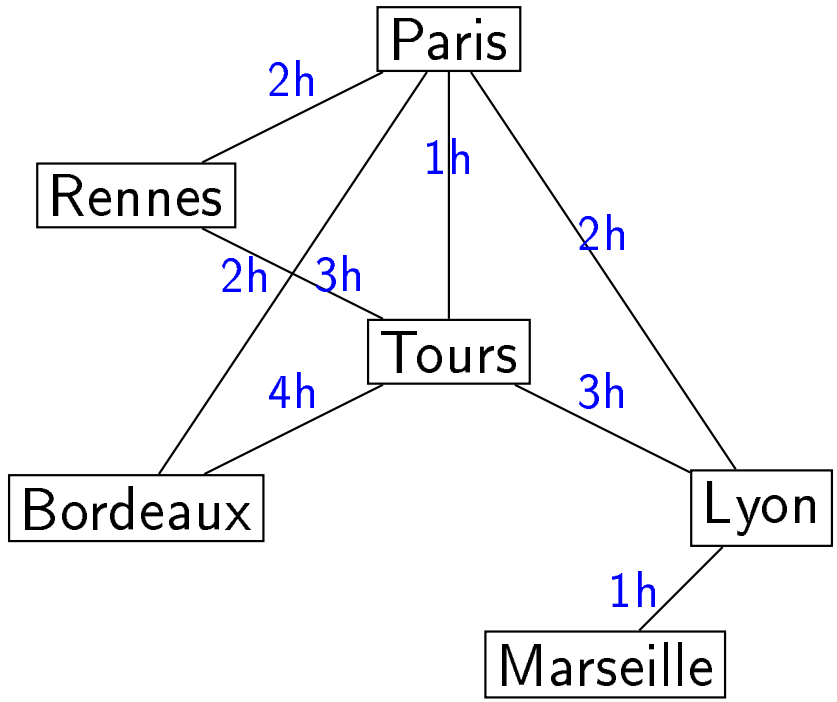
Parcours de graphes

Anne Bouillard

20 octobre 2016

Modélisation par des graphes

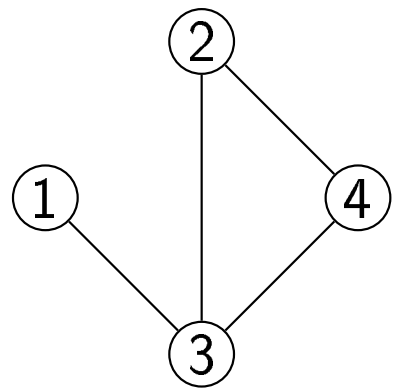
1. Comment aller d'un endroit à un autre par le plus court chemin ?



Modélisation par des graphes

1. Comment aller d'un endroit à un autre par le plus court chemin ?
2. Gestion de salles de classe.

Anglais	jeudi	8h-10h
Mathématiques	jeudi	10h-12h
Physique	jeudi	9h-12h
Latin	jeudi	11h-12h



Modélisation par des graphes

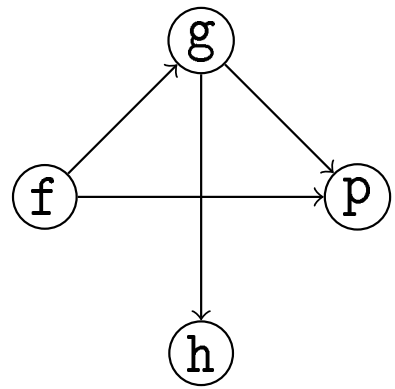
1. Comment aller d'un endroit à un autre par le plus court chemin ?
2. Gestion de salles de classe.
3. Dépendances entre fonctions dans un programme.

```
int f (int z)=
      {... x=g(y);
      y=p(x);...}

int g (int y)=
      {... z=h(x);...;
      z= p(z)...}

int p = ...

int h = ...
```

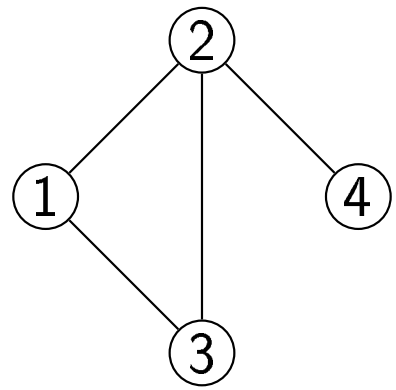


Définition

Définition (Graphe non-orienté)

$G = (S, A)$ est un graphe non orienté fini si

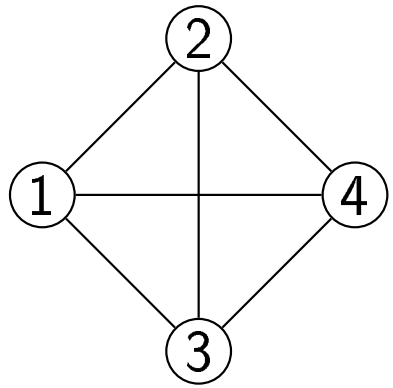
- *S est un ensemble fini de sommets ;*
- *$A \subseteq \mathcal{P}_2(S)$ est un ensemble fini d'arêtes*



Voisins : v voisin de u si $uv \in A$. On pose $V(u) = \{v \mid \{u, v\} \in A\}$.
Chemin : suite finie de sommets $u_1, \ldots, u_n$ tels que $\{u_i, u_{i+1}\} \in A \ \forall i \in \{1, \ldots, n-1\}$.
Cycle : un chemin $u_1, \ldots, u_n$ tel que $u_1 = u_n$.
Graphe connexe : $\forall u, v \in S$, il existe un chemin de u à v.

Graphes particuliers

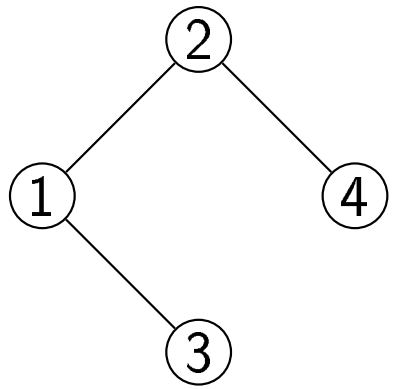
- Graphe complet : $G = (S, \mathcal{P}_2(S))$



- Arbre : graphe connexe sans cycle

Graphes particuliers

- Graphe complet : $G = (S, \mathcal{P}_2(S))$
- Arbre : graphe connexe sans cycle

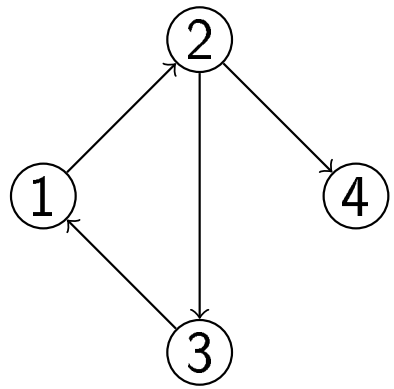


Graphes orientés

Définition (Graphe orienté)

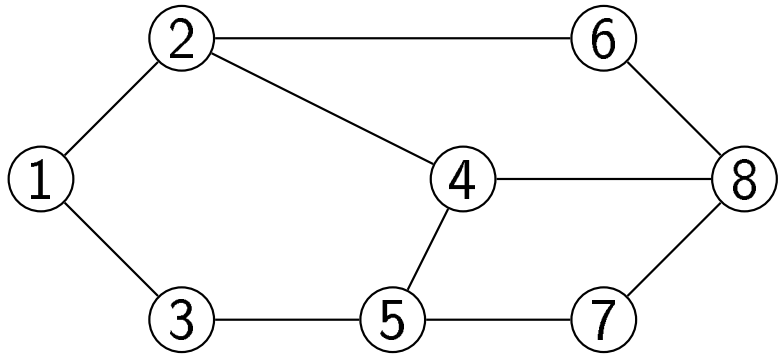
$G = (S, A)$ est un graphe orienté fini si

- *S est un ensemble fini de sommets ;*
- *$A \subseteq S^2$ est un ensemble fini d'arcs.*



Adjacence : v adjacent à u si $(u, v) \in A$.
Graphe fortement connexe : $\forall u, v \in S$, il existe un chemin de u à v.

Représentations matricielles

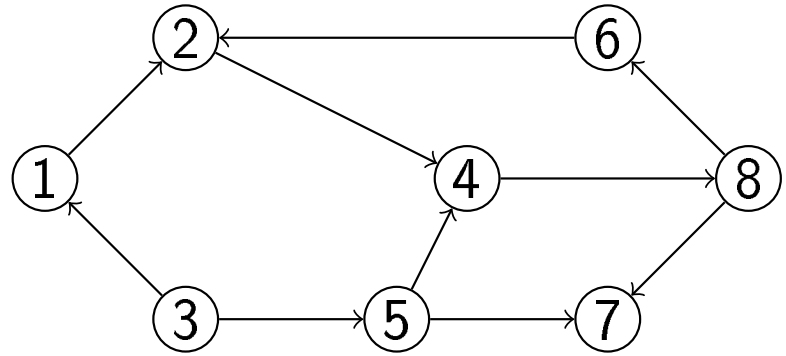


Matrice d'adjacence :

$$A_{ij} = \begin{cases} 1 \text{ si } (i,j) \in A \text{ ou } \{i,j\} \in A \\ 0 \text{ sinon.} \end{cases}$$

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Représentations matricielles



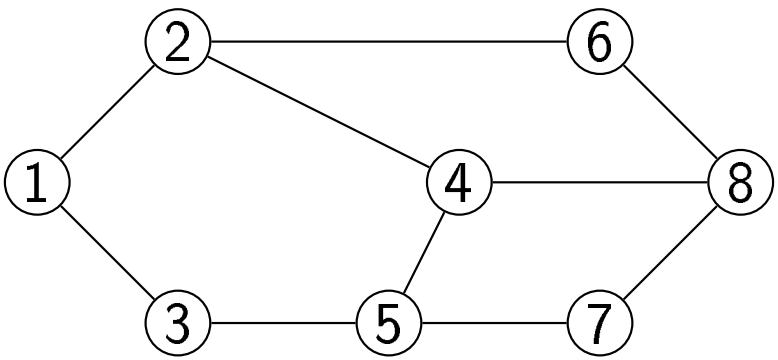
Matrice d'adjacence :

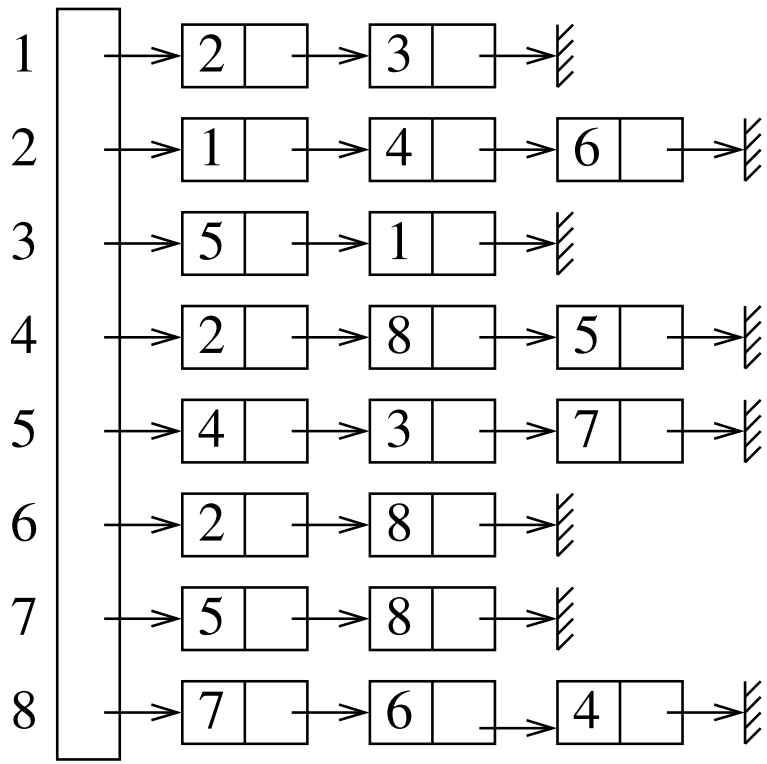
$$A_{ij} = \begin{cases} 1 \text{ si } (i,j) \in A \text{ ou } \{i,j\} \in A \\ 0 \text{ sinon.} \end{cases}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Représentations par des listes chaînées

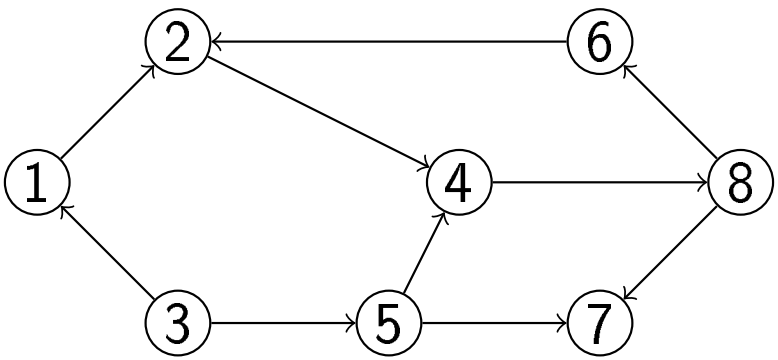
Liste des successeurs : un tableau de taille $|S|$ de listes chaînées.

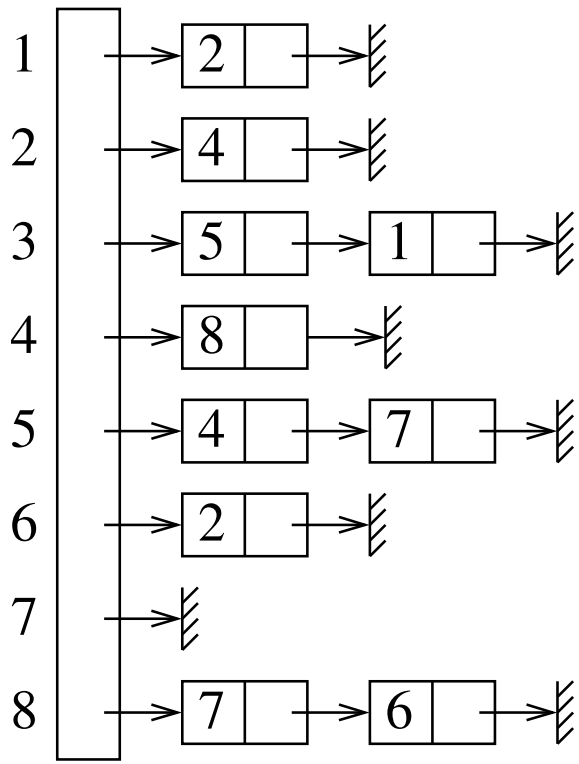




Représentations par des listes chaînées

Liste des successeurs : un tableau de taille $|S|$ de listes chaînées.





