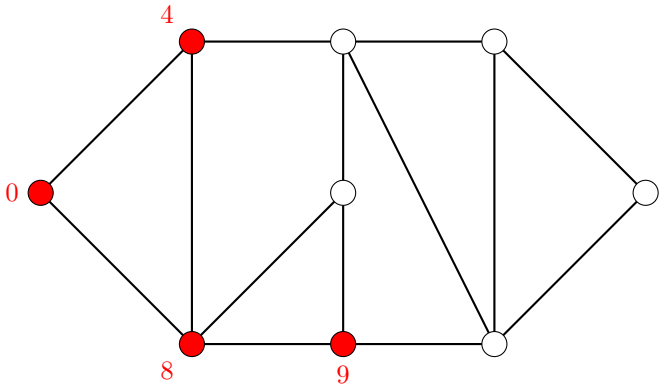


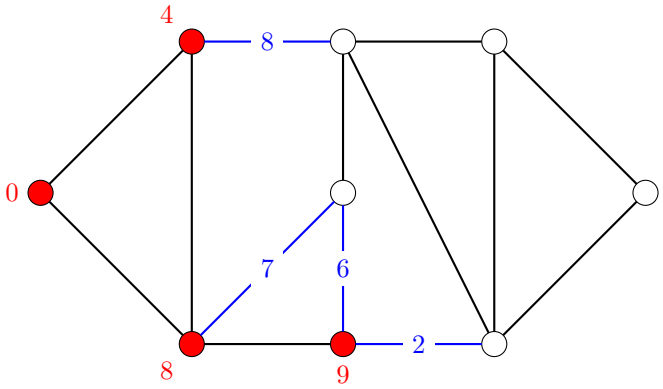
Les valeurs de la fonction $\delta + \ell$ sont $0 + 8 = 8$ pour $[AC]$, puis $4 + 11 = 15$ pour $[BC]$ et $4 + 8 = 12$ pour $[BD]$. La valeur minimale est 8 pour $[AC]$; on rajoute donc le sommet C. On a donc $L_3 = \{A, B, C\}$. Les arêtes suivantes à considérer sont $[BD]$, $[CE]$ et $[CF]$.



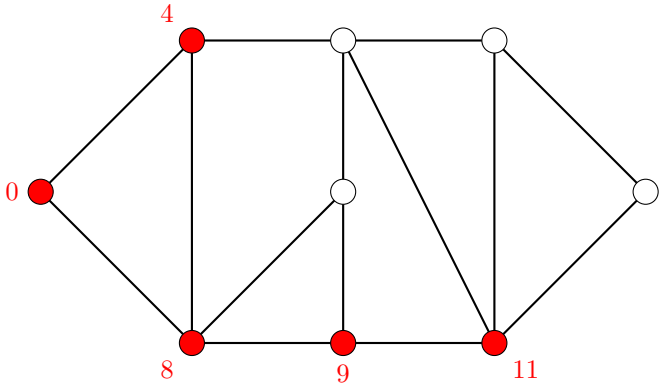


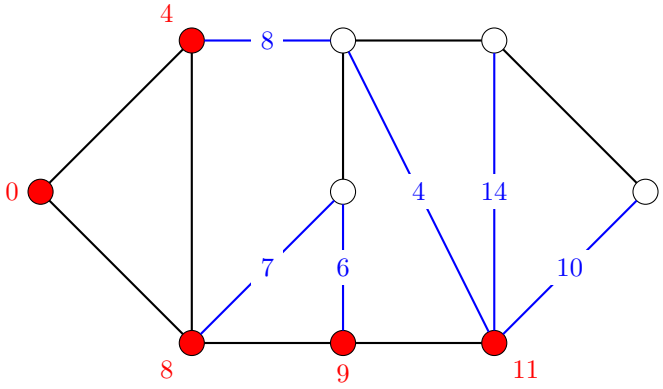
La valeur minimale de la fonction $\delta + \ell$ est de $8 + 1 = 9$ pour $[CF]$; on rajoute donc le sommet F. On continue ainsi jusqu'à atteindre le sommet I, ou bien jusqu'à ce que l'on ne puisse plus rajouter de nouveaux sommets.



