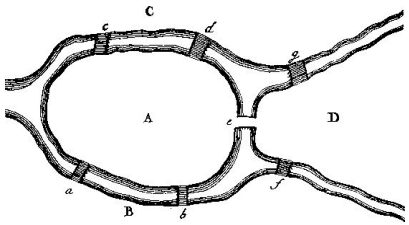
Graphes et Applications



Plan du cours

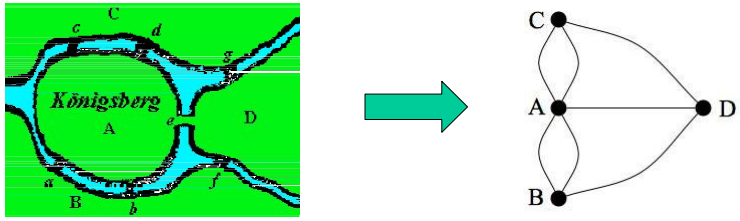
- Généralités
- Coloration - Planarité
- Problème de l'arbre couvrant minimal
- Problème du plus court chemin
- Ordonnancement

Pourquoi la théorie des graphes?

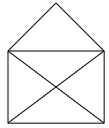
- Modélisation
  - Plusieurs problèmes dans différentes disciplines (Réseaux informatiques, télécommunications, chimie, applications industrielles, …)
  - Un graphe peut représenter simplement la structure, les connexions, les cheminements possibles d'un ensemble complexe comprenant un grand nombre de situations
- Un graphe est une structure de données puissante pour l'informatique

Bref historique de la théorie des graphes

- Naissance en 1736, communication d'Euler ou il propose une solution au problème des ponts de Konigsberg

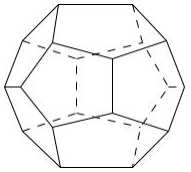


- Dessin comportant des sommets (points) et des arêtes reliant ces sommets



Bref historique de la théorie des graphes

- 1847 Kirchhoff
  théorie des arbres (analyse de circuits électriques)
- 1860 Cayley
  énumération des isomères satures des hydrocarbures C_nH_{2n+2}
- A la même époque, énoncé de problèmes importants
  - **Conjecture des quatre couleurs (1879)**
    (Mobius, De Morgan, Cayley, solution trouvée en 1976)
  - **Existence de chemins Hamiltoniens (1859)**
- 1936 Konig
  premier ouvrage sur les graphes
- 1946 ⟹ **Kuhn, Ford, Fulkerson, Roy**
  Berge



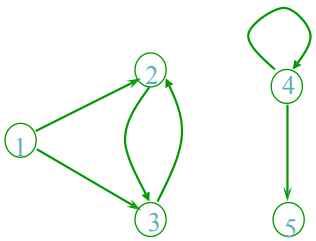
Généralités

- Définitions
- Chaînes et chemins
- Connexité (graphe non orienté)
- Forte connexité
- Représentation des graphes
- Distance dans un graphe non orienté
- Les 7 ponts de Konigsberg

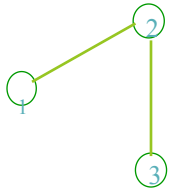
Généralités
Définitions

- Un *graphe orienté* est un couple ***G=(S,A)*** :
  S ensemble de ***sommets*** ou ***nœuds*** ($|S|=n$);
  $A \subset S \times S$ ensemble *d'**arcs*** ($|A|=m$).

  Exp:
  $S = \{ 1, 2, 3, 4, 5 \}$
  $A = \{ (1, 2), (1, 3), (2, 3), (3, 2), (4, 4), (4, 5) \}$



- Pour un arc $a = (i, j)$, i est l'extrémité initiale, j l'extrémité finale
  (ou bien ***origine*** et ***destination***).
- Un *graphe non orienté* est un couple ***G=(S,A)*** :
  $A \subset S \times S$ ensemble *d'**arêtes***



Généralités
Définitions

- Un *graphe orienté* est un couple ***G=(S,A)*** :
  - On appelle graphe valué et on note G=(S,A,c) un graphe associe une fonction c : A □ IR appelée coût ou poids ou valeurs des arcs.
    Le coût d'un arc (i,j) est noté c(i,j) ou cij
  - Un graphe est dit symétrique si : $\forall$ i, j ∈ S, (i, j)∈ A ⇒ (j, i)∈ A.
    i.e. un graphe est symétrique lorsque chaque paire de sommets reliés dans un sens l'est aussi dans l'autre
  - Un graphe est dit complet si $\forall$ i,j ∈S, (i, j) ∉A ⇒ (j, i) ∈A.
    i.e. Un graphe est complet si deux sommets quelconques sont reliés dans au moins une direction

Généralités
Définitions

- j est successeur ou suivant de i si (i, j)∈ A ; l'ensemble des successeurs de i est noté S(i) ou $\Gamma^+(i)$.

$$\Gamma^+(i)=\{j \in S \setminus (i,j) \in A\}$$

- j est prédécesseur ou précédent de i si (j, i) ∈ A; l'ensemble des prédécesseurs de I est noté P(i) ou $\Gamma^-(i)$.

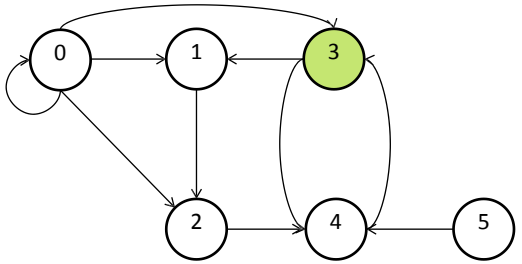
$$\Gamma^-(i)=\{j \in S \setminus (j,i) \in A\}$$

- Degrés
  - Le degré extérieur de i, d +(i), est le nombre de sommets suivants de i ; d+(i) =|S(i)|.
  - Le degré intérieur de i, d -(i), est le nombre de sommets précédents de i; d -(i) = |P(i)|.

