Parcours de Graphes.

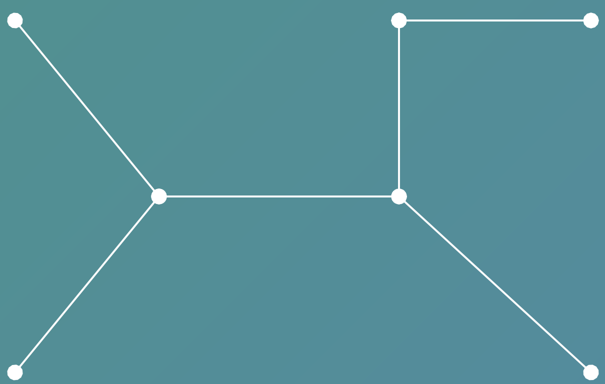
Jérémy Rouot

e-mail: jeremy.rouot@yncrea.fr

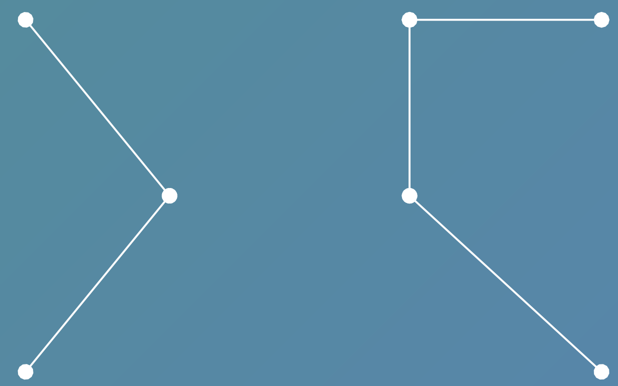
bureau 332

Exploration : procédé déterministe qui permet de fixer un ordre dans l'examen des sommets de telle façon que l'ensemble des sommets examinés engendre un graphe connexe.

L'exploration d'un graphe à partir d'un sommet de départ (appelé racine de l'exploration) permet de trouver la composante connexe à laquelle appartient ce sommet et aussi de structurer les sommets d'un graphe connexe.



Graphe *Connexe*
(1 composante connexe)



Graphe *non Connexe*
(2 composantes connexes)

Plusieurs types d'explorations :

Exploration en Largeur

et

Exploration en Profondeur

Exploration en Largeur
BFS : Breadth–First Search

Principe : Partition en couches des sommets d'un graphe connexe. On définit d'abord la distance d(x,y) entre deux sommets x et y, comme la longueur d'une plus courte chaîne entre ces deux sommets. On fixe un sommet initial r, appelé racine de l'exploration.
La séquence d'ensembles:

$$C_0=\{r\}, C_1 = \{x \in X, d(r,x) = 1\}, \ldots, C_k = \{x \in X, d(r,x) = k\}$$

est appelée la partition en couches associée à r.

On visite les sommets en construisant, une par une, les couches. On marque la racine et ensuite les sommets de la première couche, la deuxième etc.

Un sommet marqué est dit exploré quand on aura fini de marquer ses voisins. On finit de marquer une couche avant de passer à la couche suivante. On applique la règle: premier marqué – premier exploré (règle **FIFO** : First In – First Out). Pour gérer ainsi les sommets on peut utiliser un tableau FILE et deux curseurs : tête et queue.

Procédure LARGEUR (*G*: graphe; racine: entier; FILE: **tableau** [1…*n*] **de** entier);

```
pour i de 1 à n faire:
   MARQUE[i]:= faux;
fin pour
tête:=1; queue:= 1; FILE[tête]:=racine; MARQUE[r]:= vrai;
```

