Cours n: Graphes

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

2011-12

Algorithmique

Aujourd'hui

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

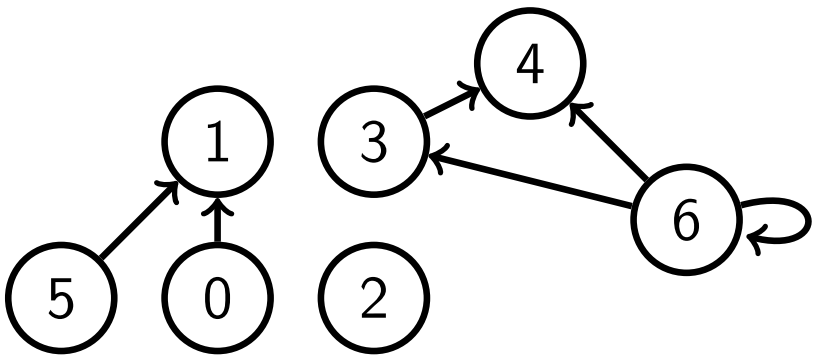
Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

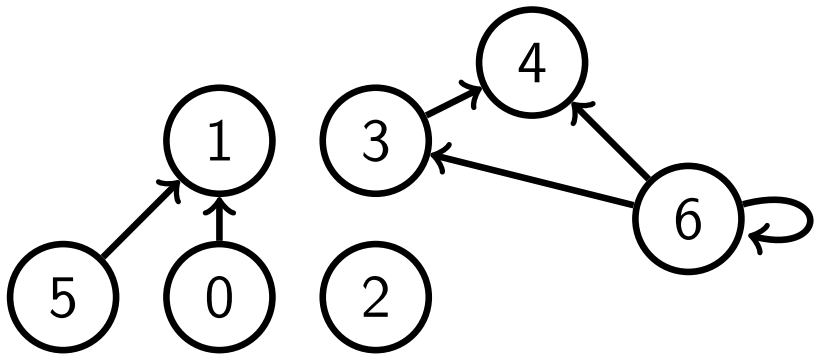
Un graphe orienté

- Un *graphe orienté* (digraph) est donné par un couple $G = (V, E)$, où
  - V est un ensemble.
  - $E \subset V \times V$.

- Exemple:
  - $V = \{0, 1, \ldots, 6\}$.
  - $E = \{(0,1), (3,4), (5,1), (6,3), (6,4), (6,6)\}$.



Vocabulaire

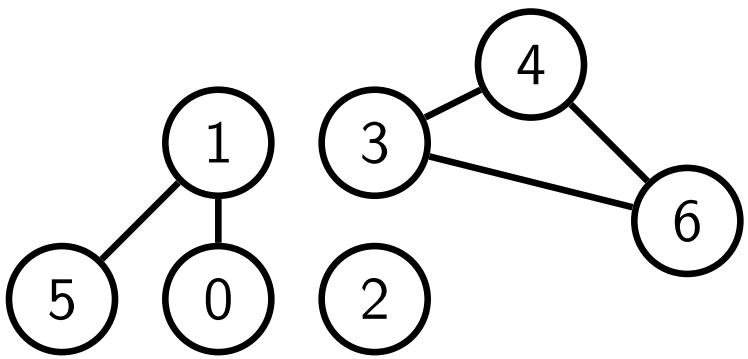


- Les éléments de V sont appelés des *sommets* (parfois aussi des *nœuds*).
- Les éléments e de E sont appelés des *arcs*.
- Si $e = (u, v)$, u est appelé *la source* de e, v est appelé *la destination* de e.
- Remarque:
  - Les boucles (les arcs (u, u)) sont autorisées.

Un graphe

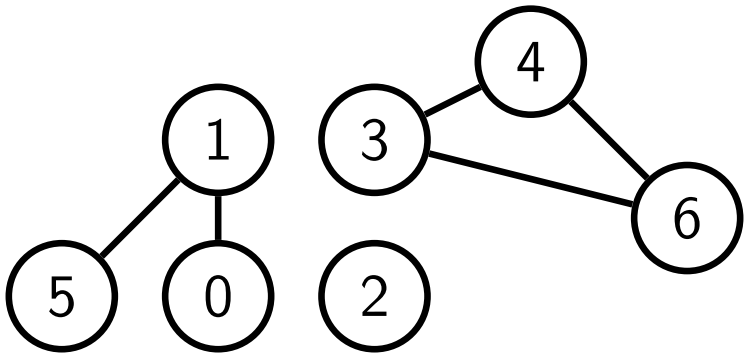
- Un *graphe*[1] est donné par un couple $G = (V, E)$, où
  - V est un ensemble.
  - E est un ensemble de paires $\{u, v\}$ avec $u, v \in V$.
- On convient de représenter une paire $\{u, v\}$ par (u, v) ou (v, u).
- Autrement dit, (u, v) et (v, u) dénotent la même arête.

- Exemple:
  - $V = \{0, 1, \ldots, 6\}$
  - $E = \{(0, 1), (3, 4), (5, 1), (6, 3), (6, 4)\}$.



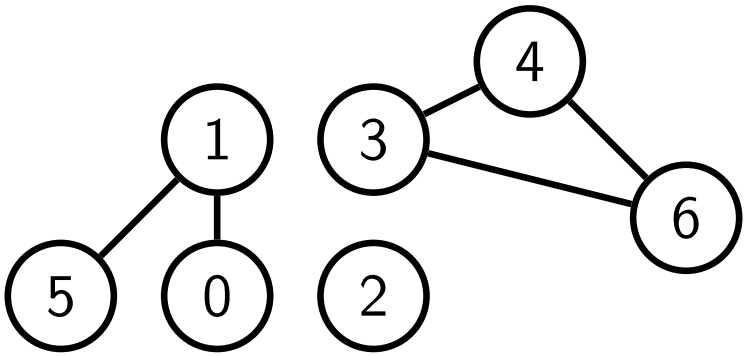
[1] Lorsqu'on ne précise pas, par défaut, un graphe est non-orienté.

Vocabulaire



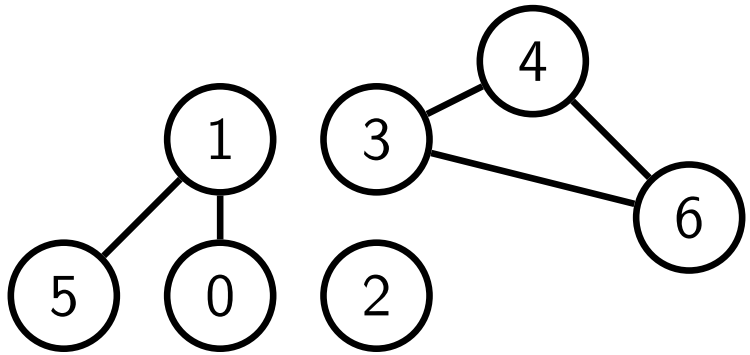
- Les éléments e de E sont appelés des *arêtes*. Si $e = (u, v)$, u et v sont appelés les *extrémités* de e.
- Remarque: (sauf autre convention explicite)
  - Les boucles ne sont pas autorisées.

Vocabulaire (cas non-orienté)



- u et v sont dits *voisins* s'il y a une arête entre u et v.
- Le degré de u est le nombre de voisins de u.

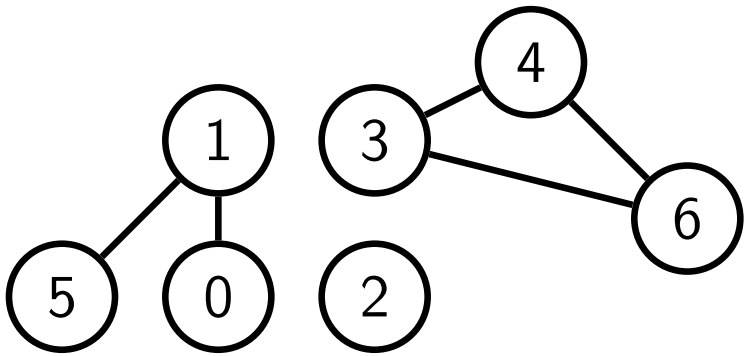
Vocabulaire (cas non-orienté)



- Un *chemin du sommet s vers le sommet t* est une suite $e_0, e_1, \cdots, e_n$ de sommets telle que $e_0 = s$, $e_n = t$, $(e_{i-1}, e_i) \in E$, pour tout $1 \leq i \leq n$.
  - n est appelé la *longueur* du chemin, et on dit que t *est joignable* à partir de s.
  - Le chemin est dit *simple* si les e_i sont distincts deux-à-deux.
  - Un *cycle* est un chemin de longueur non-nulle avec $e_0 = e_n$.
- Le sommet s est dit *à distance n* de t s'il existe un chemin de longueur n entre s et t, mais aucun chemin de longueur inférieure.

Composantes connexes (cas non-orienté)

- Prop. La relation “être joignable” est une relation d’équivalence.
- Les classes d’équivalence sont appelées les *composantes connexes*.



