

B-arbres et Graphes

Algorithmique et structures de données, 2022-2023

P. Albuquerque (B410), P. Künzli et O. Malaspinas (A401), ISC, HEPIA
2023-05-24

En partie inspirés des supports de cours de P. Albuquerque

Les B-arbres: suppression

Cas simplissime

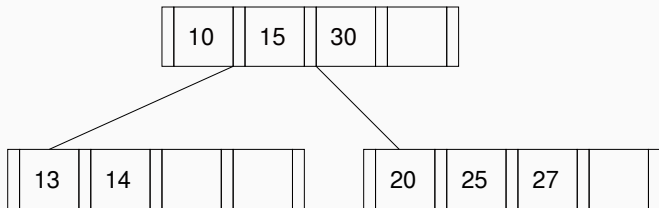


Figure 1: Suppression de 25.

Les B-arbres: suppression

Cas simplissime

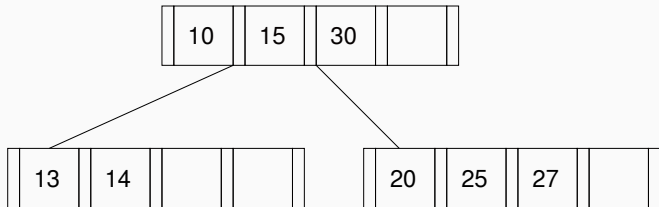


Figure 1: Suppression de 25.

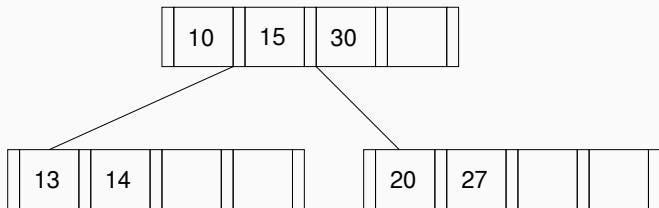


Figure 2: 25 supprimé, on décale juste 27.

Les B-arbres: suppression

Cas simple

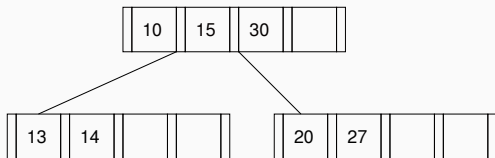


Figure 3: Suppression de 27.

Les B-arbres: suppression

Cas simple

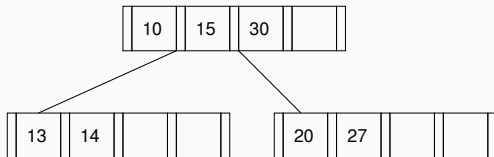


Figure 3: Suppression de 27.

- On retire 27, mais...
 - Chaque page doit avoir au moins 2 éléments.
 - On doit déplacer des éléments dans une autre feuille! Mais comment?

Les B-arbres: suppression

Cas simple

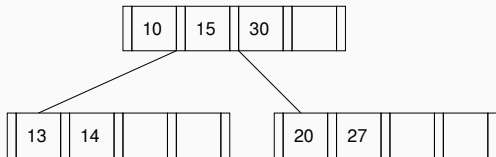


Figure 3: Suppression de 27.

- On retire 27, mais...
 - Chaque page doit avoir au moins 2 éléments.
 - On doit déplacer des éléments dans une autre feuille! Mais comment?

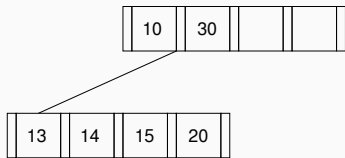


Figure 4: La médiane de la racine descend, fusion de 20 à gauche, et suppression à droite.

Les B-arbres: suppression

Cas moins simple

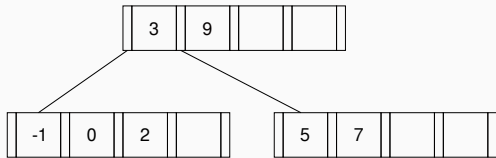


Figure 5: Suppression de 5.

Les B-arbres: suppression

Cas moins simple

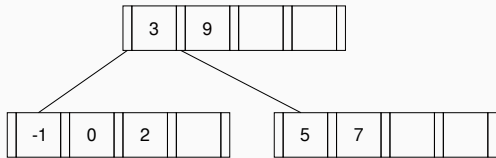


Figure 5: Suppression de 5.

- Un élément à droite, comment on fait?
 - Remonter 7, serait ok si racine, mais... c'est pas forcément.
 - On redistribue les feuilles.

Les B-arbres: suppression

Cas moins simple

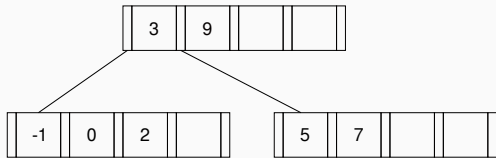


Figure 5: Suppression de 5.

- Un élément à droite, comment on fait?
 - Remonter 7, serait ok si racine, mais... c'est pas forcément.
 - On redistribue les feuilles.

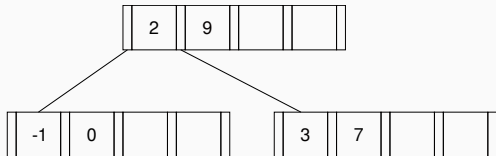


Figure 6: Descente de 3, remontée médiane des feuilles 2.

Les B-arbres: suppression

Cas ultra moins simple

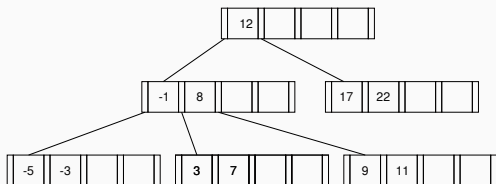


Figure 7: Suppression de 3.

Les B-arbres: suppression

Cas ultra moins simple

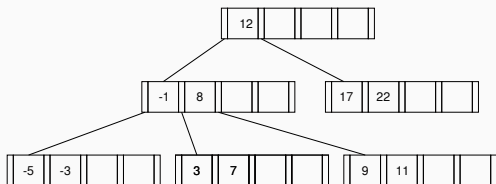


Figure 7: Suppression de 3.