Objectif

Parcourir/découvrir tous les sommets d'un graphe.
Deux principaux types de parcours :

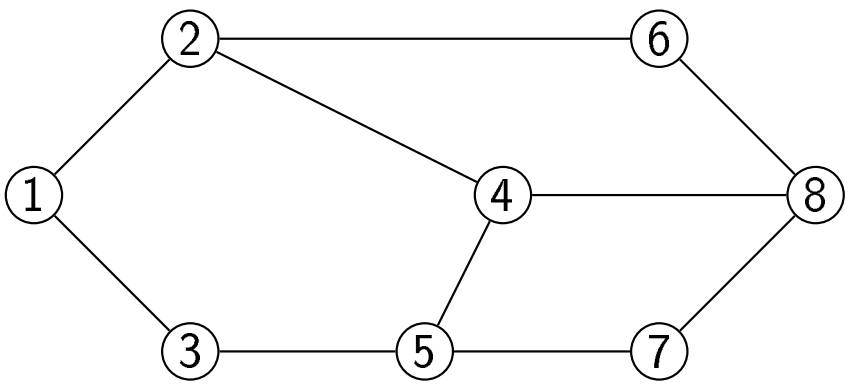
- parcours en largeur
- parcours en profondeur

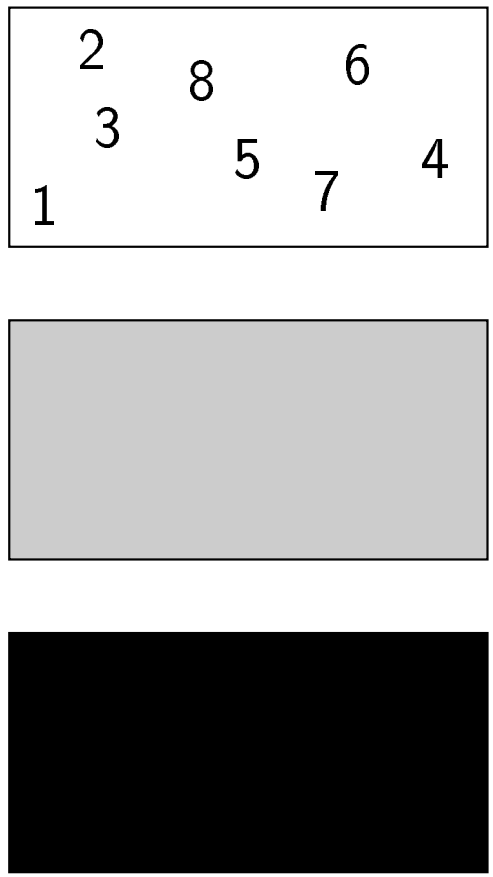
On construit en même temps un « arbre couvrant » (sous-graphe qui est un arbre et qui contient tous les sommets) qui contient les arêtes qui ont permis de découvrir les sommets.
Les algorithmes présentés sont valables pour les graphes orientés et les graphes non orientés.

Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

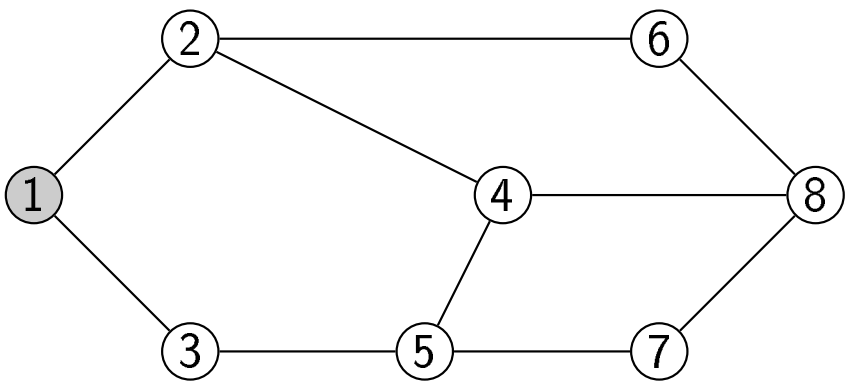


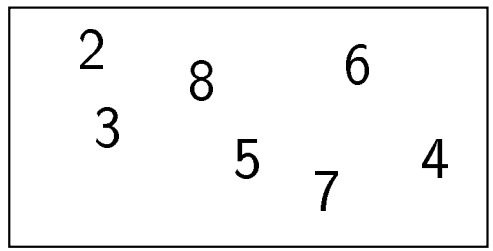


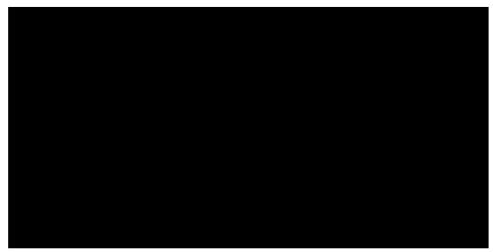
Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;



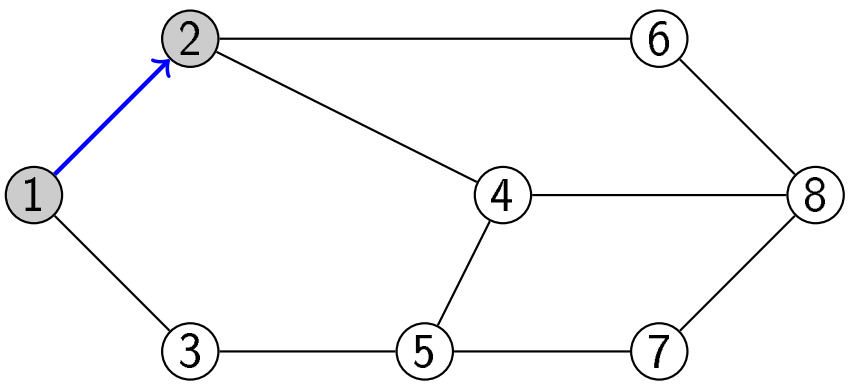


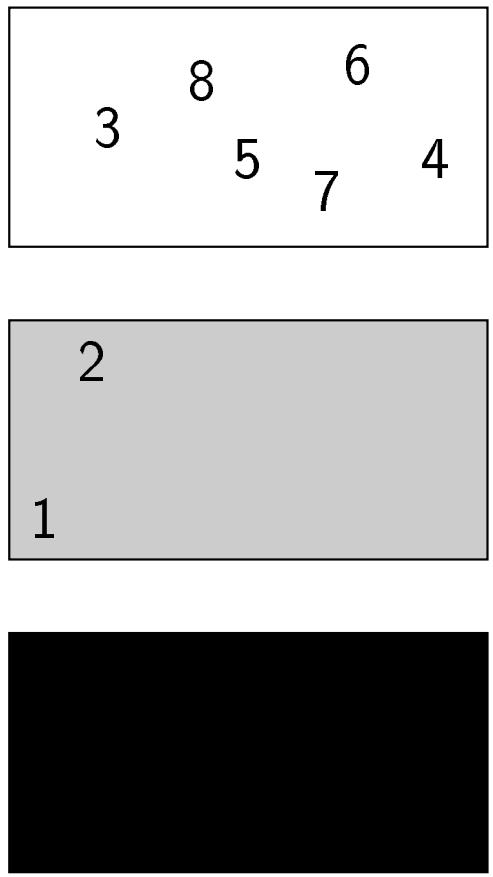


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

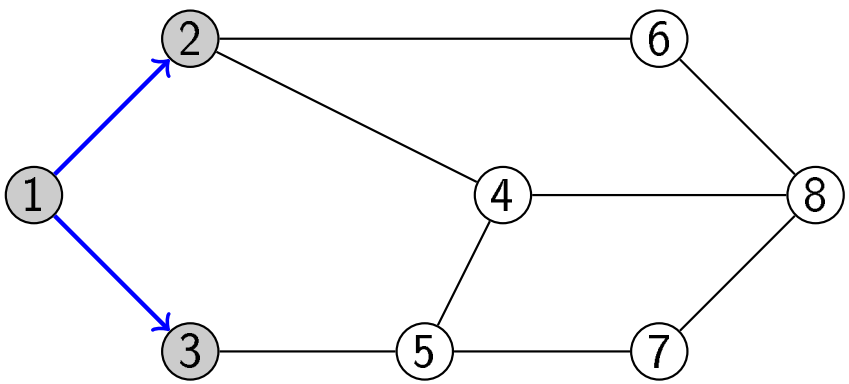


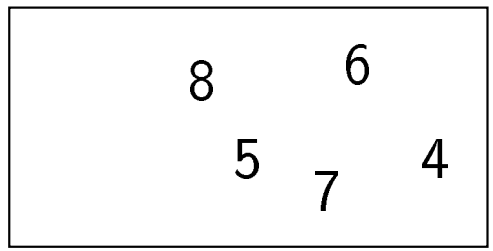


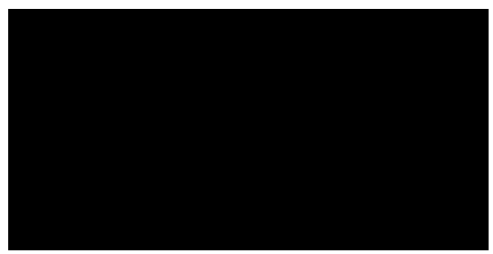
Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;



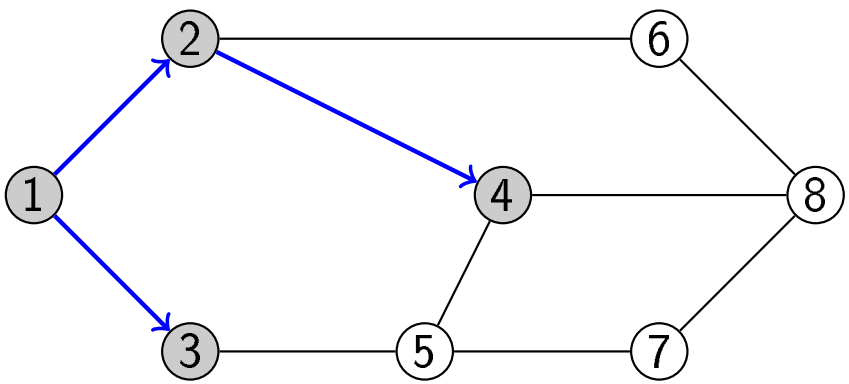


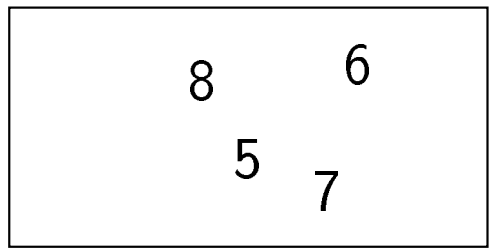


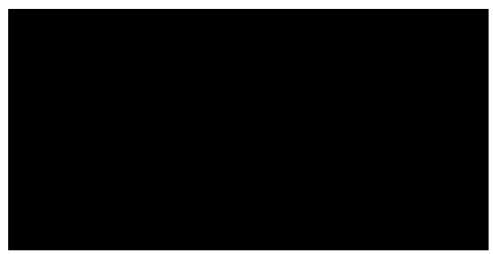
Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;



