Algorithmique et structures de données
CM 6 – Arbres binaires

Jean-Marie Le Bars
jean-marie.lebars@unicaen.fr

UNICAEN

Plan du CM 6

Évaluation de conditions

Arbres binaires particuliers

Nombre de nœuds et hauteur

Algorithmes de parcours sur un arbre binaire

Plan du CM 6

Évaluation de conditions

Arbres binaires particuliers

Nombre de nœuds et hauteur

Algorithmes de parcours sur un arbre binaire

Évaluation de conditions

condition1 ou condition2

- la condition1 est d'abord évaluée
- si la condition1 est vérifiée, on retourne `Vrai`
- si la condition1 n'est pas vérifiée, on évalue la condition2
- on retourne `Vrai` si la condition2 est vérifiée et `Faux` sinon

Exemple

```
base(A : arbreBinaire): booléen
   retourner A = None ou (A->gauche = None et A->droit = None)
```

La procédure teste d'abord si l'arbre binaire *A* est l'arbre vide et ensuite, si ce n'est pas le cas, si c'est l'arbre racine.

Évaluation de conditions

condition1 ou condition2

L'ordre entre les deux conditions a une importance

Exemple

```
base2(A : arbreBinaire): booléen
    retourner (A->gauche = None et A->droit = None) ou A = None
```

Si l'arbre binaire *A* est vide la procédure retourne une erreur car on ne peut utiliser l'instruction `A->gauche = None` lorsque *A* vaut None.

Évaluation de conditions

condition1 ou condition2

La procédure

```
base(A : arbreBinaire): booléen
    retourner A = None ou (A->gauche = None et A->droit = None)
```

est équivalente à la procédure

```
base(A : arbreBinaire): booléen
    si A = None alors
        retourner Vrai
    si (A->gauche = None et A->droit = None) alors
        retourner Vrai
    retourner Faux
```

Évaluation de conditions

condition1 et condition2

- la condition1 est d'abord évaluée
- si la condition1 n'est pas vérifiée, on retourne `Faux`
- si la condition1 est vérifiée, on évalue la condition2
- on retourne `Vrai` si la condition2 est vérifiée et `Faux` sinon

Exemple

```
mystere(A : arbreBinaire): booléen
   retourner A <> None et A->gauche <> None
```

Évaluation de conditions

condition1 et condition2

L'ordre entre les deux conditions a une importance

Exemple

```
mystere(A : arbreBinaire): booléen
   retourner A <> None et A->gauche <> None
```

est différent de

```
mystere(A : arbreBinaire): booléen
   retourner A->gauche <> None et A <> None
```

car on ne peut utiliser l'instruction `A->gauche` lorsque *A* vaut None.

Plan du CM 6

Évaluation de conditions

Arbres binaires particuliers

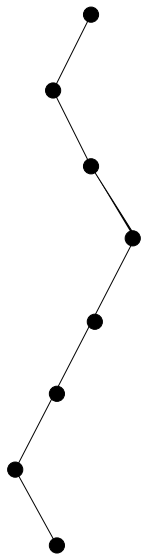
Nombre de nœuds et hauteur

Algorithmes de parcours sur un arbre binaire

Arbre dégénéré ou filiforme

Un arbre est dégénéré lorsque tout nœud possède au plus un enfant.

Exemple



Lien entre hauteur et nombre de nœuds

$$N(A) = h(A) + 1.$$

Arbre dégénéré ou filiforme

Un arbre est dégénéré lorsque tout nœud possède au plus un enfant.

Procédure vérifiant si un arbre A est dégénéré

```
estDegenere(A : arbreBinaire) : booléen
   si A = None alors
      retourner Vrai
   si A->gauche <> None et A->droit <> None alors
      retourner Faux
   si A->gauche <> None alors
      retourner estDegenere(A->gauche)
   si A->droit <> None alors
      retourner estDegenere(A->droit)
   retourner Vrai  #condition terminale le noeud est une feuille
```

Procédure vérifiant si un arbre A est dégénéré

```
estDegenere(A : arbreBinaire) : booléen
   si A = None alors
      retourner Vrai
   si A->gauche <> None et A->droit <> None alors
      retourner Faux
   si A->gauche <> None alors
      retourner estDegenere(A->gauche)
   si A->droit <> None alors
      retourner estDegenere(A->droit)
   retourner Vrai  #condition terminale le noeud est une feuille
```