- 7 seul:
 - Fusionner les feuilles et redistribuer, comment?

Les B-arbres: suppression

Cas ultra moins simple

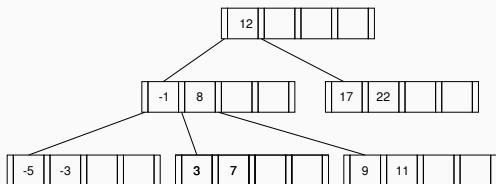


Figure 7: Suppression de 3.

- 7 seul:
 - Fusionner les feuilles et redistribuer, comment?

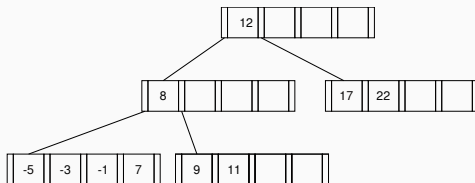


Figure 8: Descendre -1, déplacer 7 à gauche, et décaler les éléments de droite

Les B-arbres: suppression

Cas ultra moins simple

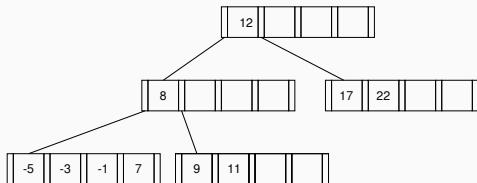


Figure 9: On a pas fini...

Les B-arbres: suppression

Cas ultra moins simple

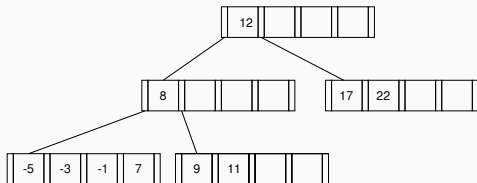


Figure 9: On a pas fini...

- 8 est seul, c'est plus un B-arbre :
 - Fusionner le niveau 2 et redistribuer, comment?

Les B-arbres: suppression

Cas ultra moins simple

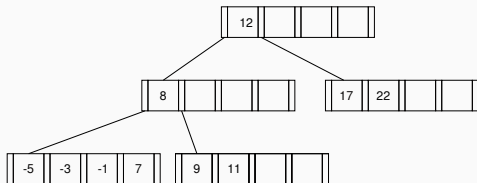


Figure 9: On a pas fini...

- 8 est seul, c'est plus un B-arbre :
 - Fusionner le niveau 2 et redistribuer, comment?

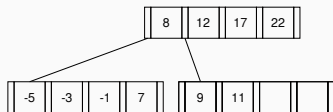


Figure 10: Fusionner 8, 17, 22 et descendre 12.

Les B-arbres: suppression

Cas ultra moins simple

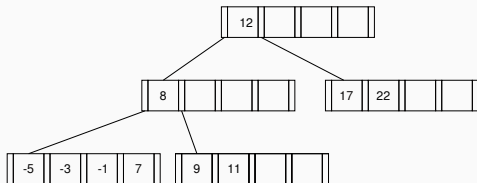


Figure 9: On a pas fini...

- 8 est seul, c'est plus un B-arbre :
 - Fusionner le niveau 2 et redistribuer, comment?

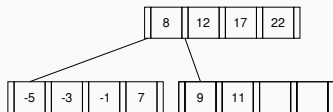


Figure 10: Fusionner 8, 17, 22 et descendre 12.

- La profondeur a diminué de 1.

Algorithme pour les feuilles!

- Si la clé est supprimée d'une feuille:
 - Si on a toujours n (ordre de l'arbre) clés dans la feuille on décale simplement les clés.
 - Sinon on combine (récursivement) avec le nœud voisin et on descend la clé médiane.

Les B-arbres: suppression

Cas non-feuille!

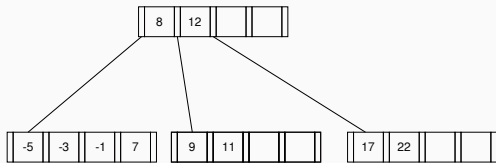


Figure 11: Suppression de 8.

Les B-arbres: suppression

Cas non-feuille!

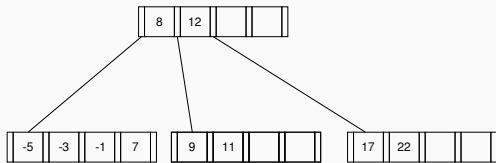


Figure 11: Suppression de 8.

- On sait comment effacer une valeur d'une feuille, donc?

Les B-arbres: suppression

Cas non-feuille!

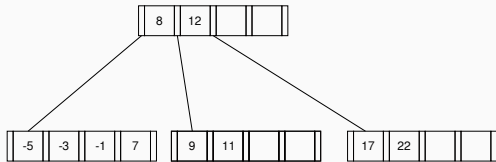


Figure 11: Suppression de 8.

- On sait comment effacer une valeur d'une feuille, donc?

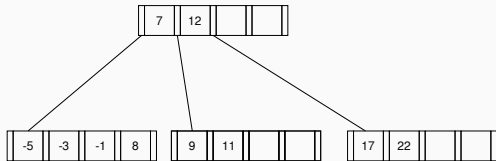


Figure 12: Échanger le 8 avec le plus grand du sous-arbre de gauche.

- Ensuite?

Les B-arbres: suppression

Cas non-feuille!

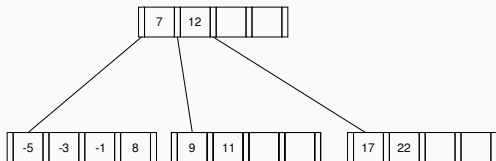


Figure 13: Suppression de 8.

Les B-arbres: suppression

Cas non-feuille!

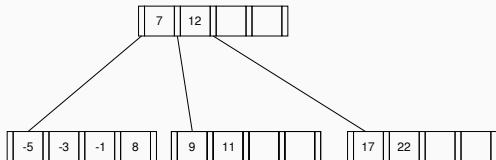


Figure 13: Suppression de 8.

- On sait comment effacer une valeur d'une feuille!

