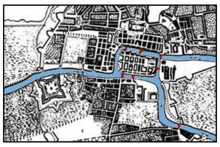
Théorie des graphes

I
Introduction
&
Définitions

Historique

- 1726 Leonhard Euler expose une solution formelle au problème des 7 ponts de Königsberg (Kaliningrad):

« Lors d'une promenade, est-il possible de passer sur tous les ponts de la ville une et une seule fois ? »



Domaines d'applications

- Chimie :
  Modélisation des molécules
- Mécanique :
  Treillis
- Biologie :
  Réseau de neurones
  Séquencement du génome
- Sciences sociales :
  Modélisation des relations
- Et biensûr dans divers domaines de l'informatique

Avertissement

- Il faut se méfier de l'apparente simplicité de certains résultats.
- Dans la pratique, la taille des graphes ne permet pas de représentation graphique.

ATTENTION :
Veiller à toujours appliquer les algorithmes présentés même si le résultat peut être trouvé « artisanalement »

Définition : graphe

- Un **graphe orienté** G c'est un couple (S,A) avec :
  - S un ensemble fini : **ensemble des sommets**
  - A une relation binaire sur S : **ensemble des arcs**
- Un **graphe NON orienté** G c'est un couple (S,A) :
  - S un ensemble fini : **ensemble des sommets**
  - A paires non ordonnées : **ensemble des arêtes**

Degré d'un sommet

- Dans un graphe non orienté :
  - On appel **degré** d'un sommet :
    le nombre d'arêtes qui lui sont incidentes
- Dans un graphe orienté :
  - On appelle **degré sortant** d'un sommet :
    le nombre d'arcs qui partent de ce sommet
  - On appelle **degré entrant** d'un sommet :
    le nombre d'arcs qui arrivent à ce sommet
  - On appelle **degré** d'un sommet :
    la somme des degrés entrant et sortant du sommet

Chemin

- Un **chemin** d'un sommet u au sommet u' est une séquence de sommets $(v_0, v_1, v_2, \ldots v_{k-1}, v_k)$ tel que :
  $$u = v_0, \quad u' = v_k \quad et \quad \forall i, (v_{i-1}, v_i) \in A$$
- On dit que ce chemin a une **longueur** k
- Ce chemin est **élémentaire** ssi $\forall i, j,\ v_i \neq v_j$
- Un sommet u **accessible** depuis un sommet v ssi :
  il existe un chemin du sommet u au sommet v

Degré d'un sommet

- Dans un graphe orienté :
  - Un chemin $(v_0, v_1, \ldots, v_k)$ forme un **circuit** ssi $v_0 = v_k$
  - Ce circuit est **élémentaire** ssi $\forall\, i, j \in [1,k\text{-}1],\ v_i \neq v_j$
  - Une **boucle** est un circuit de longueur 1
  - Un est graphe **acyclique** ssi il ne contient aucun circuit
- Dans un graphe non orienté :
  - Un chemin $(v_0, v_1, \ldots, v_k)$ forme un **cycle** ssi
    $(v_0 = v_k) \wedge (\forall\, i, j \in [1,k\text{-}1],\ v_i \neq v_j)$
  - Un graphe est **acyclique** ssi il ne contient aucun cycle

Propriétés

- On dit d'un graphe qu'il est :
  - **Réflexif** ssi : $\forall\, u_i \in S,\ (u_i, u_i) \in A$
  - **Irréflexif** ssi : $\forall\, u_i \in S,\ (u_i, u_i) \notin A$
  - **Transitif** ssi :
    $\forall\, u_i,\ u_j,\ u_k \in S,\ (u_i, u_j) \in A \wedge (u_j, u_k) \in A \Rightarrow (u_i, u_k) \in A$
- On dit d'un graphe orienté qu'il est :
  - **Symétrique** ssi : $\forall\, u_i, u_j \in S,\ (u_i, u_j) \in A \Rightarrow (u_j, u_i) \in A$
  - **Anti-Symétrique (Assymetrique)** ssi :
    $\forall\, u_i,\ u_j \in S,\ (u_i, u_j) \in A \wedge (u_j, u_i) \in A \Rightarrow u_i = u_j$

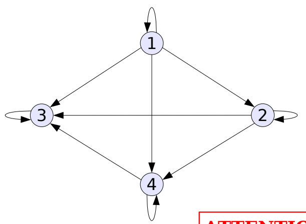
Connexité

- On dit d'un graphe non orienté qu'il est :
  - **Connexe** ssi pour toute paire de sommets *[u,v]* il existe une chaîne entre les sommets *u* et *v*.
  - **Complet** ssi tous les sommets sont «reliés» 2 à 2 :
    $\forall\, u, v \in S,\ (u, v) \in A$
- On dit d'un graphe orienté qu'il est :
  - **Connexe** ssi le graphe non-orienté correspondant est connexe
  - **Fortement connexe** ssi si pour tout *(u,v)* il existe un chemin de *u* à *v* et de *v* à *u*
  - **Complet** ssi tous les sommets sont «reliés» 2 à 2 :
    $\forall\, u, v \in S,\ (\,(u, v) \in A\,) \vee (\,(v, u) \in A\,)$

K-Connexe

- Un graphe non-orienté est **k-connexe** ssi :
  - il reste connexe après suppression d'un ensemble quelconque de k-1 arêtes et s'il existe un ensemble de k arêtes qui déconnecte le graphe.
  - autrement dit s'il existe au moins k chaînes indépendantes entre chaque couple de sommets.
- Un graphe orienté est **k-connexe** ssi :
  - le graphe non-orienté correspondant est k-connexe
- Cette notion est utilisée :
  - en électronique pour le calcul de la fiabilité
  - dans l'étude de jeux de stratégie (cut and connect).

Exemple



- Ce graphe orienté est :
  - Réflexif
  - Antisymétrique
  - Transitif
  - Connexe
  - Complet

ATTENTION :
Dans un graphe orienté :
Complet n'implique pas Fortement connexe.
Ex : il n'y a pas de chemin pour aller de 2 a 1

Graphes remarquables

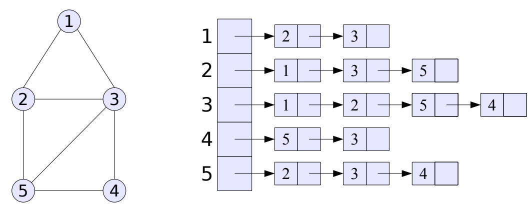
- Certains graphes portent des noms particuliers :
  - **Biparti =** graphe qui peut être partitionner en deux sous ensembles de sommets S_1 et S_2 tel que :
    $\forall\, (u,v) \in A,\ (u \in S_1 \wedge v \in S_2) \vee (v \in S_1 \wedge u \in S_2)$
  - **Hypergraphe =** graphe non orienté ou chaque arrête est une **hyperarête** qui relie un sommet à un sous ensemble de sommets.
  - **Forêt =** graphe non orienté acyclique.
  - **Arbre =** graphe connexe non orienté acyclique.

Représentation d'un graphe

- Il existe deux façons de représenter un graphe *(S,A)* :
  - **Liste adjacente** : pour les graphes peu denses
    $$Card\,(A) << (\,Card\,(S)\,)^2$$
  - **Matrice d'incidence** : pour les graphes denses
    $$Card\,(A) \simeq (\,Card\,(S)\,)^2$$

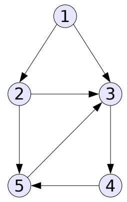
Liste d'adjacence

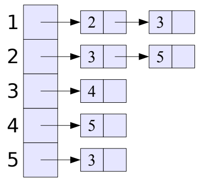
- Pour chaque sommet $u \in S$ on a une liste d'adjacence :
  Adj [u] liste des sommets $v \in S$ tel que $(u,v) \in A$



Liste d'adjacence

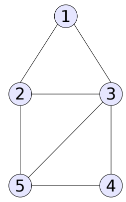
- Pour chaque sommet $u \in S$ on a une liste d'adjacence :
  Adj [u] liste des sommets $v \in S$ tel que $(u,v) \in A$





Matrice d'adjacence

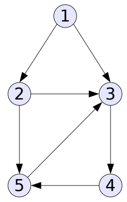
- *Pour un graphe non orienté* : $\forall i, j \in S \quad a_{ij} = \begin{cases} 1 & si (i,j) \in A \\ 0 & sinon \end{cases}$



$$Mat(S,A) = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Matrice d'adjacence

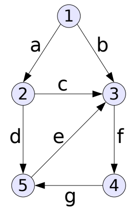
- *Pour un graphe orienté* : $\forall i, j \in S \quad a_{ij} = \begin{cases} 1 & si (i,j) \in A \\ 0 & sinon \end{cases}$



$$Mat(S,A) = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Matrice d'incidence

- *Pour un graphe orienté* : $\forall i \in S, \forall j \in A \quad a_{ij} = \begin{cases} 1 & si\ \vec{j} = (i,x) \\ -1 & si\ \vec{j} = (x,i) \\ 0 & sinon \end{cases}$



$$Mat(S,A) = \begin{pmatrix} & a & b & c & d & e & f & g \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & -1 & -1 & 0 & -1 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 5 & 0 & 0 & 0 & -1 & 1 & 0 & -1 \end{pmatrix}$$

II

Parcours en largeur

&

Parcours en profondeur

Bordure

- On appel **bordure** d'un sous-ensemble de sommet S' :

  L'ensemble des sommets de V – S', où V est l'ensemble des voisins des sommets de S' dans le graphe G.

- Elle est notée :

$$\begin{aligned} B(S',G) &= V - S' \\ &= \{ \ v \in S \mid (v \notin S') \ \wedge \ (\exists s \in S, \ (s,v) \in A) \ \} \end{aligned}$$

- **Rappel :**

  L'ensemble { x | P (x) }

  ⇔

  l'ensemble des x tels que la condition P (x) soit vraie

Parcours

- La permutation L(s_1,…, s_n) est un **parcours** de G ssi :

  $\forall j \in [1..n], \ \ B(L[1,j-1],G) \neq 0 \ \Rightarrow \ s_j \in B(L[1,j-1],G)$

  avec $L[i,j]$ une sous liste de la permutation L

- Dans cette définition :
  - $L[1,j]$: Liste des *j* sommets déjà visités
  - $L(s_1, \ldots, s_n)$: Un ordre pour parcourir les graphes tq :

    Si au moins l'un des sites déjà parcourus a un voisin dans le graphe alors le prochain site visité doit être l'un de ces voisins.

