Recherche de plus courts chemins

Ivan Noyer

Lycée Thiers

Graphe pondéré

Définition

On dit qu'un triplet $G = (V, E, p)$ est un graphe *pondéré* si le couple (V, E) est un graphe et si p est une fonction de E dans $\mathbb{R}$.

- On dit que p est la (fonction de) *pondération*. Souvent notée w pour *weight*.

Graphe pondéré

Définition

On dit qu'un triplet $G = (V, E, p)$ est un graphe *pondéré* si le couple (V, E) est un graphe et si p est une fonction de E dans $\mathbb{R}$.

- On dit que p est la (fonction de) *pondération*. Souvent notée w pour *weight*.
- Si $e \in E$, alors $p(e)$ est appelé *poids de* e ou *pondération de* e.

Graphe pondéré

Définition

On dit qu'un triplet $G = (V, E, p)$ est un graphe *pondéré* si le couple (V, E) est un graphe et si p est une fonction de E dans $\mathbb{R}$.

- On dit que p est la (fonction de) *pondération*. Souvent notée w pour *weight*.
- Si $e \in E$, alors $p(e)$ est appelé *poids de* e ou *pondération de* e.
- Les graphes sont munis par défaut de la fonction de pondération $p : E \to \mathbb{R},\ e \mapsto 1$. Donc tout graphe est un graphe pondéré qui s'ignore.

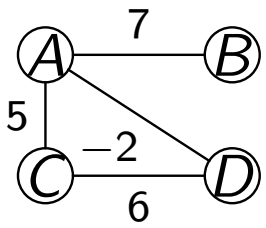
Poids d'une chaîne

Définition

Le *poids* d'une chaîne (resp. chemin) d'un graphe pondéré non orienté (resp. orienté) est la somme des poids des arêtes (resp. arcs) qui la (resp. le) constituent.

Si c est la chaîne $x_1 \dots x_n$:

$$p(c) = \sum_{i=1}^{n-1} p(\{x_i, x_{i+1}\})$$



$$p(ACDA) = -2 + 6 + 5 = 9$$
$$p(ADC) = -2 + 6 = 4 \text{ et } p(AC) = 5$$
Le *plus court chemin* de A à C est ADC

Plus court chemin

- Etant donné un graphe pondéré, on recherche le plus court chemin d'un sommet à un autre (ou à tous les autres, ou le plus court chemin entre tous les couples de sommets...)

Plus court chemin

- Etant donné un graphe pondéré, on recherche le plus court chemin d'un sommet à un autre (ou à tous les autres, ou le plus court chemin entre tous les couples de sommets...)
- Exemple, dans une carte routière de la France, les villes étant des sommets, les routes étant des arcs, chercher le (ou les) trajet(s) de Paris à Mareille qui optimise(nt) l'un des critères suivants :

Plus court chemin

- Etant donné un graphe pondéré, on recherche le plus court chemin d'un sommet à un autre (ou à tous les autres, ou le plus court chemin entre tous les couples de sommets...)
- Exemple, dans une carte routière de la France, les villes étant des sommets, les routes étant des arcs, chercher le (ou les) trajet(s) de Paris à Mareille qui optimise(nt) l'un des critères suivants :
  - la distance parcourue,

Plus court chemin

- Etant donné un graphe pondéré, on recherche le plus court chemin d'un sommet à un autre (ou à tous les autres, ou le plus court chemin entre tous les couples de sommets...)
- Exemple, dans une carte routière de la France, les villes étant des sommets, les routes étant des arcs, chercher le (ou les) trajet(s) de Paris à Mareille qui optimise(nt) l'un des critères suivants :
  - la distance parcourue,
  - le temps mis,

Plus court chemin

- Etant donné un graphe pondéré, on recherche le plus court chemin d'un sommet à un autre (ou à tous les autres, ou le plus court chemin entre tous les couples de sommets...)
- Exemple, dans une carte routière de la France, les villes étant des sommets, les routes étant des arcs, chercher le (ou les) trajet(s) de Paris à Mareille qui optimise(nt) l'un des critères suivants :
  - la distance parcourue,
  - le temps mis,
  - le prix des péages,

Recherche de plus courts chemins

- Dans tout ce qui suit, les graphes sont considérés orientés. Si le graphe G est non orienté, on l'oriente en construisant le graphe orienté tel que

Recherche de plus courts chemins

- Dans tout ce qui suit, les graphes sont considérés orientés. Si le graphe G est non orienté, on l'oriente en construisant le graphe orienté tel que
  - G' a les mêmes sommets que G,

Recherche de plus courts chemins

- Dans tout ce qui suit, les graphes sont considérés orientés. Si le graphe G est non orienté, on l'oriente en construisant le graphe orienté tel que
  - G' a les mêmes sommets que G,
  - pour toute arête $\{x, y\}$ de G, G' possède les arcs (x, y) et (y, x)

Recherche de plus courts chemins

- Dans tout ce qui suit, les graphes sont considérés orientés. Si le graphe G est non orienté, on l'oriente en construisant le graphe orienté tel que
  - G' a les mêmes sommets que G,
  - pour toute arête $\{x, y\}$ de G, G' possède les arcs (x, y) et (y, x)
- On recherche pour tout couple de sommets (i, j) le plus court chemin entre i et j et on calcule sa longueur $d(i, j)$.

Recherche de plus courts chemins

- Dans tout ce qui suit, les graphes sont considérés orientés. Si le graphe G est non orienté, on l'oriente en construisant le graphe orienté tel que
  - G' a les mêmes sommets que G,
  - pour toute arête $\{x, y\}$ de G, G' possède les arcs (x, y) et (y, x)
- On recherche pour tout couple de sommets (i, j) le plus court chemin entre i et j et on calcule sa longueur $d(i, j)$.
- Si j n'est pas accessible depuis i, on pose que $d(i, j) = +\infty$.

Longueur de plus courts chemins

Cas particuliers des graphes orientés non pondérés

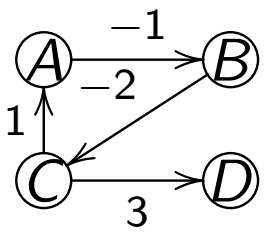
- Si le graphe G est non pondéré, on considère sa pondération par défaut.

Longueur de plus courts chemins

Cas particuliers des graphes orientés non pondérés

- Si le graphe G est non pondéré, on considère sa pondération par défaut.
- Lors du parcours en largeur, les sommets sont explorés par distance croissante au sommet source. Grâce à cette propriété on résout le problème de cheminement suivant : calculer les longueurs des plus courts chemins entre un sommet source et tous les sommets du graphe (voir TD).

Circuits de poids négatif

- Pour rechercher un PCC dans un graphe, il faut s'assurer au préalable que celui-ci ne possède pas de circuit de poids négatif.
- Cela ne veut pas dire que le graphe ne peut pas contenir d'arcs de poids négatif.
- 

$$p(ABCA) = -2$$
$$p(ABCD) = 0$$
$$p(ABCABCD) = -2$$
$$p(ABCABCABCD) = -4$$

Sous-chemin du plus court chemin

- Si un PCC de A à C passe par B, alors le sous-chemin entre A et B est un chemin de poids minimum.

Sous-chemin du plus court chemin

- Si un PCC de A à C passe par B, alors le sous-chemin entre A et B est un chemin de poids minimum.
- En effet, s'il existe un autre chemin plus court entre A et B, il suffit de le mettre à la place du premier pour obtenir un nouveau chemin de A à C encore plus court.

Sous-chemin du plus court chemin

- Si un PCC de A à C passe par B, alors le sous-chemin entre A et B est un chemin de poids minimum.
- En effet, s'il existe un autre chemin plus court entre A et B, il suffit de le mettre à la place du premier pour obtenir un nouveau chemin de A à C encore plus court.
- Ceci contredit le fait que le premier chemin avait un poids minimum.

Sous-chemin du plus court chemin

- Si un PCC de A à C passe par B, alors le sous-chemin entre A et B est un chemin de poids minimum.
- En effet, s'il existe un autre chemin plus court entre A et B, il suffit de le mettre à la place du premier pour obtenir un nouveau chemin de A à C encore plus court.
- Ceci contredit le fait que le premier chemin avait un poids minimum.
- Ceci est un cas particulier du *principe d'optimalité de Bellman* qui dit que l'on peut déduire une solution optimale d'un problème en combinant des solutions optimales d'une série de sous-problèmes.

Sous-chemin du plus court chemin

- Si un PCC de A à C passe par B, alors le sous-chemin entre A et B est un chemin de poids minimum.
- En effet, s'il existe un autre chemin plus court entre A et B, il suffit de le mettre à la place du premier pour obtenir un nouveau chemin de A à C encore plus court.
- Ceci contredit le fait que le premier chemin avait un poids minimum.
- Ceci est un cas particulier du *principe d'optimalité de Bellman* qui dit que l'on peut déduire une solution optimale d'un problème en combinant des solutions optimales d'une série de sous-problèmes.
- Pour les PCC, on l'utilise en calculant d'abord les PCC passant par un sous-ensemble de sommets avant de s'attaquer aux PCC passant par un ensemble de sommets plus gros.

Présentation

- L'algorithme de Floyd-Warshall (parfois appelé algorithme de Roy-Floyd-Warshall car décrit par Bernard Roy en 1959) est un algorithme pour déterminer les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté et pondéré, en temps cubique en le nombre de sommets.

Présentation

- L'algorithme de Floyd-Warshall (parfois appelé algorithme de Roy-Floyd-Warshall car décrit par Bernard Roy en 1959) est un algorithme pour déterminer les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté et pondéré, en temps cubique en le nombre de sommets.
- Prend en entrée un graphe orienté et valué, décrit par une matrice d'adjacence donnant le poids d'un arc lorsqu'il existe et la valeur $+\infty$ sinon. Le graphe ne doit pas posséder de circuit de poids strictement négatif.

