Algo 29 – Cycles et chemins dans un graphe

I – Détection d'un cycle dans un graphe

1.1 Principe général pour le graphe NON ORIENTE (2 conditions)

On a détecté un cycle si ces deux condition sont remplies lors de la visite en profondeur d'un sommet u :

1- **u possède un voisin v découvert GRIS**

2- **v n'est pas le parent de u** (c'est à dire qu'on est pas arrivé en u en partant de v)

VERSION AVEC UN DICTIONNAIRE parcours : est_cyclique()

```
1   BLANC = 0
2   GRIS = 1
3   NOIR = 2
4
5   def est_cyclique(g:dict[int, list]) -> bool:
6       """Prédicat qui renvoie True si le graphe possède un cycle (au moins un)"""
7
8       # INITIALISATION
9       parcours = {}                    # Création du dictionnaire (qui sera un dict de dict)
10      for u in g.keys():               # Pour chaque sommet du graphe
11          parcours[u] = {}           # La clé u de parcours a comme valeur associée un dict
12          parcours[u]['couleur'] = BLANC      # Initialisation de la couleur
13          parcours[u]['parent'] = None        # Initialisation du parent
14
15      # PARCOURIR
16      for u in g.keys():                              # Pour chaque sommet du graphe
17          if parcours[u]['couleur'] == BLANC:         # S'il est encore Non découvert
18              if visiter_profondeur(u, g, parcours):  # Si la visite de u détecte un cycle
19                  return True                         # on répond qu'on a un cycle
20      return False # si on arrive ici, c'est qu'il n'y a pas de cycle dans le graphe
21
22  def visiter_profondeur(u:int, g:dict[int, list], parcours:dict) -> bool:
23      """Visite un sommet u en particulier et renvoie True si on détecte un cycle"""
24
25      parcours[u]['couleur'] = GRIS                   # Signale que u est découvert
26
27      for v in g[u]:                            # Pour chaque sommet v de la liste d'adj de u
28          if parcours[v]['couleur'] == GRIS and parcours[u]['parent'] != v:
29              return True
30          elif parcours[v]['couleur'] == BLANC:       # Si v est non découvert
31              parcours[v]['parent'] = u               # u est son parent
32              if visiter_profondeur(v, g, parcours):  # si la visite de v détecte un cycle
33                  return True             # on signale qu'on peut atteindre un cycle depuis u
34
35      parcours[u]['couleur'] = NOIR      # u est totalement exploré
36      return False     # si on arrive ici c'est qu'on a pas détecté de cycle depuis ce sommet
```

VERSION sans couleur mais avec juste le tableau decouverts (voir la suite derrière)

```
1   def est_cyclique(g:dict[int, list]) -> bool:
2       """Prédicat qui renvoie True si le graphe possède un cycle (au moins un)"""
3
4       # INITIALISATION
5       decouverts = [] # Création du tableau des sommets decouverts
6
7       # PARCOURIR
8       for u in g.keys():                                      # Pour chaque sommet du graphe
9           if not(u in decouverts):                            # S'il est encore Non découvert
10              if visiter_profondeur(u, g, decouverts, None): # Si la visite de u détecte un cycle
11                  return True  # on répond qu'on a un cycle
12      return False  # si on arrive ici, c'est qu'il n'y a pas de cycle dans le graphe
13
14  def visiter_profondeur(u:int, g:dict[int, list], decouverts:list, parent:int|None) -> bool:
15      """Visite un sommet u en particulier et renvoie True si on détecte un cycle"""
16
17      decouverts.append(u)                        # Signale que u est découvert
18      for v in g[u]:                              # Pour chaque sommet v de la liste d'adj de u
19          if v in decouverts and parent != v:
20              return True
21          elif not(v in decouverts):              # Si v est non découvert
22              if visiter_profondeur(v, g, decouverts, u ): # si la visite de v détecte un cycle
23                  return True                 # on signale qu'on peut atteindre un cycle depuis u
24
25      return False      # si on arrive ici c'est qu'on a pas détecté de cycle depuis ce sommet
```

