Boudjémâa Salah & Max Fillere
Encadrant: Johan Arcile

Travail d'Étude et de Recherche

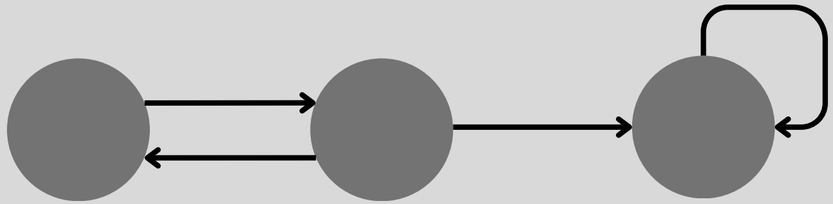
Identification d'états se répétant infiniment souvent dans les structures de Kripke

Sommaire

Structure de Kripke

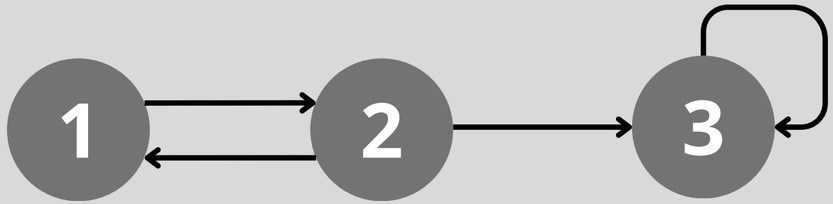
K = (Q, q0, T, AP, L)

Graphe orienté



Structure de Kripke

Graphe orienté

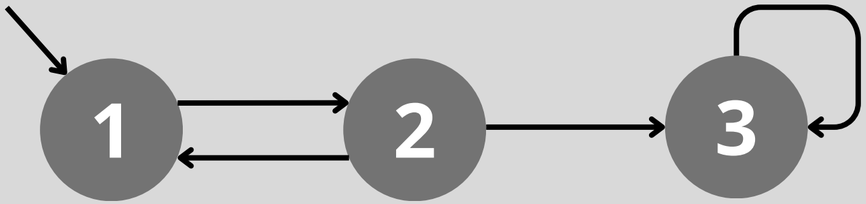


K = (Q, q0, T, AP, L)

Q = {1, 2, 3}

Structure de Kripke

Graphe orienté

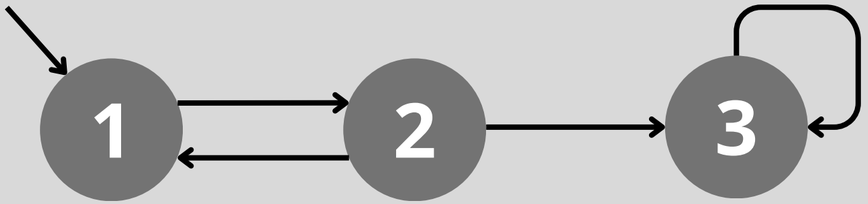


K = (Q, q0, T, AP, L)

Q = {1, 2, 3}
q0 = 1

Structure de Kripke

Graphe orienté



K = (Q, q0, T, AP, L)

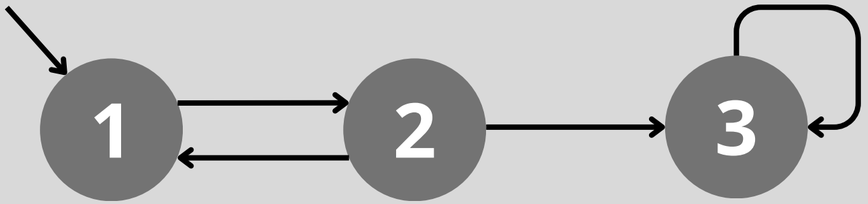
Q = {1, 2, 3}

q0 = 1

T = {(1,2),(2,1),(2,3),(3,3)}

Structure de Kripke

Graphe orienté



K = (Q, q0, T, AP, L)

Q = {1, 2, 3}

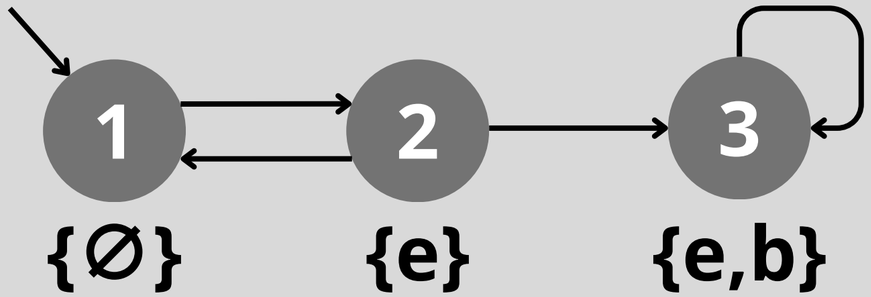
q0 = 1

T = {(1,2),(2,1),(2,3),(3,3)}

AP = {e,b}

Structure de Kripke

Graphe orienté



K = (Q, q0, T, AP, L)