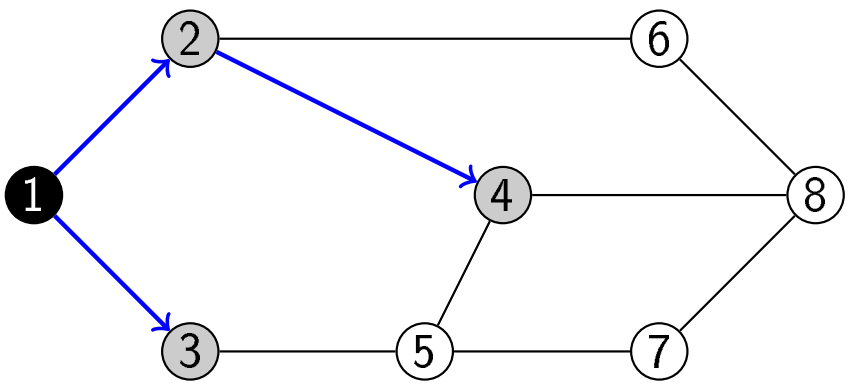


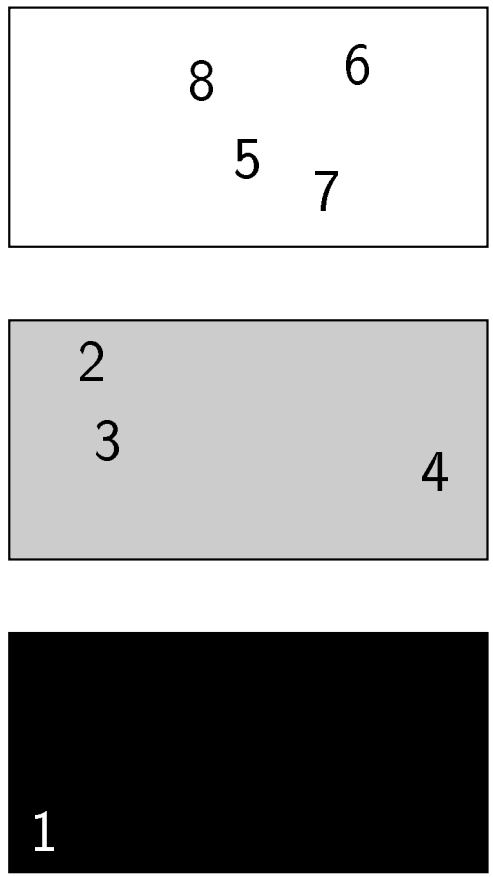


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

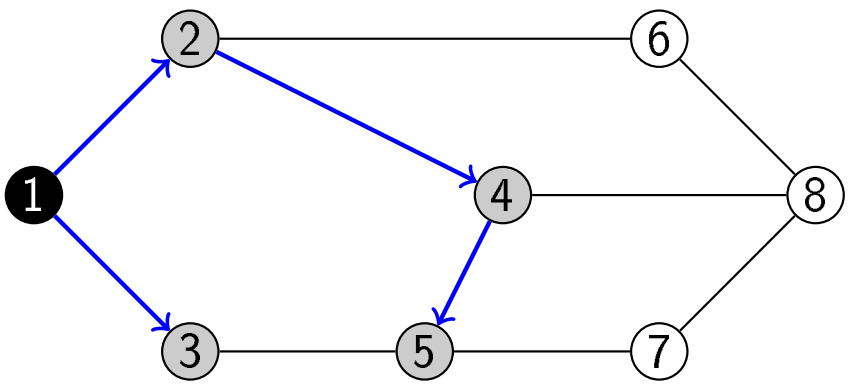


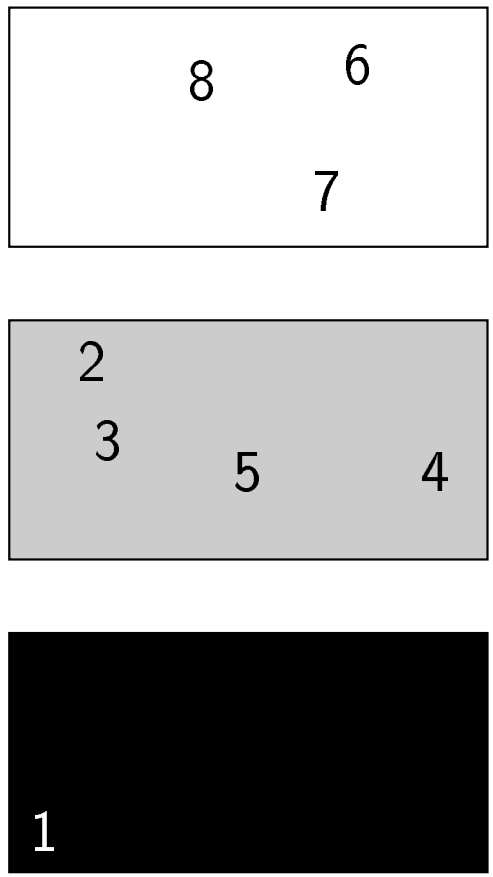


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

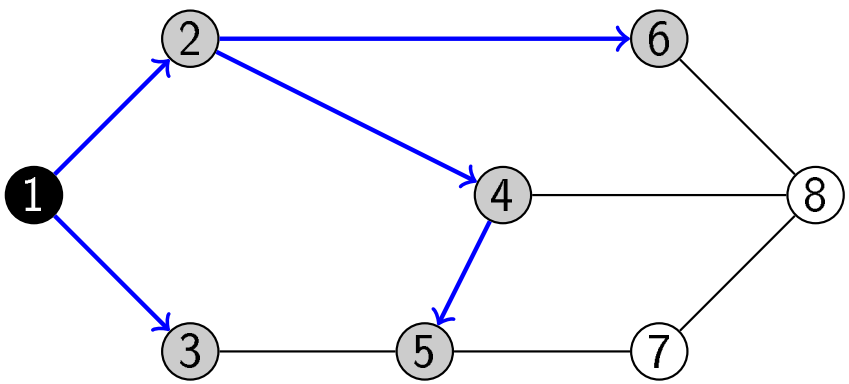


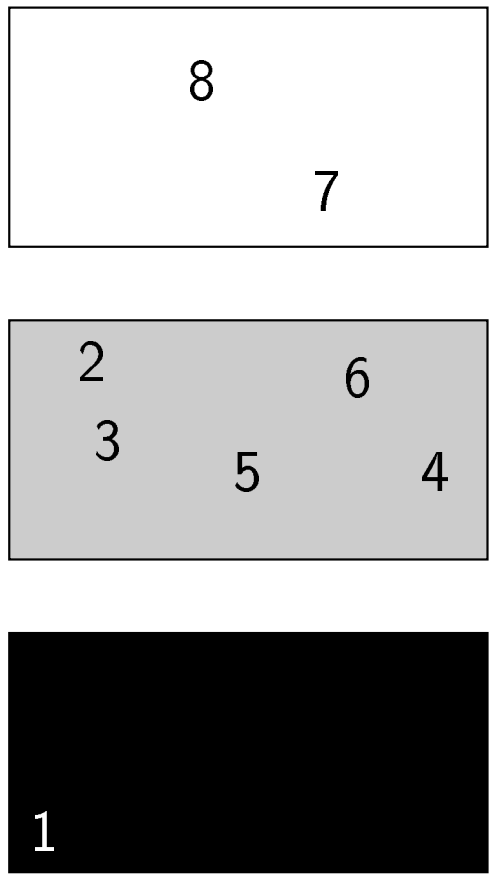


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

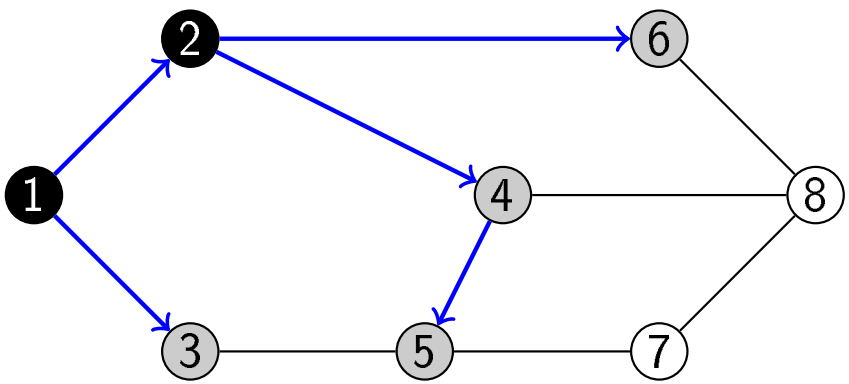


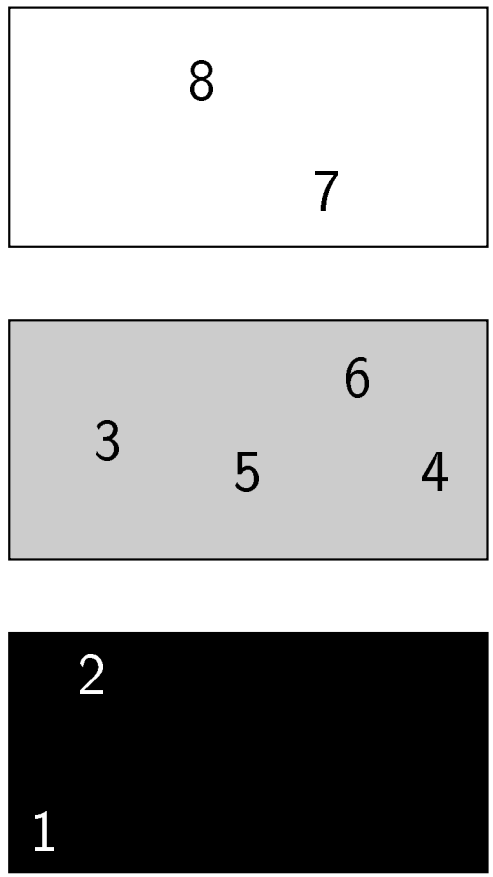


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

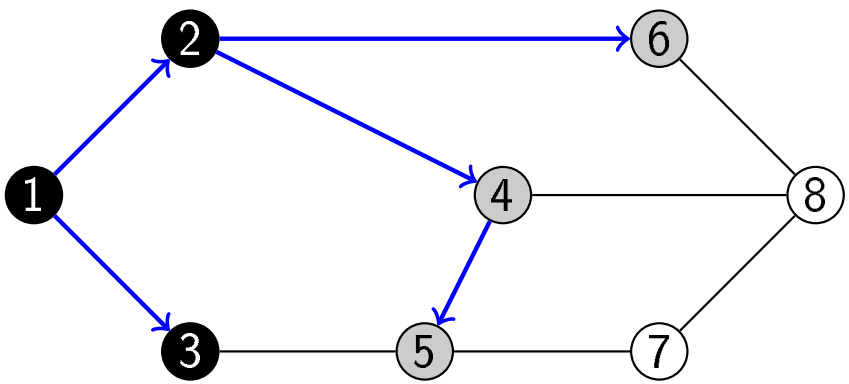


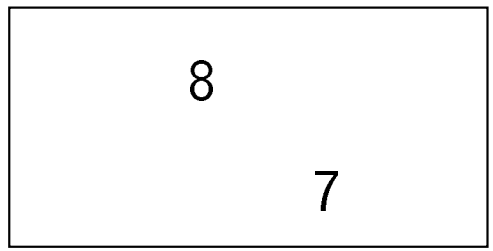


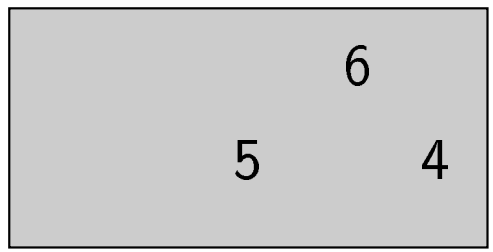
Parcours générique

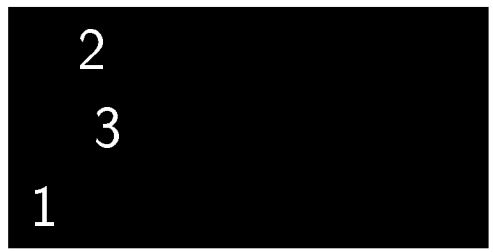
Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;