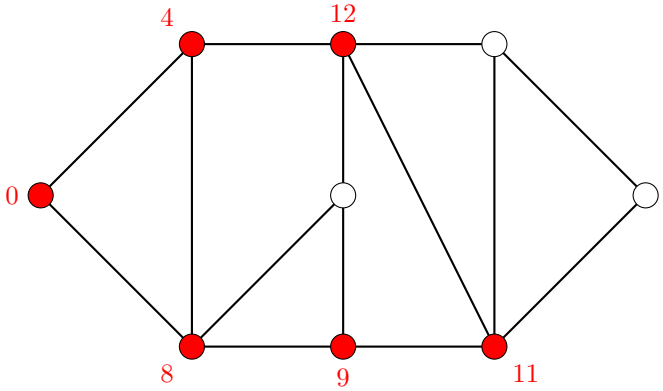


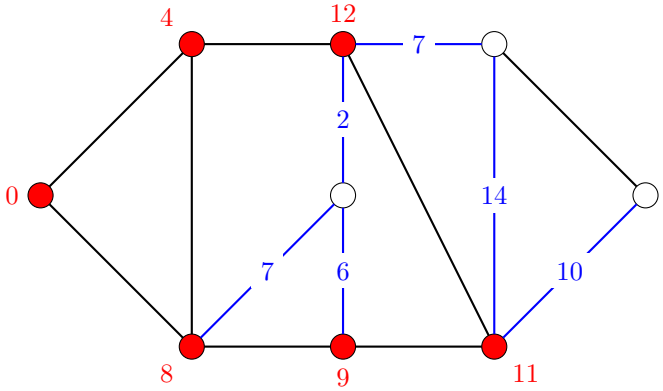
Comme tout exercice bien conçu, cela peut prendre du temps.



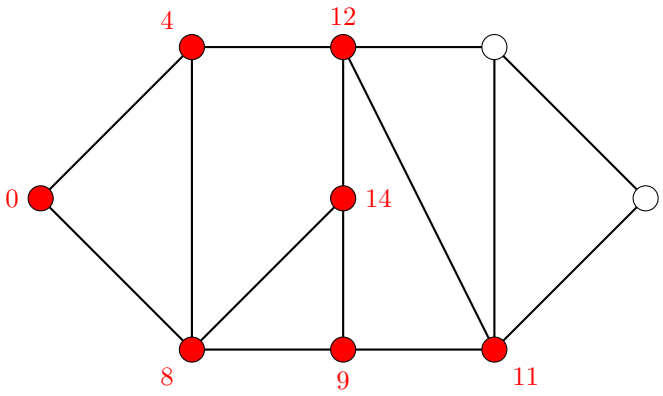


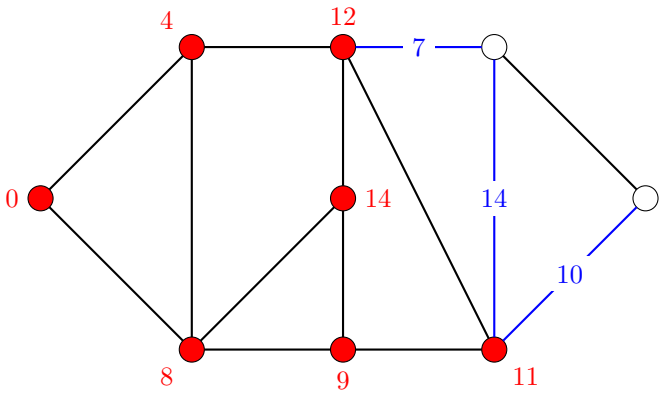
Il ne faudrait pas que les élèves s'en tirent après seulement une ou deux étapes !