Q = {1, 2, 3}

q0 = 1

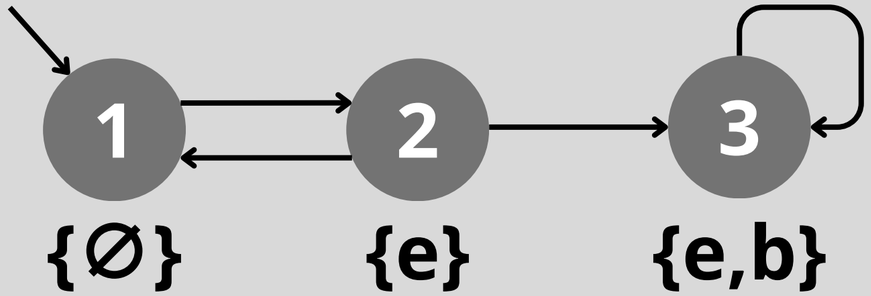
T = {(1,2),(2,1),(2,3),(3,3)}

AP = {e,b}

L = {1 : ø, 2 : {e}, 3 : {e,b} }

Structure de Kripke

Graphe orienté



Chaque état possède au moins une transition sortante

K = (Q, q0, T, AP, L)

Q = {1, 2, 3}

q0 = 1

T = {(1,2),(2,1),(2,3),(3,3)}

AP = {e,b}

L = {1 : ø, 2 : {e}, 3 : {e,b} }

Directed Feedback Vertex Set

Dans un graphe G=(V,E)

V : ensemble des sommets

E : ensemble des arcs

Trouver S ⊆ V minimal
tel que G\S soit acyclique

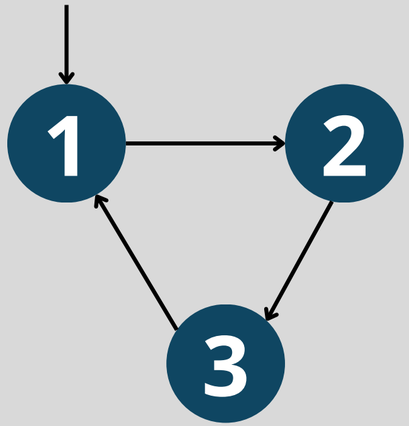
Directed Feedback Vertex Set

Dans un graphe G=(V,E)

V : ensemble des sommets

E : ensemble des arcs

Trouver S ⊆ V minimal
tel que G\S soit acyclique



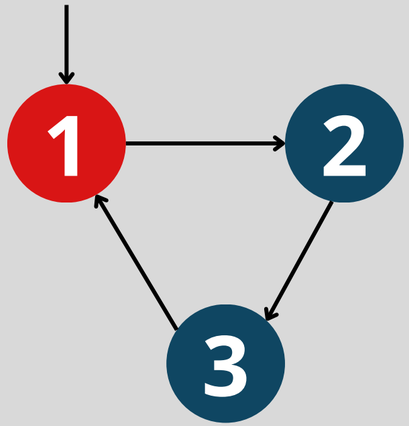
Directed Feedback Vertex Set

Dans un graphe G=(V,E)

V : ensemble des sommets

E : ensemble des arcs

Trouver S ⊆ V minimal
tel que G\S soit acyclique



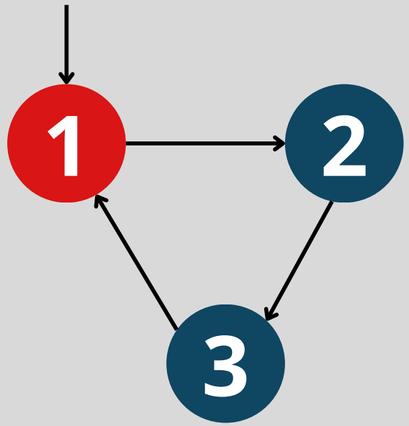
Directed Feedback Vertex Set

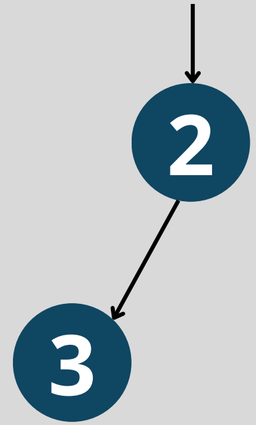
Dans un graphe G=(V,E)

V : ensemble des sommets

E : ensemble des arcs

Trouver S ⊆ V minimal
tel que G\S soit acyclique





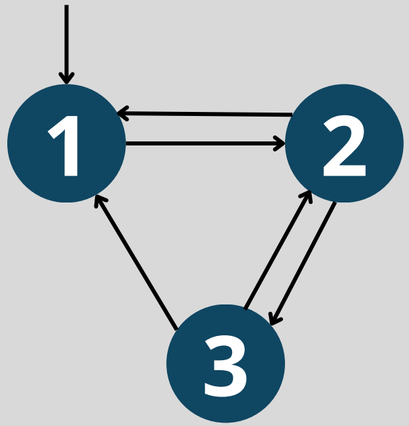
1 est présent infiniment souvent

Directed Feedback Vertex Set

Dans un graphe G=(V,E)

- V : ensemble des sommets
- E : ensemble des arcs

Trouver S ⊆ V minimal
tel que G\S soit acyclique



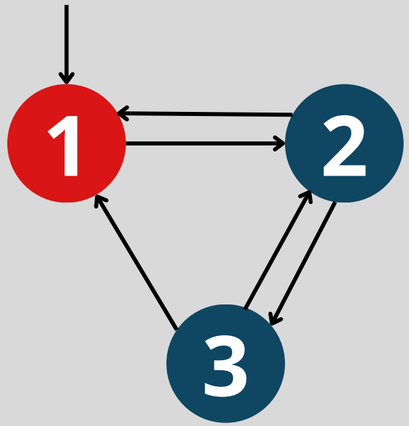
Directed Feedback Vertex Set

Dans un graphe G=(V,E)