Sommet ouvert ou fermé

- Un sommet *s* est **ouvert dans L[1..i]** ssi :

  $\exists \ v \in L[i+1..n], \ (s,v) \in A$

Sommet ouvert ou fermé

- Un sommet *s* est **ouvert dans L[1..i]** ssi :

$$\exists\ v \in L[i+1..n],\ (s,v) \in A$$

- Cette définition signifie :

Il y a des voisins de *s* qui n'ont pas encore été visités (dans les i premiers sommets du parcours)

Sommet ouvert ou fermé

- Un sommet *s* est **ouvert dans L[1..i]** ssi :

$$\exists\ v \in L[i+1..n],\ (s,v) \in A$$

- Cette définition signifie :

Il y a des voisins de *s* qui n'ont pas encore été visités (dans les i premiers sommets du parcours)

- Un sommet *s* est **fermé dans L[1..i]** ssi :

$$\textbf{NOT}\ (\ \exists\ v \in L[i+1..n],\ (s,v) \in A\ ) \Leftrightarrow \forall\ v \in L[i+1..n],\ (s,v) \notin A$$

Sommet ouvert ou fermé

- Un sommet *s* est **ouvert dans L[1..i]** ssi :

$$\exists\ v \in L[i+1..n],\ (s,v) \in A$$

- Cette définition signifie :

Il y a des voisins de *s* qui n'ont pas encore été visités (dans les i premiers sommets du parcours)

- Un sommet *s* est **fermé dans L[1..i]** ssi :

$$\textbf{NOT}\ (\ \exists\ v \in L[i+1..n],\ (s,v) \in A\ ) \Leftrightarrow \forall\ v \in L[i+1..n],\ (s,v) \notin A$$

- Cette définition signifie :

Tous les voisins de *s* ont déjà été visités (dans les i premiers sommets du parcours)

Parcours en largeur : BFS

- On appel **parcours en largeur** - **B**readth **F**irst **S**earch

Un parcours où pour tout sommet $s_i \in L[1..i]$ le prédécesseur est le premier sommet ouvert dans L[1..i-1]

Parcours en largeur : BFS

- On appel **parcours en largeur** - **B**readth **F**irst **S**earch

Un parcours où pour tout sommet $s_i \in L[1..i]$ le prédécesseur est le premier sommet ouvert dans L[1..i-1]

- Cette définition signifie :

Lorsque l'on a visité j-1 sommets le prochain sommet est un voisin du premier site déjà visité qui a encore au moins un voisin non visité.

Parcours en largeur : BFS

- On appel **parcours en largeur** - **B**readth **F**irst **S**earch

Un parcours où pour tout sommet $s_i \in L[1..i]$ le prédécesseur est le premier sommet ouvert dans L[1..i-1]

- Cette définition signifie :

Lorsque l'on a visité j-1 sommets le prochain sommet est un voisin du premier site déjà visité qui a encore au moins un voisin non visité.

- Autrement dit :

A chaque étape du parcours on visite tous les voisins non visités des sites visités à l'étape précédente.

Parcours en Profondeur : DFS

- On appel **parcours en profondeur** - **D**epth **F**irst **S**earch

Un parcours où pour tout sommet $s_i \in L[1..i]$ le prédécesseur est le dernier sommet ouvert dans L[1..i-1]

-

Parcours en Profondeur : DFS

- On appel **parcours en profondeur** - **D**epth **F**irst **S**earch

Un parcours où pour tout sommet $s_i \in L[1..i]$ le prédécesseur est le dernier sommet ouvert dans L[1..i-1]

- Cette définition signifie :

Lorsque l'on a visité j-1 sommets le prochain sommet est un voisin du *dernier* site déjà visité qui a encore au moins un voisin non visité.

-

Parcours en Profondeur : DFS

- On appel **parcours en profondeur** - **D**epth **F**irst **S**earch

Un parcours où pour tout sommet $s_i \in L[1..i]$
le prédécesseur est le dernier sommet ouvert dans L[1..i-1]

- Cette définition signifie :

Lorsque l'on a visité j-1 sommets le prochain sommet est un voisin du *dernier* site déjà visité qui a encore au moins un voisin non visité.

- Autrement dit :

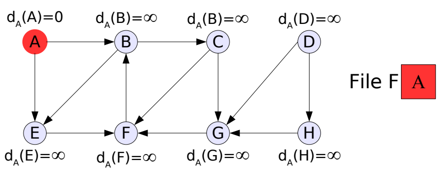
On descend le plus possible dans le parcours.
Quand on ne peut plus on remonte le moins possible

Algorithme BFS

```
BFS (graphe G, sommet s)
  POUR CHAQUE v ≠ s FAIRE couleur(v) ← Blanc ; distance(v) ← ∞
  couleur(s) ← Rouge
  distance(s) ← 0
  F ← {s}
  TANT-QUE non FileVide(F ) FAIRE
      s ← Défiler(F )
      POUR CHAQUE v ∈ Adj[s]
          SI couleur(v) = Blanc ALORS
              couleur(v) ← Rouge
              distance(v) ← distance(s) + 1
              père(v) ← s
              Enfiler(F,v)
          FIN SI
      FIN POUR
      couleur(s) ← Noir
  FIN-TANT-QUE
```

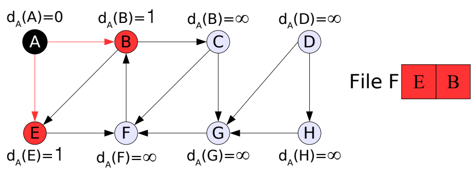

Exemple de l'algorithme BFS

- A l'état initial :
  - seul le sommet A est rouge
  - La file est réduite au site A



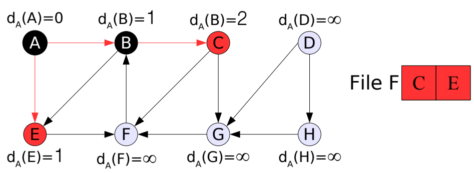
Exemple de l'algorithme BFS

- On défile le sommet A
- On visite les voisins blancs de A : B et E
- Le site A devient noir



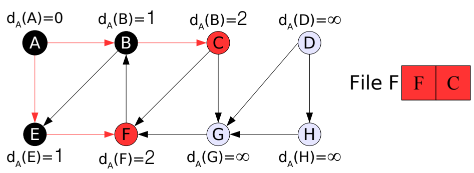
Exemple de l'algorithme BFS

- On défile le sommet B
- On visite le voisin blanc de B : C
- Le site B devient noir



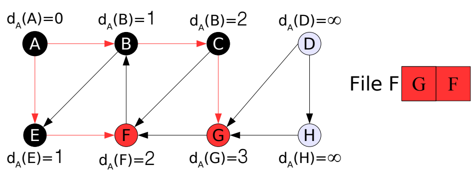
Exemple de l'algorithme BFS

- On défile le sommet E
- On visite le voisin blanc de B : F
- Le site B devient noir



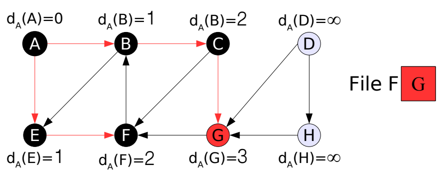
Exemple de l'algorithme BFS

- On défile le sommet C
- On visite le voisin blanc de C : G
- Le site C devient noir



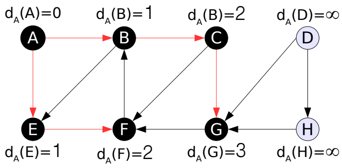
Exemple de l'algorithme BFS

- On défile le sommet F
- F n'a pas de voisin blanc
- Le site F devient noir



Exemple de l'algorithme BFS

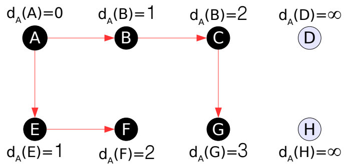
- On défile le sommet G
- G n'a pas de voisin blanc
- Le site G devient noir



File F : **VIDE**

Exemple de l'algorithme BFS

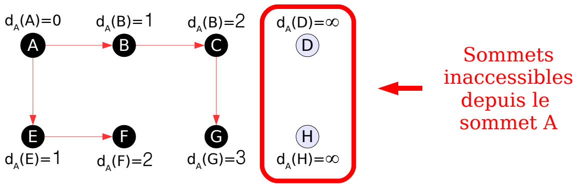
- Il n'y a plus de sommet à défiler : Fin de l'algorithme
- On obtient une **arborescence en largeur**
- $v \in S$, $d_A(v)$ = longueur du **plus court chemin** entre A et v



File F : **VIDE**

Exemple de l'algorithme BFS

ATTENTION :
Dans un parcours en largeur : tous les sommets ne sont pas visités
Ainsi les sites inaccessibles depuis l'origine gardent une distance ∞



Initialisation BFS

VARIABLE
date : un compteur d'étape

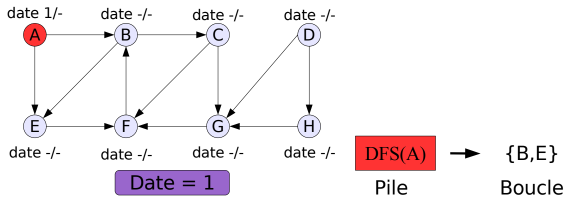
DFS_run (graphe *G*)
POUR CHAQUE $\sigma \in S$ **FAIRE**
couleur(s) ← Blanc
FIN POUR
date ← 0
POUR CHAQUE $s \in S$ **FAIRE**
SI couleur(s) = Blanc **ALORS**
DFS (G , s)
FIN SI
FIN POUR

Algorithme récursif BFS

DFS (graphe *G*, sommet *s*)
couleur(s) ← Rouge
dateDebut(s) ← inc(date)
POUR CHAQUE $v \in$ Adj[s]
SI couleur(v) = Blanc **ALORS**
père(v) ← s
DFS (G,v)
FIN SI
FIN POUR
couleur(s) ← Noir
dateFin(s) ← inc(date)

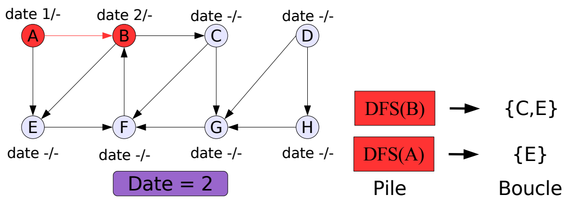
Exemple de l'algorithme DFS

- La fonction DFS_run() appelle DFS(A).
- Seul le sommet A est rouge
- La pile des appels de fonction est réduite au DFS(A)



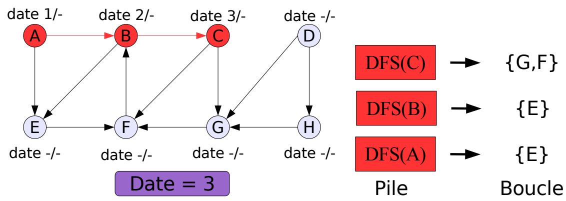
Exemple de l'algorithme DFS

- On empile la fonction DFS(B)
- B devient rouge et on note la date : dateDebut(B) ← 2