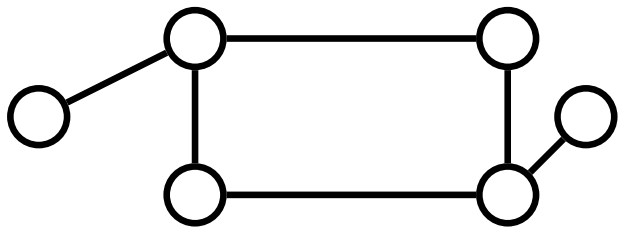
- Un graphe est dit *connexe* s’il n’y a qu’une seule classe d’équivalence.
  - Autrement dit, tout sommet est joignable à partir de tout sommet.

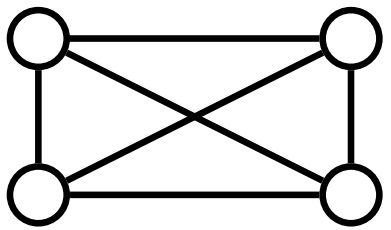
Les graphes sont partout!

- Beaucoup de problèmes se modélisent par des objets et des relations entre objets.
- Exemples:
  - Le graphe routier.
  - Les réseaux informatiques.
  - Le graphe du web.
- Beaucoup de problèmes se ramènent à des problèmes sur les graphes.
- Théorie des graphes:
  - Euler, Hamilton, Kirchhoff, König, Edmonds, Berge, Lovász, Seymour,...
- Les graphes sont omniprésents en informatique.

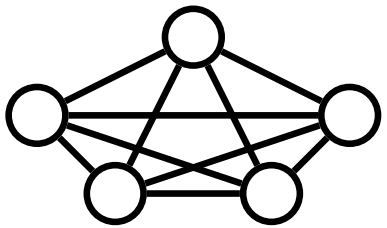
Théorie des graphes. Exemple 1: graphes planaires.

- Un graphe est dit *planaire* s'il peut se représenter sur un plan sans qu'aucune arête n'en croise une autre.
- Planaire:

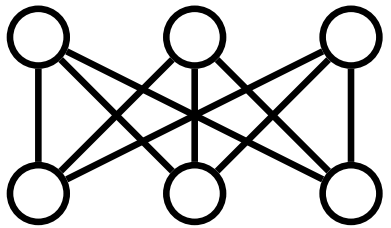


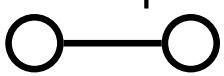
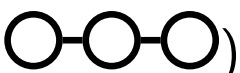


- Non-planaire: K_5



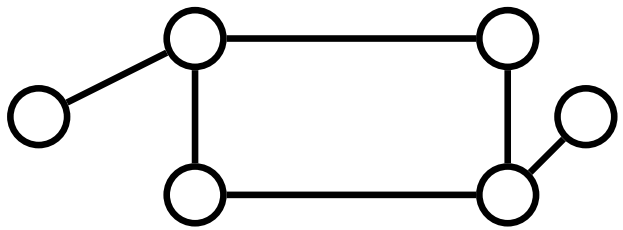
$K_{3,3}$

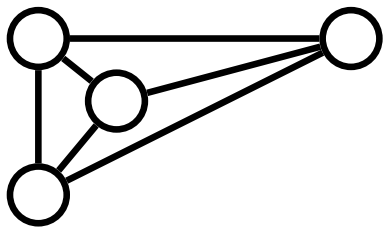


- Théorème [Kuratowski-Wagner] Un graphe fini est planaire ssi il ne contient pas de sous-graphe qui soit une expansion de K_5 ou de $K_{3,3}$.
  - Une expansion consiste à ajouter un ou plusieurs sommets sur une ou plusieurs arêtes (exemple:  devient )

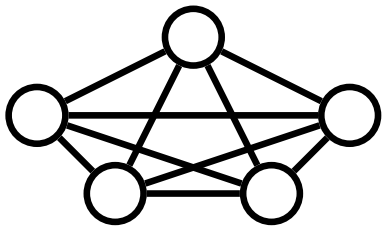
Théorie des graphes. Exemple 1: graphes planaires.

- Un graphe est dit *planaire* s'il peut se représenter sur un plan sans qu'aucune arête n'en croise une autre.
- Planaire:

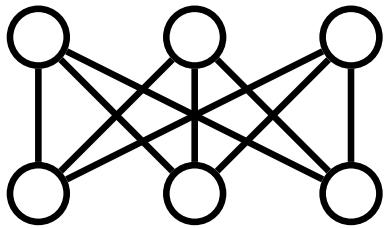


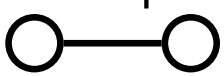
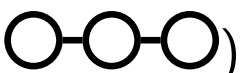


- Non-planaire: K_5



$K_{3,3}$



- Théorème [Kuratowski-Wagner] Un graphe fini est planaire ssi il ne contient pas de sous-graphe qui soit une expansion de K_5 ou de $K_{3,3}$.
  - Une expansion consiste à ajouter un ou plusieurs sommets sur une ou plusieurs arêtes (exemple:  devient )

Théorie des graphes. Exemple 2: Coloriage de graphe.

- Allouer des fréquences GSM correspond à colorier les sommets d'un graphe.
  - sommets: des émetteurs radio.
  - arête entre u et v: le signal de u perturbe v ou réciproquement.
  - couleur: fréquence radio.
- Le problème de *coloriage d'un graphe*: colorier les sommets d'un graphe de telle sorte qu'il n'y ait aucune arête entre deux sommets d'une même couleur.

Un coloriage avec 4 couleurs

Théorie des graphes. Exemple 2: Coloriage de graphe.

- Théorème: Appel et Haken (76): tout graphe planaire est coloriable avec 4 couleurs (preuve avec 1478 cas critiques).
  - Robertson, Sanders, Seymour, Thomas, Gonthier, Werner.

- On ne connaît aucun algorithme pour déterminer si un graphe général est coloriable avec 4 couleurs (même avec 3) qui fonctionne en temps polynomial (le problème est *NP*-complet).

Applications des graphes. Exemple 3: évaluer une page de la toile

- Brevet "Method for Node Ranking in a Linked Database" (Université Stanford, Janvier 1997).
- En simplifiant.
  - On considère le graphe des liens entre pages.
  - Modèle théorique:
    - Avec probabilité $0 < c < 1$, un surfeur abandonne la page actuelle et recommence sur une des n pages du web, choisie de manière équiprobable.
    - Avec probabilité $1 - c$, le surfeur suit un des liens de la page actuelle j, choisie de manière équiprobable parmi tous les liens l_j émis.
  - Quelle que soit la distribution initiale, on convergera vers une unique distribution stationnaire.
  - On évalue une page par la probabilité de cette page dans la distribution stationnaire.

Aujourd'hui

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

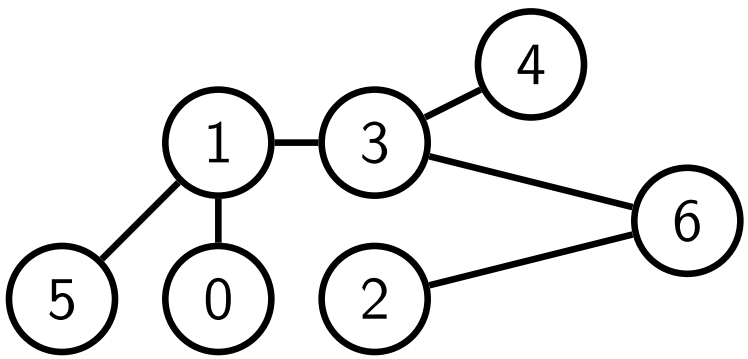
Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Les arbres

- Un graphe[2] connexe sans cycle est appelé un *arbre* (libre).
- Un graphe[2] sans-cycle est appelé une *forêt*:
  - chacune de ses composantes connexes est un arbre.

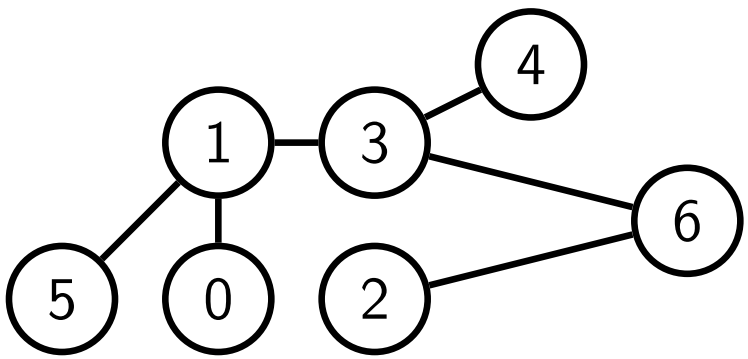


[2]non-orienté.

Les arbres sont partout!

- Un graphe[2] connexe sans cycle est appelé un *arbre* (libre).
- Un graphe[2] sans-cycle est appelé une *forêt*:
  - chacune de ses composantes connexes est un arbre.