V : ensemble des sommets

E : ensemble des arcs

Trouver S ⊆ V minimal
tel que G\S soit acyclique



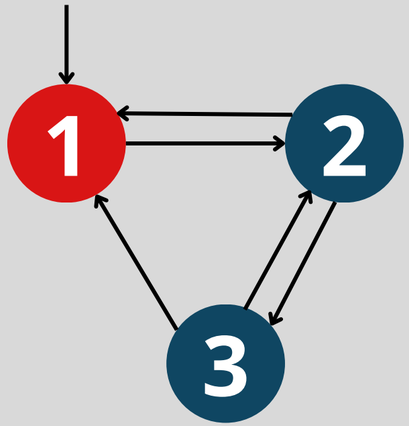
Directed Feedback Vertex Set

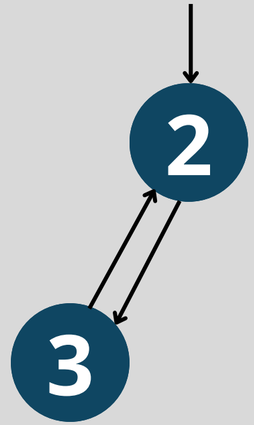
Dans un graphe G=(V,E)

V : ensemble des sommets

E : ensemble des arcs

Trouver S ⊆ V minimal
tel que G\S soit acyclique





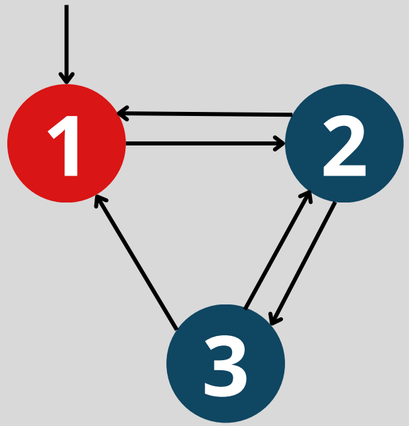
Directed Feedback Vertex Set

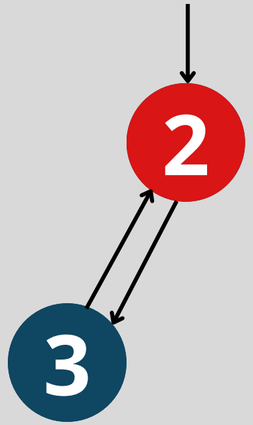
Dans un graphe G=(V,E)

V : ensemble des sommets

E : ensemble des arcs

Trouver S ⊆ V minimal
tel que G\S soit acyclique





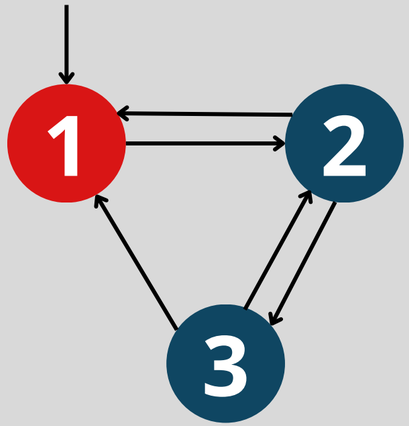
Directed Feedback Vertex Set

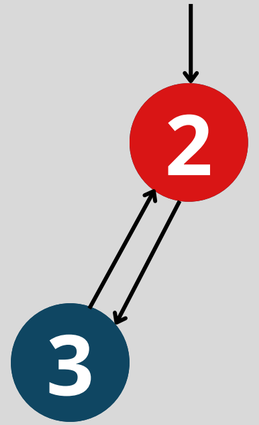
Dans un graphe G=(V,E)

V : ensemble des sommets

E : ensemble des arcs

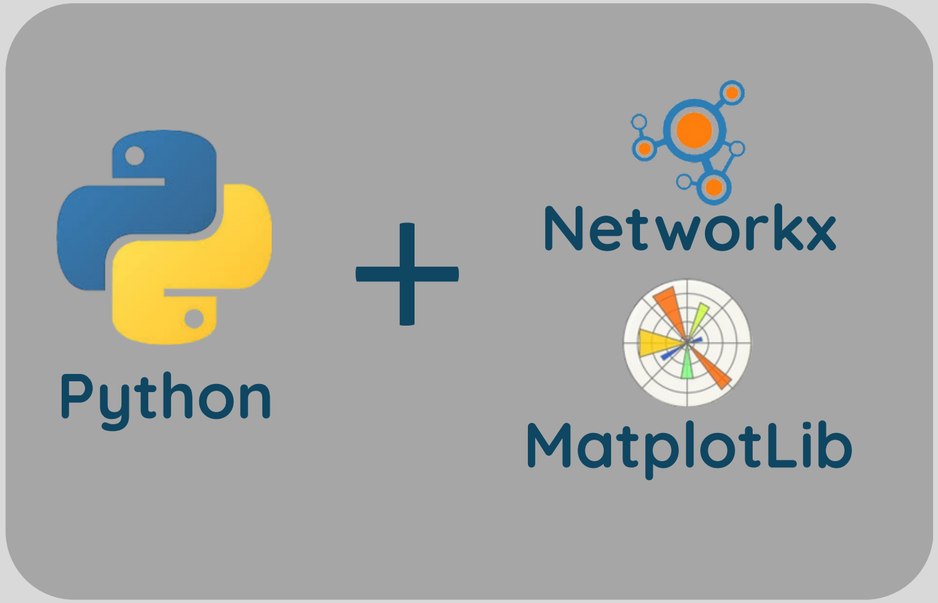
Trouver S ⊆ V minimal
tel que G\S soit acyclique





