Plus courts chemins

Problème typique de cheminement dans les graphes orientés ou non orientés :
recherche d'un plus court chemin entre deux sommets.

Nombreuses applications :

- Trouver le moyen le plus économique pour aller de Brest à Lyon, connaissant pour chaque ligne aérienne le prix de billet d'avion.
- Trouver l'itinéraire le plus économique pour aller de Brest à Lyon par la route (en termes de kilomètres)
- Trouver l'itinéraire le plus économique pour aller de Brest à Lyon par la route (en termes d'heures en route)
- Problèmes d'optimisation de réseaux (réseaux routiers ou réseaux de télécommunications)
- Problèmes d'ordonnancement
- Problèmes d'intelligence artificielle tels que la circulation dans un labyrinthe

Définition, conditions d'existence

Le graphe G = <S, A> s'appelle **valué**

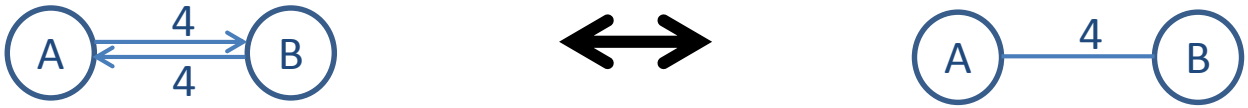
⇔ à chaque arc (ou arête) **u** est associé une longueur (un coût) l (u).

On appelle coût (poids) d'un chemin μ la somme des coûts des arcs (arêtes) qui le compose noté coût(μ) ou l(μ).

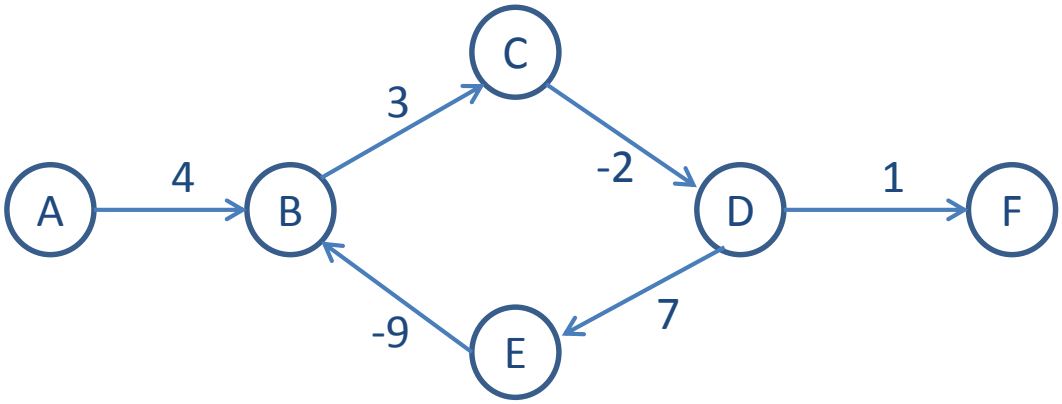
$$l(\mu) = \sum_{u \in \mu(x,y)} l(u)$$

On appelle plus court chemin de x vers y un chemin (une chaîne) allant de x à y de coût minimum.
Un tel chemin n'existe pas toujours.

Remarque. On peut si l'on veut considérer un graphe non orienté comme un graphe orienté symétrique, chaque arête du graphe non orienté correspondant à deux arcs du graphe orienté liant les sommets en question dans les deux sens et ayant le même poids. Un graphe non orienté devient alors un cas particulier d'un graphe orienté.
Cela nous permet d'utiliser uniquement les graphes orientés dans la plupart des discussions et des exemples.



Définition, conditions d'existence



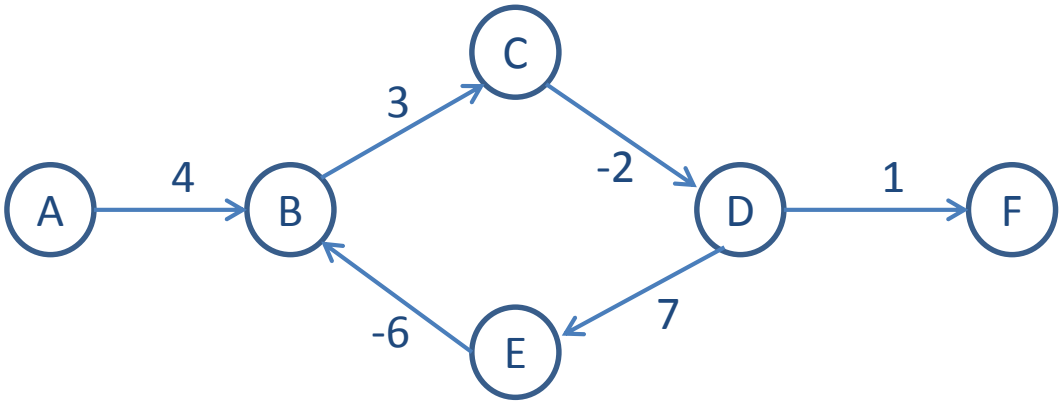
Considérons un chemin μ allant de x à y; supposons ce chemin contient un circuit Γ. Soit μ' le chemin obtenu en supprimant ce circuit de μ. On a :

$$coût(\mu) = coût(\mu') + coût(\Gamma)$$

Pour le graphe ci-dessus, le circuit présent (un seul) est Γ= BCDEB, et, par exemple, pour un chemin de A à F on a

$$coût(ABCDEBCDF) = coût(ABCDF) + coût(\Gamma)$$

Conditions d'existence

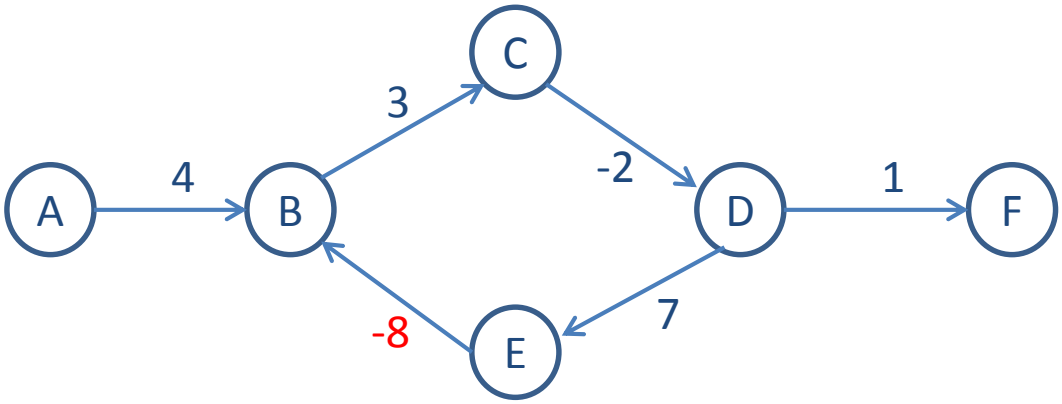


coût(μ) = coût(μ') + coût(Γ)

Si coût(Γ) est strictement positif, tout chemin de x à y contenant Γ est plus long que celui qui ne le contient pas. Donc un plus court chemin ne peut pas contenir un circuit de coût strictement positif.

Ici, coût(Γ) = 3 + (-2) + 7 + (-6) = 2. Le plus court chemin entre A et F est ABCDF, qui ne contient pas le circuit BCDEB.

Conditions d'existence



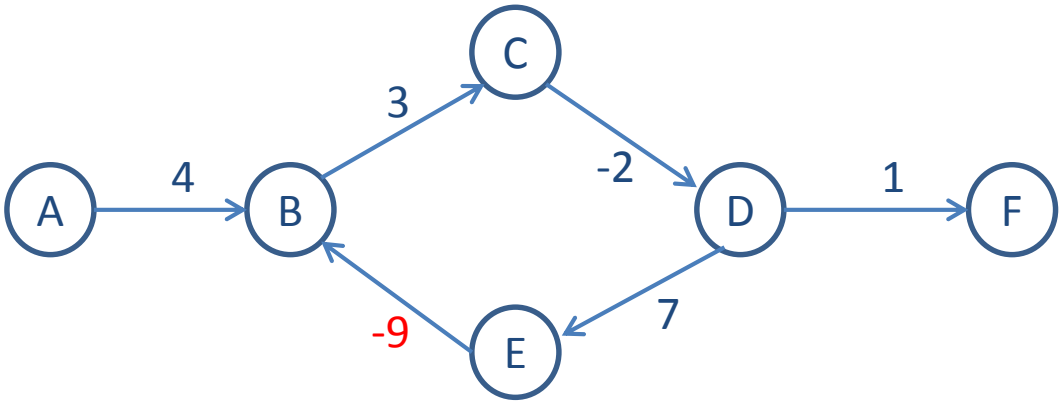
coût(μ) = coût(μ') + coût(Γ)

Si coût(Γ) est nulle, le fait d'ajouter Γ au chemin une ou plusieurs fois ne modifie pas son coût. Il existe donc plusieurs plus court chemins de x à y qu'on obtient en passant un nombre de fois par le circuit.

Ici, coût(Γ) = 3 + (-2) + 7 + (-8) = 0 : il y un nombre infini de plus court chemins entre A et F.

coût(ABCDF)= coût(ABCDEBCDF) = coût(ABCDEBCDEBCDF) =…

Conditions d'existence



coût(μ) = coût(μ') + coût(Γ)

Si coût(Γ) est négatif, le fait d'ajouter le circuit à un chemin résulte en un chemin plus court. Plus de fois on l'ajoute, plus le chemin devient court, il n'existe donc pas de plus court chemin de x à y.

Ici, coût(Γ) = 3 + (-2) + 7 + (-9) = -1 : il n'y a pas de plus court chemin entre A et F.

Les circuits de coût négatif sont appelés **circuits absorbants**.

Dans la suite on suppose qu'il n'y pas de circuits absorbants ; après, on développera des algorithmes de recherche de circuits.

Conditions d'existence

- *Dans un graphe fini,* le nombre de chemins **élémentaires** de x vers y (chemins qui ne contiennent pas plusieurs fois un même sommet) est fini.

  Donc, il existe au moins un plus court chemin élémentaire de x vers y.

- *S'il n'y a pas de circuit absorbant,*
  - un plus court chemin élémentaire est un plus court chemin de x vers y,
  - le problème de la recherche d'un plus court chemin de x vers y, admet une solution dès qu'il existe un chemin de x vers y.

Variantes de problème

Trois variantes du problème de la recherche d'un plus court chemin:

On recherche

1. un plus court chemin entre deux sommets donnés

2. les plus courts chemins entre un sommet donné, appelé source et tous les autres sommets

3. les plus courts chemins entre tous les sommets pris deux à deux.

On ne connaît pas de solution meilleure pour le problème 1 que celle qui consiste à passer par les solutions de 2.

Il existe différents algorithmes qui sont plus ou moins bien adaptés selon les propriétés du graphe et la forme du problème étudié (2 ou 3), mais il n'y a pas d'algorithme général qui soit efficace et intéressant dans tous les cas.

Algorithme de Dijkstra

Edsger Wybe Dijkstra (né à Rotterdam le 11 mai 1930, mort à Nuenen le 6 août 2002) est un mathématicien et informaticien néerlandais du XX^{e} siècle.

Après des études de physique théorique, il s'engage dès 1955 dans le domaine de l'informatique alors naissante, dont il est l'un des pionniers les plus éclairés. Parmi ses contributions se trouve un algorithme de calcul du plus court chemin dans les graphes, connu sous le nom **d'algorithme de Dijkstra**.

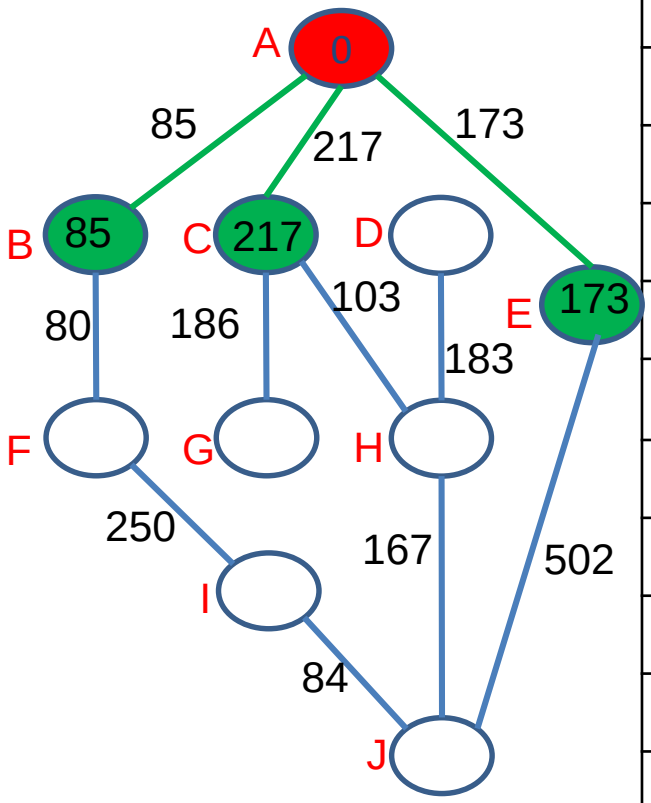
Algorithme de Dijkstra

Cet algorithme n'est utilisable que dans le cas, très fréquent, où les coûts des arcs sont tous positifs ou nuls.

Il calcule un plus court chemin entre **source X et tous les sommets accessibles depuis X** :
on obtient alors une arborescence de racine X formée par ces plus courts chemins.

Avant de formuler l'algorithme de façon formelle, nous commencerons par un exemple de son fonctionnement.

Algorithme de Dijkstra, exemple 1

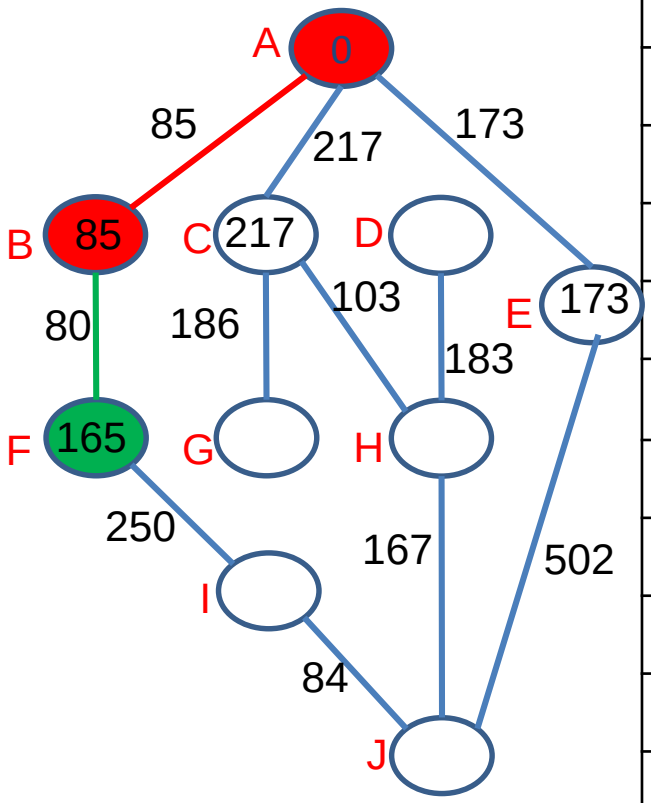


CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
	•									
	•									
	•									
	•									
	•									
	•									
	•									
	•									
	•									

On met les points rouges dans les cases correspondant aux villes qui font partie de CC.

Étape 1 (initialisation). La ville A (la source) est, évidemment, à une distance de 0 km d'elle-même. On la place donc dans l'ensemble CC (les villes vers lesquelles on connaît la distance minimum), et le reste des villes constituent l'ensemble appelé M. A partir de la ville A, les villes B, C et E sont accessibles à des distances respectives de de 85, 217, 173 km (qui ne sont pas considérées comme des distances finales). Les autres villes sont affectées d'une distance infinie.

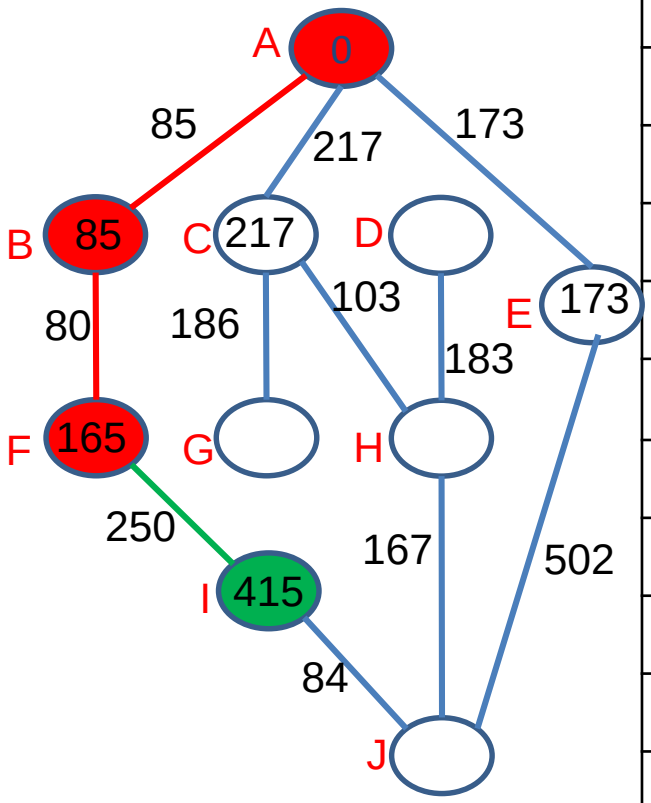
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
	•	•								
	•	•								
	•	•								
	•	•								
	•	•								
	•	•								
	•	•								
	•	•								

Étape 2 : la distance la plus courte est celle menant à la ville B (85). B passe vers CC. Le passage par la ville B ouvre la voie à la ville F (85+80 = 165).