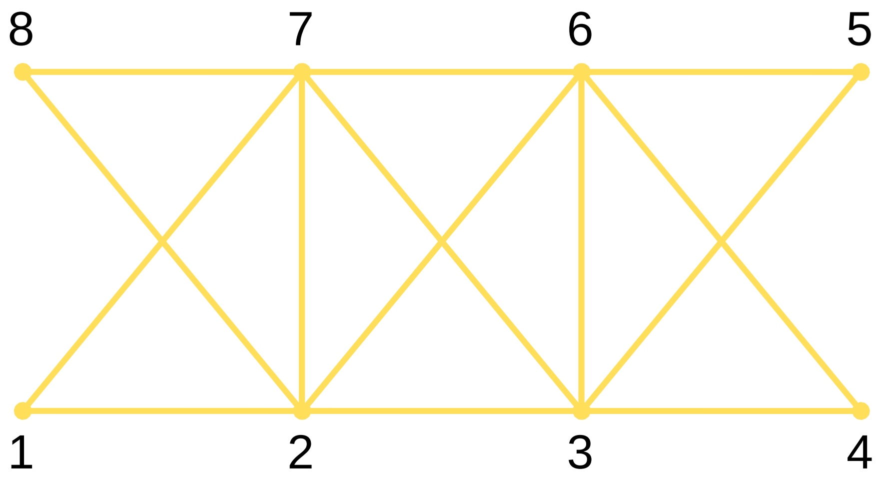
↑
initialisation

```
tantque tête ≤ queue  faire:
x:= FILE[tête];
Liste:= liste de voisins de x dans G;

     tantque Liste  ≠ nil  faire:
      y:= Liste↑.nom;

      si non MARQUE[y] alors
      MARQUE[y]:= vrai;
         queue:= queue +1;
         FILE[queue]:= y;
         fin si

     Liste:= Liste↑.suivant;
     fin tantque

     tête:= tête +1;
 fin tantque
fin LARGEUR
```

← *boucle principale*

Exemple

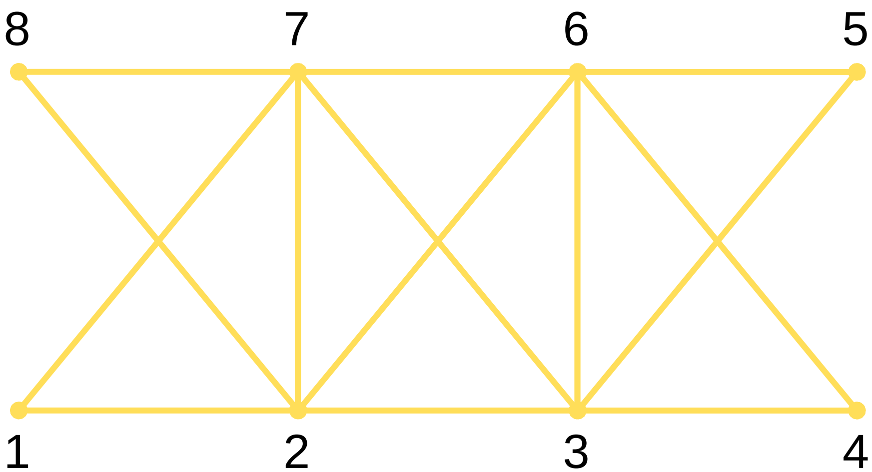
Le graphe



1: 2 → 7 → nil
2: 6 → 8 → 1 → 3 → 7 → nil
3: 4 → 5 → 2 → 6 → 7 → nil
4: 3 → 6 → nil
5: 3 → 6 → nil
6: 7 → 3 → 4 → 5 → 2 → nil
7: 3 → 6 → >8 → 1 → 2 → nil
8: 2 → 7 → nil

Exemple

Le graphe



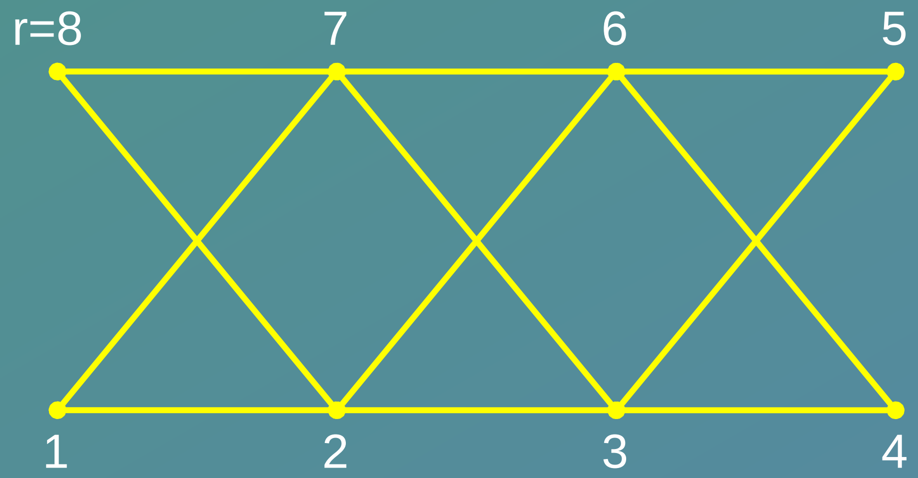
1: 2 → 7 → nil
2: 6 → 8 → 1 → 3 → 7 → nil
3: 4 → 5 → 2 → 6 → 7 → nil
4: 3 → 6 → nil
5: 3 → 6 → nil
6: 7 → 3 → 4 → 5 → 2 → nil
7: 3 → 6 → >8 → 1 → 2 → nil
8: 2 → 7 → nil