- Dès qu'on a des objets, des relations entre objets, et pas de cycle, on a donc un arbre ou une forêt.

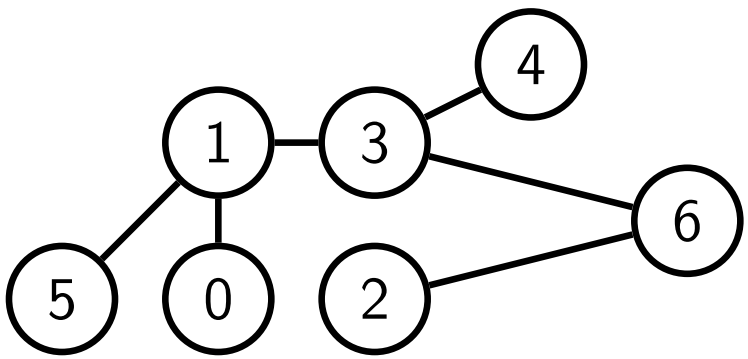


[2] non-orienté.

Les arbres sont partout!

- Un graphe[2] connexe sans cycle est appelé un *arbre* (libre).
- Un graphe[2] sans-cycle est appelé une *forêt*:
  - chacune de ses composantes connexes est un arbre.

- Dès qu'on a des objets, des relations entre objets, et pas de cycle, on a donc un arbre ou une forêt.



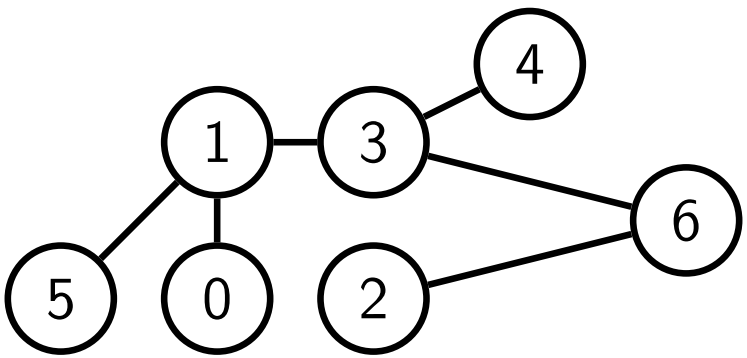
- Les arbres sont omniprésents en informatique.

[2] non-orienté.

Une caractérisation & Quelques propriétés

Soit $G = (V, E)$ un graphe[3]. Les propriétés suivantes sont équivalentes:

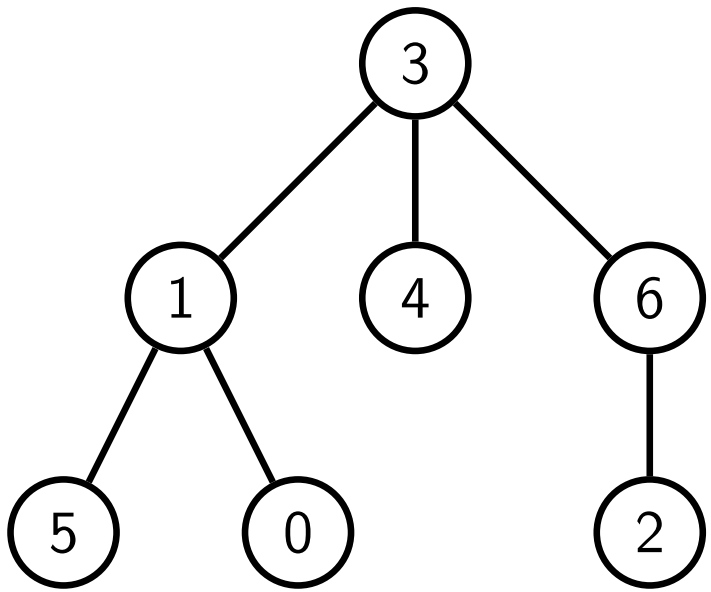
- G est un arbre (libre).
- Deux sommets quelconques de V sont connectés par un unique chemin simple.
- G est connexe, mais ne l'est plus si on enlève n'importe laquelle de ses arêtes.
- G est connexe, et $|E| = |V| - 1$.
- G est sans cycle, et $|E| = |V| - 1$.
- G est sans cycle, mais ne l'est plus si l'on ajoute n'importe quelle arête.



[3] non-orienté.

Les arbres poussent de haut en bas en informatique

- On distingue souvent un sommet que l'on appelle *sa racine*.
- On dessine un arbre
  - en plaçant la racine tout en haut.
  - puis en plaçant les sommets à distance i de la racine à la ligne i.
- Exemple: pour l'arbre libre précédent, en prenant le sommet d'étiquette 3 comme racine.



Une vision récursive: cas général

- Un arbre correspond à un couple formé
  1. d'un sommet particulier, appelé sa racine,
  2. et d'une partition des sommets restants en un ensemble d'arbres.

- Cette vision permet de faire des preuves inductives.
  - Exemple: montrer qu'un arbre dont tous les sommets sont d'arité au moins 2 possède plus de feuilles que de sommets internes.

- Mais reste informatiquement encore un peu compliquée.

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

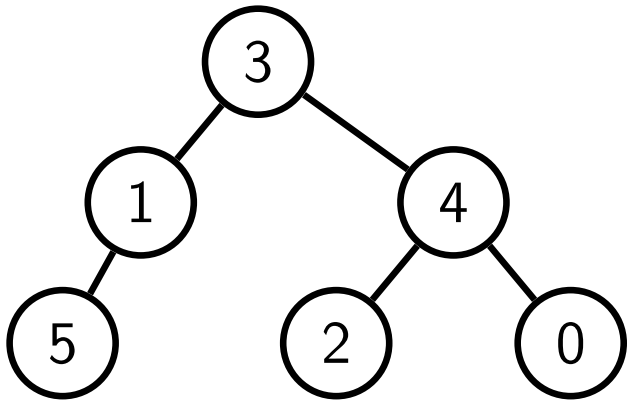
Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Une vision récursive: arbres binaires

- Plus simple: Les arbres binaires.



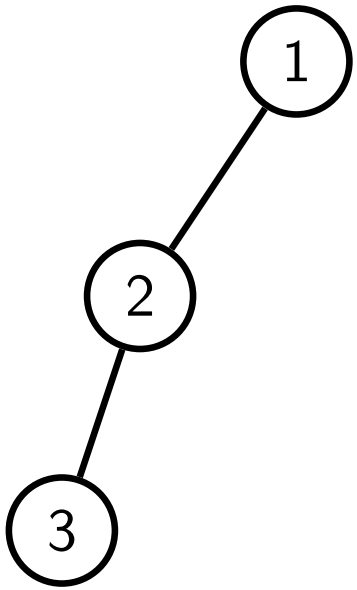
- Un arbre binaire est
  - soit vide
  - soit l'union disjointe d'un sommet, appelé *sa racine*, d'un arbre binaire, appelé *sous-arbre gauche*, et d'un arbre binaire, appelé *sous-arbre droit*.

- Autrement dit, un arbre binaire d'entiers est solution de l'équation:

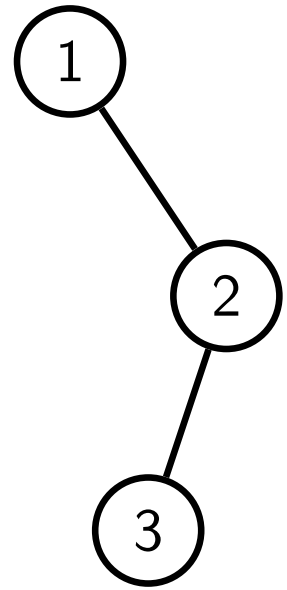
$$A = \textbf{null} \uplus (A \times \textbf{int} \times A)$$

Attention: Arbre binaire $\neq$ Arbre d'arité ≤ 2.