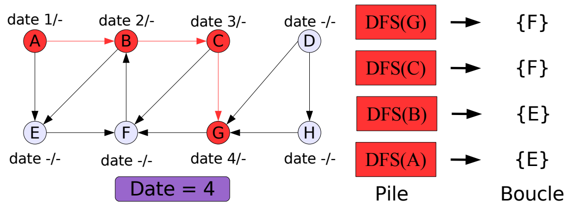
Exemple de l'algorithme DFS

- On empile la fonction DFS(C)
- C devient rouge et on note la date : dateDebut(B) ← 3



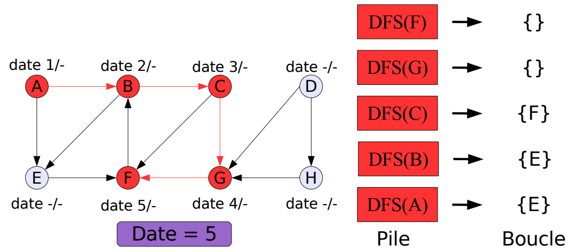
Exemple de l'algorithme DFS

- On empile la fonction DFS(G)
- G devient rouge et on note la date : dateDebut(G) ← 4



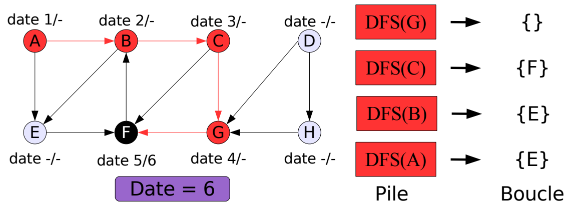
Exemple de l'algorithme DFS

- On empile la fonction DFS(F)
- F devient rouge et on note la date : dateDebut(F) ← 5



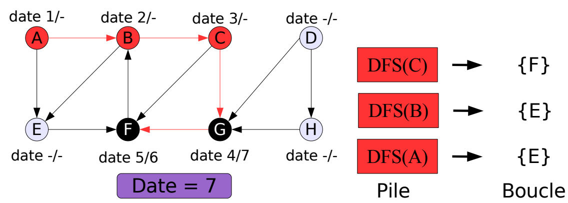
Exemple de l'algorithme DFS

- Fin de la boucle dans la fonction DFS(F)
- F devient noir et on note la date : dateFin(F) ← 6



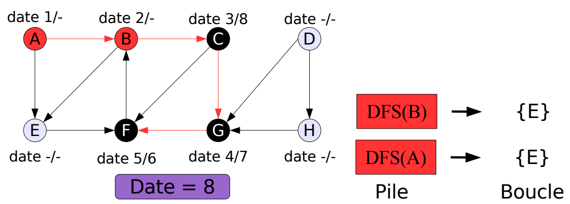
Exemple de l'algorithme DFS

- Fin de la boucle dans la fonction DFS(G)
- F devient noir et on note la date : dateFin(G) ← 7



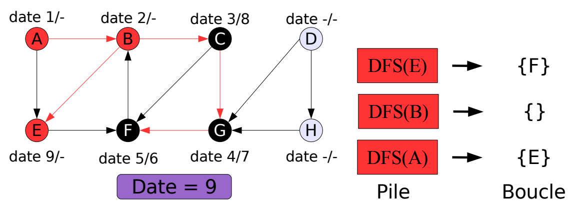
Exemple de l'algorithme DFS

- Le sommet F n'est pas blanc ⇒ pas d'appel à DFS(F)
- Fin de la boucle dans la fonction DFS(C)
- C devient noir et on note la date : dateFin(C) ← 8



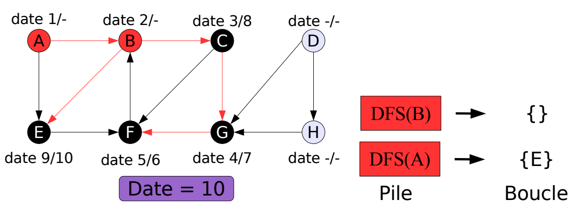
Exemple de l'algorithme DFS

- Retour à la boucle dans la fonction DFS(B)
- On empile la fonction DFS(E)
- E devient rouge et on note la date : dateDebut(E) ← 9



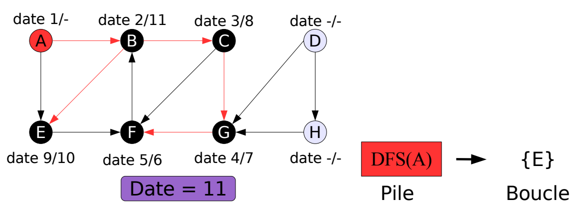
Exemple de l'algorithme DFS

- Le sommet F n'est pas blanc ⇒ pas d'appel a DFS(E)
- Fin de la boucle dans la fonction DFS(E)
- E devient noir et on note la date : dateFin(E) ← 10



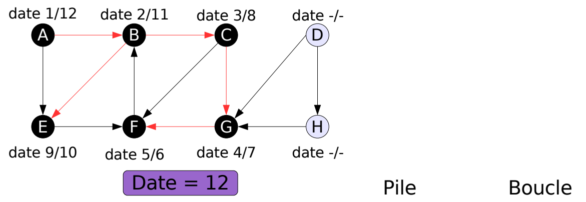
Exemple de l'algorithme DFS

- Fin de la boucle dans la fonction DFS(B)
- B devient noir et on note la date : dateFin(B) ← 11



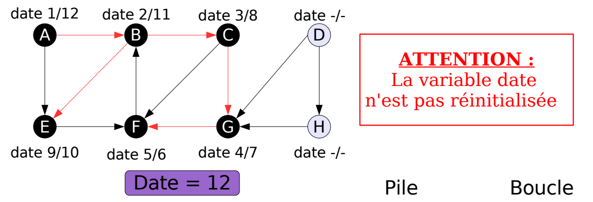
Exemple de l'algorithme DFS

- Le sommet E n'est pas blanc ⇒ pas d'appel à DFS(A)
- Fin de la boucle dans la fonction DFS(A)
- A devient noir et on note la date : dateFin(A) ← 12



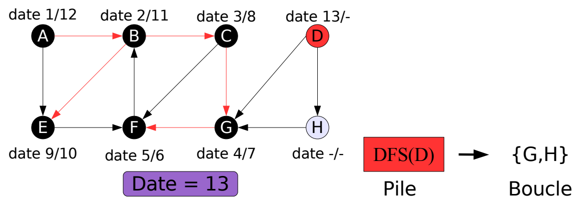
Exemple de l'algorithme DFS

- L'appel à la fonction DFS(A) est terminé.
- La boucle principale de DFS_run() appel ensuite DFS(A) et DFS(B) qui ce terminent tout de suite : « pas de voisin blanc »



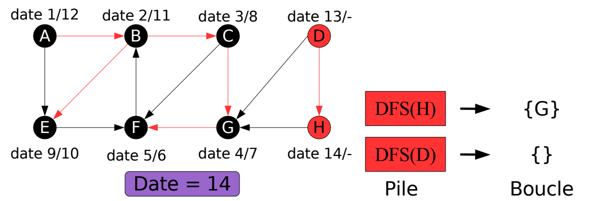
Exemple de l'algorithme DFS

- On empile la fonction DFS(D)
- F devient rouge et on note la date : dateDebut(F) ← 13



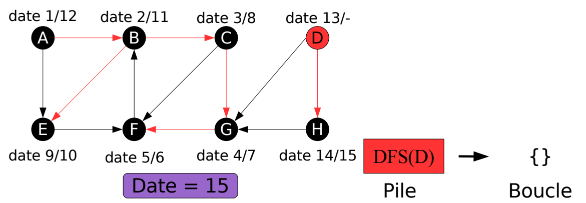
Exemple de l'algorithme DFS

- Le sommet G n'est pas blanc ⇒ pas d'appel a DFS(G)
- On empile la fonction DFS(H)
- H devient rouge et on note la date : dateDebut(H) ← 14



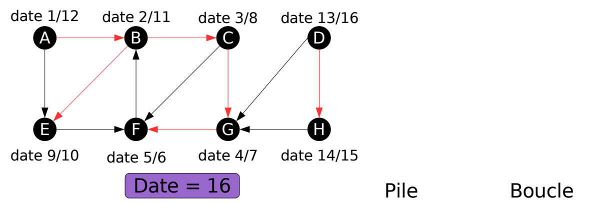
Exemple de l'algorithme DFS

- Le sommet G n'est pas blanc ⇒ pas d'appel a DFS(G)
- Fin de la boucle dans la fonction DFS(H)
- H devient noir et on note la date : dateFin(H) ← 15



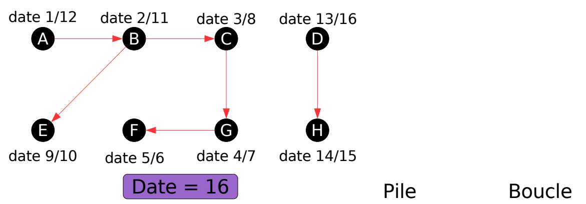
Exemple de l'algorithme DFS

- Fin de la boucle dans la fonction DFS(H)
- H devient noir et on note la date : dateFin(H) ← 16

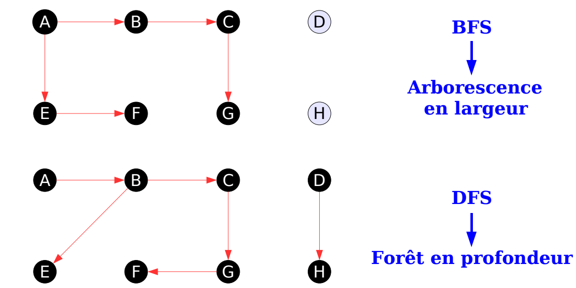


Exemple de l'algorithme DFS

- Toutes les fonctions lancées par DFS_init() avortent.
- Parcours en profondeur ⇒ tous les sommets sont visités
- On obtient une **foret en profondeur**



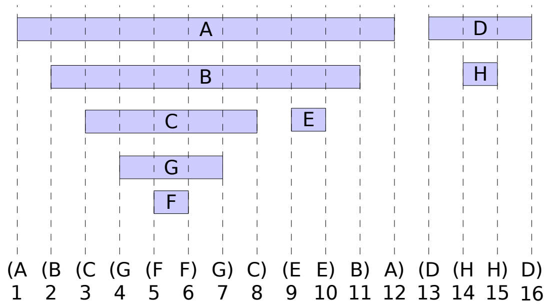
Comparaison BFS et DFS



Théorème des parenthèses

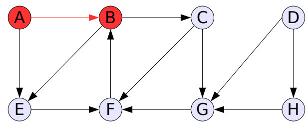
- Les dates de découvertes et fin de traitement ont : **une structure parenthésée**
- Théorèmes :
  $\forall u, v \in S$ une seul des 3 propositions suivantes est vraie :
  - *[début[u], fin[u]] ∩ [début[v], fin[v]] = Ø*
  - *[début[u], fin[u]] ⊂ [début[v], fin[v]]*
    ET u descendant de v dans une arborescence de la foret
  - *[début[v], fin[v]] ⊂ [début[u], fin[u]]*
    ET v descendant de u dans une arborescence de la foret