Le degré de i est d(i) = d+(i) + d-(i).

Généralités
Définitions

Exemple:

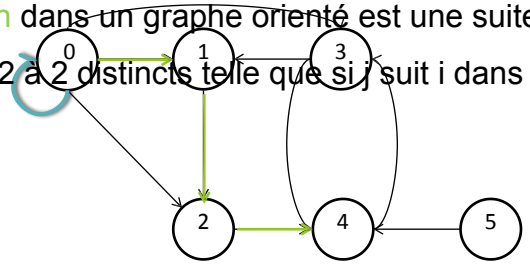


- $d^+(3) = |S(3)| = 2.$
- $d^-(3) = |P(3)| = 2.$

$\Rightarrow$ $d(3) = d^+(3) + d^-(3) = 4.$

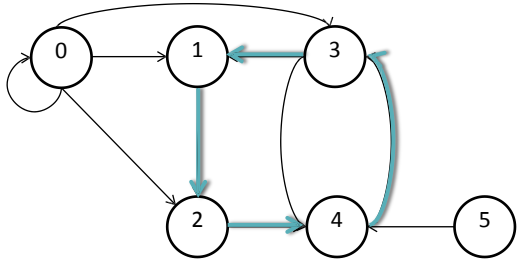
Généralités
Définitions

- Deux sommets sont adjacents (ou voisins) s'ils sont reliés par un arc.
- Deux arcs sont adjacents s'ils ont au moins une extrémité commune.
- Un arc (i, i) est appelé une boucle.
- Un chemin dans un graphe orienté est une suite (x,…,y) de sommets 2 à 2 distincts telle que si j suit i dans cette suite, (i,j) est un arc

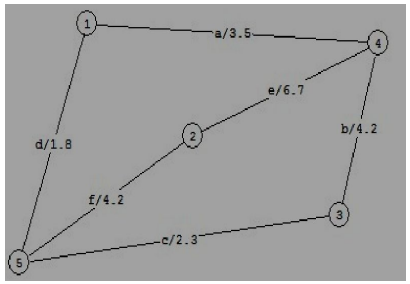


Généralités
Définitions

- Une chaîne est dans un graphe orienté (non orienté) est une suite (x,…,y) de sommets 2 à 2 distincts telle que si i suit j dans cette suite, (i,j) ou (j,i) est un arc (une arête).
- Longueur d'un chemin (ou d'une chaîne) : nombre d'arcs du chemin (ou d'arêtes de la chaîne)
- Un chemin qui se referme sur lui–même est un circuit.
- Une chaine qui se referme sur elle–même est un cycle.

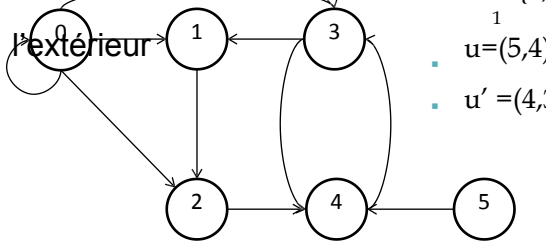


Exemple:



- Chaîne
- Longueur de la chaîne
- 1-4-3-5-1 cycle
- 5-4-3-5 n'est pas un cycle
- 1-4-2-5-3-4-1 □ n'est pas un cycle, l'arête 1-4 est parcourue deux fois □ chaîne fermée

- i est dit ascendant de j s'il existe un **chemin** d'origine i et de destination j; j est alors un descendant de i .
- Soit S1⊂ S. On dit qu'un arc u est incident à S1 de l'extérieur si son extrémité terminale est dans S1 et son extrémité initiale dans S\ S1
- Il est dit incident à S1 de l'intérieur s'il est incident à S\S1 de l'extérieur



- u=(5,4) est incident à S_1 de l'extérieurr
- u' =(4,3) est incident à S_1 de l'intérieur

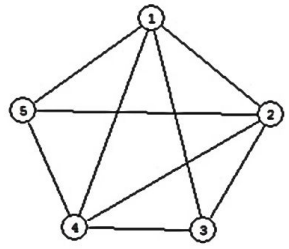
- DISTANCE: la distance entre deux sommets d'un graphe est la plus petite longueur des chaînes, ou des chemins, reliant ces deux sommets.

NB:

- Distance(x,x)=0
- Soit x et y ∈X et s'il n'existe pas une chaîne entre x et y alors distance(x,y)=∞

- ORDRE D'UN GRAPHE : nombre de sommets du graphe
- DIAMÈTRE : Le diamètre d'un graphe est la plus longue des distances entre deux sommets.

- Sous-graphe: G'(S',A') est un sous-graphe de G(S,A) ssi S'⊆S et A' est l'ensemble des arêtes de A reliant deux sommets de S'.
- Un sous graphe stable est un sous-graphe sans arêtes
- Exemple:
  - G'(S'={1,3,5}, A'={{1,3},{1,5}}) est un sous graphe
  - G'(S'={5,2,1}, A'{{5,2},{2,1},{5,1}}) est un sous graphe complet
  - G'(S'={1,5,4,2}, A'{{5,1},{2,5},{5,4},{2,1},{2,4},{2,1}}) est un sous graphe complet



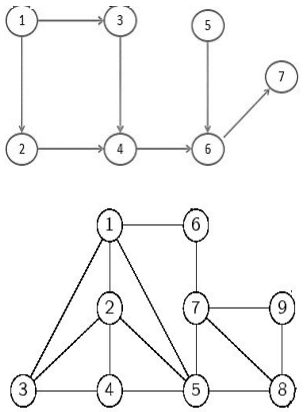
Généralités
Connexité

- Un graphe G(S,A) est dit simplement connexe ou connexe si ∀ i,j∈ S, il existe une chaîne entre i et j.
- Un graphe orienté est connexe si le graphe non orienté associé est connexe.
- Une composante connexe C d'un graphe G = (S,A) est un sous-ensemble maximal de sommets tels que deux quelconques d'entre eux soient reliés par une chaîne : si i∈C, alors :
  - ∀j ∈C, il existe une chaîne reliant i à j,
  - ∀k ∈S\C, il n'existe pas de chaîne reliant i à k.

