



Université
de Paris

Algorithmie Avancée

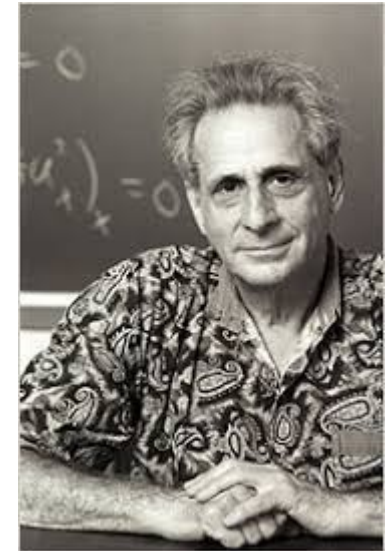
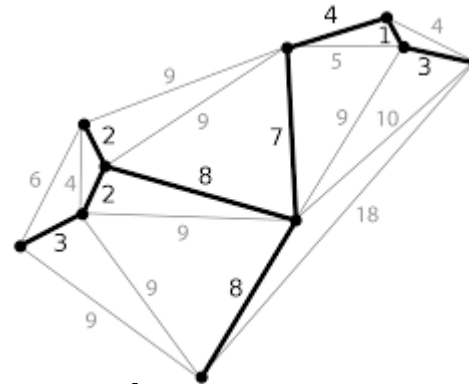
Mise en Contexte / Mise en Oeuvre

Année 2020-2021 par Prof. Nicolas Loménie
Sur la base du cours de Prof. Etienne Birmelé (2016-2020)

Mise en Contexte

Martin D. Kruskal

Edsger W. Dijkstra (1930-2002)



« Il a été initialement développé en 1930 par le mathématicien tchèque Vojtěch Jarník et plus tard, en 1957, a été développé indépendamment par Robert C. Prim. En 1959, il a été redécouvert par Edsger Dijkstra. En raison des contributions de ces chercheurs il est souvent cité comme algorithme DJP (Dijkstra, Jarnik, Prim), l'algorithme Jarnik ou un algorithme de Prim-Jarnik »

<https://boowiki.info/art/les-algorithmes-d-optimisation/algorithme-de-prim.html>



Along with coworker Joseph Kruskal, they developed two different algorithms for finding a minimum spanning tree in a weighted graph, a basic stumbling block in computer network design.

Dr. Robert Clay Prim
(By OpenGenus Foundation)

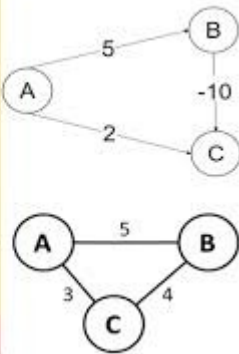
<https://www.nytimes.com/2007/01/13/obituaries/13kruskal.html>

https://fr.wikipedia.org/wiki/Programmation_dynamique

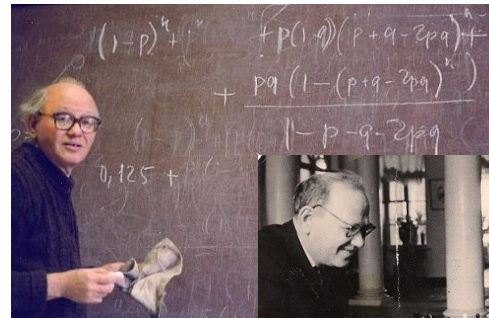
Un peu d'histoire des algorithmes et des hommes

Errements, découvertes, redécouvertes, financement, politique, buzz,
recherche d'algorithmes pour des problèmes réels :

« I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named [Wilson](#). He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities. »



https://en.wikipedia.org/wiki/Evgenii_Landis



Georgii Adelson-Velsky



<https://dailyscience.be/05/04/2018/la-vie-en-graphes/>

https://fr.wikipedia.org/wiki/Richard_Bellman

https://fr.wikipedia.org/wiki/Robert_Tarjan

Théorie des Graphes 4

- [AlgoAvanceeParE_Birmele.pdf](#)

Support de cours de Prof. Etienne Birmelé

Planche 64 à 84 (MST, Kruskal, Complexité UNION-FIND)

Enumérer les sommets

ParcoursGenerique(G,s):

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

ParcoursNumerotation(G,s):

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

Bug
u


UneComposante(G,s):

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

ComposantesConnexes(G)

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

Implémenter BFS(s) avec niveau

- $Discovered[s] := true; Discovered[v] := false$ 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
- Manque Fin Pour
- 

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

Bug
Boucle infinie ?