Théorème des parenthèses



Théorème du chemin blanc

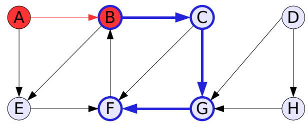
- Théorèmes :
  Dans une foret en profondeur, un sommet *u* est un descendant d'un sommet *v* si seulement si :
  lorsque l'on découvre *u* (*dateDebut[u]*),
  il existe un **chemin de sommet BLANC** entre u et v



- On vient de découvrir B

Théorème du chemin blanc

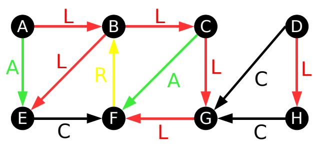
- Théorèmes :
  Dans une foret en profondeur, un sommet *u* est un descendant d'un sommet *v* si seulement si :
  lorsque l'on découvre *u* (*dateDebut[u]*),
  il existe un **chemin de sommet BLANC** entre u et v



- On vient de découvrir B
- **(B, C, G, F)** chemin BLANC
- Donc : F descendant de B

Classement des arcs

- Un arc *(u,v)* est appelé :
  - **arc de liaison** : les arcs de la forêt en profondeur.
  - **arc de retour** : si *v* est un des ancêtres de *u*
  - **arc avant** : si *u* est un des ancêtres de *v* ET qu'il n'est pas un arc de liaison
  - **arc couvrant** : tous les autres



Théorème des arcs en retour

- **Théorèmes :**
  Un graphe orienté est acyclique ssi : un parcours en profondeur de G ne génère pas d'arc retour.
- **Démonstration :**
  - (**Acyclique ⇒ Pas d'arc retour**) ⇔ (arc de retour ⇒ cycle) :
    - Soit *(u,v)* un arc retour ⇒ *v* est un ancêtre de *u*
      ⇒ il existe un chemin de *v* a *u*
      ⇒ Ce chemin et *(u,v)* forme un cycle
  - (**Pas d'arc retour ⇒ Acyclique**) ⇔ (cycle ⇒ arc de retour)
    - Soit C un cycle et *v* le premier sommet de *C* visité dans le parcours. On note *(u,v)* l'arc du cycle qui conduit a *v*.
    - *v* premier sommet visité ⇒ les autres sommets de C sont blanc
      ⇒ u descendant de v (Th. Chemin blanc)
      ⇒ (u,v) est un arc retour

Algorithme de décomposition

- On cherche ici à décomposer un graphe orienté en composantes fortement connexes
- L'algorithme utilise :
  un **double parcours en profondeur**

Décomposition (graphe *G*)

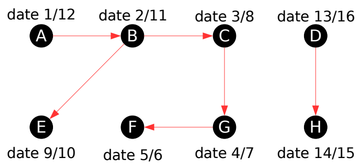
DFS_run(G)

Calculer ${}^tG$: transposé de G (inversion du sens de tous les arcs)

DFS_run(${}^tG$) dans la boucle principale qui appelle DFS(s), on parcourt les sommets par ordre décroissant des *dateFin* calculées lors du premier DFS(G)

Exemple de décomposition

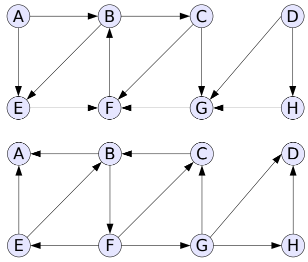
- A la fin du premier appel DFS (G), on obtient :



- On ordonne les sommets suivants *dateFin* (décroissant):
  D – H – A – B – E – C – G – F
- C'est l'ordre qui sera utilisé dans la boucle du 2[em] DFS_run

Exemple de décomposition

- On inverse les arcs pour obtenir : le graphe transposé



Graphe G

Transposition : Inversion des arcs

Graphe ${}^tG$

Exemple de décomposition

- La transposition est une opération matricielle
- La matrice d'adjacence du graphe transposé ${}^tG$ est : la transposée de la matrice d'adjacence du graphe G
- *Ex : L'inversion de l'arc* $\overrightarrow{BC} \Leftrightarrow \begin{cases} a_{23}=0 \\ a_{32}=1 \end{cases}$

Graphe G

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

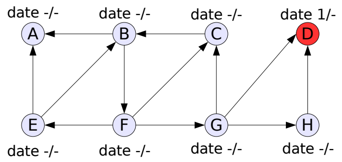
Transposition : $\longrightarrow$ $\forall\, i,j \in S : a_{ij} \leftarrow a_{ji}$

Graphe ${}^tG$

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

Exemple de décomposition

- On lance une deuxième fois le parcours en profondeur
- D est le premier sommet dans la boucle de DFS_run()
- Appel a DFS(D); D devient rouge.



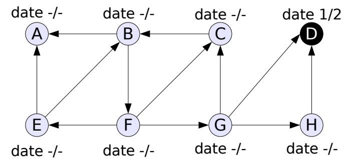
DFS(D) → {}

Date = 1

Pile Boucle

Exemple de décomposition

- D n'a pas de voisins blancs
- D devient noir
- Fin du 1[er] appel de DFS() dans la boucle de DFS_run()

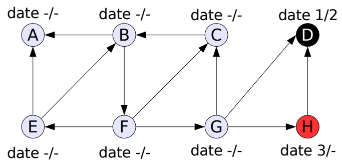


Date = 2

Pile Boucle

Exemple de décomposition

- H est le 2[em] sommet dans la boucle de DFS_run()
- Appel à DFS(H); H devient rouge.



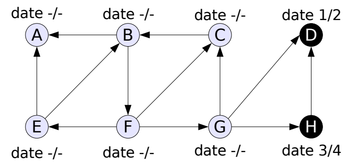
DFS(H) → {D}

Date = 3

Pile Boucle

Exemple de décomposition

- H n'a pas de voisins blancs
- H devient noir
- Fin du 2[em] appel de DFS() dans la boucle de DFS_run()

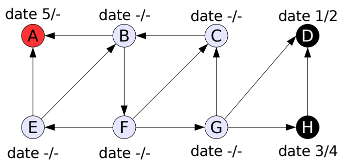


Date = 4

Pile Boucle

Exemple de décomposition

- A est le 3[em] sommet dans la boucle de DFS_run()
- Appel a DFS(A)
- A devient rouge.



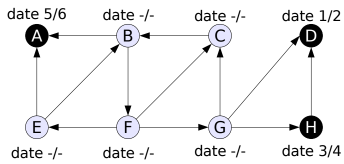
DFS(A) → {}

Date = 5

Pile Boucle

Exemple de décomposition

- A n'a pas de voisins blancs
- A devient noir
- Fin du 3[em] appel de DFS() dans la boucle de DFS_run()

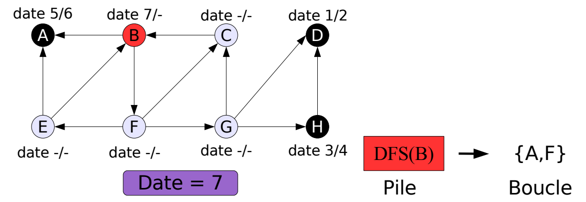


Date = 6

Pile Boucle

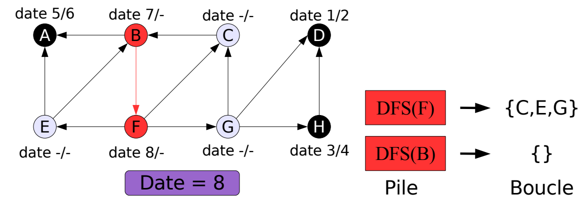
Exemple de décomposition

- A est le 4[em] sommet dans la boucle de DFS_run()
- Appel à DFS(B)
- B devient rouge.



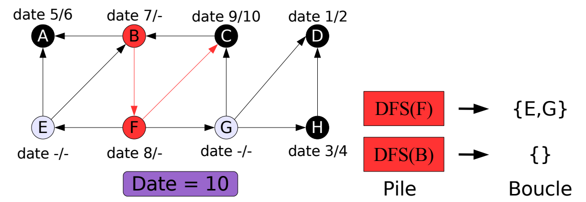
Exemple de décomposition

- Le sommet A n'est pas blanc ⇒ pas d'appel à DFS(A)
- Appel à DFS(F)
- F devient rouge



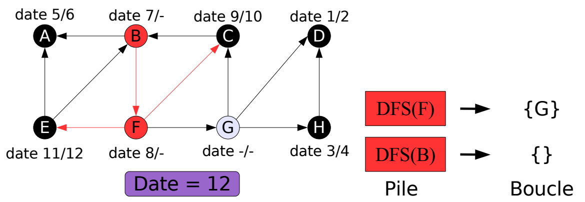
Exemple de décomposition

- Appel a DFS(C)
- C devient rouge
- C n'a pas de voisins blancs ⇒ C devient noir



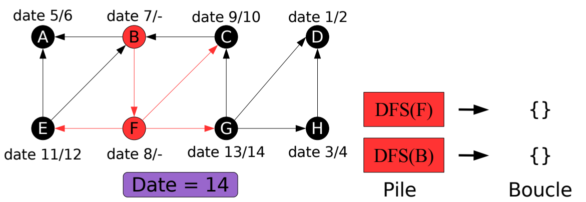
Exemple de décomposition

- Appel à DFS(E)
- E devient rouge
- E n'a pas de voisins blancs ⇒ E devient noir



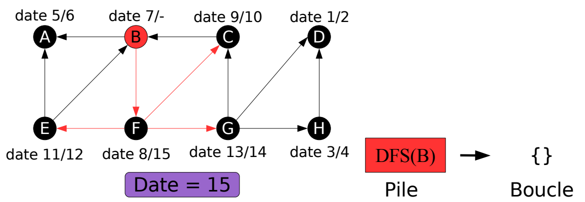
Exemple de décomposition

- Appel à DFS(G)
- G devient rouge
- G n'a pas de voisins blancs ⇒ G devient noir



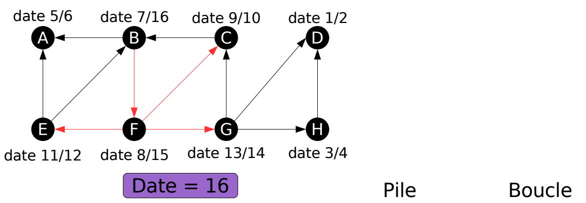
Exemple de décomposition

- Fin de la boucle dans la fonction DFS(F)
- F devient noir et on note la date : dateFin(F) ← 15