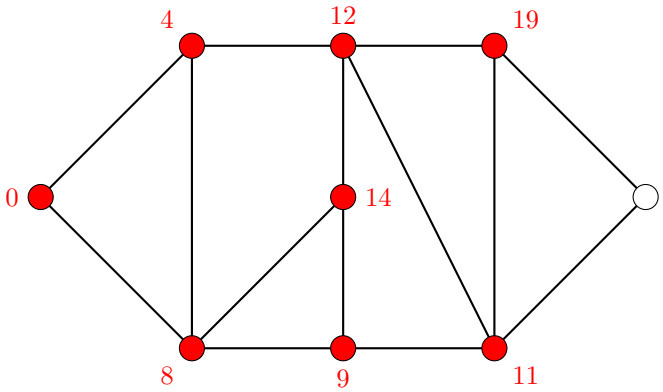


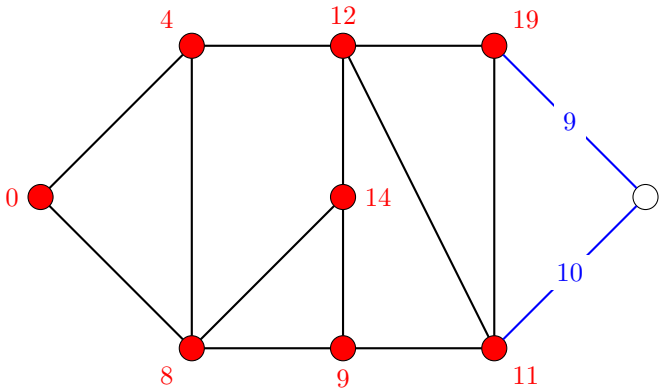
Il ne reste que deux sommets ! Nous y sommes presque.



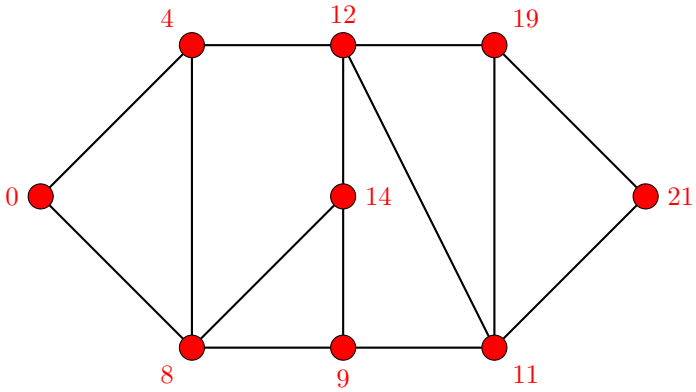


Et, bien entendu, le sommet d'arrivée est le dernier visité !





Voici donc l'étape finale. Remarquons que les deux conditions de sortie de l'algorithme sont satisfaites : on a atteint le sommet I, et on ne peut pas trouver d'arête qui relie un sommet de L_9 à un sommet de $V \setminus L_9$, ce dernier ensemble étant vide.



Comme tout le graphe a été visité, nous connaissons maintenant non seulement la distance de A à I, qui est de 21, mais aussi la distance de A à chacun des sommets du graphe ; c'est le nombre écrit en rouge ci-dessus, à côté de chaque sommet.

1.3 Recherche du plus court chemin

Cet algorithme ne donne que la distance entre deux sommets E et S. Cependant, on peut facilement l'améliorer pour obtenir le plus court chemin entre ces sommets. Pour cela, revenons au Lemme 1.1. À l'étape n de l'algorithme, on cherche une arête $[BC]$ qui minimise la quantité