1 et 2 sont présents infiniment souvent

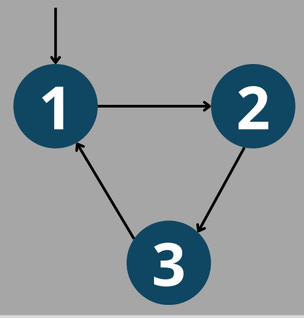
Implémentation



Implémentation

Python + Networkx + MatplotLib

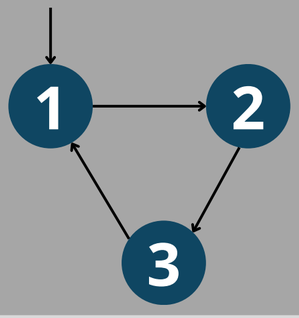
K = (Q, q0, T, AP, L)



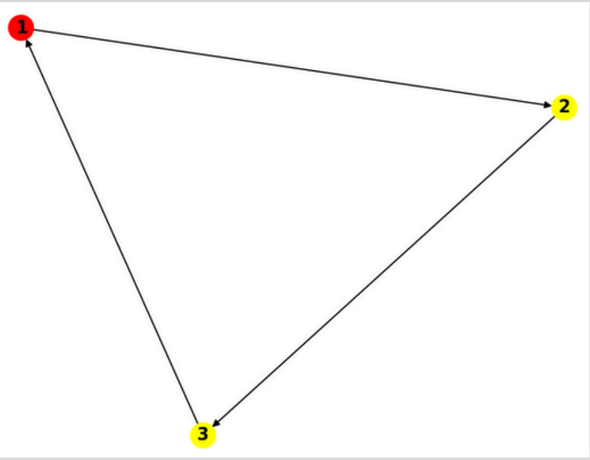
Implémentation

Python + Networkx + MatplotLib

K = (Q, q0, T, AP, L)



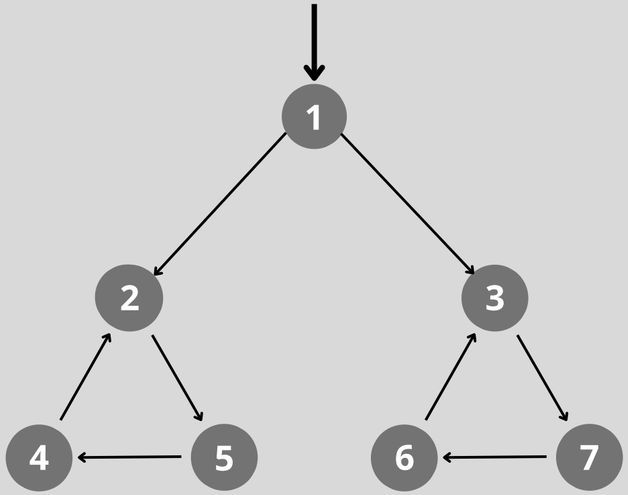
Graphe(
[1,2,3], {1:[2],2:[3],3:[1]},1)