Les B-arbres: suppression

Cas non-feuille!

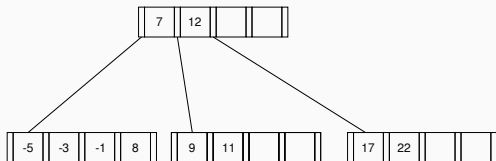


Figure 13: Suppression de 8.

- On sait comment effacer une valeur d'une feuille!

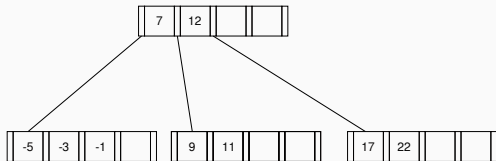


Figure 14: Yaka enlever le 8 de la feuille comme avant!

Algorithme pour les non-feuilles!

- Si la clé est supprimée d'une page qui n'est pas une feuille:
 - On échange la valeur avec la valeur de droite de la page de gauche
 - On supprime comme pour une feuille!

Et maintenant des exercices par millions!

Les graphes! Historique

Un mini-peu d'histoire...

L. Euler et les 7 ponts de Königsberg:

- Existe-t-il une promenade sympa, passant **une seule fois** par les 7 ponts et revenant au point de départ?

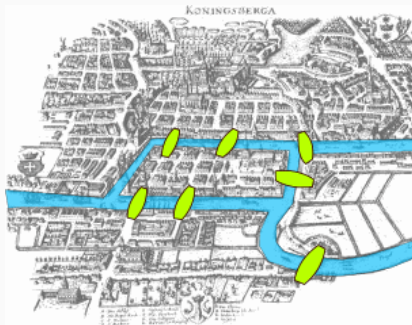


Figure 15: Les ponts c'est beau. Source: Wikipédia, <https://bit.ly/37h0yOG>

Les graphes! Historique

Un mini-peu d'histoire...

L. Euler et les 7 ponts de Königsberg:

- Existe-t-il une promenade sympa, passant **une seule fois** par les 7 ponts et revenant au point de départ?

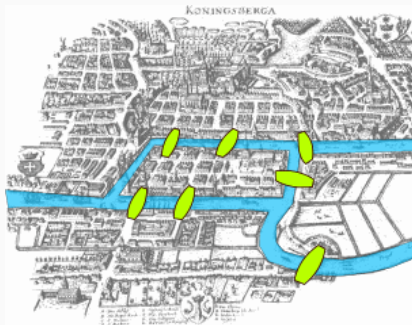


Figure 15: Les ponts c'est beau. Source: Wikipédia, <https://bit.ly/37h0yOG>

- Réponse: ben non!

Réseau social

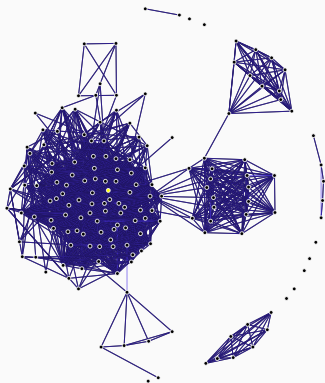


Figure 16: Source, Wikipedia: <https://bit.ly/3kG6cgo>

- Chaque sommet est un individu.
- Chaque trait une relation d'amitié.
- Miam, Miam, Facebook.

Moteurs de recherche

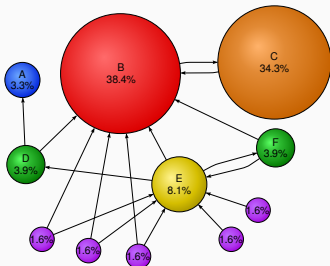


Figure 17: Source, Wikipedia: <https://bit.ly/3kG6cgo>

- Sommet est un site.
- Liens sortants;
- Liens entrants;
- Notion d'importance d'un site: combien de liens entrants, pondérés par l'importance du site.
- Miam, Miam, Google (PageRank).

Introduction

Définition, plus ou moins

- Un graphe est un ensemble de sommets, reliés par des lignes ou des flèches.

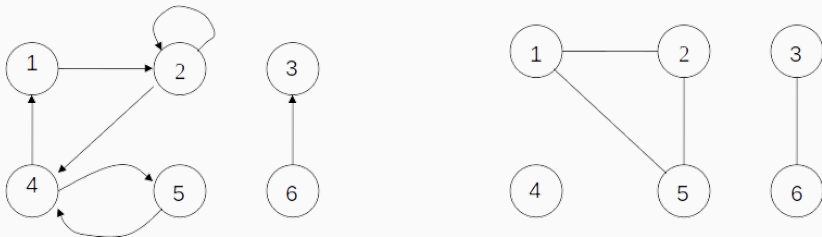


Figure 18: Deux exemples de graphes.

- Des sommets (numérotés 1 à 6);
- Connectés ou pas par des traits ou des flèches!

Définitions

- Un **graphe** $G(V, E)$ est constitué
 - V : un ensemble de sommets;
 - E : un ensemble d'arêtes.
- Une **arête** relie une **paire** de sommets de V .

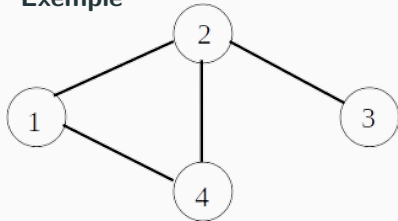
Remarques

- Il y a **au plus** une arête E par paire de sommets de V .
- La **complexité** d'un algorithme dans un graphe se mesure en terme de $|E|$ et $|V|$, le nombre d'éléments de E et V respectivement.

Notations

- Une arête d'un graphe **non-orienté** est représentée par une paire **non-ordonnée** $(u, v) = (v, u)$, avec $u, v \in V$.
- Les arêtes ne sont pas orientées dans un graphe non-orienté (elles sont bi-directionnelles, peuvent être parcourues dans n'importe quel ordre).

Exemple



Que valent V , $|V|$, E , et $|E|$?

Figure 19: Un graphe non-orienté.