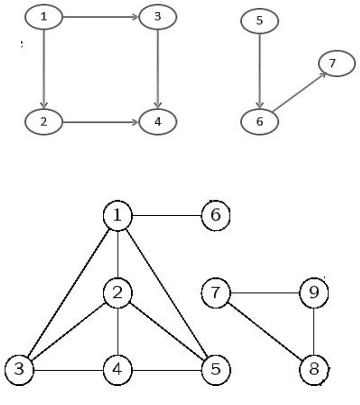
Généralités
Connexité

EXEMPLE:

Graphes connexes

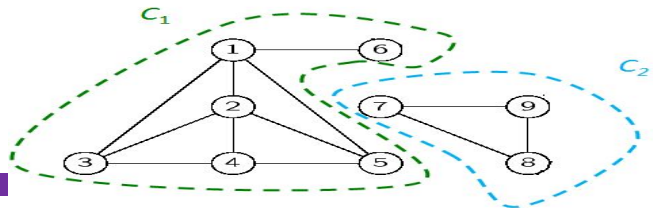


Graphes non connexes



Généralités
Connexité

- Les composantes connexes d'un graphe G = (S,A) forment une partition de S.
- Un graphe est connexe si et seulement s'il a une seule composante connexe
- Le sous-graphe induit par une composante connexe C est connexe.
- La composante connexe C qui contient un sommet i∈S est C = { j }



Graphe ayant deux composantes connexes

Généralités
Connexité

- Un graphe G = (S,A) est **fortement connexe** si : ∀ i, j ∈ S, il existe un **chemin** entre i et j.

Remarque : La définition implique que si i et j sont deux sommets d'un graphe fortement connexe, alors il existe un chemin de i à j et un chemin de j à i.

- Propriétés:
  - *Théorème 1* : Un graphe orienté fortement connexe est connexe.
  - *Théorème 2 :* Un graphe est fortement connexe si et seulement si ∀ i, j ∈ S, il existe un circuit passant par i et .

Généralités
Connexité

- Une composante fortement connexe Cf d'un graphe G = (S,A) est un sous-ensemble maximal de sommets tels que deux quelconques d'entre eux soient reliés par un **chemin**: si i∈ Cf, alors :
  - ∀ j ∈ Cf, il existe un circuit passant par i et j,
  - ∀ k ∈ S\Cf, il n'existe pas de circuit passant par i et k
- Les composantes fortement connexes d'un graphe G = (S,A) forment une partition de S.
- Un graphe est fortement connexe si et seulement s'il a une seule composante fortement connexe.
- Le sous-graphe induit par une composante fortement connexe Cf est fortement connexe.
- La composante fortement connexe Cf qui contient un sommet i ∈S est Cf = {j ∈S | il existe un chemin reliant i à j et un chemin reliant j à i }

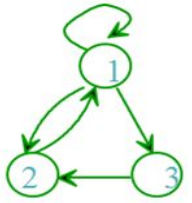
Généralités
Représentation des graphes

Matrice d'adjacence

Soit le graphe $G = (S,A)$. On suppose que les sommets de S sont numérotés de 1 à n, avec $n = |S|$. La représentation par **matrice d'adjacence** de G consiste en une matrice M de taille $n \times n$ telle que :

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

$S = \{1, 2, 3\}$
$A = \{(1, 1), (1, 2), (1, 3), (2, 1), (3, 2)\}$



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

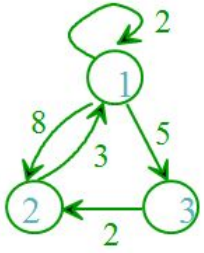
Place mémoire n^2

Généralités
Représentation des graphes

Matrice d'adjacence

Si le graphe est valué (par exemple, si des distances sont associées aux arcs), on peut utiliser une matrice d'entiers, de telle sorte que m_{ij} soit égal à la valuation de l'arc $(i\,;j)$ si $(i\,;j) \in A$. S'il n'existe pas d'arc entre 2 sommets i et j, on peut placer une valeur particulière.

Exemple :



$$M = \begin{bmatrix} 2 & 8 & 5 \\ 3 & \infty & \infty \\ \infty & 2 & \infty \end{bmatrix}$$

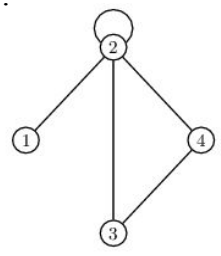
Généralités
Représentation des graphes

Matrice d'adjacence

Dans le cas de graphes non orientés, la matrice est symétrique par rapport à sa diagonale descendante. Dans ce cas, on peut ne mémoriser que la composante triangulaire supérieure de la matrice d'adjacence.

$$m_{ij} = \begin{cases} 1 \text{ si i et j sont adjacents} \\ 0 \text{ sinon} \end{cases}$$

Exemple :