- Un arbre *binaire* a tous ses sommets d'arité 0, 1 ou 2.
- Mais les concept d'arbre de degré ≤ 2 et d'arbre binaire sont différents.
- En fait,



et



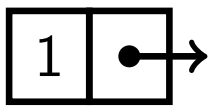
ne sont pas les même arbres binaires, car

$$(((\emptyset, 3, \emptyset), 2, \emptyset), 1, \emptyset) \neq (\emptyset, 1, ((\emptyset, 3, \emptyset), 2, \emptyset))$$

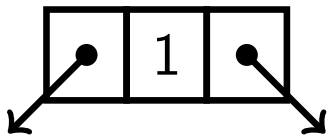
Définir un arbre binaire en JAVA

- Slogan:

Liste:



Arbre binaire:

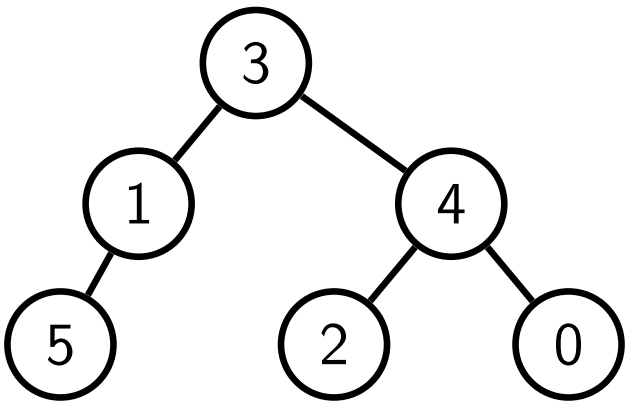


- On code un arbre vide par la référence **null**, et un arbre non-vide par une instance de la classe `Arbre`.

```
class Arbre {
    int val; Arbre gauche, droite;
    Arbre (Arbre gauche, int val, Arbre droite) {
        this.gauche = gauche;
        this.val = val;
        this.droite = droite; }}
```

Construire un arbre binaire

```
new Arbre (
   new Arbre (
      new Arbre (null,5,null),
      1,
      null),
   3,
   new Arbre (
      new Arbre (null,2,null),
      4,
      new Arbre (null,0,null)))
```



Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

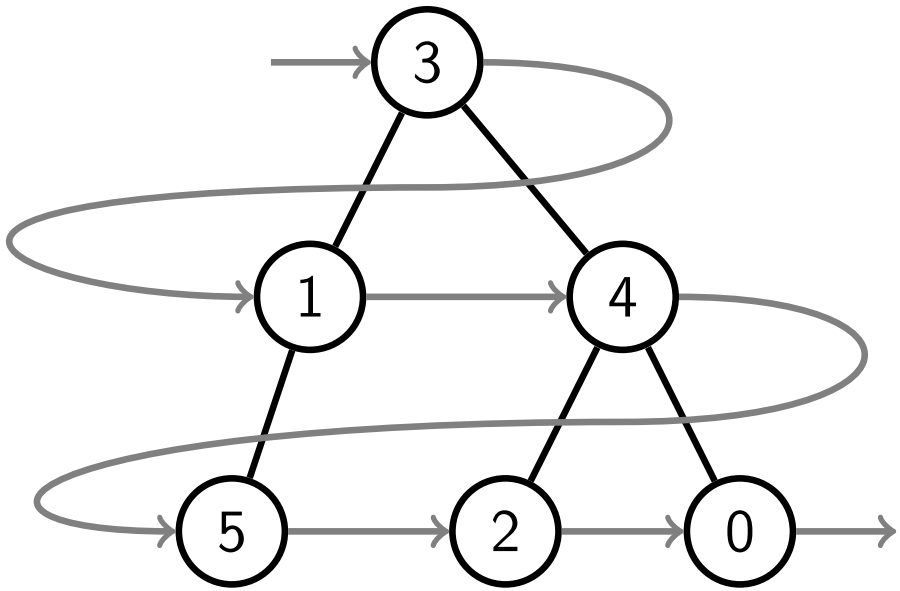
Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Parcourir un arbre

- Beaucoup d'algorithmes sur les arbres nécessitent de parcourir (traiter) tous les sommets
  - Exemple: Tester l'existence d'une valeur particulière dans un arbre.
  - Exemple: Afficher un arbre.
- Il existe une terminologie standard pour qualifier les parcours.
  - On peut parcourir de gauche à droite, ou de droite à gauche.
  - Une fois ce choix fait, on distingue les parcours
    - en largeur.
    - en profondeur:
      - préfixe,
      - infixe,
      - suffixe.

Parcourir en largeur

- Parcours en largeur d'abord.
  - on parcourt par distance croissante à la racine.



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 3, 1, 4, 5, 2, 0.
- C'est peut-être le parcours le plus naturel, mais c'est le plus délicat à programmer avec les arbres binaires.

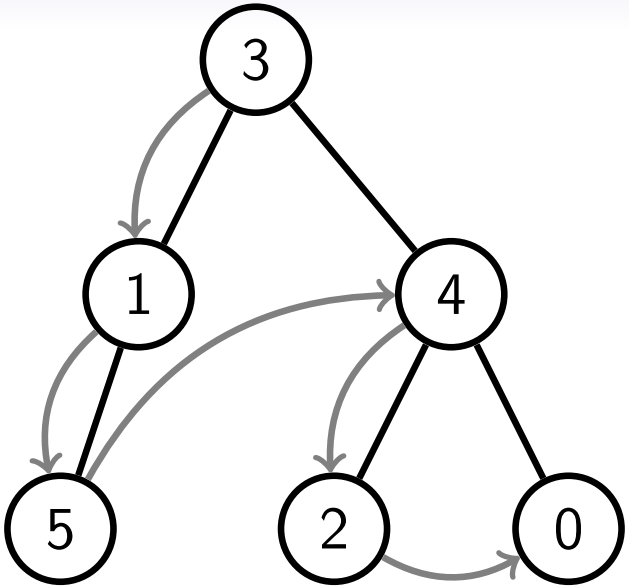
Parcourir en profondeur

- Parcours en profondeur d'abord:
  - on parcourt récursivement.

- Mais il reste trois possibilités

  - Préfixe: traiter la racine, parcourir le sous-arbre gauche, puis le sous-arbre droit.
  - Infixe: parcourir le sous-arbre gauche, traiter la racine, parcourir les sous-arbre droit.
  - Suffixe (appelé aussi postfixe): parcourir le sous-arbre gauche, le sous-arbre droit, puis traiter la racine.

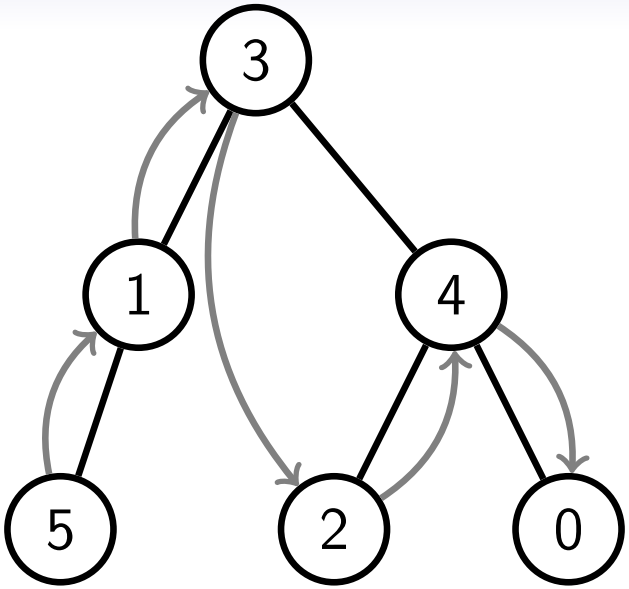
Parcours en profondeur: préfixe



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 3, 1, 5, 4, 2, 0.

```
static void Affiche(Arbre a) {
   if (a==null) return;
   System.out.println(a.val+" ");
   Affiche(a.gauche);
   Affiche(a.droite);
 }
```

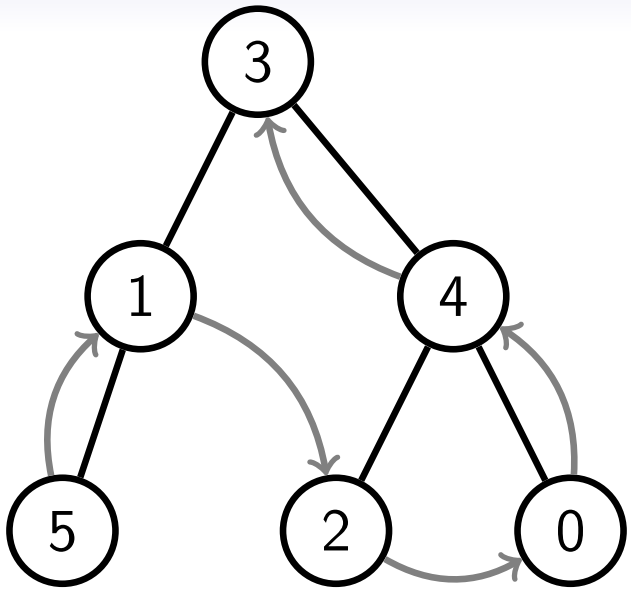
Parcours en profondeur: infixe



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 5, 1, 3, 2, 4, 0.

```
static void Affiche(Arbre a) {
   if (a==null) return;
   Affiche(a.gauche);
   System.out.println(a.val+" ");
   Affiche(a.droite);
 }
```

Parcours en profondeur: postfixe



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 5, 1, 2, 0, 4, 3.

```
static void Affiche(Arbre a) {
   if (a==null) return;
   Affiche(a.gauche);
   Affiche(a.droite);
   System.out.println(a.val+" ");
 }
```

De l'itératif? Parcourir en largeur?

- Voici un algorithme générique itératif de parcours d'arbre.
  ParcoursGenerique(Arbre a, racine s):
  - $L := \{s\}$
  - $B := \{s.gauche, s.droite\}$
  - Tant que $B \neq \emptyset$
    - choisir u dans B
    - $L := L \cup \{u\}$.
    - $B := B - \{u\}$.
    - $B := B \cup (\{u.gauche, u.droite\} - L)$
- Si B est implémenté
  - par une pile, on parcourt en profondeur (préfixe).
  - par une file, on parcourt en largeur.