Notations

- Une arête d'un graphe **non-orienté** est représentée par une paire **non-ordonnée** $(u, v) = (v, u)$, avec $u, v \in V$.
- Les arêtes ne sont pas orientées dans un graphe non-orienté (elles sont bi-directionnelles, peuvent être parcourues dans n'importe quel ordre).

Exemple

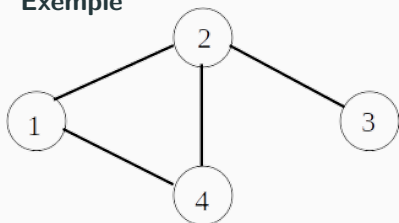


Figure 19: Un graphe non-orienté.

Que valent V , $|V|$, E , et $|E|$?

$$V = \{1, 2, 3, 4\},$$

$$|V| = 4,$$

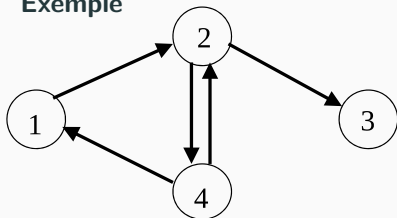
$$E = \{(1, 2), (2, 3), (2, 4), (4, 1)\},$$

$$|E| = 4.$$

Notations

- Une arête d'un graphe **orienté** est représentée par une paire **ordonnée** $(u, v) \neq (v, u)$, avec $u, v \in V$.
- Les arêtes sont orientées dans un graphe orienté (étonnant non?).

Exemple



Que valent V , $|V|$, E , et $|E|$?

Figure 20: Un graphe orienté.

Notations

- Une arête d'un graphe **orienté** est représentée par une paire **ordonnée** $(u, v) \neq (v, u)$, avec $u, v \in V$.
- Les arêtes sont orientées dans un graphe orienté (étonnant non?).

Exemple

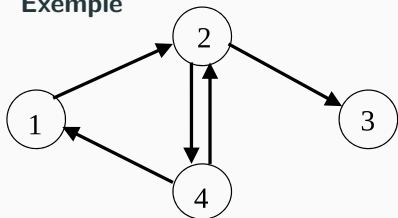


Figure 20: Un graphe orienté.

Que valent V , $|V|$, E , et $|E|$?

$$V = \{1, 2, 3, 4\},$$

$$|V| = 4,$$

$$E = \{(1, 2), (2, 3), (2, 4), (4, 1), (4, 2)\}$$

$$|E| = 5.$$

Généralités

Définition

- Le sommet v est **adjacent** au sommet u , si et seulement si $(u, v) \in E$;
- Si un graphe non-orienté contient une arête (u, v) , v est adjacent à u et u est adjacent à v .

Exemple

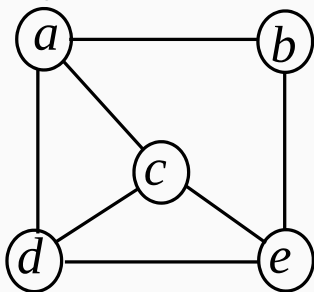


Figure 21: Sommet a adjacent à c , c adjacent à a .

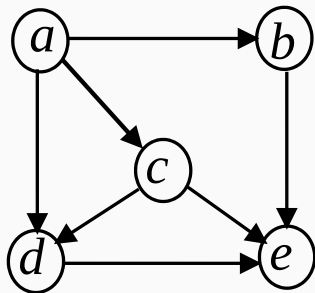


Figure 22: Sommet a adjacent à c .

Définition

- Un **graphe pondéré** ou **valué** est un graphe dont chaque arête a un poids associé, habituellement donné par une fonction de pondération $w : E \rightarrow \mathbb{R}$.

Exemples

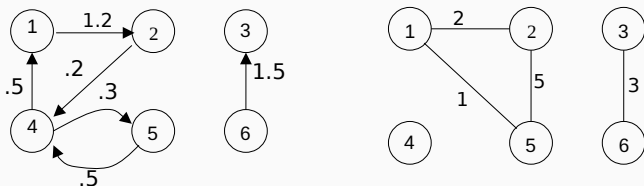


Figure 23: Graphe pondéré orienté (gauche) et non-orienté (droite).

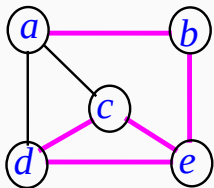
Définition

- Dans un graphe $G(V, E)$, une **chaîne** reliant un sommet u à un sommet v est une suite d'arêtes entre les sommets, w_0, w_1, \dots, w_k , telles que

$$(w_i, w_{i+1}) \in E, \quad u = w_0, \quad v = w_k, \quad \text{pour } 0 \leq i < k,$$

avec k la longueur de la chaîne (le nombre d'arêtes du chemin).

Exemples



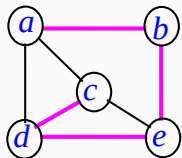
abedce est une chaîne
cdeb est une chaîne
bca n'est pas une chaîne

Figure 24: Illustration d'une chaîne, ou pas chaîne dans un graphe.

Définition

- Une **chaîne élémentaire** est une chaîne dont tous les sommets sont distincts, sauf les extrémités qui peuvent être égales

Exemples



abedc est une chaîne élémentaire
cdec est une chaîne élémentaire
abedce n'est pas une chaîne élémentaire

Figure 25: Illustration d'une chaîne élémentaire.

Définition

- Une **boucle** est une arête (v, v) d'un sommet vers lui-même.

Exemples

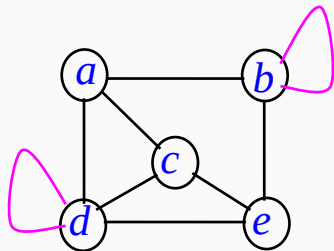


Figure 26: Illustration d'une boucle.

Définition

- Un graphe non-orienté est dit **connexe**, s'il existe un chemin reliant n'importe quelle paire de sommets distincts.