VERSION impérative

```
 1  def est_cyclique(g:dict[int, list]) -> bool:
 2      """Prédicat qui renvoie True si le graphe possède un cycle"""
 3
 4      # INITIALISATION
 5      d = []                  # Tableau des sommets découverts
 6
 7      # PARCOURIR
 8      for u in g.keys():              # Pour chaque sommet u du graphe
 9          # On remplace la pile d'appels par une vraie Pile
10          pile = []                   # Création d'une Pile vide
11          pile.append( (u, None) )    # On empile u et son parent (None)
12
13          # Equivalent impératif de visiter(u)
14          while pile:                 # Tant que la pile n'est pas vide
15              s, parent = pile.pop()  # On dépile le sommet s suivant
16              if not(s in d):         # Si s est non découvert
17                  d.append(s)         # Signale que u est découvert
18                  for v in g[s]:      # Pour chaque sommet v de la liste d'adj de s
19                      if (v in d) and (parent != v):
20                          return True
21                      else:
22                          pile.append( (v, s) ) # On empile v et son parent s
23
24          return False
```

1.2 Principe général pour le graphe ORIENTE (une condition)

On a détecté un cycle si, lors de la visite en profondeur d'un sommet u, **u possède un voisin v découvert GRIS.** La seule différence est donc en ligne 28 ici.

```
 1   BLANC = 0
 2   GRIS = 1
 3   NOIR = 2
 4
 5   def est_cyclique(g:dict[int, list]) -> bool:
 6       """Prédicat qui renvoie True si le graphe possède un cycle (au moins un)"""
 7
 8       # INITIALISATION
 9       parcours = {}                    # Création du dictionnaire (qui sera un dict de dict)
10       for u in g.keys():               # Pour chaque sommet du graphe
11           parcours[u] = {}            # La clé u de parcours a comme valeur associée un dict
12           parcours[u]['couleur'] = BLANC      # Initialisation de la couleur
13           parcours[u]['parent'] = None        # Initialisation du parent
14
15       # PARCOURIR
16       for u in g.keys():                              # Pour chaque sommet du graphe
17           if parcours[u]['couleur'] == BLANC:         # S'il est encore Non découvert
18               if visiter_profondeur(u, g, parcours):  # Si la visite de u détecte un cycle
19                   return True                         # on répond qu'on a un cycle
20       return False # si on arrive ici, c'est qu'il n'y a pas de cycle dans le graphe
21
22   def visiter_profondeur(u:int, g:dict[int, list], parcours:dict) -> bool:
23       """Visite un sommet u en particulier et renvoie True si on détecte un cycle"""
24
25       parcours[u]['couleur'] = GRIS                   # Signale que u est découvert
26
27       for v in g[u]:                             # Pour chaque sommet v de la liste d'adj de u
28->         if parcours[v]['couleur'] == GRIS:
29               return True
30           elif parcours[v]['couleur'] == BLANC:       # Si v est non découvert
31               parcours[v]['parent'] = u               # u est son parent
32               if visiter_profondeur(v, g, parcours):  # si la visite de v détecte un cycle
33                   return True             # on signale qu'on peut atteindre un cycle depuis u
34
35       parcours[u]['couleur'] = NOIR      # u est totalement exploré
36       return False     # si on arrive ici c'est qu'on a pas détecté de cycle depuis ce sommet
```

Notez bien qu'ici, il faut un sommet v GRIS et pas NOIR.

2 - Parcours en profondeur d'un graphe : algorithme complet

On rajoute deux attributs aux sommets en plus de couleur, parent et liste d'adjacence :

→ **debut** : date à laquelle on a commencé l'exploration du sommet (passage de BLANC à GRIS)

→ **fin** : date à laquelle on a fini son exploration (passage de GRIS à NOIR)

Le graphe doit être muni d'un attribut **date** commençant çà 0 et augmentant de 1 à chaque fois qu'on commence l'exploration d'un sommet ou qu'on finit l'exploration d'un sommet.

