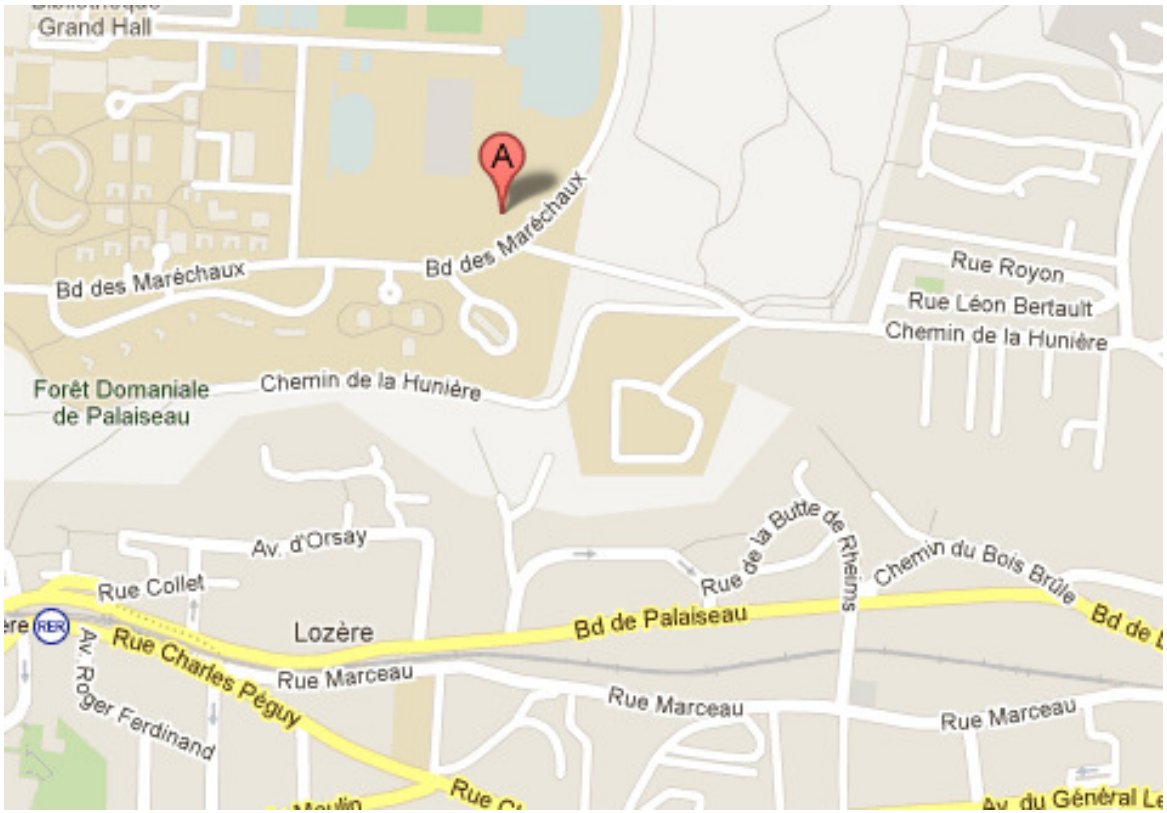
IN103
Résolution de problèmes algorithmiques
une approche fondée sur les structures de données

Alexandre Chapoutot

Année académique 2023-2024

Plan du cours

Motivation – assistance au déplacement



Motivation – assistance au déplacement

Représentation discrète de l'espace

- Intersections : **sommets**
- Routes : **arcs** entre les sommets avec potentiellement des propriétés
  - une **orientation** (sens unique, double sens, ...)
  - une **valuation** ou **poids** (vitesse max, distance, charge, ...)
- Carte (topologique) : **graphe orienté pondéré**
- But : calculer un chemin avec un coût minimal par analyse du graphe

Applications des graphes

- Réseaux de communication (routes, trains, métros, télécoms . . .).
- Planification de tâches (dépendance / antériorité).
- Compilation (flot de contrôle de programme).
- Physique (chaînes de Markov).
- Robotique (carte topologique)
- Automatique (automates de contrôle).
- Biologie (réassemblage de sections de génome).
- . . .

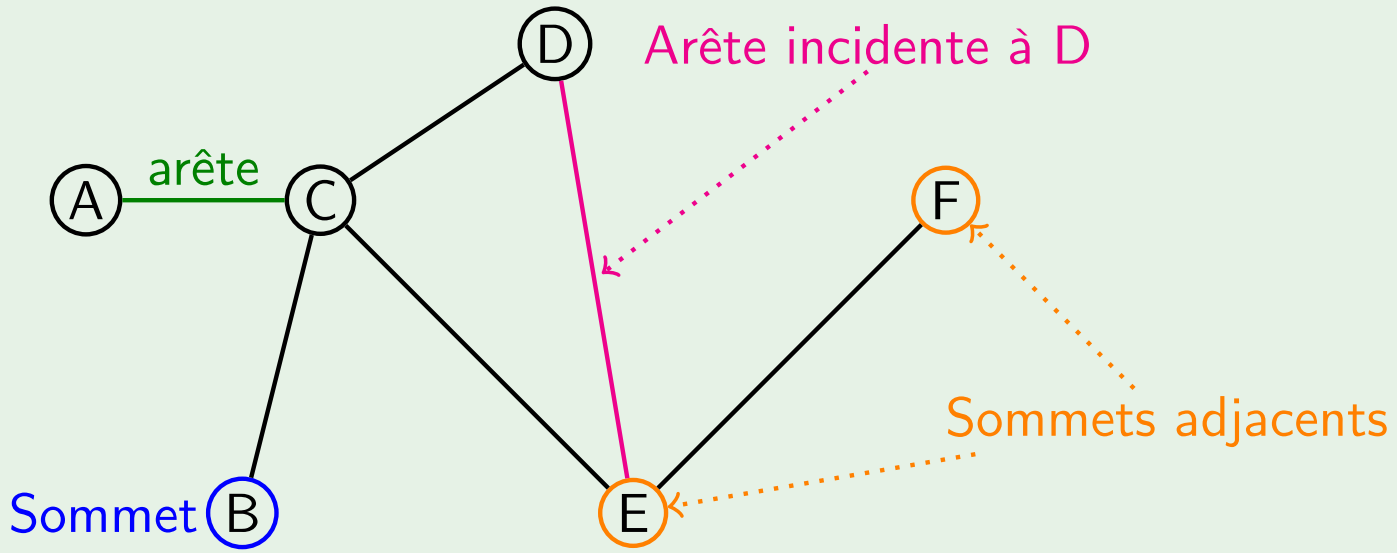
Graphes non orientés

Graphes non orientés

Un **graphe non orienté** (ou simplement **graphe**) est une paire (S, A)

- S ensemble fini de **sommets** (*vertex/vertices*)
- A ensemble fini d'**arêtes** (*edges*) sous ensemble de $S \times S$
- L'**ordre** d'un graphe est le nombre de sommets qui le compose.
- Les sommets a et b sont **adjacents** s'ils sont reliés par une arête.
- Un sommet s est **incident** à une arête a si s est une des deux extrémités de a, et une arête est incidente à un sommet si celui-ci est une des extrémités.

Exemple

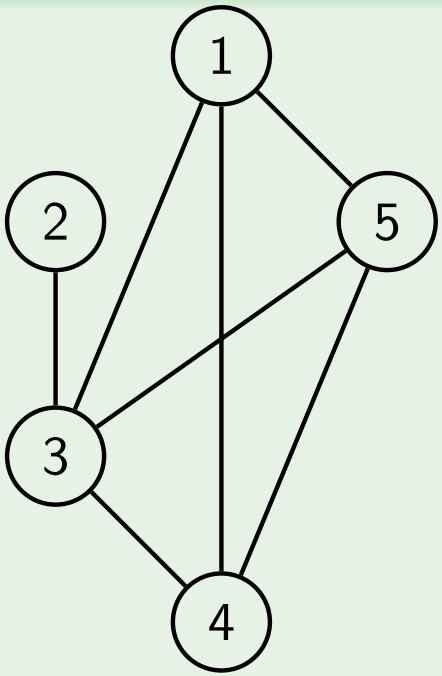


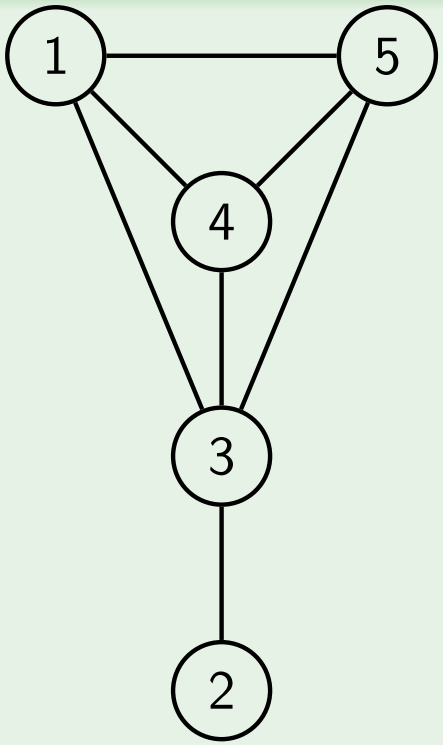
Graphes planaires

Définition

Un graphe est **planaire** s'il peut être dessiné dans un plan de telle manière que ses arêtes ne se coupent pas en dehors de leurs extrémités.

Exemple



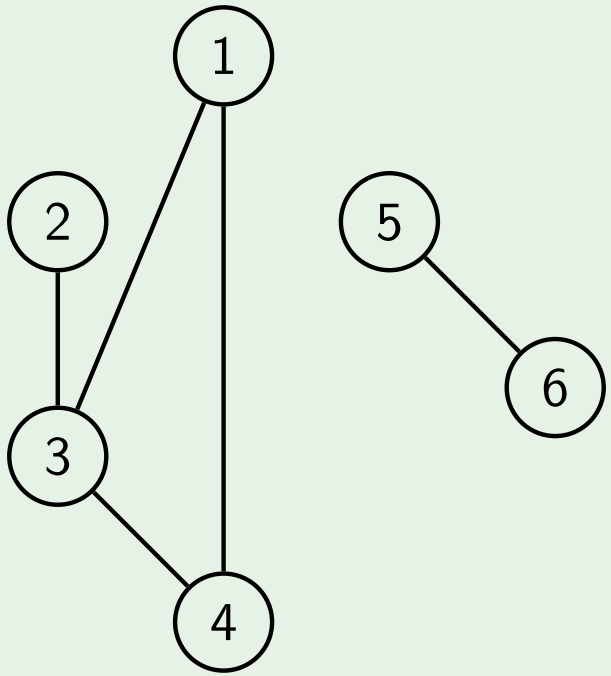


Graphes connexes

Définition

Un graphe est **connexe** si à partir de n'importe quel sommet il est possible de rejoindre les autres sommets en suivant les arêtes.