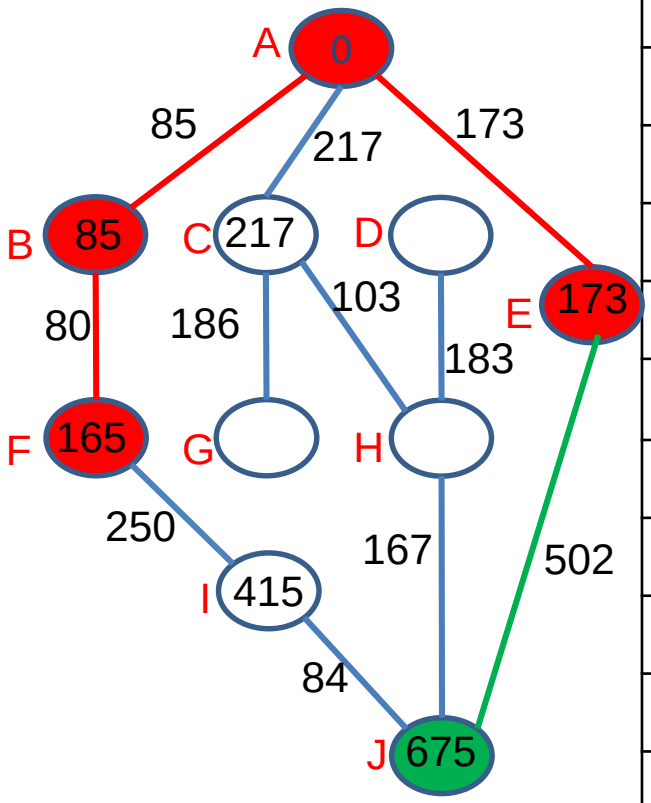
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
ABF	•	•	217_A	∞	173_A	•	∞	∞	415_F	∞
	•	•				•				
	•	•				•				
	•	•				•				
	•	•				•				
	•	•				•				
	•	•				•				
	•	•				•				

Étape 3 : la distance la plus courte dans M est celle menant à la ville F (165). F passe vers CC. Le passage par la ville F ouvre la voie à la ville I (415).

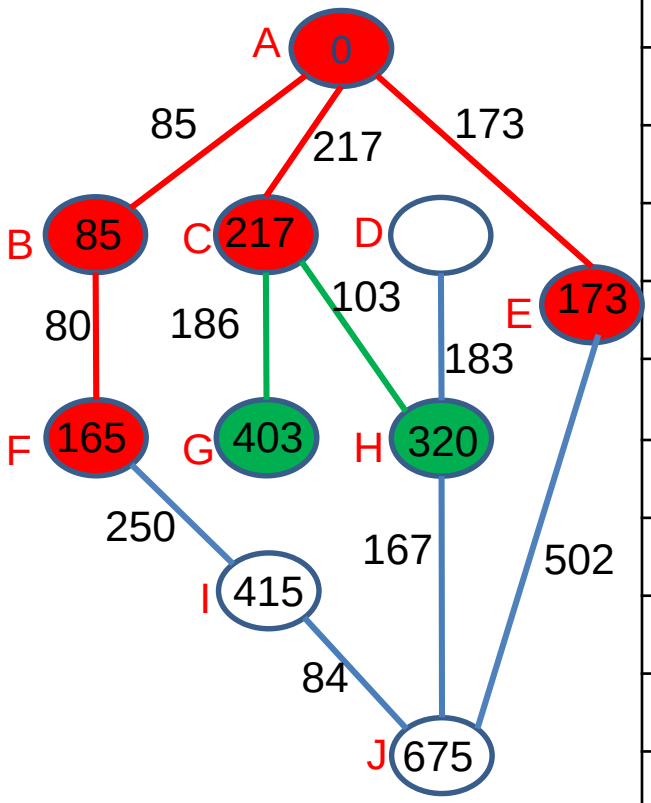
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
ABF	•	•	217_A	∞	173_A	•	∞	∞	415_F	∞
ABEF	•	•	217_A	∞	•	•	∞	∞	415_F	675_E
	•	•			•	•				
	•	•			•	•				
	•	•			•	•				
	•	•			•	•				
	•	•			•	•				
	•	•			•	•				

Étape 4 : La distance la plus courte suivante dans M est alors celle menant à la ville E (173). E passe vers CC. Le passage par la ville E ouvre une voie vers la ville J (675).

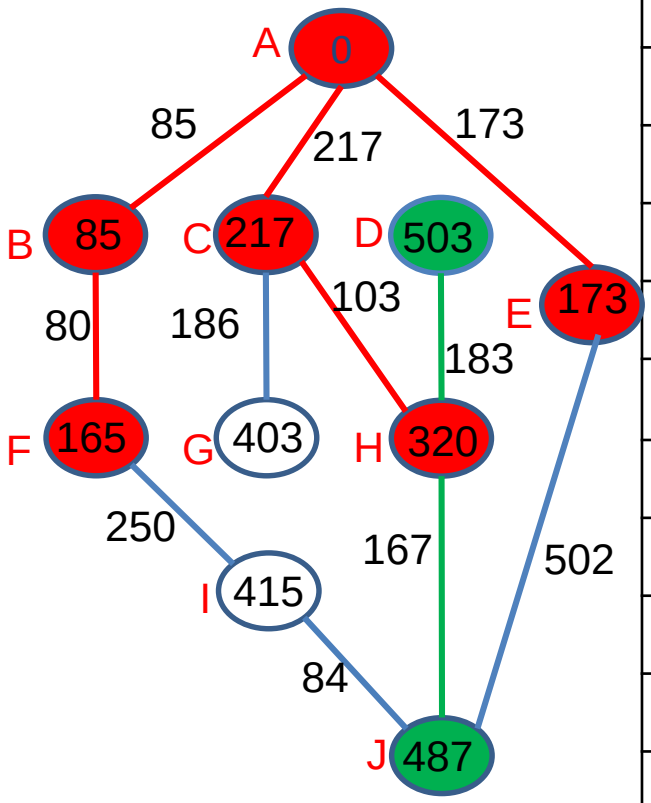
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
ABF	•	•	217_A	∞	173_A	•	∞	∞	415_F	∞
ABEF	•	•	217_A	∞	•	•	∞	∞	415_F	675_E
ABCEF	•	•	•	∞	•	•	403_C	320_C	415_F	675_E
	•	•	•		•	•				
	•	•	•		•	•				
	•	•	•		•	•				
	•	•	•		•	•				
	•	•	•		•	•				

Étape 5 : la distance la plus courte suivante dans M mène alors à la ville C (217). C passe vers CC. Le passage par la ville C ouvre une voie vers la ville G (403) et la ville H (320).

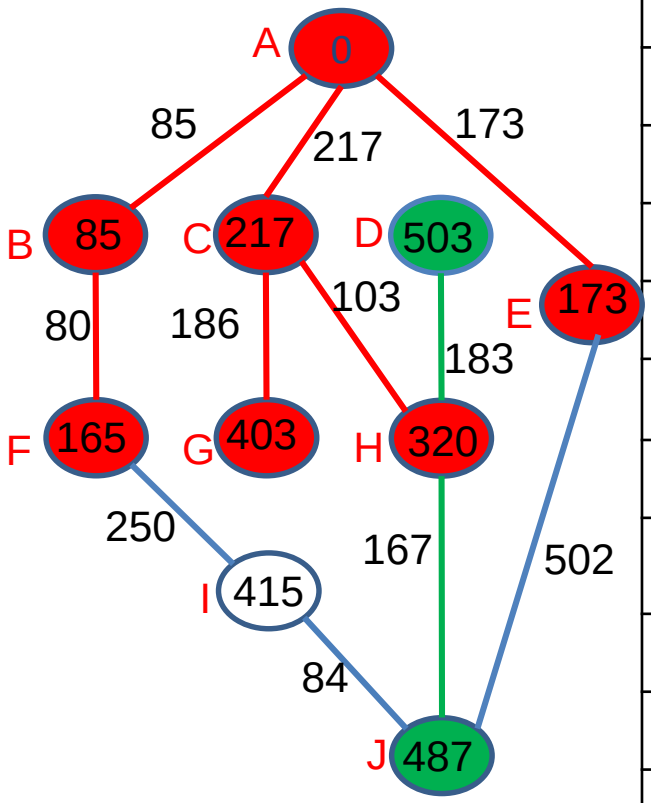
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
ABF	•	•	217_A	∞	173_A	•	∞	∞	415_F	∞
ABEF	•	•	217_A	∞	•	•	∞	∞	415_F	675_E
ABCEF	•	•	•	∞	•	•	403_C	320_C	415_F	675_E
ABCEFH	•	•	•	503_H	•	•	403_C	•	415_F	487_H
	•	•	•		•	•		•		
	•	•	•		•	•		•		
	•	•	•		•	•		•		
	•	•	•		•	•		•		

Étape 6: la distance la plus courte suivante dans M mène à ville H(320). H passe vers CC. Le passage par la ville H ouvre une voie vers la ville D (503) et un raccourci vers la ville J (487< 675).

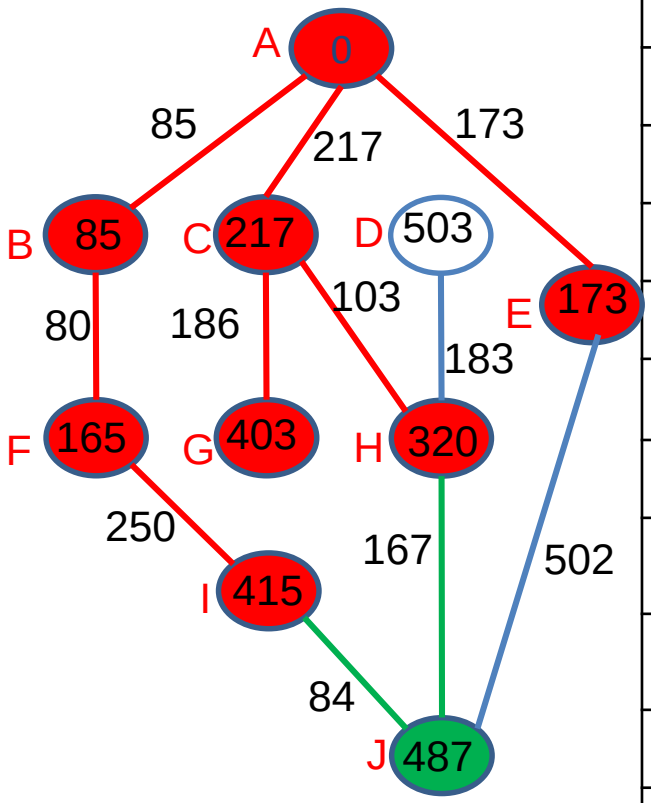
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
ABF	•	•	217_A	∞	173_A	•	∞	∞	415_F	∞
ABEF	•	•	217_A	∞	•	•	∞	∞	415_F	675_E
ABCEF	•	•	•	∞	•	•	403_C	320_C	415_F	675_E
ABCEFH	•	•	•	503_H	•	•	403_C	•	415_F	487_H
ABCEFGH	•	•	•	503_H	•	•	•	•	415_F	487_H
	•	•	•		•	•	•	•		
	•	•	•		•	•	•	•		
	•	•	•		•	•	•	•		

Étape 7 : la distance la plus courte suivante dans M mène à la ville G (403), qui passe donc vers CC, et ne change aucune autre distance.

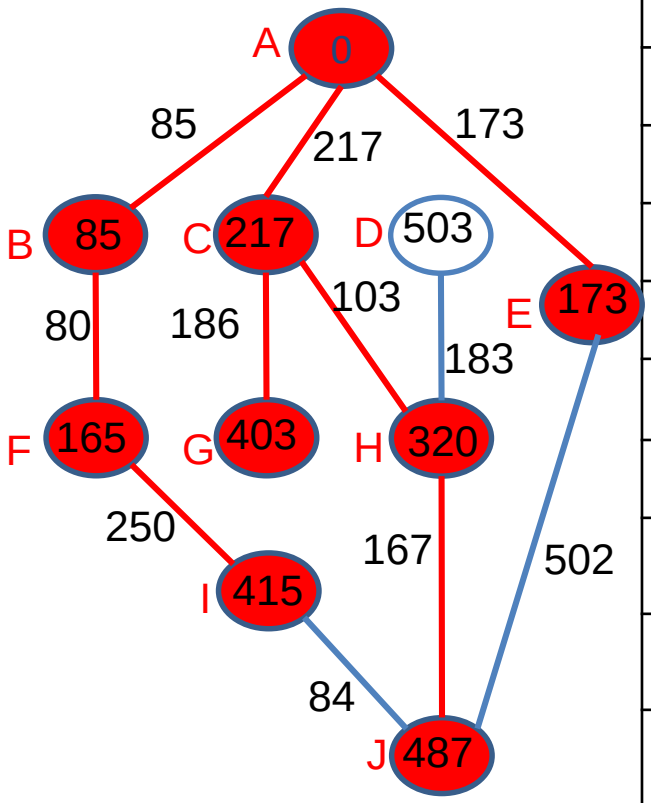
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
ABF	•	•	217_A	∞	173_A	•	∞	∞	415_F	∞
ABEF	•	•	217_A	∞	•	•	∞	∞	415_F	675_E
ABCEF	•	•	•	∞	•	•	403_C	320_C	415_F	675_E
ABCEFH	•	•	•	503_H	•	•	403_C	•	415_F	487_H
ABCEFGH	•	•	•	503_H	•	•	•	•	415_F	487_H
ABCEFGHI	•	•	•	503_H	•	•	•	•	•	487_H
	•	•	•		•	•	•	•	•	
	•	•	•		•	•	•	•	•	

Étape 8 : la distance la plus courte suivante dans M mène à la ville I (415), qui passe vers CC. Cela pourrait changer la distance vers J, mais comme $499_I > 487_H$, la distance de J reste la même.

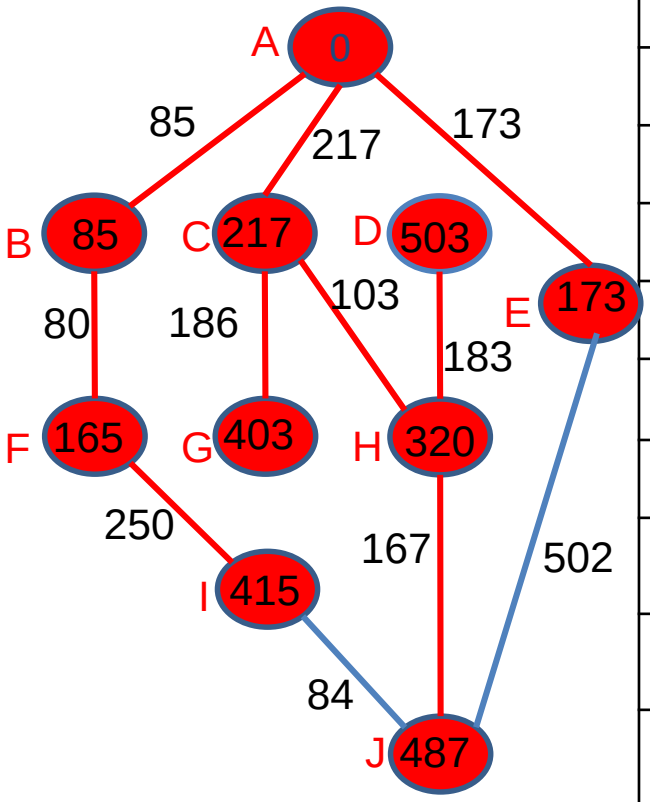
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
ABF	•	•	217_A	∞	173_A	•	∞	∞	415_F	∞
ABEF	•	•	217_A	∞	•	•	∞	∞	415_F	675_E
ABCEF	•	•	•	∞	•	•	403_C	320_C	415_F	675_E
ABCEFH	•	•	•	503_H	•	•	403_C	•	415_F	487_H
ABCEFGH	•	•	•	503_H	•	•	•	•	415_F	487_H
ABCEFGHI	•	•	•	503_H	•	•	•	•	•	487_H
ABCEFGHIJ	•	•	•		•	•	•	•	•	•
ABCDEFGHIJ	•	•	•		•	•	•	•	•	•

Étape 9 : la distance la plus courte suivante dans M mène à la ville J (487), pour laquelle il ne reste pas de voisins dans M, donc aucune autre distance ne change. J passe vers CC.