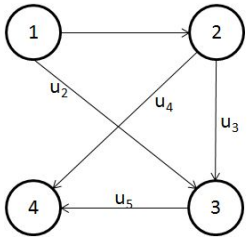
$$M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Matrice d'incidence

Soit le graphe $G = (S ; A)$. On suppose que les sommets de S sont numérotés de 1 à n, avec $n = |S|$ et $A = a_j$, $1 \le j \le m$. La représentation par matrice d'incidence de G consiste en une matrice M de taille $n \times m$ t.q :

$$m_{ij} = \begin{cases} 1 \text{ si i est l'origine de a} \\ 0 \text{ sinon} \\ -1 \text{ si i est la destination de a} \end{cases} \quad j$$

Exemple :



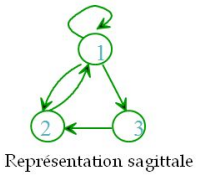
$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

Place mémoire n * m

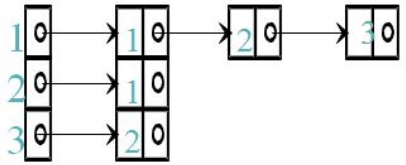
Listes de successeurs

Il s'agit d'un tableau $\Gamma^+(S)$ de n liste chaînées le tableau est indicé par les sommets ;la liste $\Gamma^+(i)$ contient tout les successeurs de i .

Exemple :



Représentation sagittale



Les informations supplémentaires peuvent être stockées dans les structures.

Qu'est-ce qu'un parcours de graphe (orienté ou non) ?
Visite de tous les sommets accessibles depuis un sommet de départ donné.

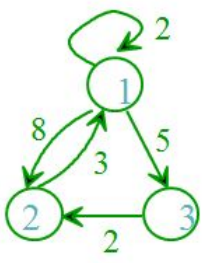
Pourquoi parcourir un graphe ?
Beaucoup de problèmes sur les graphes nécessitent que l'on parcourt l'ensemble des sommets et des arcs ou des arêtes du graphe.
Deux principales stratégies d'exploration :

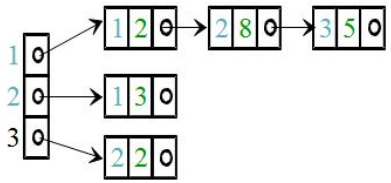
- Le parcours en largeur consiste à explorer les sommets du graphe niveau par niveau, à partir d'un sommet donné.
- Le parcours en profondeur consiste, à partir d'un sommet donné, à suivre un chemin le plus loin possible puis à faire des retours en arrière pour reprendre tous les chemins ignorés précédemment.

Arborescence couvrante associée à un parcours

- On parcourt un graphe à partir d'un sommet donné s_0 .
  L'arborescence de découverte des sommets accessibles depuis s_0 est appelée **arborescence couvrante de s_0**.
- Cette arborescence contient un arc (i , j) si et seulement si le sommet j a été découvert à partir du sommet i .
- C'est un graphe où chaque sommet a au plus un prédécesseur à partir duquel il a été découvert. s_0 est la racine de cette arborescence

Listes de successeurs





La taille requise est le nombre d'arcs. Les degrés extérieurs sont visibles, mais pas les degrés intérieurs. Il peut être utile parfois de déduire la représentation par liste des prédécesseurs de celle-ci.

Les notions de pile et de file sont deux structures de données abstraites importantes en informatique. On se limite à la présentation de ces notions aux besoins des parcours de graphes.

- **PILE** (stack)

La structure de pile est celle d'une pile d'assiettes : Pour ranger les assiettes, on les empile les unes sur les autres.
Lorsqu'on veut utiliser une assiette, c'est l'assiette qui a été empilée en dernier qui est utilisée ⇒ Structure **LIFO** (last in, first out)

- **FILE** (queue)

La structure de file est celle d'une file d'attente a un guichet : Les nouvelles personnes qui arrivent se rangent à la fin de la file d'attente. La personne servie est celle qui est arrivée en premier dans la file ⇒ Structure **FIFO** (first in, first out).

Parcours en largeur (Breadth First Search : BFS)

Structures de données utilisées :On utilise une file F , pour laquelle on suppose définies les opérations :

init_file(**F**) qui initialise la file F à vide,

ajoute_fin_file(**F**, **s**), qui ajoute le sommet s à la fin de la file F ,

est_vide(**F**)qui retourne vrai si la file F est vide et faux sinon,

enleve_debut_file(**F**, **s**) qui enlève le sommet s au début de la file F .

Parcours en largeur (Breadth First Search : BFS)

- On utilise un tableau π qui associe à chaque sommet le sommet qui l'a fait entrer dans la file (et un tableau couleur qui associe à chaque sommet sa couleur : blanc, gris ou noir).

- On utilise de plus un tableau **d** qui associe à chaque sommet son niveau de profondeur par rapport au sommet de départ s_0 ($d[s]$ est la longueur du chemin dans l'arborescence π de la racine s_0 jusqu'à s).

