Algorithmique des graphes

4 — Graphes pondérés, la suite; graphes orientés

Anthony Labarre

17 février 2021

Université Gustave Eiffel

Correction de l'algorithme de Prim

- On a vu la fois passée l'algorithme de Prim, qui construit un ACPM à partir d'un sommet donné pour la composante connexe qui le contient ;
- Prouvons aujourd'hui que cet algorithme fonctionne (reprenez le pseudocode !) ;

Correction de l'algorithme de Prim

On doit prouver que :

1. l'algorithme se termine :

Correction de l'algorithme de Prim

On doit prouver que :

1. l'algorithme se termine : évident, on arrête quand le tas est vide, sa taille maximum est $|E|$, et chaque itération en extrait au moins un élément ;

Correction de l'algorithme de Prim

On doit prouver que :

1. l'algorithme se termine : évident, on arrête quand le tas est vide, sa taille maximum est $|E|$, et chaque itération en extrait au moins un élément ;
2. il renvoie un :

Correction de l'algorithme de Prim

On doit prouver que :

1. l'algorithme se termine : évident, on arrête quand le tas est vide, sa taille maximum est $|E|$, et chaque itération en extrait au moins un élément ;
2. il renvoie un :
   1. arbre :

Correction de l'algorithme de Prim

On doit prouver que :

1. l'algorithme se termine : évident, on arrête quand le tas est vide, sa taille maximum est $|E|$, et chaque itération en extrait au moins un élément ;
2. il renvoie un :
   1. arbre : oui, on rejette explicitement toute arête créant un cycle ;

Correction de l'algorithme de Prim

On doit prouver que :

1. l'algorithme se termine : évident, on arrête quand le tas est vide, sa taille maximum est $|E|$, et chaque itération en extrait au moins un élément ;
2. il renvoie un :
   1. arbre : oui, on rejette explicitement toute arête créant un cycle ;
   2. couvrant :

Correction de l'algorithme de Prim

On doit prouver que :

1. l'algorithme se termine : évident, on arrête quand le tas est vide, sa taille maximum est $|E|$, et chaque itération en extrait au moins un élément ;
2. il renvoie un :
   1. arbre : oui, on rejette explicitement toute arête créant un cycle ;
   2. couvrant : oui (si G connexe), chaque itération ajoute une arête contenant un sommet hors de l'arbre ;

Correction de l'algorithme de Prim

On doit prouver que :

1. l'algorithme se termine : évident, on arrête quand le tas est vide, sa taille maximum est $|E|$, et chaque itération en extrait au moins un élément ;
2. il renvoie un :
    1. arbre : oui, on rejette explicitement toute arête créant un cycle ;
    2. couvrant : oui (si G connexe), chaque itération ajoute une arête contenant un sommet hors de l'arbre ;
    3. de poids minimum : c'est ce qu'on va voir.

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
    - hypothèse d'induction (HI) :

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
   - hypothèse d'induction (HI) : il existe un ACPM T_{opt} contenant T, avec $|V(T)| = p < |V(G)|$.

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
   - hypothèse d'induction (HI) : il existe un ACPM T_{opt} contenant T, avec $|V(T)| = p < |V(G)|$.
   - à prouver :

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
   - hypothèse d'induction (HI) : il existe un ACPM T_{opt} contenant T, avec $|V(T)| = p < |V(G)|$.
   - à prouver : HI $\Rightarrow \exists$ ACPM T_{opt} contenant $T \cup e$ (la nouvelle arête sûre sélectionnée) ;

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
   - hypothèse d'induction (HI) : il existe un ACPM T_{opt} contenant T, avec $|V(T)| = p < |V(G)|$.
   - à prouver : HI $\Rightarrow \exists$ ACPM T_{opt} contenant $T \cup e$ (la nouvelle arête sûre sélectionnée) ;
   - si $e \in T_{opt}$, alors $T \cup e \subseteq T_{opt}$ (cf. HI) ;

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
   - hypothèse d'induction (HI) : il existe un ACPM T_{opt} contenant T, avec $|V(T)| = p < |V(G)|$.
   - à prouver : HI $\Rightarrow \exists$ ACPM T_{opt} contenant $T \cup e$ (la nouvelle arête sûre sélectionnée) ;
   - si $e \in T_{opt}$, alors $T \cup e \subseteq T_{opt}$ (cf. HI) ; sinon, on construit un T'_{opt} contenant $T \cup e$ comme suit :

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
   - hypothèse d'induction (HI) : il existe un ACPM T_{opt} contenant T, avec $|V(T)| = p < |V(G)|$.
   - à prouver : HI $\Rightarrow \exists$ ACPM T_{opt} contenant $T \cup e$ (la nouvelle arête sûre sélectionnée) ;
   - si $e \in T_{opt}$, alors $T \cup e \subseteq T_{opt}$ (cf. HI) ; sinon, on construit un T'_{opt} contenant $T \cup e$ comme suit :
     - T_{opt} est un arbre couvrant $\Rightarrow$ $T_{opt} \cup e$ contient un cycle C ;

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
   - hypothèse d'induction (HI) : il existe un ACPM T_{opt} contenant T, avec $|V(T)| = p < |V(G)|$.
   - à prouver : HI $\Rightarrow \exists$ ACPM T_{opt} contenant $T \cup e$ (la nouvelle arête sûre sélectionnée) ;
   - si $e \in T_{opt}$, alors $T \cup e \subseteq T_{opt}$ (cf. HI) ; sinon, on construit un T'_{opt} contenant $T \cup e$ comme suit :
     - T_{opt} est un arbre couvrant $\Rightarrow T_{opt} \cup e$ contient un cycle C ;
     - e est de poids minimum $\Rightarrow \forall\ f \in C : w(f) \geq w(e)$;

L'arbre couvrant que calcule Prim est de poids minimum

Procédons par **induction** sur $|V(T)|$, en montrant qu'à chaque étape, il existe un ACPM T_{opt} contenant T.

