



INFO0501

ALGORITHMIQUE AVANCÉE

COURS 3

**GRAPHES
ALGORITHMES ÉLÉMENTAIRES**



**UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE**

Pierre Delisle
Département de Mathématiques, Mécanique et Informatique
Septembre 2019

Plan de la séance

- Parcours en largeur
 - Parcours en profondeur
-
- Bibliographie
 - T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Algorithmique", 3^e édition, Dunod, 2010

Parcours de graphe ?

- Sert de base à plusieurs algorithmes
- Permet d'étudier les propriétés du graphe
 - Le graphe est-il connexe ?
 - Le graphe est-il biparti ?
- On pourra aussi faire des traitements sur les sommets et les arcs/arêtes durant le parcours
- 2 types de parcours
 - Largeur
 - Profondeur



PARCOURS DE GRAPHE EN LARGEUR

Parcours en largeur

- Soit un graphe $G = (S, A)$ et un sommet origine s
- Emprunte les arêtes/arcs de G pour découvrir tous les sommets accessibles depuis s
- Calcule la distance (plus petit nombre d'arcs)
 - Entre s et chaque sommet accessible
- Construit un arbre de parcours en largeur
 - De racine s
 - Qui découvre tous les sommets situés à une distance k avant les sommets de distance $k + 1$
- Durant le parcours, chaque sommet devient successivement
 - Non Découvert
 - Découvert
 - Découvert et tous ses sommets adjacents ont été découverts
- Utilise une file pour gérer la découverte des sommets



FLASHBACK

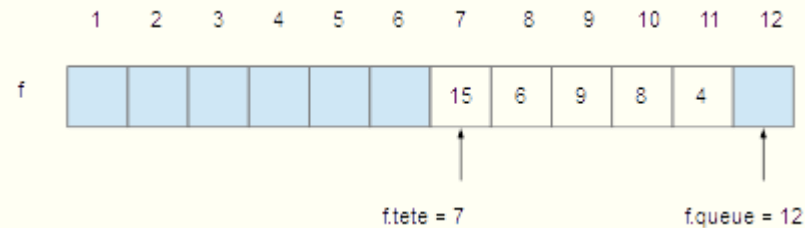
Files

Files

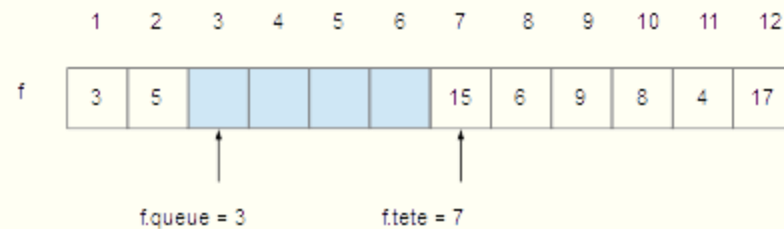
- Premier entré, premier sorti → FIFO
- Insérer → Enfiler
- Supprimer → Défiler
- Implémentation par tableau (circulaire)
 - Au plus $n - 1$ éléments → tableau $f[1...n]$
 - Attribut $f.queue$ qui indexe la queue
 - Enfiler : insertion en $f[f.queue]$
 - Attribut $f.tête$ qui indexe la tête
 - Défiler : suppression en $f[f.tete]$

Files

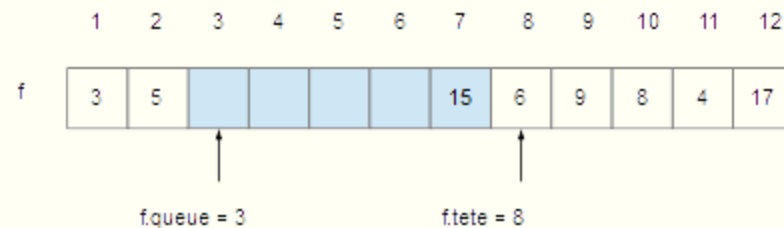
- File contenant 5 éléments



- Après enfilage des valeurs 17, 3 et 5



- Après défilage
 - Retourne la valeur 15
 - L'emplacement 7 n'est plus accessible



- ENFILER (f, x)
 - $O(1)$
- DÉFILER (f)
 - $O(1)$
- RECHERCHER ?