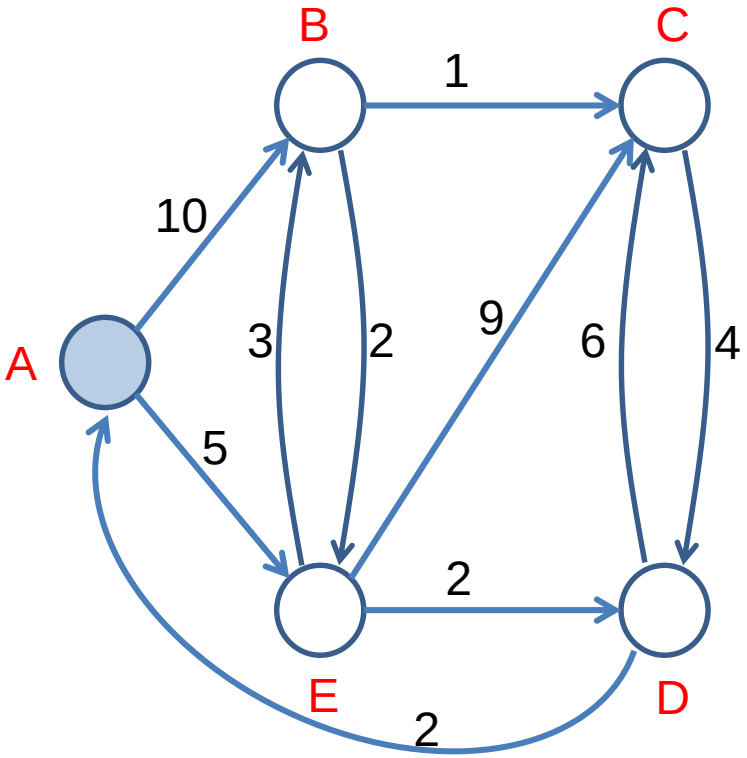
Algorithme de Dijkstra, exemple 1



CC	A	B	C	D	E	F	G	H	I	J
A	0	85_A	217_A	∞	173_A	∞	∞	∞	∞	∞
AB	•	•	217_A	∞	173_A	165_B	∞	∞	∞	∞
ABF	•	•	217_A	∞	173_A	•	∞	∞	415_F	∞
ABEF	•	•	217_A	∞	•	•	∞	∞	415_F	675_E
ABCEF	•	•	•	∞	•	•	403_C	320_C	415_F	675_E
ABCEFH	•	•	•	503_H	•	•	403_C	•	415_F	487_H
ABCEFGH	•	•	•	503_H	•	•	•	•	415_F	487_H
ABCEFGHI	•	•	•	503_H	•	•	•	•	•	487_H
ABCEFGHIJ	•	•	•	503_H	•	•	•	•	•	•
ABCDEFGHIJ	•	•	•	•	•	•	•	•	•	•

Étape 10 : le seul sommet restant dans M est la ville D (503), qui passe vers l'ensemble CC; il ne reste rien dans M, FIN.
Les distances les plus courtes calculées sont marquées dans la table par le rouge.
Les arêtes faisant partie des plus courtes chaînes (en rouge sur le graphe) forment **un arbre**.

Algorithme de Dijkstra, exemple 2



•Notation : x : étiquette du sommet;
$\pi(x)$: distance courante (qui dépend de l'itération) entre les sommets A et x.

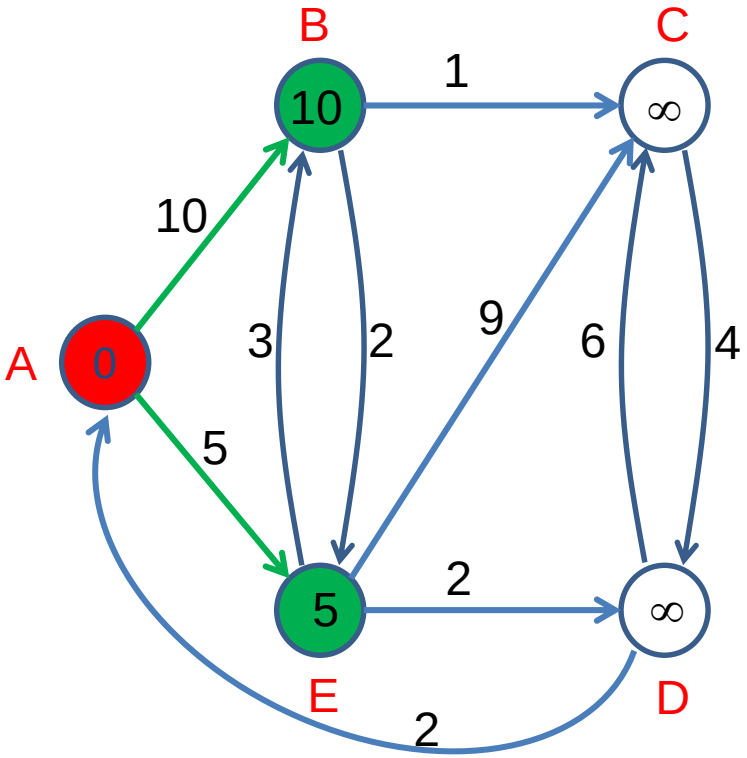
•On initialise tous les $\pi(x)= l_{Ax}$. l_{xy} = la longueur de l'arc (x,y) si (x,y) existe, sinon $l_{xy} = \infty$. $l_{xx} = 0$.

•On note $\pi^*(x)$ la longueur minimum des chemins de A à x ; donc $\pi^*(A) = 0$.

•L'ensemble des sommets S est partitionné en deux : M et CC, M = S \ CC. Au début CC={A} et M = {B,C,D,E}. On construit progressivement un ensemble CC des sommets x pour lesquels on connaît le plus court chemin de A vers x.

Algorithme de Dijkstra, exemple 2

Au début, CC = {A}, M = {B,C,D,E}.
Les successeurs de A dans M sont B et E (marqués par le vert). On marque les arcs AB et AE par le vert, et on met à jours les poids de B et de E.

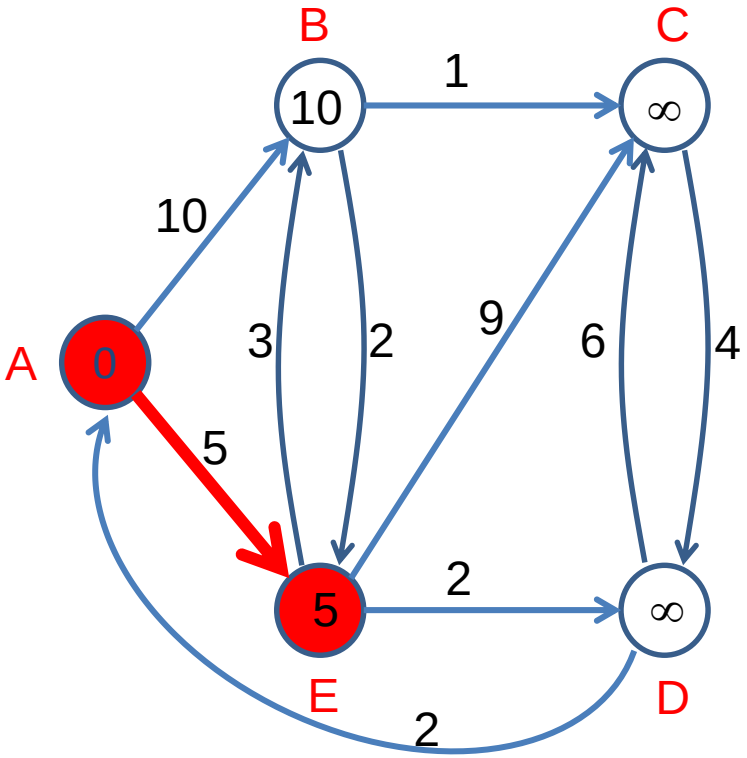


CC	A	B	C	D	E
A	0	10_A	∞	∞	5_A
	•				
	•				
	•				

Dans les colonnes, on mets la distance de x à A, et le sommet par lequel on accede à x.

Algorithme de Dijkstra, exemple 2

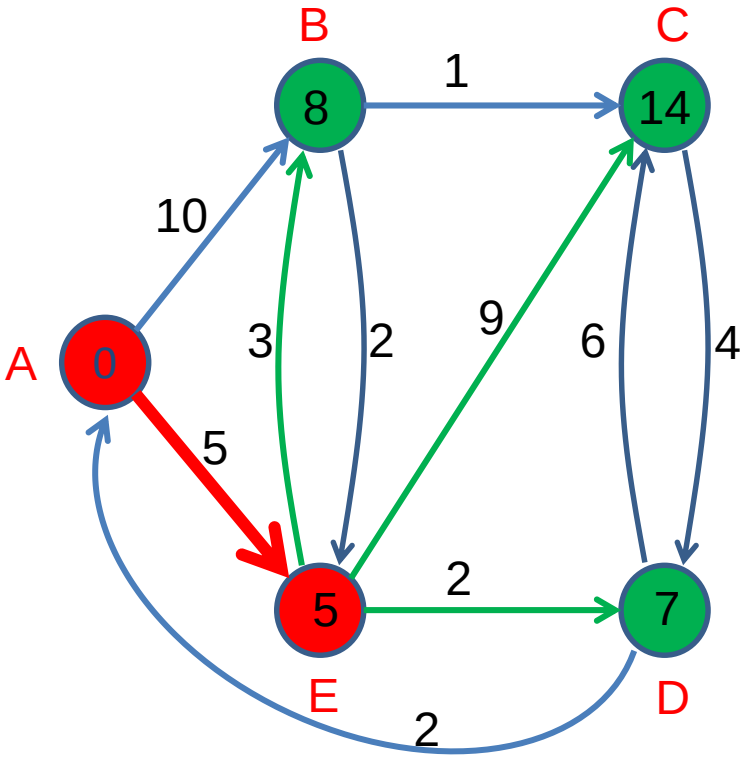
Dans M, le sommet de plus petit poids est E. On le met dans CC.
$\pi^*(E)=5$. CC={A,E}, M={B,C,D}. On marque l'arc AE par le rouge et on enlève la couleur de l'arc AB.



CC	A	B	C	D	E
A	•	10_A	∞	∞	5_A
AE	•				•
	•				•
	•				•

Algorithme de Dijkstra, exemple 2

Les successeurs de E dans M sont B, C et D. On met à jour leurs poids si le fait de passer par E diminue le poids courant.

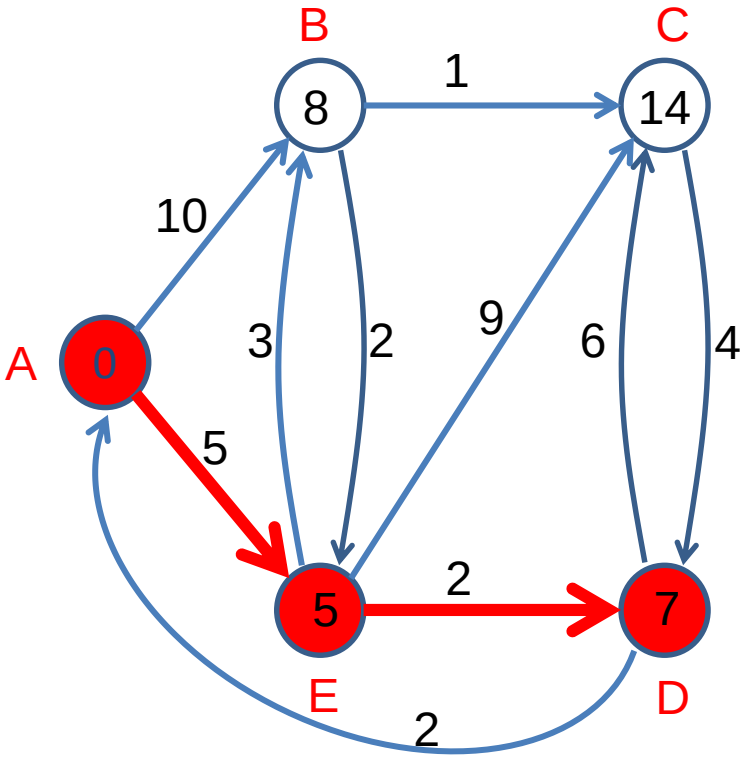


CC	A	B	C	D	E
A	•	10_A	∞	∞	5_A
AE	•	8_E	14_E	7_E	•
	•				•
	•				•

Remarque : ici on peut avoir la tentation de mettre la distance 11_B pour C au lieu de 14_E. Mais selon l'algorithme, en mettant à jour de nouveaux sommets, on ne considère que des chemins qui passent par le sommet qui vient d'être ajouté à CC (ici, E) , et contiennent un arc de plus à la fin. Le fait de mettre à jour C en utilisant B à ce point ne change pas le résultat mais nécessite plus d'opérations en total qu'il est nécessaire (ici, une comparaison de plus)..

Algorithme de Dijkstra, exemple 2

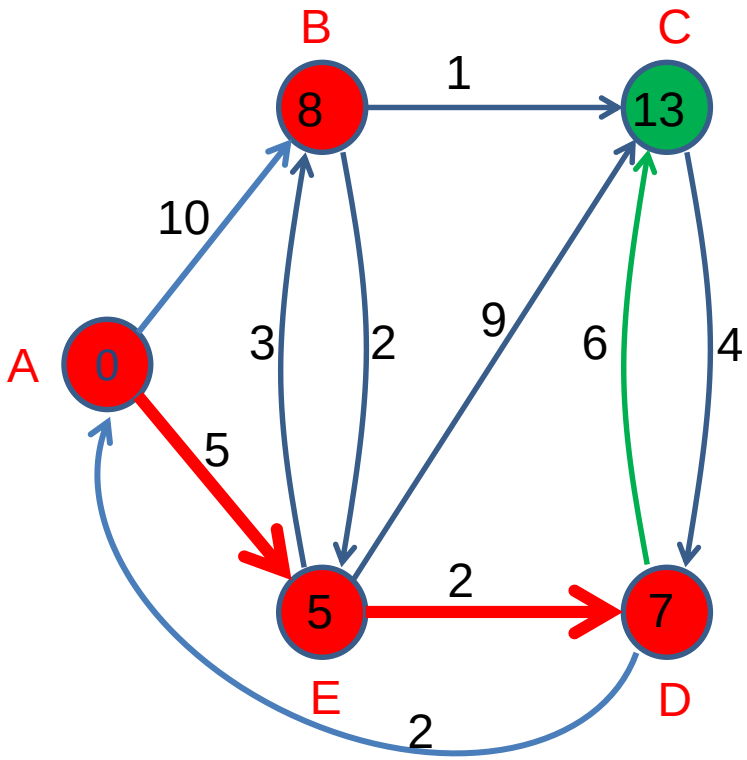
Dans M, le sommet de plus petit poids est D. On le met dans CC.
$\pi^*(D)=7$. CC={A,D,E}, M={B,C}. On marque l'arc ED par le rouge et on enlève la couleur des autres arcs et sommets dans M.



CC	A	B	C	D	E
A	•	10_A	∞	∞	5_A
AE	•	8_E	14_E	7_E	•
ADE	•			•	•
	•			•	•

Algorithme de Dijkstra, exemple 2

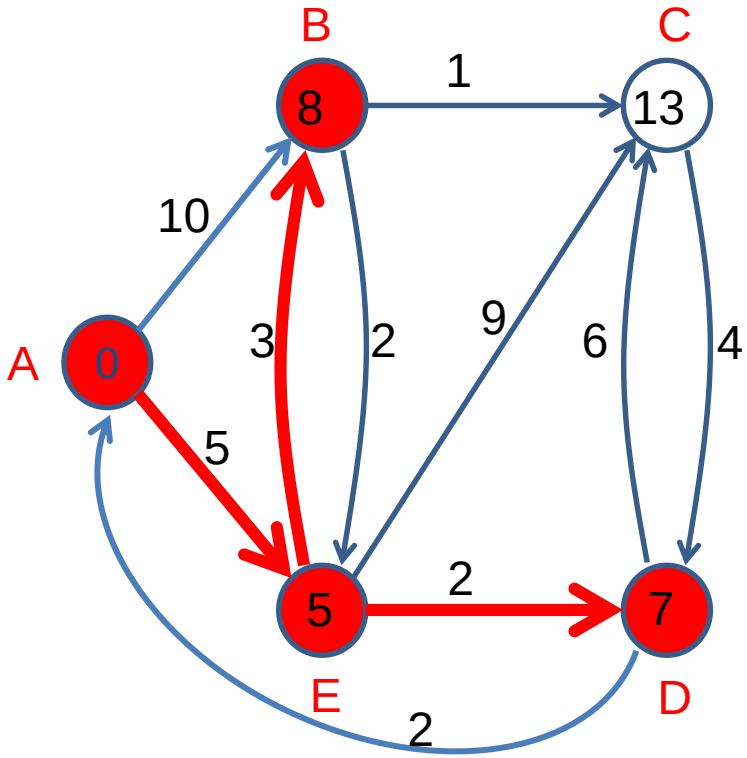
Le seul successeur de D dans M est C. On met à jour son poids car le fait de passer par D diminue le poids courant.