Implémentation

Calcul des cycles

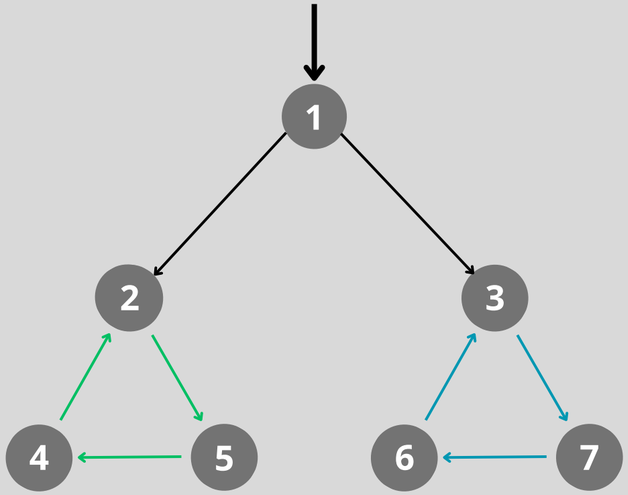
Objectif : Trouver les cycles du graphe

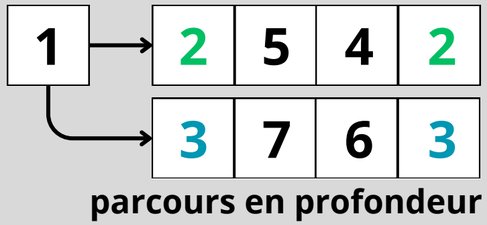


Implémentation

Calcul des cycles

Objectif : Trouver les cycles du graphe



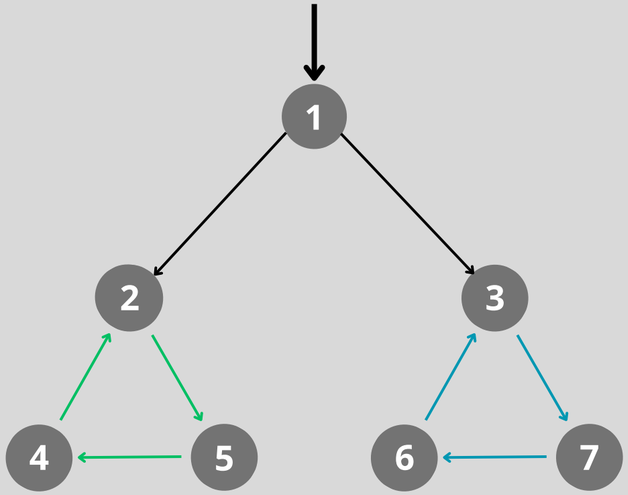


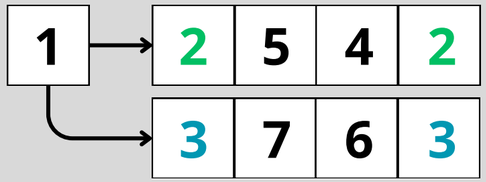
parcours en profondeur

Implémentation

Calcul des cycles

Objectif : Trouver les cycles du graphe





parcours en profondeur

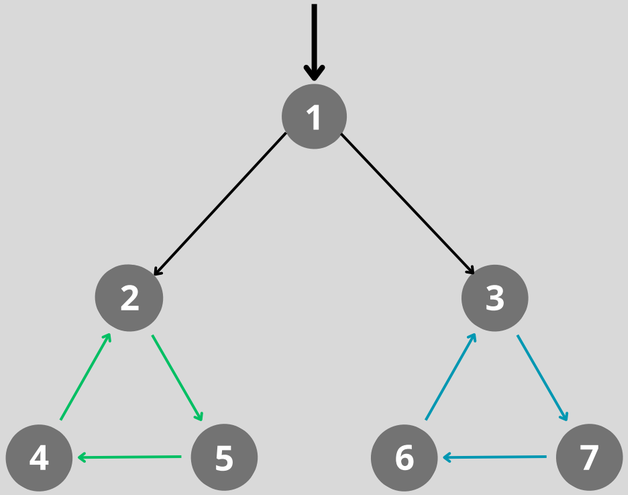
```
Chemin = [etat_initial]
Pile = [(etat_initial,0)]

Tant que pile non vide :
    Dépiler
    (E,i) = élément dépilé
    Si E à au moins i voisins :
        Empiler (E,i+1)
        Si V dans le chemin : # cycle détecté
            Si cycle inconnu :
                Cycles += sous liste du chemin [V:fin]
        Sinon :
            Ajouter V au chemin
            Empiler (V,0)
    Sinon :
        Dépiler le chemin

Retourner Cycles
```

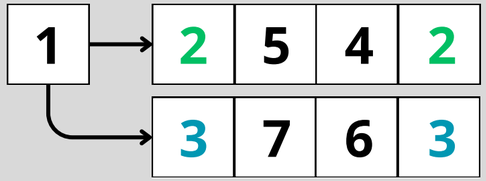
(E, i) :
E: état étudié,
i: rang de son voisin **V** dans sa liste de voisins

Implémentation

Calcul des cycles

Objectif : Trouver les cycles du graphe



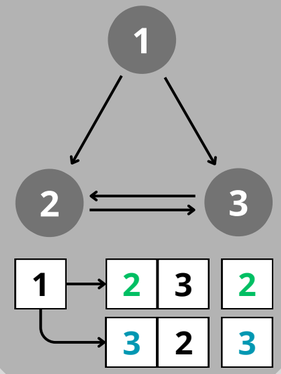


parcours en profondeur

```
Chemin = [etat_initial]
Pile = [(etat_initial,0)]

Tant que pile non vide :
    Dépiler
    (E,i) = élément dépilé
    Si E à au moins i voisins :
        Empiler (E,i+1)
        Si V dans le chemin : # cycle détecté
            Si cycle inconnu :
                Cycles += sous liste du chemin [V:fin]
        Sinon :
            Ajouter V au chemin
            Empiler (V,0)
    Sinon :
        Dépiler le chemin

Retourner Cycles
```

(E, i) :
E: état étudié,
i: rang de son voisin **V** dans sa liste de voisins

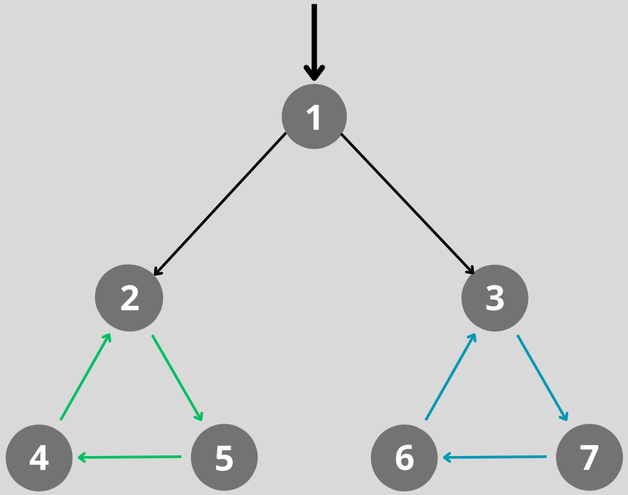
On sélectionne les cycles **uniques** parmi les cycles trouvés.

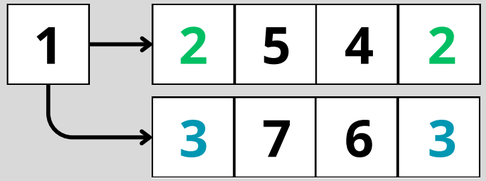


Implémentation

Calcul des cycles

Objectif : Trouver les cycles du graphe



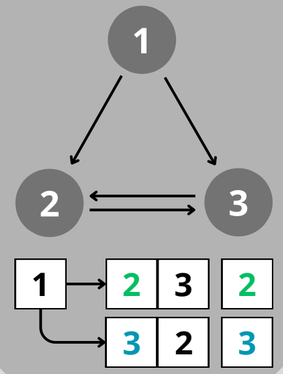


parcours en profondeur

```
Chemin = [etat_initial]
Pile = [(etat_initial,0)]

Tant que pile non vide :
    Dépiler
    (E,i) = élément dépilé
    Si E à au moins i voisins :
        Empiler (E,i+1)
        Si V dans le chemin : # cycle détecté
            Si cycle inconnu :
                Cycles += sous liste du chemin [V:fin]
        Sinon :
            Ajouter V au chemin
            Empiler (V,0)
    Sinon :
        Dépiler le chemin

Retourner Cycles
```

(E, i) :
E: état étudié,
i: rang de son voisin **V** dans sa liste de voisins

On sélectionne les cycles **uniques** parmi les cycles trouvés.