Présentation

- L'algorithme de Floyd-Warshall (parfois appelé algorithme de Roy-Floyd-Warshall car décrit par Bernard Roy en 1959) est un algorithme pour déterminer les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté et pondéré, en temps cubique en le nombre de sommets.
- Prend en entrée un graphe orienté et valué, décrit par une matrice d'adjacence donnant le poids d'un arc lorsqu'il existe et la valeur $+\infty$ sinon. Le graphe ne doit pas posséder de circuit de poids strictement négatif.
- C'est un exemple de programmation dynamique.

Présentation

- L'algorithme de Floyd-Warshall (parfois appelé algorithme de Roy-Floyd-Warshall car décrit par Bernard Roy en 1959) est un algorithme pour déterminer les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté et pondéré, en temps cubique en le nombre de sommets.
- Prend en entrée un graphe orienté et valué, décrit par une matrice d'adjacence donnant le poids d'un arc lorsqu'il existe et la valeur $+\infty$ sinon. Le graphe ne doit pas posséder de circuit de poids strictement négatif.
- C'est un exemple de programmation dynamique.
- Notons $\{1, 2, 3, 4, \ldots, n\}$ les sommets.

Présentation

- L'algorithme de Floyd-Warshall (parfois appelé algorithme de Roy-Floyd-Warshall car décrit par Bernard Roy en 1959) est un algorithme pour déterminer les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté et pondéré, en temps cubique en le nombre de sommets.
- Prend en entrée un graphe orienté et valué, décrit par une matrice d'adjacence donnant le poids d'un arc lorsqu'il existe et la valeur $+\infty$ sinon. Le graphe ne doit pas posséder de circuit de poids strictement négatif.
- C'est un exemple de programmation dynamique.
- Notons $\{1, 2, 3, 4, \ldots, n\}$ les sommets.
- On note $\mathcal{W}_{ij}^k$ le poids minimal d'un chemin du sommet i au sommet j n'empruntant que des sommets intermédiaires dans $\{1, 2, 3, \ldots, k\}$ s'il en existe un, et $+\infty$ sinon.

Présentation

- On note $\mathcal{W}^k$ la matrice des $\mathcal{W}^k_{i,j}$.

Présentation

- On note $\mathcal{W}^k$ la matrice des $\mathcal{W}^k_{i,j}$.
- Pour $k = 0$, $\mathcal{W}^0$ est la matrice d'adjacence par poids.

Présentation

- On note $\mathcal{W}^k$ la matrice des $\mathcal{W}^k_{i,j}$.
- Pour $k = 0$, $\mathcal{W}^0$ est la matrice d'adjacence par poids.
- Trouvons une relation de récurrence. On considère un chemin C entre i et j de poids minimal parmi ceux dont les sommets intermédiaires sont dans $\{1, 2, 3, \ldots, k\}$. De deux choses l'une :

Présentation

- On note $\mathcal{W}^k$ la matrice des $\mathcal{W}^k_{i,j}$.
- Pour $k = 0$, $\mathcal{W}^0$ est la matrice d'adjacence par poids.
- Trouvons une relation de récurrence. On considère un chemin C entre i et j de poids minimal parmi ceux dont les sommets intermédiaires sont dans $\{1, 2, 3, \ldots, k\}$. De deux choses l'une :
  - soit C n'emprunte pas le sommet k ;

Présentation

- On note $\mathcal{W}^k$ la matrice des $\mathcal{W}^k_{i,j}$.
- Pour $k = 0$, $\mathcal{W}^0$ est la matrice d'adjacence par poids.
- Trouvons une relation de récurrence. On considère un chemin C entre i et j de poids minimal parmi ceux dont les sommets intermédiaires sont dans $\{1, 2, 3, \ldots, k\}$. De deux choses l'une :
  - soit C n'emprunte pas le sommet k ;
  - soit C emprunte exactement une fois le sommet k (car les circuits sont de poids positifs ou nuls) et C est donc la concaténation de deux chemins, $C_{i,k}$ entre i et k et $C_{k,j}$ entre k et j respectivement, dont les sommets intermédiaires sont dans $\{1, 2, 3, \ldots, k-1\}$. Par principe de sous-optimalité, si C est optimal, $C_{i,k}$, $C_{k,j}$ aussi.

Présentation

- On note $\mathcal{W}^k$ la matrice des $\mathcal{W}^k_{i,j}$.
- Pour $k = 0$, $\mathcal{W}^0$ est la matrice d'adjacence par poids.
- Trouvons une relation de récurrence. On considère un chemin C entre i et j de poids minimal parmi ceux dont les sommets intermédiaires sont dans $\{1, 2, 3, \ldots, k\}$. De deux choses l'une :
  - soit C n'emprunte pas le sommet k ;
  - soit C emprunte exactement une fois le sommet k (car les circuits sont de poids positifs ou nuls) et C est donc la concaténation de deux chemins, $C_{i,k}$ entre i et k et $C_{k,j}$ entre k et j respectivement, dont les sommets intermédiaires sont dans $\{1, 2, 3, \ldots, k-1\}$. Par principe de sous-optimalité, si C est optimal, $C_{i,k}$, $C_{k,j}$ aussi.
- Cela nous donne la relation de récurrence $\mathcal{W}^k_{i,j} = \min(\mathcal{W}^{k-1}_{i,j}, \mathcal{W}^{k-1}_{i,k} + \mathcal{W}^{k-1}_{k,j})$, pour tous i, j et k dans $\{1, 2, 3, 4, \ldots, n\}$. Ainsi on résoud les sous-problèmes par valeur de k croissante.

Pseudocode (1)

```
1  fonction FloydWarshall (G)
2          entree : un graphe orienté pondéré
3          sortie : la matrice des PCC W^n
4          W^0 ← matrice n×n d'adjacence pondérée
5          pour k allant de 1 à n faire
6              créer une matrice W^k
7              pour i allant de 1 à n faire
8                pour j allant de 1 à n faire
9                W^k_{i,j} = min(W^{k-1}_{i,j}, W^{k-1}_{i,k} + W^{k-1}_{k,j})
10               fin_faire
11             fin_faire
12         fin_faire
13         renvoyer W^n
```

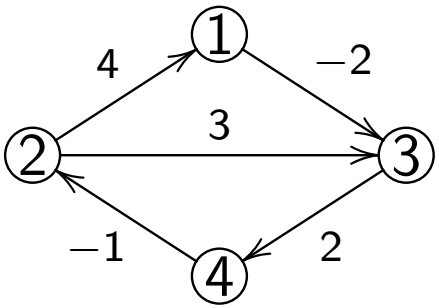
Inconvénient : on crée une matrice à chaque itération. Coûteux en mémoire.

Pseudocode (2)

```
1  fonction FloydWarshall (G)
2          entree : un graphe orienté pondéré
3          sortie : la matrice des PCC W^n
4          W ← matrice n × n d'adjacence pondérée
5          pour k allant de 1 à n faire
6              pour i allant de 1 à n faire
7                  pour j allant de 1 à n faire
8                      W_{i,j} = min(W_{i,j}, W_{i,k} + W_{k,j})
9                  fin_faire
10             fin_faire
11         fin_faire
12         renvoyer W
```

- Avantage : il n'y a plus qu'une matrice modifiée à chaque itération.
- Complexité temporelle en $\ominus(n^3)$ (triple boucle toujours parcourue).
- Complexité en mémoire de l'ordre du nombre de coefficients, donc $\ominus(n^2)$.

Exemple



initialisation

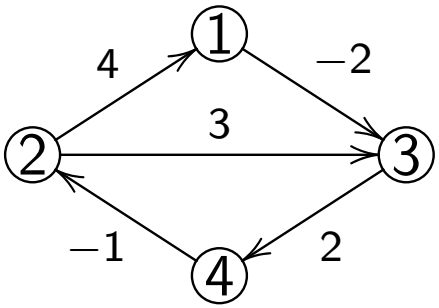
$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

Chemins : $\begin{pmatrix} 13 & \\ 21 & 23 \\ 34 & \\ 42 & \end{pmatrix}$

Pas de circuit de poids négatif.

On ne fera pas mieux que 0 pour aller de x à x

Exemple

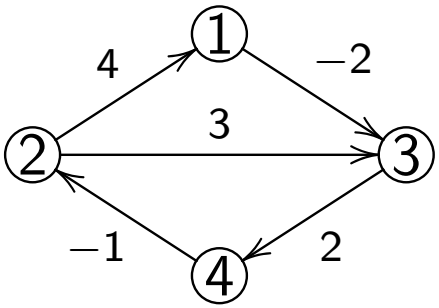


Sommets interm. dans $\{1\}$

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & \min(3, 4-2) & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

Chemins : $\begin{pmatrix} 13 & \\ 21 & 213 \\ 34 & \\ 42 & \end{pmatrix}$

Exemple



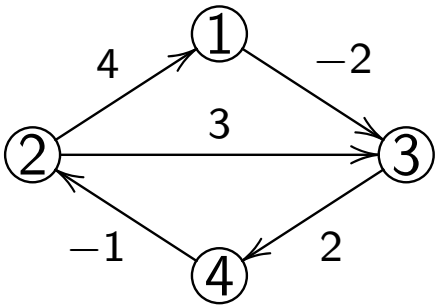
Sommets interm. dans $\{1;2\}$

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 2 & \infty \\ \infty & \infty & 0 & 2 \\ \min(+\infty, -1+4) & -1 & \min(+\infty, -1+2) & 0 \end{pmatrix}$$

Chemins :

$$\begin{pmatrix} 13 & & & \\ 21 & 213 & & \\ 34 & & & \\ 42 & 421 & 4213 & \end{pmatrix}$$

Exemple

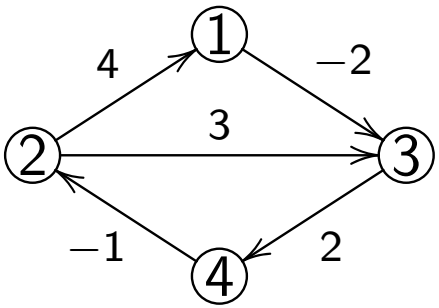


Sommets interm. dans $\{1;2;3\}$

$$\begin{pmatrix} 0 & \infty & -2 & \min(\infty, -2+2) \\ 4 & 0 & 2 & \min(\infty, 2+2) \\ \infty & \infty & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

Chemins : $\begin{pmatrix} 13 & 134 & \\ 21 & 213 & 2134 \\ 34 & & \\ 42 & 421 & 4213 \end{pmatrix}$

Exemple



Sommets interm. dans $\{1;2;3;4\}$

$$\begin{pmatrix} 0 & \min(\infty, 0-1) & -2 & 0 \\ 4 & 0 & 2 & 4 \\ \min(\infty, 3+2) & \min(\infty, -1+2) & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

Chemins : $\begin{pmatrix} 13 & 134 & 1342 \\ 21 & 213 & 2134 \\ 34 & 342 & 3421 \\ 42 & 421 & 4213 \end{pmatrix}$

$k = 4$. On s'arrête là.