Exemples

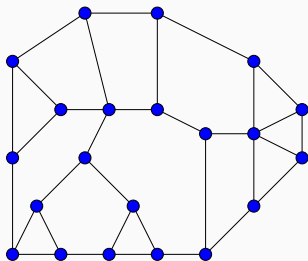


Figure 27: Graphe connexe. Source, Wikipédia: <https://bit.ly/3yiUzUv>

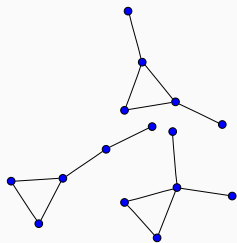


Figure 28: Graphe non-connexe avec composantes connexes. Source, Wikipédia: <https://bit.ly/3KJB76d>

Définition

- Un graphe orienté est dit **fortement connexe**, s'il existe un chemin reliant n'importe quelle paire de sommets distincts.

Exemples

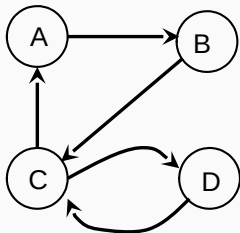


Figure 29: Graphe fortement connexe.

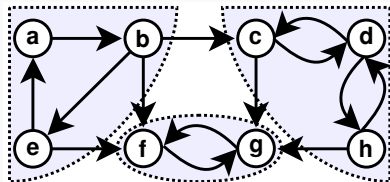
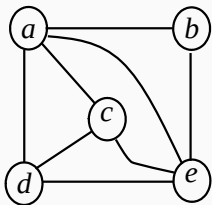


Figure 30: Composantes fortement connexes. Source, Wikipédia:
<https://bit.ly/3w5PL2l>

Définition

- Un **cycle** dans un graphe *non-orienté* est une chaîne de longueur ≥ 3 telle que le 1er sommet de la chaîne est le même que le dernier, et dont les arêtes sont distinctes.
- Pour un graphe *orienté* on parle de **circuit**.
- Un graphe sans cycles est dit **acyclique**.

Exemples



aba n'est pas un cycle
abedceda n'est pas un cycle
abedcea est un cycle, mais pas élémentaire
abea est un cycle élémentaire

Figure 31: Illustration de cycles, ou pas.

Question de la mort

- Qu'est-ce qu'un graphe connexe acyclique?

Question de la mort

- Qu'est-ce qu'un graphe connexe acyclique?
- Un arbre!

Représentations

- La complexité des algorithmes sur les graphes s'expriment en fonction du nombre de sommets V , et du nombre d'arêtes E :
 - Si $|E| \sim |V|^2$, on dit que le graphe est **dense**.
 - Si $|E| \sim |V|$, on dit que le graphe est **peu dense**.
- Selon qu'on considère des graphes denses ou peu denses, différentes structure de données peuvent être envisagées.

Question

- Comment peut-on représenter un graphe informatiquement? Des idées (réflexion de quelques minutes)?

Représentations

- La complexité des algorithmes sur les graphes s'expriment en fonction du nombre de sommets V , et du nombre d'arêtes E :
 - Si $|E| \sim |V|^2$, on dit que le graphe est **dense**.
 - Si $|E| \sim |V|$, on dit que le graphe est **peu dense**.
- Selon qu'on considère des graphes denses ou peu denses, différentes structure de données peuvent être envisagées.

Question

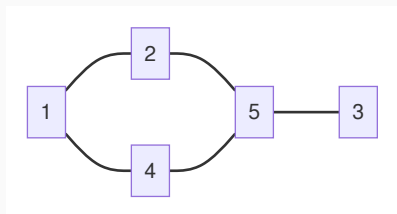
- Comment peut-on représenter un graphe informatiquement? Des idées (réflexion de quelques minutes)?
- Matrice/liste d'adjacence.

Matrice d'adjacence

- Soit le graphe $G(V, E)$, avec $V = \{1, 2, 3, \dots, n\}$;
- On peut représenter un graphe par une **matrice d'adjacence**, A , de dimension $n \times n$ définie par

$$A_{ij} = \begin{cases} 1 & \text{si } i, j \in E, \\ 0 & \text{sinon.} \end{cases}$$

Exemple



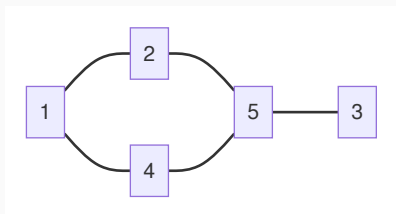
Quelle matrice d'adjacence?

Matrice d'adjacence

- Soit le graphe $G(V, E)$, avec $V = \{1, 2, 3, \dots, n\}$;
- On peut représenter un graphe par une **matrice d'adjacence**, A , de dimension $n \times n$ définie par

$$A_{ij} = \begin{cases} 1 & \text{si } i, j \in E, \\ 0 & \text{sinon.} \end{cases}$$

Exemple



Quelle matrice d'adjacence?

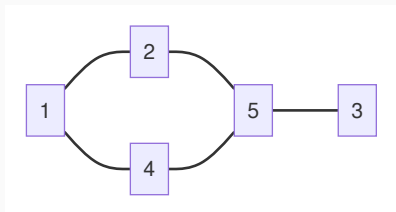
	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	1	0	0	0	1
5	0	1	1	1	0

Matrice d'adjacence

Remarques

- Zéro sur la diagonale.
- La matrice d'un graphe non-orienté est symétrique

$$A_{ij} = A_{ji}, \forall i, j \in [1, n]$$

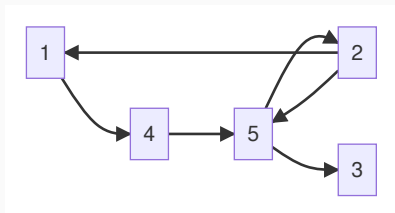


	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	1	0	0	0	1
5	0	1	1	1	0

Matrice d'adjacence

- Pour un graphe orienté (digraphe)

Exemple

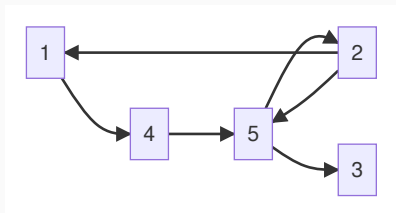


Quelle matrice d'adjacence?

Matrice d'adjacence

- Pour un graphe orienté (digraphe)

Exemple



Quelle matrice d'adjacence?