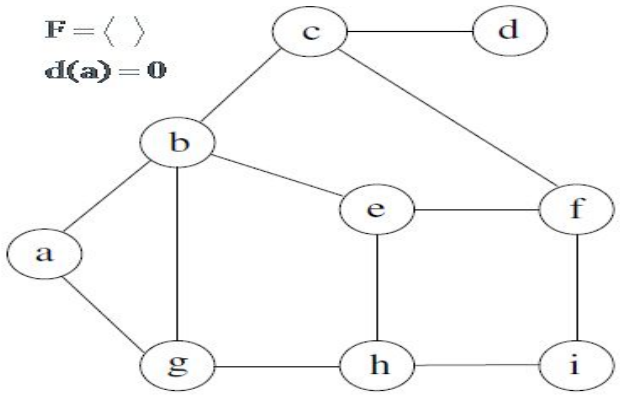
Fonction *BFS*(g, s_0)

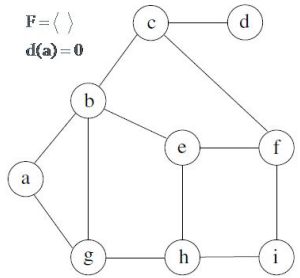
Soit f une file (FIFO) initialisée à vide
pour *chaque sommet* s_i *de* g **faire**
  $\pi[s_i] \leftarrow null$
  Colorier s_i en blanc
  $d[s_i] \leftarrow \infty$
Ajouter s_0 dans f et colorier s_0 en gris
$d[s_0] \leftarrow 0$
tant que f *n'est pas vide* **faire**
  Soit s_k le sommet le plus ancien dans f
  tant que $\exists s_i \in succ(s_k)$ *tq* s_i *soit blanc* **faire**
    Ajouter s_i dans f et colorier s_i en gris
    $d[s_i] \leftarrow d[s_k] + 1$
    $\pi[s_i] \leftarrow s_k$
  Enlever s_k de f et colorier s_k en noir
retourne d **retourne** π

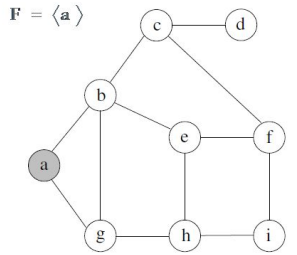
Parcours en largeur (Breadth First Search : BFS)

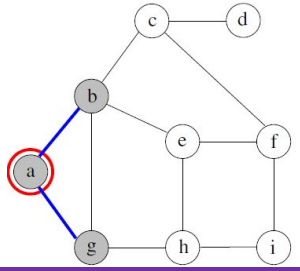
L'exécution de l'algorithme sera présentée sous forme d'un tableau :

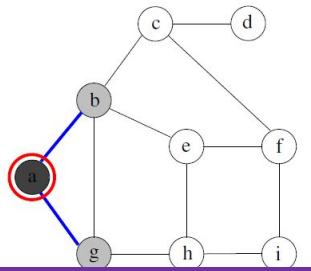
sommets : s_i	s_0	s_1	. . .	s_n
$\pi(s_i)$	nill	-	-	-
$d(s_i)$	-	-	-	-
couleur(s_i)	-	-	-	-