Aujourd'hui

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

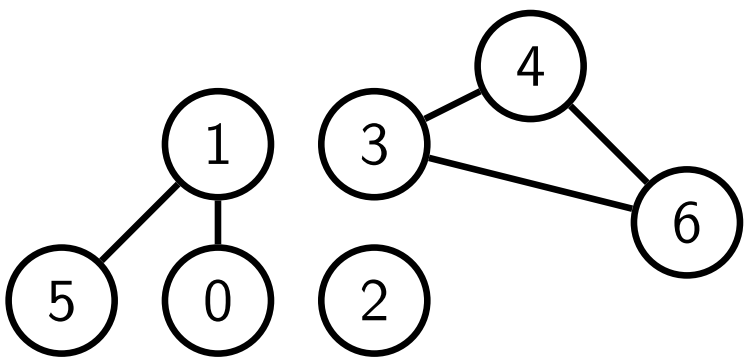
Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Représentation des graphes: matrice d'adjacence

- Matrice d'ajacence. Pour $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, on représente G par une matrice M $n \times n$.

$$M_{i,j} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

  - Si graphe valué: $M_{i,j} = v(i,j)$.



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

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

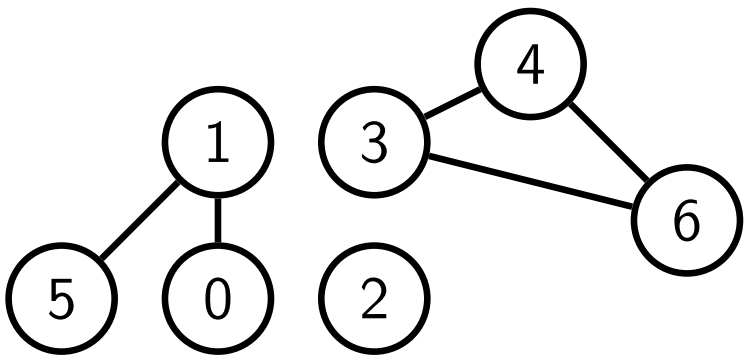
Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Représentation des graphes: liste de successeurs

- On associe à chaque sommet i, la liste des sommets j tels que $(i, j) \in E$.



- $L[0] = (1)$
- $L[1] = (0, 5)$
- $L[2] = ()$
- $L[3] = (4, 6)$
- $L[4] = (3, 6)$
- $L[5] = (1)$
- $L[6] = (3, 4)$

Meilleure représentation?

- Matrice: mémoire $O(n^2)$ (mieux pour graphes denses).
- Listes: mémoire $O(n + m)$ (mieux pour graphes creux).

- où n nombre de sommets, m nombre d'arêtes.

- Le mieux?: cela dépend du contexte.

Aujourd'hui

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Parcours générique

ParcoursGenerique(G,s):

- $L := \{s\}$
- $B := \mathit{Voisins}(s)$
- Tant que $B \neq \emptyset$
  - choisir u dans B
  - $L := L \cup \{u\}$.
  - $B := B - \{u\}$.
  - $B := B \cup (\mathit{Voisins}(u) - L)$

Application 1: numérotation

ParcoursNumerotation(G,s):

- $num := 0$
- $numero[s] := num$; $num := num + 1$.
- $L := \{s\}$
- $B := Voisins(s)$
- Tant que $B \neq \emptyset$
  - choisir u dans B
  - $numero[s] := num$; $num := num + 1$.
  - $L := L \cup \{u\}$.
  - $B := B - \{u\}$.
  - $B := B \cup (Voisins(u) - L)$

Application 2: composantes connexes

UneComposante(G,s):

- $L := \{s\}$
- $B := Voisins(s)$
- Tant que $B \neq \emptyset$
  - choisir u dans B
  - $L := L \cup \{u\}$.
  - $B := B - \{u\}$.
  - $B := B \cup (Voisins(u) - L)$
- Retourner L

ComposantesConnexes(G)

- Tant que $V \neq \emptyset$
  - Choisir s dans V
  - $L := UneComposante(G, s)$
  - Afficher L
  - $V := V - L$

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

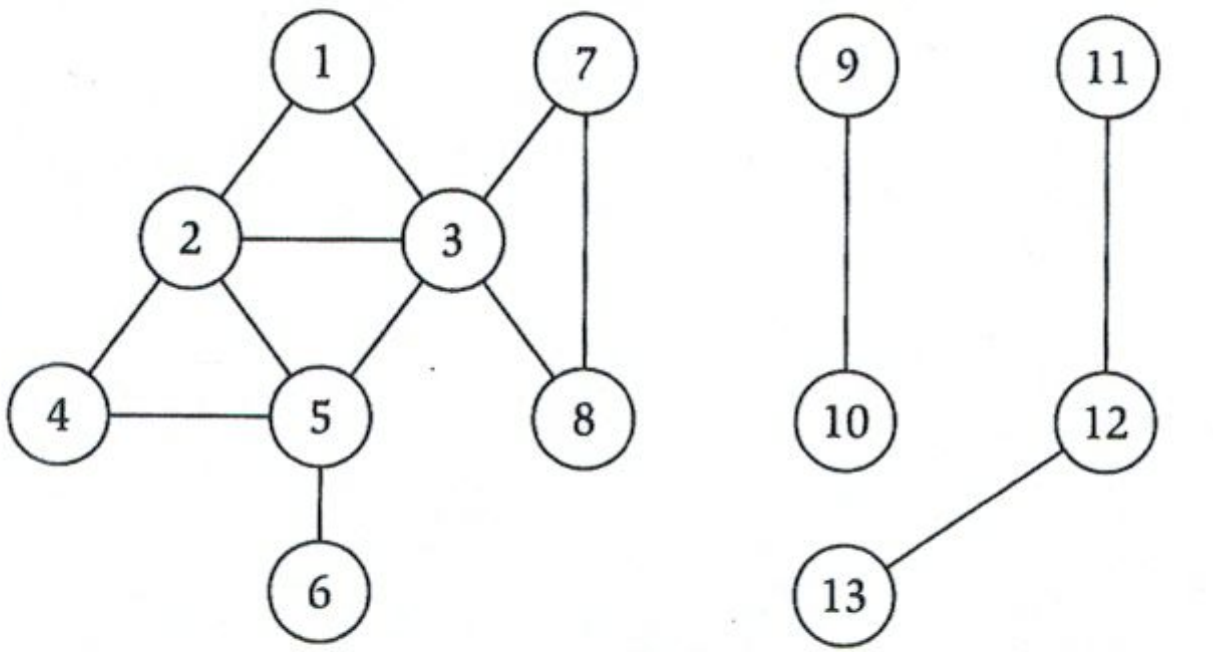
Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

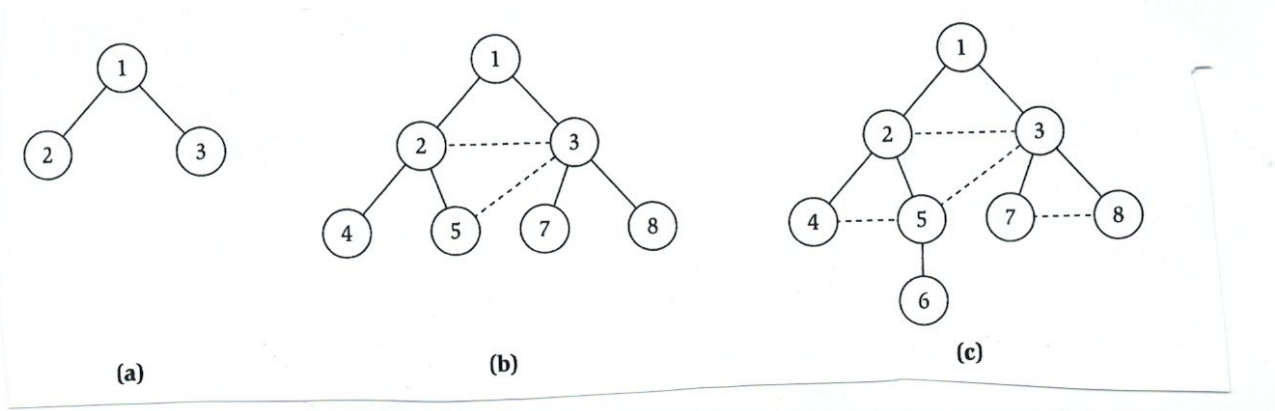
Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Parcours en largeur BFS

- B est implémenté par une file (FIFO = First In First Out).
- On explore les sommets par niveau: le niveau k correspond aux sommets à distance k du sommet s.
- Exemple:



BFS sur l'exemple précédent: (a), (b) et (c) montre les niveaux successifs qui sont considérés. Les sommets sont visités dans l'ordre $1, 2, 3, 4, 5, 7, 8, 6$.