1. **cas de base :** $|V(T)| = 1$, trivial ;
2. **induction** :
   - hypothèse d'induction (HI) : il existe un ACPM T_{opt} contenant T, avec $|V(T)| = p < |V(G)|$.
   - à prouver : HI $\Rightarrow \exists$ ACPM T_{opt} contenant $T \cup e$ (la nouvelle arête sûre sélectionnée) ;
   - si $e \in T_{opt}$, alors $T \cup e \subseteq T_{opt}$ (cf. HI) ; sinon, on construit un T'_{opt} contenant $T \cup e$ comme suit :
     - T_{opt} est un arbre couvrant $\Rightarrow T_{opt} \cup e$ contient un cycle C ;
     - e est de poids minimum $\Rightarrow \forall\ f \in C : w(f) \geq w(e)$;
     - $T'_{opt} = T_{opt} \cup e \setminus f$ contient $T \cup e$; et $w(T'_{opt}) = w(T_{opt}) + w(e) - w(f) \leq w(T_{opt})$; $\square$

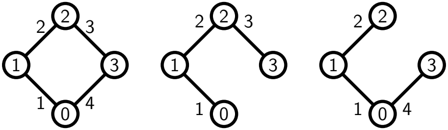
Plus courts chemins dans un graphe

- Les graphes pondérés modélisent fréquemment des réseaux routiers ;
- Comment faire pour trouver un chemin de moindre coût entre deux sommets ?
- Si le graphe n'était pas pondéré, comment calculerait-on un plus court chemin entre deux sommets ?

ACPM et plus courts chemins

Comme le montre l'exemple suivant, les algorithmes de calcul d'un arbre couvrant de poids minimum ne fonctionneront pas pour atteindre cet objectif en général.

Exemple 1 (plus courts chemins depuis 0)



(1) (2)

1. l'ACPM de poids 6 "rate" le plus court chemin 0–3 ;
2. on a bien tous les plus courts chemins, mais le poids de l'arbre n'est pas minimum ;

L'algorithme de Dijkstra

L'algorithme de Dijkstra

- Attention :  "Dij" =  "Day" ;

L'algorithme de Dijkstra

- Attention : "Dij" = "Day" ;
- L'algorithme de Dijkstra construit un **arbre des plus courts chemins** :

L'algorithme de Dijkstra

- Attention : "Dij" = "Day" ;
- L'algorithme de Dijkstra construit un **arbre des plus courts chemins** :
  - la racine de cet arbre est le sommet de départ (la *source*) ;

L'algorithme de Dijkstra

- Attention : "Dij" = "Day" ;
- L'algorithme de Dijkstra construit un **arbre des plus courts chemins** :
  - la racine de cet arbre est le sommet de départ (la *source*) ;
  - l'arbre ne contient qu'un chemin de poids minimum entre la source et chaque sommet du graphe.

L'algorithme de Dijkstra

- Attention : "Dij" = "Day" ;
- L'algorithme de Dijkstra construit un **arbre des plus courts chemins** :
  - la racine de cet arbre est le sommet de départ (la *source*) ;
  - l'arbre ne contient qu'un chemin de poids minimum entre la source et chaque sommet du graphe.
- Hypothèses : **le graphe est simple et sans arête de poids négatif**.

Description de l'algorithme de Dijkstra

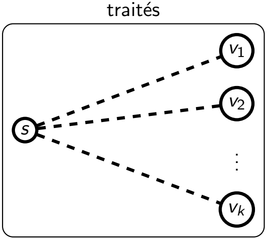
- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;

Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;

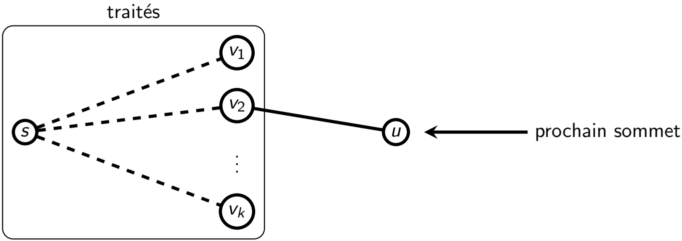
Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



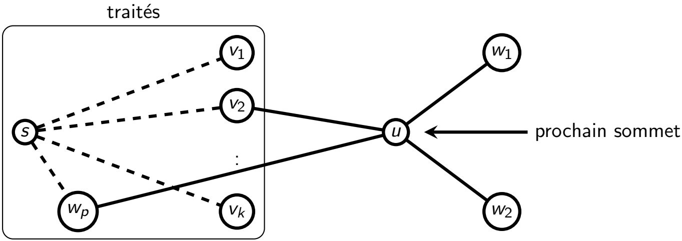
Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



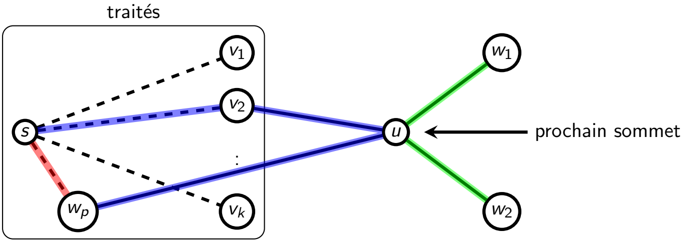
Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



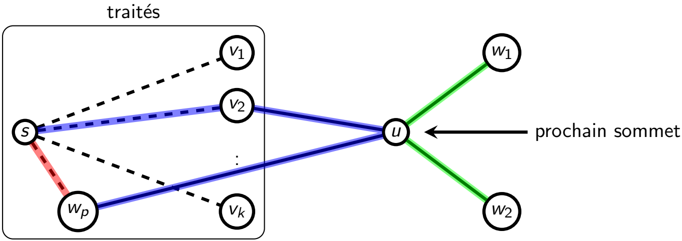
Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



Description de l'algorithme de Dijkstra

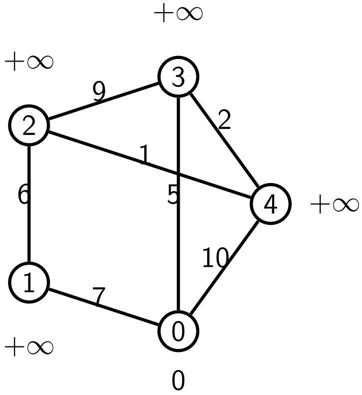
- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



- **Attention :** on ne traite chaque sommet qu'une fois, mais la distance d'un sommet déjà traité peut changer !