Floyd-Warshall

Fermeture transitive

- Le problème de la *fermeture transitive* dans un graphe non pondéré $G = (V, E)$ consiste à déterminer si deux sommets a et b peuvent être reliés par un chemin allant de a à b.

Floyd-Warshall

Fermeture transitive

- Le problème de la *fermeture transitive* dans un graphe non pondéré $G = (V, E)$ consiste à déterminer si deux sommets a et b peuvent être reliés par un chemin allant de a à b.
- Pour cela, on utilise la *matrice d'adjacence bouléenne* $\mathcal{W}$ dans laquelle $\mathcal{W}_{i,j}$ vaut **true** si $(i, j) \in E$ et **false** sinon.

Floyd-Warshall

Fermeture transitive

- Le problème de la *fermeture transitive* dans un graphe non pondéré $G = (V, E)$ consiste à déterminer si deux sommets a et b peuvent être reliés par un chemin allant de a à b.
- Pour cela, on utilise la *matrice d'adjacence bouléenne* $\mathcal{W}$ dans laquelle $\mathcal{W}_{i,j}$ vaut **true** si $(i, j) \in E$ et **false** sinon.
- La relation de récurrence devient

$$\mathcal{W}_{i,j}^{k} \leftarrow \mathcal{W}_{i,j}^{k-1} \vee (\mathcal{W}_{i,k}^{k-1} \wedge \mathcal{W}_{k,j}^{k-1})$$

On ne cherche pas la longueur d'un chemin mais seulement s'il en existe un.

Correction de Floyd-Warshall lorsque G ne contient pas de cycle de poids strictement négatif

Invariant : $\mathcal{W}_{i,j}^k$ est égal au poids d'un chemin minimal reliant v_i à v_j et ne passant que par des sommets intermédiaires de la liste $v_1, v_2, \ldots, v_k$.

- Cas de base : Vrai si $k = 0$ car $\mathcal{W}_{i,j}^k$ est le poids du chemin minimal qui relie v_i à v_j sans passer par aucun sommet intermédiaire. OK

Correction de Floyd-Warshall lorsque G ne contient pas de cycle de poids strictement négatif

Invariant : $\mathcal{W}_{i,j}^k$ est égal au poids d'un chemin minimal reliant v_i à v_j et ne passant que par des sommets intermédiaires de la liste $v_1, v_2, \ldots, v_k$.

- Si $k < n$, on suppose l'invariant réalisé et on considère $C = v_i \rightsquigarrow v_j$ un PCC ne passant que par les sommets intermédiaire $v_1, \ldots, v_{k+1}$.

Correction de Floyd-Warshall lorsque G ne contient pas de cycle de poids strictement négatif

Invariant : $\mathcal{W}_{i,j}^k$ est égal au poids d'un chemin minimal reliant v_i à v_j et ne passant que par des sommets intermédiaires de la liste $v_1, v_2, \ldots, v_k$.

- Si $k < n$, on suppose l'invariant réalisé et on considère $C = v_i \rightsquigarrow v_j$ un PCC ne passant que par les sommets intermédiaire $v_1, \ldots, v_{k+1}$.
  - Si C ne passe pas par v_{k+1}, alors, par HR, son poids est $\mathcal{W}_{i,j}^{k+1} = \mathcal{W}_{i,j}^k$.

Correction de Floyd-Warshall lorsque G ne contient pas de cycle de poids strictement négatif

Invariant : $\mathcal{W}_{i,j}^k$ est égal au poids d'un chemin minimal reliant v_i à v_j et ne passant que par des sommets intermédiaires de la liste $v_1, v_2, \ldots, v_k$.

- Si $k < n$, on suppose l'invariant réalisé et on considère $C = v_i \rightsquigarrow v_j$ un PCC ne passant que par les sommets intermédiaire $v_1, \ldots, v_{k+1}$.
  - Si C ne passe pas par v_{k+1}, alors, par HR, son poids est $\mathcal{W}_{i,j}^{k+1} = \mathcal{W}_{i,j}^k$.
  - S'il passe par v_{k+1}, alors il n'y passe qu'une fois (pas de circuit de poids négatif). Il se décompose en $C_1 = v_i \rightsquigarrow v_{k+1}$ et $C_2 = v_{k+1} \rightsquigarrow v_j$ qui sont des chemins ne passant que par des sommets dans $v_1, \ldots, v_k$.

Correction de Floyd-Warshall lorsque G ne contient pas de cycle de poids strictement négatif

Invariant : $\mathcal{W}_{i,j}^k$ est égal au poids d'un chemin minimal reliant v_i à v_j et ne passant que par des sommets intermédiaires de la liste $v_1, v_2, \ldots, v_k$.

- Si $k < n$, on suppose l'invariant réalisé et on considère $C = v_i \rightsquigarrow v_j$ un PCC ne passant que par les sommets intermédiaire $v_1, \ldots, v_{k+1}$.
  - Si C ne passe pas par v_{k+1}, alors, par HR, son poids est $\mathcal{W}_{i,j}^{k+1} = \mathcal{W}_{i,j}^k$.
  - S'il passe par v_{k+1}, alors il n'y passe qu'une fois (pas de circuit de poids négatif). Il se décompose en $C_1 = v_i \rightsquigarrow v_{k+1}$ et $C_2 = v_{k+1} \rightsquigarrow v_j$ qui sont des chemins ne passant que par des sommets dans $v_1, \ldots, v_k$.
  - Par principe d'optimalité, ces chemins sont les meilleurs ne passant que par $v_1, \ldots, v_k, v_{k+1}$. Donc, par HR, leurs poids sont $\mathcal{W}_{i,k+1}^k$ et $\mathcal{W}_{k+1,j}^k$. Et le poids de C est $\mathcal{W}_{i,k+1}^k + \mathcal{W}_{k+1,j}^k$.

Correction de Floyd-Warshall lorsque G ne contient pas de cycle de poids strictement négatif

Invariant : $\mathcal{W}_{i,j}^k$ est égal au poids d'un chemin minimal reliant v_i à v_j et ne passant que par des sommets intermédiaires de la liste $v_1, v_2, \ldots, v_k$.

- Si $k < n$, on suppose l'invariant réalisé et on considère $C = v_i \rightsquigarrow v_j$ un PCC ne passant que par les sommets intermédiaire $v_1, \ldots, v_{k+1}$.
  - Si C ne passe pas par v_{k+1}, alors, par HR, son poids est $\mathcal{W}_{i,j}^{k+1} = \mathcal{W}_{i,j}^k$.
  - S'il passe par v_{k+1}, alors il n'y passe qu'une fois (pas de circuit de poids négatif). Il se décompose en $C_1 = v_i \rightsquigarrow v_{k+1}$ et $C_2 = v_{k+1} \rightsquigarrow v_j$ qui sont des chemins ne passant que par des sommets dans $v_1, \ldots, v_k$.
  - Par principe d'optimalité, ces chemins sont les meilleurs ne passant que par $v_1, \ldots, v_k, v_{k+1}$. Donc, par HR, leurs poids sont $\mathcal{W}_{i,k+1}^k$ et $\mathcal{W}_{k+1,j}^k$. Et le poids de C est $\mathcal{W}_{i,k+1}^k + \mathcal{W}_{k+1,j}^k$.
  - Finalement, $\min(\mathcal{W}_{i,j}^k, \mathcal{W}_{i,k+1}^k + \mathcal{W}_{k+1,j}^k)$ est le poids minimal d'un chemin reliant v_i à v_j et ne passant que par les sommets intermédiaire $v_1, \ldots, v_{k+1}$.

Correction de Floyd-Warshall lorsque G ne contient pas de cycle de poids strictement négatif

Invariant : $\mathcal{W}^k_{i,j}$ est égal au poids d'un chemin minimal reliant v_i à v_j et ne passant que par des sommets intermédiaires de la liste $v_1, v_2, \ldots, v_k$.

- Si $k < n$, on suppose l'invariant réalisé et on considère $C = v_i \rightsquigarrow v_j$ un PCC ne passant que par les sommets intermédiaire $v_1, \ldots, v_{k+1}$.
  - Si C ne passe pas par v_{k+1}, alors, par HR, son poids est $\mathcal{W}^{k+1}_{i,j} = \mathcal{W}^k_{i,j}$.
  - S'il passe par v_{k+1}, alors il n'y passe qu'une fois (pas de circuit de poids négatif). Il se décompose en $C_1 = v_i \rightsquigarrow v_{k+1}$ et $C_2 = v_{k+1} \rightsquigarrow v_j$ qui sont des chemins ne passant que par des sommets dans $v_1, \ldots, v_k$.
  - Par principe d'optimalité, ces chemins sont les meilleurs ne passant que par $v_1, \ldots, v_k, v_{k+1}$. Donc, par HR, leurs poids sont $\mathcal{W}^k_{i,k+1}$ et $\mathcal{W}^k_{k+1,j}$. Et le poids de C est $\mathcal{W}^k_{i,k+1} + \mathcal{W}^k_{k+1,j}$.
  - Finalement, $\min(\mathcal{W}^k_{i,j}, \mathcal{W}^k_{i,k+1} + \mathcal{W}^k_{k+1,j})$ est le poids minimal d'un chemin reliant v_i à v_j et ne passant que par les sommets intermédiaire $v_1, \ldots, v_{k+1}$.
- Si $k = n$, W contient les longueurs de tous les PCC.