Exemple : graphe non connexe

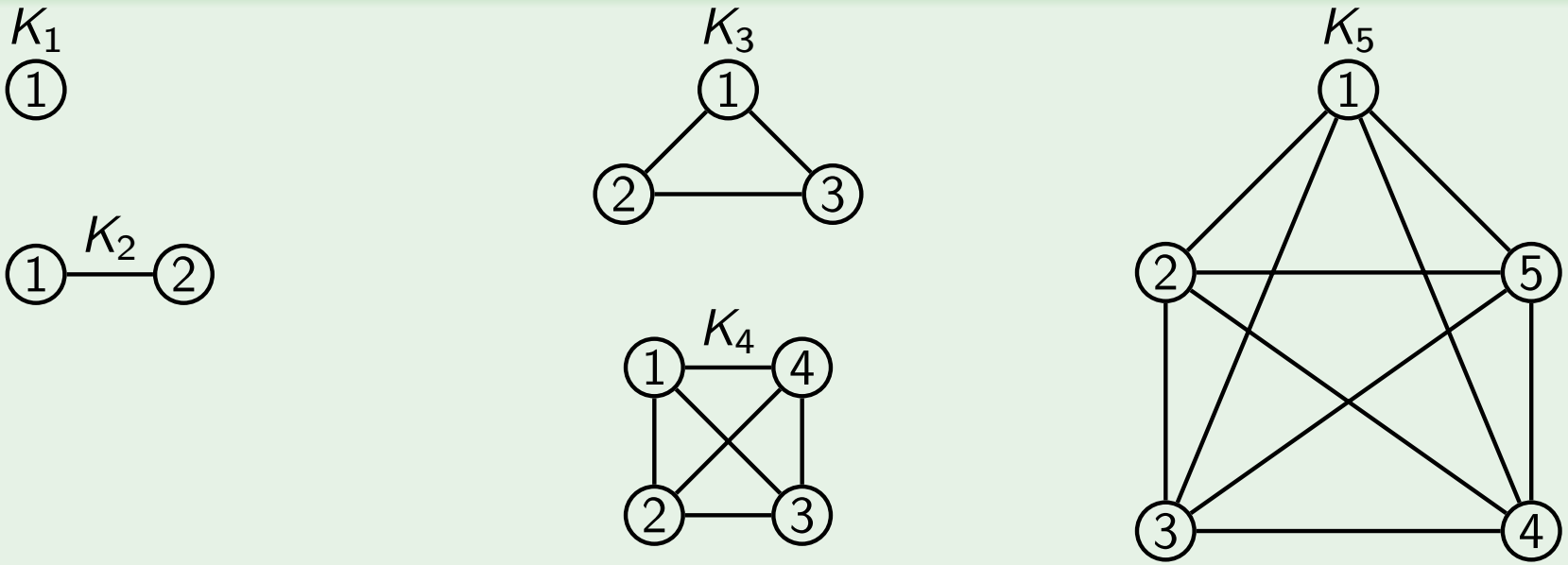


Graphes complets

Définition

Un graphe G est **complet** si chaque sommet de G est relié directement à tous les autres sommets.

Exemple, graphes complets de K_1 à K_5



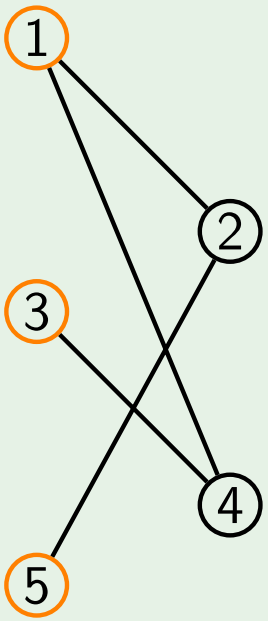
Remarque : K_5 est le plus petit graphe complet non planaire.

Graphes bipartis

Définition

Un graphe G est **biparti** si ses sommets S peuvent être divisés en deux sous-ensembles S_1 et S_2 telles que toutes les arêtes de G relient un sommet de S_1 à un sommet de S_2.

Exemple



- $S_1 = \{1, 3, 5\}$
- $S_2 = \{2, 4\}$

Graphes partiels et sous-graphes

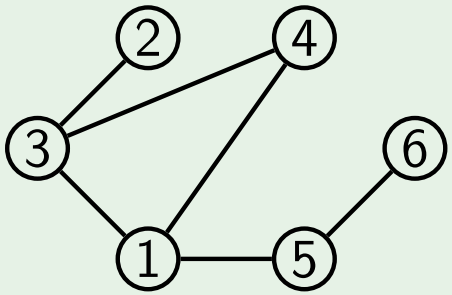
Définitions

Soit $G = (S, A)$ un graphe

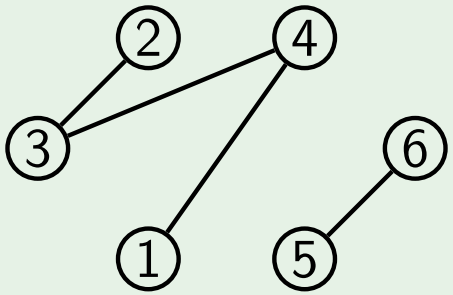
- $G' = (S, A')$ est **graphe partiel** de G si $A' \subseteq A$ (G' est obtenu en ôtant des arêtes de G)
- $G' = (S', A(S'))$ est un **sous-graphe** de G induit par $S' \subseteq S$ avec $A(S')$ l'ensemble d'arêtes de G ayant leurs extrémités dans S'.

Exemple

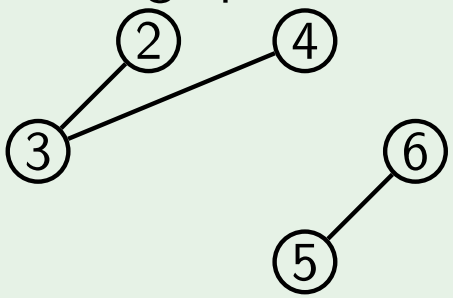
Graphe G



Graphe partiel de G



Sous graphe de G



Une clique

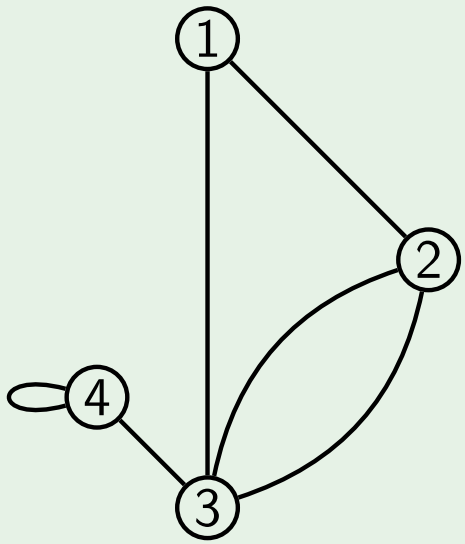
Un sous-graphe complet d'un graphe.

Graphes simples et multigraphes

Définition

Un graphe G est **simple** si au plus une arête lie deux sommets et il n'y a pas de boucle sur un sommet. Autrement G est un **multigraphe**.

Exemple de multigraphe



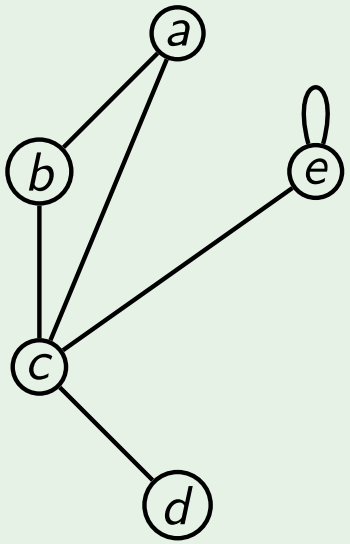
Hypothèse de travail nous nous concentrerons que sur les graphes simples dans cet enseignement.

Degrés

Définition degré d'un sommet

Le **degré** d'un sommet s, noté $d(s)$, est le nombre d'arêtes incidentes à ce sommet (une boucle compte pour 2).

Exemple



- $d(a) = 2$
- $d(b) = 2$
- $d(c) = 4$
- $d(d) = 1$
- $d(e) = 3$

Degré d'un graphe

Degré maximum de tous les sommets du graphe. Si tous les sommets ont un même degré k alors le graphe est k-**régulier**.

Chaînes

Chaîne

Une suite alternée de sommets et d'arcs, débutant et finissant par un sommet. La **longueur** d'une chaîne est le nombre d'arêtes la constituant.

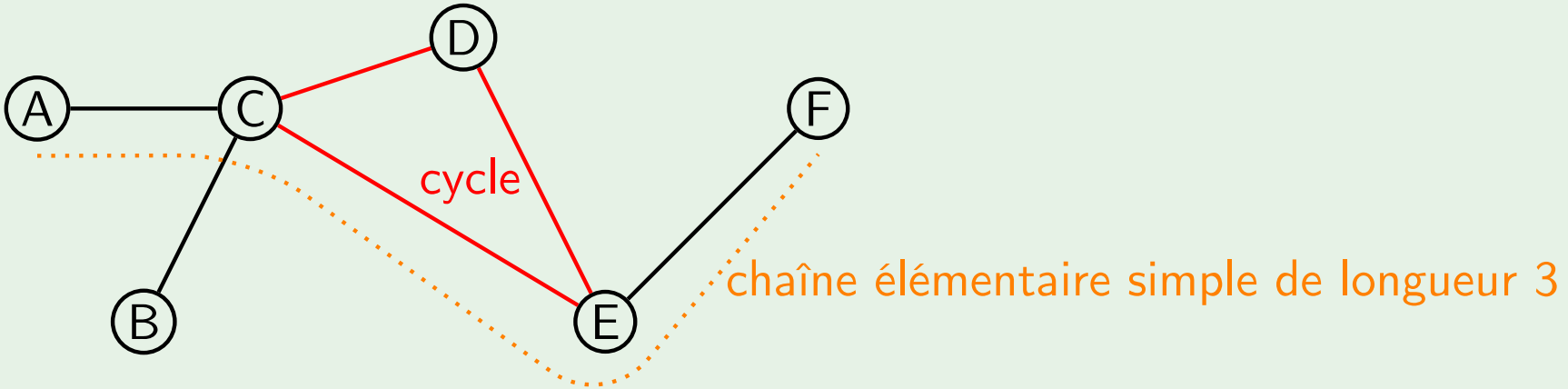
Une chaîne est

- **élémentaire** si chaque sommet n'y apparaît au plus une fois.
- **simple** si chaque arête n'y apparaît au plus une fois.
- **fermée** si le sommet de départ et d'arrivée est le même.