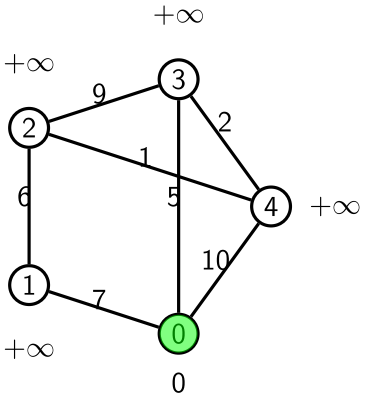
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



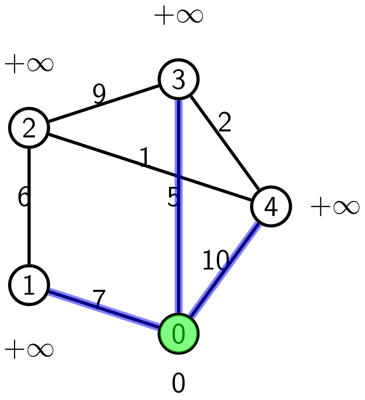
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



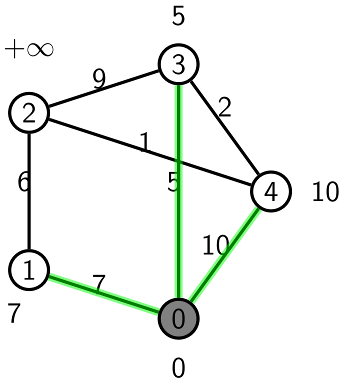
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



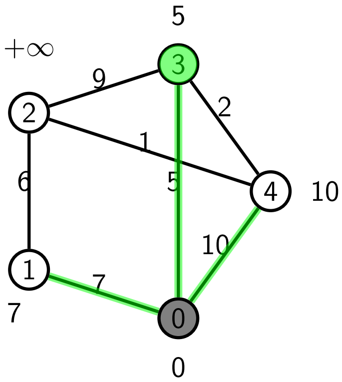
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



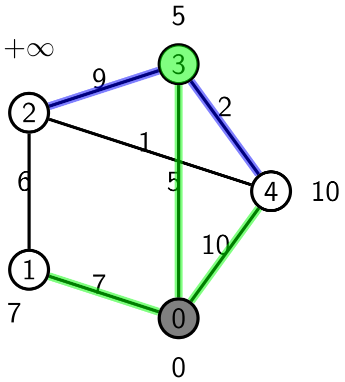
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



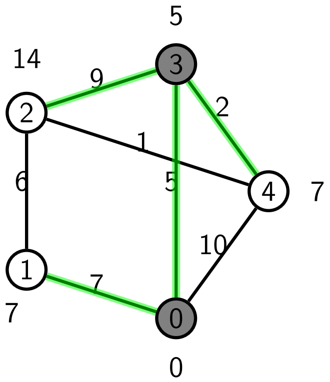
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



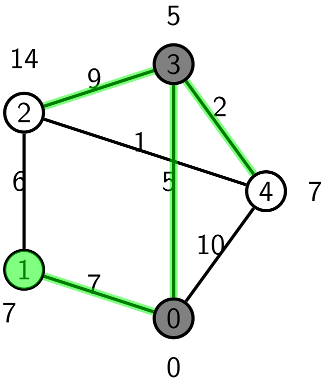
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



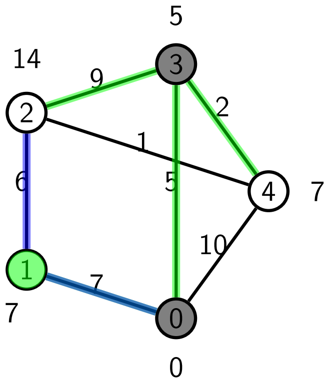
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



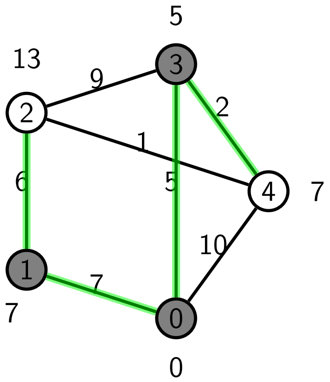
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



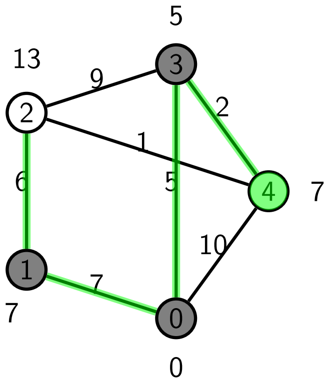
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



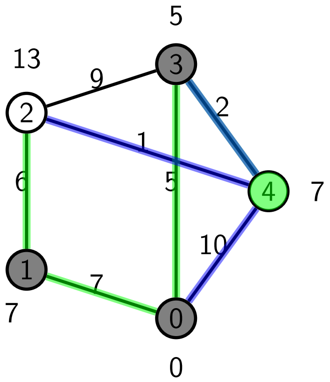
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



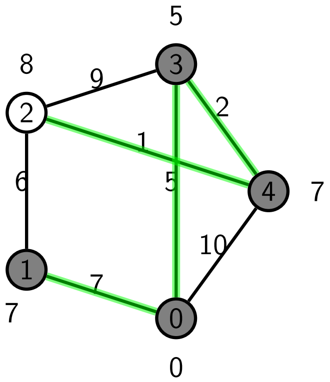
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



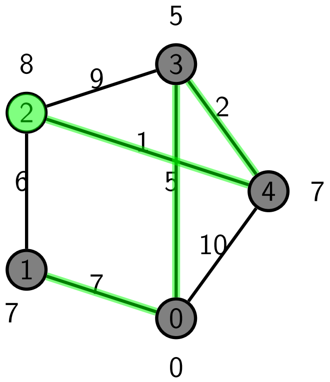
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



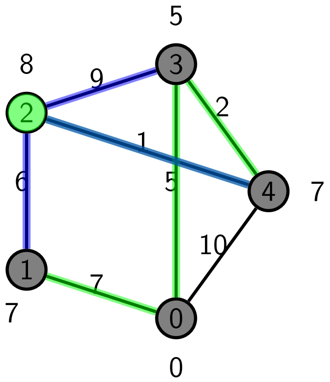
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



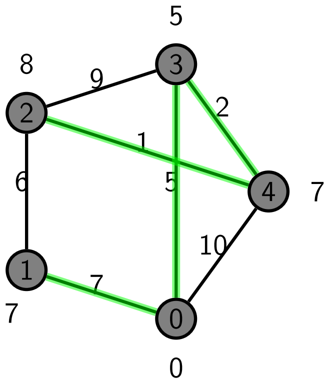
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



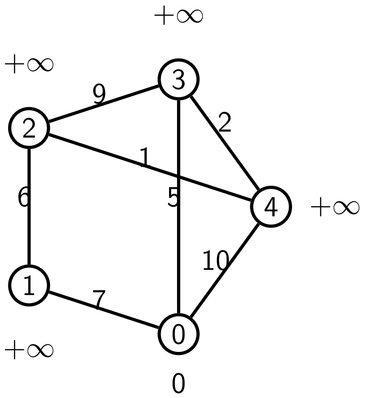
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