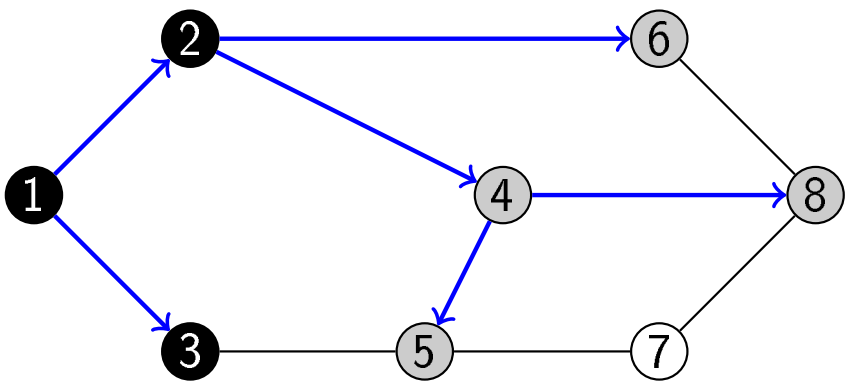


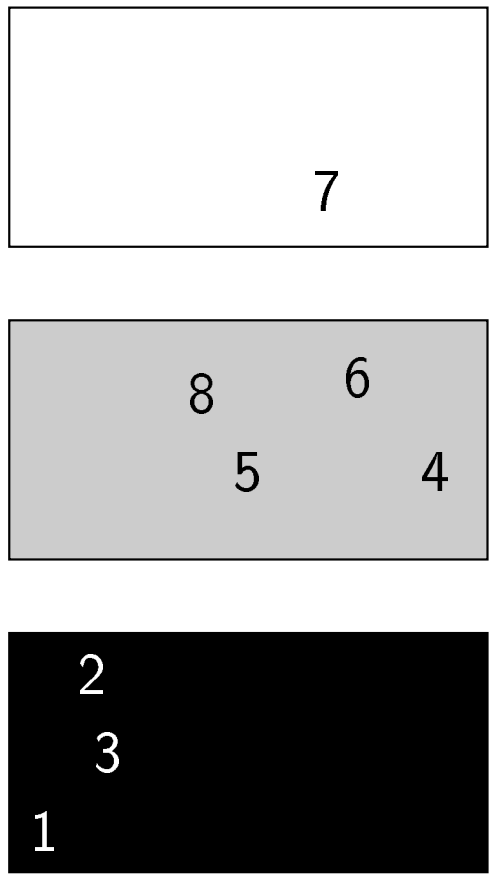


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

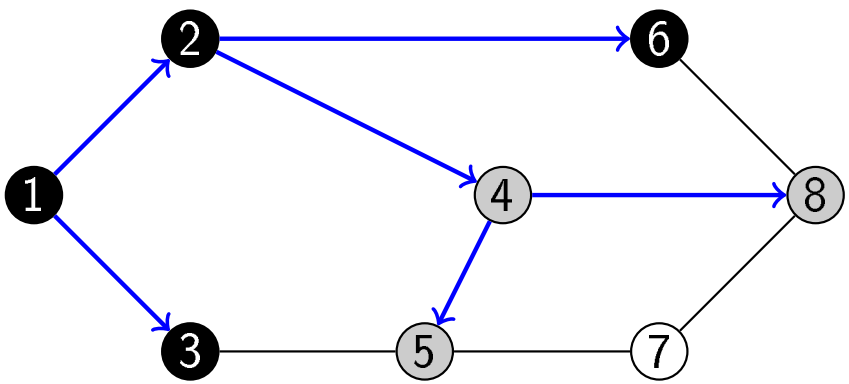


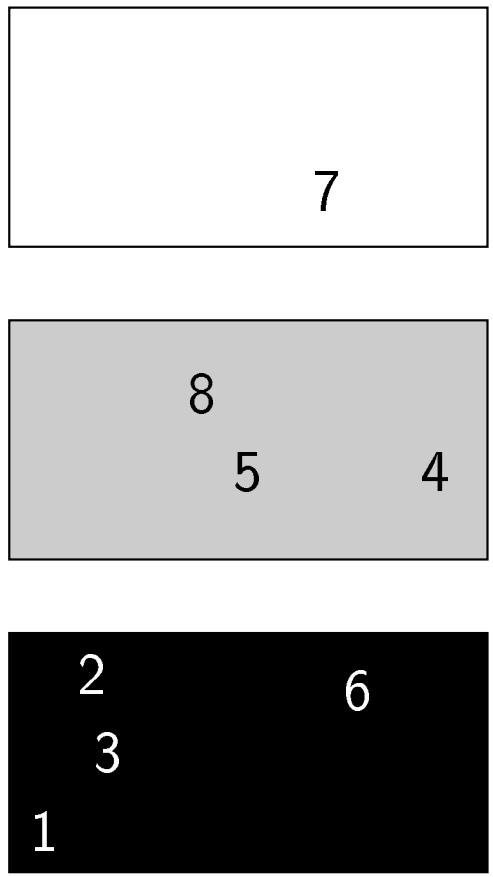


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

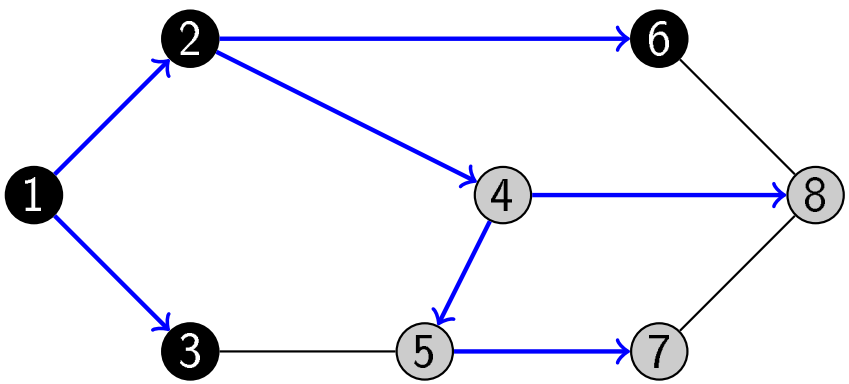


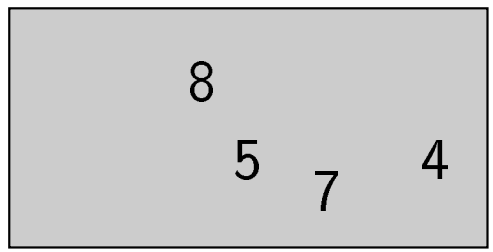


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

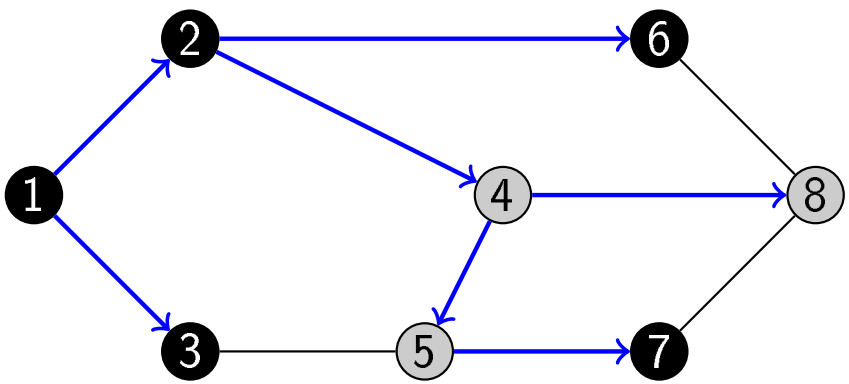


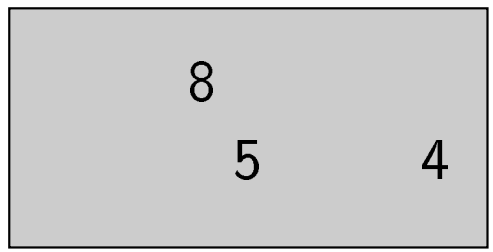


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

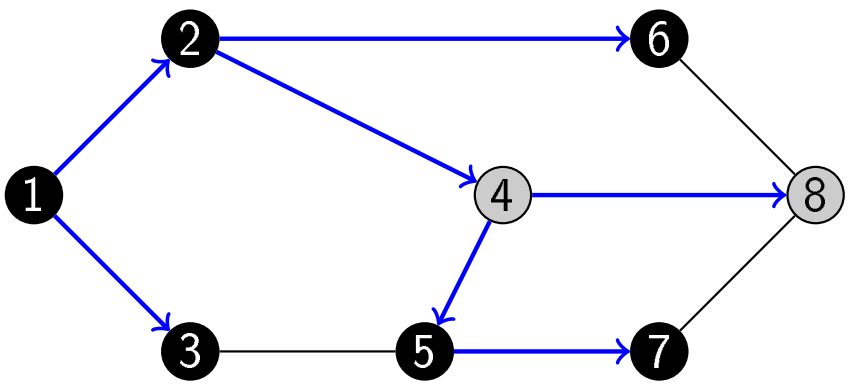


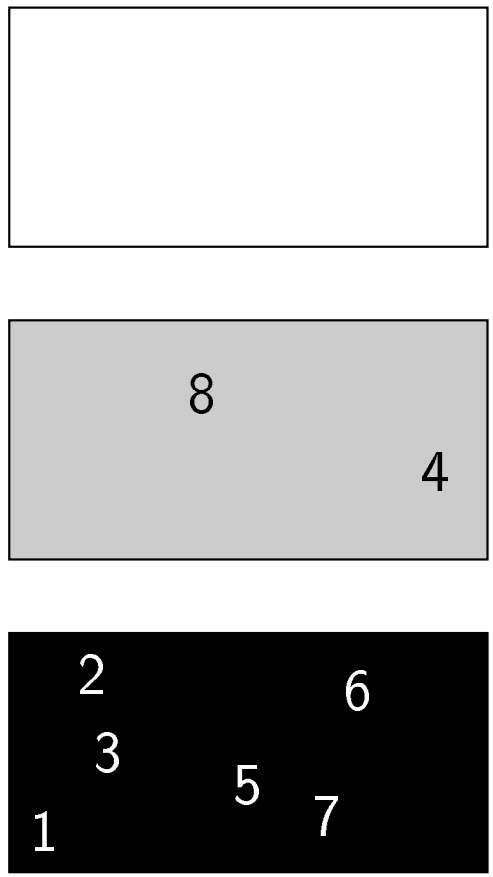


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

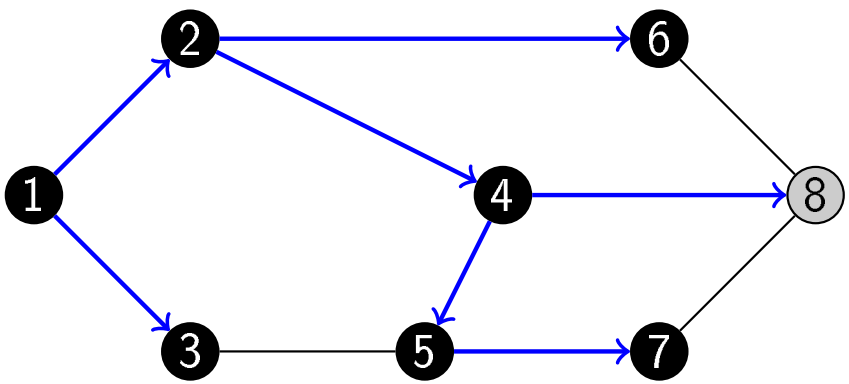


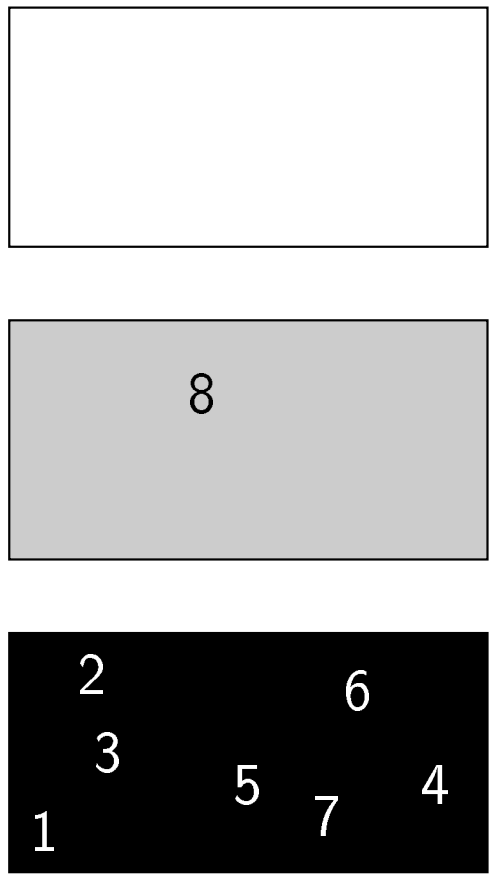


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;

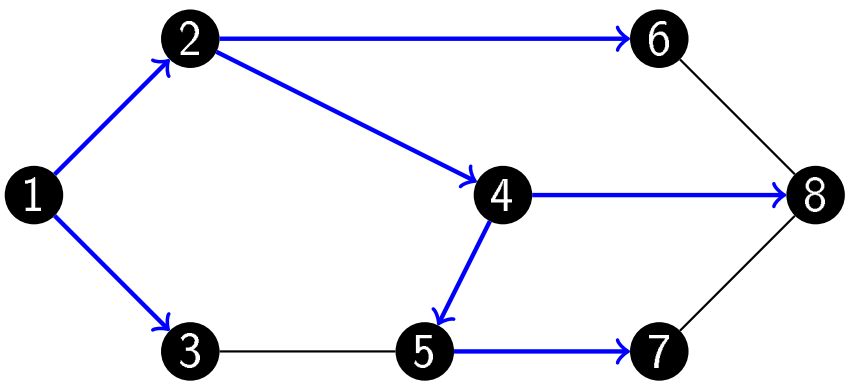


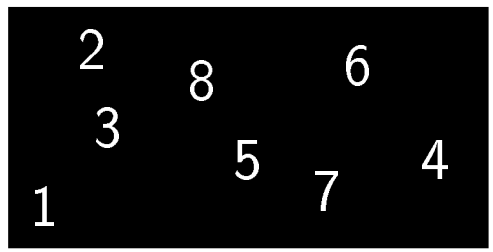


Parcours générique

Parcours(G, s)

pour chaque $v \in S$ **faire** $c[v] \leftarrow$ `blanc`;
$c[s] \leftarrow$ `gris` ; $\pi[s] \leftarrow$ `nil`;
tant que $\exists u \in S$ *tel que* $c[u] =$`gris` **faire**
  si $\exists v \in V[u]$ *tel que* $c[v] =$`blanc` **alors**
    $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
  sinon
    $c[u] \leftarrow$`noir`;



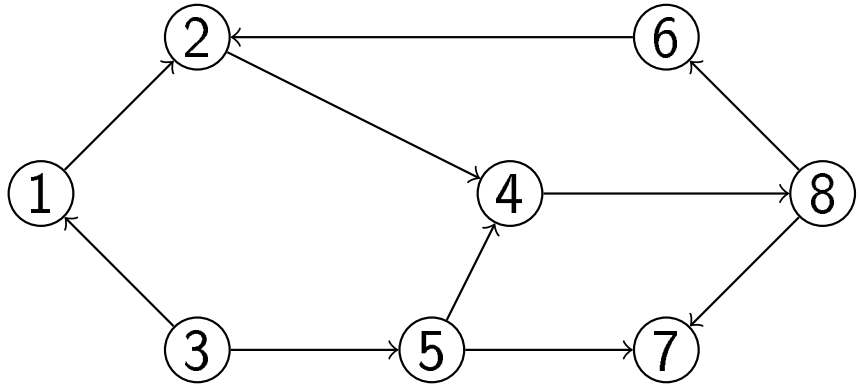


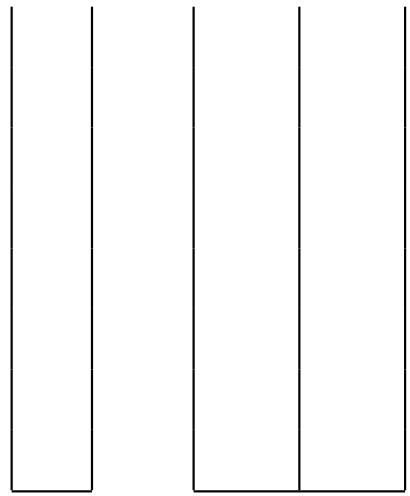
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
    $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
    $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
    $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
    si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
        $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
        `Empiler`(P, v) ;
    sinon
        $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
        $f[u] \leftarrow t$



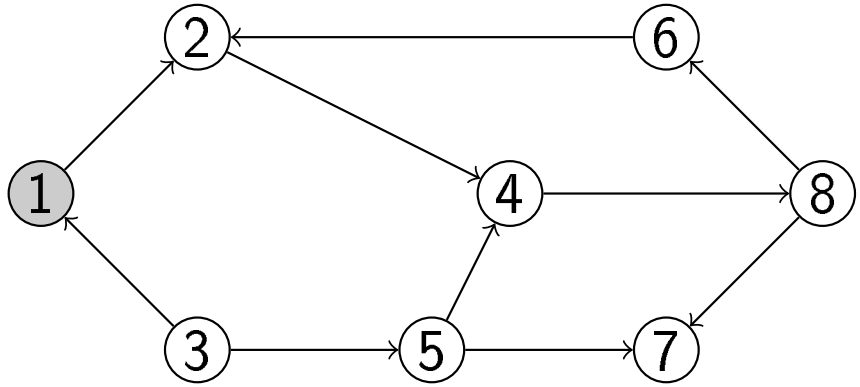


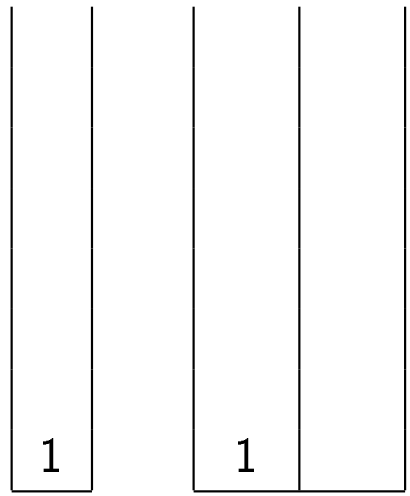
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
    $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
    $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
    $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
    si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
        $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
        `Empiler`(P, v) ;
    sinon
        $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
        $f[u] \leftarrow t$



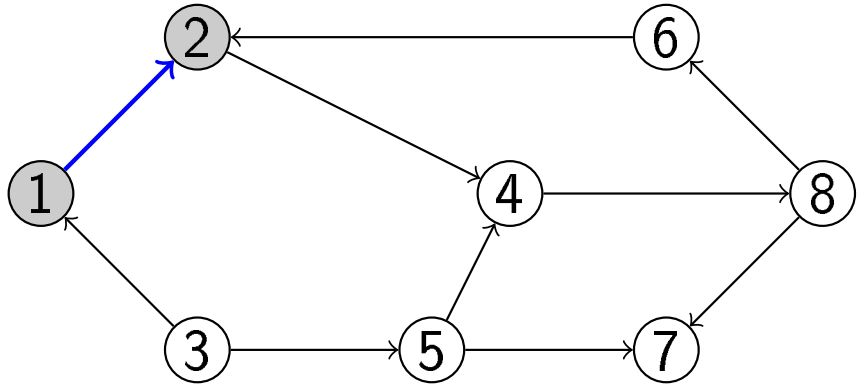


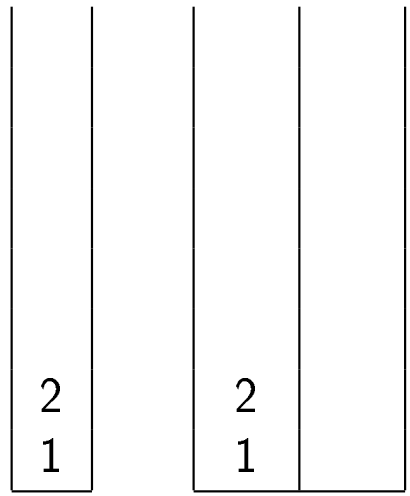
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



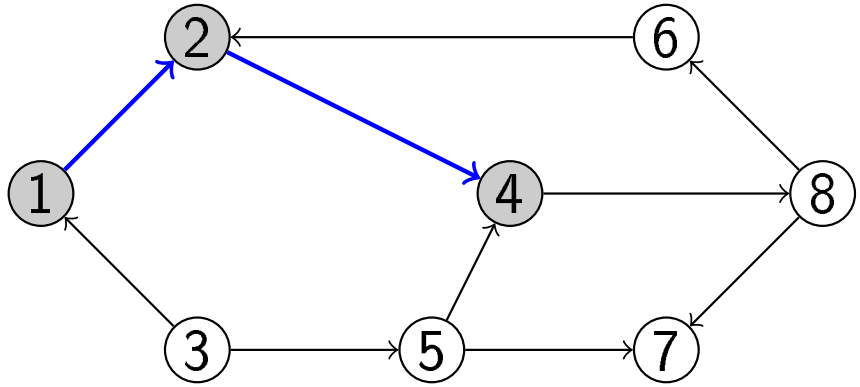


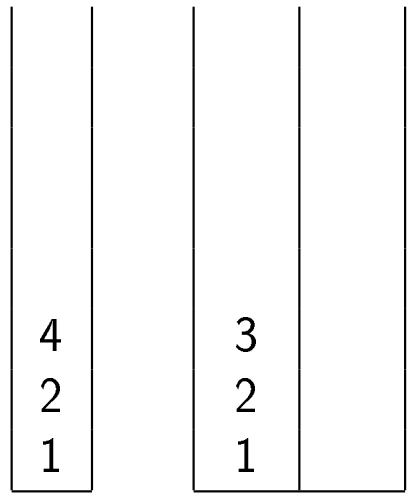
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



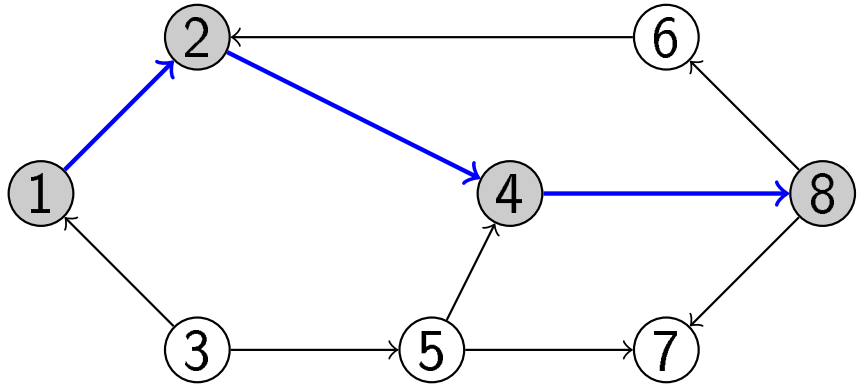


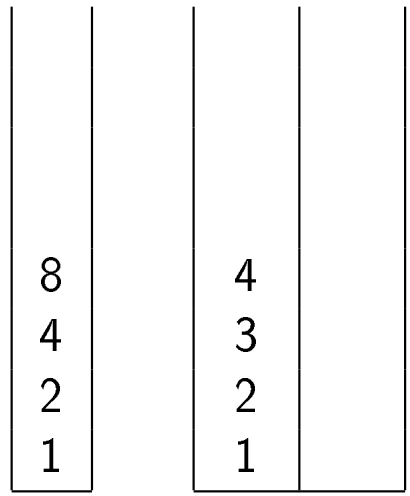
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



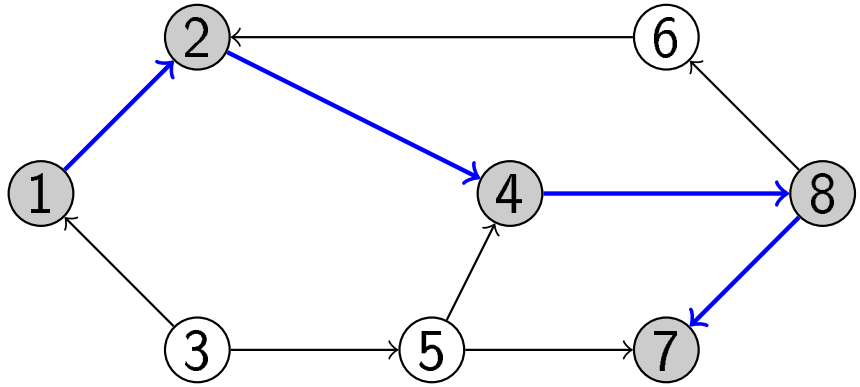


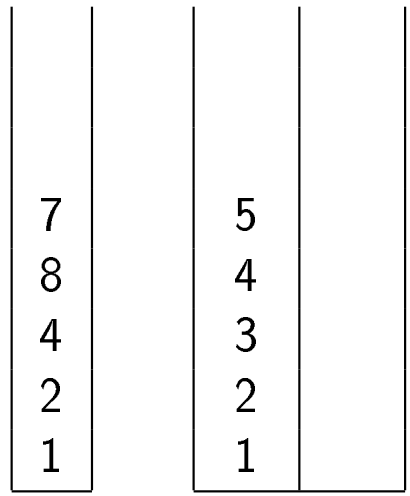
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