- Les piles et les files sont des sacs.
- Une pile est un sac LIFO (Last-In First-Out)
 - ▶ lorsqu'on retire un élément, on récupère toujours le dernier ajouté.
 - ▶ ajouter se dit *empiler* (push), retirer se dit *dépiler* (pop).
- Une file est un sac FIFO (First-in First-Out)
 - ▶ on retire les éléments dans l'ordre de leur insertion.
 - ▶ ajouter se dit *enfiler*, retirer se dit *défiler*.



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

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 $DFS(v)$

FIFO / LIFO

Parcours Préfixe/Suffixe/Infixe

Déf. Dans un **parcours préfixe** (*preorder traversal*), chaque nœud est visité avant que ses enfants soient visités.

Déf. Dans un **parcours postfixe** (*postorder traversal*), chaque nœud est visité après que ses enfants sont visités.

Algo PARCOURS-PRÉFIXE(x)

```
1 if  $x \neq \text{null}$  then  
2   visiter  $x$   
3   for  $i \leftarrow 1, \dots, k$  do PARCOURS-PRÉFIXE(enfant( $x, i$ ))
```

Algo PARCOURS-POSTFIXE(x)

```
1 if  $x \neq \text{null}$  then  
2   for  $i \leftarrow 1, \dots, k$  do PARCOURS-POSTFIXE(enfant( $x, i$ ))  
3   visiter  $x$ 
```


FIFO / LIFO

Parcours Préfixe/Suffixe/Infixe

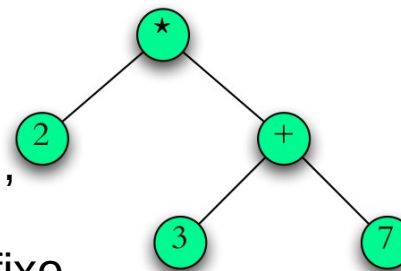
Spécifiquement pour un arbre binaire, une troisième possibilité :

Déf. Dans un **parcours infixe** (*inorder traversal*), chaque nœud est visité après son enfant gauche mais avant son enfant droit.

Algo PARCOURS-INFIXE(x)

```
1 if  $x \neq \text{null}$  then  
2   PARCOURS-INFIXE( $\text{gauche}(x)$ )  
3   visiter  $x$   
4   PARCOURS-INFIXE( $\text{droit}(x)$ )
```

Arbre syntaxique



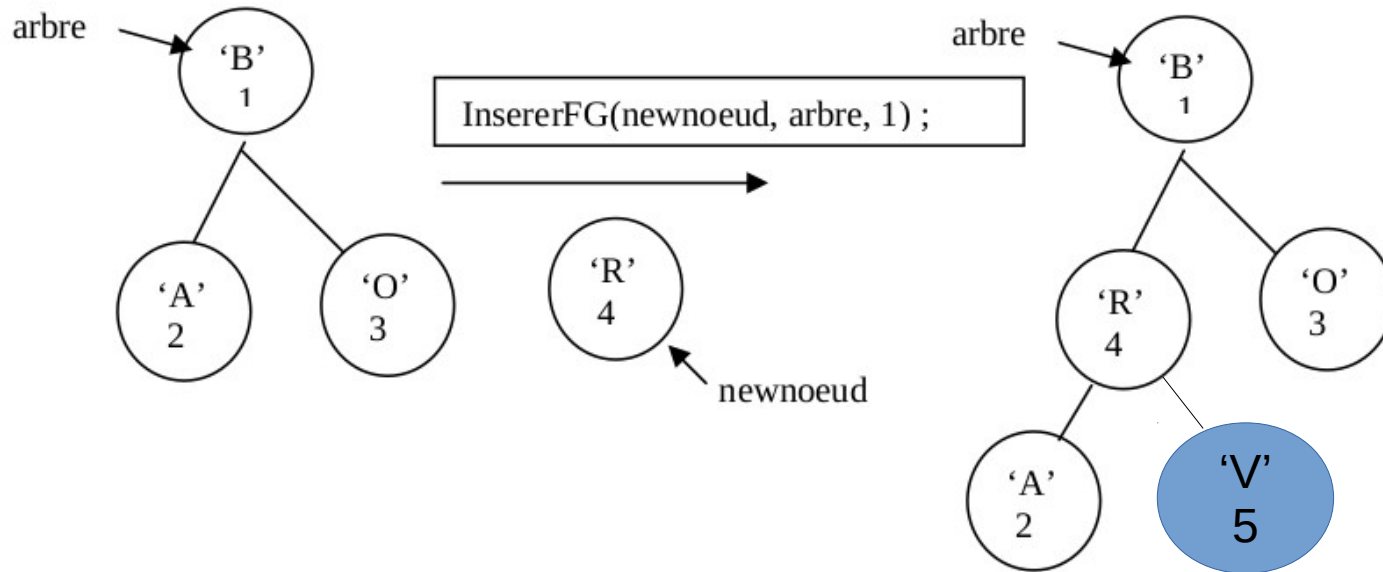
notation infixe: $2 \star (3 + 7)$
notation préfixe: $\star 2 + 3 7$
notation postfixe: $2 3 7 + \star$