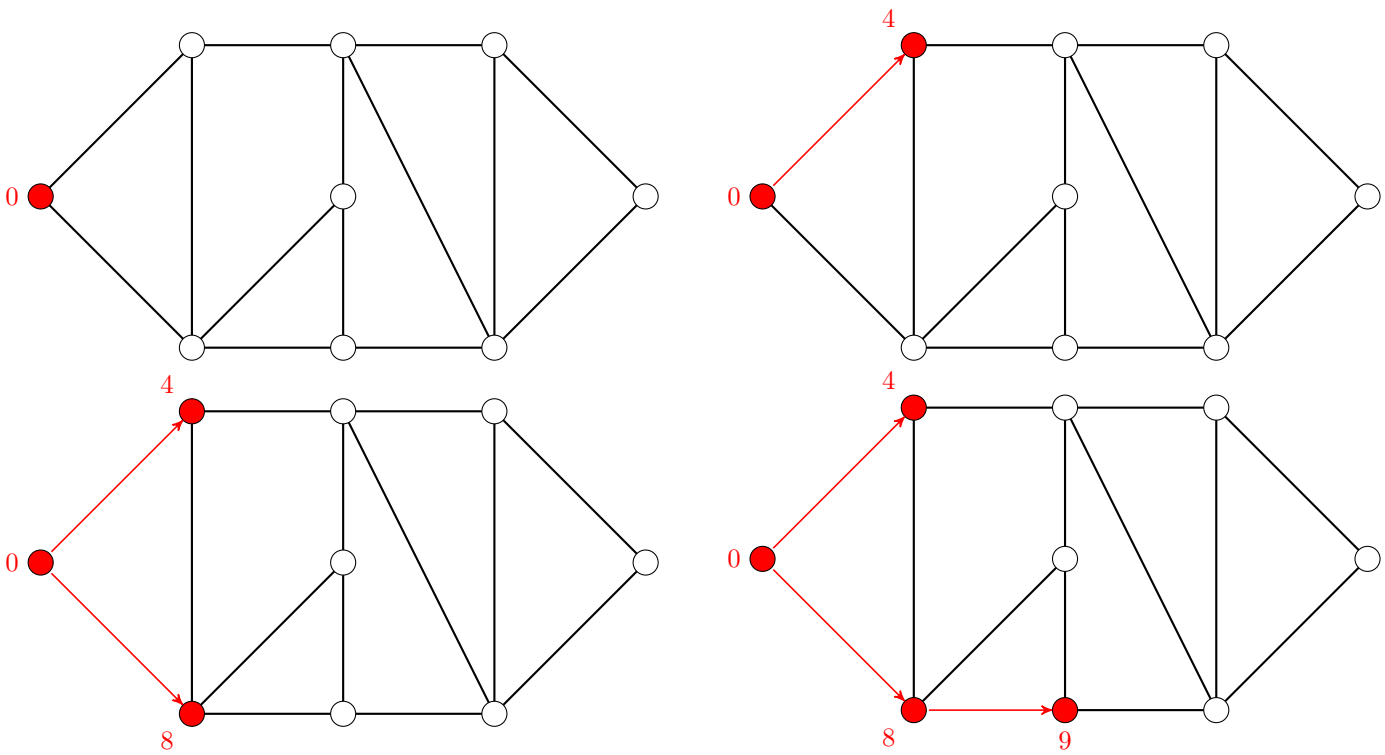
$$\min_{\substack{[B'C']\in A \\ B'\in V\setminus L_n,\ C'\in L_n}} \{\delta(C') + \ell([B'C'])\}, \tag{1.1}$$

et on ajoute à l'ensemble L_n des sommets visités l'extrémité $B \in V \setminus L_n$ de cette arête.