ALGORITHME PARCOURS EN PROFONDEUR

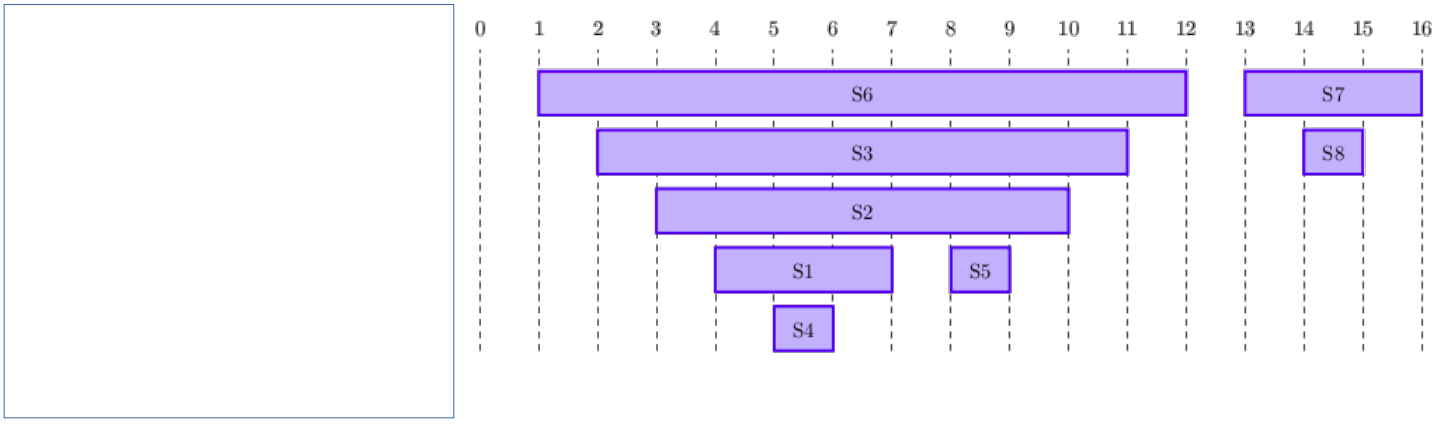
```
# on initialise le compteur d'étapes
g.date ← 0
# on initialise tous les sommets
POUR chaque sommet u appartenant au graphe g
        u.couleur ← BLANC
        u.parent ← VIDE
        u.debut ← 0
        u.fin ← 0
Fin Pour
# lance le traitement des sommets un par un
POUR chaque sommet u appartenant au graphe g
        SI u.couleur est BLANC
                # si on arrive ici, c'est que
                le sommet n'a pas encore été
                découvert
                visiter_en_profondeur(u, g)
        Fin Si
Fin Pour
Renvoyer VIDE (⌀)
```

ALGORITHME de visiter_en_profondeur(u, g)

```
g.date ← g.date + 1
u.debut ← g.date
u.couleur ← GRIS
POUR chaque sommet v de la liste d'adj. de u
        SI v.couleur est BLANC
                # si on arrive ici, c'est que
                le sommet n'a pas encore été
                visité
                v.parent ← u.
                visiter_en_profondeur(v, g)
        Fin Si
Fin Pour
# ici, tous les sommets adjacents ont été
tentés
g.date ← g.date + 1
u.fin ← g.date
u.couleur ← NOIR
Renvoyer VIDE (⌀)
```

3 - Détection des cycles dans un graphe orienté

Voici la Forêt d'exploration et les dates des différents sommets **sur le graphe de la question 3.**



On notera U l'intervalle du sommet u et V l'intervalle du sommet v.

3.1 Arcs de liaison : on descend d'un niveau exactement

Les **arcs de liaison** sont ceux qui caractérisent l'Arbre de parcours obtenu : ils font descendre exactement d'un niveau dans l'Arbre de parcours.

Exemples : (S6, S3), (S1,S4), (S2,S5)...

Comment le programme peut les détecter ?

Un arc (u,v) est un arc de liaison si u est le parent du sommet v dans l'Arbre de parcours.

$$u \neq v.parent$$

05° Pourquoi l'arc (S8,S7) n'est-il pas un arc de liaison sur ce parcours ?

05b° Que deviendrait l'arc (S8,S7) dans le cas où on commencerait l'exploration par S8 par exemple ?

05c° La dénomination des arcs est-elle dépendante ou indépendante du choix de l'ordre des sommets dans un graphe quelconque ?

3.2 Arcs arrière : on remonte d'un ou plusieurs niveaux U ⊂ V

Les **arcs arrière** sont les arcs qui relient un sommet à l'un de ces ancêtres dans l'Arbre de profondeur. Ils permettent de revenir en arrière dans le parcours et créer un cycle.

Exemple : (S4, S2)

Comment le programme peut-il les détecter ?