	1	2	3	4	5
1	0	0	0	1	0
2	1	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	1
5	0	1	1	0	0

- La matrice d'adjacence n'est plus forcément symétrique

$$A_{ij} \neq A_{ji}.$$

- Quel est l'espace nécessaire pour stocker une matrice d'adjacence pour un graphe orienté?

- Quel est l'espace nécessaire pour stocker une matrice d'adjacence pour un graphe orienté?
- $\mathcal{O}(|V|^2)$.
- Quel est l'espace nécessaire pour stocker une matrice d'adjacence pour un graphe non-orienté?

- Quel est l'espace nécessaire pour stocker une matrice d'adjacence pour un graphe orienté?
- $\mathcal{O}(|V|^2)$.
- Quel est l'espace nécessaire pour stocker une matrice d'adjacence pour un graphe non-orienté?
- $\mathcal{O}(|V| - 1)|V|/2$.

Considérations d'efficacité

- Dans quel type de graphes la matrice d'adjacence est utile?

Considérations d'efficacité

- Dans quel type de graphes la matrice d'adjacence est utile?
- Dans les graphes denses.
- Pourquoi?

Considérations d'efficacité

- Dans quel type de graphes la matrice d'adjacence est utile?
- Dans les graphes denses.
- Pourquoi?
- Dans les graphes peu denses, la matrice d'adjacence est essentiellement composée de 0.

Remarque

- Dans la majorité des cas, les grands graphes sont peu denses.
- Comment représenter un graphe autrement?

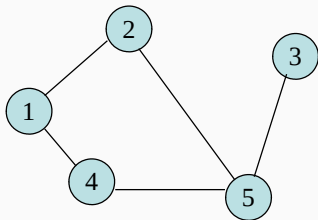
La liste d'adjacence (non-orienté)

- Pour chaque sommet $v \in V$, stocker les sommets adjacents à v -
- Quelle structure de données pour la liste d'adjacence?

La liste d'adjacence (non-orienté)

- Pour chaque sommet $v \in V$, stocker les sommets adjacents à v .
- Quelle structure de données pour la liste d'adjacence?
- Tableau de liste chaînée, vecteur (tableau dynamique), etc.

Exemple



Quelle liste d'adjacence?

Figure 32: Un graphe non-orienté.

La liste d'adjacence (non-orienté)

- Pour chaque sommet $v \in V$, stocker les sommets adjacents à v .
- Quelle structure de données pour la liste d'adjacence?
- Tableau de liste chaînée, vecteur (tableau dynamique), etc.

Exemple

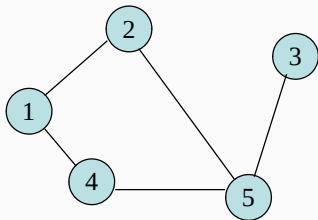


Figure 32: Un graphe non-orienté.

Quelle liste d'adjacence?

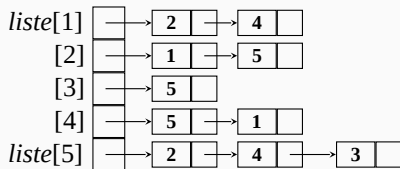
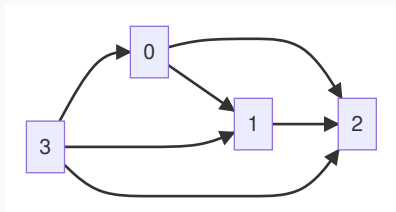


Figure 33: La liste d'adjacence.

La liste d'adjacence (orienté)

Quelle liste d'adjacence pour...

- Matrix (2min)



Stockage

- Quelle espace est nécessaire pour stocker une liste d'adjacence (en fonction de $|E|$ et $|V|$)?

Stockage

- Quelle espace est nécessaire pour stocker une liste d'adjacence (en fonction de $|E|$ et $|V|$)?

$$\mathcal{O}(|E|)$$

- Pour les graphes *non-orientés*: $\mathcal{O}(2|E|)$.
- Pour les graphes *orientés*: $\mathcal{O}(|E|)$.

Définition

- Le **degré** d'un sommet v , est le nombre d'arêtes incidentes du sommet (pour les graphes orientés on a un degré entrant ou sortant).
- Comment on retrouve le degré de chaque sommet avec la liste d'adjacence?

Stockage

- Quelle espace est nécessaire pour stocker une liste d'adjacence (en fonction de $|E|$ et $|V|$)?

$$\mathcal{O}(|E|)$$

- Pour les graphes *non-orientés*: $\mathcal{O}(2|E|)$.
- Pour les graphes *orientés*: $\mathcal{O}(|E|)$.

Définition

- Le **degré** d'un sommet v , est le nombre d'arêtes incidentes du sommet (pour les graphes orientés on a un degré entrant ou sortant).
- Comment on retrouve le degré de chaque sommet avec la liste d'adjacence?
- C'est la longueur de la liste chaînée.

- Beaucoup d'applications nécessitent de parcourir des graphes:
 - Trouver un chemin d'un sommet à un autre;
 - Trouver si le graphe est connexe;
- Il existe *deux* parcours principaux:
 - en largeur (Breadth-First Search);
 - en profondeur (Depth-First Search).
- Ces parcours créent *un arbre* au fil de l'exploration (si le graphe est non-connexe cela crée une *forêt*, un ensemble d'arbres).

Illustration: parcours en largeur

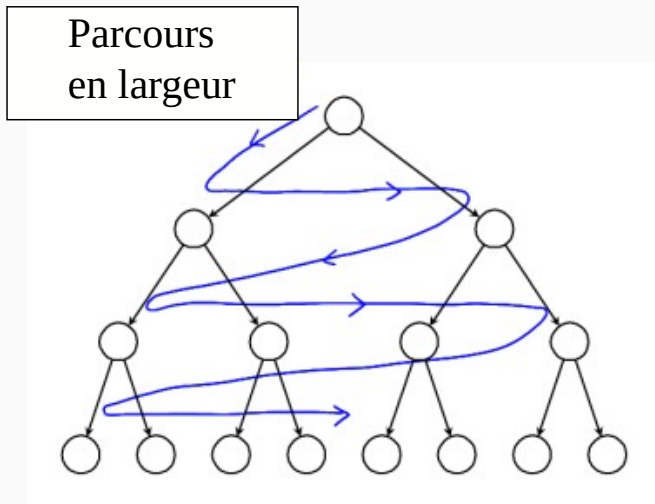


Figure 34: Le parcours en largeur.