Présentation

- Calcule, dans un graphe orienté pondéré **par des réels positifs**, les plus courts chemins à partir d'une *source unique* en direction de *tous les autres sommets*.

Présentation

- Calcule, dans un graphe orienté pondéré **par des réels positifs**, les plus courts chemins à partir d'une *source unique* en direction de *tous les autres sommets*.
- Dû à l'informaticien néerlandais Edsger Dijkstra (par ailleurs prix Turing),

Présentation

- Calcule, dans un graphe orienté pondéré **par des réels positifs**, les plus courts chemins à partir d'une *source unique* en direction de *tous les autres sommets*.
- Dû à l'informaticien néerlandais Edsger Dijkstra (par ailleurs prix Turing),
- publié en 1959.

Présentation

- Calcule, dans un graphe orienté pondéré **par des réels positifs**, les plus courts chemins à partir d'une *source unique* en direction de *tous les autres sommets*.
- Dû à l'informaticien néerlandais Edsger Dijkstra (par ailleurs prix Turing),
- publié en 1959.
- Comme les arcs sont de poids positifs, on peut supprimer les boucles : passer par une boucle ne raccourcira jamais un chemin. Dans la suite, nos graphes sont sans boucle.

Nécessité de la valuation positive

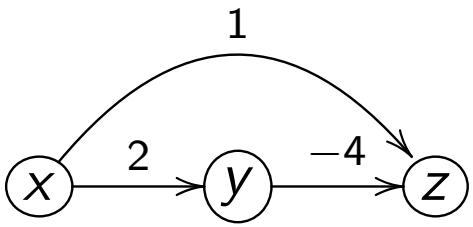
- On serait tenté, si un graphe possède des valuations positives, d'ajouter une même constante à chaque valuation pour les rendre toutes positives afin d'appliquer l'algorithme de Dijkstra.

Nécessité de la valuation positive

- On serait tenté, si un graphe possède des valuations positives, d'ajouter une même constante à chaque valuation pour les rendre toutes positives afin d'appliquer l'algorithme de Dijkstra.
- Malheureusement cette approche permet certes de trouver les PCC dans le nouveau graphe mais le chemin suivi peut très bien ne pas être le meilleur dans l'ancien.

Nécessité de la valuation positive

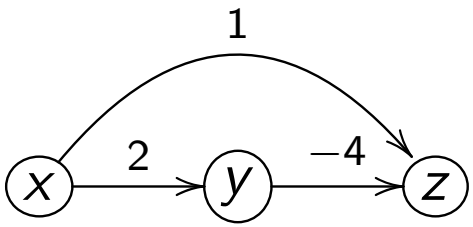
- On serait tenté, si un graphe possède des valuations positives, d'ajouter une même constante à chaque valuation pour les rendre toutes positives afin d'appliquer l'algorithme de Dijkstra.
- Malheureusement cette approche permet certes de trouver les PCC dans le nouveau graphe mais le chemin suivi peut très bien ne pas être le meilleur dans l'ancien.
- Considérons



Le PCC de x à z passe par y.

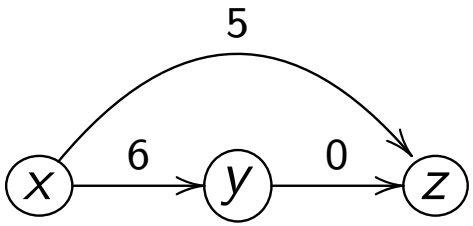
Nécessité de la valuation positive

- On serait tenté, si un graphe possède des valuations positives, d'ajouter une même constante à chaque valuation pour les rendre toutes positives afin d'appliquer l'algorithme de Dijkstra.
- Malheureusement cette approche permet certes de trouver les PCC dans le nouveau graphe mais le chemin suivi peut très bien ne pas être le meilleur dans l'ancien.
- Considérons



Le PCC de x à z passe par y.

- Ajoutons 4 à toutes les valuations



Le PCC de x à z emprunte l'arc $x \to z$.

Principe

- G a pour ensemble de sommets $[\![1, n]\!]$.

Principe

- G a pour ensemble de sommets $[\![1, n]\!]$.
- On cherche les PCC depuis la *source* (ou *entrée*) e vers tous les autres sommets.

Principe

- G a pour ensemble de sommets $[\![1, n]\!]$.
- On cherche les PCC depuis la *source* (ou *entrée*) e vers tous les autres sommets.
- Principe :

Principe

- G a pour ensemble de sommets $[\![1, n]\!]$.
- On cherche les PCC depuis la *source* (ou *entrée*) e vers tous les autres sommets.
- Principe :
  - A chaque tour, on choisit parmi tous les sommets verts celui dont la distance à l'entrée est minimale (le plus court chemin). Ce sommet devient rouge.

Principe

- G a pour ensemble de sommets $[\![1, n]\!]$.
- On cherche les PCC depuis la *source* (ou *entrée*) e vers tous les autres sommets.
- Principe :
  - A chaque tour, on choisit parmi tous les sommets verts celui dont la distance à l'entrée est minimale (le plus court chemin). Ce sommet devient rouge.
  - Tout voisin bleu du sommet choisi est ajouté à l'ensemble des sommets verts (en ce sens c'est un parcours en largeur).
    Les informations concernant les distances à la source des voisins du sommet qui vient de devenir rouge sont mises à jour.

Principe

- G a pour ensemble de sommets $[\![1, n]\!]$.
- On cherche les PCC depuis la *source* (ou *entrée*) e vers tous les autres sommets.
- Principe :
  - A chaque tour, on choisit parmi tous les sommets verts celui dont la distance à l'entrée est minimale (le plus court chemin). Ce sommet devient rouge.
  - Tout voisin bleu du sommet choisi est ajouté à l'ensemble des sommets verts (en ce sens c'est un parcours en largeur).
    Les informations concernant les distances à la source des voisins du sommet qui vient de devenir rouge sont mises à jour.
- C'est un algorithme glouton dont le résultat est optimal.
  Nous avons vu qu'il existe des algos gloutons non optimal (par exemple pour la coloration).

Principe

L'ensemble des sommets verts est noté F car il est souvent implémenté avec une file de priorité. Graphe $G = (S, A)$.

```
1  F:={e} /*sommets en cours de traitement (=verts)*/
2  E:=∅ /*sommets traités = sommets rouges*/
3  D:= tableau des distances minimales (d_e = 0, k ≠ e ⟹ d_k = ∞)
4  tant que F ≠ ∅ faire
5        choisir k ∈ F avec d_k minimal
6        F:=F \ {k}
7        E:=E ∪ {k} /* k devient rouge*/
8        pour tout voisin v de k non rouge faire
9              si v ∉ F : /*si v est bleu*/
10                F:=F ∪ {v} /*v devient vert*/
11             fin_si
12             si d_k + w(k → v) < d_v :
13                   /*passer par k pour atteindre v est plus rentable*/
14                   d_v:=d_k + w(k → v) /*maj tab. des distances*/
15             fin_si
16       fin_faire
17 fin_faire
```

Variant

Pour un graphe $G = (S, A)$, on note $B = S \setminus (F \cup E)$ l'ensemble des sommets bleus.

- Variant : $|F| + |B|$ (cardinal de l'ensemble de sommets verts ou bleus). Quantité entière positive.

Variant

Pour un graphe $G = (S, A)$, on note $B = S \setminus (F \cup E)$ l'ensemble des sommets bleus.

- Variant : $|F| + |B|$ (cardinal de l'ensemble de sommets verts ou bleus). Quantité entière positive.
- C'est une quantité décroissante strictement à la fin de chaque tour de boucle car :

Variant

Pour un graphe $G = (S, A)$, on note $B = S \setminus (F \cup E)$ l'ensemble des sommets bleus.

- Variant : $|F| + |B|$ (cardinal de l'ensemble de sommets verts ou bleus). Quantité entière positive.
- C'est une quantité décroissante strictement à la fin de chaque tour de boucle car :
  - on retire toujours un sommet de F (si ce n'est pas possible, $F = \emptyset$ et l'algorithme s'arrête).

Variant

Pour un graphe $G = (S, A)$, on note $B = S \setminus (F \cup E)$ l'ensemble des sommets bleus.

- Variant : $|F| + |B|$ (cardinal de l'ensemble de sommets verts ou bleus). Quantité entière positive.
- C'est une quantité décroissante strictement à la fin de chaque tour de boucle car :
  - on retire toujours un sommet de F (si ce n'est pas possible, $F = \emptyset$ et l'algorithme s'arrête).
  - tout sommet ajouté à F est retiré de B.

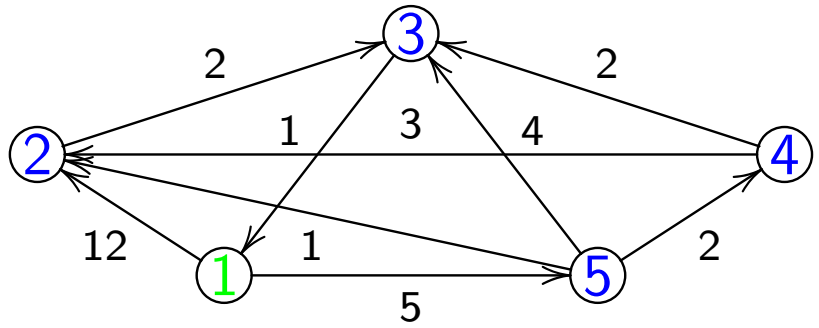
Variant

Pour un graphe $G = (S, A)$, on note $B = S \setminus (F \cup E)$ l'ensemble des sommets bleus.