peut être représenté par une table de successeurs et les tableaux suivants

	1	2	3	4	5	6	7	8
P	1	3	8	13	15	17	22	26

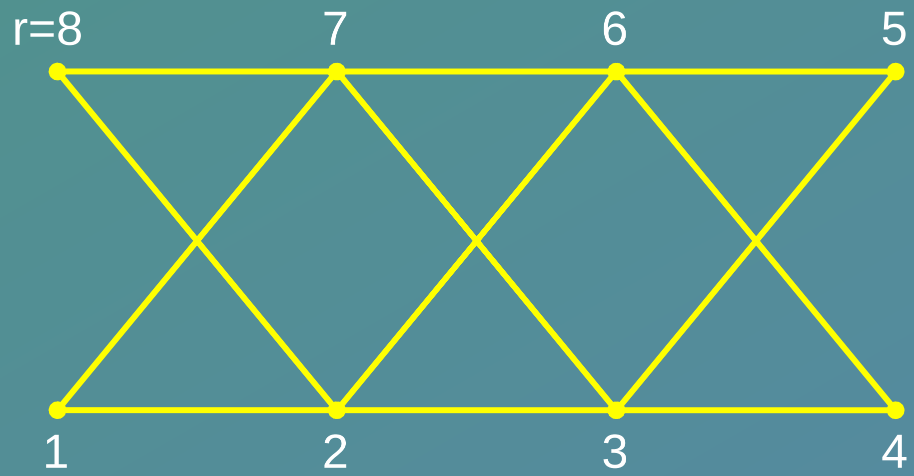
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
N	2	7	6	8	1	3	7	4	5	2	6	7	3	6	3	6	7	3	4	5	2	3	6	8	1	2	2	7

Graphe initial



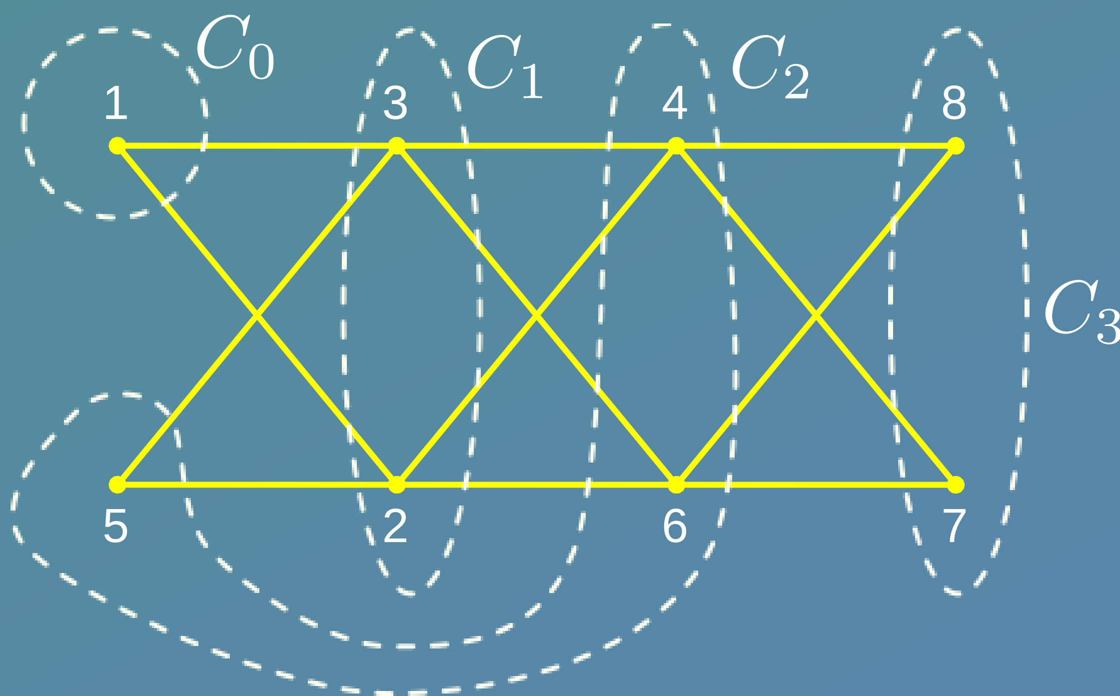
$$C_0 = \{r\}, C_1 = \{x \in X, d(r,x) = 1\}, \ldots, C_k = \{x \in X, d(r,x) = k\}$$