FIN DU FLASHBACK

Files



FLASHBACK

Arbres

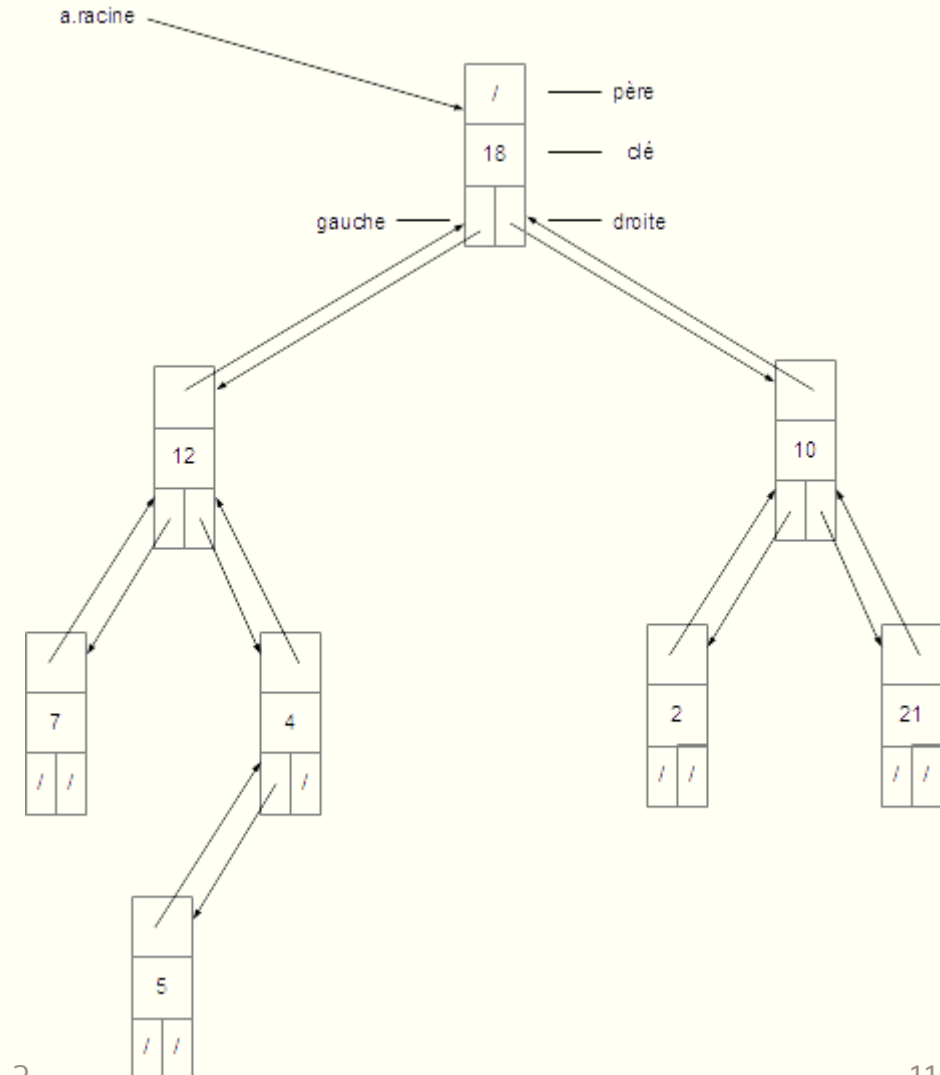
Arbres (binaires)

■ Représentation chaînée

- Chaque nœud/objet contient les attributs
 - *clé*
 - *père* → pointeur vers le père (NIL pour la racine)
 - *gauche* → pointeur vers le fils gauche
 - *droite* → pointeur vers le fils droit
- racine → pointeur sur l'élément racine (NIL si arbre vide)

■ Temps d'exécution ?

- Dépend de l'ordre et de l'organisation des noeuds
- Certaines propriétés doivent être respectées pour que les arbres puissent constituer un dictionnaire efficace
- On verra ça un peu plus tard...





FIN DU FLASHBACK

Arbres

Parcours en largeur

- Exemple 1 : Parcours en largeur

Temps d'exécution du parcours en largeur

- Initialisation de S sommets
 - $\mathcal{O}(S)$
- Chaque sommet est enfilé une fois et défilé une fois
 - $\mathcal{O}(S)$
- Chaque liste d'adjacence est parcourue une fois (quand le sommet est défilé)
 - Somme des longueurs de toutes les listes $\rightarrow \mathcal{O}(A)$
- Total
 - $\mathcal{O}(S + A)$



PARCOURS DE GRAPHERS EN PROFONDEUR

Parcours en profondeur

- On descend plus profondément dans le graphe chaque fois que c'est possible
 - On explore les arcs du sommet découvert le plus récemment
 - Si on trouve un sommet non découvert, on l'explore tout de suite même si on n'a pas exploré tous les autres arcs du sommet en cours
 - On revient en arrière plus tard pour explorer les arcs restants
- Construit une forêt de parcours en profondeur
- Le parcours en profondeur date chaque sommet
 - Date de début → découverte du sommet
 - Date de fin → toute la liste d'adjacence du sommet a été examinée
- Exemple 2
 - Parcours en profondeur