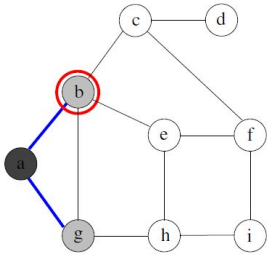


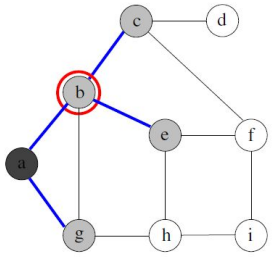


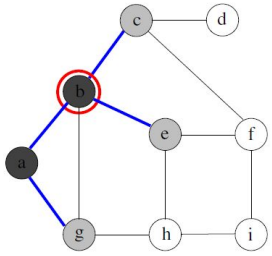


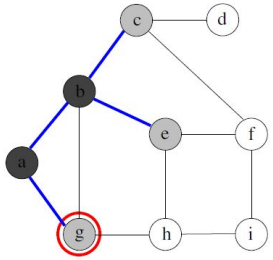


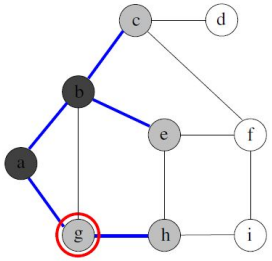


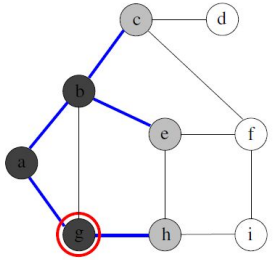

c d
b
e f
a
g h i

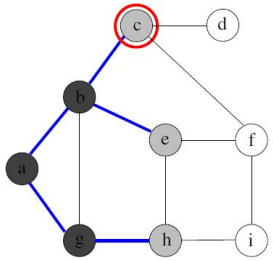

c d
b
e f
a
g h i

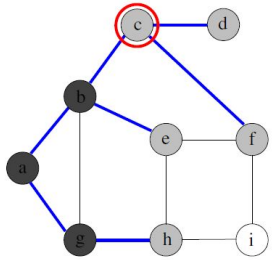

c d
b
e f
a
g h i

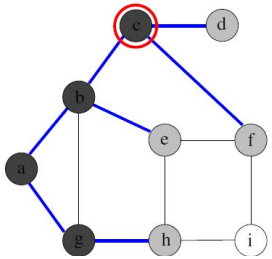

c d
b
e f
a
g h i

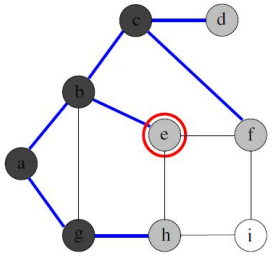

c d
b
e f
a
g h i

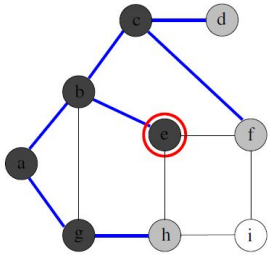

c d
b
e f
a
g h i

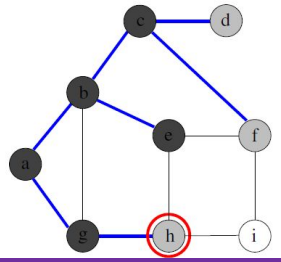

c d
b
e f
a
g h i

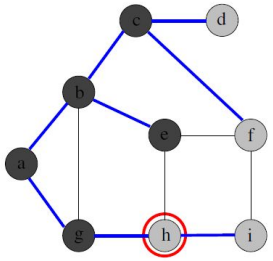

c d
b
e f
a
g h i

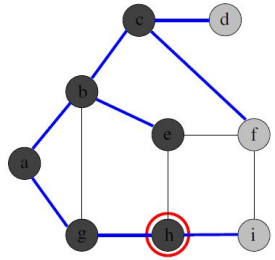

c d
b
e f
a
g h i

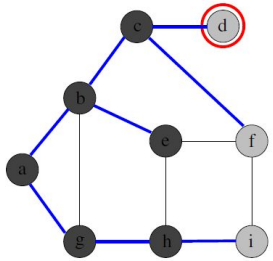

c d
b
e f
a
g h i

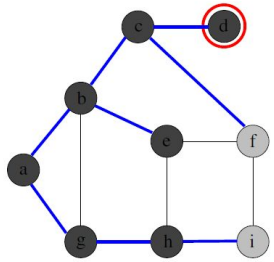

c d
b
e f
a
g h i

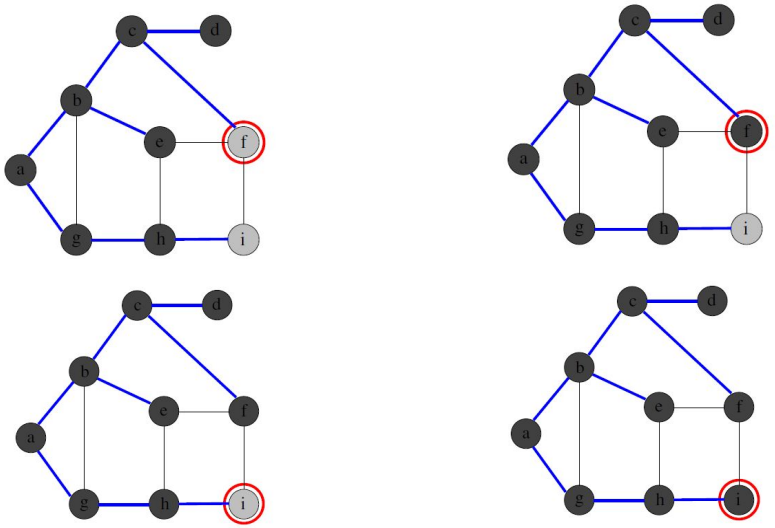

c d
b
e f
a
g h i

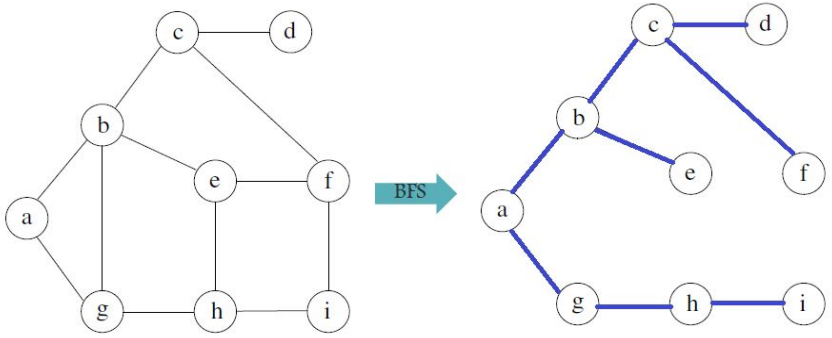

c d
b
e f
a
g h i


c d
b
e f
a
g h i


c d
b
e f
a
g h i


c d
b
e f
a
g h i





Complexité de BFS pour un graphe ayant *n* sommets et *m* arcs ?
$O(n + m)$ (sous réserve d'une implémentation par listes d'adjacence)

Applications du parcours en largeur

Recherche de l'ensemble des sommets accessibles à partir de s_0 : après l'exécution de *BFS* (S ; A; s_0) les sommets noirs sont ceux accessibles depuis s_0 ; les sommets blancs sont ceux qui ne sont pas des descendants de s_0.

$\Rightarrow$ Le parcours en largeur permet de déterminer les composantes connexes d'un graphe non orienté : On appliquer *BFS* à partir d'un sommet s quelconque : tous les sommets en noirs $\in$ à C (s)

S'il reste des sommets blancs $\Rightarrow$ il y a d'autres composantes connexes. Il faut alors relancer *BFS* sur le sous-graphe induit par les sommets blancs, pour découvrir une autre composante connexe. . .

Le nombre de fois où l'algorithme de parcours en largeur a été lancé correspond au nombre de composantes connexes.

Parcours en Profondeur (Depth First Search : DFS)

Structures de données utilisées : On utilise une pile P, pour laquelle on suppose définies les opérations :

- **init_pile(P)** qui initialise la pile P à vide,
- **empile(P, s)** qui ajoute s au sommet de la pile *P* ,
- **est_vide(P)** qui retourne vrai si la pile P est vide et faux sinon,
- **sommet(P)** qui retourne le sommet s au sommet de la pile *P,*
- **depile(P, s)** qui enlève s du sommet de la pile *P* .

On utilise, comme pour le parcours en largeur, un tableau qui associe à chaque sommet le sommet qui l'a fait entrer dans la pile, et un tableau couleur qui associe à chaque sommet sa couleur (blanc, gris ou noir).