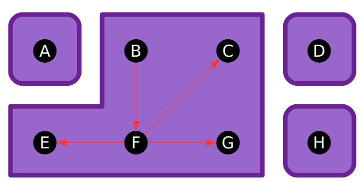
Exemple de décomposition

- Fin de la boucle dans la fonction DFS(F)
- F devient noir et on note la date : dateFin(F) ← 15



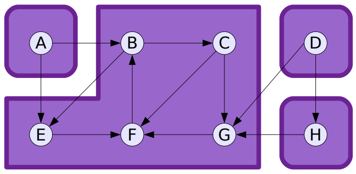
Exemple de décomposition

- Le résultat du deuxième parcours en profondeur est : une forêt de 4 arborescences



Exemple de décomposition

- Chacune de ces arborescences correspondent à :
1 **composante fortement connexe** du graphe de départ



Algorithme du tri topologique

- Les graphes sont utilisés pour représenter :

la précédence d'évènement

- Ces graphes forment une classe :

les graphes orientés acycliques

- Une séquence respectant cette précédence est :

tri topologique

- Si on aligne les sommets du graphe sur une droite dans l'ordre du tri topologique :

tous les arcs sont orientés vers l'avant

Algorithme du tri topologique

- Il y a deux algorithmes pour faire un tri topologique

1) On cherche un évènement qui peut se produire.
On enregistre cet évènement puis on recommence

Tri_topologique_1 (graphe G)
TANT-QUE resteSommet(G) **FAIRE**
chercher sommet s tq degréEntrant(s) = 0
enregistrer s
supprimer tous arcs sortant de s
supprimer s
FIN-TANT-QUE

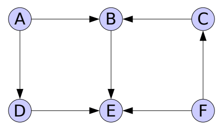
2) On utilse le parcours en profondeur

Tri_topologique_2 (graphe G)
DFS_run(G)
Ordonner les sommets suivant dateFin(s) par ordre décroissant

Exemple du tri topologique

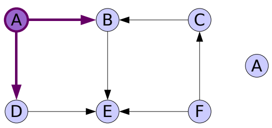
- On considère le graphe de précèdence suivant

ATTENTION :
Un graphe de précédence n'est pas un arbre.
Un arbre est **non orienté** connexe et acyclique.



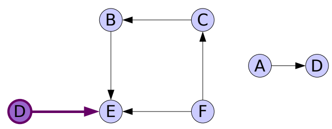
Exemple du tri topologique

- degreEntrant(A) = 0
- Enregistrement et supression du sommet A
- Supression des arcs partant de A



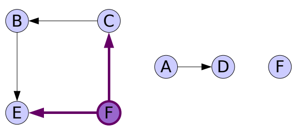
Exemple du tri topologique

- degreEntrant(D) = 0
- Enregistrement et supression du sommet D
- Supression des arcs partant de D



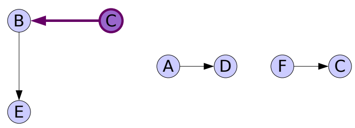
Exemple du tri topologique

- degreEntrant(F) = 0
- Enregistrement et supression du sommet F
- Supression des arcs partant de F



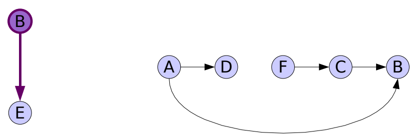
Exemple du tri topologique

- degreEntrant(C) = 0
- Enregistrement et supression du sommet C
- Supression des arcs partant de C



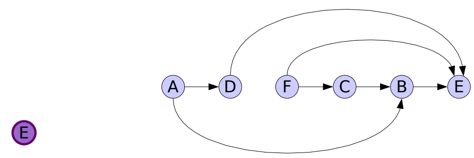
Exemple du tri topologique

- degreEntrant(B) = 0
- Enregistrement et supression du sommet B
- Supression des arcs partant de B



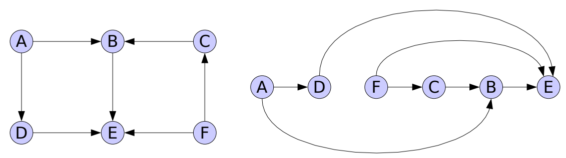
Exemple du tri topologique

- degreEntrant(E) = 0
- Enregistrement et supression du sommet E
- Supression des arcs partant de E



Exemple du tri topologique

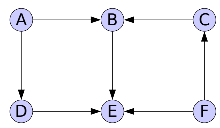
- Fin de l'algorithme
- On a obtenu un tri topologique du graphe de départ
- Tous les arcs sont orientés vers l'avant



Exemple du tri topologique

- On considère maintenant :
la **matrice d'adjacence** du même graphe de précédence

$$\begin{array}{cc} & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



Exemple du tri topologique

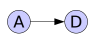
- degreEntrant(A) = 0 ⇔ colonne 1 ne contient que des 0
- Suppression du sommet A ⇔ suppression colonne 1
- Suppression des arcs partant de A ⇔ suppression ligne 1

$$\begin{array}{cc} & \begin{array}{cccccc} \boldsymbol{A} & B & C & D & E & F \end{array} \\ \begin{array}{c} \boldsymbol{A} \\ B \\ C \\ D \\ E \\ F \end{array} & \begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 0 & 1 & 0 \\ \mathbf{0} & 1 & 0 & 0 & 0 & 0 \\ \mathbf{0} & 0 & 0 & 0 & 1 & 0 \\ \mathbf{0} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Exemple du tri topologique

- degreEntrant(A) = 0 ⇔ colonne 1 ne contient que des 0
- Suppression du sommet A ⇔ suppression colone 1
- Suppression des arcs partant de A ⇔ suppression ligne 1

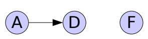
$$\begin{array}{cc} & \begin{array}{ccccc} B & C & \boldsymbol{D} & E & F \end{array} \\ \begin{array}{c} B \\ C \\ \boldsymbol{D} \\ E \\ F \end{array} & \begin{pmatrix} 0 & 0 & \mathbf{0} & 1 & 0 \\ 1 & 0 & \mathbf{0} & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 0 & 0 & \mathbf{0} & 0 & 0 \\ 0 & 1 & \mathbf{0} & 1 & 0 \end{pmatrix} \end{array}$$



Exemple du tri topologique

- degreEntrant(A) = 0 ⇔ colonne 1 ne contient que des 0
- Suppression du sommet A ⇔ suppression colonne 1
- Suppression des arcs partant de A ⇔ suppression ligne 1

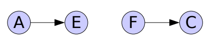
$$\begin{array}{cc} & \begin{array}{cccc} B & C & E & \boldsymbol{F} \end{array} \\ \begin{array}{c} B \\ C \\ E \\ \boldsymbol{F} \end{array} & \begin{pmatrix} 0 & 0 & 1 & \mathbf{0} \\ 1 & 0 & 0 & \mathbf{0} \\ 0 & 0 & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{pmatrix} \end{array}$$



Exemple du tri topologique

- degreEntrant(C) = 0 ⇔ colonne 2 ne contient que des 0
- Suppression du sommet C ⇔ suppression colone 2
- Suppression des arcs partant de C ⇔ suppression ligne 2

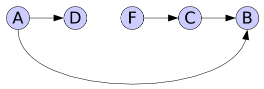
$$\begin{array}{cc} & \begin{array}{ccc} B & \boldsymbol{C} & E \end{array} \\ \begin{array}{c} B \\ \boldsymbol{C} \\ E \end{array} & \begin{pmatrix} 0 & \mathbf{0} & 1 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & \mathbf{0} & 0 \end{pmatrix} \end{array}$$



Exemple du tri topologique

- degreEntrant(B) = 0 ⇔ colonne 1 ne contient que des 0
- Suppression du sommet B ⇔ suppression colonne 1
- Suppression des arcs partant de B ⇔ suppression ligne 1

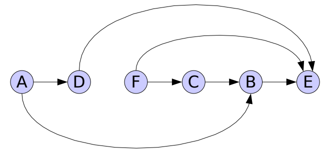
$$\begin{array}{c|cc} & \boldsymbol{B} & E \\ \hline \boldsymbol{B} & \mathbf{0} & \mathbf{1} \\ E & \mathbf{0} & 0 \end{array}$$



Exemple du tri topologique

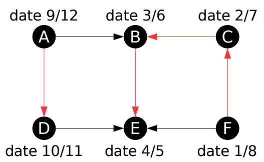
- degreEntrant(E) = 0 ⇔ colonne 2 ne contient que des 0
- Suppression du dernier sommet E ⇔ dernière case 0
- On a obtenu le même tri topologique

$$\begin{array}{c|c} & \boldsymbol{E} \\ \hline \boldsymbol{E} & \mathbf{0} \end{array}$$



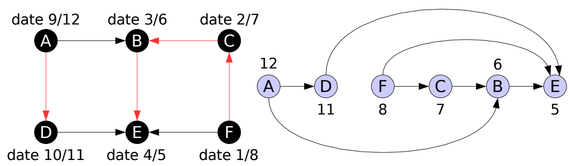
Exemple du tri topologique

- Deuxième algorithme de tri topologique
- On considère maintenant un parcours en profondeur sur le même graphe de précédence.



Exemple du tri topologique

- On aligne les sommets du graphe sur une ligne suivant les dates de fin dans l'ordre décroissant
- On obtient alors un tri topologique du graphe



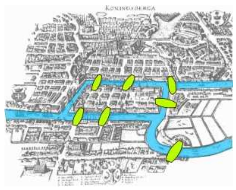
III

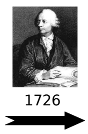
Graphes eulériens

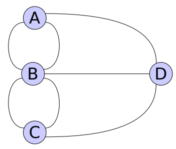
&

Graphes hamiltoniens

Problème des 7 ponts







« Lors d'une promenade, est-il possible de passer sur tous les ponts de la ville de Königsberg une et une seule fois ? » → *« Existe-t-il dans le graphe, un chemin où les arêtes sont différentes deux à deux et qui revient sur le sommet de départ ? »*

Lemme des poignées de mains

- Théorème - (lemme des poignées de main)

  i. La somme de tous les degrés est un nombre pair. C'est le double du nombre d'arêtes

  ii. Le nombre de sommets de degré impair est pair.

- Démonstration *(i)*
  - Chaque arête est comptée deux fois :
    Une fois pour le sommet de départ.
    Une fois pour le sommet d'arrivée.