Notation polonaise pour préfixe (DFS récursif),
Polonaise inversée pour postfixe,
Et ajout de parathèse par sous-arbre pour l'infixe

Le parcours infixe permet également de trier un arbre binaire de recherche

FIFO / LIFO

Parcours Préfixe/Suffixe/Infixe



Création récursive d'un arbre binaire ?

FIFO / LIFO

Parcours Préfixe/Suffixe/Infixe

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 (mais récursion lourde)

Exemple: montrer qu'un arbre dont tous les sommets sont d'arité au moins 2 possède plus de feuilles que de sommets internes.

Plus simple pour 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.

De façon très formelle, certains diront qu'un arbre binaire d'entiers est solution de l'équation (voir Cours de l'X):

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

FIFO / LIFO

Parcours Préfixe/Suffixe/Infixe

Arbres binaires

Un arbre binaire possède des **nœuds** et des **feuilles**. Certains disent plutôt **nœuds internes** et **nœuds externes**.

On peut définir la structure d'arbre binaire de façon récursive.

Si \mathcal{N} (*resp.* \mathcal{F}) désigne l'ensemble des valeurs des nœuds (*resp.* des valeurs des feuilles), l'ensemble $\mathcal{A}(\mathcal{N}, \mathcal{F})$ des arbres binaires est défini par :

- ▷ toute feuille est un arbre : $\mathcal{F} \subset \mathcal{A}(\mathcal{N}, \mathcal{F})$;
- ▷ si α et β sont deux arbres, et si $n \in \mathcal{N}$, alors (n, α, β) est un arbre.

Le typage Camlcorrespondant est le suivant :

```
type ('n,'f) arbre_binaire =  
  | Feuille of 'f  
  | Noeud of 'n * ('n,'f) arbre_binaire * ('n,'f) arbre_binaire ;;
```

Un exemple d'arbre binaire du type `(int,string) arbre_binaire` est :

```
Noeud(1,Feuille "a",  
      Noeud(6,Noeud(4,Feuille "c",  
                    Feuille "d")  
            Feuille "b"))
```

→ Manque une ,

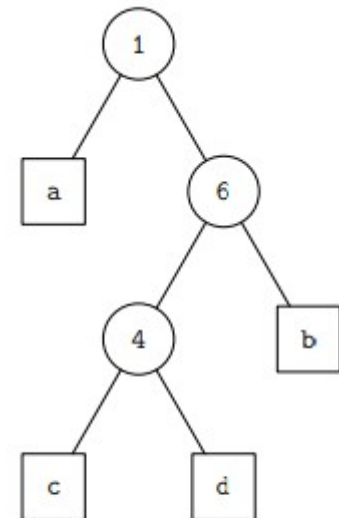


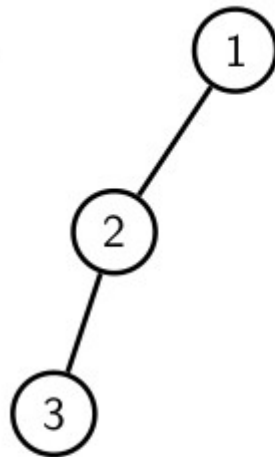
Figure 1 Un premier exemple d'arbre binaire

FIFO / LIFO

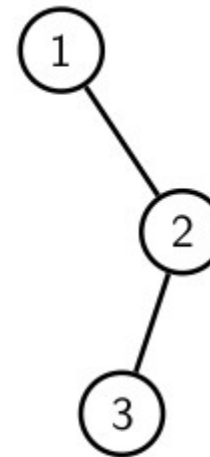
Parcours Préfixe/Suffixe/Infixe

Un arbre binaire est différent d'un arbre dont tous les degrés sont ≤ 2 car la notion de filsG et filsD (donc de fils ordonné dans un tableau ou liste à deux éléments indicés) est importante

En fait,



et



ne sont pas les même arbres binaires, car

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

Reproduisez cette notation pour le cas de l'arbre « Bravo »
(format Caml ou Lisp : langages de programmation fonctionnel dont la structure de base sous forme de liste récursive en font un puissant outil de modélisation pour les graphes)