Graphe initial



$$C_0 = \{r\}, C_1 = \{x \in X, d(r,x) = 1\}, \ldots, C_k = \{x \in X, d(r,x) = k\}$$

Ordre parcours largeur à partir de la source 8



Exploration en Profondeur
DFS : Depth–First Search

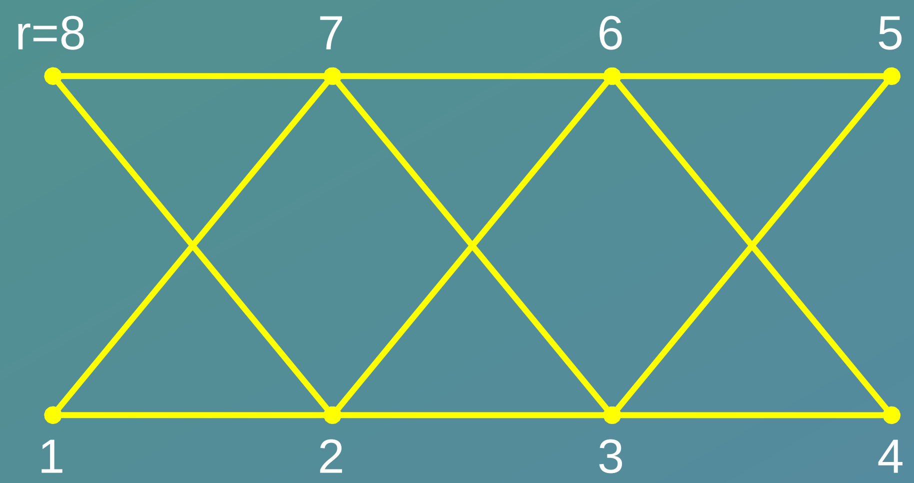
Principe : Prolongement d'une chaîne élémentaire d'un graphe connexe. On visite les sommets en partant de la racine r de l'exploration. Lorsqu'on arrive à un sommet x on ne visite pas tous ses voisins, mais un de ses voisins non encore atteints: on cherche à prolonger la chaîne de r à x. Si tous les voisins de x ont été visités, alors on l'abandonne pour examiner le sommet y qui le précède dans la chaîne de r à x et on cherche à prolonger la chaîne de r à y.

On applique la règle: dernier marqué -- premier exploré (règle **LIFO**: Last In – First Out). Pour gérer ainsi les sommets on peut utiliser un tableau PILE et un indice hauteur.

Remarque : L'ordre dépend de la représentation du graphe.