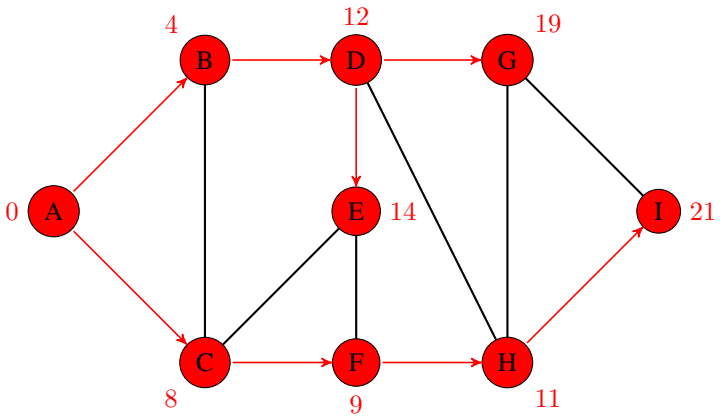
D'après la démonstration de ce lemme, un chemin qui réalise ce minimum est obtenu en empruntant un chemin le plus court de E à C, puis l'arête $[CB]$. Ceci permet de construire récursivement un chemin le plus court de E à chaque sommet visité :

- le chemin le plus court de E à lui-même est le chemin vide ;
- à l'étape n, si $[BC]$ réalise le minimum dans la quantité (1.1), alors un chemin le plus court de E à B est la concaténation d'un chemin le plus court de E à C, et de $[CB]$.

Il suffit donc, à chaque étape, de retenir un sommet par lequel un chemin le plus court arrive. Dans la représentation graphique utilisée dans l'exemple précédent, cela peut se faire en dessinant une arête orientée de C à B. Voici les premières et la dernière étape de l'algorithme de Dijktstra simplifié, avec cette donnée supplémentaire.



Passons maintenant au résultat final :



Pour trouver le chemin le plus court de A à I, on part de I, et on parcourt les arêtes en sens inverse jusqu'à arriver en A. Le chemin le plus court de A à I est donc le chemin (A, C, F, H, I).

1.4 Généralisations

Lors de la présentation de cet algorithme, nous avons fait deux restrictions sur le graphe : celui-ci doit être simple (sans arête multiple ni boucle) et non orienté. En fait, l'algorithme de Dijkstra fonctionne parfaitement sans ces restrictions.

Si le graphe n'est pas simple, les arêtes ne sont plus caractérisées par leurs extrémités. Le principal changement est une modification de la syntaxe dans la présentation ci-dessus. Par exemple, la quantité

$$\min_{\substack{[C'B']\in A \\ B'\in V\setminus L_n,\ C'\in L_n}} \{\delta(C') + \ell([C'B'])\}$$

doit être comprise comme un minimum sur les arêtes dont une extrémité est dans L_n et l'autre dans $V \setminus L_n$. De même, les chemins ne sont plus des suites de sommets ; il faut aussi retenir la suite des arêtes correspondantes. Ceci rend la présentation un peu moins élémentaire. De plus, on peut se ramener facilement au cas d'un graphe simple en effaçant toutes les boucles et, entre deux sommets, en ne gardant que l'arête de longueur minimale, les autres n'étant jamais empruntées par un chemin de longueur minimale.

L'algorithme de Dijkstra fonctionne aussi bien avec des graphes orientés, en recherchant un chemin de longueur minimale de E vers S. La seule modification est que, quand l'on considère la quantité

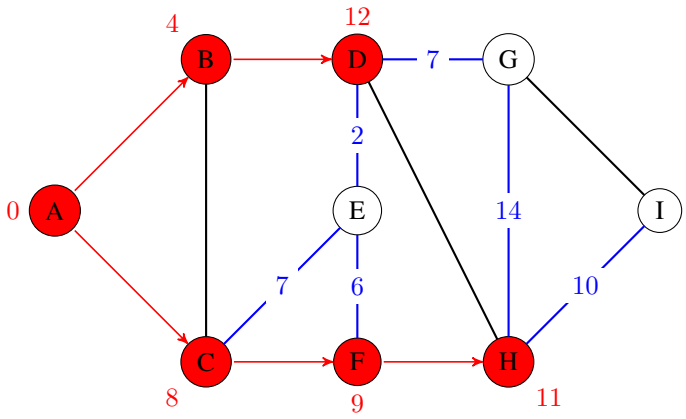
$$\min_{\substack{[C'B']\in A \\ B'\in V\setminus L_n,\ C'\in L_n}} \{\delta(C') + \ell([C'B'])\}$$

on ne prend le minimum que sur les arêtes qui partent de $C' \in L_n$ et qui arrivent en $B' \in V \setminus L_n$.