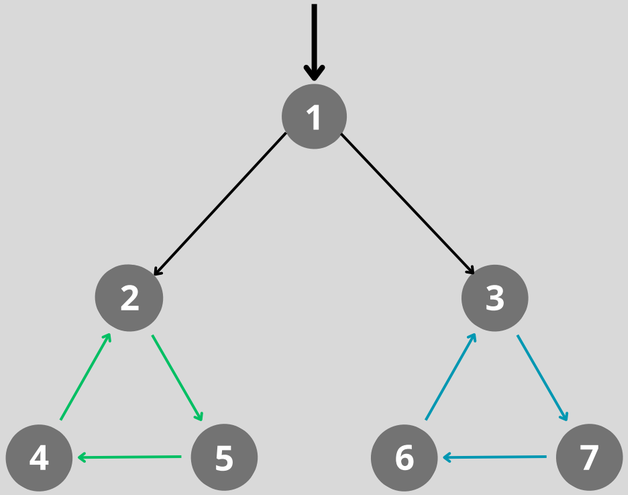
O(n!)

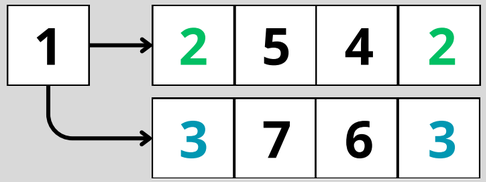
n : nb d'états

Implémentation

Calcul des cycles

Objectif : Trouver les cycles du graphe



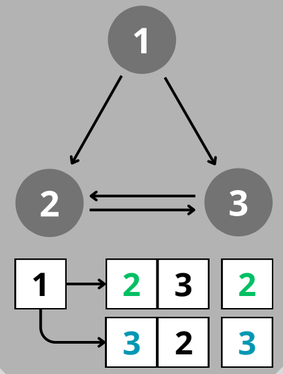


parcours en profondeur

```
Chemin = [etat_initial]
Pile = [(etat_initial,0)]

Tant que pile non vide :
    Dépiler
    (E,i) = élément dépilé
    Si E à au moins i voisins :
        Empiler (E,i+1)
        Si V dans le chemin : # cycle détecté
            Si cycle inconnu :
                Cycles += sous liste du chemin [V:fin]
        Sinon :
            Ajouter V au chemin
            Empiler (V,0)
    Sinon :
        Dépiler le chemin

Retourner Cycles
```

(E, i) :
E: état étudié,
i: rang de son voisin **V** dans sa liste de voisins

On sélectionne les cycles **uniques** parmi les cycles trouvés.



O(n×c)

n : nb d'états
c : nb de cycles

Implémentation

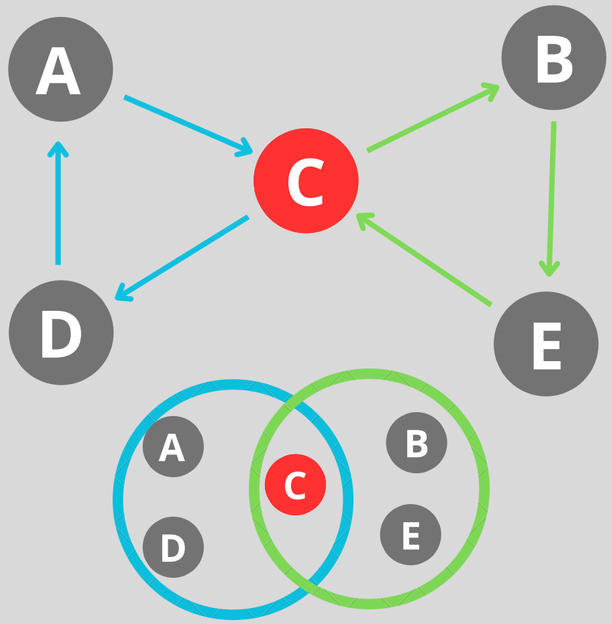
État infini

Objectif : Trouver un état présent infiniment souvent

Implémentation

État infini

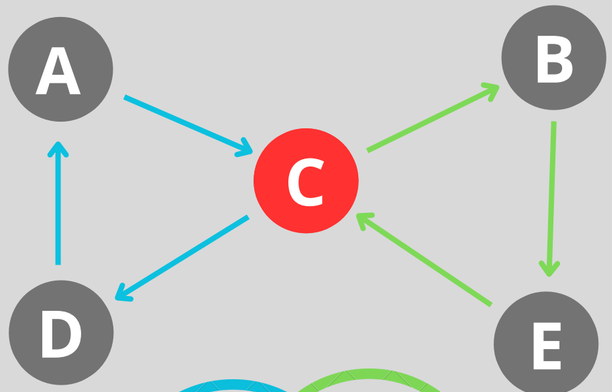
Objectif : Trouver un état présent infiniment souvent