- Variant : $|F| + |B|$ (cardinal de l'ensemble de sommets verts ou bleus). Quantité entière positive.
- C'est une quantité décroissante strictement à la fin de chaque tour de boucle car :
  - on retire toujours un sommet de F (si ce n'est pas possible, $F = \emptyset$ et l'algorithme s'arrête).
  - tout sommet ajouté à F est retiré de B.
  - ainsi, l'ensemble ds sommets verts ou bleus comporte un élément de moins à la fin d'un tour par rapport au tour précédent.

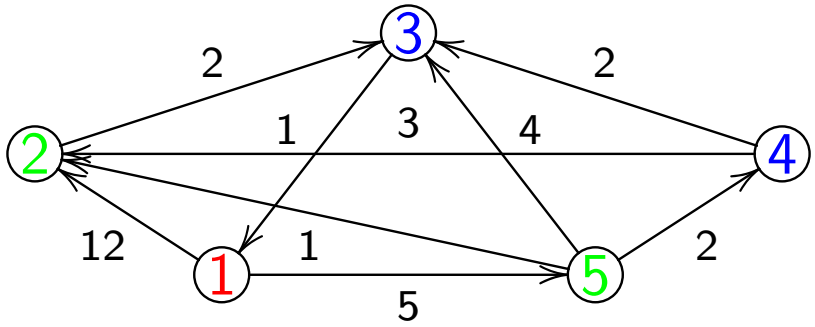
Exemple

On gère en plus un tableau **P** (prédecesseur)qui garde en mémoire la dernière étape sur le chemin de la source au sommet considéré.



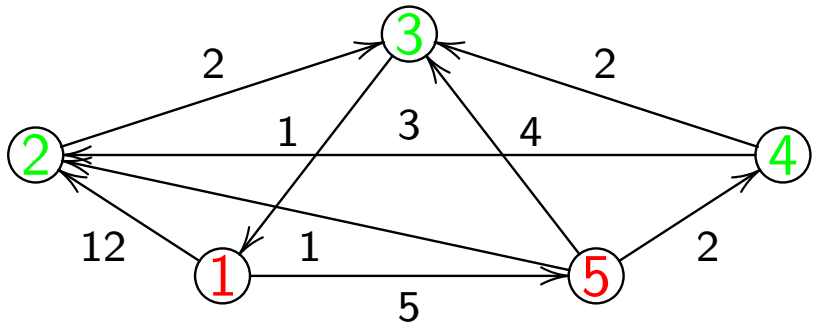
F	E	T	P
$\{1\}$	$\emptyset$	$[\![0;\infty;\infty;\infty;\infty]\!]$	$[\![1;-1;-1;-1;-1]\!]$

Exemple



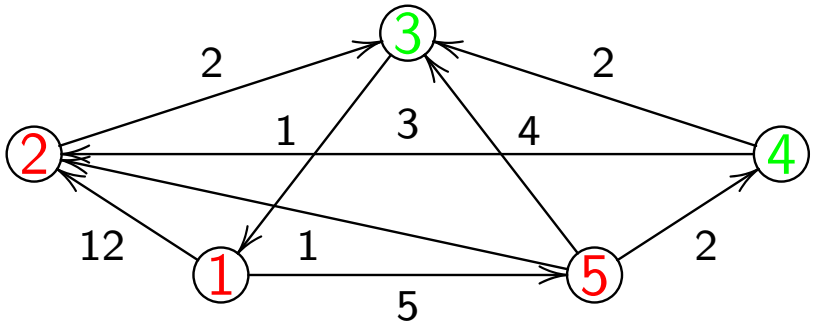
F	E	T	P
$\{1\}$	$\emptyset$	$[\![0;\infty;\infty;\infty;\infty]\!]$	$[\![1;-1;-1;-1;-1]\!]$
$\{2;5\}$	$\{1\}$	$[\![0;12;\infty;\infty;5]\!]$	$[\![1;1;-1;-1;1]\!]$

Exemple



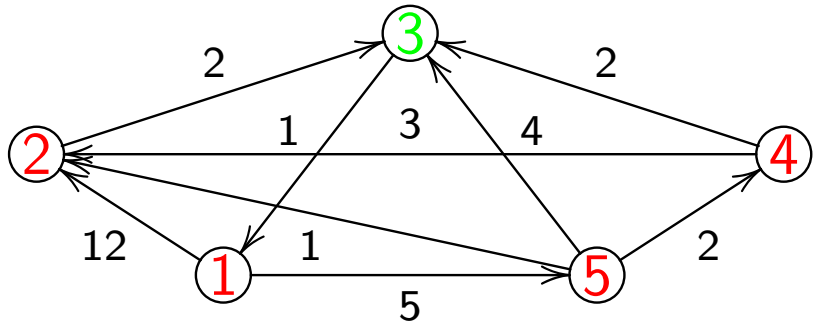
F	E	T	P
$\{1\}$	$\emptyset$	$[\![0;\infty;\infty;\infty;\infty]\!]$	$[\![1;-1;-1;-1;-1]\!]$
$\{2;5\}$	$\{1\}$	$[\![0;12;\infty;\infty;5]\!]$	$[\![1;1;-1;-1;1]\!]$
$\{2;3;4\}$	$\{1;5\}$	$[\![0;6;9;7;5]\!]$	$[\![1;5;5;5;1]\!]$

Exemple



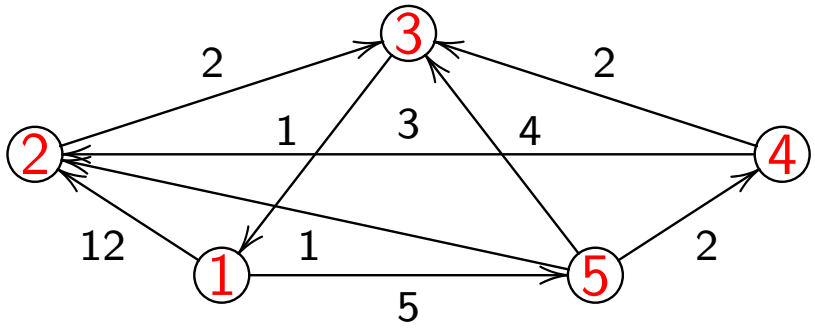
F	E	T	P
$\{1\}$	$\emptyset$	$[\![0;\infty;\infty;\infty;\infty]\!]$	$[\![1;-1;-1;-1;-1]\!]$
$\{2;5\}$	$\{1\}$	$[\![0;12;\infty;\infty;5]\!]$	$[\![1;1;-1;-1;1]\!]$
$\{2;3;4\}$	$\{1;5\}$	$[\![0;6;9;7;5]\!]$	$[\![1;5;5;5;1]\!]$
$\{3;4\}$	$\{1;5;2\}$	$[\![0;6;8;7;5]\!]$	$[\![1;5;2;5;1]\!]$

Exemple



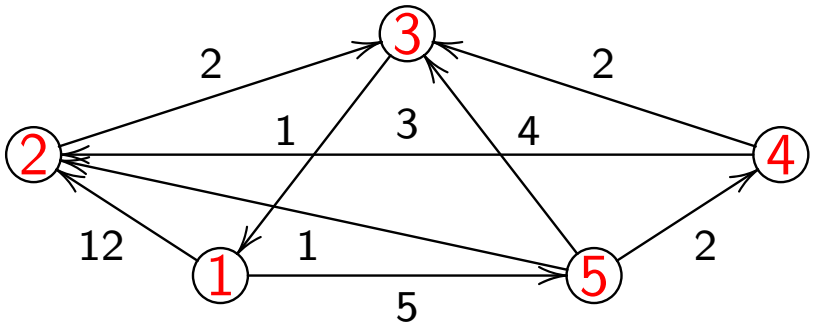
F	E	T	P
$\{1\}$	$\emptyset$	$[\![0;\infty;\infty;\infty;\infty]\!]$	$[\![1;-1;-1;-1;-1]\!]$
$\{2;5\}$	$\{1\}$	$[\![0;12;\infty;\infty;5]\!]$	$[\![1;1;-1;-1;1]\!]$
$\{2;3;4\}$	$\{1;5\}$	$[\![0;6;9;7;5]\!]$	$[\![1;5;5;5;1]\!]$
$\{3;4\}$	$\{1;5;2\}$	$[\![0;6;8;7;5]\!]$	$[\![1;5;2;5;1]\!]$
$\{3\}$	$\{1;5;2;4\}$	$[\![0;6;8;7;5]\!]$	$[\![1;5;2;5;1]\!]$

Exemple



F	E	T	P
$\{1\}$	$\emptyset$	$[\![0;\infty;\infty;\infty;\infty]\!]$	$[\![1;-1;-1;-1;-1]\!]$
$\{2;5\}$	$\{1\}$	$[\![0;12;\infty;\infty;5]\!]$	$[\![1;1;-1;-1;1]\!]$
$\{2;3;4\}$	$\{1;5\}$	$[\![0;6;9;7;5]\!]$	$[\![1;5;5;5;1]\!]$
$\{3;4\}$	$\{1;5;2\}$	$[\![0;6;f8;7;5]\!]$	$[\![1;5;2;5;1]\!]$
$\{3\}$	$\{1;5;2;4\}$	$[\![0;6;8;7;5]\!]$	$[\![1;5;2;5;1]\!]$
$\emptyset$	$\{1;5;2;4;3\}$	$[\![0;6;8;7;5]\!]$	$[\![1;5;2;5;1]\!]$

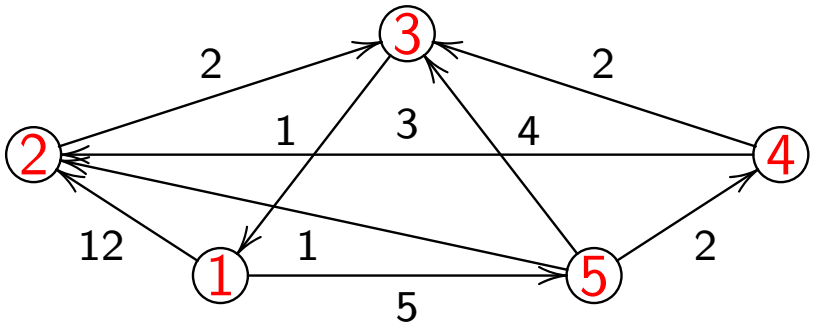
Exemple



PCC pour aller de 1 à 3 :

- On a $T = [\![0; 6; 8; 7; 5]\!]$ et $P = [\![1; 5; 2; 5; 1]\!]$.