CC	A	B	C	D	E
A	•	10_A	∞	∞	5_A
AE	•	8_E	14_E	7_E	•
ADE	•	8_E	13_D	•	•
	•			•	•

Algorithme de Dijkstra, exemple 2

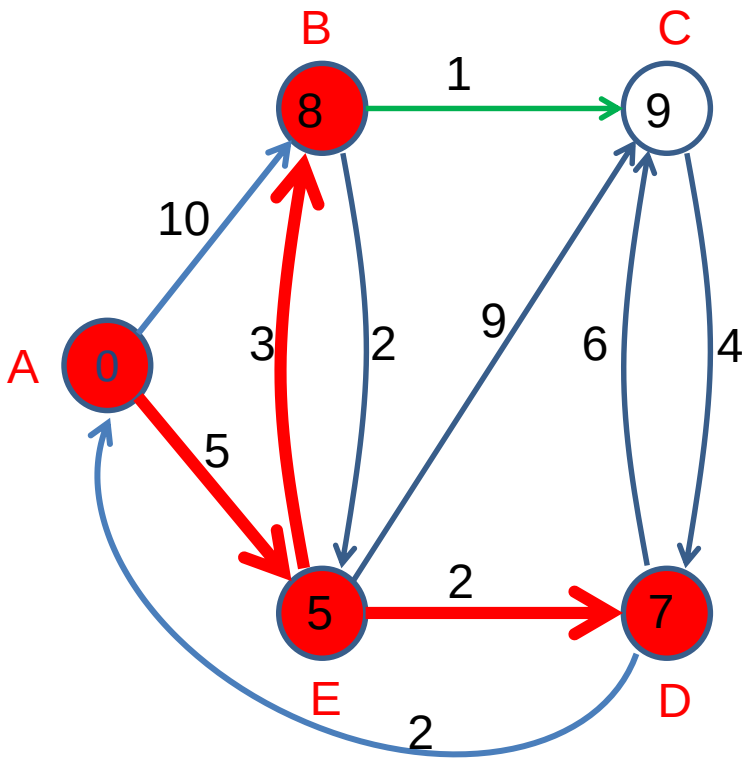
Dans M, le sommet de plus petit poids est B. On le met dans CC.
$\pi^*(B)=8$. CC={A,B,D,E}, M={C}. On marque l'arc EB par le rouge et on enlève la couleur des autres arcs et sommets dans M.



CC	A	B	C	D	E
A	•	10_A	∞	∞	5_A
AE	•	8_E	14_E	7_E	•
ADE	•	8_E	13_D	•	•
ABDE	•	•		•	•

Algorithme de Dijkstra, exemple 2

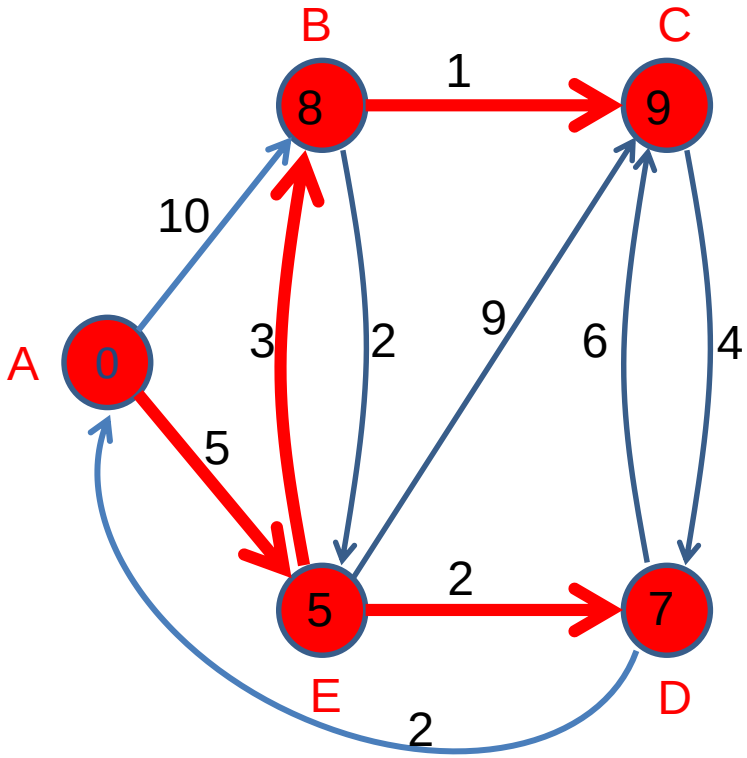
Le seul successeur de B dans M est C. On met à jour son poids car le fait de passer par B diminue le poids courant.



CC	A	B	C	D	E
A	•	10_A	∞	∞	5_A
AE	•	8_E	14_E	7_E	•
ADE	•	8_E	13_D	•	•
ABDE	•	•	9_B	•	•

Algorithme de Dijkstra, exemple 2

L'ensemble M consiste d'un seul sommet C, son poids est par définition minimal. Donc C passe dans CC. Il n'y reste plus de sommets à explorer. FIN



CC	A	B	C	D	E
A	•	10_A	∞	∞	5_A
AE	•	8_E	14_E	7_E	•
ADE	•	8_E	13_D	•	•
ABDE	•	•	9_B	•	•
ABCDE	•	•	•	•	•

Si l'on ne considère que les flèches rouges, on obtient un arbre, un graphe sans cycle.

Algorithme itératif de Dijkstra (formulation)

- Restriction: il n'y a pas d'arcs (d'arêtes) de poids négatif dans le graphe G.
- Notation: le graphe G = <S, A> ; S = {1, 2, …, N}, 1 est le sommet source, et on note:
  $\pi(i)$ la distance courante (fonction de l'itération) du sommet A au sommet i.
  $\pi^*(i)$ la longueur minimum des chemins de 1 à i, donc $\pi^*(1) = 0$.
  l_{ij} la longueur de l'arc (i,j) si $(i,j) \in A$, sinon $l_{ij} = \infty$; $\forall i \in S$, $l_{ii} = 0$.
- L'ensemble des sommets S est partitionné en deux : M et CC, où M = S \ CC.
  CC : ensemble des sommets vers lesquels on a déjà trouvé le chemin le plus court à partir de 1.
  M : ensemble des sommets vers lesquels on cherche toujours le chemin le plus court.
- Initialisation:
  $\forall i \in S$, $\pi(i) = l_{1i}$. (Cela veut dire que pour tout i tel que $(1,i) \notin A$, $\pi(i) = \infty$, et que $\pi(1)=0$).
- Au début CC={1} et M = {2, 3, …, N}. On construit progressivement un ensemble CC des sommets i pour lesquels on connaît le plus court chemin de 1 vers x.

Algorithme itératif de Dijkstra en pseudolangage

//Initialisation
CC={1} ; M = {2, 3, …, N}

$\pi(1)=0$; $\pi(j) = \begin{cases} l_{1j} \text{ si } j \in \Gamma(1) \\ +\infty \text{ sinon} \end{cases}$

TANT QUE $M \neq \varnothing$ **FAIRE**
  Sélectionner $j \in M$ tel que $\pi(j) = \min_{i \in M} \pi(i)$
  $M = M \setminus \{j\}$
  $CC = CC \cup \{j\}$
  $\pi^*(j) = \pi(j)$

  SI $M \neq \varnothing$ **ALORS**
    POUR tout $i \in (\Gamma(j) \cap M)$ **FAIRE**
      $\pi(i) = \min(\pi(i), \pi(j) + l_{ji})$
    FINPOUR
  FINSI
FINTANTQUE

Tant que M n'est pas vide

– On choisit dans l'ensemble M un sommet j tel que $\pi(j)$ est le plus petit et on ajoute ce sommet à l'ensemble CC Ainsi, la distance minimale de 1 à j a été trouvée.

– Pour tous les sommets i qui sont successeurs de j et qui appartiennent à M on met dans $\pi(i)$ le minimum de l'ancien $\pi(i)$ et la somme de $\pi(j)$ et l'arc qui mène de j à i.

Algorithme itératif de Dijkstra, exemple 2.

Init.
CC = {1} ; M = {2, 3, 4, 5, 6}
$\pi(2)=7$; $\pi(3) = 1$, $\pi(4) =+\infty$, $\pi(5) =+\infty$, $\pi(6) =+\infty$
Sélect. 3.

$M = \{2, 4, 5, 6\}$; $CC = \{1, 3\}$; $\pi^*(3)=1$
$\Gamma(3)=\{2,5,6\}$; $\Gamma(3)\cap M=\{2,5,6\}$
$\pi(2) = \min(7, 1+5)=6$
$\pi(5) = \min(+\infty, 1+2)=3$
$\pi(6) = \min(+\infty, 1+7)=8$

Sélect. 5

$M = \{2, 4, 6\}$; $CC = \{1, 3, 5\}$; $\pi^*(5)=3$
$\Gamma(5)=\{2,4\}$; $\Gamma(5)\cap M=\{2,4\}$
$\pi(2) = \min(6, 3+2)=5$
$\pi(4) = \min(+\infty, 3+5)=8$

Sélect. 2

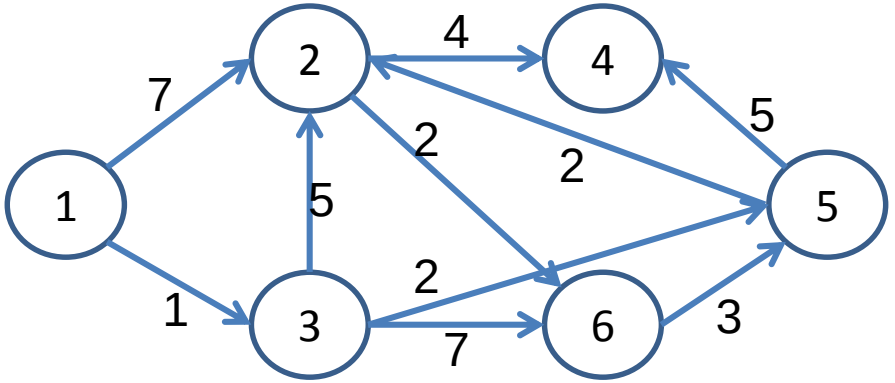
$M = \{4, 6\}$; $CC = \{1, 3, 5, 2\}$; $\pi^*(2)=5$
$\Gamma(2)=\{4,6\}$; $\Gamma(2)\cap M=\{4,6\}$
$\pi(4) = \min(8, 5+4)=8$
$\pi(6) = \min(8, 5+2)=7$

Sélect. 6

$M = \{4\}$; $CC = \{1, 3, 5, 2, 6\}$; $\pi^*(6)=7$
$\Gamma(6)=\{5\}$; $\Gamma(5)\cap M= \varnothing$

Sélect. 4
$M = \varnothing$; $CC = \{1, 2, 3, 4, 5, 6\}$; $\pi^*(4)=8$
FIN
Résultat:

Sommets	1	2	3	4	5	6
Distances	0	5	1	8	3	7



CC	1	2	3	4	5	6
1	0	7	1	∞	∞	∞
13	•	6	•	∞	3	8
135	•	5	•	8	•	8
1235	•	•	•	8	•	7
12356	•	•	•	8	•	•
123456	•	•	•	•	•	•

Algorithme itératif de Dijkstra, remarque sur le nombre d'étapes.

- Si tous les sommets sont accessibles à partir du sommet source (pour un graphe non orienté, cela revient à dire « si le graphe est connexe », mais un graphe orienté peut contenir des sommets non accessibles tout en restant connexe) , alors le nombre d'itérations (d'étapes) est égal au nombre de sommets.
- Dans le cas contraire, la procédure prend fin dès qu'il ne reste que des sommets de poids $+\infty$ dans M et que le dernier sommet ajouté à CC n'ouvre la voie à aucun de ces sommets. Le nombre d'itérations est alors égal au nombre de sommets accessibles.

Algorithme de Dijkstra, cas des longueurs égales

Dans le cas particulier où toutes les longueurs sont égales, on peut utiliser une version de l'algorithme de Dijkstra bien plus efficace. SI on met la valeur de cette longueur égale pour tous les arcs à 1 ($\forall \boldsymbol{u} \in \boldsymbol{A}, \boldsymbol{l}(\boldsymbol{u}) = \mathbf{1}$), on calcule juste le nombre d'arcs (arêtes) minimal pour arriver du sommet de départ à tout autre sommet.

Principe

•On définit $\pi(i)$ comme étant le plus court chemin du sommet 1 au sommet i

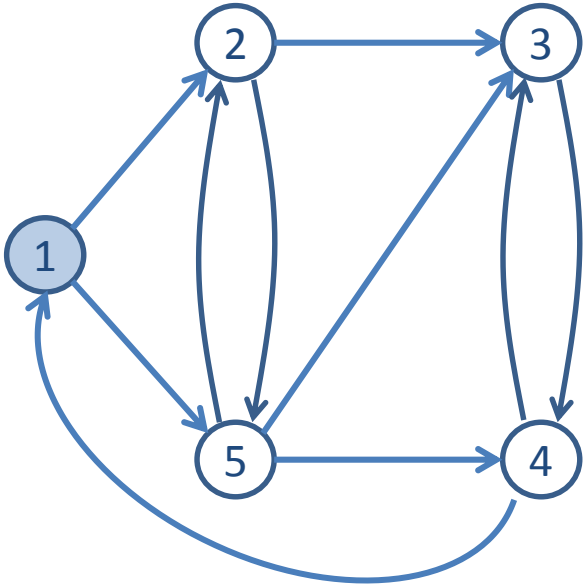
•On initialise $\pi(i) = +\infty$, $\forall i \geq 2$. Dès que $\pi(i)$ devient $\neq +\infty$, cela devient la longueur du plus court chemin

•Le parcours du graphe est un parcours en largeur et chaque sommet n'est visité qu'une seule fois

•La valeur de $\pi(i)$ correspond au nombre d'arcs minimum pour atteindre i

Algorithme de Dijkstra, cas des longueurs égales, exemple

- $\pi(1)=0$, toutes les autres longueurs sont $+\infty$
- On obtient accès aux sommets 2 et 5: $\pi(2)= \pi(5)=1$
- On obtient accès aux sommets 3 et 4: $\pi(3)= \pi(4)=2$
- FIN



1	2	3	4	5
0	∞	∞	∞	∞
•	1_1	∞	∞	1_1
•	•	2_2	2_5	•
•	•	•	•	•

Algorithme de Bellman

- Cet algorithme est valable pour des graphes **sans circuits absorbants.** Les arcs ou les arêtes peuvent avoir des coûts négatifs, contrairement à l'algorithme de Dijkstra.

Algorithme de Bellman

- Cet algorithme est valable pour des graphes **sans circuits absorbants.** Les arcs ou les arêtes peuvent avoir des coûts négatifs, contrairement à l'algorithme de Dijkstra.
- Tout comme l'algorithme de Dijkstra, il donne un plus court chemin **d'un sommet source à tous les autres sommets**.

Algorithme de Bellman

- Cet algorithme est valable pour des graphes **sans circuits absorbants.** Les arcs ou les arêtes peuvent avoir des coûts négatifs, contrairement à l'algorithme de Dijkstra.
- Tout comme l'algorithme de Dijkstra, il donne un plus court chemin **d'un sommet source à tous les autres sommets**.
- Au cas où il y a un circuit absorbant accessible depuis le sommet source, l'algorithme **ne donne pas les chemins les plus courts car il n'y en a pa**s, mais il permet de détecter l'existence du circuit absorbant.

Algorithme de Bellman, pseudocode

Notation:

- 1 : sommet source ;
- $\pi^k(i)$: longueur courante du chemin ($1 \rightarrow i$) à l'itération k ;
- l_{ij} : longueur de l'arc (i, j).

Comme toujours, les arêtes d'un graphe non orienté sont considérées comme des arcs bidirectionnels.

Initialisation: $\pi^0(1) = 0$;
$\forall i \neq 1, \ \pi^0(i) = +\infty$;
$k = 1$.

Tous les sommets sont à $+\infty$, le sommet source à 0

Algorithme de Bellman, pseudocode

Notation:

- 1 : sommet source ;
- $\pi^k(i)$: longueur courante du chemin $(1 \rightarrow i)$ à l'itération k ;
- l_{ij} : longueur de l'arc (i, j).

Comme toujours, les arêtes d'un graphe non orienté sont considérées comme des arcs bidirectionnels.

Initialisation: $\pi^0(1) = 0$;
$\forall i \neq 1, \ \pi^0(i) = +\infty$;
$k = 1$.