Temps d'exécution du parcours en profondeur

- Initialisation de S sommets
 - $O(S)$
- Chaque liste d'adjacence est parcourue une fois
 - Somme des longueurs de toutes les listes $\rightarrow O(A)$
- Total
 - $O(S + A)$

Parcours en profondeur itératif

- Utilise une pile pour gérer la découverte des sommets



FLASHBACK

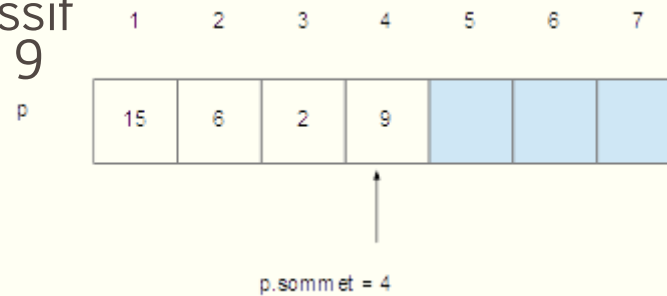
Piles

Piles

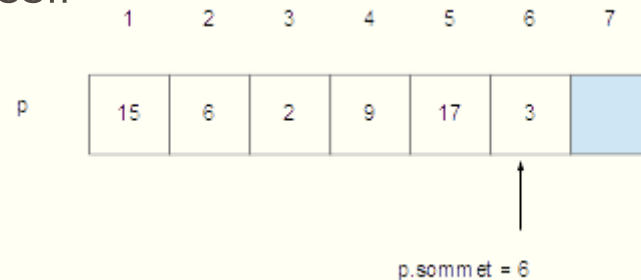
- Dernier entré, premier sorti → LIFO
- Insérer → Empiler
- Supprimer → Dépiler
- Implémentation par tableau
 - Au plus n éléments → tableau p $[1...n]$
 - Possède un attribut $p.sommet$ qui indexe l'élément le plus récemment inséré
 - $p[1]$ → élément situé à la base de la pile
 - $p[p.sommet]$ → élément situé au sommet de la pile

Piles

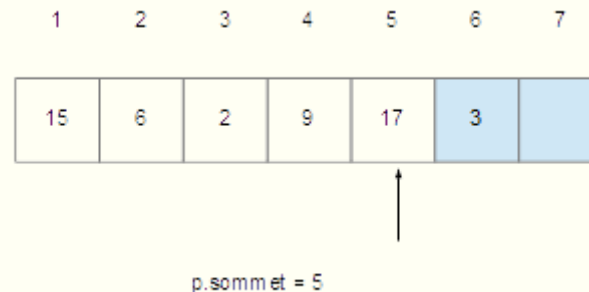
- Après empilage successif des valeurs 15, 6, 2 et 9



- Après empilage successif des valeurs 17 et 3



- Après un dépilage
 - On récupère la valeur 3
 - La case 6 est ensuite non définie, on ne peut pas la réutiliser



- EMPLER (p, x)
 - $\alpha(1)$
- DÉPILER (p)
 - $\alpha(1)$
- PILE-VIDE (p)
 - $\alpha(1)$
- RECHERCHER ?

Programmation ?

Java

- Piles
 - Classe Stack (legacy)
- Piles et Files
 - Interface Deque (ArrayDeque, LinkedList)
 - addFirst(..), addLast(..), removeFirst(), removeLast()
- Listes
 - Interface List (ArrayList, LinkedList)
- Tas/Files de priorité
 - Classe PriorityQueue

C++

- Piles
 - Template stack (containers : vector, deque, list)
 - push(..) (push_back), pop() (pop_back), ...
- Files
 - Template queue (containers : deque, list)
 - push(..) (push_back), pop() (pop_front), ...
- Listes
 - Template list (container : doubly linked list)
- Tas/Files de priorité
 - Template priority_queue (cont. : vector, deque)

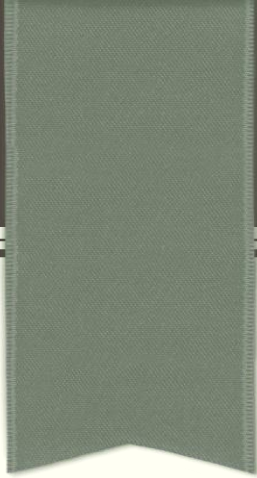


FIN DU FLASHBACK

Piles

Parcours en profondeur itératif

- Exemple 3 : parcours en profondeur itératif en utilisant une pile – Version simple
 - La version itérative avec comportement identique à la version récursive est un peu plus complexe et sera vue en TD



PROCHAIN COURS

ARBRES COUVRANT DE POIDS MINIMUM