Un **cycle** est une chaîne simple fermée.

La **distance** entre deux sommets est la longueur de la plus petite chaîne les reliant.

Exemple



Graphes eulériens

Hypothèse : on considère des graphes simples connexes.

Définition : chaîne eulérienne

Une chaîne qui passe une et une seule fois par chacunes des arêtes du graphe.

Définition : cycle eulérien

Un cycle qui passe une et une seule fois par chaque arête du graphe.

Graphe eulérien

Un graphe qui possède un cycle eulérien.

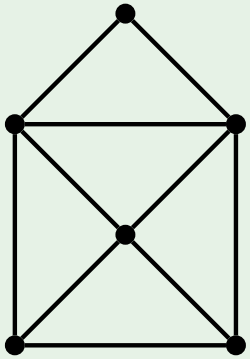
Graphe semi-eulérien

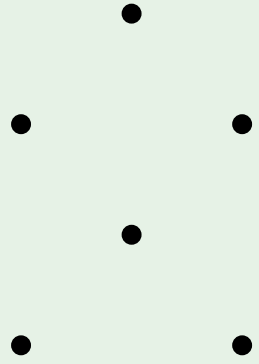
Un graphe qui possède que des chaînes eulériennes.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





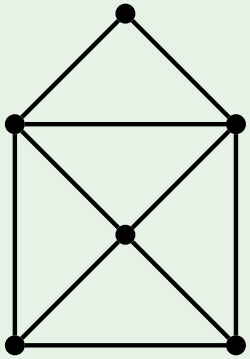
Théorèmes d'Euler

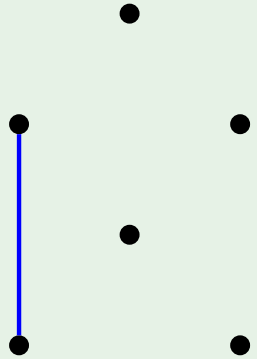
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





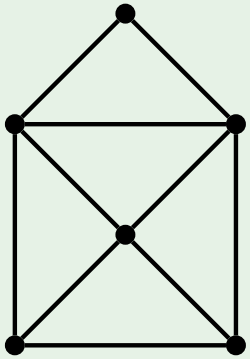
Théorèmes d'Euler

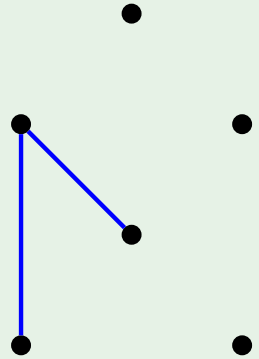
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





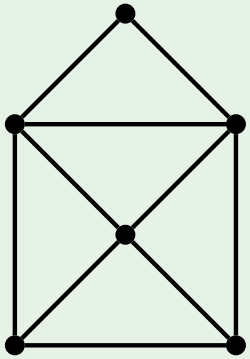
Théorèmes d'Euler

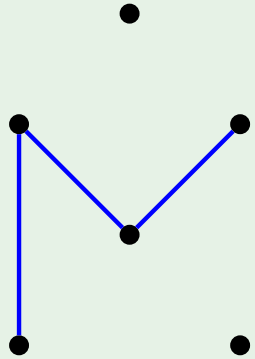
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





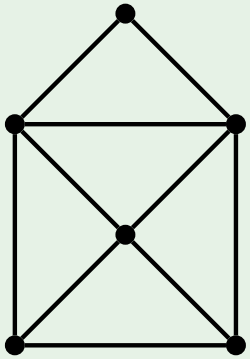
Théorèmes d'Euler

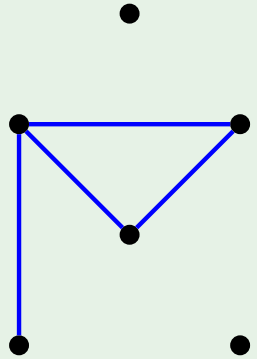
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





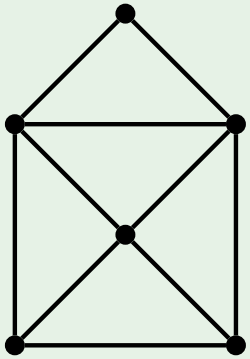
Théorèmes d'Euler

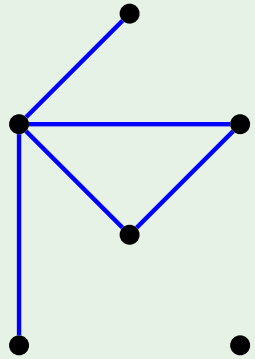
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





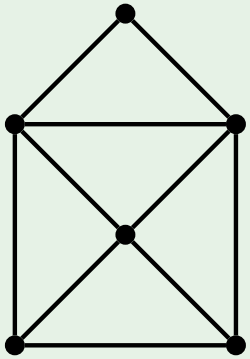
Théorèmes d'Euler

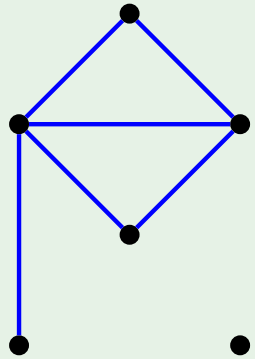
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





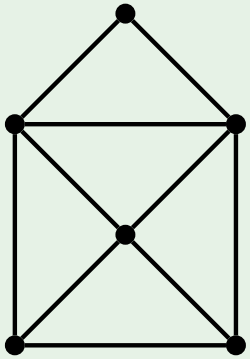
Théorèmes d'Euler

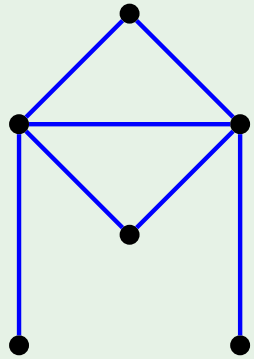
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





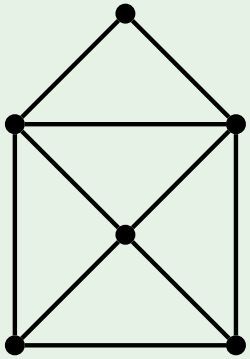
Théorèmes d'Euler

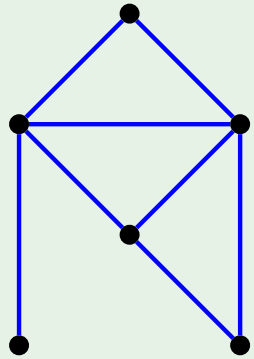
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





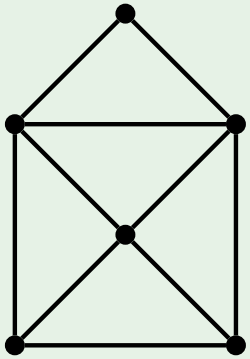
Théorèmes d'Euler

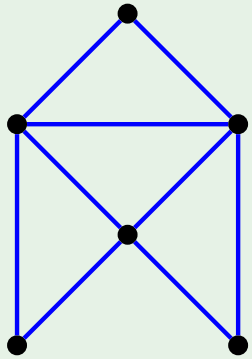
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





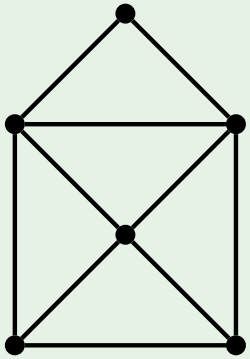
Théorèmes d'Euler

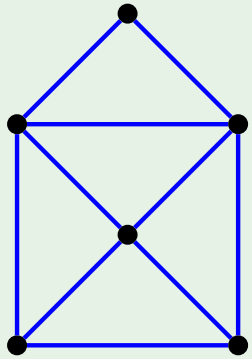
- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphe eulérien

Intuition : un graphe est eulérien s'il est possible de le dessiner sans lever la main.

Exemple





Théorèmes d'Euler

- Un graphe connexe admet une chaîne eulérienne ssi ses sommets sont tous de degré pair sauf au plus deux.
- Un graphe connexe admet un circuit eulérien ssi tous ses sommets ont un degré pair.

Graphes hamiltoniens

Hypothèse : on considère des graphes simples connexes.

Définition : chaîne hamiltonienne

Une chaîne qui passe une et une seule fois par chaque sommet du graphe.

Définition : cycle hamiltonien

Un cycle qui passe une et une seule fois par chacuns des sommets du graphe.

Graphe hamiltonien

Un graphe qui possède un cycle hamiltonien.

Graphe semi-hamiltonien

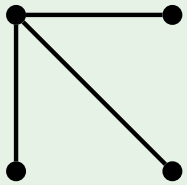
Un graphe qui possède que des chaînes hamiltoniennes.

Exemple graphe hamiltonien

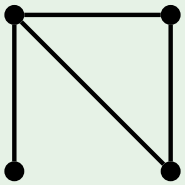
Pas de définition intuitive des graphes hamiltoniens

Exemples

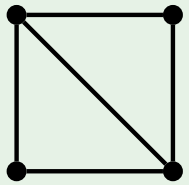
Graphe non-hamiltonien



Graphe semi-hamiltonien



Graphe hamiltonien



Pas de théorème général mais

- **Théorème de Dirac** un graphe simple $G = (S, A)$ à $n = |S|$ sommets $(n \geqslant 3)$ avec $\forall s \in S$, $d(s) \geqslant \frac{n}{2}$ est hamiltonien.
- **Théorème de Ore** Un graphe simple à n sommets $(n \geqslant 3)$ tel que la somme des degrés de toute paire de sommets non adjacents vaut au moins n est hamiltonien.
- ...