Tous les sommets sont à $+\infty$, le sommet source à 0

A l'itération k

FAIRE $\forall i \neq 1, \pi^k(i) = \min(\pi^{k-1}(i), \min_{j \in \Gamma^{-1}(i)}(\pi^{k-1}(j) + l_{ji}))$

Pour tout sommet autre que 1, on choisit comme poids courant le minimum entre :
- son poids à l'itération précédente et
- les poids obtenus en passant par tous ses prédécesseurs dont les poids ont changé à l'itération précédente

Algorithme de Bellman, pseudocode

Notation:

- 1 : sommet source ;
- $\pi^k(i)$: longueur courante du chemin $(1 \rightarrow i)$ à l'itération k ;
- l_{ij} : longueur de l'arc (i, j).

Comme toujours, les arêtes d'un graphe non orienté sont considérées comme des arcs bidirectionnels.

Initialisation: $\pi^0(1) = 0$;
$\forall i \neq 1, \ \pi^0(i) = +\infty$;
$k = 1$.

Tous les sommets sont à $+\infty$, le sommet source à 0

A l'itération k

FAIRE $\forall i \neq 1, \pi^k(i) = \min(\pi^{k-1}(i), \min_{j \in \Gamma^{-1}(i)}(\pi^{k-1}(j) + l_{ji}))$

Pour tout sommet autre que 1, on choisit comme poids courant le minimum entre :
- son poids à l'itération précédente et
- les poids obtenus en passant par tous ses prédécesseurs dont les poids ont changé à l'itération précédente

SI $\exists$ i | $\pi^k(i) \neq \pi^{k-1}(i)$ ALORS
SI $k \leq n - 1$ ALORS $k \leftarrow k + 1$
SINON /* k = n */ il existe un circuit de longueur négative.

Algorithme de Bellman, pseudocode

Notation:

- 1 : sommet source ;
- $\pi^k(i)$: longueur courante du chemin $(1 \rightarrow i)$ à l'itération k ;
- l_{ij} : longueur de l'arc (i, j).

Comme toujours, les arêtes d'un graphe non orienté sont considérées comme des arcs bidirectionnels.

Initialisation: $\pi^0(1) = 0$;
$\forall i \neq 1, \ \pi^0(i) = +\infty$;
$k = 1$.

Tous les sommets sont à $+\infty$, le sommet source à 0

A l'itération k

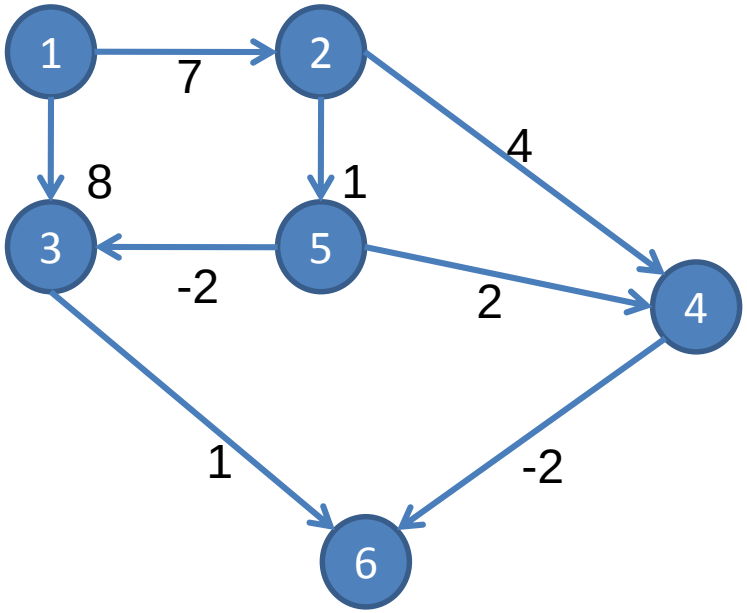
FAIRE $\forall i \neq 1, \pi^k(i) = \min(\pi^{k-1}(i), \min_{j \in \Gamma^{-1}(i)}(\pi^{k-1}(j) + l_{ji}))$

Pour tout sommet autre que 1, on choisit comme poids courant le minimum entre :
- son poids à l'itération précédente et
- les poids obtenus en passant par tous ses prédécesseurs dont les poids ont changé à l'itération précédente

SI $\exists$ i | $\pi^k(i) \neq \pi^{k-1}(i)$ ALORS
SI $k \leq n - 1$ ALORS $k \leftarrow k + 1$
SINON /* k = n */ il existe un circuit de longueur négative.

Donc, si le graphe n'admet pas de circuit de longueur négative, les valeurs finales sont obtenues en **au plus** n – 1 itérations. Si on est sûr que le graphe ne contient pas de circuit absorbant, on peut s'arrêter. Si on ne le sait pas, on fait la n-ième itération, et si on obtient un jeu de $\pi^n(i)$ différent des $\pi^{n-1}(i)$, on s'arrête en déclarant que le graphe comporte un circuit absorbant.

Algorithme de Bellman, exemple



Init. : $\pi^1(1) = 0_1$

k = 1 Successeurs de 1 : 2, 3

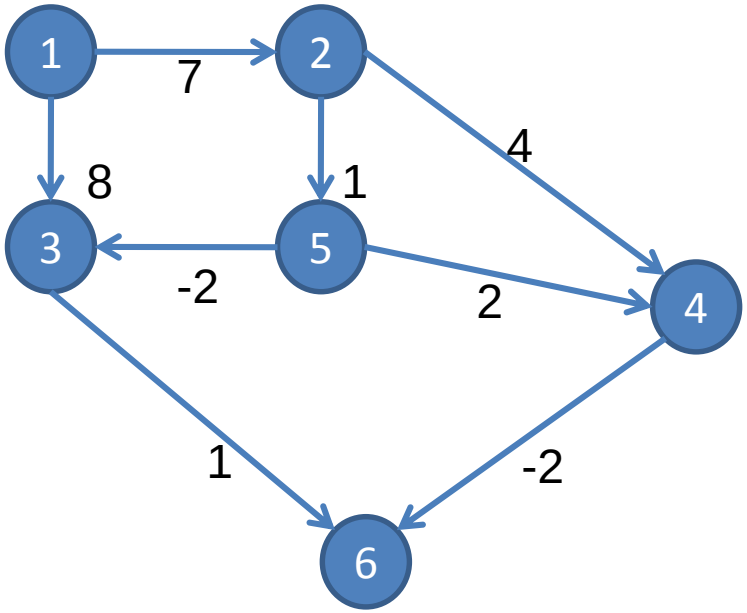
$\pi^1(2) = \min(7_1, +\infty) = 7_1$

$\pi^1(3) = \min(8_1, +\infty) = 8_1$

Les autres restent tels quels

	1	2	3	4	5	6
k=1	0(1)	7(1)	8(1)	+∞	+∞	+∞
2						
3						
4						
5						

Algorithme de Bellman, exemple



Init. : $\pi^1(1) = 0_1$

k = 1 Successeurs de 1 : 2, 3

$\pi^1(2) = \min(7_1, +\infty) = 7_1$

$\pi^1(3) = \min(8_1, +\infty) = 8_1$

Les autres restent tels quels

k = 2 Successeurs de 2, 3 : 4,5,6

$\pi^2(4) = \min((7 + 4)_2, +\infty)$
$= 11_2$

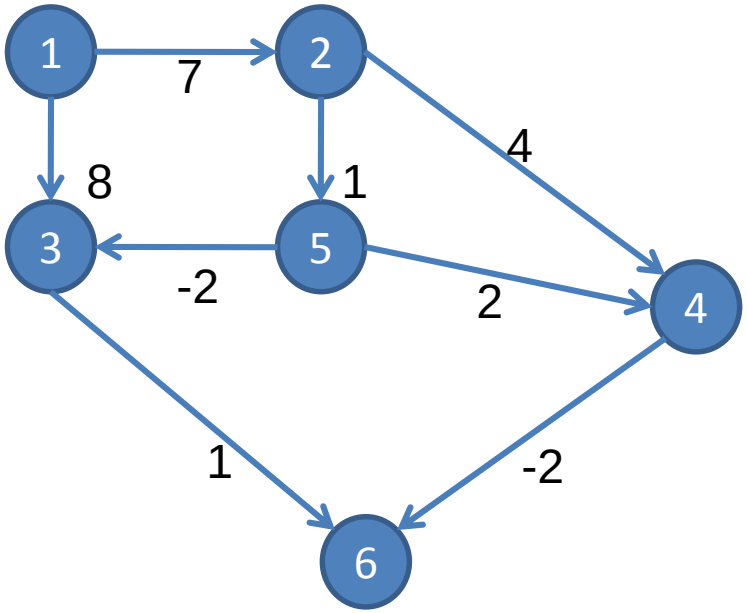
$\pi^2(5) = \min((7 + 1)_2, +\infty)$
$= 8_2$

$\pi^2(6) = \min((8 + 1)_3, +\infty)$
$= 9_3$

Les autres restent tels quels

	1	2	3	4	5	6
k=1	0(1)	7(1)	8(1)	+∞	+∞	+∞
2	0(1)	7(1)	8(1)	11(2)	8(2)	9(3)
3						
4						
5						

Algorithme de Bellman, exemple



Init. : $\pi^1(1) = 0_1$

k = 1 Successeurs de 1 : 2, 3

$\pi^1(2) = \min(7_1, +\infty) = 7_1$

$\pi^1(3) = \min(8_1, +\infty) = 8_1$

Les autres restent tels quels

k = 2 Successeurs de 2, 3 : 4,5,6

$\pi^2(4) = \min((7 + 4)_2, +\infty)$
$= 11_2$

$\pi^2(5) = \min((7 + 1)_2, +\infty)$
$= 8_2$

$\pi^2(6) = \min((8 + 1)_3, +\infty)$
$= 9_3$

Les autres restent tels quels

k = 3 Successeurs de 4: 6, de 5: 3 et 4, de 6 : rien

$\pi^3(3) = \min(8_1, (8 - 2)_5) = 6_5$

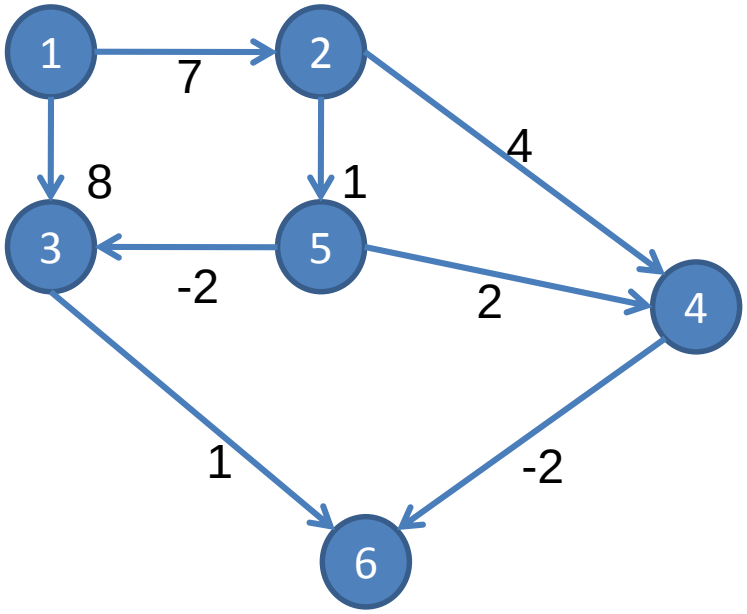
$\pi^3(4) = \min(11_2, (8 + 2)_5) = 10_5$

$\pi^3(6) = \min(9_3, (11 - 2)_4)$
$= 9_{3 \text{ ou } 4}$

Les autres restent tels quels

	1	2	3	4	5	6
k=1	0(1)	7(1)	8(1)	+∞	+∞	+∞
2	0(1)	7(1)	8(1)	11(2)	8(2)	9(3)
3	0(1)	7(1)	6(5)	10(5)	8(2)	9(3ou4)
4						
5						

Algorithme de Bellman, exemple



Init. : $\pi^1(1) = 0_1$

k = 1 Successeurs de 1 : 2, 3

$\pi^1(2) = \min(7_1, +\infty) = 7_1$

$\pi^1(3) = \min(8_1, +\infty) = 8_1$

Les autres restent tels quels

k = 2 Successeurs de 2, 3 : 4,5,6

$\pi^2(4) = \min((7 + 4)_2, +\infty)$
$= 11_2$

$\pi^2(5) = \min((7 + 1)_2, +\infty)$
$= 8_2$

$\pi^2(6) = \min((8 + 1)_3, +\infty)$
$= 9_3$

Les autres restent tels quels

k = 3 Successeurs de 4: 6, de 5: 3 et 4, de 6 : rien

$\pi^3(3) = \min(8_1, (8 - 2)_5) = 6_5$

$\pi^3(4) = \min(11_2, (8 + 2)_5) = 10_5$

$\pi^3(6) = \min(9_3, (11 - 2)_4)$
$= 9_{3 \text{ ou } 4}$

Les autres restent tels quels

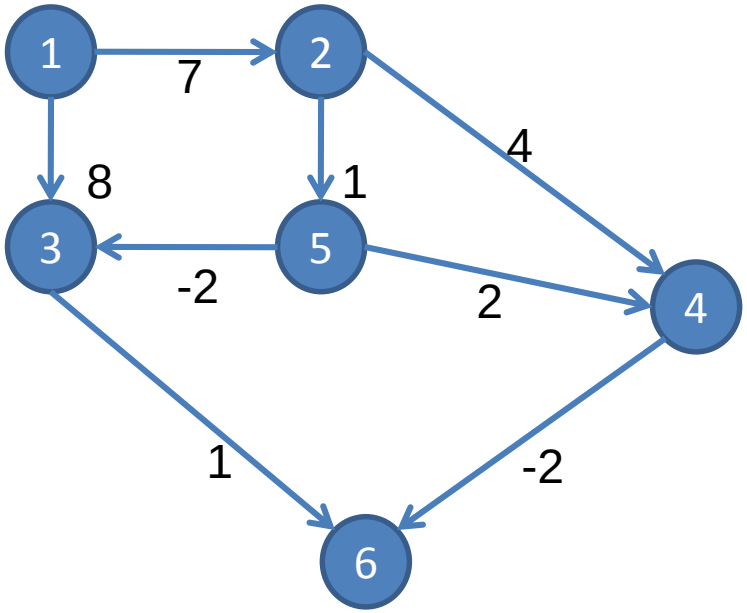
k = 4 Successeurs de 3: 6, de 4: 6, de 6: rien

$\pi^4(6)$
$= \min(9_{3 \text{ ou } 4}, (10 - 2)_4, (6+1)_3)$
$= 7_3$

Les autres restent tels quels

	1	2	3	4	5	6
k=1	0(1)	7(1)	8(1)	+∞	+∞	+∞
2	0(1)	7(1)	8(1)	11(2)	8(2)	9(3)
3	0(1)	7(1)	6(5)	10(5)	8(2)	9(3ou4)
4	0(1)	7(1)	6(5)	10(5)	8(2)	7(3)
5						

Algorithme de Bellman, exemple



Init. : $\pi^1(1) = 0_1$

k = 1 Successeurs de 1 : 2, 3

$\pi^1(2) = \min(7_1, +\infty) = 7_1$

$\pi^1(3) = \min(8_1, +\infty) = 8_1$

Les autres restent tels quels

k = 2 Successeurs de 2, 3 : 4,5,6

$\pi^2(4) = \min((7 + 4)_2, +\infty)$
$= 11_2$

$\pi^2(5) = \min((7 + 1)_2, +\infty)$
$= 8_2$

$\pi^2(6) = \min((8 + 1)_3, +\infty)$
$= 9_3$

Les autres restent tels quels

k = 3 Successeurs de 4: 6, de 5: 3 et 4, de 6 : rien

$\pi^3(3) = \min(8_1, (8 - 2)_5) = 6_5$

$\pi^3(4) = \min(11_2, (8 + 2)_5) = 10_5$

$\pi^3(6) = \min(9_3, (11 - 2)_4)$
$= 9_{3 \text{ ou } 4}$

Les autres restent tels quels

k = 4 Successeurs de 3: 6, de 4: 6, de 6: rien

$\pi^4(6)$
$= \min(9_{3 \text{ ou } 4}, (10 - 2)_4, (6+1)_3)$
$= 7_3$

Les autres restent tels quels

k = 5 Successeurs de 6: rien

aucun changement

(car le seul sommet dont le poids a été modifié à k=4 est 6 qui n'a pas de successeurs)

FIN : car l'itération 5 est identique à l'itération 4.

	1	2	3	4	5	6
k=1	0(1)	7(1)	8(1)	+∞	+∞	+∞
2	0(1)	7(1)	8(1)	11(2)	8(2)	9(3)
3	0(1)	7(1)	6(5)	10(5)	8(2)	9(3ou4)
4	0(1)	7(1)	6(5)	10(5)	8(2)	7(3)
5	0(1)	7(1)	6(5)	10(5)	8(2)	7(3)

Algorithme de Bellman, cas de graphe sans circuit

Si G est sans circuit et si l'ordre des sommets est bien choisi, alors l'algorithme de Bellman converge en une seule fois.