Exercice

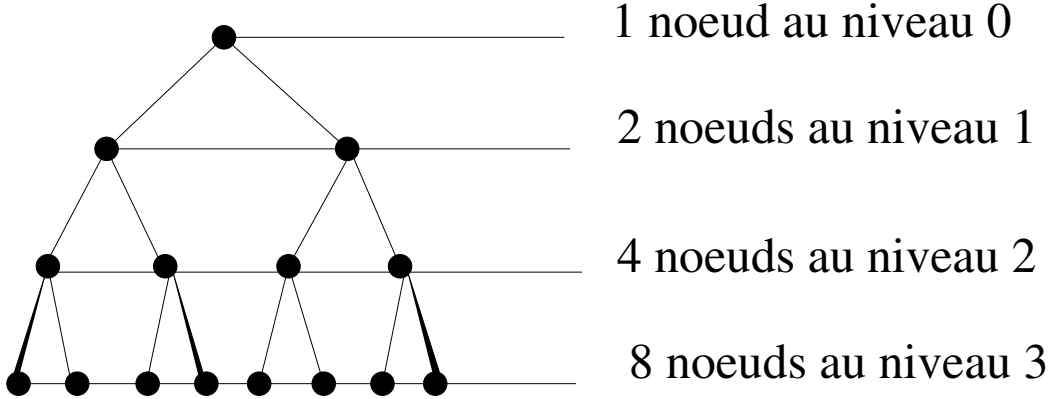
- donnez le nombre d'appels récursifs
- proposez une procédure itérative

Arbre complet

Définition d'un arbre complet

Un arbre de hauteur *h* est complet lorsque tous les niveaux de 0 à *h* sont remplis.

Exemple



Arbre complet

Lien entre hauteur et nombre de nœuds

Nous avons un seul arbre complet C pour une hauteur h, C vérifie

$$N(C) = 2^{h(C)+1} - 1.$$

Nœuds internes et feuilles

- l'arbre complet de hauteur h possède $2^h - 1$ nœuds internes
- l'arbre complet de hauteur h possède 2^h feuilles

Bilan

Tout arbre complet C vérifie

- $N_f(C) = N_i(C) + 1$.
- $N(C) = 2N_f(C) - 1 = 2N_i(C) + 1$

Lien entre hauteur et nombre de nœuds

Cas général

Soit A un arbre binaire.

$$h(A) + 1 \leq N(A) \leq 2^{h(A)+1} - 1.$$

- égalité à gauche pour les arbres dégénérés
- égalité à droite pour les arbres complets

Exercice

Donnez un schéma d'induction pour construire les arbres dégénérés et les arbres complets.

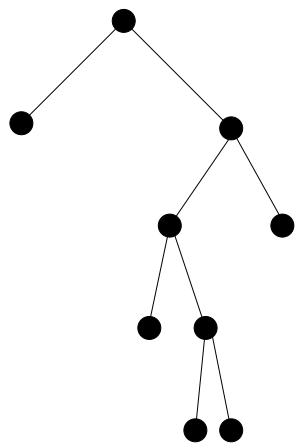
Arbre localement complet

Définition

Un arbre binaire est localement complet lorsque chaque nœud possède

- soit deux enfants
- soit aucun enfant (c'est une feuille)

Exemple



Lien entre nombre de nœuds internes et nombre de feuilles

$$N_f(A) = N_i(A) + 1 \qquad \text{et } N(A) = 2N_i(A) + 1.$$

Mêmes relations que pour les arbres complets (normal, un arbre complet est un arbre localement complet).

Arbre localement complet

Définition

Un arbre binaire est localement complet lorsque chaque nœud possède

- soit deux enfants
- soit aucun enfant (c'est une feuille)

Schéma d'induction

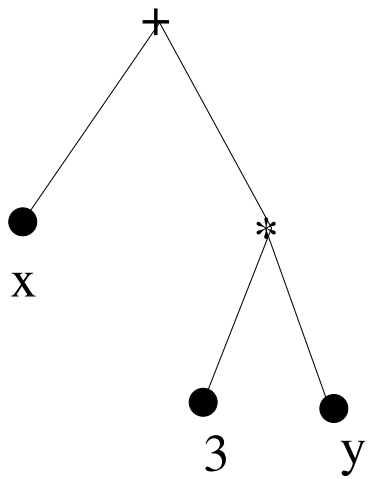
i) l'arbre racine $A = \bullet$ est localement complet

ii) Soient B et C deux arbres localement complets.