Nouvelles définitions pour les arbres

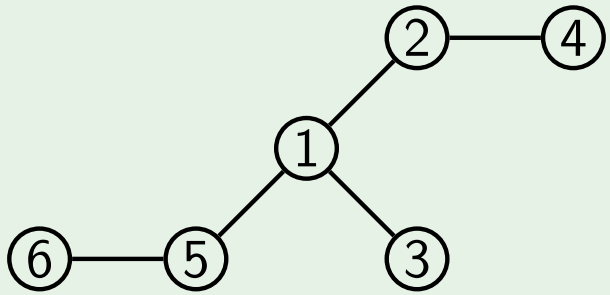
Définition

Un arbre est un graphe connexe sans cycle.

Définition

Un arbre est un graphe dans lequel deux sommets quelconques sont reliés par un et un seul chemin.

Exemple



Remarque Le nombre d'arêtes dans un arbre est égal au nombre de ses sommets moins 1

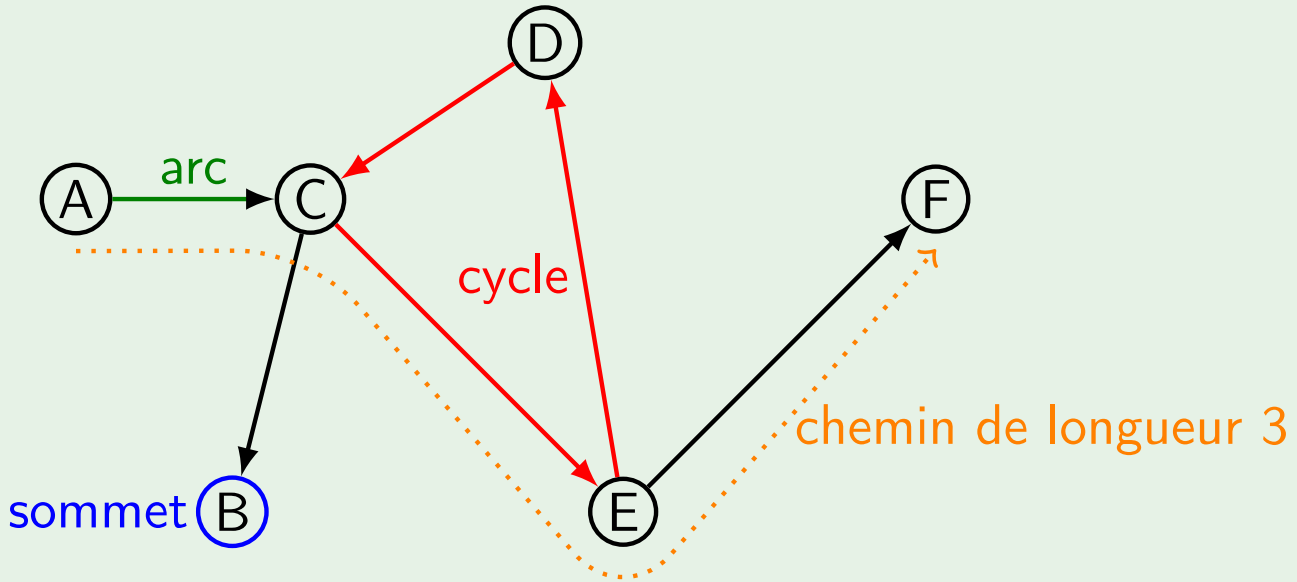
Graphes orientés

Graphes orientés

Un **graphe orienté** ou **digraphe** (*directed graph*) est une paire (S, A)

- S ensemble fini de **sommets** (*vertex/vertices*)
- A ensemble fini d'**arcs** (*arcs*) sous ensemble de $S \times S$
- un **chemin** (*path*) est une suite alternant sommets et arcs qui débute et termine par un sommet. Note **le chemin est dirigé.**
- le reste du vocabulaire de ne change pas : longueur, cycle, sommets adjacents, arc incident, eulérien, hamiltonien, ...

Exemple



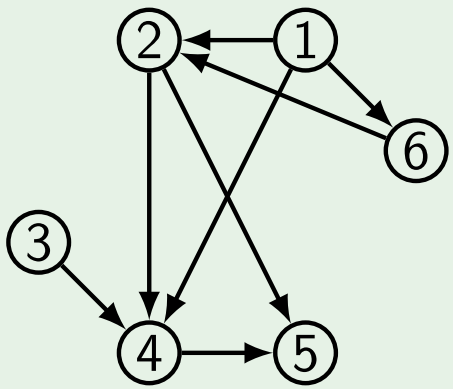
Degrés

Degré entrant et sortant

Soit s le somme d'un graphe orienté G

- le **degré sortant** de s, noté $d^+(s)$, est le nombre d'arcs ayant s comme origine.
- le **degré entrant** de s, noté $d^-(s)$, est le nombre d'arcs ayant s comme destination.
- le degré d'un sommet s, noté $d(s)$ est défini par $d^+(s) + d^-(s)$.

Exemple



- $d^+(1) = 3$ et $d^-(1) = 0$
- $d^+(2) = 2$ et $d^-(2) = 2$
- $d^+(3) = 1$ et $d^-(3) = 0$
- $d^+(4) = 1$ et $d^-(4) = 3$
- $d^+(5) = 0$ et $d^-(5) = 2$
- $d^+(6) = 1$ et $d^-(6) = 1$

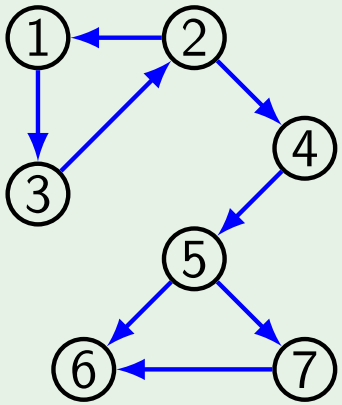
Graphe transposé

Définition

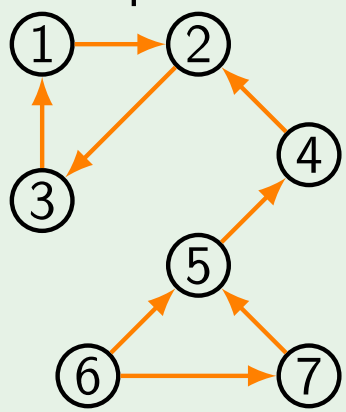
Le **graphe transposé** ou **graphe inverse** du digraphe $G = (S, A)$ est le graphe $G^T(S, A^T)$ avec $A^T = \{(y, x) : (x, y) \in A\}$.
G^T est le graphe obtenu à partir des sommets de G et dont les arcs sont dans le sens opposé à ceux de G.

Exemple

Graphe G



Graphe G^T



Propriétés

- $(G^T)^T = G$

Forte connexité

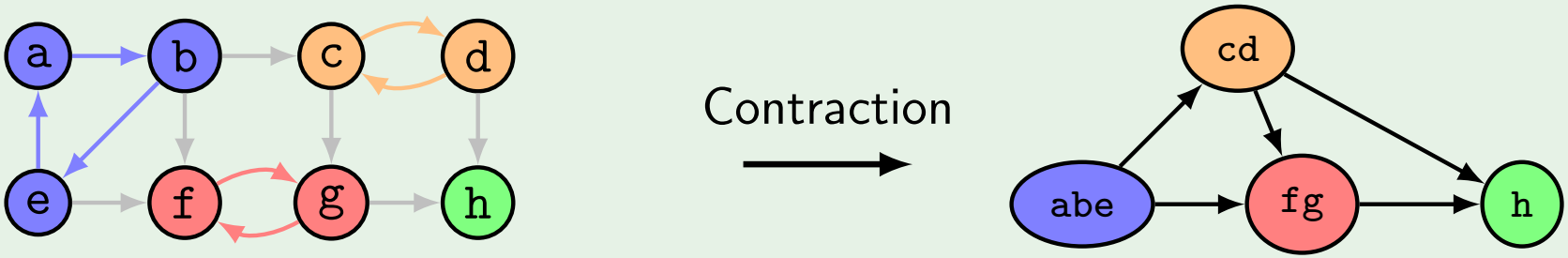
Définition : digraphe fortement connexe

Un digraphe $G = (S, A)$ est **fortement connexe** si pour toutes paires de sommets distincts (s_1, s_2) il existe un chemin de s_1 à s_2.

Définition : composante fortement connexe (CFC)

Une **composante fortement connexe** est un sous-graphe orienté G' du digraphe $G = (S, A)$ ayant la propriété de forte connexité.

Exemple



Remarque

La contraction des CFC produit nécessairement un DAG (*directed acyclic graph*)

Représentations mathématico-informatiques des (di)graphes

Représentation des graphes : matrice d'adjacence

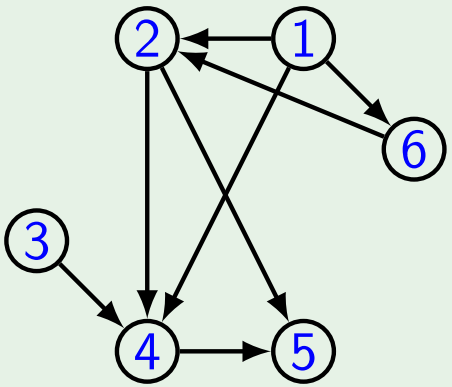
Définition

Pour un (di)graphe $G = (S, A)$, une matrice d'adjacence M est de dimension $(|S| \times |S|)$ et dont les cœfficients sont

$$\forall (i,j) \in |S| \times |S|, \quad M_{ij} = \begin{cases} 1 & \text{si} \quad (i,j) \in A \\ 0 & \text{autrement} \end{cases}$$

Remarque : dans le cas d'un graphe non orienté la matrice d'adjacence est symétrique. Dans les deux cas, les éléments diagonaux représentent les boucles.