Lemme des poignées de mains

- Démonstration *(ii)*
  - Soit S_{total} le nombre de sommets du graphe
  - Soit S_{imp} le nombre de sommets de degré impair
  - $$\begin{aligned} \textit{Somme des degrés} &= \sum_{i=1}^{S_{imp}} degImp_i + \sum_{i=S_{imp}+1}^{S_{total}} degPaire_i \\ &= \sum_{i=1}^{S_{imp}} (2\,k_i+1) + \sum_{i=S_{imp}+1}^{S_{total}} (2\,k_i) \\ &= 2\sum_{i=1}^{S_{total}} (k_i) + S_{imp} \end{aligned}$$
  - Somme des degrés est paire ⇒ S_{imp} est paire

Lacet de Jordan

- Dans un graphe non orienté, on dit qu'un chemin $(v_0, v_1, v_2, \ldots v_{k-1}, v_k)$ est un :
  - **Chemin de Jordan** si les arêtes qu'il emprunte sont distinctes deux à deux :

    $$\forall i, j \in [0,k-1],\ i \neq j \Rightarrow (v_{i}, v_{i+1}) \neq (v_{j}, v_{j+1})$$
  - **Lacet de Jordan** si c'est un chemin de Jordan

    $$\text{avec } v_0 = v_k$$
  - **Cycle de Jordan** si c'est un lacet de Jordan et si les sommets intermédiaires sont distincts 2 à 2

    $$\forall i, j \in [1,k-1],\ i \neq j \Rightarrow v_i \neq v_j$$

Graphe Eulérien

- On dit qu'un graphe non orienté est :
  - **eulérien** s'il existe un lacet de Jordan contenant toutes les arêtes du graphe.
  - **Semi-eulérien** s'il existe un chemin de Jordan contenant toutes les arêtes du graphe (mais pas de lacet de Jordan).
  - **Pré-eulérien** ou **chinois** s'il existe un lacet contenant au moins une fois chacune des arêtes du graphe.

Théorème de caractérisation

- Théorème de caractérisation :
  - *Un graphe connexe est **eulérien** ssi tous ses sommets sont de degré paire*
  - *Un graphe connexe est **semi-eulérien** ssi il ne contient que 2 sommets de degré impaire*

Démonstration

- *Eulérien ⇒ Tous les sommets ont un degré pair*
  - Eulérien ⇒ un lacet de Jordan qui passe par toutes les arrêtes.
  - En suivant ce lacet on passe par tous les arcs une et une seul fois
  - On suit ce lacet en enregistrant pour :
    - Le sommet départ :

      l'arc sortant ⇒ *DEG + 1*
    - Les sommets intermédiaires :

      l'arc entrant et l'arc sortant ⇒ *DEG + 2*
    - Le sommet d'arrivée :

      l'arc entrant ⇒ *DEG + 1*
  - Cycle ⇒ sommet départ = sommet arrivée ⇒ *DEG + 1 + 1*
  - Tous les degrés obtenus sont paires

Démonstration

- *Semi-eulérien ⇒ Exactement 2 degrés impairs*
  - On applique la même méthode
  - Semi-eulérien ⇒ sommet départ ≠ sommet arrivée
  - Si un sommet n'est ni le départ ni le sommet d'arrivée :
    - A chaque occurrence de ce sommet dans le chemin on fait

      DEG + 2
    - Le degré obtenu pour ce sommet est paire
  - Pour le sommet de départ (resp. d'arrivé) :
    - On fait *DEG + 1* au départ (resp. a l'arrivée)
    - A chaque occurrence de ce sommet dans le chemin on fait

      DEG + 2
    - Le degré obtenu pour ce sommet est impaire

      DEG = (nbOccurrences x 2) + 1

Démonstration

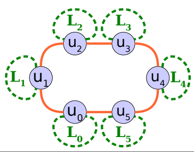
- **Lemme :**

  Si tous les sommets ont un degré pair, on peut toujours étendre un chemin de Jordan vers un lacet de Jordan
- Démonstration :
  - Soit un chemin de Jordan de u a v si $u \neq v$ alors :

    On a emprunter un nombre impaire arêtes de v
  - Puisque par hypothèse v a un nombre paire d'arêtes, il reste au moins une arrête qui n'appartient pas au chemin de Jordan.
  - Donc $u \neq v$ ⇒ on peut étendre le chemin
  - Or il y a un nombre fini d'arêtes ⇒ extension pas infini
  - On finit donc par avoir $u = v$
  - On peut toujours étendre ce chemin vers un lacet de Jordan

Démonstration

- Tous les sommets ont un degré pair ⇒ Eulérien
- Raisonnements par récurrence sur n le nombre d'arêtes
  - **Pour n = 1**, il n'existe que deux graphes : (A) (A)—(D)
    - Seul le premier n'a que des degrés paires et il est Eulérien
  - **Supposons la proposition vraie pour les graphes à n-1 arêtes**
    - D'après le lemme on peut construire un lacet de Jordan
    - Les arêtes n'appartenant pas au lacet forment des **comp. connexes**



- Dans ces **composantes** tous les degrés sont paires
- Par hypothèse de récurrence **elles** sont Eulériennes
- Soit L_i les lacets de Jordan les couvrant totalement
- $u_0\ L_0\ u_0\ u_1\ L_1\ u_1\ u_2\ L_2\ u_2\ u_3\ L_3\ u_3\ u_4\ L_4\ u_4\ u_5\ L_0\ u_5\ u_0$

  Forme un lacet de Jordan qui couvre tout le graphe

  ⇒ **le graphe est Eulérien**

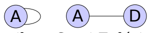
Démonstration

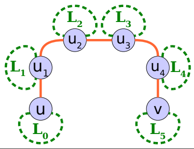
- **Lemme :**

  Si exactement 2 sommets u et v ont un degré impair, on peut toujours étendre un chemin de Jordan partant de u vers un chemin de Jordan reliant u et v
- Démonstration :
  - Soit un chemin de Jordan de u a w
    - si w ≠ v et w ≠ u alors : On a emprunté un nombre impair d'arêtes de w qui avait par hypothèse un nombre pair d'arêtes.

      si w = u :On a emprunté un nombre pair d'arêtes de u qui avait par hypothèse un nombre impair d'arêtes.
  - Dans les 2 cas il reste au moins une arête qui n'appartient pas au chemin de Jordan. ⇒ on peut étendre le chemin
  - Or il y a un nombre fini d'arêtes ⇒ extension pas infinie
  - On finit donc par avoir w = v et donc chemin de Jordan reliant u et v

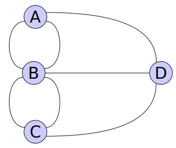
Démonstration

- 2 sommets (u et v) avec un degré impair ⇒ Semi-Eulérien
- Raisonnements par récurrence sur n le nombre d'arêtes
  - **Pour n = 1**, il n'existe que deux graphes : 
    - Seul le deuxième a deux degrés impairs et il est Semi-Eulérien
  - **Supposons la proposition vraie pour les graphes à n-1 arêtes**
    - D'après le lemme on peut construire un **chemin de Jordan**
    - Les arêtes n'appartenant pas au chemin forment des **comp. connexes**



- Dans ces **composantes** tous les degrés **sont pairs**
- Par hypothèse de récurrence **elles** sont **Eulériennes**
- Soit L_i les **lacets** de Jordan les couvrant totalement
- **$u\ L_0\ u\ u_1\ L_1\ u_1\ u_2\ L_2 u_2\ u_3\ L_3\ u_3\ u_4\ L_4\ u_4\ v\ L_0\ v$**
  Forme un chemin de Jordan qui couvre tout le graphe
  ⇒ **le graphe est Eulérien**

Les 7 ponts : La solution

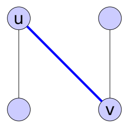


- Degré(A) = 3
- Degré(B) = 5
- Degré(C) = 3
- Degré(D) = 3

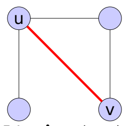
- Des théorèmes précédents on peut déduire que :
  - Königsberg n'est pas un graphe Eulérien
  - Königsberg n'est pas un graphe Semi-Eulérien
- Il n'y a pas de promenade possible
- Et ce même si on ne revient pas au point départ

Définition : Pont

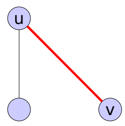
- Dans un graphe non orienté connexe, on dit qu'une arête est un **pont** si, lorsqu'on la retire en effaçant les sommets devenus isolés le nouveau graphe obtenu n'est plus connexe



L'arête (u,v) est **un pont**



L'arête (u,v) n'est pas un pont



L'arête (u,v) n'est pas un pont

Algorithme de Fleury

Fleury (graphe *G*)

Choisir un sommet u

TANT-QUE degré(u) ≠ 0 **FAIRE**
 Choisir une arête (u,v) qui n'est **pas un pont**
 Effacer (u,v)
 SI degré(u)=0 **ALORS**
  effacer u
 FIN SI
 $u \leftarrow v$
FIN TANT-QUE

Liveness & Safety

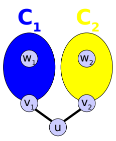
- Pour démontrer un algorithme, il suffit de démontrer deux propriétés :
  - La propriété de **vivacité** : **Liveness**

    « Il peut toujours arriver quelque chose de bien »
  - La propriété de **sûreté** : **Safety**

    « Il n'arrive jamais quelque chose de mauvais »

Démonstration : Liveness

- *« degré(u) > 0 ⇒ on peut choisir un (u,v) qui n'est pas un pont »*
  - Par définition : s'il y a un pont ⇒ degré(u) > 1
  - Uniquement des ponts ⇒ au moins deux ponts
  - Soit C_i la composante connexe au bout du pont (u,v_i)
    - Le degré de v_i dans le graphe connexe C_i est impair.
    - D'après le lemme des poignées de main :
      il existe dans C_i un autre sommet w_i de degré impair.



  - Que des ponts ⇒ au moins deux sommets de degré impair.
  - Or un graphe est eulérien ssi tous ses degrés sont pairs.
  - Au cours du parcours il ne peut y avoir en plus du sommet u qu'au plus un autre sommet au degré impair (l'origine du parcours).
  - Donc il ne peut pas y avoir plus d'un pont partant du sommet u
  - Par contraposé : pas plus d'un pont ⇒ pas uniquement des ponts

Démonstration : Safety

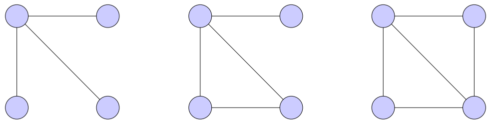
- *« degré(u) = 0 ⇒ on a parcouru toutes les arêtes du graphe »*
  - L'algorithme est ainsi fait qu'à tout instant le graphe reste connexe.
  - Or si un graphe connexe contient un point isolé, c'est qu'il est réduit à cet unique point isolé.
  - Cela signifie que la dernière arête que l'on vient d'effacer était aussi la dernière du graphe.
  - Comme les arêtes ne sont effacées qu'après avoir été parcourues : Toutes les arêtes du graphe on été parcourues une et une seule fois.