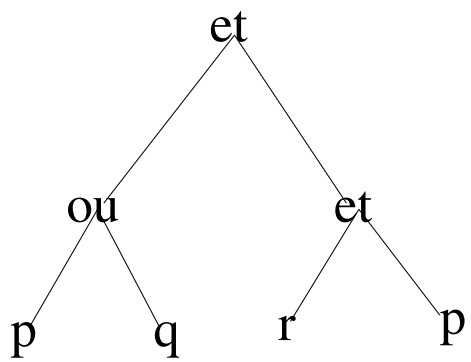
$A = (\bullet, B, C)$ est localement complet

Arbre localement complet

Expressions arithmétiques ou logiques



x+(3*y)



(p ou q) et (r et p)

Il faut avoir uniquement des opérateurs binaires

Arbre localement complet

Définition

Un arbre binaire est localement complet lorsque chaque nœud possède

- soit deux enfants
- soit aucun enfant (c'est une feuille)

Procédure vérifiant si un arbre binaire est localement complet

```
estLC(A : arbreBinaire) : booléen
   si A = None alors # l'arbre vide n'est pas localement complet
      retourner Faux

   si A->gauche = None et A->droit = None alors # condition terminale
      retourner Vrai # l'arbre racine est localement complet

   si A->gauche <> None et A->droit <> None alors#le noeud possède
      retourner estLC(A->gauche) et estLC(A->droit)#deux enfants

   retourner Faux # le noeud ne possède qu'un seul enfant
```

Arbre localement complet

Procédure vérifiant si un arbre binaire est localement complet

```
estLC(A : arbreBinaire) : booléen
   si A = None alors # l'arbre vide n'est pas localement complet
      retourner Faux

   si A->gauche = None et A->droit = None alors # condition terminale
      retourner Vrai # l'arbre racine est localement complet

   si A->gauche <> None et A->droit <> None alors#le noeud possède
      retourner estLC(A->gauche) et estLC(A->droit)#deux enfants

   retourner Faux # le noeud ne possède qu'un seul enfant
```

Calcul de $C(A)$, le nombre d'appels de la fonction

- nous avons au moins un appel de la fonction sur A
- pas d'appel de fonction pour les feuilles
- pas d'appel de fonction pour les nœuds internes avec un seul enfant
- deux appels de fonction pour les nœuds internes avec deux enfants

$$C(A) \leq 1 + 2N_i(A).$$

Arbre localement complet

Calcul de $C(A)$, le nombre d'appels de la fonction

$$C(A) = 1 + 2N_i(A).$$

Calcul de $C(A)$ en fonction de $N(A)$ lorsque A est localement complet

$$C(A) = N(A).$$

Preuve. *C'est immédiat car nous avons*

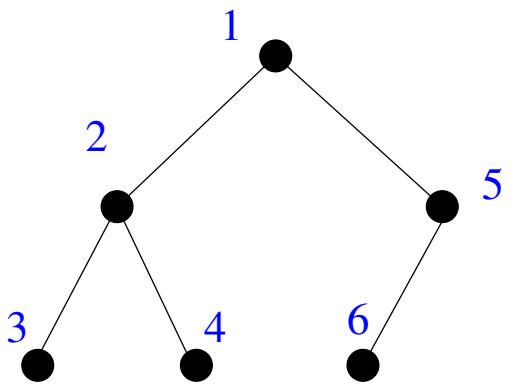
$$N_f(A) = N_i(A) + 1 \qquad \text{et } N(A) = 2N_i(A) + 1.$$

Arbre localement complet

Calcul de $C(A)$ en fonction de $N(A)$ lorsque A n'est pas localement complet

Cela dépend fortement de l'arbre A.

Exemple



- on ne s'aperçoit que l'arbre n'est pas localement complet qu'au dernier nœud

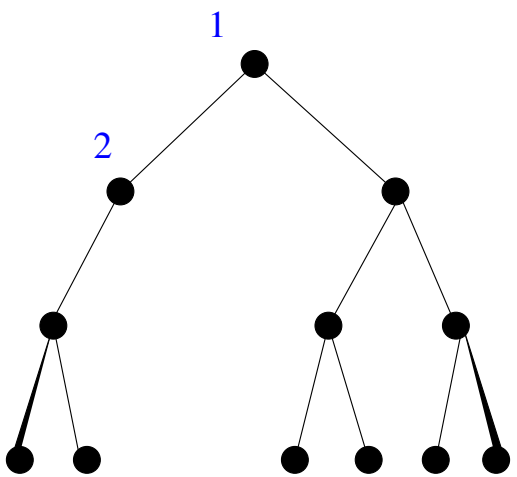
$$C(A) = N(A).$$

Arbre localement complet

Calcul de $C(A)$ en fonction de $N(A)$ lorsque A n'est pas localement complet

Cela dépend fortement de l'arbre A.