- Pour chaque $k \geq 1$, le niveau k correspond aux sommets à distance exactement k de s.
- Il y a un chemin de s vers t si et seulement si t apparaît dans un certain niveau. Si t apparaît au niveau k, alors t est à distance k de s.

- Soit x et y deux sommets aux niveaux L_i et L_j avec $(x, y) \in E$. Alors i et j diffèrent d'au plus 1.

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

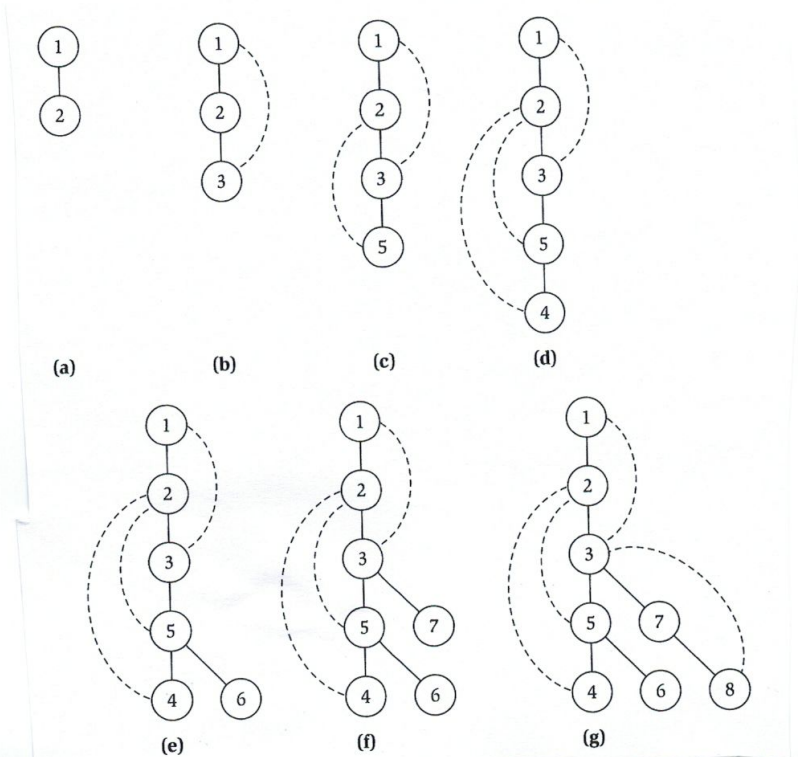
Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Parcours en profondeur DFS

DFS(s)

- Marquer s comme "exploré" et ajouter s à L.
- Pour chaque arête $(s, v) \in E$
  - si v n'est pas marqué "exploré" alors
    - appeler récursivement DFS(v)

Sur le graphe précédent, les sommets sont visités cette fois dans l'ordre $1, 2, 3, 5, 4, 6, 7, 8$. Le dessin plus bas montre les sommets dans l'ordre où ils sont découverts.



Si on appelle T l'arbre couvrant produit (celui qui contient "l'histoire du parcours": on décrète que v a u comme père si l'arête (u, v) a permis de découvrir v), on a:

- Pour chaque appel récursif DFS(u), tous les sommets qui sont marqués "explorés" entre l'invocation de l'appel et son retour sont des descendants de u dans l'arbre T produit.
- Soit T un arbre DFS, et x et y deux sommets dans T, avec $(x, y) \in E$ qui n'est pas une arête de T. Alors soit x est un ancêtre de y, soit le contraire.

Implémenter BFS(s)

- $Discovered[s] := true$; $Discovered[v] := false$ pour $v \neq s$.
- $L[0] = \{s\}$; i:=0 // Layer counter
- Set $T = \emptyset$ // Arbre
- Tant que $L[i] \neq \emptyset$
  - $L[i+1] := \emptyset$
  - Pour chaque $u \in L[i]$
    - Pour chaque arête $(u, v) \in E$
    - Si $Discovered[v] == false$ alors
    - Discovered[v]:=true;
    - Ajouter (u, v) à l'arbre T $(\pi(v) = u)$.
    - Ajouter v à $L[i+1]$
    - Fin si
  - Fin pour
  - i:=i+1
- Fin tant que

Temps $O(n+m)$, où n est le nombre de sommets, et m le nombre d'arêtes avec une représentation par liste de successeurs.

Implémenter DFS(s)

- $Discovered[s] := true$; $Discovered[v] := false$ pour $v \neq s$.
- Créer S comme la pile contenant uniquement l'élément s
- Tant que $S \neq \emptyset$
  - Prendre le sommet u de S
  - Si $Discovered[u] == false$ alors
    - $Discovered[u] := true$
    - Pour chaque arête $(u, v) \in E$
    - Ajouter u à la pile S
    - Fin pour
  - Fin si
- Fin while

Temps $O(n + m)$, où n est le nombre de sommets, et m le nombre d'arêtes, avec une représentation par liste de successeurs.

Aujourd'hui

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

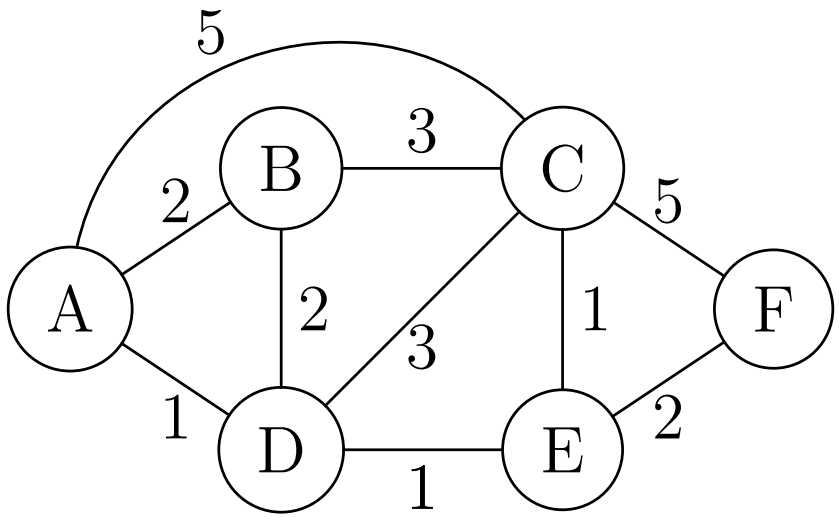
Algorithme de Bellman-Ford

Entrée : $\begin{cases} \text{Un graphe } G = (V, E) \text{ avec une source } s \\ \text{Une fonction de poids } w : E \to \mathbb{R}^+ \end{cases}$

Sortie: $\begin{cases} \text{Un vecteur distance } d \\ \text{Une fonction père } \pi : V \to V \end{cases}$

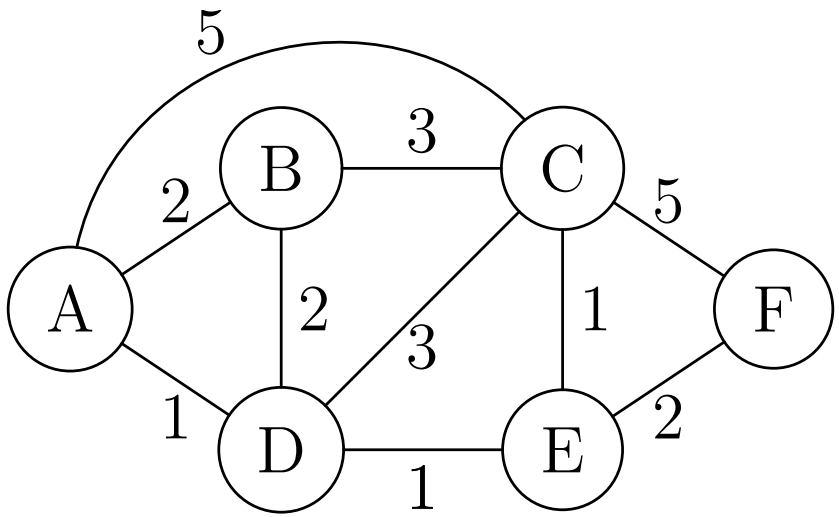
1. Initialisation des variables d et π
   - 1.1 $d[s] \leftarrow 0$; $\pi[s] \leftarrow s$
   - 1.2 Pour chaque sommet v de V faire $\begin{cases} d[v] \leftarrow \infty \\ \pi(v) \leftarrow NIL \end{cases}$
2. Calcul des distances et de l'arbre de plus court chemin
   - 2.1 Pour $i \leftarrow 1$ à $|V| - 1$ faire
     Pour chaque arête $(u, v) \in E$ faire
     Si $(d[v] > d[u] + w(u, v))$ alors $\begin{cases} d[v] \leftarrow d[u] + w(u, v) \\ \pi(v) \leftarrow u \end{cases}$
3. retourner d et π