Graphe Hamiltonien

- On dit qu'un graphe non orienté connexe est :
  - **hamiltonien** s'il existe un cycle de Jordan contenant toutes les sommets du graphe.
  - **semi-hamiltonien** s'il existe un chemin de Jordan élémentaire contenant toutes les sommets du graphe (mais pas de cycle de Jordan).
- Rappel :
  - Un chemin $(v_0, v_1, v_2, \ldots v_{k-1}, v_k)$ est **élémentaire** ssi $\forall i, j,\ v_i \neq v_j$
  - Un cycle est toujours élémentaire

Exemple

Les graphes suivants sont :



Non Hamiltonien — **Semi Hamiltonien** — **Hamiltonien**

Caractérisation

- Contrairement au cas des graphes eulériens : on n'a encore trouvé aucune **condition nécessaire et suffisante** assurant qu'un graphe est hamiltonien ou semi-hamiltonien.
- Il existe, cependant, de nombreux théorèmes donnant des **conditions suffisantes**.

Caractérisation

- **Théorème de caractérisation de O. Ore** :
  - *Soit G un graphe simple possédant n>2 sommets :*

    $\forall$ u,v non adjacents, degré(u) + degré(v) $\geq$ n

    $\Rightarrow$

    Le graphe G est Hamiltonien
- Rappel :
  - Un graphe est **simple** s'il ne contient pas de boucle et que deux sommets sont reliés par au plus une arête.
  - La propriété intéressante d'un graphe simple : degré(s) = nbVoisin(s)

Caractérisation

- **Corollaire de Dirac** :
  - *Soit G un graphe simple possédant n>2 sommets :*

    $\forall$ u, degré(u) $\geq$ n/2

    $\Rightarrow$

    Le graphe G est Hamiltonien

IV

Graphes planaires

&

Le théorème des 4 couleurs

Théorème des 4 couleurs

Le coloriage est une activité de détente réservée aux enfants...

Pas en mathématiques !

Théorème des 4 couleurs

- **Le théorème des 4 couleurs :**

  Toute carte de géographie est coloriable avec quatre couleurs sans que deux régions frontalières n'aient pas la même couleur.
- Ce théoreme pourrait appartenir aux:

  « conjectures pour les nuls »
- C'est à son apparente simplicité qu'il doit sa popularité
- La démonstration de cette conjecture semblait accessible à *Mr ToutLeMonde*

Domaines d'applications

- Dans la téléphonie mobile :

  Les questions de coloriage permettent de réduire les fréquences d'émissions utilisées.

  1 couleur = 1 fréquences

Historique : La conjecture

- **1852** - Un cartographe anglais **Francis Guthrie**, remarque en coloriant la carte des cantons anglais, qu'il lui suffit de quatre couleurs pour que deux cantons ayant une frontière commune n'aient pas la même couleur.
- Il fait part de cette observation à son frère mathématicien.
- **Frederick Guthrie** en parle à son professeur **De Morgan**.
- **Première trace écrite** :

  lettre de De Morgan à Sir Hamilton.
- **1878** : **Arthur Cayley** publie la conjecture aux :
  « Société mathématique de Londres »
  « Société géographique »

Historique : La conjecture

Découpage
de l'Angleterre,
du Pays de Gales
et de l'Ecosse
avant les changements
de frontièreen **1974**

Historique : Une preuve ?

- **1879** : un avocat anglais **Alfred Bray Kempe** publie une démonstration du théorème.
- Kempe reçoit une décoration pour cette preuve.
- **1890** : un avocat anglais **John Heawood** trouve une faille dans la preuve de Kempe.
- Il démontre alors le théorème pour 5 couleurs.
- Il élargit le problème à d'autres surfaces : tore, ruban, ...
- Heawood est décoré pour la restauration d'un château.

Historique : Par la force

- **1971** : Première tentative d'utilisation de la puissance informatique par le japonais **Matsumoto**.
- **1976** : **Kenneth Appel et Wolfgang Haken** établissent enfin une preuve du théorème des 4 couleurs.
- Leur démonstration du théorème se basait sur une approche mathématique conventionnelle et utilisait un ordinateur dans le seul but de venir à bout de plus d'un milliard de combinaisons de calculs :
  - Mathématiquement ils montrent qu'il y a 1478 *configurations inévitables* dans une carte géographique
  - Puis avec 1200 heures de calcul qu'elles sont coloriables.

Historique : Preuve assistée

- **1995** : **Robertson, Sanders, Seymour et Thomas.** diminuent le nombre de configurations 633 et automatisent une partie de la démonstration («inévitabilité»)
- **2005** : **Georges Gonthier** et **Benjamin Werner (INRIA)** s'attaquent au problème sous un angle différent :
  ils utilisent exclusivement des outils d'aide à la preuve.
- **Coq** : Un outil informatique capable d'effectuer et de vérifier la démonstration étape par étape, s'affranchissant du moindre risque d'erreurs de programmation.

Les limites

- Ce théorème a ses limites :

  Il ne doit pas y avoir de contraintes extérieures sur les choix des couleurs.
- Dans la pratique ce n'est pas toujours le cas :
  - La mer et les lacs doivent être bleus.
  - Certains pays peuvent avoir des enclaves.

Théorème des enclaves

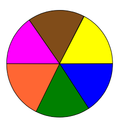
- Théorème :

  Si chacune des régions d'une carte de géographie est constituée de 1 ou 2 morceaux (au plus une enclave), il est toujours possible de la colorier avec 12 couleurs de façon à ce que deux régions frontalières n'aient pas la même couleur.

Graphe d'incidence

- Pour modéliser ce problème on peut utiliser la théorie des graphes.
- A chaque carte on associe un **graphe d'incidence**
  - A chaque pays correspond un sommet
  - A chaque frontière correspond une arête

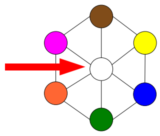
ATTENTION :
On ne considère pas les frontières réduites à un seul point.
*Ex : Pas de frontière entre le **camembert « sport »** et le **camembert « art et littérature »***



Graphe d'incidence

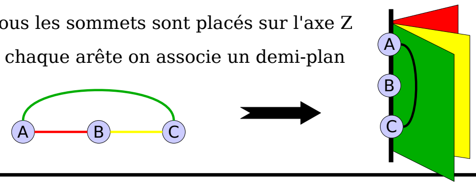
- Pour modéliser ce problème on peut utiliser la théorie des graphes.
- A chaque carte on associe un **graphe d'incidence**
  - A chaque pays correspond un sommet
  - A chaque frontière correspond une arête

ATTENTION :
Ne pas oublier le pays « bordure »



Graphe planaire

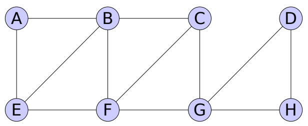
- On dit d'un graphe qu'il est **planaire** si :
  - S'il existe une représentation dans un plan de ce graphe tel que les arêtes ne s'entrecoupent pas.
- Par contre, on peut toujours représenter un graphe dans l'espace tel que les arêtes ne se croisent pas :
  - Tous les sommets sont placés sur l'axe Z
  - A chaque arête on associe un demi-plan



Identité d'Euler

- Pour tout graphe planaire connexe on a :

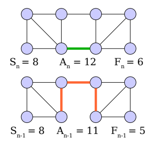
  l'identité d'Euler : **S – A + F = 2**
- **S** : le nombre de sommets
- **A** : le nombre d'arêtes
- **F** : le nombre de faces, ie le nombre de régions délimitées par des arêtes, y compris la face extérieure la seule à ne pas être bornée



- S = 8
- A = 12
- F = 6
- S – A + F = 8 – 12 + 6 = 2

Démonstration

- *Graphe connexe planaire* ⇒ S – A + F = 2
- Raisonnements par récurrence sur *n* le nombre d'arêtes
  - **Pour *n = 0*,** la connexité impose un graphe réduit à un sommet : A
    - S = 1, A = 0 et F = 1 ⇒ **S - A + F = 1 – 0 + 1 = 2**
  - **Supposons la proposition vraie pour les graphes à *n-1* arêtes**
    - Soit un graphe connexe planaire à *n* arêtes. Si on supprime 1 arête :
      - Si **l'arête supprimée** est bordée par deux faces :



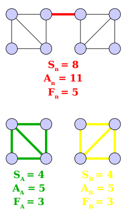
- Une seule face est non bornée ⇒ 1 de ces faces est bornée
- **Le reste du contour** de cette face maintient la connexité.
- $S_n = S_{n-1}$ mais on a diminué : $A_n - 1 = A_{n-1}$ et $F_n - 1 = F_{n-1}$
- Par hypothèse de récurrences on a : $S_{n-1} - A_{n-1} + F_{n-1} = 2$
- $S_{n-1} - A_{n-1} + F_{n-1} = S_n - (A_n - 1) + (F_n - 1) = \mathbf{S_n - A_n + F_n = 2}$

Démonstration (suite)

- *Graphe connexe planaire* ⇒ S – A + F = 2
  - Si l'arête supprimée est bordée par une face :
    - Alors cette face est la face extérieure non bornée
    - On obtient alors 2 composantes connexes : G_A et G_B.
    - Elles sont planaires donc par hypothèse de récurrences on a :
      - $\mathbf{S_A - A_A + F_A = 2}$ et $\mathbf{S_B - A_B + F_B = 2}$
    - On a partitionné les sommets : $S_A + S_B = S_n$
    - On a supprimé une arête : $A_A + A_B = A_n - 1$
    - La face extérieure est commune : $F_A + F_B = F_n + 1$
      - On a donc : $S_A + S_B - (A_A + A_B) + F_A + F_B = 2 + 2$

        $S_n - (A_n - 1) + F_n + 1 = 4$

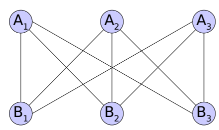
        $\mathbf{S_n - A_n + F_n = 2}$



Graphes non planaires

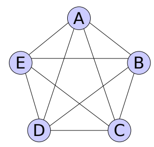
- **1930** le mathématicien polonais **Kuratowski** montre :
  - Tout graphe connexe non planaire contient un sous graphe homéomorphe à l'un des 2 graphes suivants:

K_{33}



Le graphe bipartie avec deux ensembles à 3 sommets «3 maisons reliées à 3 usines»

K_5



Le graphe complet à 5 sommets (reliés 2 à 2)

Démonstration : K_5

- Chaque face d'un graphe planaire a au moins 3 cotés :

  ⇒ A ⩾ 3F
- Si l'on considère l'identité d'Euler : S – A + F = 2
  - F = 2 – S + A ⇒ A ⩾ 6 – 3S +3A ⇒ 3S – 6 ⩾ A
- **K_5**
  - S=5 et A=4+3+2+1=10
  - 3 x 5 – 6 < 10
  - **N'est pas planaire**
- **K_{33}**
  - S=6 et A=9
  - 3 x 6 – 6 ⩾ 9
  - **Pourrait être planaire**

Démonstration : K_{33}

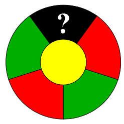
- Dans **K_{33}** un chemin de longueur 3 ne peut être fermé
- Les faces du graphe **K_{33}** ne peuvent être triangulaires:

  ⇒ A ⩾ 4F
- Si l'on considère l'identité d'Euler : S – A + F = 2
  - F = 2 – S + A ⇒ A ⩾ 8 – 4S +4A ⇒ 4S – 2 ⩾ A
  - **K_{33}**
    - S=6 et A=9
    - 3 x 6 – 6 < 9
    - **N'est pas planaire**

La remarque de Morgan

- Comme K_5 n'est pas planaire :
  - Un graphe d'incidence ne peut avoir :
    5 sommets reliés 2 à 2
  - Une carte de géographie ne peut avoir :
    5 pays mutuellement frontaliers

ATTENTION :
Cette propriété n'est pas suffisante pour démontrer le théorème des 4 couleurs.
Ex : On peut construire une carte où il n'y a pas 4 pays mutuellement frontaliers et où 3 couleurs ne sont pas suffisantes.