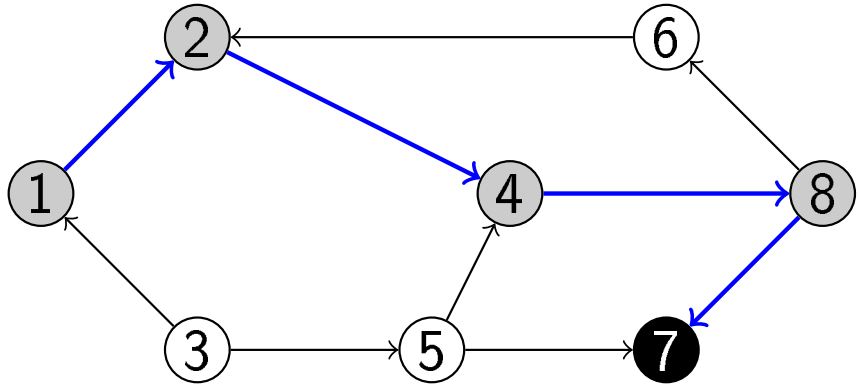


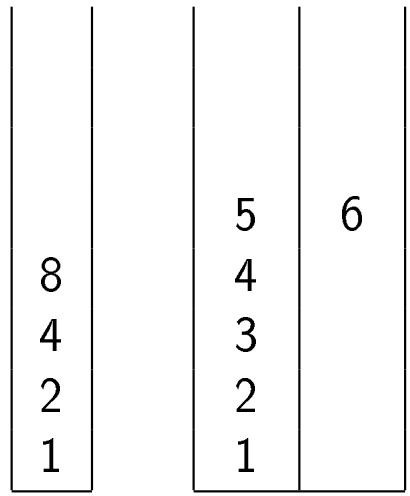
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



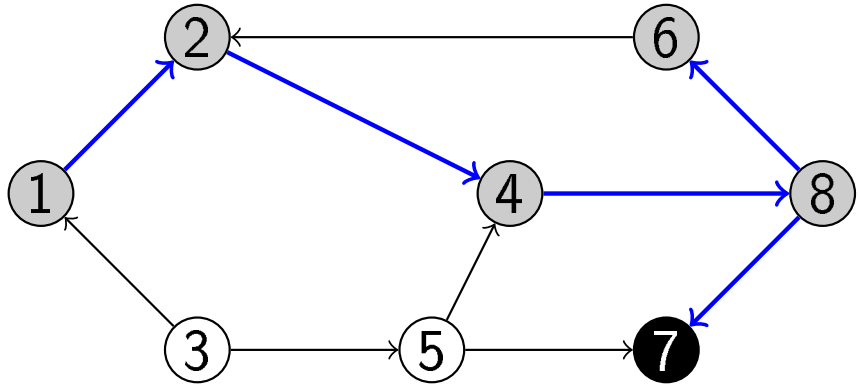


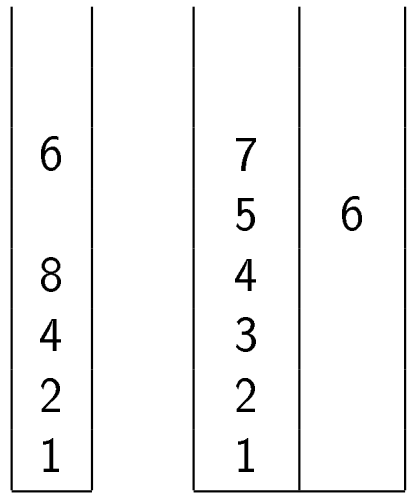
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



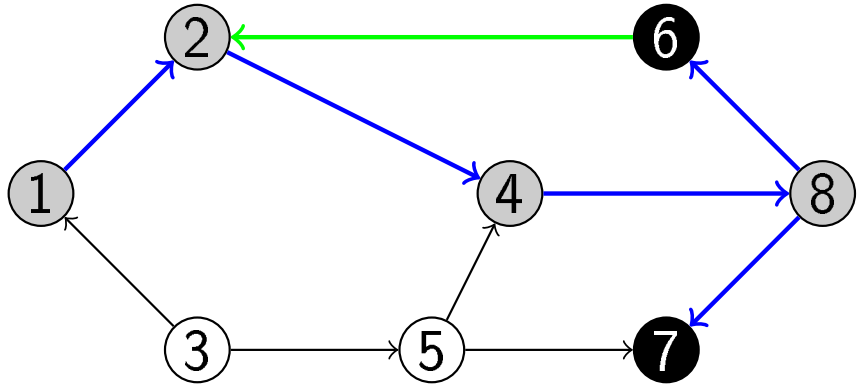


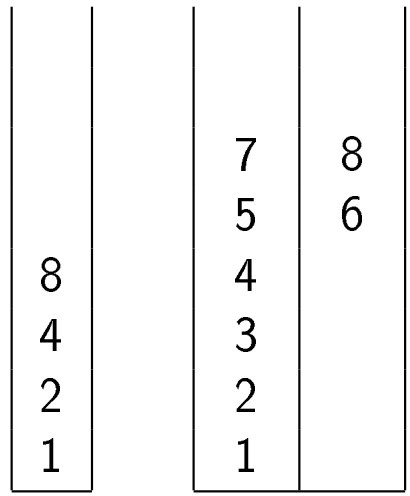
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



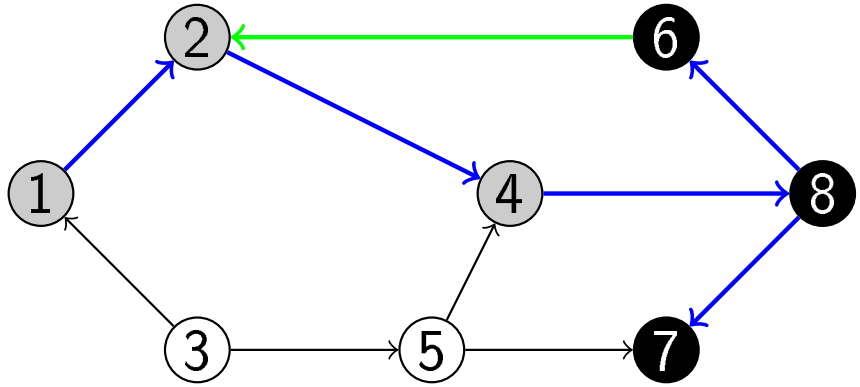


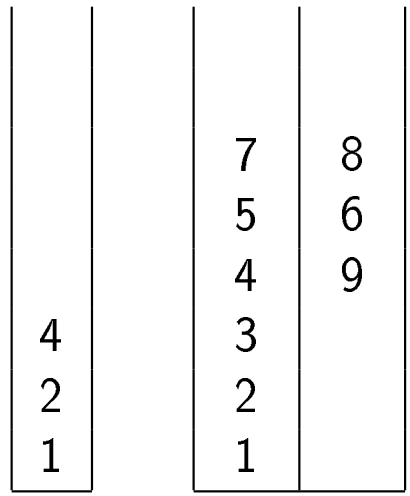
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



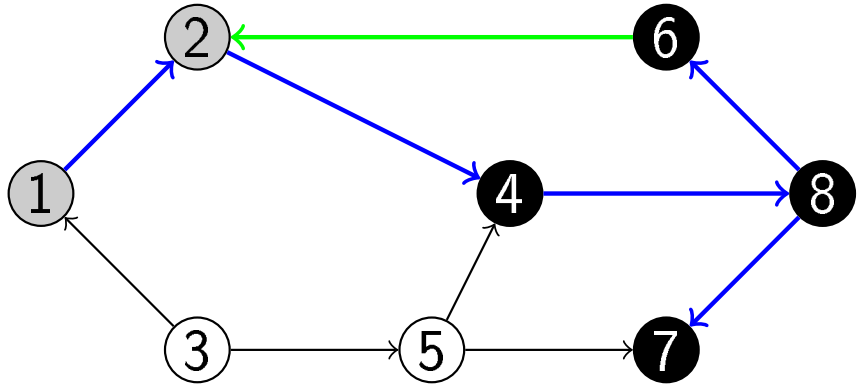


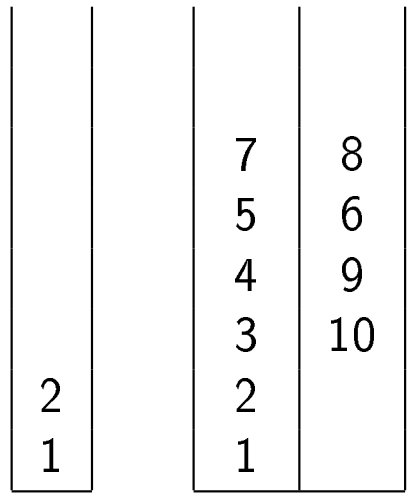
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



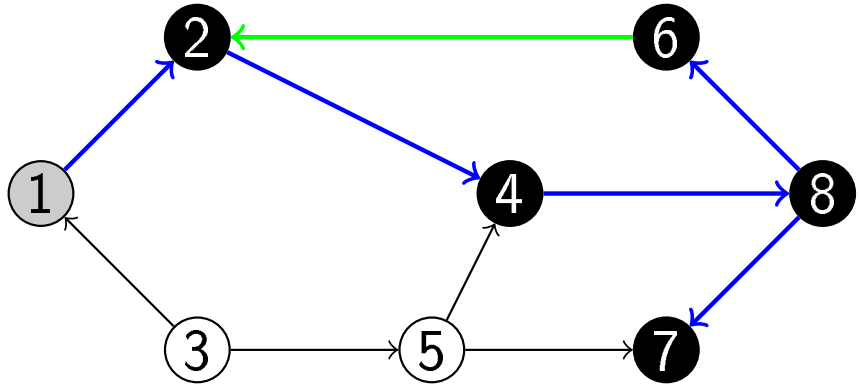


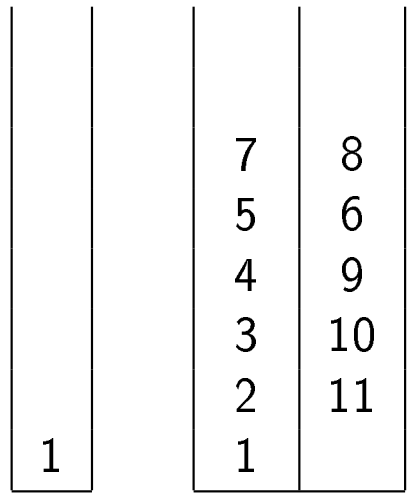
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



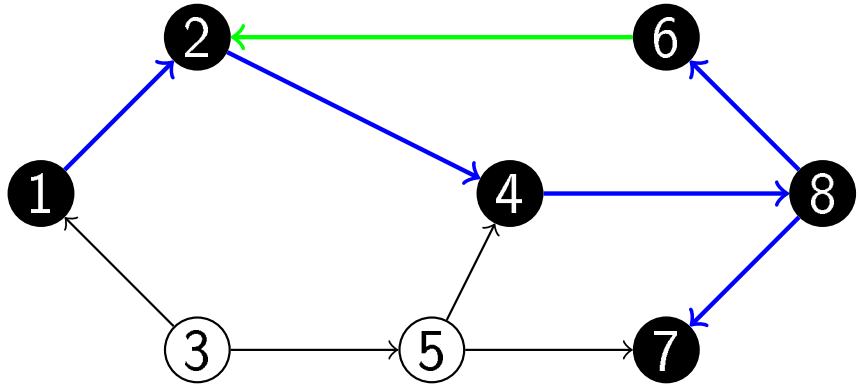


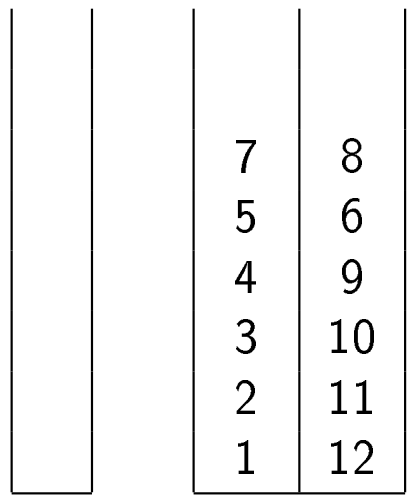
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



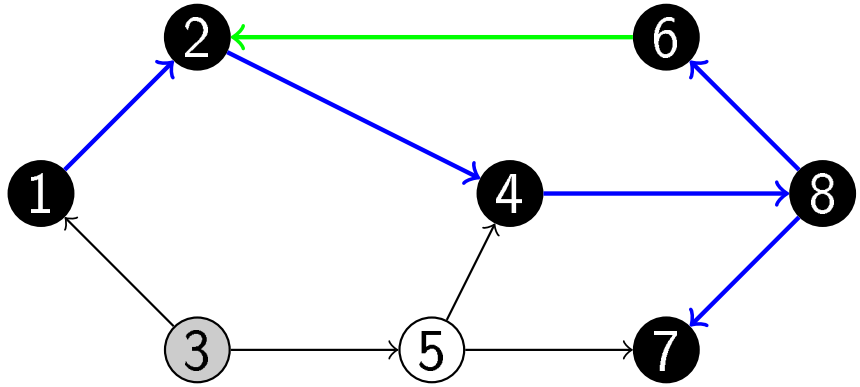


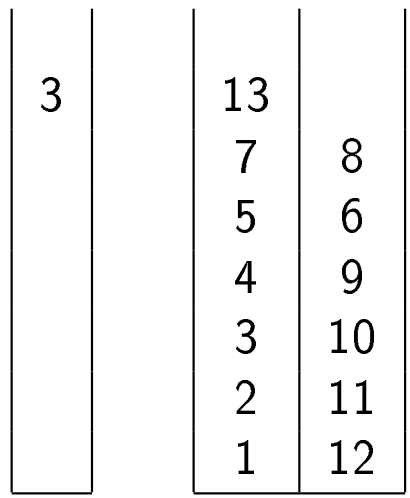
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



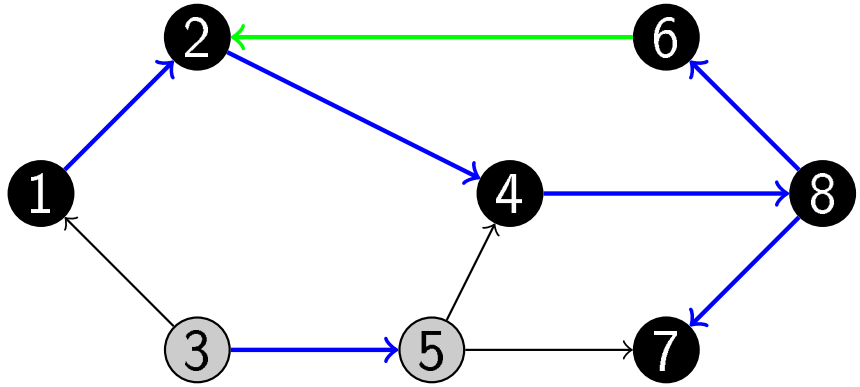


Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



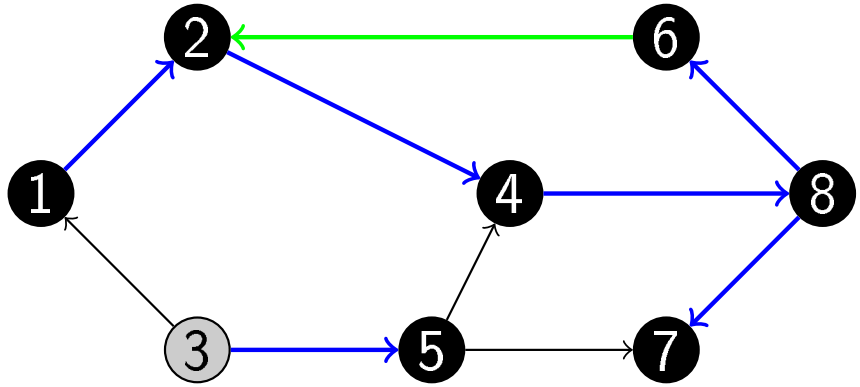
5		14	
3		13	
		7	8
		5	6
		4	9
		3	10
		2	11
		1	12

Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



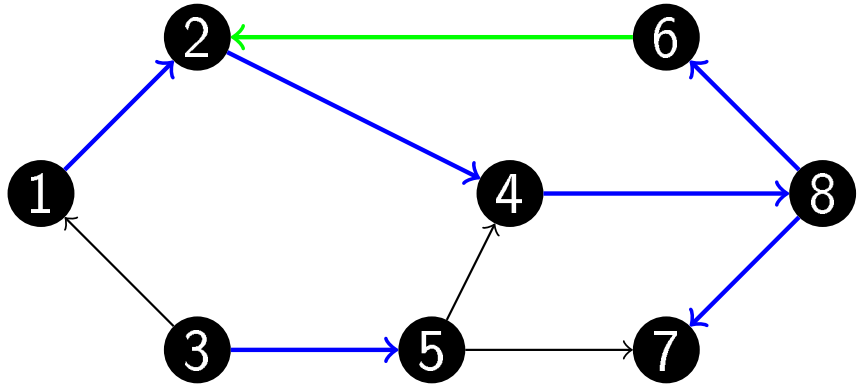
3	14	15
	13	
	7	8
	5	6
	4	9
	3	10
	2	11
	1	12

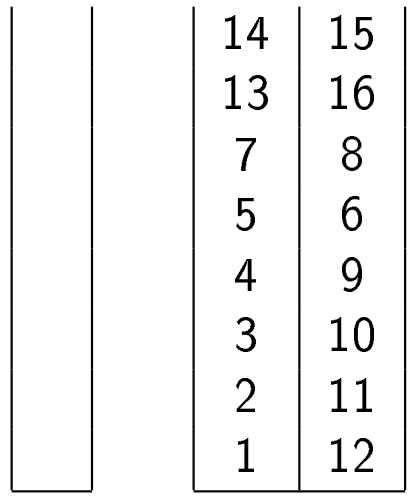
Parcours en profondeur

Les sommets gris sont stockés dans une pile.

PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



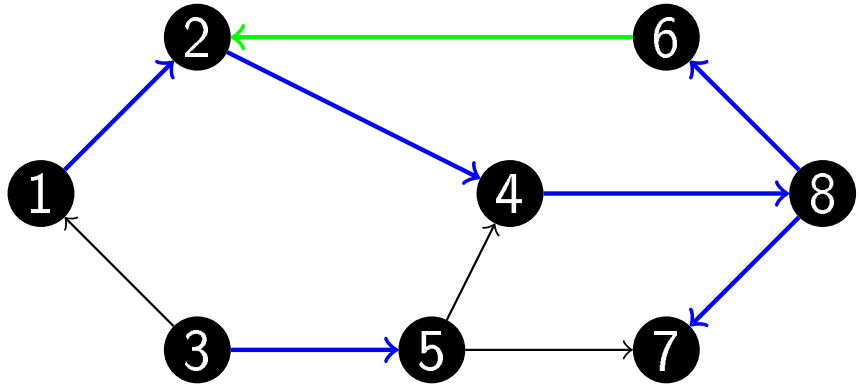


Parcours en profondeur

Les sommets gris sont stockés dans une pile.

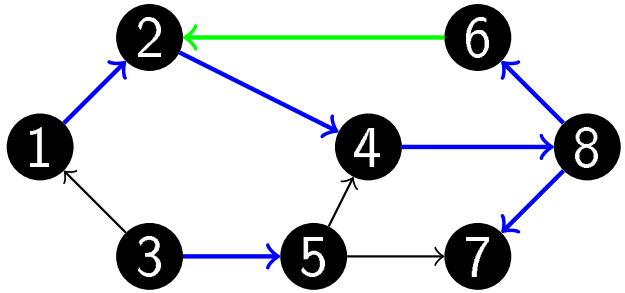
PPronfondeur($G = (S, A), s$)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ blanc ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$;
$P \leftarrow [s]$; $t \leftarrow 1$;
$c[s] \leftarrow$ gris ; $d[s] \leftarrow t$;
tant que $P \neq \emptyset$ **faire**
  $t \leftarrow t + 1$; $u \leftarrow$ `Sommet(P)`;
  si $\exists v \in V[u]$ *tel que* $c[v] = blanc$ **alors**
    $c[v] \leftarrow$ gris ; $\pi[v] \leftarrow u$; $d[v] \leftarrow t$;
    `Empiler`(P, v) ;
  sinon
    $c[u] \leftarrow$ noir ; `Dépiler`(P, u);
    $f[u] \leftarrow t$



5	14	15
3	13	16
6	7	8
7	5	6
8	4	9
4	3	10
2	2	11
1	1	12

Propriétés du parcours en profondeur



	5	3	6	7	8	4	2	1
d	14	13	7	5	4	3	2	1
f	15	16	8	6	9	10	11	12

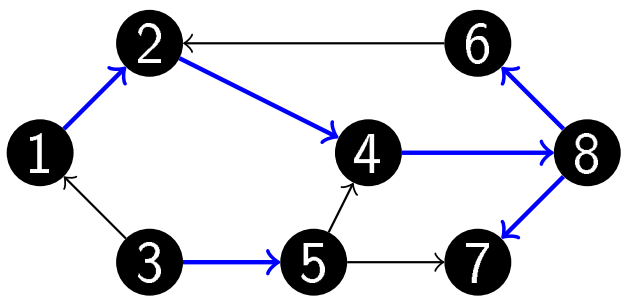
- À partir d'un sommet, on détecte tous ses descendants.
- les intervalles $[d[u], f[u]]$ forment un bon parenthésage : ou bien $[d[u], f[u]] \subset [[d[v], f[v]]$, ou bien $[d[u], f[u]] \cap [[d[v].f[v]] = \emptyset$
- Complexité : $O(|S| + |A|)$.
- Détection des cycles : on détecte des cycles à chaque fois que l'on rencontre un sommet gris (arcs retour).
- Tri topologique : l'ordre donné par l'inverse de f est un tri topologique sur les sommets dans le cas où il n'y a pas de cycle.

Composantes fortement connexes

CFC($G = (S, A)$)

`Pprofondeur` (G) et classer les sommets dans l'ordre décroissant de f;
`Pprofondeur` $({}^tG)$ en prenant les sommets dans l'ordre donné précédemment;

Chaque arbre donné par cet algorithme est une composante fortement connexe.



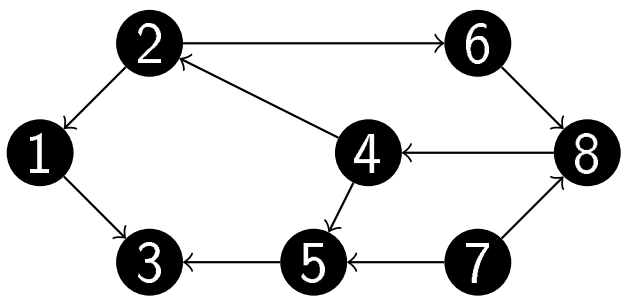
	5	3	6	7	8	4	2	1
d	14	13	7	5	4	3	2	1
f	15	16	8	6	9	10	11	12

Composantes fortement connexes

CFC($G = (S, A)$)

Pprofondeur (G) et classer les sommets dans l'ordre décroissant de f;
Pprofondeur $({}^tG)$ en prenant les sommets dans l'ordre donné précédemment;

Chaque arbre donné par cet algorithme est une composante fortement connexe.



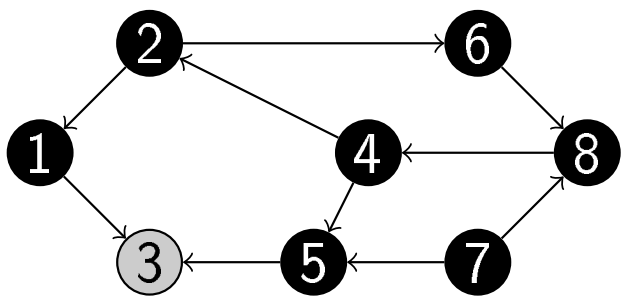
	5	3	6	7	8	4	2	1
d	14	13	7	5	4	3	2	1
f	15	16	8	6	9	10	11	12

Composantes fortement connexes

CFC($G = (S, A)$)

Pprofondeur (G) et classer les sommets dans l'ordre décroissant de f;
Pprofondeur $({}^tG)$ en prenant les sommets dans l'ordre donné précédemment;

Chaque arbre donné par cet algorithme est une composante fortement connexe.



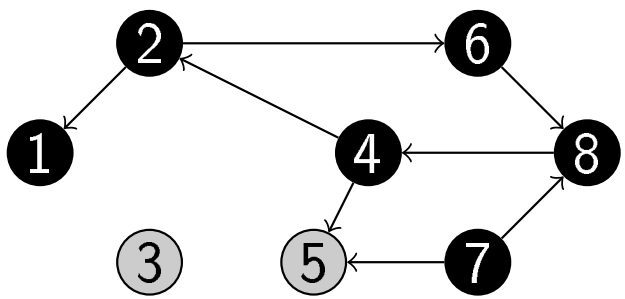
	5	3	6	7	8	4	2	1
d	14	13	7	5	4	3	2	1
f	15	16	8	6	9	10	11	12

Composantes fortement connexes

CFC($G = (S, A)$)

`Pprofondeur` (G) et classer les sommets dans l'ordre décroissant de f;
`Pprofondeur` $({}^tG)$ en prenant les sommets dans l'ordre donné précédemment;

Chaque arbre donné par cet algorithme est une composante fortement connexe.



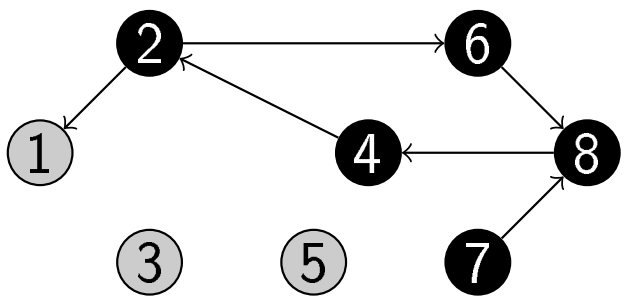
	5	3	6	7	8	4	2	1
d	14	13	7	5	4	3	2	1
f	15	16	8	6	9	10	11	12

Composantes fortement connexes

CFC($G = (S, A)$)

`Pprofondeur` (G) et classer les sommets dans l'ordre décroissant de f;
`Pprofondeur` $({}^tG)$ en prenant les sommets dans l'ordre donné précédemment;

Chaque arbre donné par cet algorithme est une composante fortement connexe.



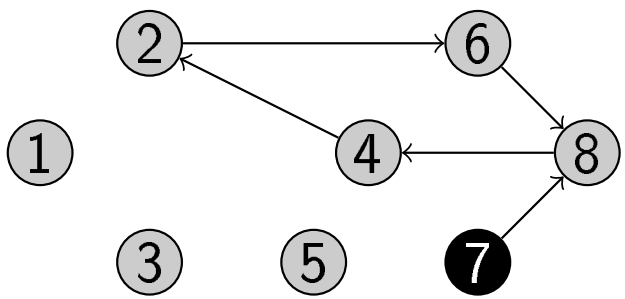
	5	3	6	7	8	4	2	1
d	14	13	7	5	4	3	2	1
f	15	16	8	6	9	10	11	12

Composantes fortement connexes

CFC($G = (S, A)$)

Pprofondeur (G) et classer les sommets dans l'ordre décroissant de f;
Pprofondeur $({}^tG)$ en prenant les sommets dans l'ordre donné précédemment;

Chaque arbre donné par cet algorithme est une composante fortement connexe.



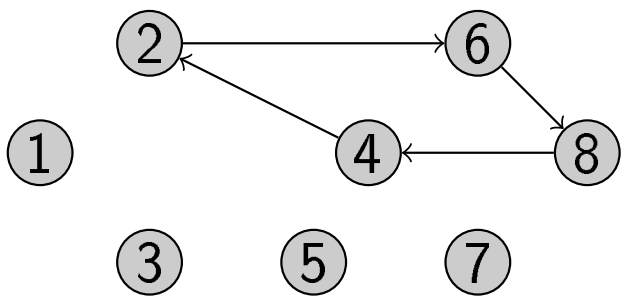
	5	3	6	7	8	4	2	1
d	14	13	7	5	4	3	2	1
f	15	16	8	6	9	10	11	12

Composantes fortement connexes

CFC($G = (S, A)$)

Pprofondeur (G) et classer les sommets dans l'ordre décroissant de f;
Pprofondeur $({}^tG)$ en prenant les sommets dans l'ordre donné précédemment;

Chaque arbre donné par cet algorithme est une composante fortement connexe.



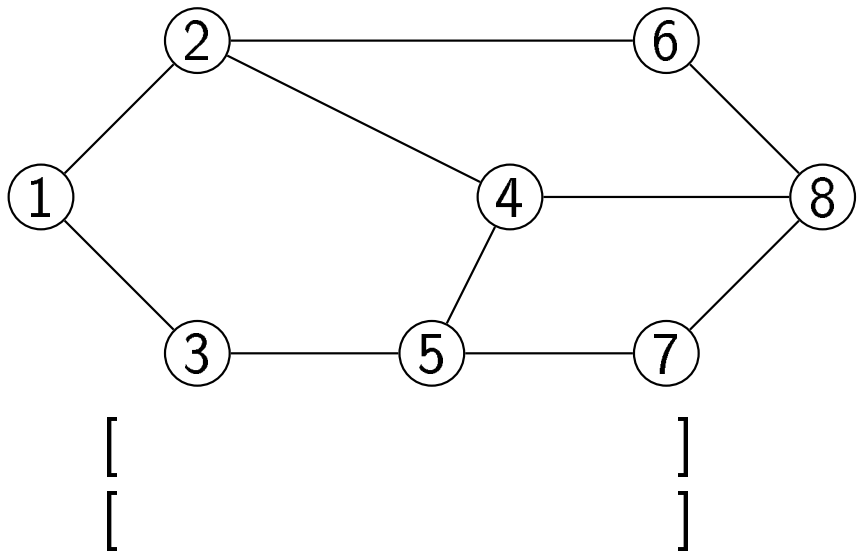
	5	3	6	7	8	4	2	1
d	14	13	7	5	4	3	2	1
f	15	16	8	6	9	10	11	12

Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);

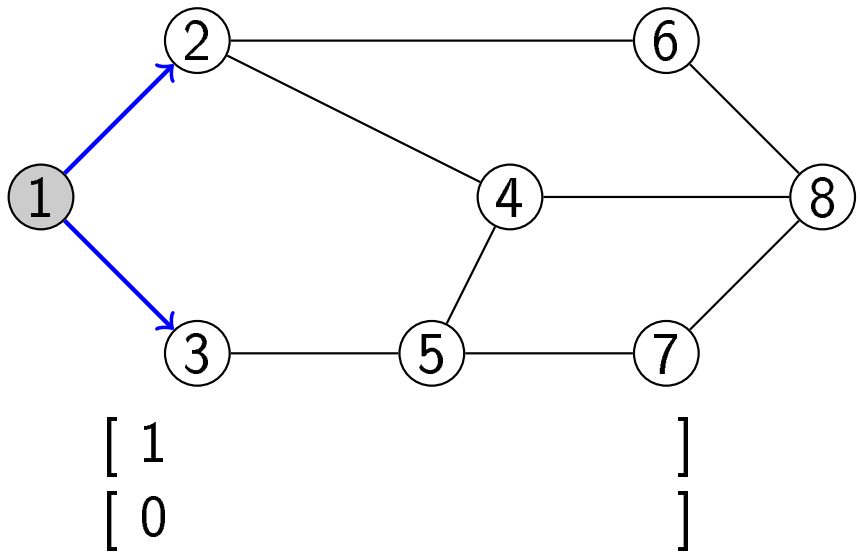


Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);

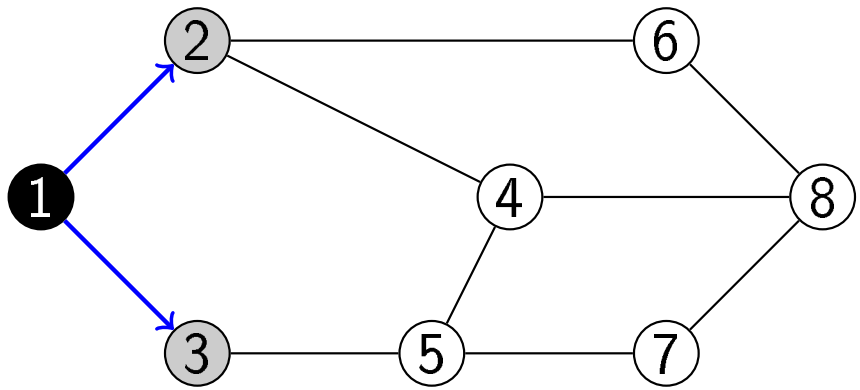


Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);