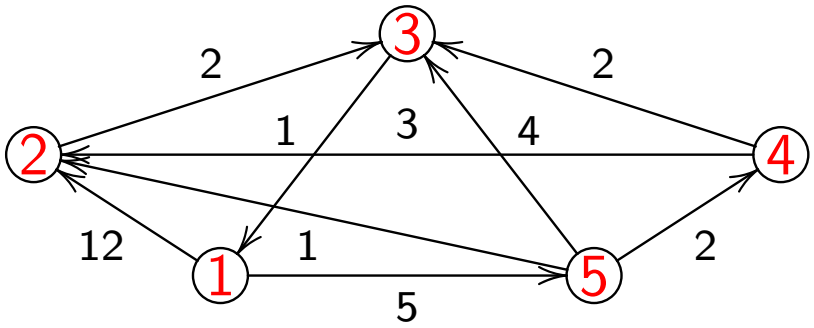
Exemple



PCC pour aller de 1 à 3 :

- On a $T = [\![0; 6; 8; 7; 5]\!]$ et $P = [\![1; 5; 2; 5; 1]\!]$.
- prédecesseur de 3 : 2

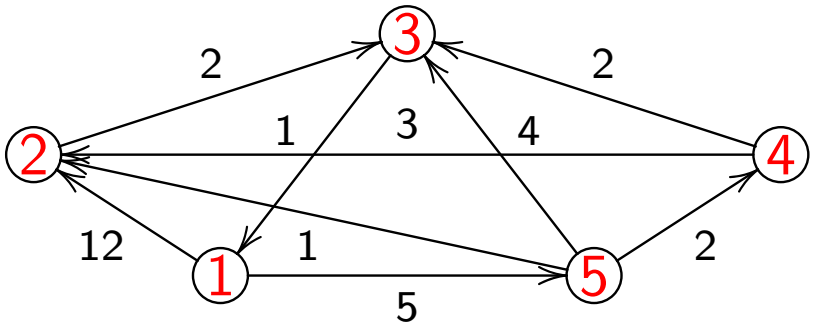
Exemple



PCC pour aller de 1 à 3 :

- On a $T = [\![0; 6; 8; 7; 5]\!]$ et $P = [\![1; 5; 2; 5; 1]\!]$.
- prédecesseur de 3 : 2
- prédecesseur de 2 : 5

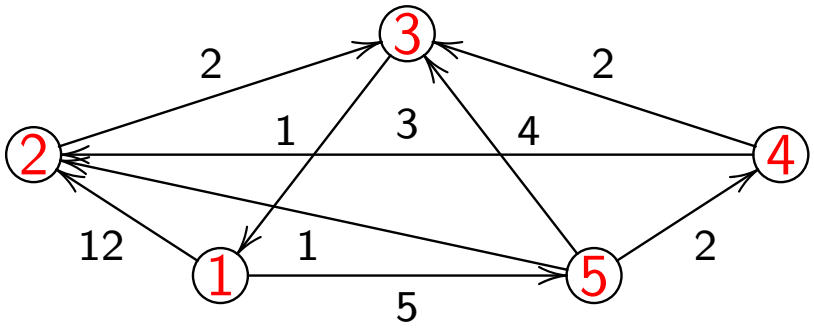
Exemple



PCC pour aller de 1 à 3 :

- On a $T = [\![0; 6; 8; 7; 5]\!]$ et $P = [\![1; 5; 2; 5; 1]\!]$.
- prédecesseur de 3 : 2
- prédecesseur de 2 : 5
- prédecesseur de 5 : 1

Exemple



PCC pour aller de 1 à 3 :

- On a $T = [\![0; 6; 8; 7; 5]\!]$ et $P = [\![1; 5; 2; 5; 1]\!]$.
- prédecesseur de 3 : 2
- prédecesseur de 2 : 5
- prédecesseur de 5 : 1
- donc : 1523 avec un coût de 8

Rappel algorithme

```
1  F:={e} /*sommets en cours de traitement (=verts)*/
2  E:=∅ /*sommets traités = sommets rouges*/
3  D:= tableau des distances minimales, avec d_e = 0 et autres d_k = ∞
4  tant que F ≠ ∅ faire
5        choisir k ∈ F avec d_k minimal
6        F:=F \ {k}
7        E:=E ∪ {k} /* k devient rouge*/
8        pour tout voisin v de k non rouge faire
9              si v ∉ F : /*si v est bleu*/
10                F:=F ∪ {v} /*v devient vert*/
11             fin_si
12             si d_k + w(k → v) < d_v :
13                   d_v:=d_k + w(k → v) /*maj tab. des distances*/
14             fin_si
15       fin_faire
16 fin_faire
```

Correction

Pour un graphe G d'ensemble de sommets S, notons D le tableau des distances et $\delta(e,u)$ la longueur d'un PCC de e à u, $w(a,b)$ le poids de l'arc $a \to b$.

Théorème

A la fin de l'algorithme de Dijkstra, on a $d_u = \delta(e,u)$ pour tout $u \in S$.

`Notations`

- On note $F^k; E^k; d_u^k$ les valeurs de F, E, d_u à la fin de l'étape k.

Correction

Pour un graphe G d'ensemble de sommets S, notons D le tableau des distances et $\delta(e,u)$ la longueur d'un PCC de e à u, $w(a,b)$ le poids de l'arc $a \to b$.

Théorème

A la fin de l'algorithme de Dijkstra, on a $d_u = \delta(e,u)$ pour tout $u \in S$.

`Notations`

- On note $F^k; E^k; d_u^k$ les valeurs de F, E, d_u à la fin de l'étape k.
- F^k contient les sommets verts à la fin du passage k, E^k les sommets rouges et $S \setminus (F^k \cup E^k)$ les sommets bleus.

Correction

Pour un graphe G d'ensemble de sommets S, notons D le tableau des distances et $\delta(e,u)$ la longueur d'un PCC de e à u, $w(a,b)$ le poids de l'arc $a \to b$.

Théorème

A la fin de l'algorithme de Dijkstra, on a $d_u = \delta(e,u)$ pour tout $u \in S$.

`Notations`

- On note $F^k; E^k; d_u^k$ les valeurs de F, E, d_u à la fin de l'étape k.
- F^k contient les sommets verts à la fin du passage k, E^k les sommets rouges et $S \setminus (F^k \cup E^k)$ les sommets bleus.
- Un sommet u est bleu à la fin de l'étape k si et seulement si $d_u = +\infty$ si et seulement si u n'est voisin d'aucun sommet rouge de l'étape k.

Correction

Pour un graphe G d'ensemble de sommets S, notons D le tableau des distances et $\delta(e,u)$ la longueur d'un PCC de e à u, $w(a,b)$ le poids de l'arc $a \to b$.

Théorème

A la fin de l'algorithme de Dijkstra, on a $d_u = \delta(e,u)$ pour tout $u \in S$.

`Notations`

- On note $F^k; E^k; d_u^k$ les valeurs de F, E, d_u à la fin de l'étape k.
- F^k contient les sommets verts à la fin du passage k, E^k les sommets rouges et $S \setminus (F^k \cup E^k)$ les sommets bleus.
- Un sommet u est bleu à la fin de l'étape k si et seulement si $d_u = +\infty$ si et seulement si u n'est voisin d'aucun sommet rouge de l'étape k.
- Pour $u \in S$, $d_u < +\infty$ documente un chemin de e à u n'empruntant que des sommets verts ou rouges.

Correction : Invariant en deux points

Rappel À la fin du parcours, les sommets accessibles depuis e sont dans E

Correction : Invariant en deux points

Rappel À la fin du parcours, les sommets accessibles depuis e sont dans E

Invariant À la fin de chaque passage dans la boucle :

Correction : Invariant en deux points

Rappel À la fin du parcours, les sommets accessibles depuis e sont dans E

Invariant À la fin de chaque passage dans la boucle :

1. Sommets rouges : $\forall u \in E^k$, $d_u^k = \delta(e, u)$ (point 1).

Correction : Invariant en deux points

Rappel À la fin du parcours, les sommets accessibles depuis e sont dans E

Invariant À la fin de chaque passage dans la boucle :

1. Sommets rouges : $\forall u \in E^k$, $d_u^k = \delta(e, u)$ (point 1).
2. Sommet v vert ou bleu différents de e : $d_v^k = \min(\{d_u^k + w(u, v) \mid u \in E^k\})$. Ce qui revient, par le point 1, à $d_v^k = \min(\{\delta(e, u) + w(u, v) \mid u \in E^k\})$ (point 2).

Correction : Invariant en deux points

Rappel À la fin du parcours, les sommets accessibles depuis e sont dans E

Invariant À la fin de chaque passage dans la boucle :

1. Sommets rouges : $\forall u \in E^k$, $d_u^k = \delta(e,u)$ (point 1).
2. Sommet v vert ou bleu différents de e : $d_v^k = \min(\{d_u^k + w(u,v) \mid u \in E^k\})$. Ce qui revient, par le point 1, à $d_v^k = \min(\{\delta(e,u) + w(u,v) \mid u \in E^k\})$ (point 2).