Un arc (u,v) est un arc arrière si l'intervalle du sommet-départ u est inclus dans l'intervalle-sommet v de destination.

$$U \subset V$$

06-a° (S4,S2) est un arc arrière**.** Que peut-on dire de l'intervalle du sommet de départ S4 par rapport à celui du sommet de destination S2 ?

A. $[5; 6] \subset [3;10]$
B. $[3;10] \subset [5; 6]$
C. $[5; 6] \in [3;10]$
D. $[3;10] \in [5; 6]$

06-b° Existe-il d'autres arcs arrière sur ce parcours en profondeur ?

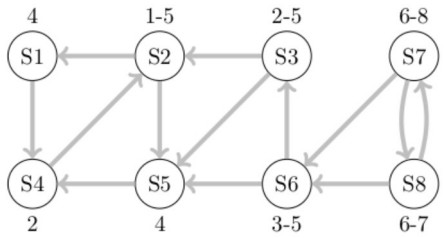
3.3 Trouver les chemins disponibles via la détection des cycles en utilisant les dates

→ Comment éviter les cycles ?

- un graphe possèdant un arc arrière contient au moins un cycle.
- un graphe ne possèdant pas d'arc arrière ne contient pas de cycle.

Eviter les cycles revient donc à ne pas utiliser les arcs arrière lors du déplacement dans le graphe.

07° Repasser en couleur les arcs utilisables dans le cadre du graphe simplifié de la Q3 :



08° Ecrire les 4 chemins partant de S6 en menant à un "cul de sac" : vous n'avez bien entendu pas le droit d'utiliser un arc arrière. 1er chemin : S6 - S3 - S2 - S1 – S4...

→ Trouver les chemins sans cycle avec l'utilisation de l'algorithme avec datation :

1.On exécute donc d'abord l'algorithme pour qu'il mémorise les dates sur les sommets.

2.On peut ensuite explorer le graphe sans emprunter les arcs arrière : ceux pour lesquelles l'intervalle du sommet de départ est inclus dans l'intervalle du sommet de destination.

09° Utiliser le graphe simplifié pour fournir tous les chemins explorables dans ce graphe partant de S7 sans réaliser de cycle. Pensez à utiliser la réponse à la question 07...

10° Imaginons que chacun des sommets représente les pièces d'un jeu où on peut perdre ou gagner des pièces d'or. Par quel chemin passer ?

S7 : +50 po	S8 : + 10 po	S6 : - 20 po
S3 : + 50 po	S2 : - 10 po	S5 : + 20 po
S1 : + 40 po	S4 : - 20 po	

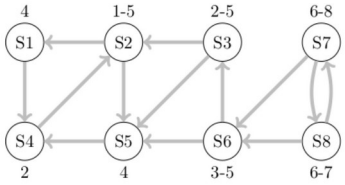
3.4 Trouver les chemins sans cycle SANS utiliser les dates

Lors de l'exploration, si on teste :

- un arc menant à **BLANC : arc de liaison**
- un arc menant à **GRIS : arc arrière**.
- un arc menant à **NOIR : ni l'un ni l'autre**.

Il reste à **stocker cette information** d'une façon ou d'une autre.

11° Utiliser l'animation du parcours en profondeur sur ce graphe orienté (celui de Q3). A quel moment détecte-t-on le premier arc arrière uniquement graçe à la couleur du sommet de destination ? Ce graphe contient-il des cycles ?



3.5 Conclusion

Pour visiter les chemins sans risque de tourner en rond, il suffit détecter et éviter les arcs arrière :

- **soit avec les dates stockées dans les sommets**
- **soit avec la couleur du sommet-destination des arcs lors du parcours initial en profondeur**

4 - Affiner la connaissance des chemins avec les dates

4.1 Arcs avant : on descend de plus d'un étage dans l'Arbre de parcours

Les arcs avant sont les arcs (u,v) reliant un sommet-départ u à l'un de ses descendants v dans l'arbre, sans que v ne soit le fils de u.

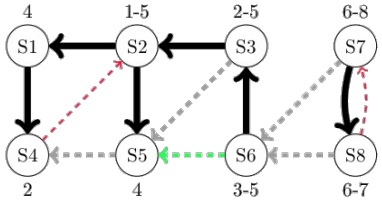
L'arc (S6, S5) est un arc avant.