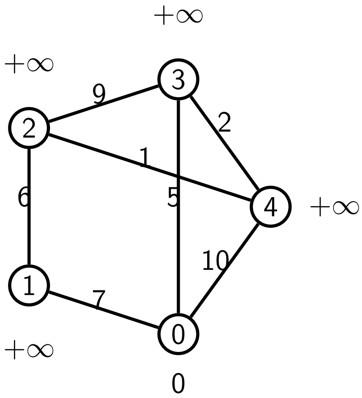
Les coulisses

distances :

étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source $= 0$)



Les coulisses

distances :

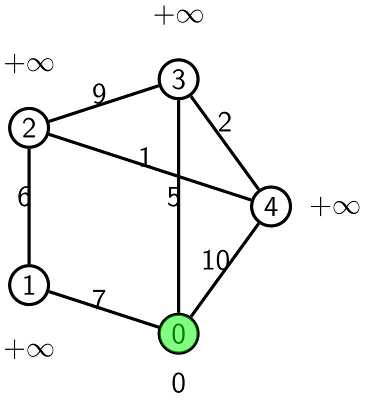
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

parents :

0	1	2	3	4

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

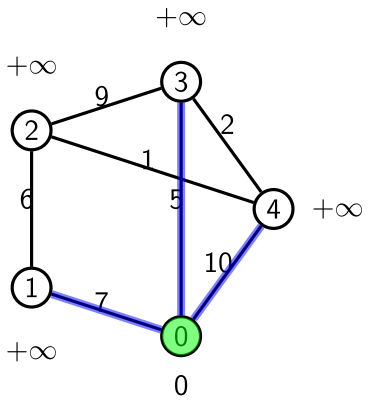
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

parents :

0	1	2	3	4

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

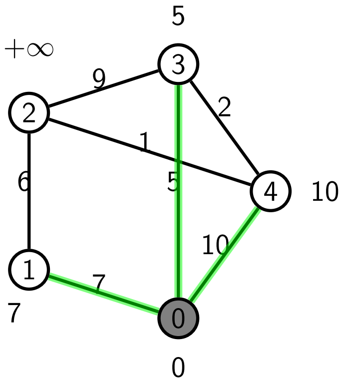
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

parents :

0	1	2	3	4

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

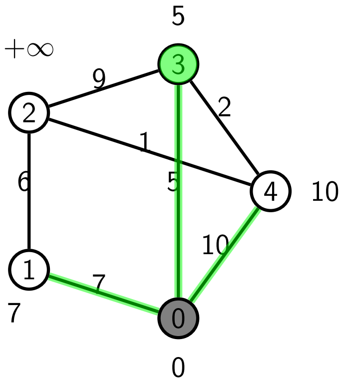
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10

parents :

0	1	2	3	4
	0		0	0

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

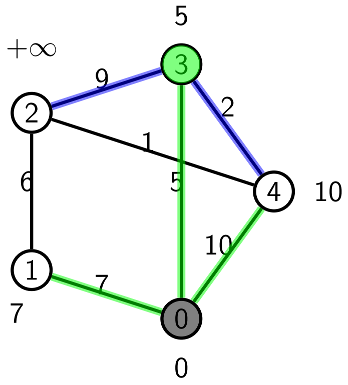
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10

parents :

0	1	2	3	4
	0		0	0

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

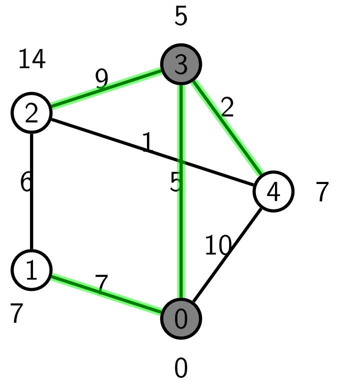
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10

parents :

0	1	2	3	4
	0		0	0

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

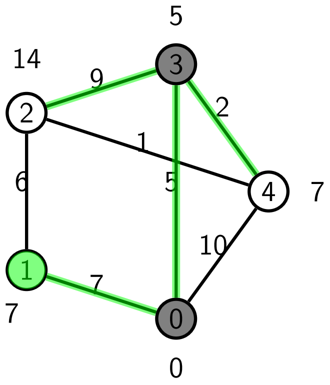
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

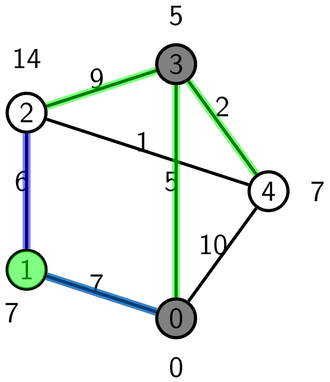
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

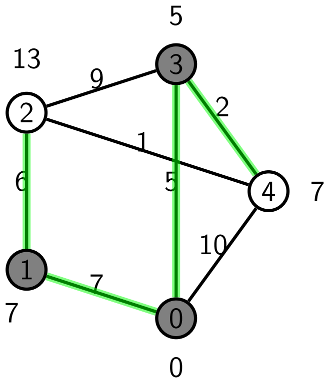
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

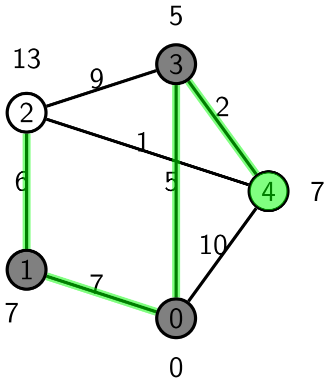
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	13	5	7

parents :

0	1	2	3	4
	0	1	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

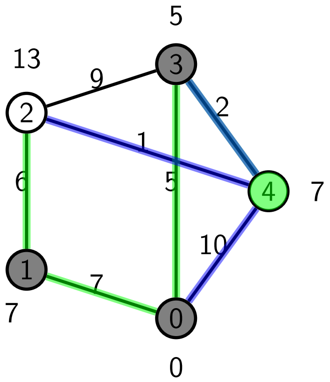
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	13	5	7

parents :

0	1	2	3	4
	0	1	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

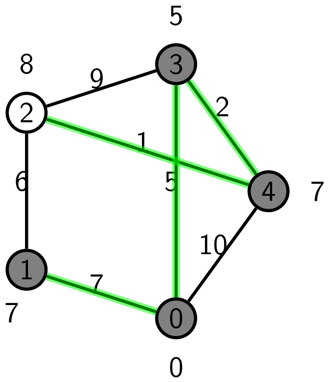
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	13	5	7

parents :

0	1	2	3	4
	0	1	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

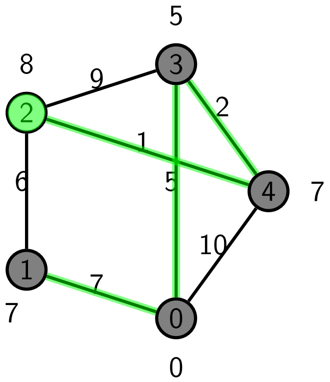
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	13	5	7
(e)	0	7	8	5	7

parents :