Remarque La distance d_v^k est donc le poids minimal d'un chemin vert-rouge de e à v dont l'avant dernier sommet est rouge.

Pour un sommet bleu v à l'étape k, les $w(u,v)$ pour u rouge étant infinis (car v n'est pas voisin d'un rouge), on retrouve que $d_v^k = \infty$

Correction : Invariant en deux points et cas de base

- Au tour 0 (avant la boucle) :

Correction : Invariant en deux points et cas de base

- Au tour 0 (avant la boucle) :
  - L'ensemble des sommets rouge est vide, donc pour un sommet $v \neq e$ le minimum des sommes $\delta(u, v) + w(u, v)$ pour les u rouges est infini. Or $d_v^0 = +\infty$: Point 2 OK.

Correction : Invariant en deux points et cas de base

- Au tour 0 (avant la boucle) :
  - L'ensemble des sommets rouge est vide, donc pour un sommet $v \neq e$ le minimum des sommes $\delta(u, v) + w(u, v)$ pour les u rouges est infini. Or $d_v^0 = +\infty$: Point 2 OK.
  - Le point 1 est vrai car il n'y a pas de sommet rouge.

Correction : Invariant en deux points et cas de base

- Au tour 0 (avant la boucle) :
  - L'ensemble des sommets rouge est vide, donc pour un sommet $v \neq e$ le minimum des sommes $\delta(u,v) + w(u,v)$ pour les u rouges est infini. Or $d_v^0 = +\infty$: Point 2 OK.
  - Le point 1 est vrai car il n'y a pas de sommet rouge.
- A la fin du 1er tour de boucle e est rouge, $d_e^1 = 0 = \delta(e,e)$ (point 1 OK) et $d_v^1 = w(e,v)$ pour tout $v \neq e$ donc $d_v^1 = \min(\{\delta(e,u) + w(e,v) \mid u \in E^1\})$ (point 2 OK) puisque seul e est rouge.

Cas des sommets rouges (Point 1 de l'invariant)

Soit u un sommet rouge à la fin de l'étape $k+1$ ($k \geq 1$).

- On a $d_u^k = d_u^{k+1}$ (pas de changement dans le tableau des distances).

Cas des sommets rouges (Point 1 de l'invariant)

Soit u un sommet rouge à la fin de l'étape $k+1$ ($k \geq 1$).

- On a $d_u^k = d_u^{k+1}$ (pas de changement dans le tableau des distances).
- Comme u est rouge, il est accessible, donc d_u^{k+1} est fini, donc le chemin qui le documente n'emprunte que des sommets verts ou rouges.

Cas des sommets rouges (Point 1 de l'invariant)

Soit u un sommet rouge à la fin de l'étape $k+1$ ($k \geq 1$).

- On a $d_u^k = d_u^{k+1}$ (pas de changement dans le tableau des distances).
- Comme u est rouge, il est accessible, donc d_u^{k+1} est fini, donc le chemin qui le documente n'emprunte que des sommets verts ou rouges.
- Si $u \notin E^{k+1} \setminus E^k$, c'est à dire si u n'est pas le sommet qu'on vient de rendre rouge, alors $u \in E^k$. Et par HR $d_u^k = \delta(e,u)$. Comme $d_u^{k+1} = d_u^k$: Point 1 OK.

Cas des sommets rouges Hérédité (Point 1 de l'invariant)

- Si u entre dans E^{k+1} alors au début du passage $k+1$, u est vert ($u \in F^k$) et $d_u^k \leq d_s^k$ pour tout sommet vert s (car u est choisi).

Cas des sommets rouges Hérédité (Point 1 de l'invariant)

- Si u entre dans E^{k+1} alors au début du passage $k+1$, u est vert ($u \in F^k$) et $d_u^k \leq d_s^k$ pour tout sommet vert s (car u est choisi).
  - Soit $e \rightsquigarrow u$ un PCC et v son premier sommet non rouge ($u = v$ possible). Les sous chemins $e \rightsquigarrow v$ et $v \rightsquigarrow u$ sont des PCC par principe d'optimalité. Comme les poids des arcs sont positifs (condition Dijkstra), il vient que $\delta(e,u) \geq \delta(e,v)$.

Cas des sommets rouges Hérédité (Point 1 de l'invariant)

- Si u entre dans E^{k+1} alors au début du passage $k+1$, u est vert ($u \in F^k$) et $d_u^k \leq d_s^k$ pour tout sommet vert s (car u est choisi).
  - Soit $e \rightsquigarrow u$ un PCC et v son premier sommet non rouge ($u = v$ possible). Les sous chemins $e \rightsquigarrow v$ et $v \rightsquigarrow u$ sont des PCC par principe d'optimalité. Comme les poids des arcs sont positifs (condition Dijkstra), il vient que $\delta(e,u) \geq \delta(e,v)$.
  - Le prédecesseur r de v dans $e \rightsquigarrow v$ est rouge et différent de u donc $d_r^k = \delta(e,r)$ par HR.

Cas des sommets rouges Hérédité (Point 1 de l'invariant)

- Si u entre dans E^{k+1} alors au début du passage $k+1$, u est vert ($u \in F^k$) et $d_u^k \leq d_s^k$ pour tout sommet vert s (car u est choisi).
  - Soit $e \rightsquigarrow u$ un PCC et v son premier sommet non rouge ($u = v$ possible). Les sous chemins $e \rightsquigarrow v$ et $v \rightsquigarrow u$ sont des PCC par principe d'optimalité. Comme les poids des arcs sont positifs (condition Dijkstra), il vient que $\delta(e,u) \geq \delta(e,v)$.
  - Le prédecesseur r de v dans $e \rightsquigarrow v$ est rouge et différent de u donc $d_r^k = \delta(e,r)$ par HR.
  - Comme v n'est pas rouge, il vérifie le point 2 donc

$$\begin{array}{rcl} d_v^k & = & \min\left(\left\{\delta(e,x)+w(x,v) \mid x \in E^k\right\}\right. \text{ par HR.(2)} \\ & \underbrace{=}_{e \rightsquigarrow r \to v \text{ PCC}} & \delta(e,r)+w(r,v) = \delta(e,v) \end{array}$$

Ainsi $d_v^k = \delta(e,v)$.

Cas des sommets rouges Hérédité (Point 1 de l'invariant)

- Si u entre dans E^{k+1} alors au début du passage $k+1$, u est vert ($u \in F^k$) et $d_u^k \leq d_s^k$ pour tout sommet vert s (car u est choisi).
  - Soit $e \rightsquigarrow u$ un PCC et v son premier sommet non rouge ($u = v$ possible). Les sous chemins $e \rightsquigarrow v$ et $v \rightsquigarrow u$ sont des PCC par principe d'optimalité. Comme les poids des arcs sont positifs (condition Dijkstra), il vient que $\delta(e,u) \geq \delta(e,v)$.
  - Le prédecesseur r de v dans $e \rightsquigarrow v$ est rouge et différent de u donc $d_r^k = \delta(e,r)$ par HR.
  - Comme v n'est pas rouge, il vérifie le point 2 donc

$$\begin{array}{rcl} d_v^k & = & \min\left(\left\{\delta(e,x)+w(x,v) \mid x \in E^k\right\}\right. \text{ par HR.(2)} \\ & \underbrace{=}_{e \rightsquigarrow r \to v \text{ PCC}} & \delta(e,r)+w(r,v) = \delta(e,v) \end{array}$$

    Ainsi $d_v^k = \delta(e,v)$.
  - On a $\delta(e,u) \geq \delta(e,v)$. Et $d_v^k \geq d_u^k$ car u est choisi. Or, d_u^k est le poids d'une chemin de e à u donc plus grand que $\delta(e,u)$. Alors

$$\delta(e,u) \geq \delta(e,v) = d_v^k \geq d_u^k \geq \delta(e,u) \text{ Point 1 OK}$$

Cas des sommets verts ou bleus : Hérédité Point 2

Sommets non voisins du sommet u qui devient rouge

Soit v un sommet vert ou bleu à la fin de l'étape $k+1$ non voisin de l'élément u qui entre dans E^{k+1}.

- Alors $d_v^k = d_v^{k+1}$ car $d_u^k + w(u,v) = +\infty$.

Cas des sommets verts ou bleus : Hérédité Point 2

Sommets non voisins du sommet u qui devient rouge

Soit v un sommet vert ou bleu à la fin de l'étape $k+1$ non voisin de l'élément u qui entre dans E^{k+1}.

- Alors $d_v^k = d_v^{k+1}$ car $d_u^k + w(u,v) = +\infty$.
- On a :

$$\begin{aligned}
d_v^{k+1} &= d_v^k \\
&= \min\left(\left\{\delta(e,x) + w(x,v) \mid x \in E^k\right\}\right) \text{ par HR} \\
&= \min\left(\left\{\delta(x,k) + w(x,v) \mid x \in E^k\right\} \cup \left\{\underbrace{\delta(e,u) + w(u,v)}_{=+\infty}\right\}\right) \\
&= \min\left(\left\{\delta(e,x) + w(x,v) \mid x \in E^{k+1}\right\}\right) \text{ : Point 2 OK}
\end{aligned}$$

Cas des sommets verts ou bleus : Hérédité Point 2

Sommets voisins du sommet u qui devient rouge

Soit v vert ou bleu à la fin de l'étape $k+1$ et voisin de l'élément rouge u entrant dans E^{k+1}.

- Le sommet v est vert ou bleu à l'étape k mais vert à l'étape $k+1$.

Cas des sommets verts ou bleus : Hérédité Point 2

Sommets voisins du sommet u qui devient rouge

Soit v vert ou bleu à la fin de l'étape $k+1$ et voisin de l'élément rouge u entrant dans E^{k+1}.

- Le sommet v est vert ou bleu à l'étape k mais vert à l'étape $k+1$.
  - $d_v^k = \min\left(\left\{\delta(e,x) + w(x,v) \mid x \in \underbrace{E^k}_{=E^{k+1}\setminus\{u\}}\right\}\right)$ par HR.2.