Exemple



- on s'aperçoit que l'arbre n'est pas localement complet dès le deuxième nœud

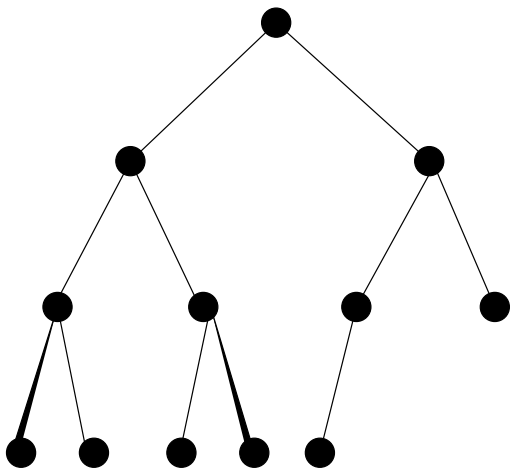
$$C(A) = 2.$$

Arbre binaire parfait

Définition d'un arbre parfait

Tous les niveaux sont remplis sauf éventuellement le dernier niveau, dans ce cas tous les noeuds sont regroupés à gauche.

Exemple



Exercice

Calculez le nombre d'arbres parfaits de hauteur h.

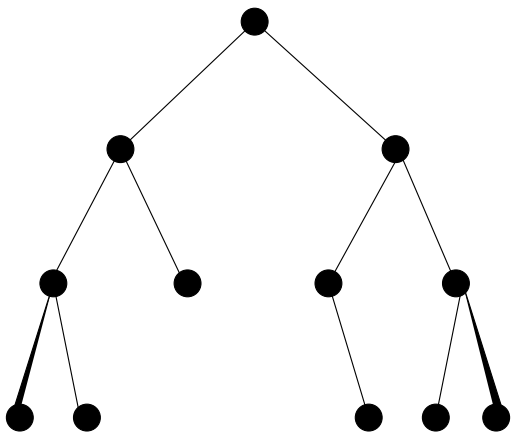
Réponse : $2^h - 1$

Arbre binaire quasi-parfait

Arbre quasi-parfait

Tous les niveaux sont remplis sauf éventuellement le dernier niveau.

Exemple



Exercice

Calculez le nombre d'arbres quasi-parfaits de hauteur h.

Réponse : $2^{2^h} - 1$

Arbre équilibré

Objectif

Contrôler la hauteur par rapport au nombre de nœuds.

Il existe plusieurs définitions possibles

Exemple de définition

Soit A un arbre binaire. A est équilibré lorsque tout sous-arbre B de A vérifie

$$|h(B_g) - h(B_d)| \leq 1,$$

où B_g et B_d sont les sous-arbres gauche et droit de B.

Exemples

Arbres complets, arbres parfaits, arbres quasi-parfaits.