0	1	2	3	4
	0	4	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

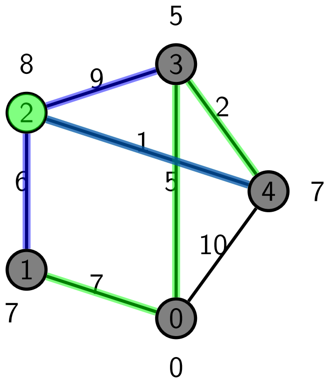
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	13	5	7
(e)	0	7	8	5	7

parents :

0	1	2	3	4
	0	4	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

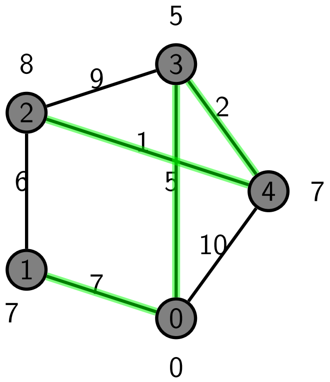
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	13	5	7
(e)	0	7	8	5	7

parents :

0	1	2	3	4
	0	4	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	13	5	7
(e)	0	7	8	5	7
(f)	0	7	8	5	7

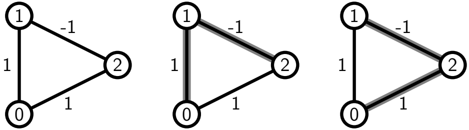
parents :

0	1	2	3	4
	0	4	0	3

Poids négatifs

L'algorithme de Dijkstra ne fonctionne (en général) pas s'il y a des arêtes de poids négatif. s'attendre à ce que l'algorithme fonctionne.

Exemple 3 (plus courts chemins depuis 0)



(1) (2)

1. la solution "rate" le plus court chemin 0–2–1 ;
2. la solution "rate" le plus court chemin 0–1–2 ;

Extraction du sommet le plus proche

L'extraction du minimum peut se faire à l'aide d'un algorithme naïf :

Algorithme 1 : EXTRAIRESOMMETLEPLUSPROCHE(S, distances)

Entrées : un ensemble S de sommets, la distance de chaque sommet de S.

Résultat : le sommet de S le plus proche est extrait et renvoyé.

```
sommet ← NIL;
distance_min ← +∞;
pour chaque candidat ∈ S faire
    si distances[candidat] < distance_min alors
        sommet ← candidat;
        distance_min ← distances[candidat];
si sommet ≠ NIL alors  S ← S\ sommet ;
renvoyer sommet;
```

Extraction du sommet le plus proche

L'extraction du minimum peut se faire à l'aide d'un algorithme naïf :

Algorithme 1 : ExtraireSommetLePlusProche(S, distances)

Entrées : un ensemble S de sommets, la distance de chaque sommet de S.

Résultat : le sommet de S le plus proche est extrait et renvoyé.

```
sommet ← NIL;
distance_min ← +∞;
pour chaque candidat ∈ S faire
    si distances[candidat] < distance_min alors
        sommet ← candidat;
        distance_min ← distances[candidat];
si sommet ≠ NIL alors  S ← S\ sommet ;
renvoyer sommet;
```

Un tas comme pour Prim serait plus efficace ; mais attention, ici, les poids des éléments changent en cours d'exécution !

L'algorithme de Dijkstra proprement dit

Algorithme 2 : DIJKSTRA(G, source)

Entrées : un graphe pondéré non orienté G, un sommet source.
Sortie : la longueur d'un plus court chemin de la source à chacun des sommets du graphe ($+\infty$ pour les sommets non accessibles).

a_traiter $\leftarrow$ G.sommets();
distances $\leftarrow$ tableau(G.nombre_sommets(), $+\infty$);
distances[source] $\leftarrow$ 0;
tant que *a_traiter.pas_vide()* **faire**
    $u \leftarrow$ EXTRAIRESOMMETLEPLUSPROCHE(a_traiter, distances);
    si $u =$ NIL **alors** **renvoyer** distances ;
    pour chaque $v \in G.voisins(u)$ **faire**
        distances[v] $\leftarrow$ min(distances[v], distances[u] + G.poids_arête(u, v));
renvoyer distances;

Complexité

- On passe $O(|V|)$ fois dans la boucle principale ;

Complexité

- On passe $O(|V|)$ fois dans la boucle principale ;
- Extraire chaque sommet coûte $O(|S|) = O(|V|)$;

Complexité

- On passe $O(|V|)$ fois dans la boucle principale ;
- Extraire chaque sommet coûte $O(|S|) = O(|V|)$;
- On examine chaque arête deux fois ;

$$\Rightarrow O(|E| + |V|^2) = O(|V|^2).$$

Complexité

- On passe $O(|V|)$ fois dans la boucle principale ;
- Extraire chaque sommet coûte $O(|S|) = O(|V|)$;
- On examine chaque arête deux fois ;

$$\Rightarrow O(|E| + |V|^2) = O(|V|^2).$$

- Une structure de tas adaptée permet de rabaisser la complexité à $O(|E| + |V| \log |V|)$;

Motivations

- Certains réseaux sont naturellement asymétriques (Twitter, réseaux routiers, graphes de dépendances, généalogies, ...);

Motivations

- Certains réseaux sont naturellement asymétriques (Twitter, réseaux routiers, graphes de dépendances, généalogies, ...) ;
- Il faut donc pouvoir représenter des graphes avec des liens asymétriques ;

Motivations

- Certains réseaux sont naturellement asymétriques (Twitter, réseaux routiers, graphes de dépendances, généalogies, ...) ;
- Il faut donc pouvoir représenter des graphes avec des liens asymétriques ;
- Ces *graphes orientés* contiennent des *arcs* (u, v) au lieu d'arêtes $\{u, v\}$: la présence d'un lien dans un sens n'implique donc plus la présence de ce lien dans l'autre sens ;

Motivations

- Certains réseaux sont naturellement asymétriques (Twitter, réseaux routiers, graphes de dépendances, généalogies, ...) ;
- Il faut donc pouvoir représenter des graphes avec des liens asymétriques ;
- Ces *graphes orientés* contiennent des *arcs* (u, v) au lieu d'arêtes $\{u, v\}$: la présence d'un lien dans un sens n'implique donc plus la présence de ce lien dans l'autre sens ;
- On combinera plus tard poids et orientation ;

Concepts de base

Définition 1

Un **graphe orienté** est un couple $G = (V, A)$, où :

- V est un ensemble de **sommets** ;
- $A \subseteq V \times V$ un ensemble d'**arcs** ;

On utilisera aussi les notations $V(G)$ et $A(G)$ pour bien distinguer le graphe G d'un autre graphe.