Exemple : matrice d'adjacence d'un digraphe



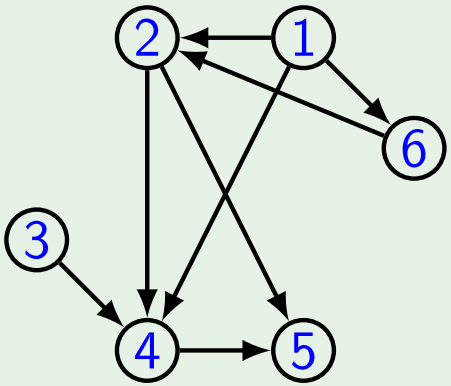
$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

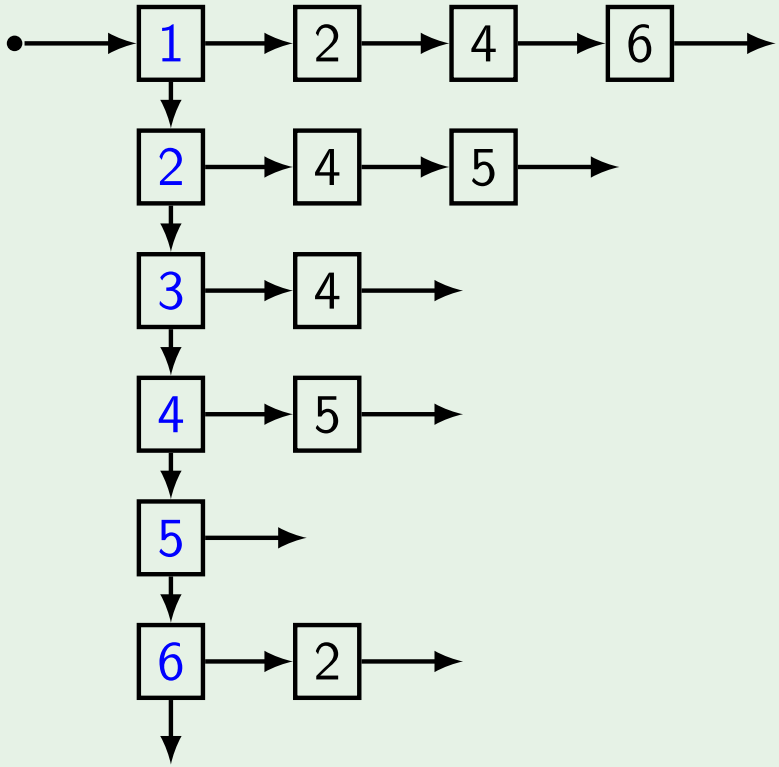
Représentation des graphes : listes d'adjacence

Définition

Structure de données fondée sur une méthode par chaînage et manipulation de pointeurs $\Rightarrow$ **représentation creuse**.

Exemple : liste d'adjacence d'un digraphe





Implémenation dans la bibliothèque `libin103`

Définition d'un graphe avec sommets identifiés avec des entiers

```
typedef struct integer_adjlist_elmt_ {
  int vertex;
  double weight;
} integer_adjlist_elmt_t;

typedef struct integer_adjlist_ {
  int vertex;
  generic_set_t adjacent;
} integer_adjlist_t;

typedef struct integer_graph_ {
  int vcount;
  int ecount;
  generic_list_t adjlists;
} integer_graph_t;
```

Point de vigilance !

Une liste d'adjacence est modélisée par

- une **liste chaînée** (générique) de structures
- dont un des champs est un **ensemble** (générique)
- dont les éléments sont des **structures**.

Remarque on considère des (di)graphes dont les arcs/arêtes peuvent avoir une valeur (`weight`). Aspect utile pour la prochaine séance.

API des graphes

API Création / Destruction

```
void integer_graph_init(integer_graph_t *g);
void integer_graph_destroy(integer_graph_t *g);
```

API Accesseurs / Prédicats

```
bool integer_graph_is_adjacent(integer_graph_t *g, int v1, int v2);
int integer_graph_vcount(integer_graph_t *g);
int integer_graph_ecount(integer_graph_t *g);
```

API Insertion / Suppression

```
int integer_graph_ins_vertex(integer_graph_t*, int v);
int integer_graph_ins_edge(integer_graph_t *g,
                           int v1, int v2, double w);
int integer_graph_rem_vertex(integer_graph_t *g, int v);
int integer_graph_rem_edge(integer_graph_t *g, int v1, int v2);
```

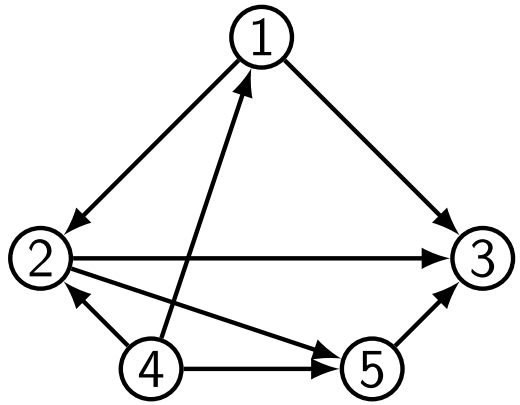
Exemple de construction d'un graphe

Code source

```
int main (int argc, char** argv) {
  integer_graph_t graph;
  integer_graph_init (&graph);

  for (int i = 1; i < 6 ; i++) {
    integer_graph_ins_vertex(&graph, i);
  }
  integer_graph_ins_edge(&graph, 1, 2, 0.0);
  integer_graph_ins_edge(&graph, 1, 3, 0.0);
  integer_graph_ins_edge(&graph, 2, 3, 0.0);
  integer_graph_ins_edge(&graph, 2, 5, 0.0);
  integer_graph_ins_edge(&graph, 4, 1, 0.0);
  integer_graph_ins_edge(&graph, 4, 5, 0.0);
  integer_graph_ins_edge(&graph, 4, 2, 0.0);
  integer_graph_ins_edge(&graph, 5, 3, 0.0);

  print_graph (&graph);
  integer_graph_destroy (&graph);
  return EXIT_SUCCESS;
}
```



Remarque

Pour représenter un graphe non orienté,

- insérer une arête et son symétrique

Complexité des opérations sur les graphes

Pour un graphe $G = (S, A)$

Fonction	**Complexité**	**Commentaire**
Initialisation	$\mathcal{O}(1)$	Mise à zéro des champs de la structure
Destruction	$\mathcal{O}(\|S\| + \|A\|)$	Pour chaque élément de la liste il faut détruire un ensemble
#Arcs / #Sommets	$\mathcal{O}(1)$	Accès direct aux champs de la structure
Test d'adjacence	$\mathcal{O}(\|S\|)$	Parcours la liste pour trouver la position
Insertion de sommet	$\mathcal{O}(\|S\|)$	Parcours de la liste pour savoir si le sommet est déjà présent
Insertion d'arc	$\mathcal{O}(\|S\|)$	On cherche dans la liste les positions des sommets
Suppression de sommet	$\mathcal{O}(\|S\| + \|A\|)$	Traverse tout le graphe pour vérifier qu'il n'y a plus d'arc impliquant le sommet
Suppression d'arc	$\mathcal{O}(\|S\|)$	On cherche dans la liste les positions du premier sommet

Fonction print_graph (le retour des itérateurs)

Code source

```
void print_graph (integer_graph_t* graph) {
  generic_list_elmt_t* elem1 =
    generic_list_head(&(graph->adjlists));
  /* Boucle sur les elements de la liste */
  for (; elem1 != NULL; elem1 = generic_list_next(elem1)) {
    integer_adjlist_t* tempV1 =
      (integer_adjlist_t*)generic_list_data(elem1);
    printf ("Vertex %d: ", tempV1->vertex);

    generic_list_elmt_t* elem2 =
      generic_list_head(&(tempV1->adjacent));
    /* Boucle sur les elements de l'ensemble (set) */
    for (; elem2 != NULL; elem2 = generic_list_next(elem2)) {
      integer_adjlist_elmt_t *tempV2 =
        (integer_adjlist_elmt_t*)generic_list_data(elem2);
      printf ("%d->", tempV2->vertex);
    }
    printf ("\n");
  }
```

Parcours de graphes

Utilité des parcours de graphes

Point de départ :

- Un graphe ne sert pas à stocker des données, c'est un **modèle de données** représentant des relations entre données.
- On **analyse** le graphe pour déduire des propriétés sur les données.

Un **parcours de graphe** permet d'extraire des informations sur le graphe

- Prouver l'existence de ou calculer des chemin(s) avec ou sans contraintes
  - p. ex., plus court chemin entre deux sommets (cf cours 6)
- Analyse la topologie
  - Composantes (fortement pour graphe orienté) connexes (notion accessibilité).
- Étudier des notions de dépendances entre les sommets : tri topologique
- Calculer des recouvrements par des sous-graphes ou arbres (cf cours 6).
- etc.

Essentiellement deux types de parcours

- **parcours en largeur**
- **parcours en profondeur**

Parcours en largeur

Équivalent du parcours par niveaux des arbres :

- On part d'un sommet,
- on visite tous les voisins,
- on visite tous les voisins des voisins. . .

$\Rightarrow$ les sommets de distance d découverts avant ceux de distance $d + 1$

Mise en œuvre

Utilisation de la **technique de marquage** (attribution d'une couleur) des sommets lors du parcours

- « Blanc » : non visité (ou « non marqué »).
- (« Gris » : en cours de visite, si besoin.)
- « Noir » : déjà visité (ou « marqué »).

pour éviter de boucler (c.-à-.d, éviter de visiter plusieurs fois le même sommet)

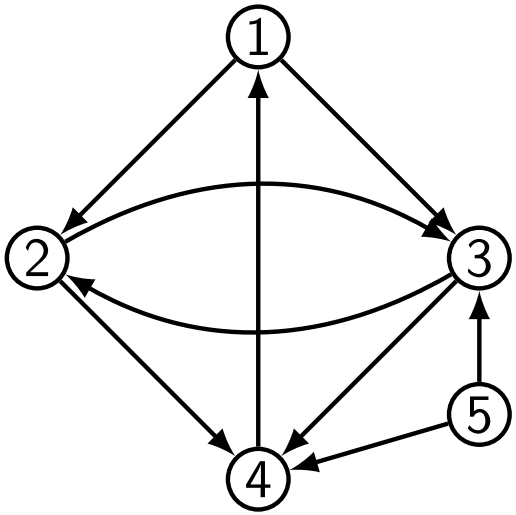
Parcours en largeur

Algorithme

```
Function bfs(graph, racine) :
    queue_t q;
    queue_enqueue(q, racine);
    set_t s;
    set_insert(s, racine);
    while queue_size(q) > 0 do
        elem = queue_dequeue(q);
        /* Traiter elem */
        for /* Pour tous les voisins de elem */ do
            if set_is_member(s, elem) == false then
                set_insert(s, elem);
                queue_enqueue(q, elem);
            end
        end
    end
```

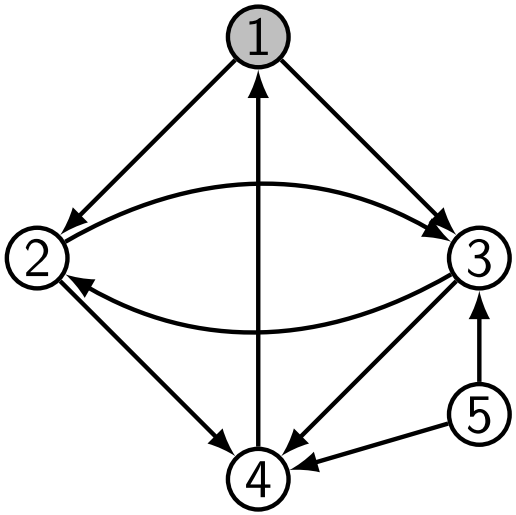
- utilisation d'une **file** pour stocker les voisins
- utilisation d'un ensemble pour « marquer » les sommets visités
- **Complexité** en $\mathcal{O}(|S| + |A|)$ avec `libin103`.