Parcours en Profondeur (Depth First Search : DFS)

On va en plus mémoriser pour chaque sommet si :

dec[s_i] = date de découverte de s_i (passage en gris)

fin[s_i] = date de fin de traitement de s_i (passage en noir) où l'unité de temps est une itération. La date courante est mémorisée dans la variable *tps* .

L'algorithme DFS :

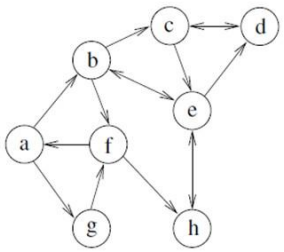
Fonction *DFS*(g, s_0)
  Soit p une pile (LIFO) initialisée à vide
  pour *tout sommet* $s_i \in S$ **faire**
    $\pi[s_i] \leftarrow null$
    Colorier s_i en blanc
  Empiler s_0 dans p et colorier s_0 en gris
  tant que p *n'est pas vide* **faire**
    Soit s_i le dernier sommet entré dans p
    si $\exists s_j \in succ(s_i)$ *tel que* s_j *soit blanc* **alors**
      Empiler s_j dans p et colorier s_j en gris
      $\pi[s_j] \leftarrow s_i$
    sinon
      Dépiler s_i de p et colorier s_i en noir
  retourne π

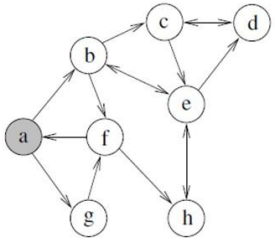
L'algorithme DFS :

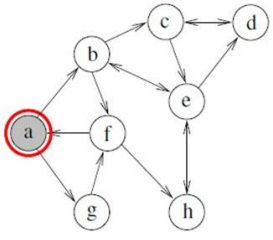
L'exécution de l'algorithme sera présentée sous forme d'un tableau :

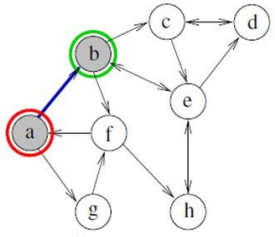
sommets : s_i	s_0	s_1	. . .	s_n
$\pi(s_i)$	nill	-	-	-
$dec(s_i)$	-	-	-	-
couleur(s_i)	-	-	-	-
$fin(s_i)$	-	-	-	-

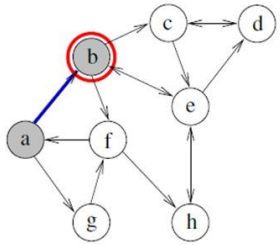
Exemple :