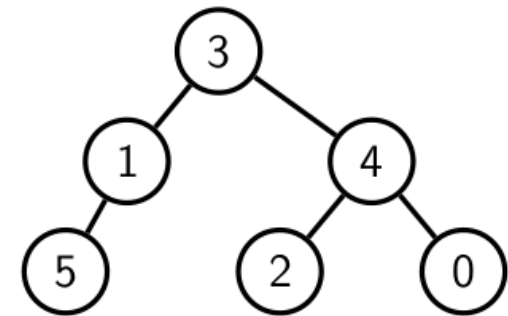
<https://caml.inria.fr/about/history.fr.html>

<https://fr.wikipedia.org/wiki/Lisp>

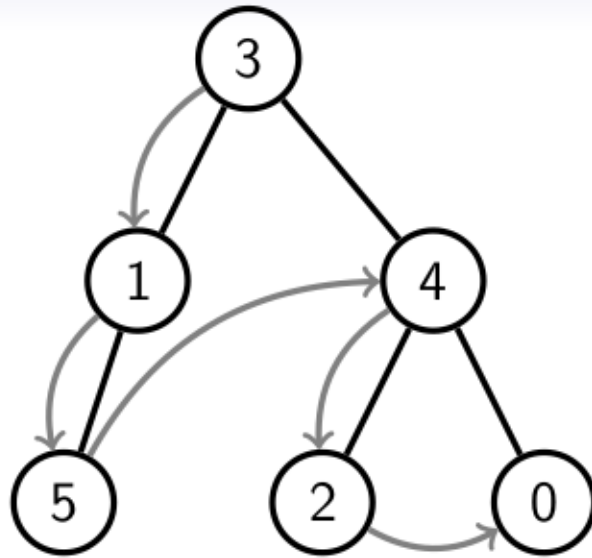
Structure réursive de construction

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

```
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)))
```



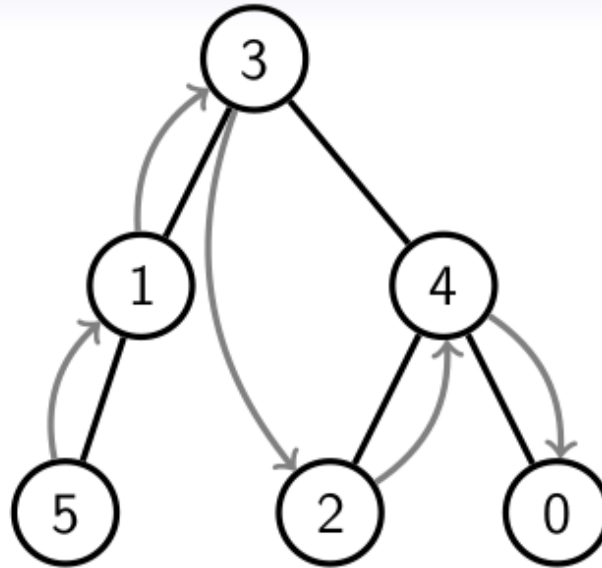
Parcours en profondeur: préfixe



- 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);  
}
```

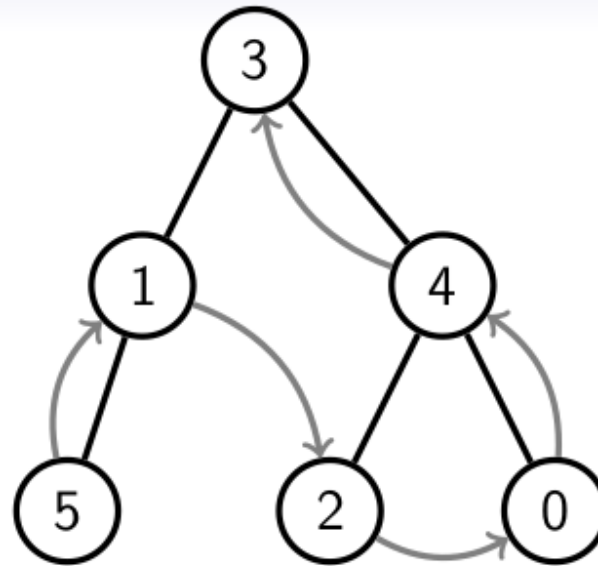
Parcours en profondeur: infixe



- 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+" ");  
}
```

Pour conclure

- DFS, BFS arbre couvrant sur des graphes non valué
- Si graphe valué sur les arête : Kruskal algorithme donne un AC- \rightarrow MST Minimum Spanning Tree (AC de poids minimal ou ACM)
- Complexité amortie : due à A. Tarjan
- En C ? En Java ? Commencer à réfléchir aux formats pour charger/créer des graphes dans vos programmes facilement