Exemple



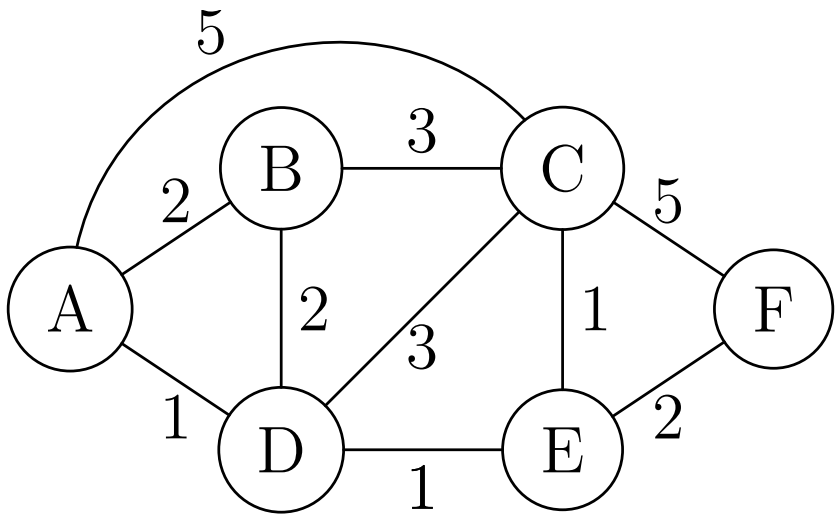
	A	B	C	D	E	F
init	$(0, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$

Exemple



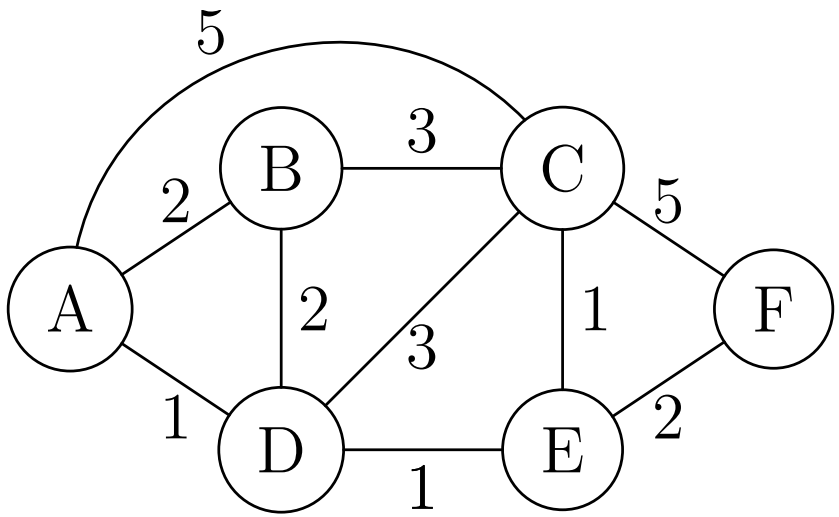
	A	B	C	D	E	F
init	$(0, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$

Exemple



	A	B	C	D	E	F
init	$(0, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(10, C)$

Exemple



	A	B	C	D	E	F
init	$(0, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(10, C)$
3	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$

les couples correspondent à $(d(.), \pi(.))$

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Algorithme de Dijkstra

Entrée : $\begin{cases} \text{Un graphe } G = (V, E) \text{ avec une source } s \\ \text{Une fonction de poids } w : E \to \mathbb{R}^+ \end{cases}$

Sortie: $\begin{cases} \text{Un vecteur distance } d \\ \text{Une fonction père } \pi : V \to V \end{cases}$

1. Initialisation de la source s
   - 1.1 $d[s] \leftarrow 0$; $\pi[s] \leftarrow s$
   - 1.2 Pour chaque sommet v de V faire $\begin{cases} \pi(v) \leftarrow NIL \\ d(v) \leftarrow \infty^+ \end{cases}$

Algorithme de Dijkstra

Entrée : $\begin{cases} \text{Un graphe } G = (V, E) \text{ avec une source } s \\ \text{Une fonction de poids } w : E \to \mathbb{R}^+ \end{cases}$

Sortie: $\begin{cases} \text{Un vecteur distance } d \\ \text{Une fonction père } \pi : V \to V \end{cases}$

1. Initialisation de la source s
   - 1.1 $d[s] \leftarrow 0$; $\pi[s] \leftarrow s$
   - 1.2 Pour chaque sommet v de V faire $\begin{cases} \pi(v) \leftarrow NIL \\ d(v) \leftarrow \infty^+ \end{cases}$

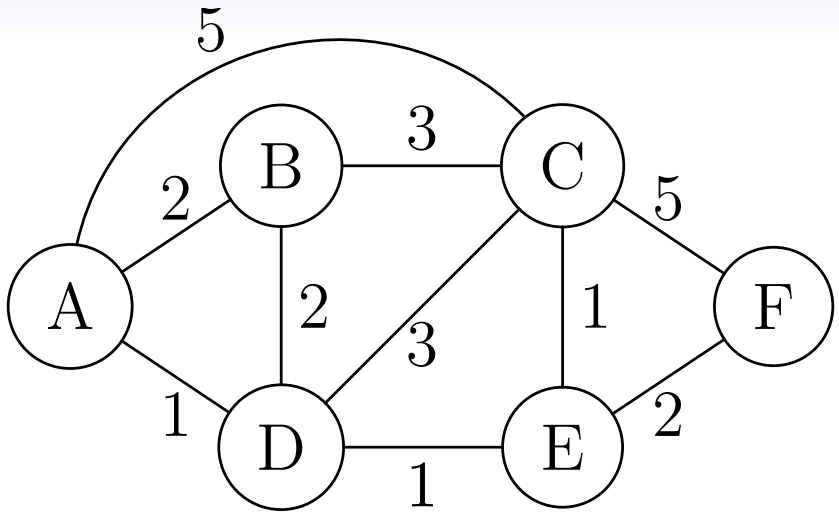
Algorithme de Dijkstra

Entrée : $\begin{cases} \text{Un graphe } G = (V, E) \text{ avec une source } s \\ \text{Une fonction de poids } w : E \to \mathbb{R}^+ \end{cases}$

Sortie: $\begin{cases} \text{Un vecteur distance } d \\ \text{Une fonction père } \pi : V \to V \end{cases}$

1. Initialisation de la source s
   - 1.1 $d[s] \leftarrow 0$; $\pi[s] \leftarrow s$
   - 1.2 Pour chaque sommet v de V faire $\begin{cases} \pi(v) \leftarrow NIL \\ d(v) \leftarrow \infty^+ \end{cases}$
2. $\mathcal{Q} \leftarrow V$; $\mathcal{S} \leftarrow \emptyset$;
3. Tant que $(\mathcal{Q} \neq \emptyset)$ faire
   - 3.1 $u \leftarrow$ Extraire-Le-Minimum$(\mathcal{Q}, d)$;
   - 3.2 $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$;
   - 3.3 Pour chaque sommet v voisin de u faire
     Si $(d[v] > d[u] + w(u, v))$ alors $\begin{cases} d[v] \leftarrow d[u] + w(u, v) \\ \pi(v) \leftarrow u \end{cases}$
4. retourner d et π