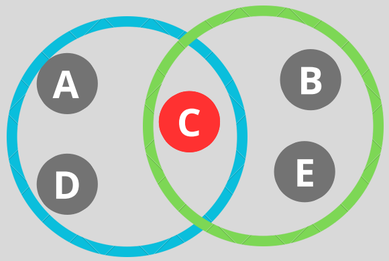


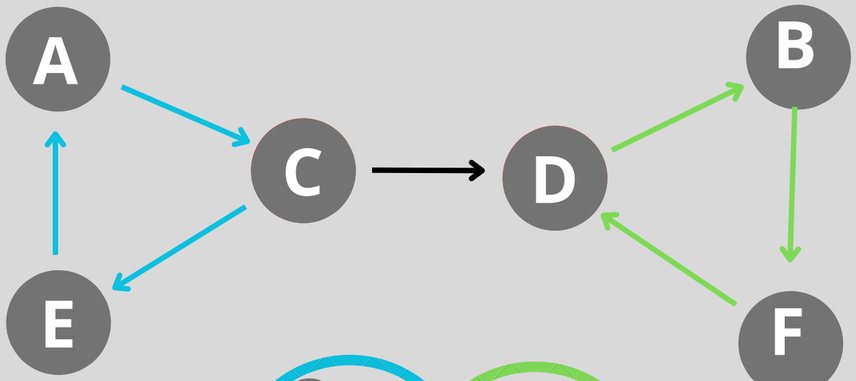
Implémentation

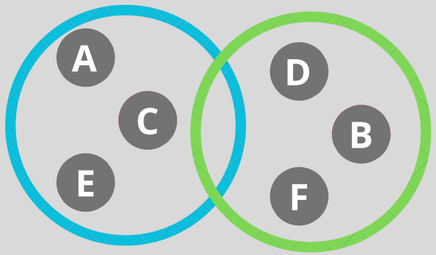
État infini

Objectif : Trouver un état présent infiniment souvent





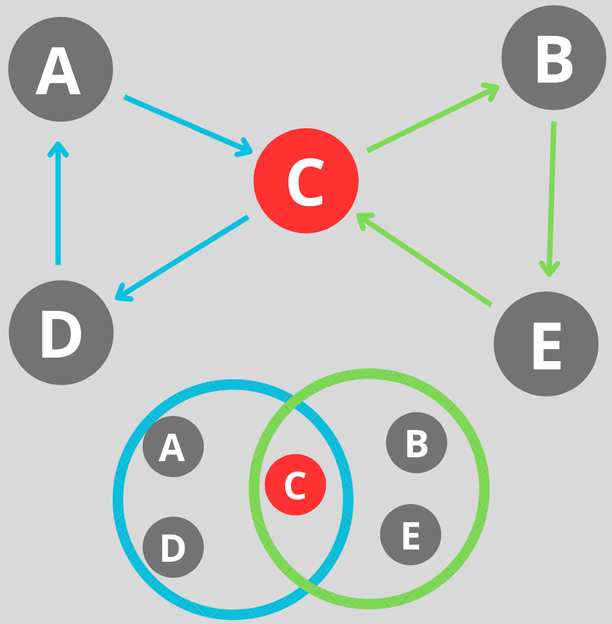


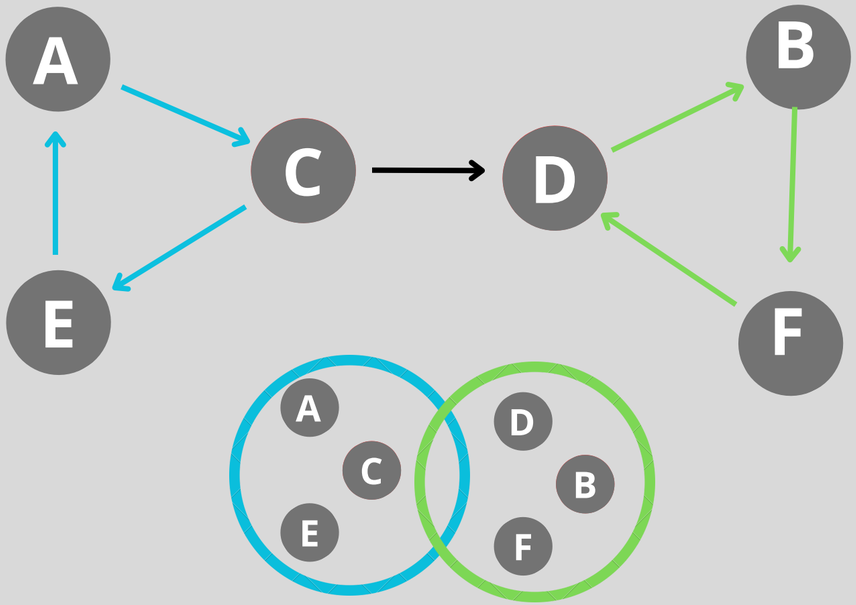


Implémentation

État infini

Objectif : Trouver un état présent infiniment souvent





O(n×c)