Concepts de base

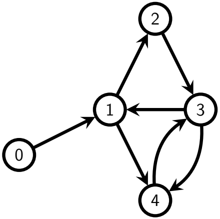
Définition 1

Un **graphe orienté** est un couple $G = (V, A)$, où :

- V est un ensemble de **sommets** ;
- $A \subseteq V \times V$ un ensemble d'**arcs** ;

On utilisera aussi les notations $V(G)$ et $A(G)$ pour bien distinguer le graphe G d'un autre graphe.

Exemple 4



Concepts de base

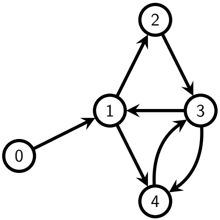
Définition 1

Un **graphe orienté** est un couple $G = (V, A)$, où :

- V est un ensemble de **sommets** ;
- $A \subseteq V \times V$ un ensemble d'**arcs** ;

On utilisera aussi les notations $V(G)$ et $A(G)$ pour bien distinguer le graphe G d'un autre graphe.

Exemple 4



- Un arc (u, v) relie une **source** à une **destination** ; on dit aussi que u est le **prédécesseur** de v, qui est le **successeur** de u ;

Concepts de base

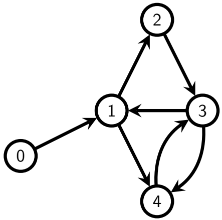
Définition 1

Un **graphe orienté** est un couple $G = (V, A)$, où :

- V est un ensemble de **sommets** ;
- $A \subseteq V \times V$ un ensemble d'**arcs** ;

On utilisera aussi les notations $V(G)$ et $A(G)$ pour bien distinguer le graphe G d'un autre graphe.

Exemple 4



- Un arc (u, v) relie une **source** à une **destination** ; on dit aussi que u est le **prédécesseur** de v, qui est le **successeur** de u ;
- Le **voisinage** d'un sommet v est l'union de ses prédécesseurs et de ses successseurs :

$$N_G(v) = N_G^-(v) \cup N_G^+(v) = \{u \mid (u, v) \in A(G)\} \cup \{w \mid (v, w) \in A(G)\}$$

Concepts de base

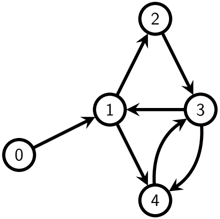
Définition 1

Un **graphe orienté** est un couple $G = (V, A)$, où :

- V est un ensemble de **sommets** ;
- $A \subseteq V \times V$ un ensemble d'**arcs** ;

On utilisera aussi les notations $V(G)$ et $A(G)$ pour bien distinguer le graphe G d'un autre graphe.

Exemple 4



- Un arc (u, v) relie une **source** à une **destination** ; on dit aussi que u est le **prédécesseur** de v, qui est le **successeur** de u ;
- Le **voisinage** d'un sommet v est l'union de ses prédécesseurs et de ses successseurs :

$$N_G(v) = N_G^-(v) \cup N_G^+(v) = \{u \mid (u, v) \in A(G)\} \cup \{w \mid (v, w) \in A(G)\}$$

- Le **degré entrant** du sommet v est $\deg_G^-(v) = |N_G^-(v)|$;

Concepts de base

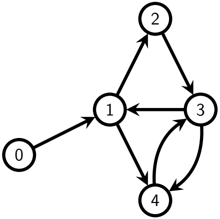
Définition 1

Un **graphe** **orienté** est un couple $G = (V, A)$, où :

- V est un ensemble de **sommets** ;
- $A \subseteq V \times V$ un ensemble d'**arcs** ;

On utilisera aussi les notations $V(G)$ et $A(G)$ pour bien distinguer le graphe G d'un autre graphe.

Exemple 4



- Un arc (u, v) relie une **source** à une **destination** ; on dit aussi que u est le **prédécesseur** de v, qui est le **successeur** de u ;
- Le **voisinage** d'un sommet v est l'union de ses prédécesseurs et de ses successseurs :

$$N_G(v) = N_G^-(v) \cup N_G^+(v) = \{u \mid (u, v) \in A(G)\} \cup \{w \mid (v, w) \in A(G)\}$$

- Le **degré entrant** du sommet v est $\deg_G^-(v) = |N_G^-(v)|$;
- Le **degré sortant** du sommet v est $\deg_G^+(v) = |N_G^+(v)|$;

Adaptation des concepts non orientés

Les définitions qu'on a utilisées jusqu'ici doivent également être adaptées pour prendre en compte l'orientation des arcs quand c'est nécessaire. Par exemple :

- un **chemin** dans un graphe **orienté** $G = (V, A)$ est une séquence de sommets **distincts** $P = (u_0, u_1, u_2, \ldots, u_{p-1})$, où $(u_i, u_{i+1}) \in A$ pour $0 \leq i \leq p - 2$;

Adaptation des concepts non orientés

Les définitions qu'on a utilisées jusqu'ici doivent également être adaptées pour prendre en compte l'orientation des arcs quand c'est nécessaire. Par exemple :

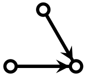
- un **chemin** dans un graphe **orienté** $G = (V, A)$ est une séquence de sommets **distincts** $P = (u_0, u_1, u_2, \ldots, u_{p-1})$, où $(u_i, u_{i+1}) \in A$ pour $0 \leq i \leq p-2$;
- un **cycle** dans un graphe **orienté** $G = (V, A)$ est une séquence de sommets $C = (u_0, u_1,\ u_2,\ \ldots,\ u_{p-1})$, où $(u_i, u_{i+1 \mod p}) \in A$ pour $0 \leq i \leq p-1$.

Adaptation des concepts non orientés

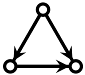
Les définitions qu'on a utilisées jusqu'ici doivent également être adaptées pour prendre en compte l'orientation des arcs quand c'est nécessaire. Par exemple :

- un **chemin** dans un graphe **orienté** $G = (V, A)$ est une séquence de sommets **distincts** $P = (u_0, u_1, u_2, \ldots, u_{p-1})$, où $(u_i, u_{i+1}) \in A$ pour $0 \leq i \leq p-2$;
- un **cycle** dans un graphe **orienté** $G = (V, A)$ est une séquence de sommets $C = (u_0, u_1, u_2, \ldots, u_{p-1})$, où $(u_i, u_{i+1 \mod p}) \in A$ pour $0 \leq i \leq p-1$.