Parcours en largeur – Exemple



Initialement tous les sommets sont « blancs »

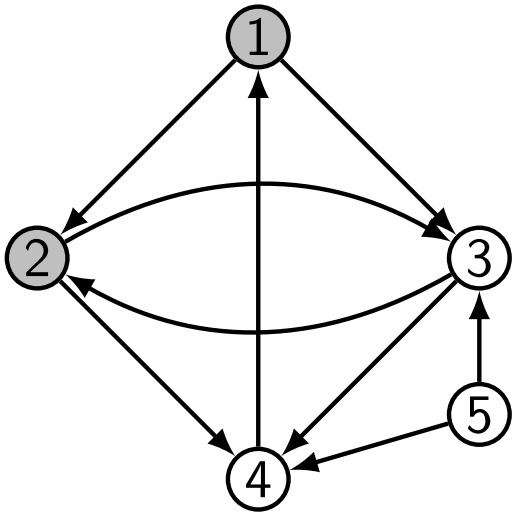
Parcours en largeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1

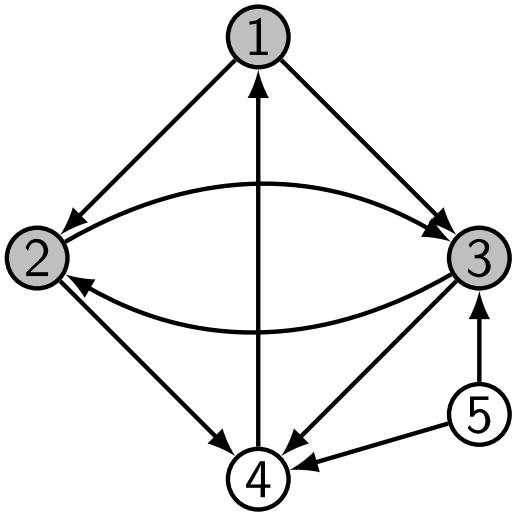
Parcours en largeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. sommet 2 dans la file

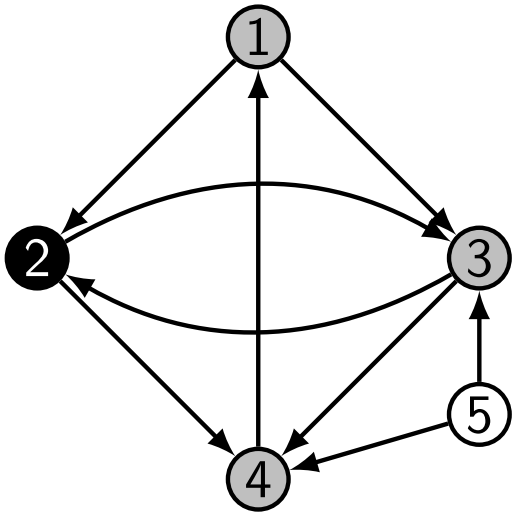
Parcours en largeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. sommet 2 dans la file
3. sommet 3 dans la file

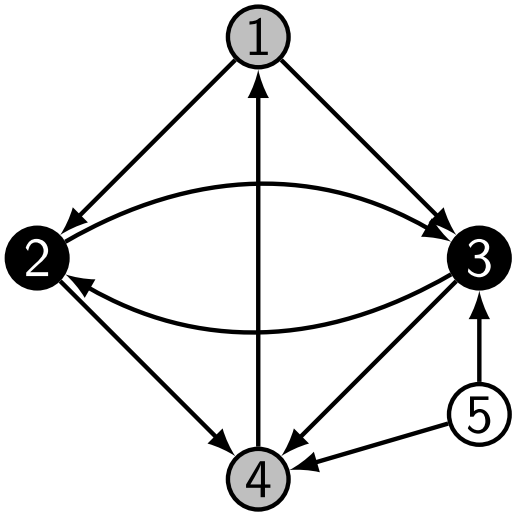
Parcours en largeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. sommet 2 dans la file
3. sommet 3 dans la file
4. sommet 4 dans la file et fin traitement sommet 2

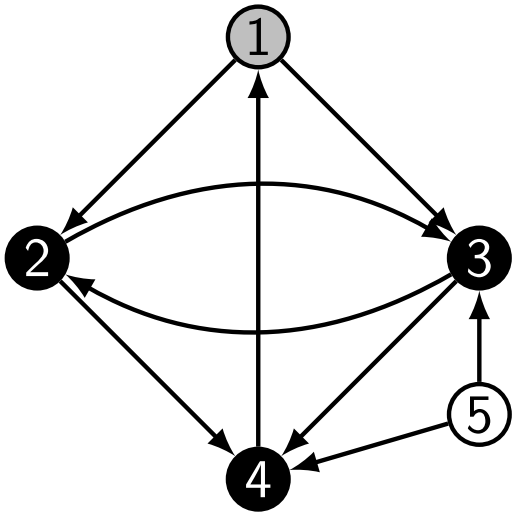
Parcours en largeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. sommet 2 dans la file
3. sommet 3 dans la file
4. sommet 4 dans la file et fin traitement sommet 2
5. fin traitement sommet 3

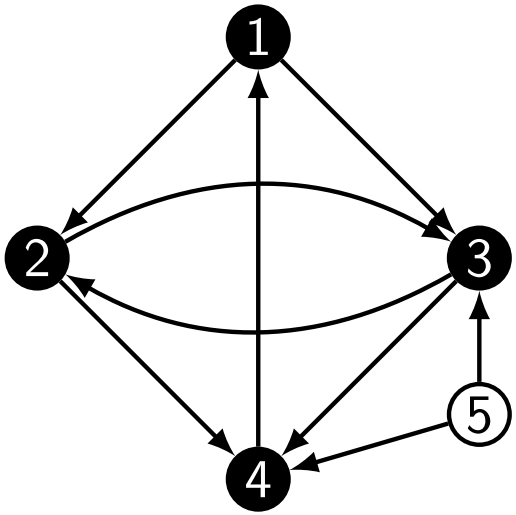
Parcours en largeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. sommet 2 dans la file
3. sommet 3 dans la file
4. sommet 4 dans la file et fin traitement sommet 2
5. fin traitement sommet 3
6. fin traitement sommet 4

Parcours en largeur – Exemple



Initialement tous les sommets sont « blancs »

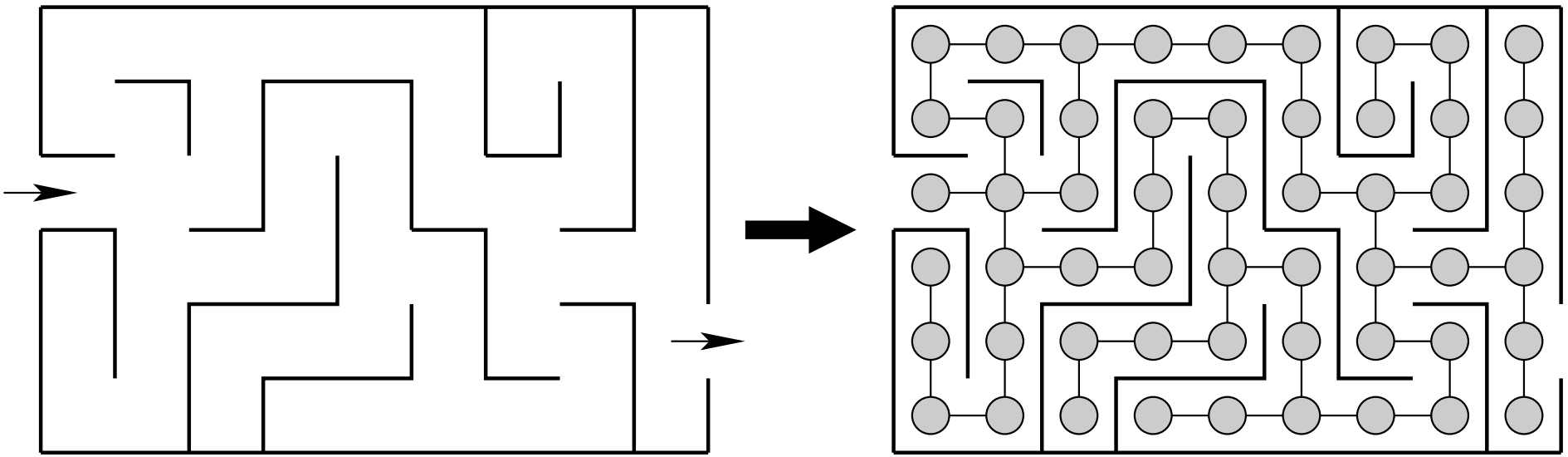
1. On commence par le sommet 1
2. sommet 2 dans la file
3. sommet 3 dans la file
4. sommet 4 dans la file et fin traitement sommet 2
5. fin traitement sommet 3
6. fin traitement sommet 4
7. fin traitement sommet 1

Remarque

- On peut avoir une couverture partielle du graphe en fonction du sommet de départ
- L'ordre de visite des sommets peut dépendre de l'ordre de construction du graphe