n : nb d'états
c : nb de cycles

Implémentation

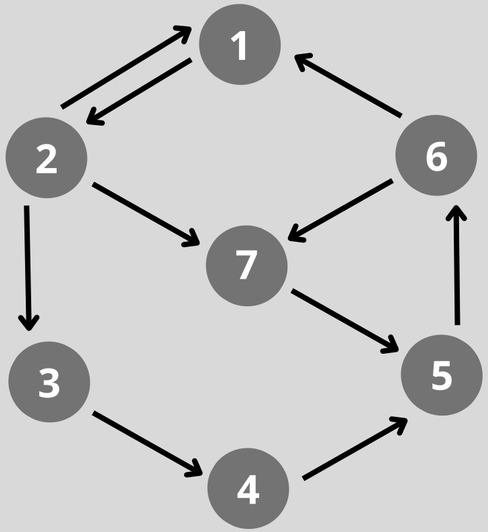
Groupe d’états infinis

Objectif : Trouver un groupe d’états présents infiniment souvent

Implémentation

Groupe d’états infinis

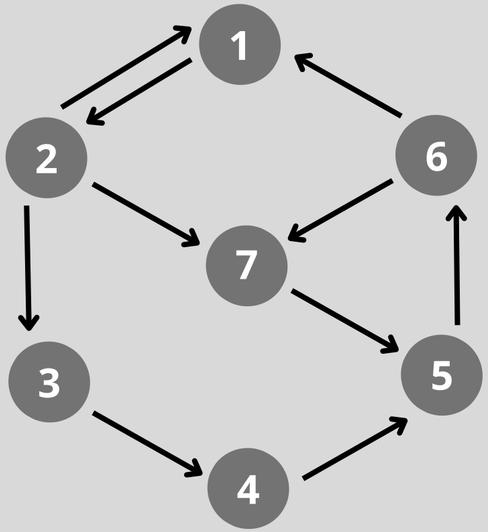
Objectif : Trouver un groupe d’états présents infiniment souvent



Implémentation

Groupe d’états infinis

Objectif : Trouver un groupe d’états présents infiniment souvent



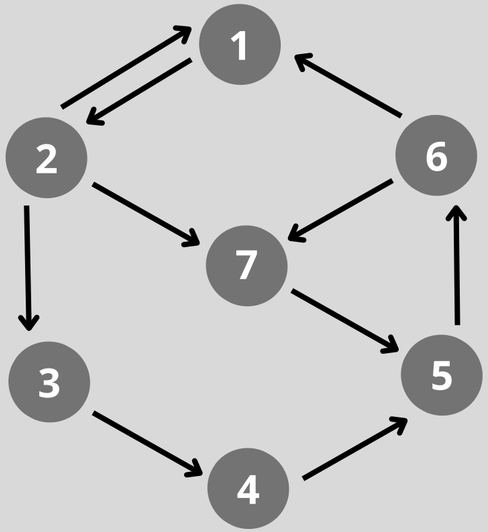
Tant que graphe cyclique :
- **Choisir état parmi états dans cycles**
- **Supprimer les cycles contenant cet état**

L’état choisi est celui avec le plus de transitions sortantes

Implémentation

Groupe d’états infinis

Objectif : Trouver un groupe d’états présents infiniment souvent



Tant que graphe cyclique :
- **Choisir état parmi états dans cycles**
- **Supprimer les cycles contenant cet état**

L’état choisi est celui avec le plus de transitions sortantes

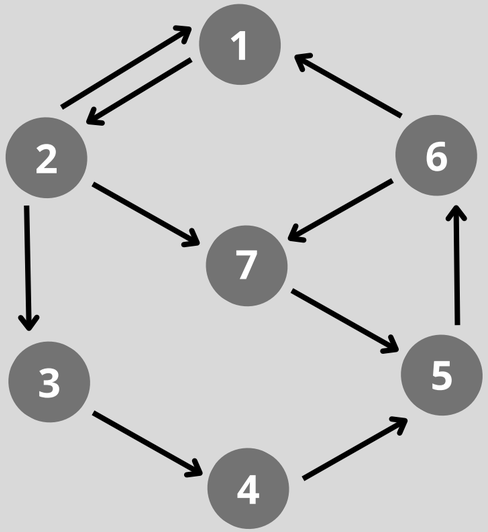
Cycles : [123456, 12756, 567, 12]

t	1
états choisis	2
cycles restants	[567]
groupe retourné	[2]

Implémentation

Groupe d’états infinis

Objectif : Trouver un groupe d’états présents infiniment souvent



Tant que graphe cyclique :
Choisir état parmi états dans cycles
Supprimer les cycles contenant cet état

L’état choisi est celui avec le plus de transitions sortantes

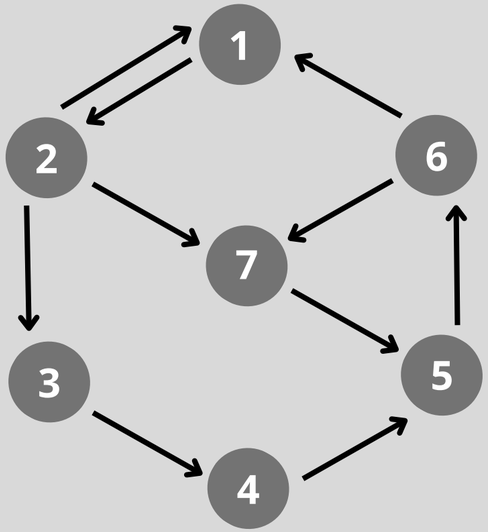
Cycles : [123456, 12756, 567, 12]

t	1	2
états choisis	2	6
cycles restants	[567]	[]
groupe retourné	[2]	[2, 6]

Implémentation

Groupe d'états infinis

Objectif : Trouver un groupe d'états présents infiniment souvent



Tant que graphe cyclique :
- **Choisir état parmi états dans cycles**
- **Supprimer les cycles contenant cet état**

L'état choisi est celui avec le plus de transitions sortantes

O(n×c)

Cycles : [123456, 12756, 567, 12]

t	1	2
états choisis	2	6
cycles restants	[567]	[]
groupe retourné	[2]	[2, 6]

$O(n \times c^2)$

n : nb d'états
c : nb de cycles

Résultats & Conclusion

Solution répondant à la problématique

Meilleure que le brute-force

Améliorations possibles