S'il n'y a pas de circuit, il y a au moins un sommet sans prédécesseurs.
Choisissons un tel sommet, et notons-le par 1 : $\Gamma^{-1}(1)=\varnothing$.
Ce sommet est appelé la **racine** du graphe. (S'il y a d'autres sommets sans prédécesseurs, ils ne sont pas accessibles à partir de 1, et leur poids restera $+\infty$).

Notation: comme dans l'algorithme de Dijkstra, CC est l'ensemble des sommets dont le poids minimal on connaît, et M = S\CC contient les autres sommets.

Algorithme

Initialisation : Poser $\pi(1) = 0$ et CC = {1}

Le choix de j peut ne pas être unique

Itérations : TANT QUE $|CC| \neq |S|$ FAIRE

Chercher un sommet $j \in M$ tel que $\Gamma^{-1}(j) \subset CC$

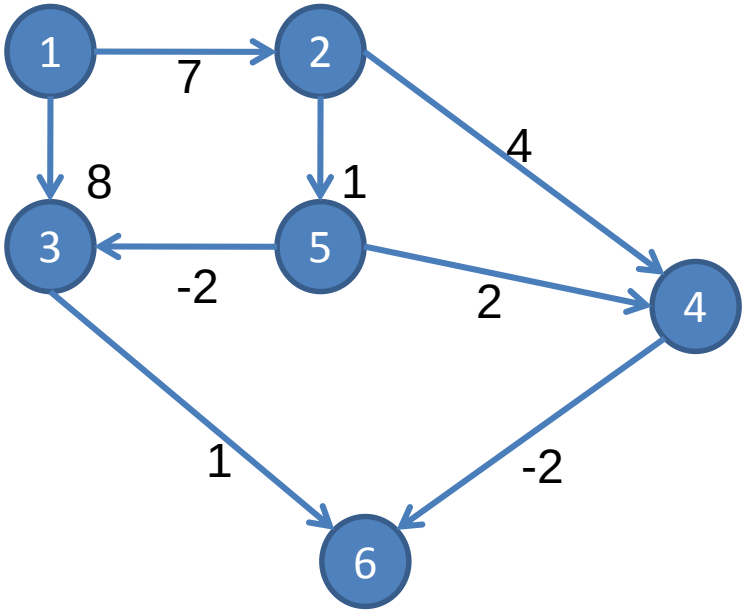
Poser $\pi(j) = \min_{i \in \Gamma^{-1}(j)} (\pi(i) + l_{ij})$

$CC \leftarrow CC \cup \{j\}$

FIN

A chaque itération, un sommet passe vers CC

Algorithme de Bellman, cas de graphe sans circuit, exemple de Bellman



C'est le même graphe que nous avons utilisé pour illustrer l'algo de Bellman "normal". Regardez à quel point cette version d'algorithme est plus efficace!

Initialisation : CC = {1}, $\pi(1)=0$

j=2 (c'est le seule chois possible: on ne peut pas choisir j=3, car il a un prédécesseur hors CC)

$\pi(2) = 7_1$

CC = {1, 2}

j = 5

$\pi(5) = 8_2$

CC = {1, 2, 5}

j = 3

$\pi(3) = \min(8_1, (8 - 2)_5) = 6_5$

CC = {1, 2, 5, 3}

j = 4

$\pi(4) = \min((7+4)_2 , (8+2)_5)=10_5$

CC = {1, 2, 5, 3, 4}

j = 6

$\pi(6) = \min((6+1)_3 ,(10-2)_4) = 7_3$

CC = {1, 2, 5, 3, 4, 6}

Rang d'un sommet

- Dans un graphe orienté sans circuit, on peut établir un ordre partiel des sommets.

- Le rang r(i) d'un sommet i est défini ainsi :

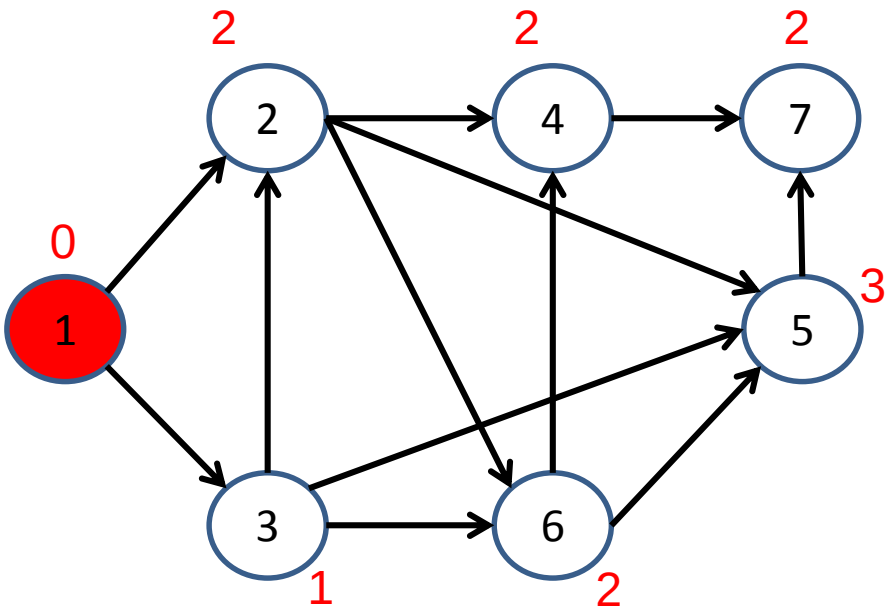
  On note 1 la racine du graphe; r(1)=0

  Pour tout autre sommet,

  r(i)=nombre d'arcs dans un chemin de cardinalité maximum entre sommets 1 et i.

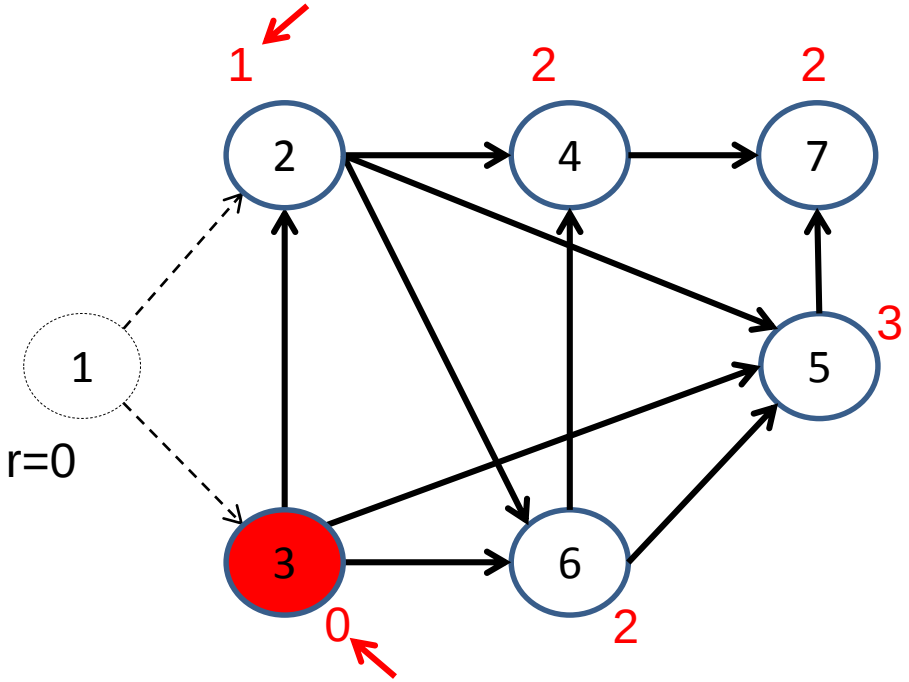
- On peut alors effectuer un tri topologique du graphe, c.à.d. numéroter les sommets de façon compatible avec le rang.

Rang d'un sommet
Exemple 1: une seule racine



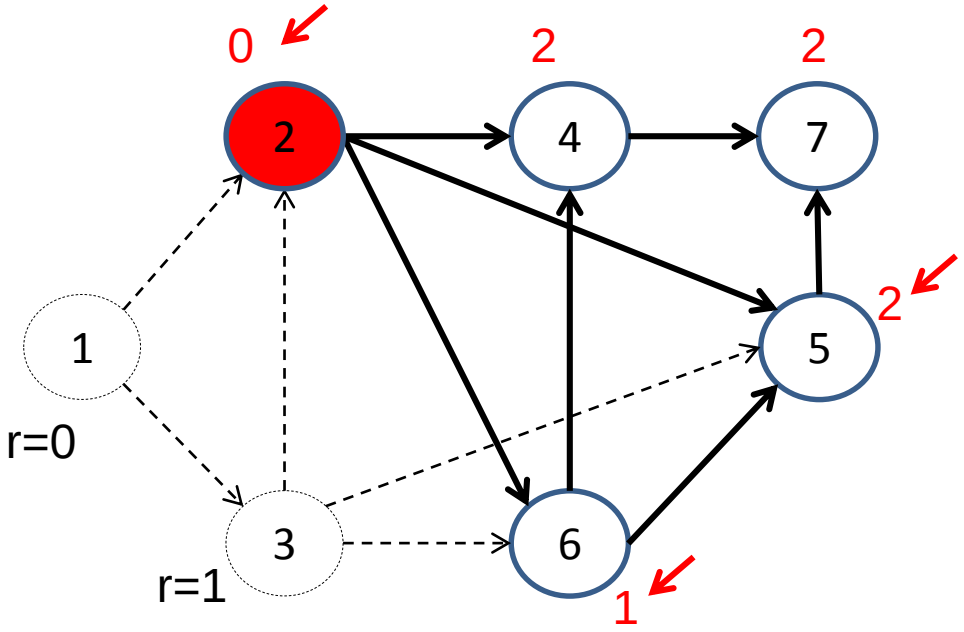
Initialisation: On marque à côté de chaque sommet, en rouge, le nombre d'arcs qui y entrent. Si aucun arc n'y entre, c'est une racine. Ici, la seule racine est 1.

Rang d'un sommet
Exemple 1: une seule racine



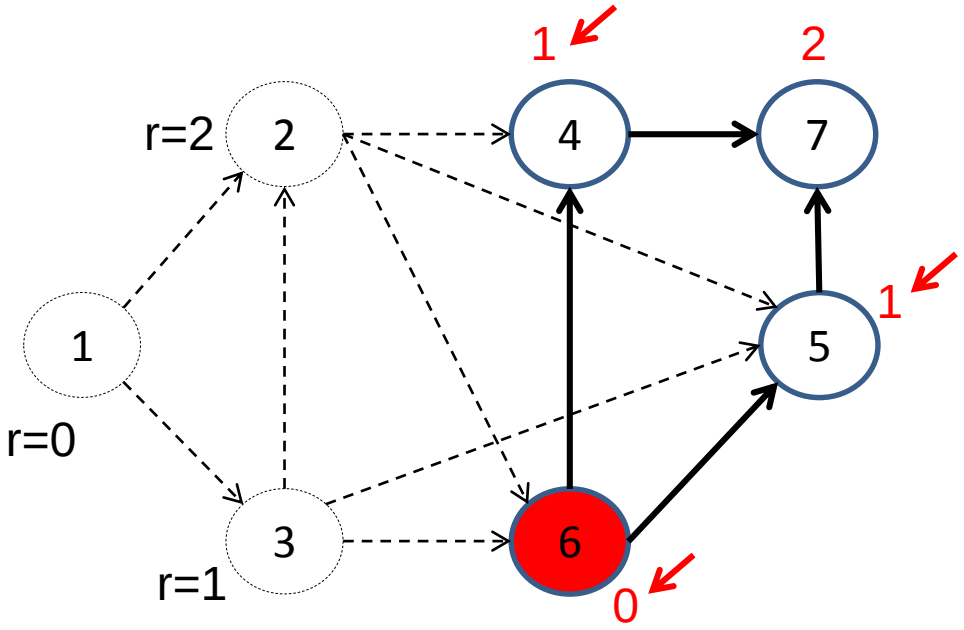
Itération k=0: L'ensemble des racines S_0={1}. On affecte à la racine 1 le rang 0 (valeur courante de k). On élimine le sommet 1 et les arcs qui en sortent. Cela résulte en ce que le nombre d'arcs qui entrent dans chaque successeur de 1 (2 et 3) diminue de 1. Il n'y a plus aucun arc qui entre dans 3; 3 est devenu une racine du sous-graphe obtenu. L'ensemble des racines à la valeur du compteur suivante, k=1, devient S_1 = {3}. On incrémente le compteur k: k=1.

Rang d'un sommet
Exemple 1: une seule racine



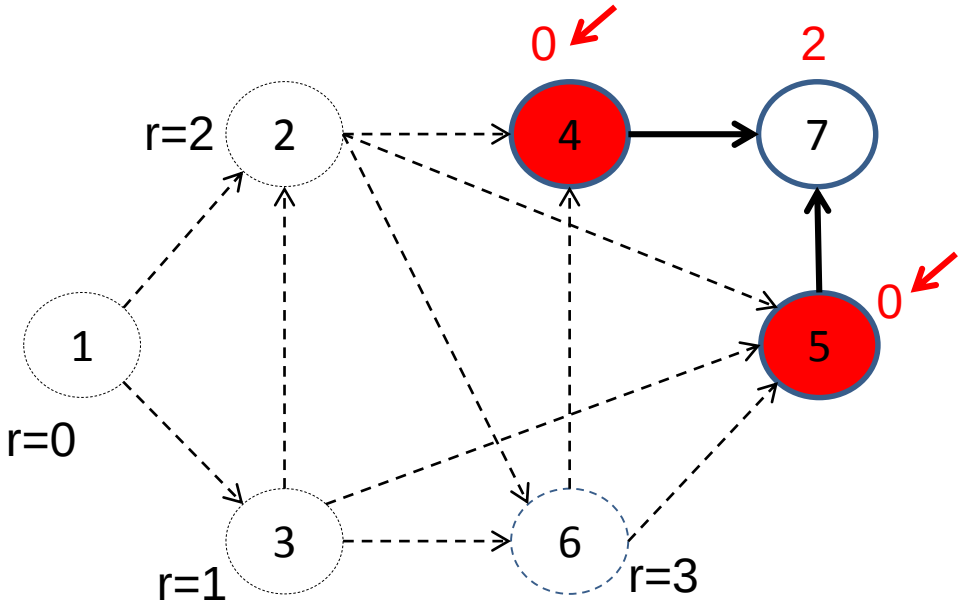
Itération k=1: L'ensemble des racines S_1={3}. On affecte à la racine 3 le rang 1 (valeur courante de k). On élimine le sommet 3 et les arcs qui en sortent. Cela résulte en ce que le nombre d'arcs qui entrent dans chaque successeur de 3 (2, 5 et 6) diminue de 1. Il n'y a plus aucun arc qui entre dans 2; 2 est devenu une racine du sous-graphe obtenu. L'ensemble des racines à la valeur du compteur suivante, k=2, devient S_2 = {2}. On incrémente le compteur k: k=2.

Rang d'un sommet
Exemple 1: une seule racine



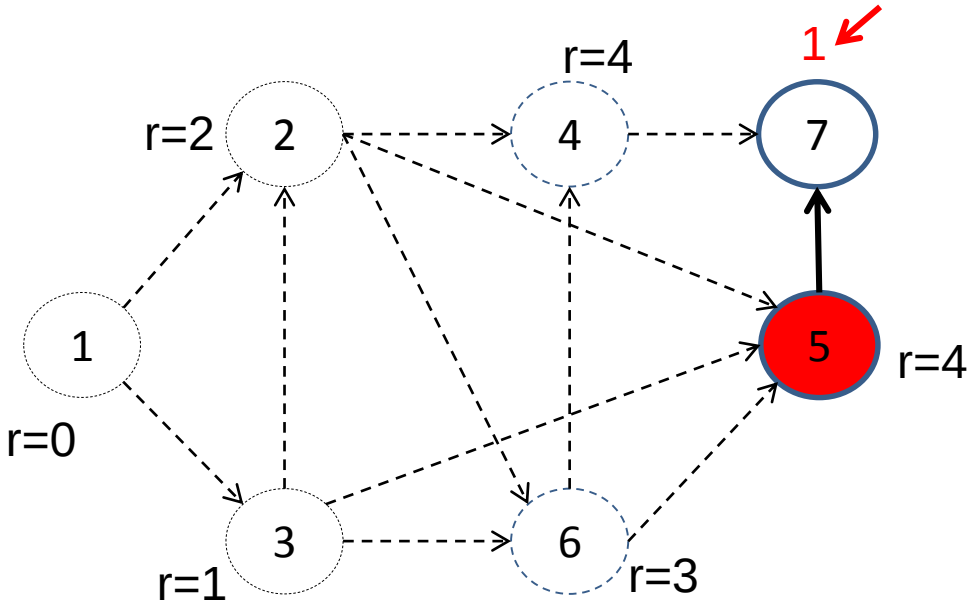
Itération k=2: L'ensemble des racines S_2={2}. On affecte à la racine 2 le rang 2 (valeur courante de k). On élimine le sommet 2 et les arcs qui en sortent. Cela résulte en ce que le nombre d'arcs qui entrent dans chaque successeur de 2 (4, 5 et 6) diminue de 1. Il n'y a plus aucun arc qui entre dans 6; 6 est devenu une racine du sous-graphe obtenu. L'ensemble des racines à la valeur du compteur suivante, k=3, devient S_3 = {6}. On incrémente le compteur k: k=3.