Exemple

Étape par étape (blanc non-visité)

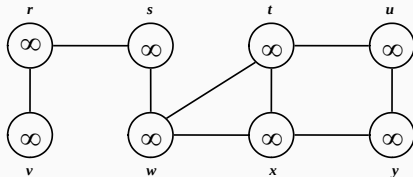


Figure 35: Initialisation.

Étape par étape (gris visité)

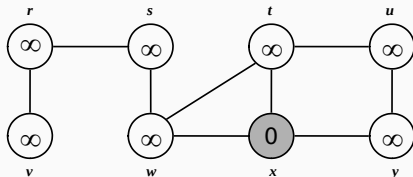


Figure 36: On commence en *x*.

Exemple

Étape par étape

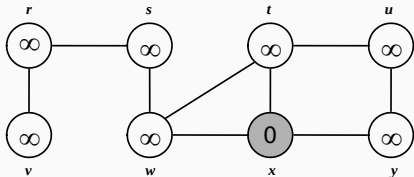


Figure 37: On commence en *x*.

Étape par étape (vert à visiter)

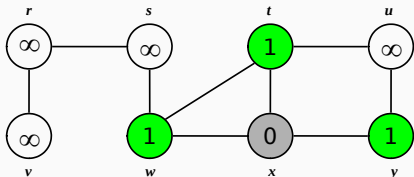


Figure 38: Visiter *w*, *t*, *y*.

Exemple

Étape par étape

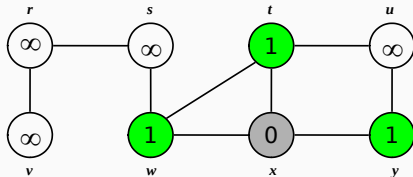


Figure 39: Visiter w, t, y.

Étape par étape

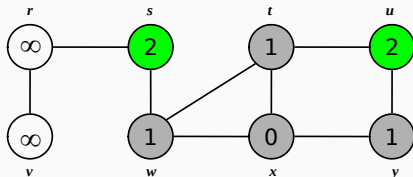


Figure 40: w, t, y visités. u, s à visiter.

Exemple

Étape par étape

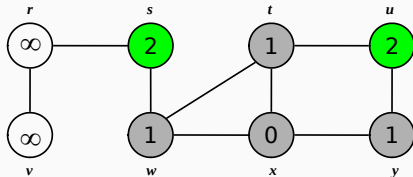


Figure 41: w , t , y visités. u , s à visiter.

Étape par étape

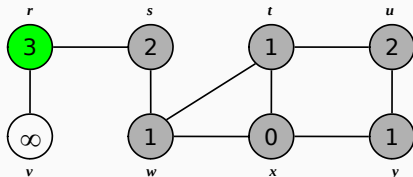


Figure 42: u , s , visités. r à visiter.

Exemple

Étape par étape

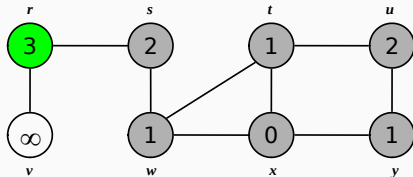


Figure 43: u, s, visités. r à visiter.

Étape par étape

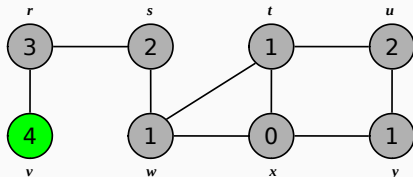


Figure 44: r visité. v à visiter.

Exemple

Étape par étape

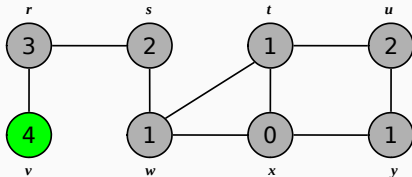


Figure 45: r visité. v à visiter.

Étape par étape

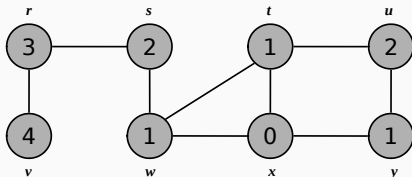
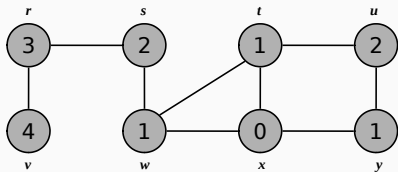


Figure 46: The end. Plus rien à visiter!

En faisant ce parcours...

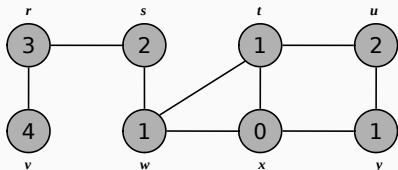
Du parcours de l'arbre



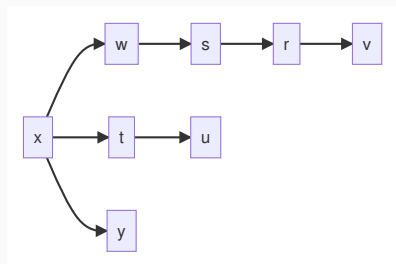
Quel arbre est créé par le parcours (2min)?

En faisant ce parcours...

Du parcours de l'arbre



Quel arbre est créé par le parcours (2min)?



Remarques

- Le parcours dépend du point de départ dans le graphe.
- L'arbre sera différent en fonction du noeud de départ, et de l'ordre de parcours des voisins d'un noeud.

L'algorithme, idée générale (3min, matrix)?

Le parcours en largeur

L'algorithme, idée générale (3min, matrix)?

`v = un sommet du graphe`

`i = 1`

`pour sommet dans graphe et sommet non-visité`

`visiter(v, sommet, i) // marquer sommet à distance i visité`

`i += 1`

Remarque

- `i` est la distance de plus courts chemin entre `v` et les sommets en cours de visite.