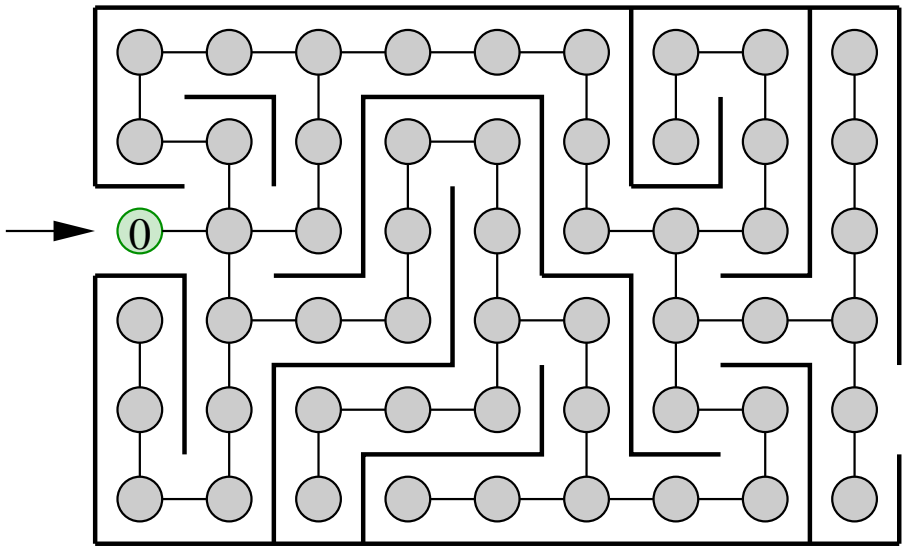
Parcours en largeur – application à un labyrinthe – 1

Objectif trouver (le plus court chemin) vers la sortie



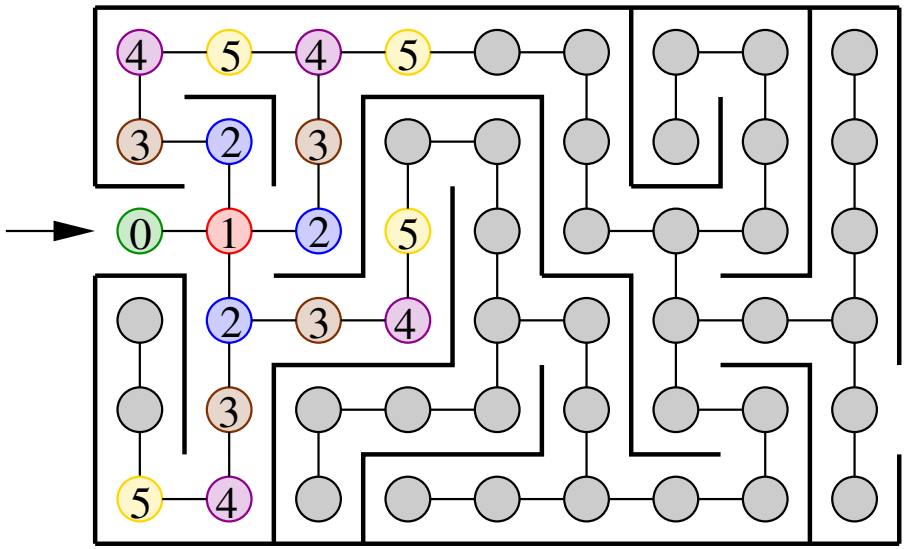
Parcours en largeur – application à un labyrinthe – 2

On commence le parcours par l'entrée (sommet vert).



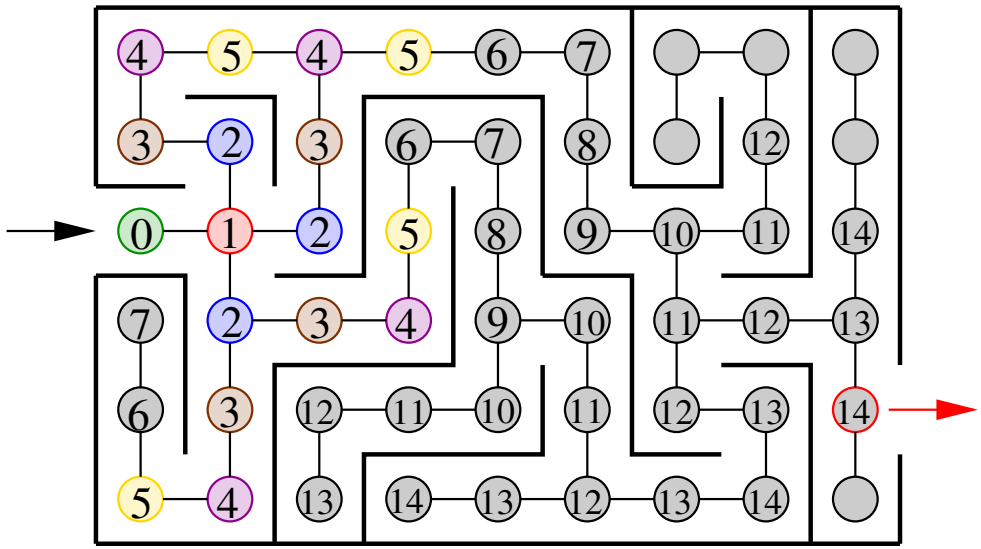
Parcours en largeur – application à un labyrinthe – 3

On descend en largeur en mémorisant le père de chaque sommet visité.



Parcours en largeur – application à un labyrinthe – 4

Arrivé à la sortie, on retrace le chemin en remontant les pères.



Parcours en profondeur

Équivalent du parcours en profondeur des arbres :

- On descend au plus profond en premier.
- Algorithme naturellement récursif.

$\Rightarrow$ aucune garantie de plus court chemin.

Mise en œuvre

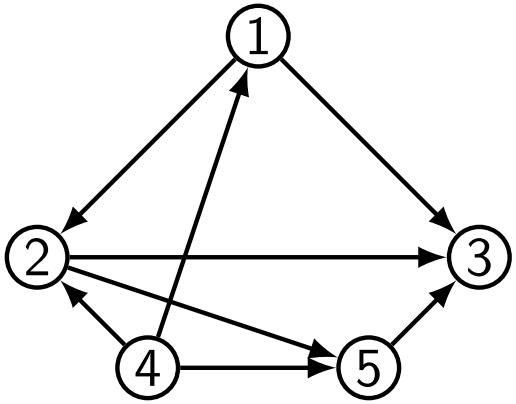
Comme pour le parcours en largeur, on veut éviter les boucles $\Rightarrow$ technique de marquage.

Algorithme

```
Function dfs(graph, sommet) :
    marquer(sommet);
    /* pre-traiter(sommet) */
    for /* Pour tous les voisins v de elem non marques */ do
        dfs(graph, v);
    end
    /* post-traiter(sommet) */
```

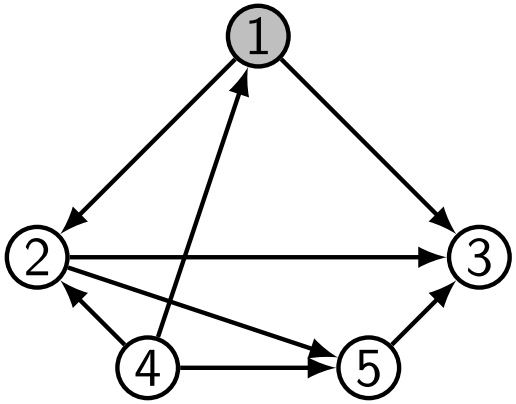
- Complexité en $\mathcal{O}(|S| + |A|)$ avec `libin103`

Parcours en profondeur – Exemple



Initialement tous les sommets sont « blancs »

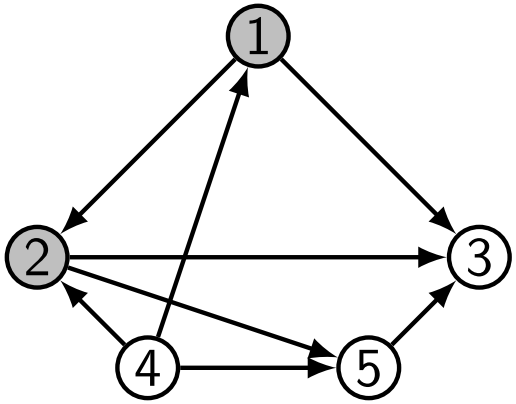
Parcours en profondeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1

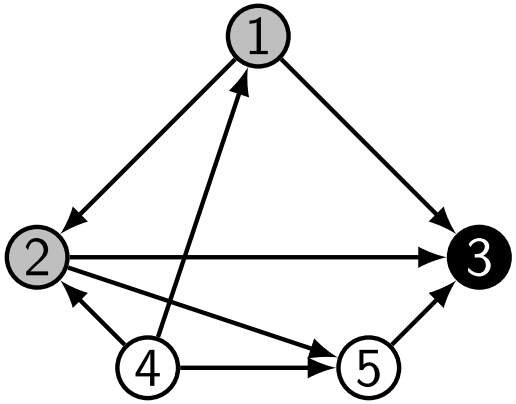
Parcours en profondeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. puis le sommet 2

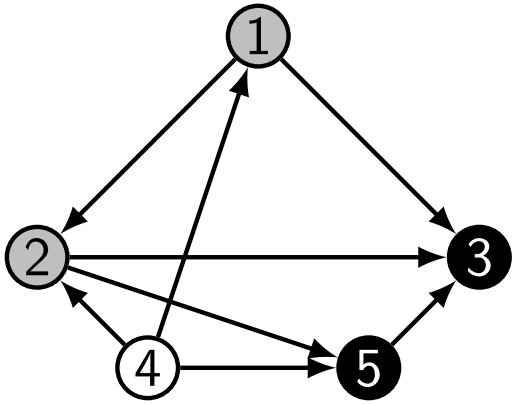
Parcours en profondeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. puis le sommet 2
3. puis le sommet 3 (et fin)