2 Une optimisation et l'algorithme de Dijkstra

L'algorithme de Dijkstra classique se retrouve en améliorant l'algorithme décrit dans la Partie 1. On peut remarquer que, dans l'algorithme présenté, certains calculs sont redondants. Dans l'exemple en Sous-partie 1.2, la quantité $\delta(B)+\ell([BD])$ est ainsi calculée 4 fois, entre le moment ou le sommet B est visité (étape 2) et celui où le sommet D est visité (étape 6). Une première idée pour optimiser cet algorithme est donc de garder en mémoire la quantité $\delta + \ell$ pour chaque arête, tant qu'elle relie un sommet visité et un sommet non visité.

Mais on peut faire mieux ! Considérons par exemple la situation suivante :



Plusieurs arêtes bleues aboutissent au sommet G, correspondant aux chemins (A, B, D, G), de longueur 19, et (A, C, F, H, G), de longueur 25. Il se sert à rien de retenir la longueur du chemin (A, C, F, H, G), car celui-ci est plus long que le chemin (A, B, D, G), et donc ne peut pas être un chemin le plus court de A à G.

Il suffit donc, pour chaque sommet $B' \in V \setminus L_n$, de ne retenir que l'arête $[C'B']$ qui minimise la quantité

$$u(B') := \min_{\substack{[C'B']\in A \\ C'\in L_n}} \{\delta(C') + \ell([C'B'])\},$$

ainsi que la valeur de $u(B')$. Ceci incite à introduire une nouvelle quantité, le *coût temporaire* u, qui est une fonction définie sur l'ensemble $V \setminus L_n$ des sommets non visités.

2.1 Formalisation de l'algorithme

Pour simplifier, comme dans la Sous-partie 1.1, on ignore la recherche d'un chemin le plus court, et on se concentre sur le calcul de la distance. L'algorithme devient alors le suivant.

DONNÉES DU PROBLÈME

Les données du problème sont : un graphe pondéré, un sommet d'entrée E, et un sommet de sortie S.

Pour tout $n \geq 1$, l'ensemble des données à calculer est composé :

- d'un sous-ensemble de sommets $L_n = \{v_1, \dots, v_n\} \subset V$;
- pour chacun de ces sommets v_k, d'un réel $\delta(v_k) \geq 0$;
- pour chacun des sommets $v' \in V \setminus L_n$, d'un nombre $u(v') > 0$, éventuellement infini,

tels que L_n est un ensemble de n sommets les plus proches de E, et $\delta(v_k) = d(E, v_k)$. La fonction u correspond au coût temporaire.

INITIALISATION

Pour $n = 1$, on pose :

- $L_1 = \{E\}$;
- $\delta(E) = 0$;
- $u(B) = +\infty$ pour tout $B \in V \setminus L_n$.

CALCUL RÉCURSIF

Par définition du coût temporaire, à l'étape n, on met à jour la valeur de u, puis on ajoute à L_n un sommet B dont le coût temporaire est minimal, et on pose $\delta(B) := u(B)$. Le problème est maintenant de calculer récursivement ce coût temporaire.

Soit $B' \in V \setminus L_n$. Soit v_n le sommet rajouté à l'étape n. Par convention, posons $\ell([v_n B']) := +\infty$ si aucune arête ne relie v_n à B'. Alors :

$$\begin{aligned} u(B') &= \min_{\substack{[C'B'] \in A \\ C' \in L_n}} \{\delta(C') + \ell([C'B'])\} \\ &= \min \left\{ \min_{\substack{[C'B'] \in A \\ C' \in L_{n-1}}} \{\delta(C') + \ell([C'B'])\}, \delta(v_n) + \ell([v_n B']) \right\} \\ &\leftarrow \min \{u(B'), \delta(v_n) + \ell([v_n B'])\} \end{aligned}$$

Autrement dit, il suffit de prendre tous les sommets $B' \in V \setminus L_n$ voisins de v_n, et pour ceux-ci de remplacer $u(B')$ par la quantité $\min \{u(B'), \delta(v_n) + \ell([v_n B'])\}$.