Démonstration de Kempe

- Théorème :

  Tout graphe planaire possède au moins un sommet de degré inférieur ou égal à 5

- Démonstration par l'absurde :
  - Soit G un graphe tel que : $\forall s \in S,\ deg(s) > 5$
  - La somme des degrés de G est donc : $\sum_{s \in S} deg(s) \geqslant 6S$
  - D'après le lemme des poignées de mains, on a :

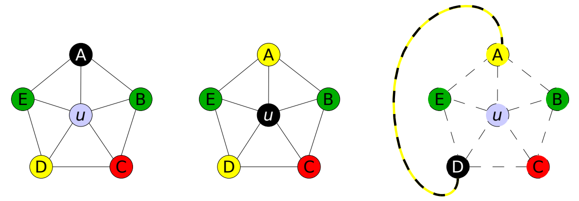
    $2A \geqslant 6S \Rightarrow A \geqslant 3S$
  - Or on a vu que dans un graphe planaire on a :

    $3S - 6 \geqslant A \Rightarrow 3S - 6 \geqslant 3S$

Démonstration de Kempe

- **La démonstration de Kempe :**
- Raisonnements par récurrence sur n le nombre de sommets.
  - **Pour $n < 5$,** le résultat est évident.
  - **Supposons la conjecture vraie pour les graphes à n-1 sommets**
    - Soit G un graphe planaire avec n sommets
    - Il existe au moins un sommet u de G de degré inférieur ou égale à 5.
    - Soit G' le graphe obtenu par surpression de u dans G
    - Par hypothèse de récurrence, il existe un coloriage de G' à 4 couleurs
    - Si les voisins u n'utilisent que 3 couleurs dans le coloriage de G' :
      - En utilisant une couleur libre pour u on obtient un coloriage de G.
    - Sinon on va essayer de modifier le coloriage de G' :
      - Si dégage une couleur pour u, on obtiendra un coloriage de G.

Démonstration de Kempe

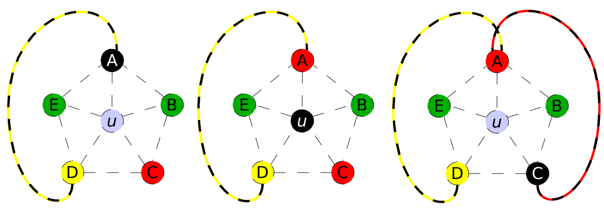


On va colorier A en jaune
⇒
De proche en proche
noir → jaune
jaune → noir

Si D reste B=jaune
***u* peut devenir noir**

Si D devient noir quand A devient jaune
⇒
Il y a une chaîne noir et rouge entre A et C

Démonstration de Kempe

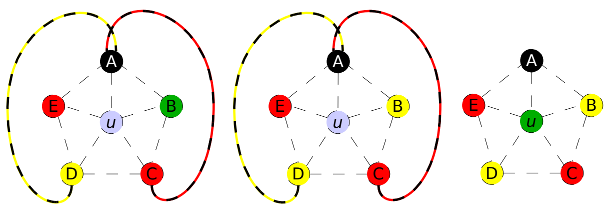


On va colorier A en rouge
⇒
De proche en proche
noir → rouge
rouge → noir

Si C reste rouge
***u* peut devenir noir**

Si C devient noir quand A devient rouge
⇒
Il y a une chaîne noir et rouge entre A et C

Démonstration de Kempe

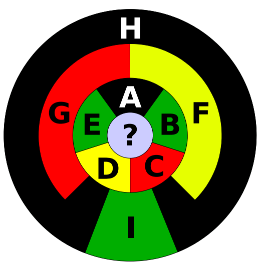


Il y a une chaîne noir et jaune entre A et D
⇒
Il n'y a pas de chaîne verte et rouge entre E et C.
E peut devenir rouge

Il y a une chaîne noir et rouge entre A et C
⇒
Il n'y a pas de chaîne verte et jaune entre B et D.
B peut devenir jaune

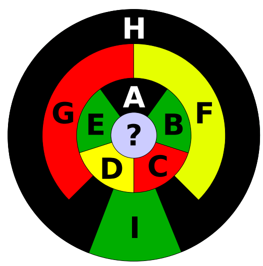
***u* peut devenir vert**

Effet de bord : Rupture de chaîne



- Comme K_5 n'est pas planaire :
- Un graphe d'incidence ne peut avoir

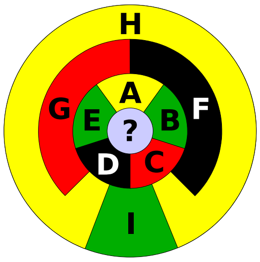
Effet de bord : Rupture de chaîne



Colorier **A** en jaune
⬇
Colorier **F** en noir
⬇
Colorier **H** en jaune
⬇
Colorier **D** en noir
⬇
Il y a une chaîne :
noir-jaune

A – F – H – D

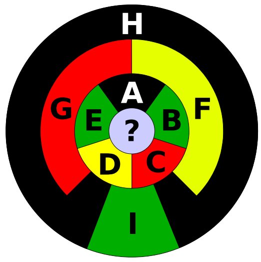
Effet de bord : Rupture de chaîne



Colorier **A** en jaune
⬇
Colorier **F** en noir
⬇
Colorier **H** en jaune
⬇
Colorier **D** en noir
⬇
Il y a une chaîne :
noir-jaune

A – F – H – D

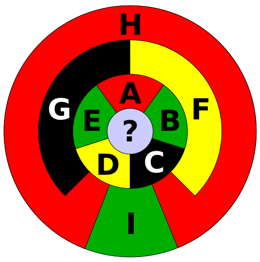
Effet de bord : Rupture de chaîne



Colorier **A** en rouge
⬇
Colorier **G** en noir
⬇
Colorier **H** en rouge
⬇
Colorier **C** en noir
⬇
Il y a une chaîne :
noir - rouge

A – G – H – C

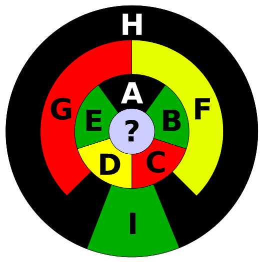
Effet de bord : Rupture de chaîne



Colorier **A** en rouge
⬇
Colorier **G** en noir
⬇
Colorier **H** en rouge
⬇
Colorier **C** en noir
⬇
Il y a une chaîne :
noir - rouge

A – G – H – C

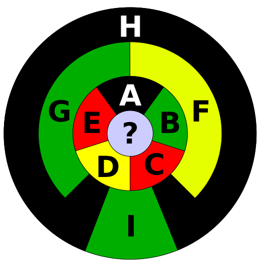
Effet de bord : Rupture de chaîne



Colorier **E** en rouge
⬇
Colorier **G** en vert

C reste rouge

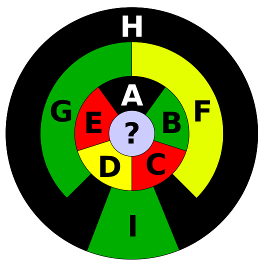
Effet de bord : Rupture de chaîne



Colorier **E** en rouge
⬇
Colorier **G** en vert

C reste rouge

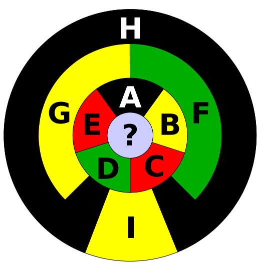
Effet de bord : Rupture de chaîne



Colorier **B** en jaune
⬇
Colorier **F** en vert
⬇
Colorier **G** en jaune
⬇
Colorier **D** en vert
⬇
Il y a une chaîne :
vert-jaune

B – F – G – D

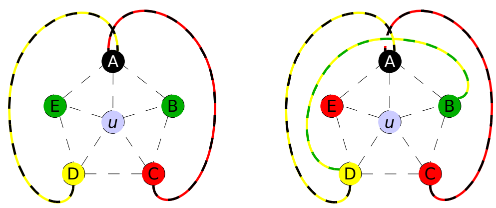
Effet de bord : Rupture de chaîne



Colorier **B** en jaune
⬇
Colorier **F** en vert
⬇
Colorier **G** en jaune
⬇
Colorier **D** en vert
⬇
Il y a une chaîne :
vert-jaune

B – F – G – D

Effet de bord : Rupture de chaîne



L'argument de Kempe repose sur l'existence de deux chaînes :
- noir et jaune entre A et D
- verte et jaune entre B et D

Effet de bord possible quand E devient rouge : une rupture dans a chaîne AC
⇒
rien n'empêche l'existence d'une **chaîne verte et jaune entre B et D**.

Principe d'une démonstration

- La démonstration de Kempe est fausse mais le principe est bon :
  - trouver un ensemble de sous-graphes tel que :

    tout graphe graphe d'incidence contienne
    au moins un de ses sous-graphes

    C'est l'ensemble des **configurations inévitables**
  - Par récurrence si l'on supprime une configuration du graphe :

    on obtient un coloriage du graphe
    réduit de cette configuration inévitable.
  - Déduire de cette coloration :

    une coloration du graphe totale

    la configuration est **réductible**
  - Preuve = toutes les configurations inévitables sont réductibles

Théorème des 5 couleurs

- C'est avec ce type de preuve que :

John Heawood démontra le théorème pour 5 couleurs

  - Il exhibe : 5 configurations inévitables
  - Il démontre leur réductibilité
- Cette démonstration sera vue en TD