Rang d'un sommet
Exemple 1: une seule racine



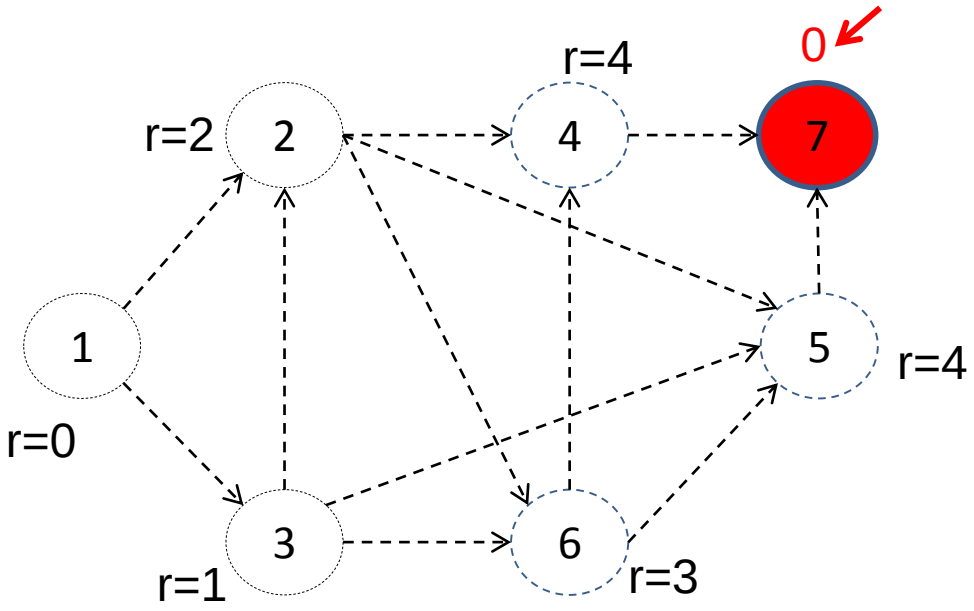
Itération k=3: L'ensemble des racines S_3={6}. On affecte à la racine 6 le rang 3 (valeur courante de k). On élimine le sommet 6 et les arcs qui en sortent. Cela résulte en ce que le nombre d'arcs qui entrent dans chaque successeur de 6 (4 et 5) diminue de 1. Il n'y a plus aucun arc qui entre ni dans 4, ni dans 5; 4 et 5 sont devenus des racines du sous-graphe obtenu. L'ensemble des racines à la valeur du compteur suivante, k=4, devient S_4 = {4,5}. On incrémente le compteur k: k=4.

Rang d'un sommet
Exemple 1: une seule racine



Itération k=4: L'ensemble des racines S_4={4,5}. On leur affecte le rang 4 (valeur courante de k). On élimine d'abord le sommet 4 et l'arc qui en sort. Cela résulte en ce que le nombre d'arcs qui entrent dans 7, le seul successeur de 4, diminue de 1.

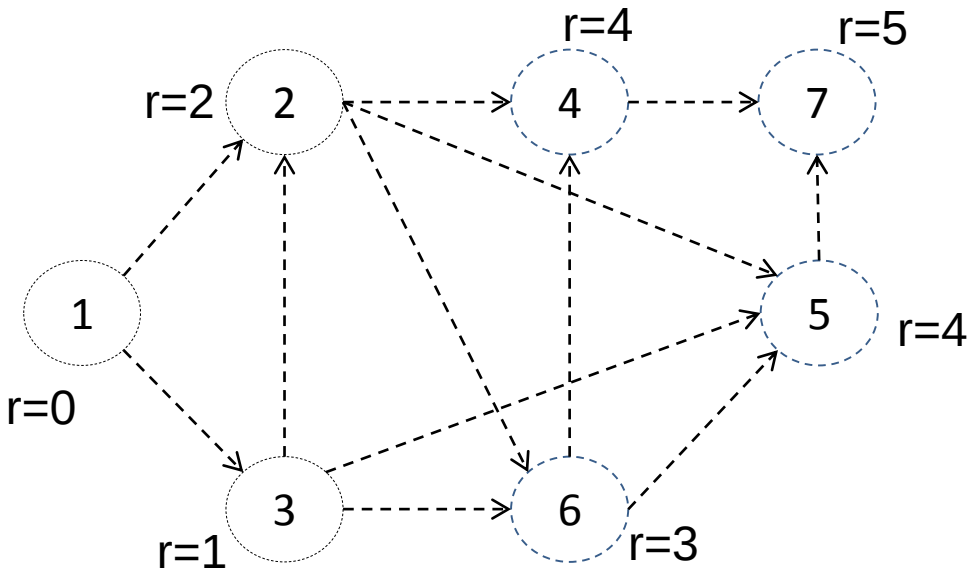
Rang d'un sommet
Exemple 1: une seule racine



Itération k=4: L'ensemble des racines S_4={4,5}. On leur affecte le rang 4 (valeur courante de k). On élimine d'abord le sommet 4 et l'arc qui en sort. Cela résulte en ce que le nombre d'arcs qui entrent dans 7, le seul successeur de 4, diminue de 1.

On élimine maintenant la deuxième racine de S_4={4,5}, le sommet 5, et l'arc qui en sort. Cela résulte en ce que le nombre d'arcs qui entrent dans 7, le seul successeur de 5, diminue de 1 et devient 0. L'ensemble des racines à la valeur du compteur suivante, k=5, devient S_5 = {7}. On incrémente le compteur k: k=5.

Rang d'un sommet
Exemple 1: une seule racine



Itération k=5 : L'ensemble des racines S_5={7}. On affecte au sommet 7 le rang 5 (valeur courante de k). On élimine le sommet 5, et il n'y a pas aucun arc qui en sort : l'ensemble de successeurs de 7 est vide, on s'arrête.

On peut formuler la condition d'arrêt de plusieurs façons :

- On a affecté le rang à tous les sommets
- L'ensemble de successeurs des éléments du dernier ensemble S_n est vide
- etc.

Rang d'un sommet, algorithme

(une des plusieurs variantes d'écriture)

$\forall i \in S, d^{-1}(i) = |\Gamma^{-1}(i)|$
$k = 0$
$S_0 = \{1\}$
$S_{k+1} \leftarrow \varnothing$
FAIRE
  POUR TOUT $i \in S_k$ FAIRE
    $r(i) \leftarrow k$
    POUR TOUT $j \in \Gamma(i)$ FAIRE
      $d^{-1}(j) \leftarrow d^{-1}(j) - 1$
      SI $d^{-1}(j) = 0$ ALORS $S_{k+1} \leftarrow S_{k+1} \cup \{j\}$
    FIN /* POUR TOUT j */
  FIN /* POUR TOUT i */
  $k \leftarrow k+1$
JUSQU'À ce que tous les sommets obtiennent un rang.

$d^{-1}(i)$ est le nombre de prédécesseurs de i ; dans un graphe simple, c'est la même chose que le demi-degré intérieur $d^{0-}(i)$

S_i : l'ensemble des sommets dont le rang est i

Tous les sommets de S_k obtiennent le rang k

On « élimine » du graphe le sommet i, donc les demi-degrés intérieurs de ses successeurs diminuent de 1

Les nouvelles racines forment l'ensemble S_{k+1}

Rang d'un sommet, le même exemple

Initialisation:

$d^{-1}(1) = 0$, $d^{-1}(2) = 2$, $d^{-1}(3) = 1$, $d^{-1}(4) = 2$, $d^{-1}(5) = 3$, $d^{-1}(6) = 2$, $d^{-1}(7) = 2$

$S_0=\{1\}$, k=0

Itérations en k :

r(1) = 0 /*k=0*/
/* $\Gamma(1)=\{2,3\}$ */
j=2
$d^{-1}(2) = 1$
j=3
$d^{-1}(3) = 0$
$S_1 = \{3\}$
k =1

r(3) = 1 /*k=1*/
/* $\Gamma(3)=\{2,5,6\}$ */
j=2
$d^{-1}(2) = 0$
j=5
$d^{-1}(5) = 2$
j=6
$d^{-1}(6) = 1$
$S_2 = \{2\}$
k =2

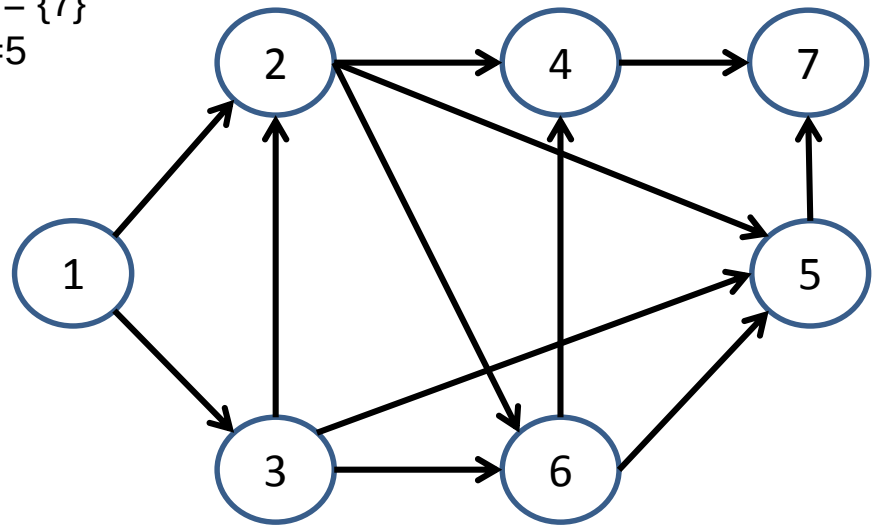
r(2) = 2 /*k=2*/
/* $\Gamma(2)=\{4,5,6\}$ */
j=4
$d^{-1}(4) = 1$
j=5
$d^{-1}(5) = 1$
j=6
$d^{-1}(6) = 0$
$S_3 = \{6\}$
k =3

r(6) = 3 /*k=3*/
/* $\Gamma(6)=\{4,5\}$ */
j=4
$d^{-1}(4) = 0$
$S_4=\{4\}$
j=5
$d^{-1}(5) = 0$
$S_4 = \{4,5\}$
k =4

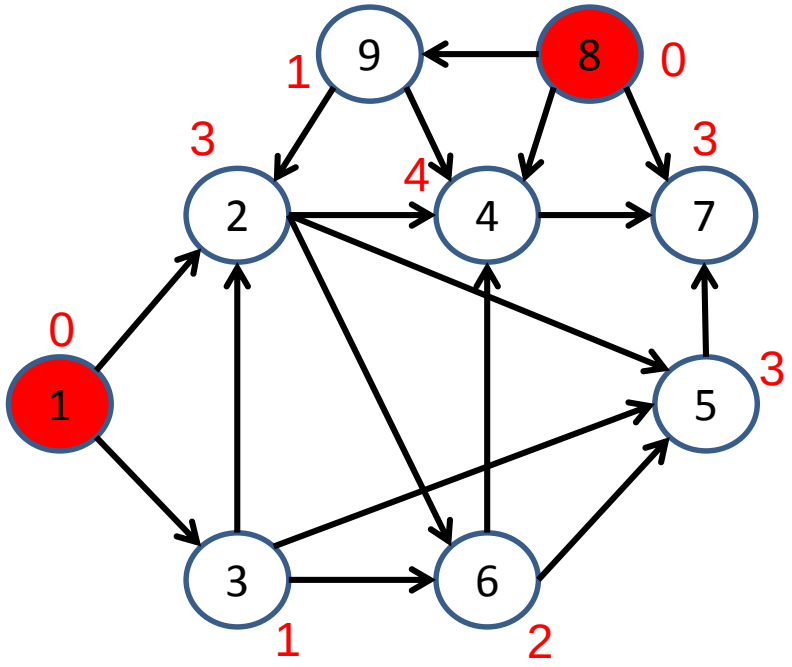
r(4) = 4 /*k=4*/
/* $\Gamma(4)=\{7\}$ */
j=7
$d^{-1}(7) = 1$

r(5) = 4
/* $\Gamma(5)=\{7\}$ */
j=7
$d^{-1}(7) = 0$
$S_5 = \{7\}$
k =5

r(7) = 5 /*k=5*/
/* $\Gamma(7)= \varnothing$ */
FIN



Rang d'un sommet
Exemple 2: deux racines

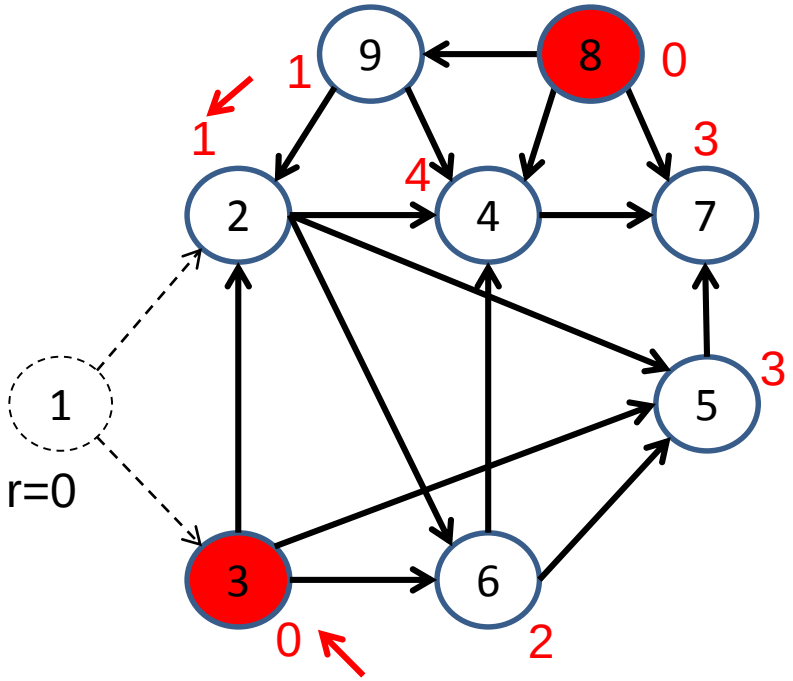


$d^{-1}(1) = 0$, $d^{-1}(2) = 3$, $d^{-1}(3) = 1$, $d^{-1}(4) = 4$,
$d^{-1}(5) = 3$, $d^{-1}(6) = 2$, $d^{-1}(7) = 3$, $d^{-1}(8) = 0$,
$d^{-1}(9) = 1$,
$S_0=\{1,8\}$, k=0

Initialisation: On marque à côté de chaque sommet, en rouge, le nombre d'arcs qui y entrent. Si aucun arc n'y entre, c'est une racine. Ici, il y a deux racines, 1 et 8.

Dans cet exemple, on utilise le même graphe que celui de l'exemple 1, en y ajoutant deux sommets, 8 et 9.

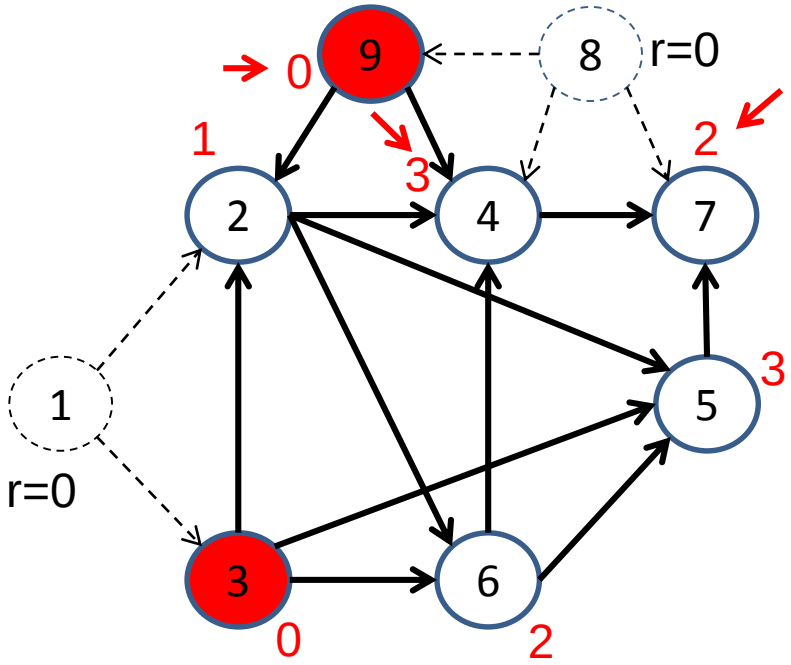
Rang d'un sommet
Exemple 2: deux racines



r(1) = 0 /*k=0*/
/* $\Gamma(1)=\{2,3\}$ */
j=2
$d^{-1}(2) = 1$
j=3
$d^{-1}(3) = 0$
$S_1 = \{3\}$

Itération k=0: L'ensemble des racines $S_0=\{1,8\}$. On affecte d'abord le rang 0 (valeur courante de k) à la racine 1. On élimine le sommet 1 et les arcs qui en sortent. Le nombre d'arcs qui entrent dans chaque successeur de 1 (2 et 3) diminue de 1. Il n'y a plus aucun arc qui entre dans 3; 3 est devenu une racine du sous-graphe obtenu. L'ensemble des racines à la valeur du compteur suivante, k=1, devient $S_1 = \{3\}$.