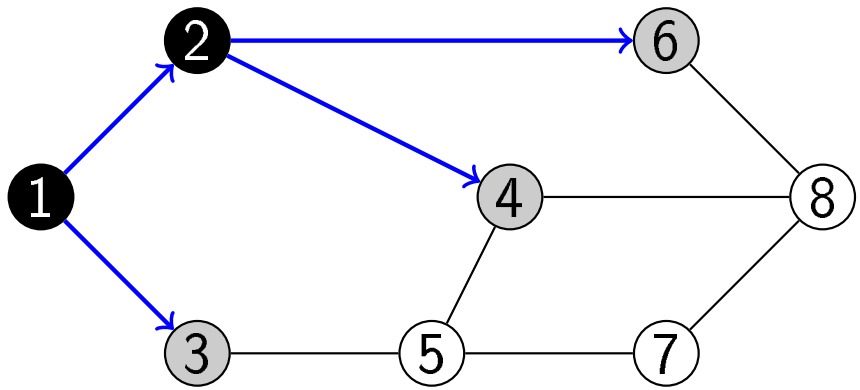
[2 3]
[0 1 1]

Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);



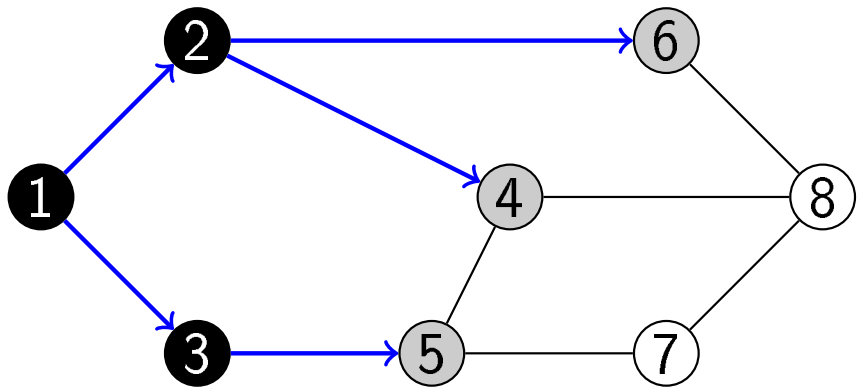
[3 4 6]
[0 1 1 2 2]

Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);



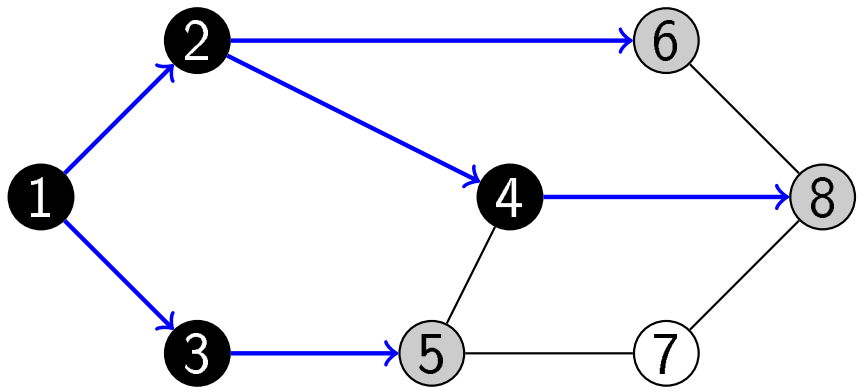
[4 6 5]
[0 1 1 2 2 2]

Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);



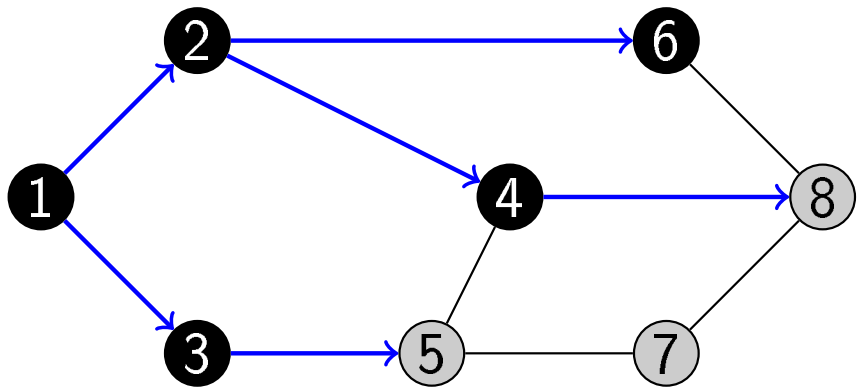
[6 5 8]
[0 1 1 2 2 2 3]

Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);



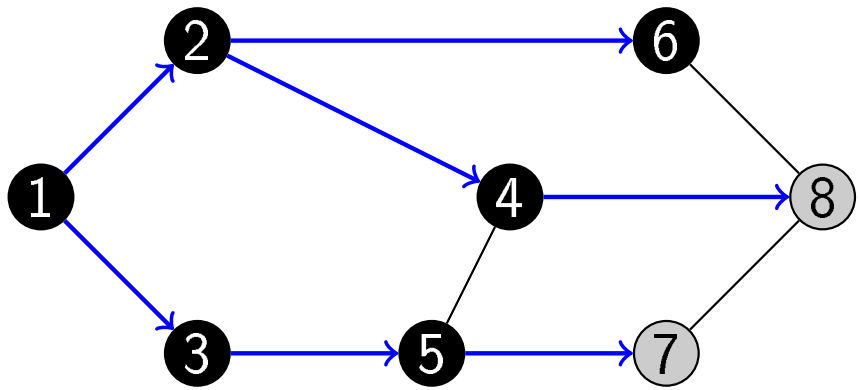
[5 8]
[0 1 1 2 2 2 3]

Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);



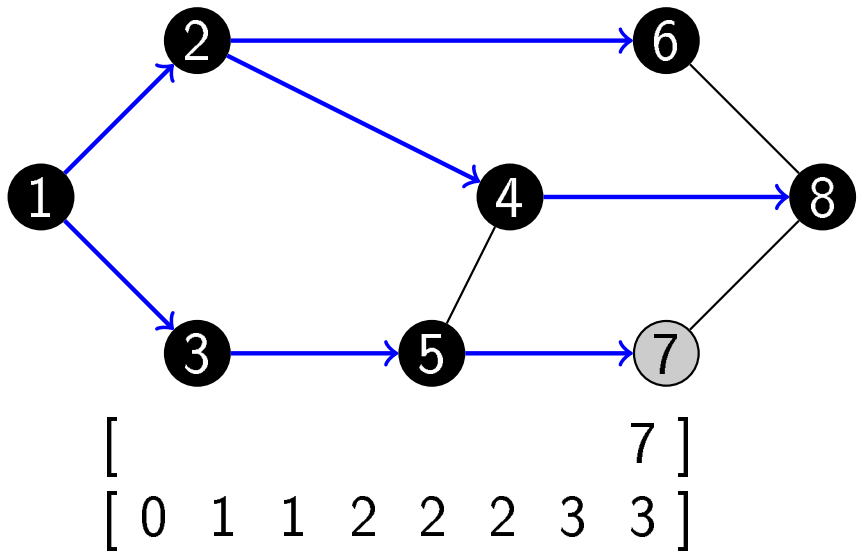
[8 7]
[0 1 1 2 2 2 3 3]

Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);

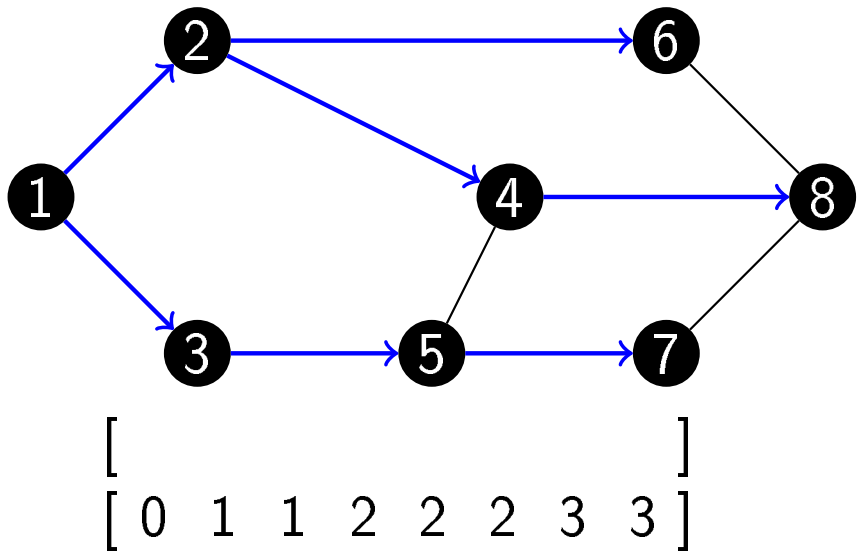


Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);

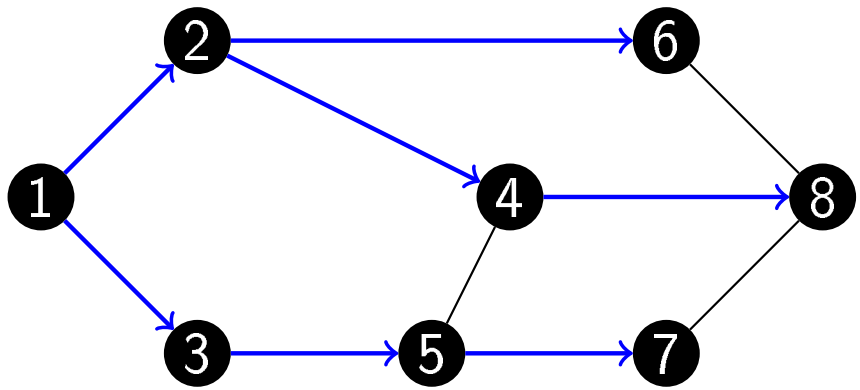


Parcours en largeur et structure de file

Les sommets gris sont stockés dans une file : on traite les voisins non encore visités de sommet de tête simultanément.

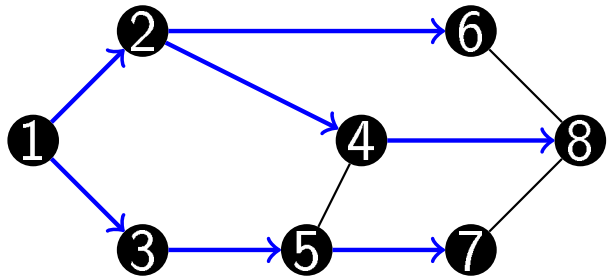
PLargeur(G, s)

pour chaque $v \in S$ **faire**
  $c[v] \leftarrow$ `blanc` ; $\pi[v] \leftarrow$ `nil`;
  $d[v] \leftarrow \infty$;
$c[s] \leftarrow$ `gris` ; $F \leftarrow [s]$; $d[s] \leftarrow 0$;
tant que $F \neq \emptyset$ **faire**
  `défiler`(F, u);
  pour chaque $v \in V[u]$ **faire**
    si $c[v] =$ `blanc` **alors**
      $c[v] \leftarrow$`gris` ; $\pi[v] \leftarrow u$;
      $d[v] \leftarrow d[u] + 1$;
      `enfiler`(F,v);



[1 2 3 4 6 5 8 7]
[0 1 1 2 2 2 3 3]

Propriétés du parcours en largeur



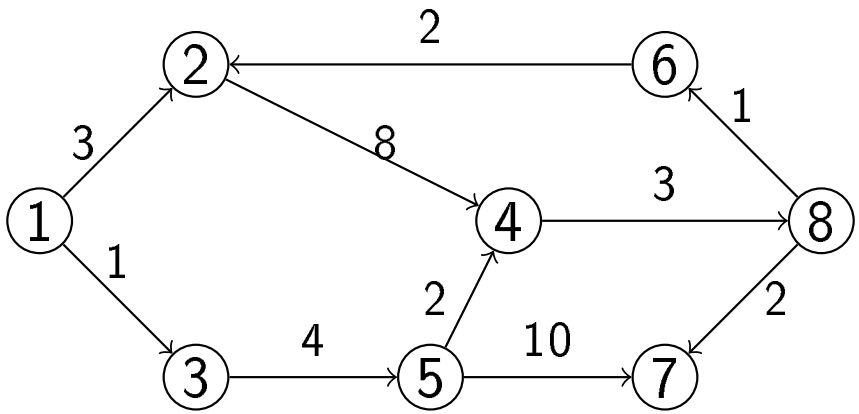
[1 2 3 4 6 5 8 7]
[0 1 1 2 2 2 3 3]

1. Les sommets « découverts » sont exactement ceux qui sont dans la même composante connexe de s (dans le cas des graphes orientés, ce sont les sommets tels qu'il existe un chemin de s vers chacun de ces sommets) ;
2. π construit l'arbre des plus courts chemins depuis s ;
3. $d[u]$ est exactement la distance de u à s ($= \infty$ si il n'existe pas de chemin de s à u).
4. Complexité : $O(|S| + |A|)$.

Graphe pondéré

Définition (Graphe pondéré)

Graphe orienté muni d'une fonction de poids $w : A \to \mathbb{R}$.



La distance entre deux points est

$$\delta(u, v) = \min_{u_0=u, u_1, \ldots, u_k=v} \sum_{\ell=1}^{k} w(u_{k-1}, u_k).$$

File de priorité

Définition (File de priorité)

Structure de données dans laquelle on peut insérer des élements (ici, des entiers naturels), les mettre à jour et extraire un élément minimal.

On ne détaillera pas ici l'implémentation d'une telle structure. La file est un cas particulier d'une file de priorité où l'ordre des éléments est leur ordre d'insertion.

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

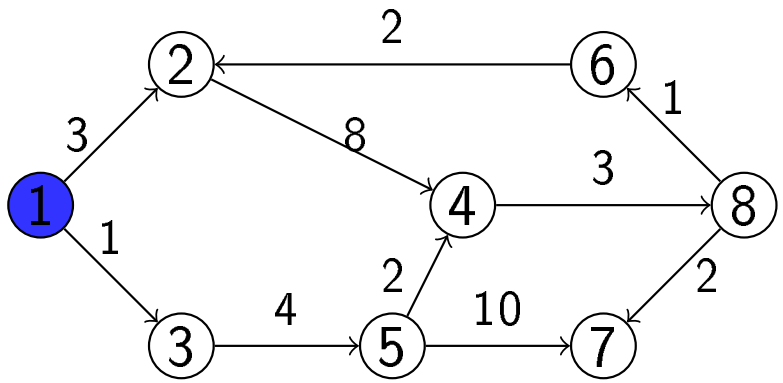
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



1	2	3	4	5	6	7	8
0	∞	∞	∞	∞	∞	∞	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

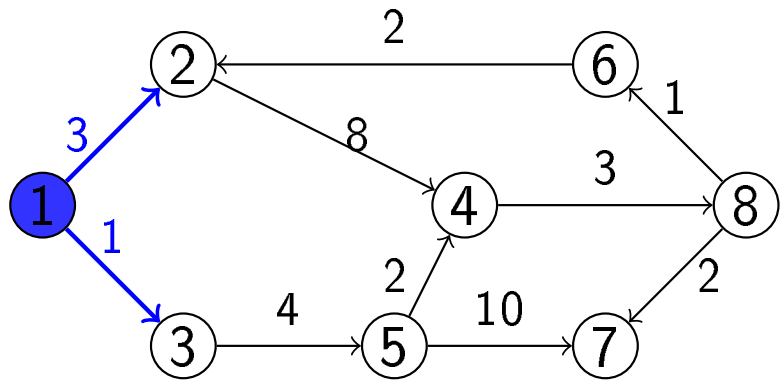
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



1	2	3	4	5	6	7	8
0	3	1	∞	∞	∞	∞	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