Cas des sommets verts ou bleus : Hérédité Point 2

Sommets voisins du sommet u qui devient rouge

Soit v vert ou bleu à la fin de l'étape $k+1$ et voisin de l'élément rouge u entrant dans E^{k+1}.

- Le sommet v est vert ou bleu à l'étape k mais vert à l'étape $k+1$.
  - $d_v^k = \min\left(\left\{\delta(e,x) + w(x,v) \mid x \in \underbrace{E^k}_{=E^{k+1}\setminus\{u\}}\right\}\right)$ par HR.2.
  - $d_v^{k+1} = \min(d_v^k, \underbrace{d_u^k}_{=\delta(e,u) \text{ par HR.1}} + w(u,v))$ par algo Dijkstra

Cas des sommets verts ou bleus : Hérédité Point 2

Sommets voisins du sommet u qui devient rouge

Soit v vert ou bleu à la fin de l'étape $k+1$ et voisin de l'élément rouge u entrant dans E^{k+1}.

- Le sommet v est vert ou bleu à l'étape k mais vert à l'étape $k+1$.
  - $d_v^k = \min\left(\left\{\delta(e,x) + w(x,v) \mid x \in \underbrace{E^k}_{=E^{k+1}\setminus\{u\}}\right\}\right)$ par HR.2.
  - $d_v^{k+1} = \min(d_v^k, \underbrace{d_u^k}_{=\delta(e,u) \text{ par HR.1}} + w(u,v))$ par algo Dijkstra
- On a donc encore

$$d_v^{k+1} = \min\left(\left\{\delta(e,x) + w(x,v) \mid x \in \underbrace{E^{k+1}}_{=E^k\cup\{u\}}\right\}\right)$$

Cas des sommets verts ou bleus : Hérédité Point 2

Sommets voisins du sommet u qui devient rouge

Soit v vert ou bleu à la fin de l'étape $k+1$ et voisin de l'élément rouge u entrant dans E^{k+1}.

- Le sommet v est vert ou bleu à l'étape k mais vert à l'étape $k+1$.
  - $d_v^k = \min\left(\left\{\delta(e,x) + w(x,v) \mid x \in \underbrace{E^k}_{=E^{k+1}\setminus\{u\}}\right\}\right)$ par HR.2.
  - $d_v^{k+1} = \min(d_v^k, \underbrace{d_u^k}_{=\delta(e,u) \text{ par HR.1}} + w(u,v))$ par algo Dijkstra
- On a donc encore

$$d_v^{k+1} = \min\left(\left\{\delta(e,x) + w(x,v) \mid x \in \underbrace{E^{k+1}}_{=E^k\cup\{u\}}\right\}\right)$$

- Point 2 OK dans ce cas.

Complexité pour une source e

Force brute pour un graphe de n sommets et p arcs

```
1  F:={e} ; E:=∅ ; D:=créer tableau des distances minimales;/*O(n)*/
2  tant que F ≠ ∅ faire
3        choisir k ∈ F avec d_k minimal /*explorer D en O(n)*/
4        F:=F \ {k}; E:=E ∪ {k} : /*MAJ en O(1) si E,F tableaux*/
5        pour tout voisin r de k non rouge faire
6              si r ∉ F : /* cas r bleu */
7                    F:=F ∪ {r} /*O(1)*/
8              si d_k + w(k → r) < d_r :
9                          d_r:=d_k + w(k → r) /*maj tab. des distances O(1)*/
```

- Au plus n transferts de F vers E (ligne $L4$). Pour chacun recherche (ligne $L3$) du plus petit élément vert dans le tableau des distances : $O(n)$

Complexité pour une source e

Force brute pour un graphe de n sommets et p arcs

```
1  F:={e} ; E:=∅ ; D:=créer tableau des distances minimales;/*O(n)*/
2  tant que F ≠ ∅ faire
3        choisir k ∈ F avec d_k minimal /*explorer D en O(n)*/
4        F:=F \ {k}; E:=E ∪ {k} : /*MAJ en O(1) si E,F tableaux*/
5        pour tout voisin r de k non rouge faire
6              si r ∉ F : /* cas r bleu */
7                    F:=F ∪ {r} /*O(1)*/
8              si d_k + w(k → r) < d_r :
9                    d_r:=d_k + w(k → r) /*maj tab. des distances O(1)*/
```

- Au plus n transferts de F vers E (ligne $L4$). Pour chacun recherche (ligne $L3$) du plus petit élément vert dans le tableau des distances : $O(n)$
- ($L5$ à $L9$) Coût des vérifications et mises à jour pour un sommet k : $O(\deg^+ k)$. Au total, complexité proportionnelle à :

$$n + \sum_{k=0}^{n-1}(n + \deg^+ k) = n + n^2 + p = O(n^2) \text{ (rappel : } p = O(n^2)).$$

Complexité pour une source e

Avec file de priorité pour un graphe de n sommets et p arcs

- Puisqu'on gère un ensemble F des sommets verts et un tableau D des distances à la source, on peut les fusionner en une seule file de priorité T d'éléments $(s, d(e, s))$.

Complexité pour une source e

Avec file de priorité pour un graphe de n sommets et p arcs

- Puisqu'on gère un ensemble F des sommets verts et un tableau D des distances à la source, on peut les fusionner en une seule file de priorité T d'éléments $(s, d(e,s))$.
- On implante les files de priorité comme des tas. On considère donc un tas-min (fils plus grands que père). Création par descente en $O(n)$. On gère en interne un tableau des positions dans le tas pour obtenir en $O(1)$ la position d'un sommet dans T en vue d'une MAJ.

```
1  E:=emptyset ; /*sommets rouges*/
2  T:= tas-min des couples (sommet, distance depuis e) ; /*O(n)*/
3  tant que T ≠ ∅ :
4        retirer la racine (k, d(e,k)) du tas T /*O(ln n)*/
5        E:=E ∪ {k}
6        pour tout voisin r de k non traité :
7              /*nb passages = deg+ k*/
8              si d(e,k) + w(k → r) < d(e,r): /O(1)/
9                    MAJ T avec d(e,r):=d(e,k) + w(k → r) /*O(ln n)*/
10                   /*utiliser le tab. des pos. dans le tas*/
```

Complexité pour une source e

Avec file de priorité pour un graphe de n sommets et p arcs

```
1  E:=emptyset ; /*sommets rouges*/
2  T:= tas-min des couples (sommet, distance depuis e) ; /*O(n)*/
3  tant que T ≠ ∅ :
4        retirer la racine (k, d(e,k)) du tas T /*O(ln n)*/
5        E:=E ∪ {k}
6        pour tout voisin r de k non traité :
7              si d(e,k) + w(k → r) < d(e,r): /*O(1)*/
8                    MAJ T avec d(e,r):=d(e,k) + w(k → r) /*O(ln(n))*/
```

- Complexité de chaque accès/maj dans la file majoré en $O(\ln n)$. La file de priorité T est de taille au plus n.

Complexité pour une source e

Avec file de priorité pour un graphe de n sommets et p arcs

```
1  E:=emptyset ; /*sommets rouges*/
2  T:= tas-min des couples (sommet, distance depuis e) ; /*O(n)*/
3  tant que T ≠ ∅ :
4        retirer la racine (k, d(e,k)) du tas T /*O(ln n)*/
5        E:=E ∪ {k}
6        pour tout voisin r de k non traité :
7              si d(e,k) + w(k → r) < d(e,r): /*O(1)*/
8                    MAJ T avec d(e,r):=d(e,k) + w(k → r) /*O(ln(n))*/
```

- Complexité de chaque accès/maj dans la file majoré en $O(\ln n)$. La file de priorité T est de taille au plus n.
- Par passage dans boucle **while** : choix puis suppression du sommet k le plus prioritaire : $O(\ln n)$. Pour ses $\deg^+ k$ voisins, au plus $\deg^+ k$ maj de clés. Donc coût pour k en $O((1+\deg^+ k)\ln n)$. Coût total :

$$n + \sum_{e=0}^{n-1} (\deg^+ k + 1) \ln n = n + n \ln n + \ln n \sum_{e=0}^{n-1} \deg^+ k \leq n \ln n + p \ln n$$

Complexité pour une source e

Avec file de priorité pour un graphe de n sommets et p arcs

- Complexité en $O((n+p)\ln n)$

Complexité pour une source e

Avec file de priorité pour un graphe de n sommets et p arcs

- Complexité en $O((n+p)\ln n)$
- Si $p \ln n = O(n^2)$, c'est à dire si $p = O(\frac{n^2}{\ln n})$ la complexité avec file de priorités est au moins aussi bonne qu'en force brute.

Complexité pour une source e

Avec file de priorité pour un graphe de n sommets et p arcs

- Complexité en $O((n+p)\ln n)$
- Si $p \ln n = O(n^2)$, c'est à dire si $p = O(\frac{n^2}{\ln n})$ la complexité avec file de priorités est au moins aussi bonne qu'en force brute.
- Si le graphe est creux (peu d'arêtes), $p = O(n)$ et donc la complexité avec file de priorité est $O(n \ln n)$.

Complexité pour une source e

Avec file de priorité pour un graphe de n sommets et p arcs

- Complexité en $O((n+p)\ln n)$
- Si $p \ln n = O(n^2)$, c'est à dire si $p = O(\frac{n^2}{\ln n})$ la complexité avec file de priorités est au moins aussi bonne qu'en force brute.
- Si le graphe est creux (peu d'arêtes), $p = O(n)$ et donc la complexité avec file de priorité est $O(n \ln n)$.
- Mais si le graphe est dense (par exemple complet) il y a un nombre d'arc ou d'arêtes en $p = O(n^2)$ et donc la complexité est en $O(n^2 \ln n)$.
  Dans ce cas, l'implémentation par file de priorité n'est pas intéressante.