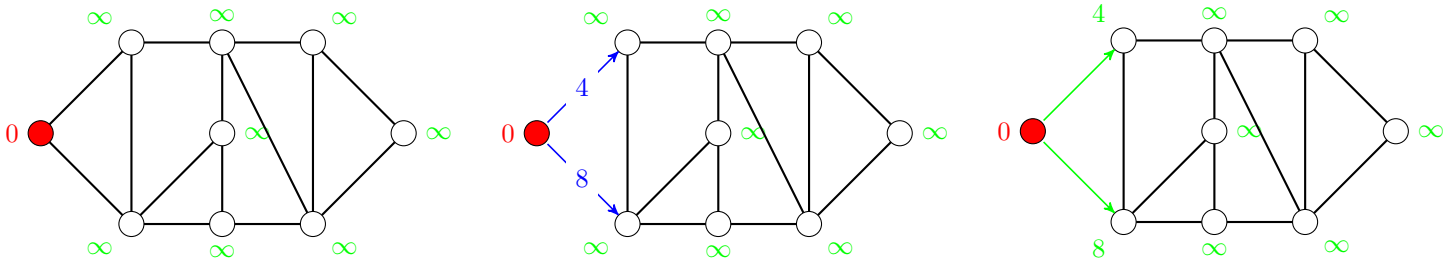
Les conditions d'arrêt sont les mêmes que pour l'algorithme non optimisé.

Si l'on veut exhiber un chemin le plus court, il suffit de garder en mémoire l'arête correspondant à la valeur de u. À l'étape n, cette arête est v_n si v_n réalise le minimum dans l'expression $\min \{u(B'), \delta(v_n) + \ell([v_n B'])\}$.

Comme précédemment, cet algorithme fonctionne aussi bien avec des graphes non simples ou non orientés.

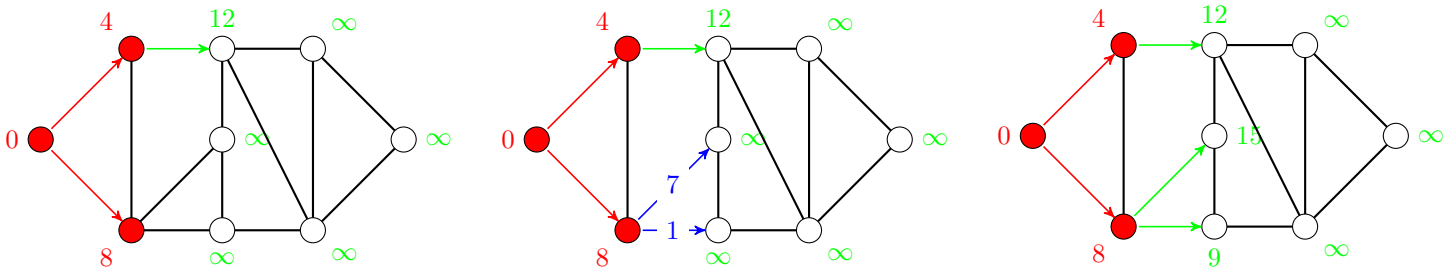
2.2 Exemple

Reprenons les premiées étapes de l'exemple de la Sous-partie 1.2. Les coûts temporaires sont indiqués en vert, de même que les arêtes correspondantes. On décompose chaque étape en deux sous-étapes : l'affichage des arêtes à prendre en compte, et la mise à jour de la fonction de coût temporaire.