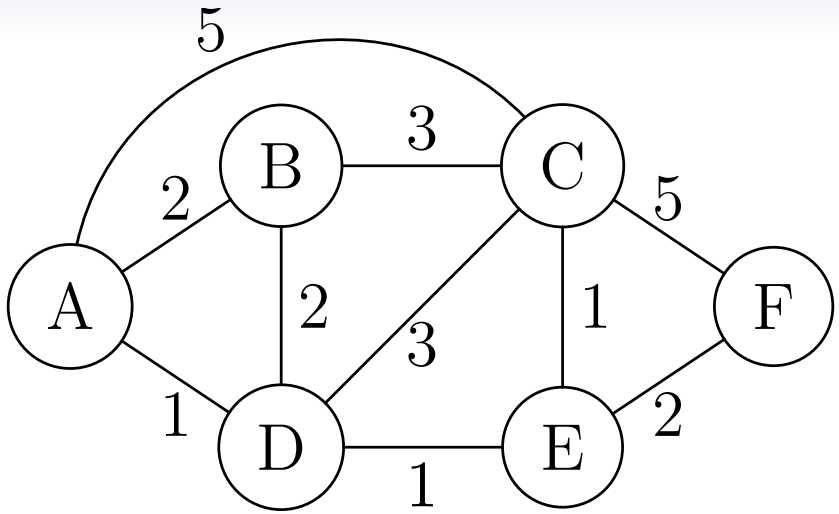
Exemple



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$

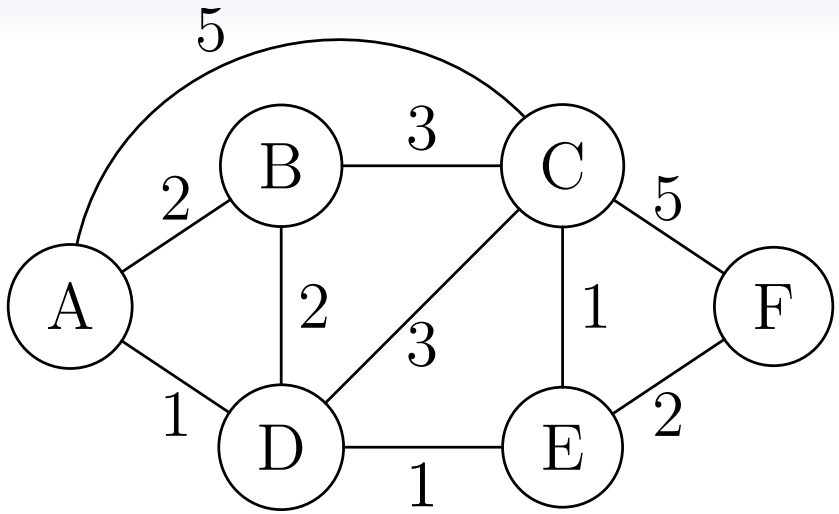
Exemple



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$

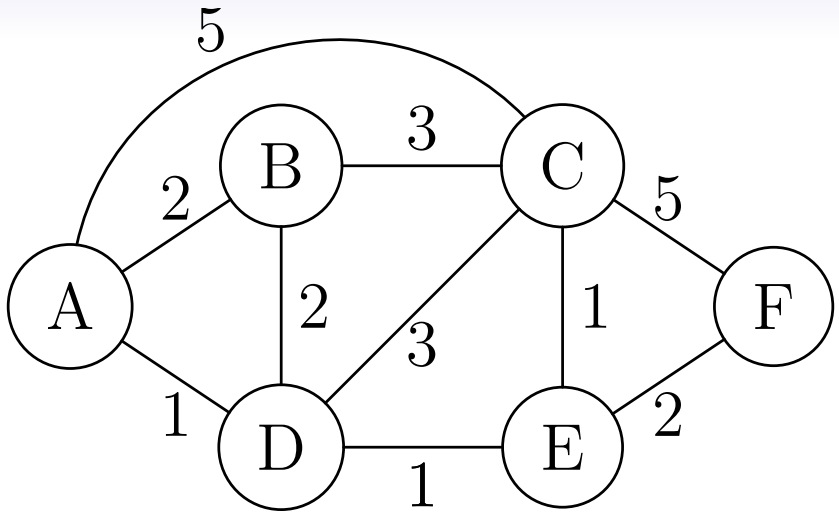
Exemple



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$

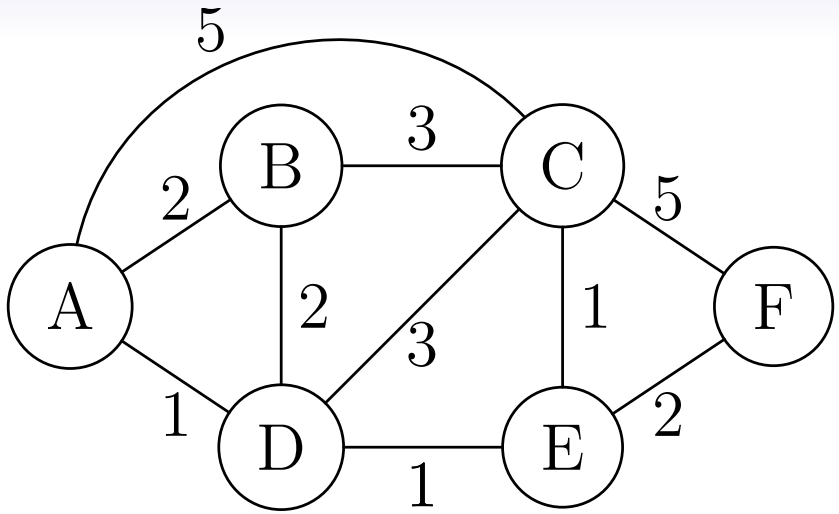
Exemple



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
3	$\{CEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$

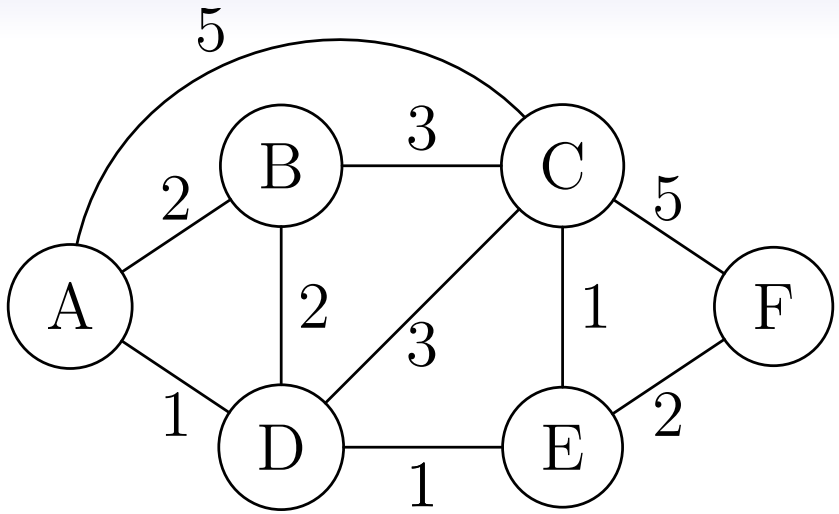
Exemple



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
3	$\{CEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
4	$\{CF\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$

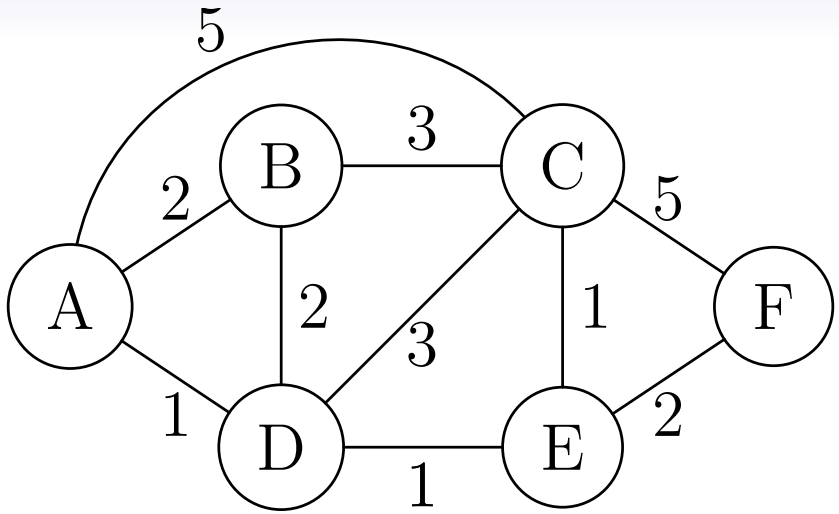
Exemple



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
3	$\{CEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
4	$\{CF\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$
5	$\{F\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$

Exemple



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
3	$\{CEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
4	$\{CF\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$
5	$\{F\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$
6	$\emptyset$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$

Complexité de l'algorithme: $O(|V|^2)$

Plan

Rappels: les graphes

Rappels: Les arbres
- Les arbres binaires
- Parcours d'arbres

Représentation des graphes
- Matrice d'adjacences
- Liste de successeurs

Parcours de graphes
- Parcours générique
- Parcours en largeur BFS
- Parcours en profondeur DFS

Calcul de distances
- Algorithme de Bellman-Ford
- Algorithme de Dijsktra
- Algorithme de Floyd Warshall

Plus court chemin dans un graphe

- Entrée : Un graphe orienté $G = (N, A)$ où chaque arc possède une longueur non-négative. Un des noeuds du graphe est appelé source.
- Problème: Trouver la longueur du plus court chemin entre toutes les paires de noeuds.
- On suppose que $N = \{1, \ldots n\}$.
- On suppose aussi que G est donné sous forme de matrice $L[1..n, 1..n]$ (on peut prendre $L[i, j] = \infty$ lorsqu'il n'y a pas d'arête entre i et j)
- Algorithme de Floyd: cet algorithme construite une matrice D qui donne la longueur du plus court chemin entre chaque paire de noeuds.

Principe

1. On initialise D à L
2. Après l'itération k, D donne la longueur du plus court chemin lorsque l'on utilise que des noeuds dans $\{1, .., k\}$ comme noeuds intermédiaires (ou éventuellement aucun noeud intermédiaire).

Définition: D_k est la matrice D après l'itération k.
Le résulat final recherché est D_n.

Récurrence

$$D_k[i,j] = \min(D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j])$$

Exercice: expliquer pourquoi.

Floyd(L[1..n,1..n])

Procédure Floyd(L[1..n,1..n])

- D:=L
- Pour $k := 1$ à n
  - Pour $i := 1$ à n
    - Pour $j := 1$ à n
    - $D[i,j] = min(D[i,j], D[i,k] + D[k,j])$
- Retourner D.

Temps dans $O(n^3)$.