Le parcours en largeur

L'algorithme, pseudo-code (3min, matrix)?

- Comment garder la trace de la distance?

Le parcours en largeur

L'algorithme, pseudo-code (3min, matrix)?

- Comment garder la trace de la distance?
- Utilisation d'une **file**

Le parcours en largeur

L'algorithme, pseudo-code (3min, matrix)?

- Comment garder la trace de la distance?
- Utilisation d'une **file**

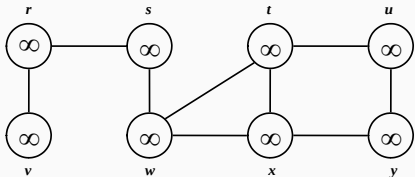
```
initialiser(graphe) // tous sommets sont non-visités
file = visiter(sommet, vide) // sommet est un sommet du graphe a
tant que !est_vide(file)
    v = défiler(file)
    file = visiter(v, file)
```

Que fait visiter?

```
file visiter(sommet, file)
    sommet = visité
    pour w = chaque arête de sommet
        si w != visité
            file = enfiler(file, w)
    retourne file
```

Exercice (5min)

Appliquer l'algorithme sur le graphe



- En partant de *v*, *s*, ou *u* (par colonne de classe).
- Bien mettre à chaque étape l'état de la file.

Complexité du parcours en largeur

Étape 1

- Extraire un sommet de la file;

Étape 2

- Traiter tous les sommets adjacents.

Quelle est la complexité?

Étape 1

- Extraire un sommet de la file;

Étape 2

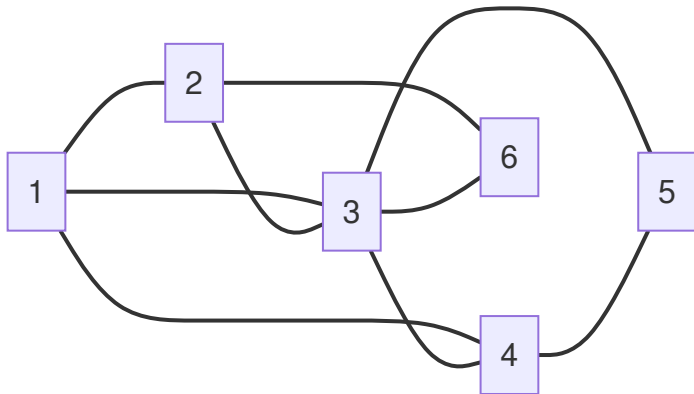
- Traiter tous les sommets adjacents.

Quelle est la complexité?

- Étape 1: $\mathcal{O}(|V|)$,
- Étape 2: $\mathcal{O}(2|E|)$,
- Total: $\mathcal{O}(|V| + 2|E|)$.

Exercice

- Établir la liste d'adjacence et appliquer l'algorithme de parcours en largeur au graphe



Idée générale

- Initialiser les sommets comme non-lus
- Visiter un sommet
- Pour chaque sommet visité, on visite un sommet adjacent s'il est pas encore visité récursivement.

Remarque

- La récursivité est équivalent à l'utilisation d'une **pile**.

Pseudo-code (5min)

Parcours en profondeur

Pseudo-code (5min)

```
initialiser(graphe) // tous sommets sont non-visités  
pile = visiter(sommet, vide) // sommet est un sommet du graphe a  
tant que !est_vide(pile)  
    v = dépiler(pile)  
    pile = visiter(v, pile)
```

Que fait visiter?

Parcours en profondeur

Pseudo-code (5min)

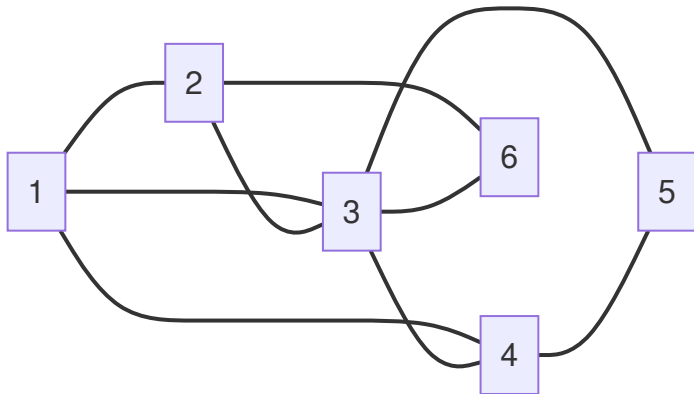
```
initialiser(graphe) // tous sommets sont non-visités
pile = visiter(sommet, vide) // sommet est un sommet du graphe a
tant que !est_vide(pile)
    v = dépiler(pile)
    pile = visiter(v, pile)
```

Que fait visiter?

```
pile visiter(sommet, pile)
    sommet = visité
    pour w = chaque arête de sommet
        si w != visité
            pile = empiler(pile, w)
    retourne pile
```


Exercice

- Établir la liste d'adjacence et appliquer l'algorithme de parcours en profondeur au graphe



Interprétation des parcours

- Un graphe vu comme espace d'états (sommet: état, arête: action);
 - Labyrinthe;
 - Arbre des coups d'un jeu.

Interprétation des parcours

- Un graphe vu comme espace d'états (sommet: état, arête: action);
 - Labyrinthe;
 - Arbre des coups d'un jeu.
- BFS (Breadth-First) ou DFS (Depth-First) parcourent l'espace des états à la recherche du meilleur mouvement.
 - Les deux parcourent *tout* l'espace;
 - Mais si l'arbre est grand, l'espace est gigantesque!

Interprétation des parcours

- Un graphe vu comme espace d'états (sommet: état, arête: action);
 - Labyrinthe;
 - Arbre des coups d'un jeu.
- BFS (Breadth-First) ou DFS (Depth-First) parcourent l'espace des états à la recherche du meilleur mouvement.
 - Les deux parcourent *tout* l'espace;
 - Mais si l'arbre est grand, l'espace est gigantesque!
- Quand on a un temps limité
 - BFS explore beaucoup de coups dans un futur proche;
 - DFS explore peu de coups dans un futur lointain.