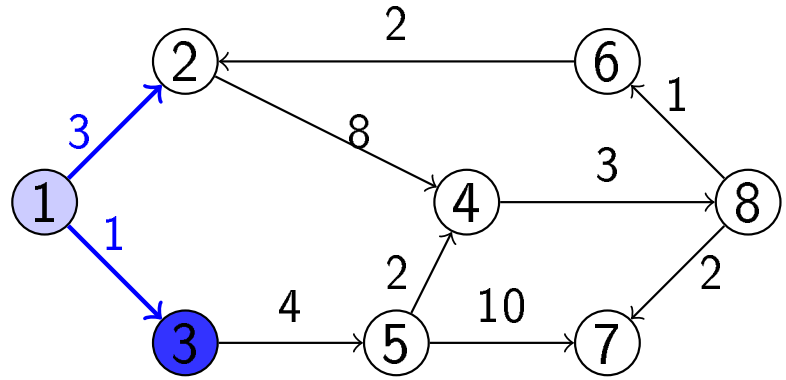
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



	2	3	4	5	6	7	8
0	3	1	∞	∞	∞	∞	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

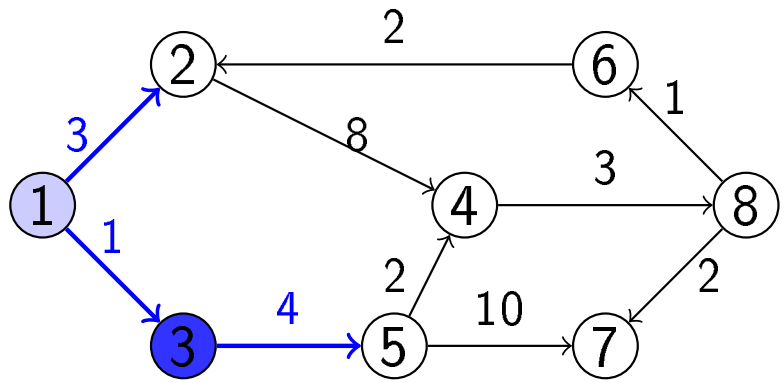
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



	2	3	4	5	6	7	8
0	3	1	∞	5	∞	∞	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

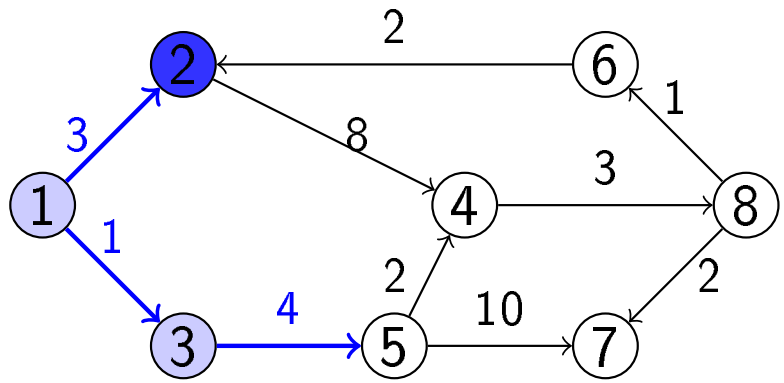
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



	2		4	5	6	7	8
0	3	1	∞	5	∞	∞	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

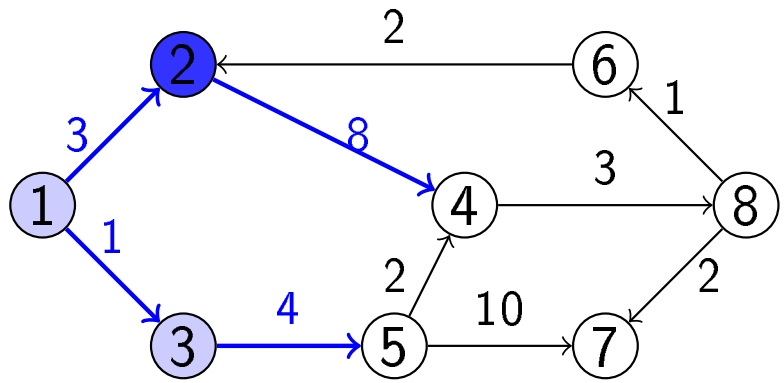
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



	2		4	5	6	7	8
0	3	1	11	5	∞	∞	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

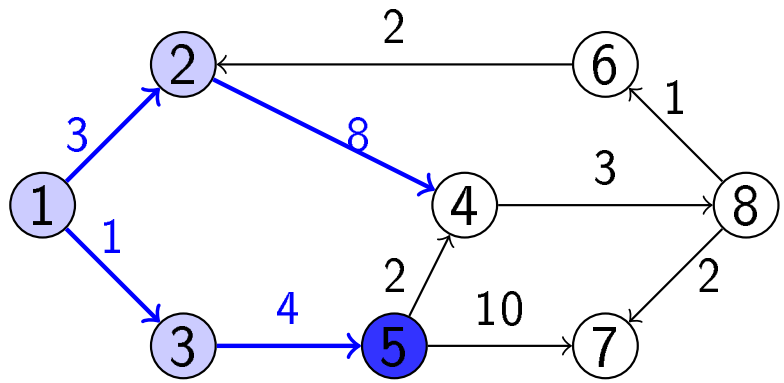
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



			4	5	6	7	8
0	3	1	11	5	∞	∞	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

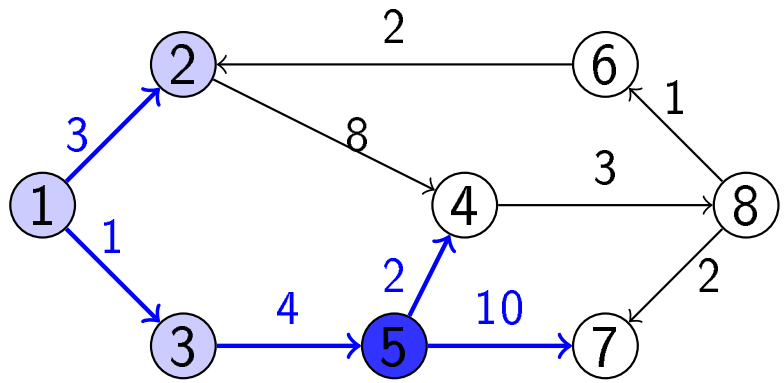
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



			4	5	6	7	8
0	3	1	7	5	∞	15	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$nil;
$d[s] \leftarrow 0$;

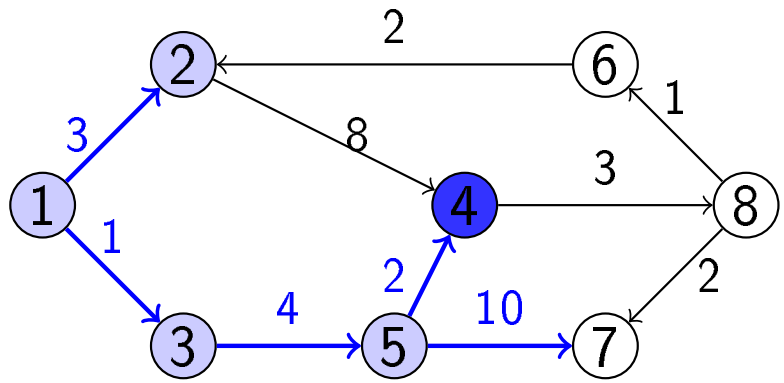
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

Initialisation(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ Extraire_min(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | Relâchement(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



			4		6	7	8
0	3	1	7	5	∞	15	∞

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$nil;
$d[s] \leftarrow 0$;

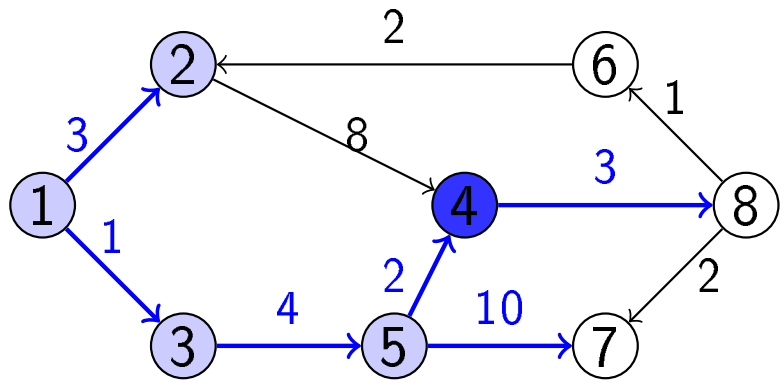
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

Initialisation(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ Extraire_min(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | Relâchement(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



			4		6	7	8
0	3	1	7	5	∞	15	10

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$nil;
$d[s] \leftarrow 0$;

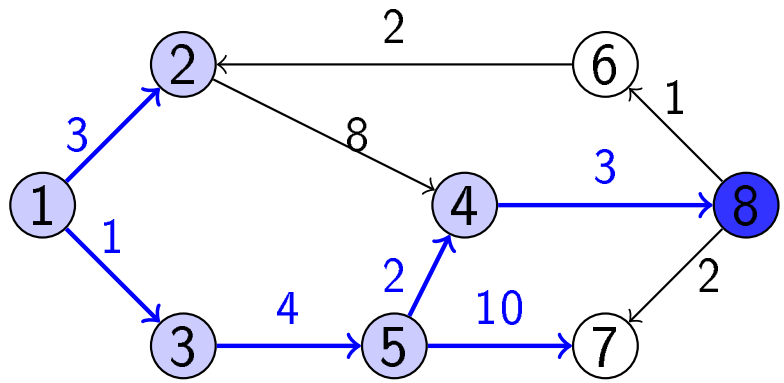
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

Initialisation(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ Extraire_min(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | Relâchement(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



					6	7	8
0	3	1	7	5	∞	15	10

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

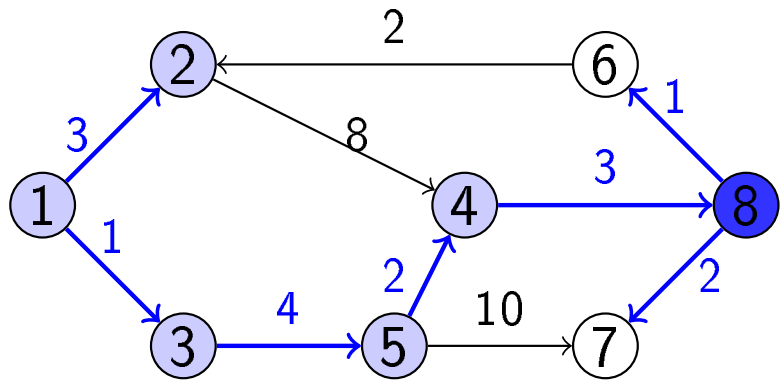
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



					6	7	8
0	3	1	7	5	11	12	10

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$nil;
$d[s] \leftarrow 0$;

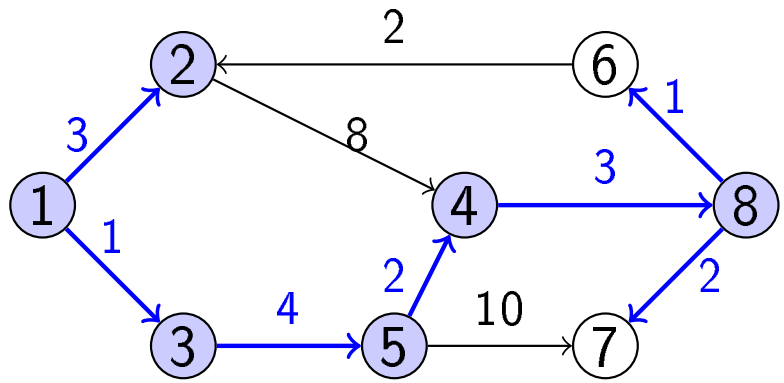
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

Initialisation(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ Extraire_min(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | Relâchement(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



					6	7	
0	3	1	7	5	11	12	10

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

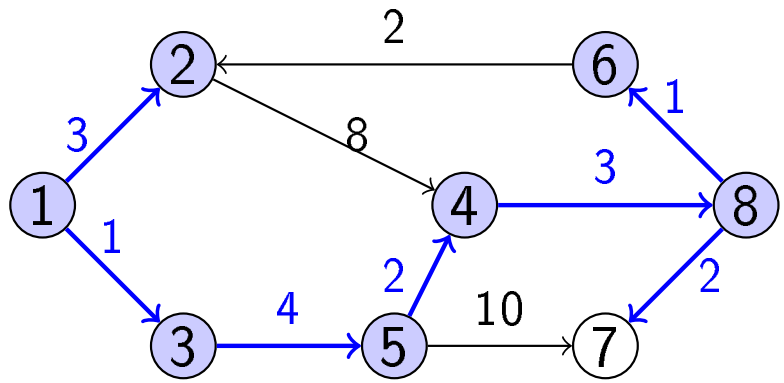
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



						7	
0	3	1	7	5	11	12	10

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

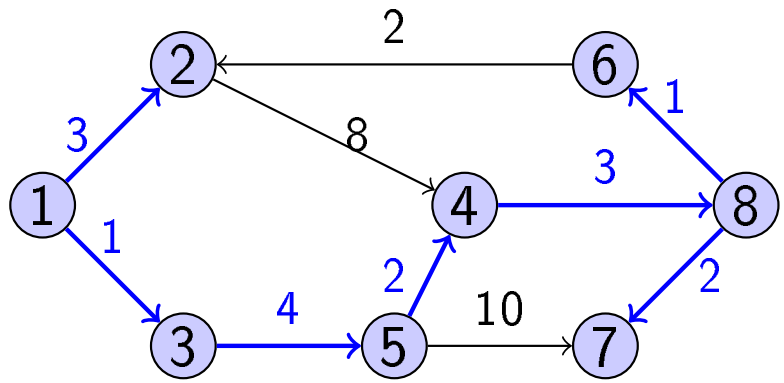
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



0	3	1	7	5	11	12	10

Algorithme de Dijkstra

Initialisation(G, s)

pour chaque $v \in S$ **faire** $d[v] \leftarrow \infty$;
  $\pi[v] \leftarrow$`nil`;
$d[s] \leftarrow 0$;

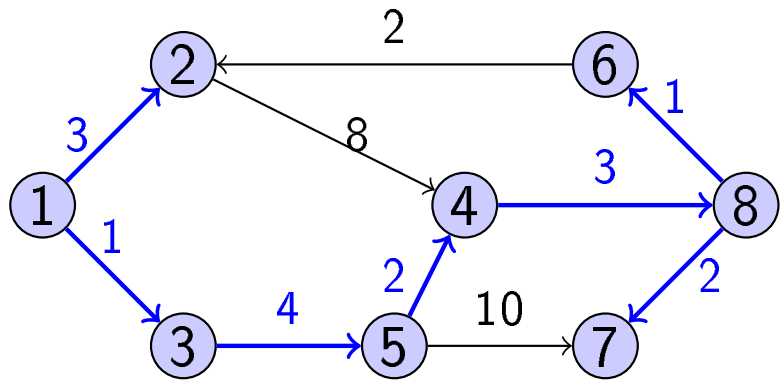
Relâchement(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  | $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$;

Dijkstra(G, w, s)

`Initialisation`(G, s);
$F \leftarrow S$; **tant que** $F \neq \emptyset$ **faire**
  | $u \leftarrow$ `Extraire_min`(F);
  | **pour chaque** $v \in Adj[u]$ **faire**
  | | `Relâchement`(u, v, w) ;

$\forall (u, v) \in A, w(u, v) \geq 0.$



1	2	3	4	5	6	7	8
0	3	1	7	5	11	12	10

Preuve de l'algorithme de Dijkstra

- Invariant de boucle : au début de la boucle **tant que**, le sommet qui minimise $d[u]$ vérifie $d[u] = \delta(s, u)$.
- Vrai au premier passage : $\delta(s, s) = 0 = d[s]$;
- Si c'est vrai pour les $k - 1$ premiers passages, cela veut dire que pour les sommets v retirés de F, $d[v] = \delta(s, v)$. Soit $u \in F$ qui minimise $d[u]$. Si $d[u] > \delta(s, u)$, alors considérons le chemin le plus court de s à u. Il commence dans $S \setminus F$ puis continue dans F. Soit z le premier sommet rencontré sur ce chemin dans F. Alors $d[z] = d[y] + w(y, z) = \delta(s, z) < d[u]$ car les poids des arcs sont positifs.