Rang d'un sommet
Exemple 2: deux racines

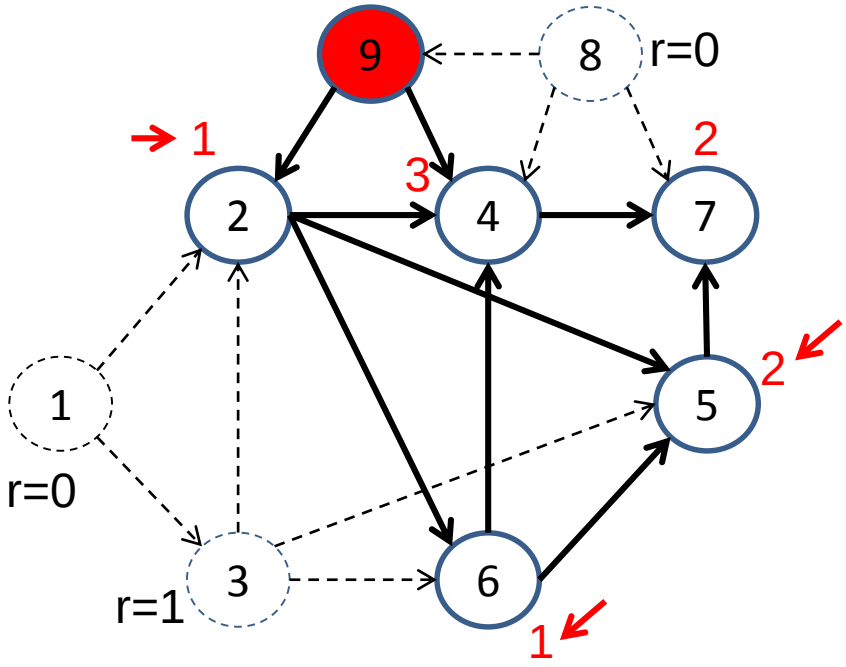


r(1) = 0 /*k=0*/
/* $\Gamma(1)=\{2,3\}$ */
j=2
$d^{-1}(2) = 1$
j=3
$d^{-1}(3) = 0$
$S_1 = \{3\}$

r(8) = 0 /*k=0*/
/* $\Gamma(8)=\{4,7,9\}$ */
j=4
$d^{-1}(4) = 3$
j=7
$d^{-1}(7) = 2$
j=9
$d^{-1}(9) = 0$
$S_1 = \{3,9\}$
k=1

Itération k=0: L'ensemble des racines $S_0=\{1,8\}$. On affecte d'abord le rang 0 (valeur courante de k) à la racine 1. On élimine le sommet 1 et les arcs qui en sortent. Le nombre d'arcs qui entrent dans chaque successeur de 1 (2 et 3) diminue de 1. Il n'y a plus aucun arc qui entre dans 3; 3 est devenu une racine du sous-graphe obtenu. L'ensemble des racines à la valeur du compteur suivante, k=1, devient $S_1 = \{3\}$.
On affecte le rang 0 à la racine 8. On l'élimine du graphe, avec les arcs qui en sortent. Le nombre d'arcs entrant dans ses successeurs diminue de 1. Il n'y a plus aucun arc qui entre dans 9; 9 est devenu une racine du sous-graphe obtenu. L'ensemble des racines à la valeur du compteur suivante, k=1, devient $S_1 = \{3,9\}$. On incrémente k: k=1.

Rang d'un sommet
Exemple 2: deux racines

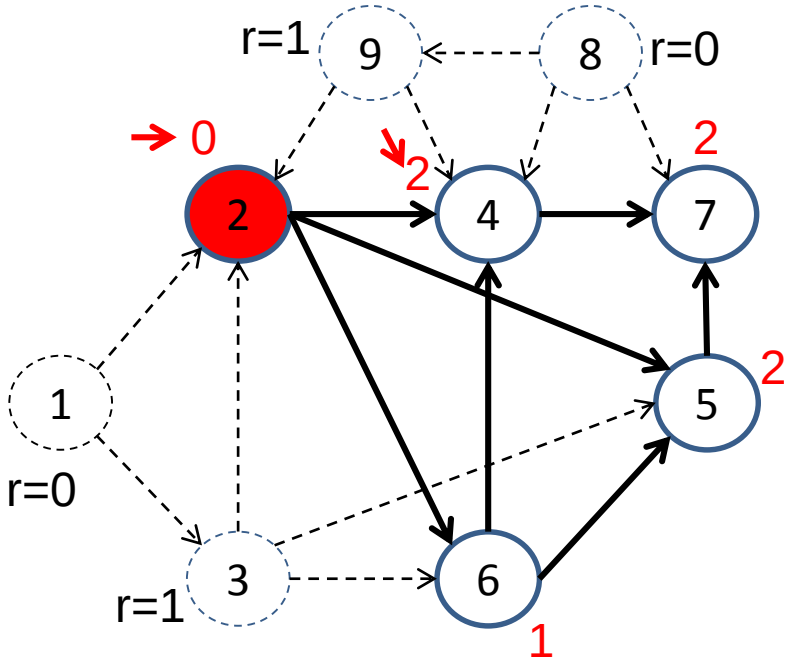


r(3) = 1 /*k=1*/
/* $\Gamma(3)=\{2,5,6\}$ */
j=2
$d^{-1}(2) = 1$
j=5
$d^{-1}(5) = 2$
j=6
$d^{-1}(6) = 1$

S_1 reste $\varnothing$

Itération k=1: L'ensemble des racines $S_1=\{3,9\}$. On affecte d'abord le rang 1 (valeur courante de k) à la racine 3. On élimine le sommet 3 et les arcs qui en sortent. Le nombre d'arcs qui entrent dans chaque successeur de 3 (2, 5 et 6) diminue de 1. Aucun de ces sommets ne devient une racine.

Rang d'un sommet
Exemple 2: deux racines



r(3) = 1 /*k=1*/
/* Γ(3)={2,5,6} */
j=2
$d^{-1}(2) = 1$
j=5
$d^{-1}(5) = 2$
j=6
$d^{-1}(6) = 1$

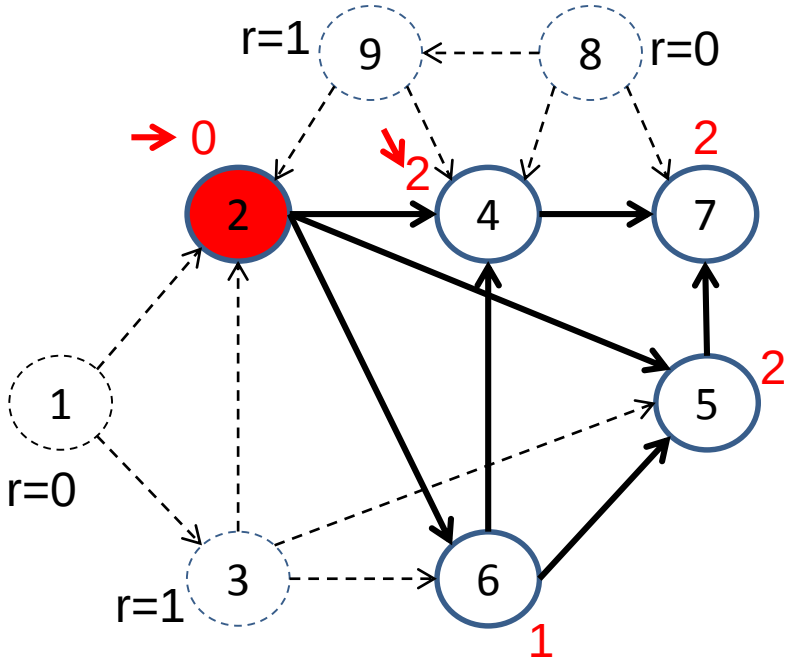
S_2 reste ∅

r(9) = 1 /*k=1*/
/* Γ(9)={2,4} */
j=2
$d^{-1}(2) = 0$
j=4
$d^{-1}(4) = 2$

S_2 ={2}
k=2

Itération k=1: L'ensemble des racines S_1={3,9}. On affecte d'abord le rang 1 (valeur courante de k) à la racine 3. On élimine le sommet 3 et les arcs qui en sortent. Le nombre d'arcs qui entrent dans chaque successeur de 3 (2, 5 et 6) diminue de 1. Aucun de ces sommets ne devient une racine.
On affecte le rang 1 à la racine 9. On élimine le sommet 9 et les arcs qui en sortent. Le nombre d'arcs qui entrent dans chaque successeur de 9 (2 et 4) diminue de 1. Il n'y a plus aucun arc qui entre dans 2; 2 est devenu une racine du sous-graphe obtenu. L'ensemble des racines à la valeur du compteur suivante, k=2, devient S_2 = {2}. On incrémente k: k=2.

Rang d'un sommet
Exemple 2: deux racines



r(3) = 1 /*k=1*/
/* Γ(3)={2,5,6} */
j=2
$d^{-1}(2) = 1$
j=5
$d^{-1}(5) = 2$
j=6
$d^{-1}(6) = 1$

S_2 reste ∅

r(9) = 1 /*k=1*/
/* Γ(9)={2,4} */
j=2
$d^{-1}(2) = 0$
j=4
$d^{-1}(4) = 2$

S_2 ={2}
k=2

On observe qu'à la fin de l'itération k=1 on est arrivé au même sous-graphe que celui de l'exemple 1 (c'est-à-dire que les sommets 8 et 9 n'affectent plus le calcul), donc
au lieu de continuer, on peut utiliser les résultats précédents (diapos 49 – 54), complémentés par les rangs de deux sommets ajoutés dans cet exemple : 8 (r=0) et 9 (r=1).

Tri topologique

La notion du tri topologique s'applique aux graphes orientés sans circuit. C'est une numérotation des sommets compatible avec le rang. Le numéro de chaque sommet doit être inférieur aux numéros de tous ces successeurs. Certains algorithmes deviennent plus efficaces si on prend les sommets selon leur ordre topologique.

Cette numérotation, pour un graphe donné, n'est pas forcément unique.

Pour obtenir une numérotation topologique, on prend tous les sommets de rang 0 et on les numérote dans n'importe quel ordre; puis tous les sommets de rang 1 ; etc.

Pour le graphe de l'exemple 2, avec la table des rangs ci-dessous :

sommets	1	2	3	4	5	6	7	8	9
rangs	0	2	1	4	4	3	5	0	1

on a un ordre topologique possible suivant: : 1, 8, 3, 9, 2, 6, 4, 5, 7.
L'affectation de l'ordre entre 1 et 8, ou 3 et 9, ou 4 et 5 est arbitraire.

Algorithmes matriciels : Algorithme de Floyd

Ils interviennent dans le cas de la recherche d'un plus court chemin entre tous les couples de sommets.

On considère successivement les chemins de i vers j qui ne passent d'abord par aucun autre sommet, puis qui passent éventuellement par le sommet 1, puis par les sommets 1 et 2 etc. A l'étape k, on calcule le coût $d_k(i,j)$ d'un plus court chemin de i à j passant par des sommets inférieurs ou égaux à k. On note $pred_k(i,j)$ le prédécesseur de j dans ce plus court chemin.

On utilise une matrice carrée L d'ordre n, initialisée aux valeurs

$$l_{ij} = \begin{cases} \text{longueur de l'arc } (i,j) \text{ si } (i,j) \in A \\ l_{ii} = 0 \\ +\infty \text{ sinon} \end{cases}$$

On utilise également une matrice carrée P d'ordre n telle que P[i,j]=pred(i,j), s'il existe un chemin de i vers j.

Algorithmes matriciels : Algorithme de Floyd

Entrée : G : Graphe
Sortie : L = tableau[1..n, 1..n] de réel, P = tableau[1..n, 1..n] d'entier
{Le graphe G est valué par des coûts quelconques ; on suppose qu'il n'y a pas de circuit absorbant ; on cherche un plus court chemin entre deux sommets quelconques}

```
{init}
POUR i ← 1 à n FAIRE
        POUR j ← 1 à n FAIRE
                {
                L[i,j] ← coût(i,j,G)
                P[i,j] ←i
                }


{calcul des plus courts chemins}
POUR k ← 1 à n FAIRE
        POUR i ← 1 à n FAIRE
                POUR j ← 1 à n FAIRE
                        SI L[i,k] + L[k,j] < L[i,j] ALORS
                                L[i,j] ← L[i,k] + L[k,j]
                                P[i,j] ← P[k,j]
                        fin
                fin
        fin
```

Algorithme de Floyd

Exemple

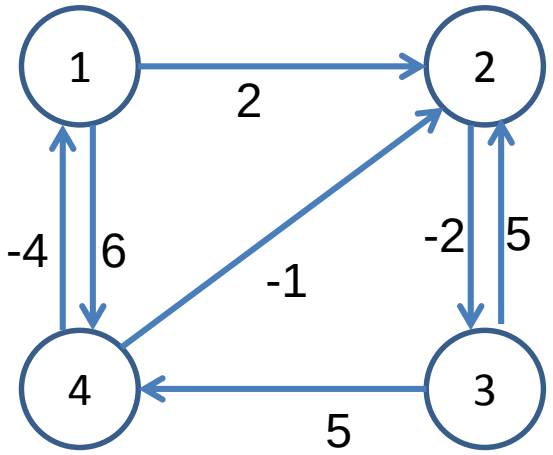
INIT.

L =

0	**2**	∞	**6**
∞	0	-2	∞
∞	5	0	5
-4	-1	∞	0

P =

1	**1**	**1**	**1**
2	2	2	2
3	3	3	3
4	4	4	4



K=1

L=

0	**2**	∞	**6**
∞	0	-2	∞
∞	5	0	5
-4	-2	∞	0

P =

1	**1**	**1**	**1**
2	2	2	2
3	3	3	3
4	1	4	4

Algorithme de Floyd

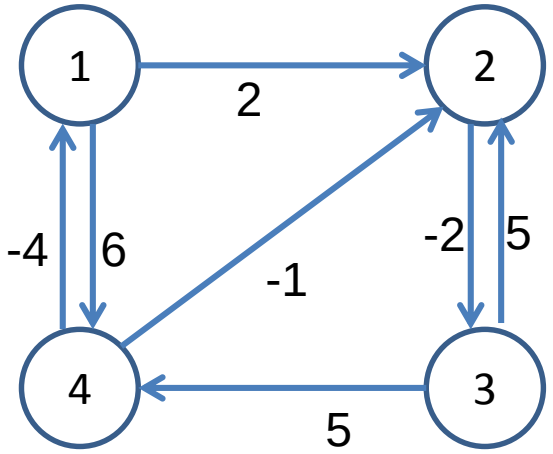
K=2

L =

0	**2**	**0**	**6**
∞	0	-2	∞
∞	5	0	5
-4	-2	-4	0

P =

1	**1**	**2**	**1**
2	2	2	2
3	3	3	3
4	1	2	4



K = 3

L =

0	**2**	**0**	**5**
∞	0	-2	3
∞	5	0	5
-4	-2	-4	0

P =

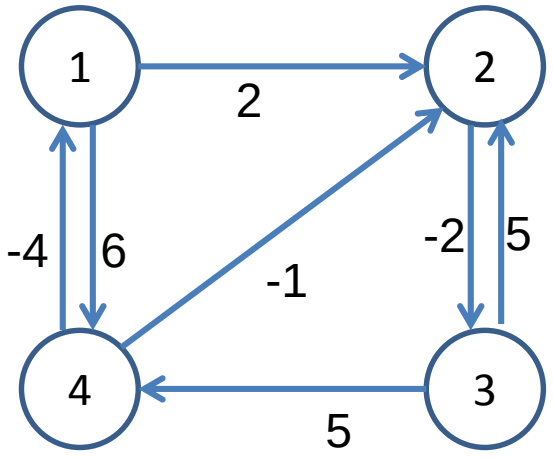
1	**1**	**2**	**3**
2	2	2	3
3	3	3	3
4	1	2	4

Algorithme de Floyd

K=4

L =

0	**2**	**0**	**5**
-1	0	-2	3
1	3	0	5
-4	-2	-4	0



P =

1	**1**	**2**	**3**
4	2	2	3
4	4	3	3
4	1	2	4