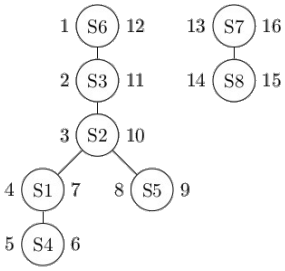
Comment le programme peut-il les détecter ?

Un arc (u,v) est un arc avant si u n'est pas le parent de v ET que l'intervalle du sommet v d'arrivée est inclus dans l'intervalle du sommet u de départ.

$V \subset U$ et $u \neq v.parent$

12° Trouver les autres arcs avant présents dans le parcours en profondeur proposé :





4.2 Arcs transverse : les liaisons entre sommets sans relation d'ancêtre-descendance

On y trouve donc les derniers arcs.

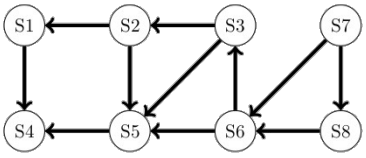
L'arc (S5,S4) est un exemple d'arc transerve entre sommet d'un même Arbre de parcours.

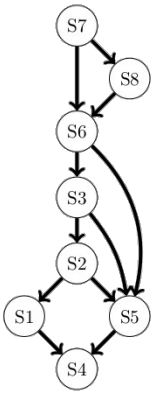
Comment le programme peut-il les détecter ?

Un arc (u,v) est un arc transverse si les intervalles des sommets u et v sont disjoints, c'est à dire qu'ils n'ont pas de dates communes.

$U \cap V = \varnothing$

4.3 Application : tri topologique





4.4 Résumé GRAPHE ORIENTE

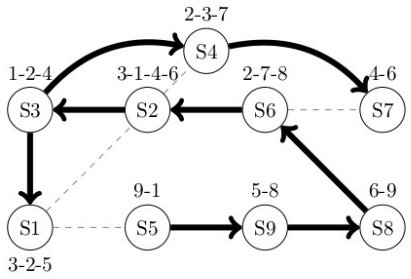
Type d'arc (u,v)	Détection avec dates		Détection sans date
	$u = v.parent$	u est le parent de v	L'arc mène vers un sommet BLANC
	$U \subset V$	Intervalle de u inclus dans celui de v	L'arc mène vers un sommet GRIS
	$V \subset U$ $u \neq v.parent$	Intervalle de v inclus dans celui de u u n'est pas le parent de v	L'arc mène vers un sommet NOIR (insuffisant)
	$U \cap V = \varnothing$	Intervalles de u et v sont disjoints	L'arc mène vers un sommet

			NOIR (insuffisant)

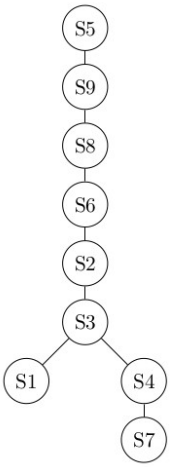
5 - Détection des cycles dans un graphe non orienté

A savoir : une arête est classifiée lors de sa première utilisation dans le parcours. Une arête ne peut donc être que DE LIAISON ou ARRIERE.

13° Utiliser l'animation du parcours en profondeur proposé sur le site. Tracer les intervalles. Trouver les arcs arrière. En déduire s'il existe des cycles dans ce graphe non orienté.



14° Tracer les intervalles obtenus sur le parcours proposé ci-contre. Trouver les arcs arrière. En déduire s'il existe des cycles dans ce graphe non orienté.



Sommets

Résumé GRAPHE NON ORIENTE

Type d'arête {u,v}	Détection avec dates		Détection sans date (1er utilisation de l'arête)
	`u = v.parent`	u est le parent de v.	L'arête mène vers un sommet BLANC
	`U ⊂ V`	Intervalle de u inclus dans celui de v	L'arête mène vers un sommet GRIS

Sur le site :

15° Utiliser l'animation du parcours en profondeur sur ce graphe non orienté. En tant qu'humain, à quel moment détecte-t-on la première arête arrière ? Ce graphe contient-il des cycles ?

16° Que peut-on dire de l'arête (S1,S3) lors de sa première utilisations ? lors de sa deuxième utilisation ?

17° Que peut-on dire de l'arête (S2,S3) lors de sa première utilisations ? lors de sa deuxième utilisation ?