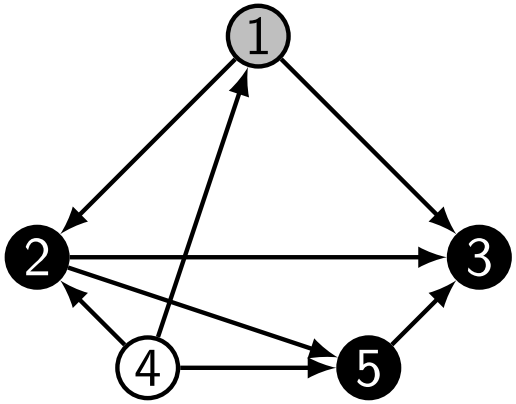
Parcours en profondeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. puis le sommet 2
3. puis le sommet 3 (et fin)
4. puis le sommet 5 (et fin)

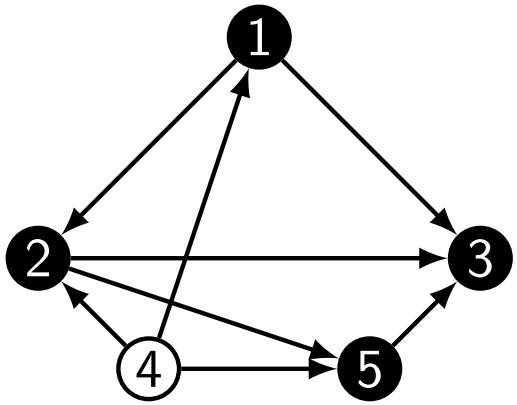
Parcours en profondeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. puis le sommet 2
3. puis le sommet 3 (et fin)
4. puis le sommet 5 (et fin)
5. fin traitement sommet 2

Parcours en profondeur – Exemple



Initialement tous les sommets sont « blancs »

1. On commence par le sommet 1
2. puis le sommet 2
3. puis le sommet 3 (et fin)
4. puis le sommet 5 (et fin)
5. fin traitement sommet 2
6. fin traitement sommet 1

Remarques

- On peut avoir une couverture partielle du graphe en fonction du sommet de départ
- L'ordre de visite des sommets peut dépendre de l'ordre de construction du graphe

Parcours en profondeur – extension

On peut étendre le parcours en profondeur pour garder une notion de temporalité

- enregistré les « dates » de début et de fin de visite.

Algorithme

```
Function dfs(graph, sommet) :
    marquer(sommet);
    debut[sommet] = cpt;
    cpt = cpt + 1;
    /* pre-traiter(sommet) */
    for /* Pour tous les voisins v de elem non marques */ do
        parent[v] = sommet;
        dfs(graph, v);
    end
    fin[sommet] = cpt;
    cpt = cpt + 1;
    /* post-traiter(sommet) */
```

Remarque

Dans un graphe sans cycle, les dates de fin de visite définissent une relation d'ordre sur les sommets : si il y a un chemin de u à v alors $\texttt{fin}[u] > \texttt{fin}[v]$.

Parcours en profondeur – application **tri topologique**

Objectif : un ensemble de tâches à ordonner selon une contrainte de précédence (certaines tâches sont à réaliser avant d'autres). Par exemple,

- l'ordre de compilation dans les `Makefile`
- gestion de projets

Modèle

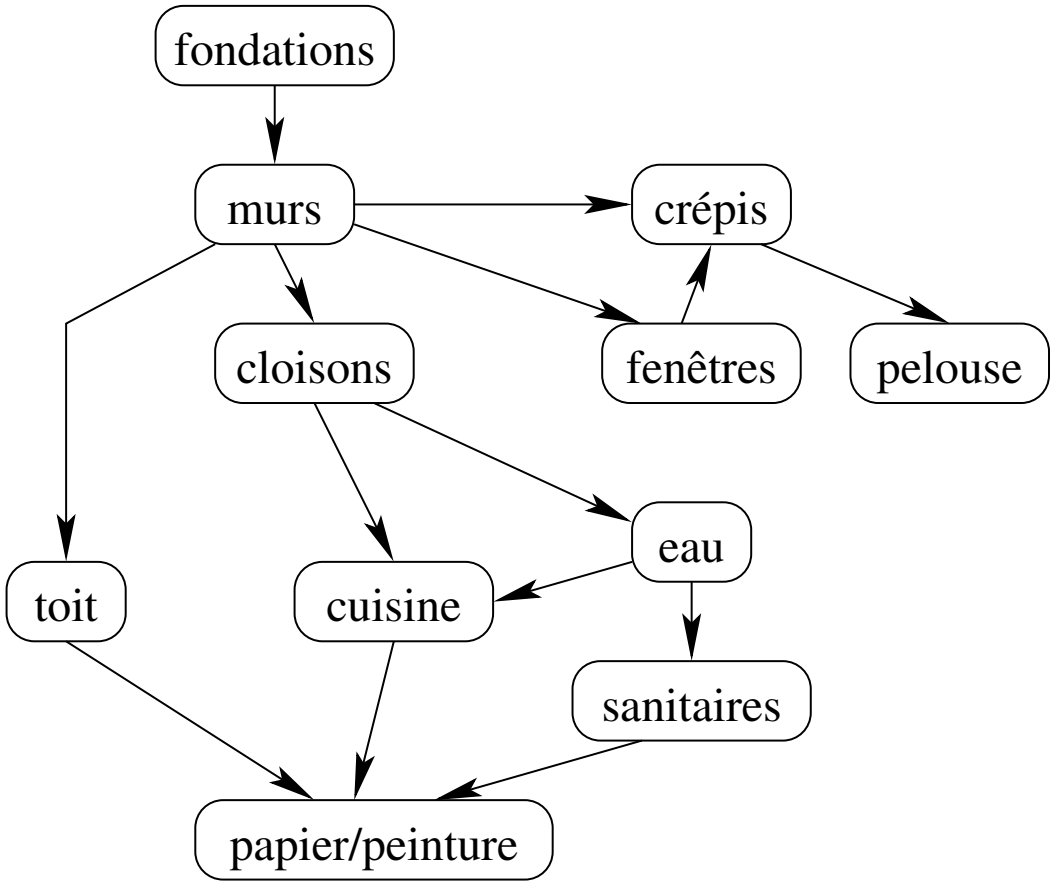
Le graphe modélise les dépendances entre les tâches.
Hypothèse : le graphe est sans cycle

Idée de l'algorithme

- Effectuer un parcours en profondeur du graphe et enregistrer date de fin
- Retourner les sommets dans l'ordre décroissant des dates de fin

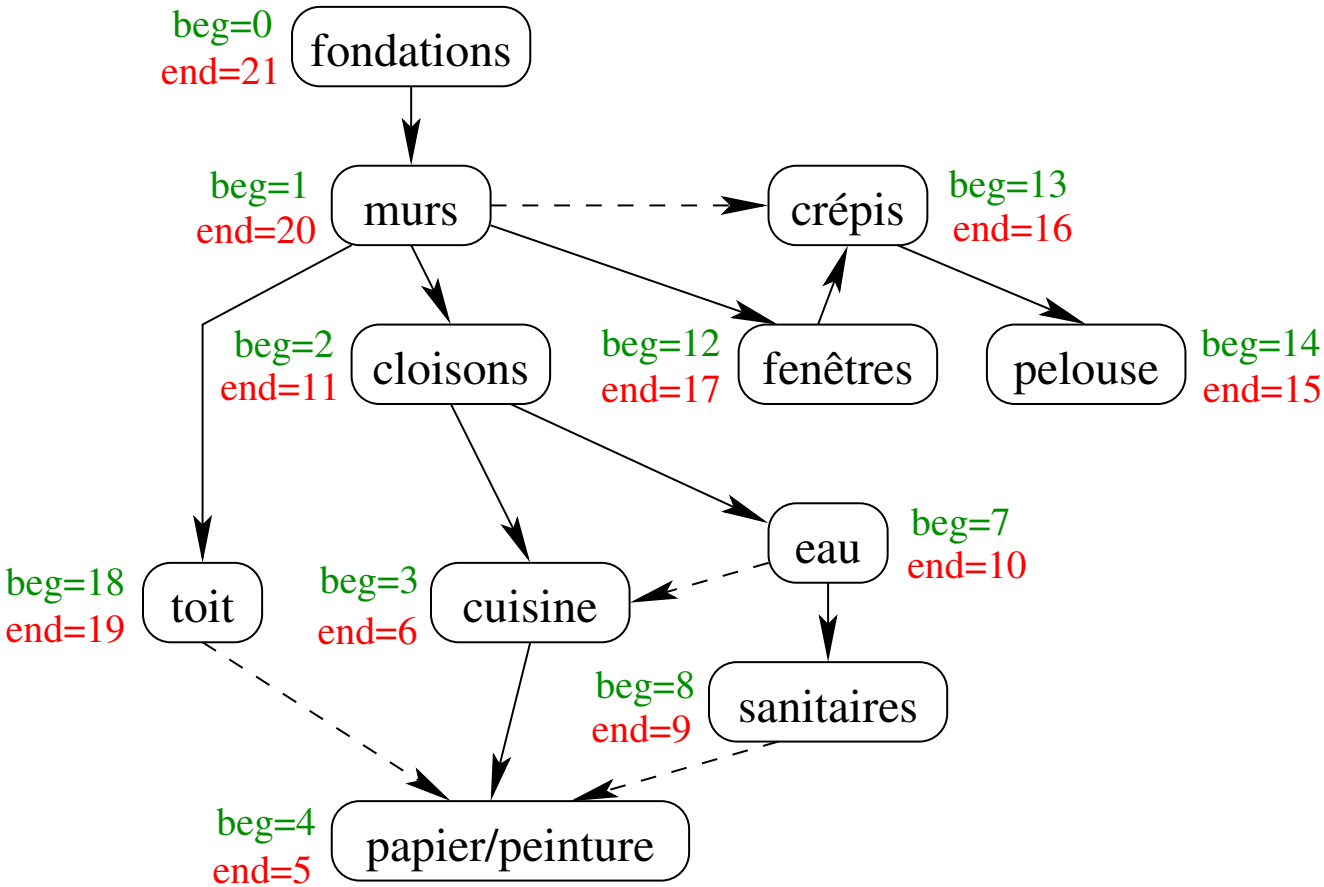
Parcours en profondeur – application **tri topologique**

Décomposition des tâches pour construire une maison, les arcs d'un graphe décrivent la relation faire u puis v.



Parcours en profondeur – application **tri topologique**

Application du parcours en profondeur



Parcours en profondeur – application **tri topologique**

Tâches à effectuer en ordre décroissant des dates de fin de visite.