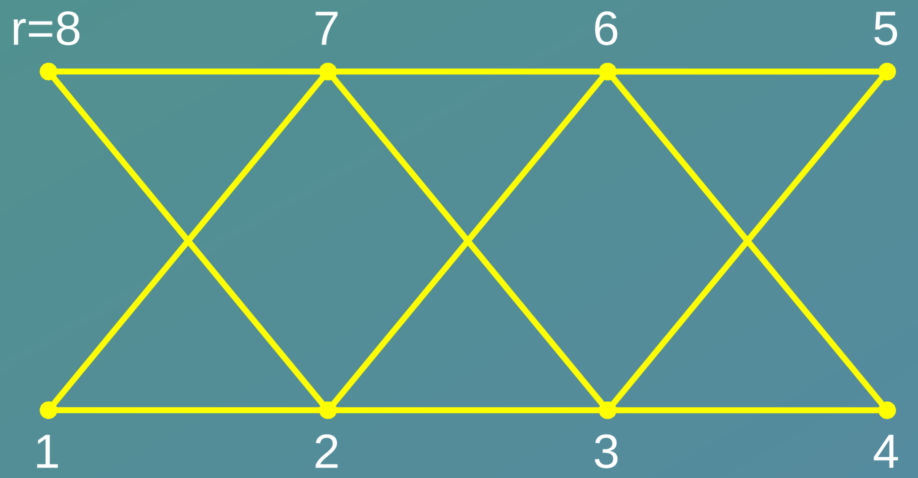
```
 Procédure PROFONDEUR
(G: graphe; racine: entier; PILE: tableau [1…n] de entier);

pour i de 1 à n faire:
      MARQUE[i]:= faux;                        ←  initialisation
fin pour
h:=1; PILE[h]:=racine; MARQUE[r]:= vrai;

tantque h ≥ 1  faire:
   x:= PILE[h];
      si  P[y]≠ nil faire:
      y:=P[N[x]];

         si non MARQUE[y] alors
         MARQUE[y]:= vrai;
         h:= h+1;                              ← boucle principale
         PILE[h]:= y;
         fin si

      Mettre à jour P[x]
     (P[x] augmente d'1 ou devient 0);
         sinon
         h:= h-1;
      fin si
fin tantque
fin PROFONDEUR
```

Graphe initial



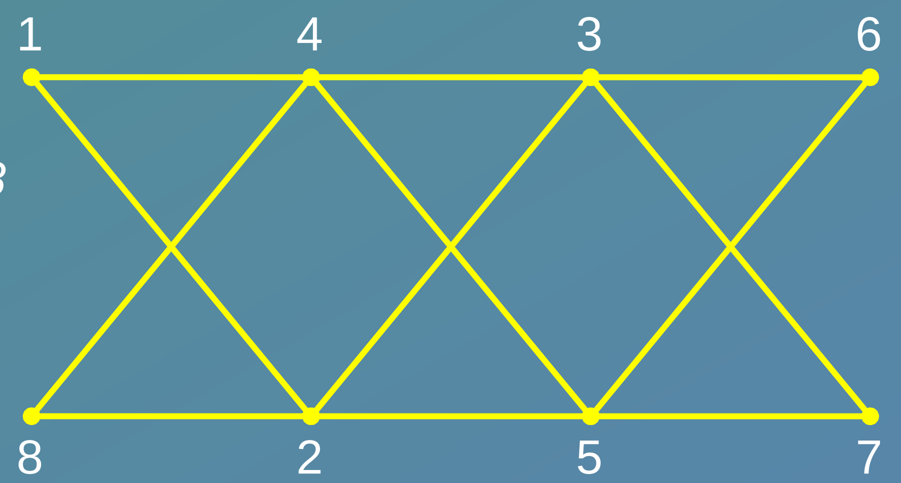
1: 2 → 7 → nil
2: 6 → 8 → 1 → 3 → 7 → nil
3: 4 → 5 → 2 → 6 → 7 → nil
4: 3 → 6 → nil
5: 3 → 6 → nil
6: 7 → 3 → 4 → 5 → 2 → nil
7: 3 → 6 → >8 → 1 → 2 → nil
8: 2 → 7 → nil

Graphe initial



1: 2→ 7→nil
2: 6→8→1→3→7→nil
3: 4→5→2→6→7→nil
4: 3→6→nil
5: 3→6→nil
6: 7→3→4→5→2→nil
7: 3→6→>8→1→2→nil
8: 2→7→nil

Ordre parcours profondeur à partir de la source 8



Complexité pour les deux parcours

Espace :

- Représentation du graphe : O(n+m)
- Structure PILE : O(n)
- Tableau MARQUE : O(n)

=> Complexité en O(n+m)

Temps :

- marquage : n opérations
- exploration : chaque sommet x nécessite d(x) opération donc en tout

$$\sum_{x \in X} d(x) = 2|E| = 2m$$

=> Complexité en O(n+m)