Exemple 5



ceci n'est pas un chemin



ceci n'est pas un cycle

Adaptation des concepts non orientés

Les définitions qu'on a utilisées jusqu'ici doivent également être adaptées pour prendre en compte l'orientation des arcs quand c'est nécessaire. Par exemple :

- un **chemin** dans un graphe **orienté** $G = (V, A)$ est une séquence de sommets **distincts** $P = (u_0, u_1, u_2, \ldots, u_{p-1})$, où $(u_i, u_{i+1}) \in A$ pour $0 \leq i \leq p-2$;
- un **cycle** dans un graphe **orienté** $G = (V, A)$ est une séquence de sommets $C = (u_0, u_1, u_2, \ldots, u_{p-1})$, où $(u_i, u_{i+1 \bmod p}) \in A$ pour $0 \leq i \leq p-1$.

Exemple 5

ceci n'est pas un chemin ceci n'est pas un cycle

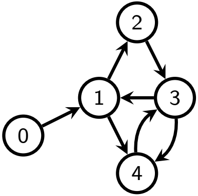
- un **descendant** d'un sommet u dans un graphe orienté G est un sommet $v \neq u$ tel qu'il existe un chemin (**orienté**) de u vers v dans G.

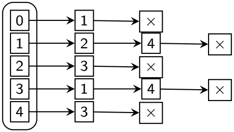
Implémentation des graphes orientés

- Les implémentations du cas non orienté s'adaptent facilement et se simplifient même :
  - matrice d'adjacence : asymétrique, un seul ajout ;
  - listes d'adjacence : successeurs, un seul ajout ;

Exemple 6

	0	1	2	3	4
0		1			
1			1		1
2				1	
3		1			1
4				1	





L'API `GrapheOrienté`

Les algorithmes étudiés supposeront l'existence d'une classe `Graphe` proposant les méthodes suivantes :

Méthode
`ajouter_arc(u, v)`
`ajouter_arcs(séquence)`
`ajouter_sommet(sommet)`
`ajouter_sommets(séquence)`
`arcs()`
`boucles()`
`contient_arc(u, v)`
`contient_sommet(u)`
`degre(sommet)`
`degre_entrant(sommet)`
`degre_sortant(sommet)`
`nombre_arcs()`
`nombre_boucles()`
`nombre_sommets()`

Méthode
`predecesseurs(sommet)`
`retirer_arc(u, v)`
`retirer_arcs(séquence)`
`retirer_sommet(sommet)`
`retirer_sommets(séquence)`
`sommets()`
`sous_graphe_induit(séquence)`
`successeurs(sommet)`
`voisins(sommet)`

Parcours dans le cas orienté

- Dans le cas non-orienté, on était sûr d'atteindre tous les sommets du graphe accessibles à partir du sommet de départ ;

Parcours dans le cas orienté

- Dans le cas non-orienté, on était sûr d'atteindre tous les sommets du graphe accessibles à partir du sommet de départ ;
- C'est aussi le cas dans le cas orienté, **mais** le fait que le graphe est "en un seul morceau" ne garantit plus qu'on puisse atteindre tous ses sommets !

Parcours dans le cas orienté

- Dans le cas non-orienté, on était sûr d'atteindre tous les sommets du graphe accessibles à partir du sommet de départ ;
- C'est aussi le cas dans le cas orienté, **mais** le fait que le graphe est "en un seul morceau" ne garantit plus qu'on puisse atteindre tous ses sommets !
- La notion de composante connexe devra donc être adaptée, et la détection sera plus complexe ;

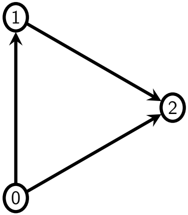
Parcours dans le cas orienté

- Dans le cas non-orienté, on était sûr d'atteindre tous les sommets du graphe accessibles à partir du sommet de départ ;
- C'est aussi le cas dans le cas orienté, **mais** le fait que le graphe est “en un seul morceau” ne garantit plus qu'on puisse atteindre tous ses sommets !
- La notion de composante connexe devra donc être adaptée, et la détection sera plus complexe ;
- Pour adapter les parcours en largeur et en profondeur : il suffit de remplacer les appels à G.voisins(v) par G.successeurs(v) ;

Détection de cycles orientés

- Dans le cas non orienté, on détectait un cycle en vérifiant si un sommet accessible avait déjà un parent ;

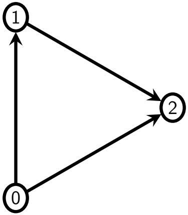
Exemple 7



Détection de cycles orientés

- Dans le cas non orienté, on détectait un cycle en vérifiant si un sommet accessible avait déjà un parent ;
- Cela ne marche plus dans le cas orienté : un graphe orienté acyclique peut contenir plusieurs chemins de u à v ;

Exemple 7



Détection de cycles orientés

- Le parcours en profondeur permet la détection de cycles orientés, mais il faut l'adapter ;

Détection de cycles orientés

- Le parcours en profondeur permet la détection de cycles orientés, mais il faut l'adapter ;
- On doit pouvoir faire la différence entre un sommet :

Détection de cycles orientés

- Le parcours en profondeur permet la détection de cycles orientés, mais il faut l'adapter ;
- On doit pouvoir faire la différence entre un sommet :

  1. $\boxed{\text{non exploré}}$: on ne l'a jamais vu ;

Détection de cycles orientés

- Le parcours en profondeur permet la détection de cycles orientés, mais il faut l'adapter ;
- On doit pouvoir faire la différence entre un sommet :

1. non exploré : on ne l'a jamais vu ;
2. en cours de traitement : on est en train d'explorer ses descendants ;

Détection de cycles orientés

- Le parcours en profondeur permet la détection de cycles orientés, mais il faut l'adapter ;
- On doit pouvoir faire la différence entre un sommet :

1. non exploré : on ne l'a jamais vu ;
2. en cours de traitement : on est en train d'explorer ses descendants ;
3. exploré : on a exploré tous ses descendants ;

Détection de cycles orientés

- Le parcours en profondeur permet la détection de cycles orientés, mais il faut l'adapter ;
- On doit pouvoir faire la différence entre un sommet :
  1. non exploré : on ne l'a jamais vu ;
  2. en cours de traitement : on est en train d'explorer ses descendants ;
  3. exploré : on a exploré tous ses descendants ;
- D'où le nom de "parcours tricolore" ;

Détection de cycles orientés

- En quoi ces couleurs aident-elles ?