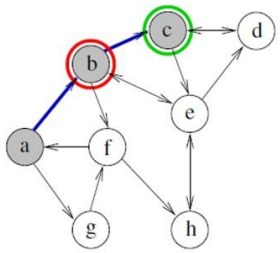


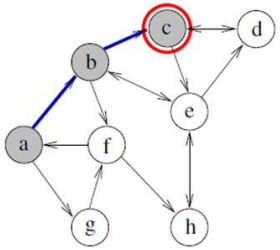


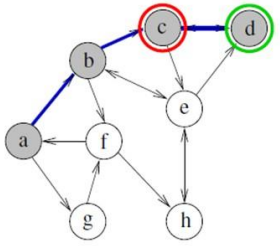


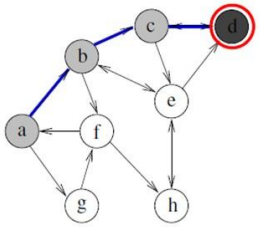


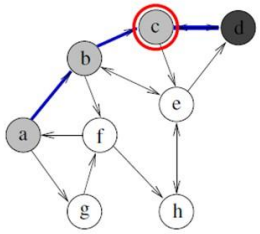

a
b
c
d
e
f
g
h

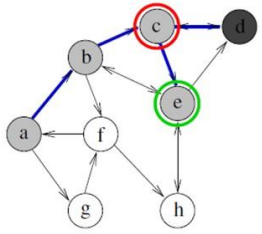

a
b
c
d
e
f
g
h

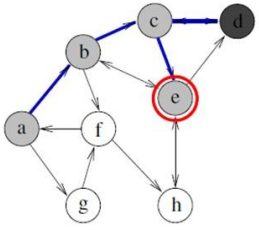

a
b
c
d
e
f
g
h

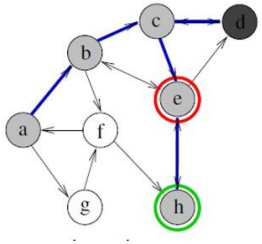

a
b
c
d
e
f
g
h

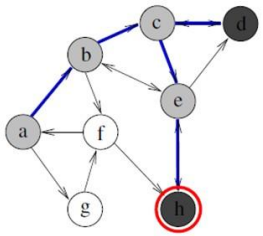

a
b
c
d
e
f
g
h

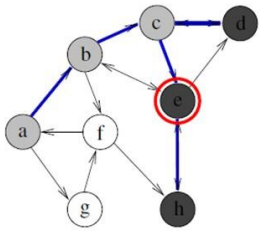

a
b
c
d
e
f
g
h

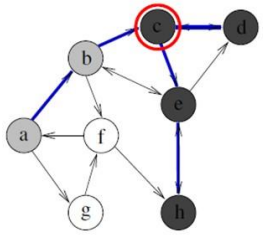

a
b
c
d
e
f
g
h

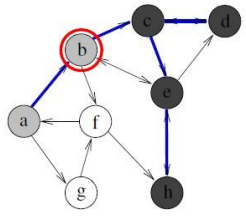

a
b
c
d
e
f
g
h

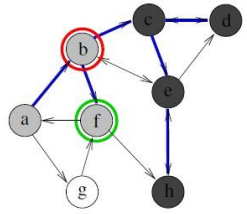

a
b
c
d
e
f
g
h

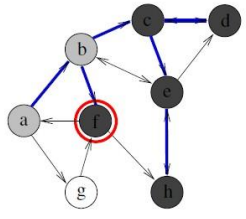

a
b
c
d
e
f
g
h

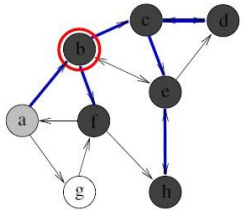

a
b
c
d
e
f
g
h

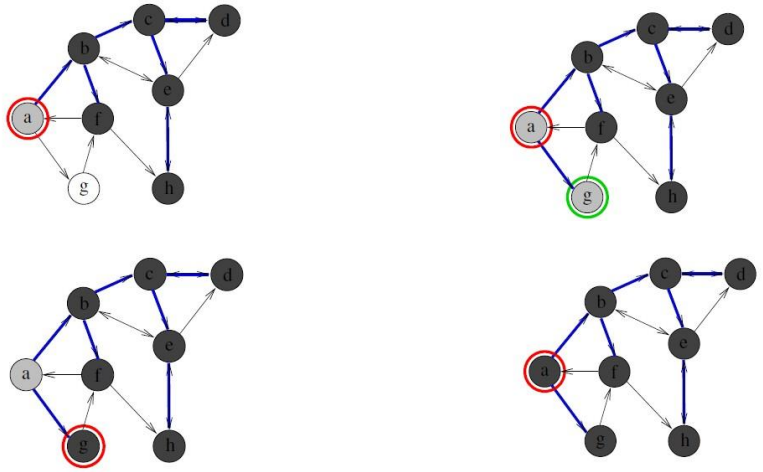

a
b
c
d
e
f
g
h

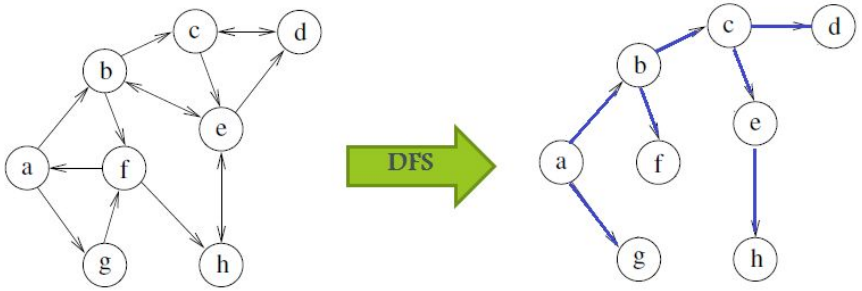

a
b
c
d
e
f
g
h

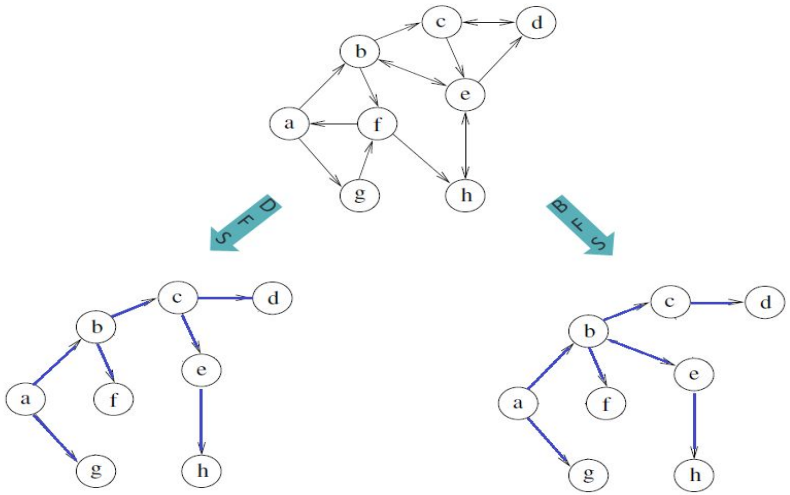

a
b
c
d
e
f
g
h


a
b
c
d
e
f
g
h


a
b
c
d
e
f
g
h







Applications du parcours en profondeur

1- Recherche de circuits

Lors du parcours en profondeur d'un graphe non orienté (resp. orienté), si un successeur *j* du sommet "courant" *i* en haut de la pile est déja gris, cela implique qu'il existe une chaîne (resp. un chemin) permettant d'aller de *j* vers *i* , et donc qu'il existe un cycle (resp. un circuit).

Procédure *DFS*(g, s_0)
- **Entrée** : Un graphe g et un sommet s_0
- **Précondition** : s_0 est blanc
- **début**
  - Colorier s_0 en gris
  - **pour** *tout* $s_j \in succ(s_0)$ **faire**
    - **si** s_j *est gris* **alors** afficher("circuit");
    - **sinon si** s_j *est blanc* **alors**
      - $\pi[s_j] \leftarrow s_0$
      - DFS (g, s_j)
  - Colorier s_0 en noir

Applications du parcours en profondeur

2-Recherche des composantes fortement connexes d'un graphe orienté

Principe de l'algorithme de Kosaraju-Sharir pour déterminer les composantes fortement connexes d'un graphe $G = (S ; A)$:

- Appliquer $DFS\ (G\ )$
- Trier les sommets par ordre de numéro de fin décroissant dans un tableau t
- Construire le graphe $G^t = (S ; A^t)$ tel que $A^t = \{(i ; j\ )/(j ; i\ ) \in A\}$
- Appliquer $DFS\ (G^t)$ en démarrant du 1^e sommet du tableau t et itérer le processus à partir du blanc suivant dans le tableau t