Il faut donc ajouter à L_1 le sommet dont le coût temporaire est minimal, soit le sommet B.

Remarquons que, pour le sommet C, on a gardé le coût temporaire de 8, qui est bien le minimum entre le coût temporaire précédent, et la valeur $4 + 11 = 15$ correspondant à l'arête $[BC]$. Montrons encore l'étape suivante.



Le sommet que l'on visite alors est donc le sommet F. La suite de l'algorithme se déroule de la même façon.

On peut aussi écrire le déroulement de cet algorithme sous forme de tableau, avec une colonne par sommet et une ligne par étape. Chaque case correspond à la fonction $\delta = d(E, \cdot)$ pour les sommets visités, au coût temporaire u pour les sommets non visités, éventuellement en rajoutant la donnée de l'arête correspondante. C'est ce qui est fait dans les documents ressources ainsi que dans les manuels de terminale ES. Pour l'exemple utilisé, et en notant en rouge le sommet ajouté à l'étape courante, le tableau correspondant est :

n	A	B	C	D	E	F	G	H	I
1	0	∞	∞	∞	∞	∞	∞	∞	∞
2	–	$4, A$	$8, A$	∞	∞	∞	∞	∞	∞
3	–	–	$8, A$	$12, B$	∞	∞	∞	∞	∞
4	–	–	–	$12, B$	$15, C$	$9, C$	∞	∞	∞
5	–	–	–	$12, B$	$15, C$	–	∞	$11, F$	∞
6	–	–	–	$12, B$	$15, C$	–	$25, H$	–	$21, H$
7	–	–	–	–	$14, D$	–	$19, D$	–	$21, H$
8	–	–	–	–	–	–	$19, D$	–	$21, H$
9	–	–	–	–	–	–	–	–	$21, H$

2.3 Discussion

Nous avons présenté deux versions de l'algorithme de Dijkstra : une version simplifiée, et la version standard (ou améliorée). Nous allons maintenant comparer ces algorithmes sous un point de vue pédagogique.

D'un point de vue combinatoire, la version optimisée de l'algorithme est plus efficace que la version simplifiée, avec une diminution notable du temps de calcul pour un prix en mémoire modique. Ceci dit, les questions de complexité mises en jeu sont hors de portée du programme de terminale S. De plus, en l'absence d'implémentation effective, les élèves traitent les exemples à la main. Dans ce cadre, la taille des graphes utilisés reste modeste, et la différence d'efficacité n'est pas significative.

Notre opinion est que l'algorithme simplifié a plusieurs avantages pédagogiques par rapport à la version optimisée :

- il est plus simple d'en exposer le principe à une audience généraliste. En particulier, il ne fait intervenir que des objets d'interprétation simple (la distance au sommet de départ, et le chemin le plus court), là ou une interprétation du coût temporaire est moins intuitive[8] ;
- il est possible de travailler directement sur le graphe, en ne notant que les sommets visités, la valeur de δ, et l'arbre des chemins les plus courts. C'est particulièrement facile au tableau, en utilisant des craies ou feutres de couleurs, mais est aussi faisable sur papier. Par comparaison, l'algorithme optimisé demande non seulement d'afficher plus de données, mais aussi de modifier ou d'effacer certaines de ces données au fur et à mesure, d'où l'utilisation d'un tableau comme dans la Sous-partie précédente.

La possibilité de travailler directement sur le graphe avec des objets simples permet d'expliquer plus facilement le sens des opérations effectuées. À notre avis, au niveau de la terminale ES, le seul inconvénient d'un travail sur le graphe est que les différentes étapes de l'algorithme n'apparaissent pas dans la trace écrite finale, ce qui rend plus difficile de repérer les erreurs des élèves et d'évaluer leur travail.

[8] Une définition du coût temporaire d'un sommet $u(B)$ est, par exemple, la longueur d'un chemin le plus court de E à B dont tous les sommets sauf le dernier sont dans L_n.