Détection de cycles orientés

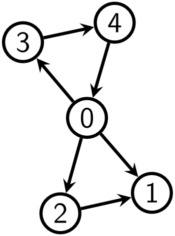
- En quoi ces couleurs aident-elles ?
- Si en partant d'un sommet u, on tombe sur un sommet v exploré, on en conclut que plusieurs chemins de u à v existent ;

Détection de cycles orientés

- En quoi ces couleurs aident-elles ?
- Si en partant d'un sommet u, on tombe sur un sommet v exploré, on en conclut que plusieurs chemins de u à v existent ;
- Si en partant d'un sommet u, on tombe sur un sommet w en cours de traitement, alors on a un cycle ;

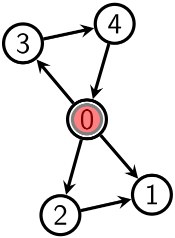
Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



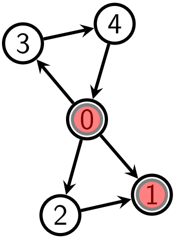
Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



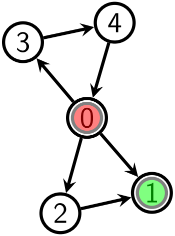
Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



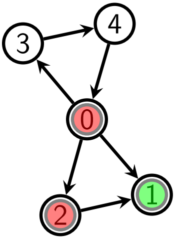
Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



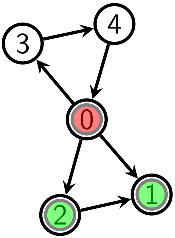
Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



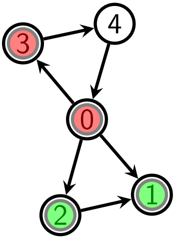
Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



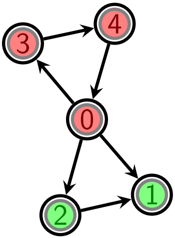
Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



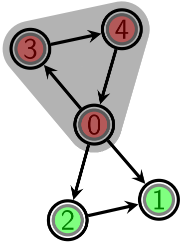
Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



Détection de cycles orientés

Exemple 8 (parcours tricolore au départ de 0)



L'algorithme de détection d'un cycle orienté

Une légère adaptation du parcours en profondeur récursif nous permet d'arriver à nos fins :

Algorithme 3 : CONTIENTCYCLEORIENTÉ(G, sommet, statuts)

Entrées : un graphe orienté G, un sommet de départ, et un tableau statuts de $|V|$ cases.

Sortie : VRAI si un cycle de G est accessible à partir du sommet de départ, FAUX sinon.

si *statuts[sommet]* $=$ *"rouge"* **alors** **renvoyer** VRAI;
si *statuts[sommet]* $=$ *"vert"* **alors** **renvoyer** FAUX;
statuts[sommet] $\leftarrow$ "rouge";
pour chaque $v \in G$*.successeurs(sommet)* **faire**
    si CONTIENTCYCLEORIENTÉ*(G, v, statuts)* **alors** **renvoyer** VRAI;
statuts[sommet] $\leftarrow$ "vert";
renvoyer FAUX;

Remarques sur la détection de cycles orientés

- **Attention :** un retour FAUX ne veut **pas** dire que G est acyclique, mais bien qu'aucun cycle n'est accessible au départ du sommet donné !

Remarques sur la détection de cycles orientés

- **Attention :** un retour FAUX ne veut **pas** dire que G est acyclique, mais bien qu'aucun cycle n'est accessible au départ du sommet donné !
- Faute de mieux, on relancera le parcours à partir de chaque sommet non encore exploré pour avoir la "vraie" réponse ;

Remarques sur la détection de cycles orientés

- **Attention :** un retour FAUX ne veut **pas** dire que G est acyclique, mais bien qu'aucun cycle n'est accessible au départ du sommet donné !
- Faute de mieux, on relancera le parcours à partir de chaque sommet non encore exploré pour avoir la "vraie" réponse ;
- Pour reconstruire le cycle, on a besoin du tableau `parents` (cf. cas non orienté) ;

Motivation

- Il est parfois utile d'enrichir la structure d'un graphe de manière à pouvoir répondre à certaines requêtes sur ce graphe plus rapidement ;

Motivation

- Il est parfois utile d'enrichir la structure d'un graphe de manière à pouvoir répondre à certaines requêtes sur ce graphe plus rapidement ;
- Par exemple : calcul répété de tous les descendants d'un sommet ;

Motivation

- Il est parfois utile d'enrichir la structure d'un graphe de manière à pouvoir répondre à certaines requêtes sur ce graphe plus rapidement ;
- Par exemple : calcul répété de tous les descendants d'un sommet ;

Définition 2

La **fermeture transitive** d'un graphe orienté $G = (V, A)$ est le graphe orienté $G' = (V, A')$ avec $A' = \{(u, v) \mid v \in \text{ descendants}(u) \text{ dans } G\}$.

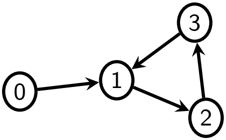
Motivation

- Il est parfois utile d'enrichir la structure d'un graphe de manière à pouvoir répondre à certaines requêtes sur ce graphe plus rapidement ;
- Par exemple : calcul répété de tous les descendants d'un sommet ;

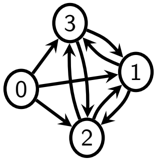
Définition 2

La **fermeture transitive** d'un graphe orienté $G = (V, A)$ est le graphe orienté $G' = (V, A')$ avec $A' = \{(u, v) \mid v \in \text{ descendants}(u) \text{ dans } G\}$.

Exemple 9



G



fermeture transitive de G

Calcul de la fermeture transitive

- Idée simple : rajouter un arc connectant chaque sommet à tous ses descendants ;

Calcul de la fermeture transitive

- Idée simple : rajouter un arc connectant chaque sommet à tous ses descendants ;
- Les descendants se calculent avec un simple parcours ;

Calcul de la fermeture transitive

- Idée simple : rajouter un arc connectant chaque sommet à tous ses descendants ;
- Les descendants se calculent avec un simple parcours ;

Algorithme 4 : $\textsc{FermetureTransitive}(G)$

Entrées : un graphe orienté connexe G.
Sortie : la fermeture transitive de G.

$F \leftarrow$ GrapheOrienté(G.sommets());
pour chaque $u \in G$*.sommets()* **faire**
  pour chaque $v \in \textsc{LargeurOrienté}(F, u)$ **faire**
    si $u \neq v$ **alors** F.ajouter_arc(u, v) ;
renvoyer F;

Graphes orientés en `dot`

- Pour représenter un graphe orienté en `dot`, on utilise :
  - `digraph` au lieu de `graph` ;
  - `->` pour les arcs (au lieu de `--` pour les arêtes) ;
- On ne peut pas mélanger `--` et `->` : soit le graphe est orienté, soit il ne l'est pas ;