Plan du CM 6

Évaluation de conditions

Arbres binaires particuliers

Nombre de nœuds et hauteur

Algorithmes de parcours sur un arbre binaire

Rappel – structure de nœud et type arbreBinaire

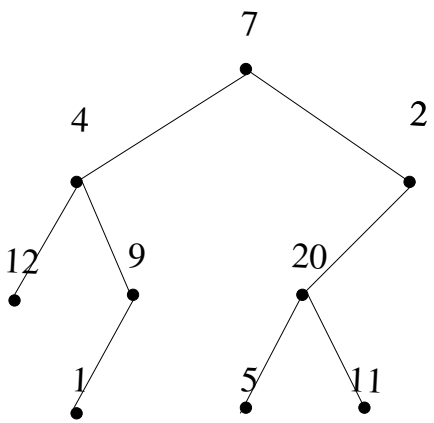
Structure de nœud

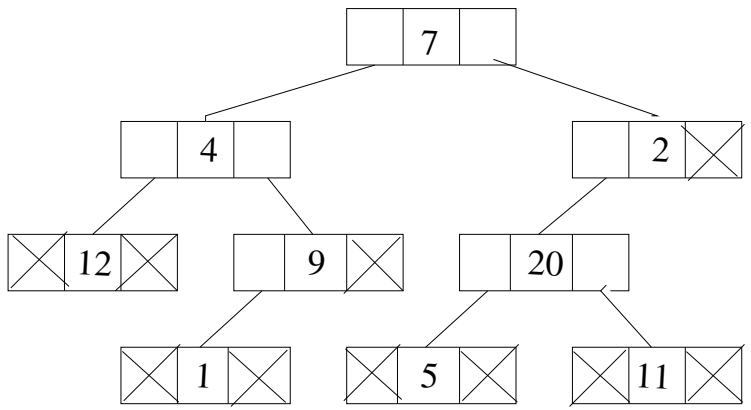
Un nœud est constitué d'une valeur (ici un entier), d'un pointeur sur le sous-arbre gauche et d'un pointeur sur le sous-arbre droit.

```
structure noeud
      valeur : entier
      gauche : pointeur sur noeud
      droit : pointeur sur noeud
```

type arbreBinaire = pointeur sur noeud

Exemple





Calcul du nombre de nœuds

Soit A un arbre binaire et A_g et A_d ses sous-arbres gauche et droit.

Nombre de nœuds en fonction de A_g et A_d

$$N(A) = N(A_g) + N(A_d) + 1.$$

Procédure

```
nombreNoeuds(A : arbreBinaire) : entier
   si A = None alors # condition terminale
      retourner 0

   retourner 1 + nombreNoeuds(A->gauche) + nombreNoeuds(A->droit)
```

Récursivité non terminale

Exercice (voir TD)

1. donnez une procédure calculant le nombre de feuilles.
2. donnez une procédure calculant le nombre de nœuds internes.

Calcul du nombre de nœuds

Calcul de $C(A)$, le nombre d'appels récursifs de la fonction

- la racine est appelée une fois
- pour chaque nœud, on effectue deux appels de fonction

$$C(A) = 1 + 2N(A).$$

Réduction du nombre d'appels de la procédure

Nous pouvons réduire le nombre d'appels de la procédure en supposant que l'arbre est non vide.

Calcul du nombre de nœuds

Réduction du nombre d'appels de la procédure

Nous pouvons réduire le nombre d'appels de la procédure en supposant que l'arbre est non vide.

Nouvelle procédure

```
nombreNoeuds(A : arbreBinaire) : entier
   N : entier ; N = 1
   si A->gauche <> None alors
      N = N + nombreNoeuds(A->gauche)

   si A->droit <> None alors
      N = N + nombreNoeuds(A->droit)

   retourner N #on renvoie 1 lorsque A est une feuille
```

Calcul de $C(A)$, le nombre d'appels de la procédure

Nous avons un appel de la procédure pour chaque nœud.

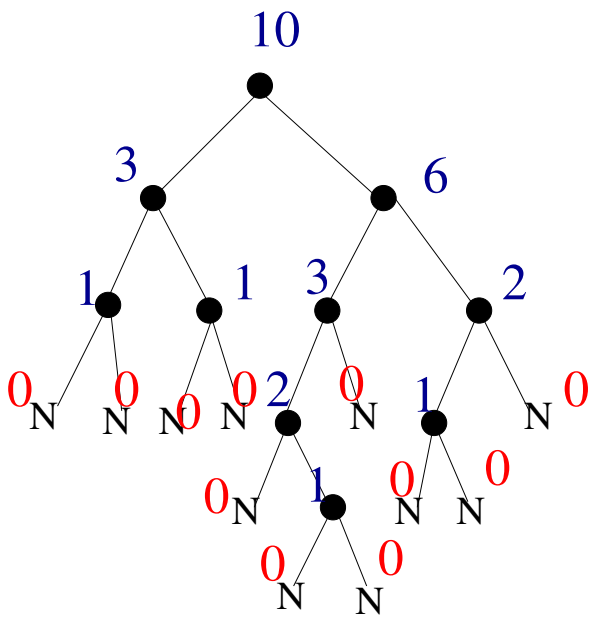
$$C(A) = N(A).$$

Calcul du nombre de nœuds

Procédure

```
nombreNoeuds(A : arbreBinaire) : entier
    si A = None alors # condition terminale
        retourner 0

    retourner 1 + nombreNoeuds(A->gauche) + nombreNoeuds(A->droit)
```

Exemple



Calcul de la hauteur d'un arbre binaire

Définition inductive

- l'arbre racine (réduit à une racine) est de hauteur 0
- par convention, l'arbre vide est de hauteur -1.
- soit A un arbre binaire de sous-arbre gauche A_g et de sous-arbre droit A_d.

$$h(A) = 1 + max(h(A_g), h(A_d)).$$

Procédure

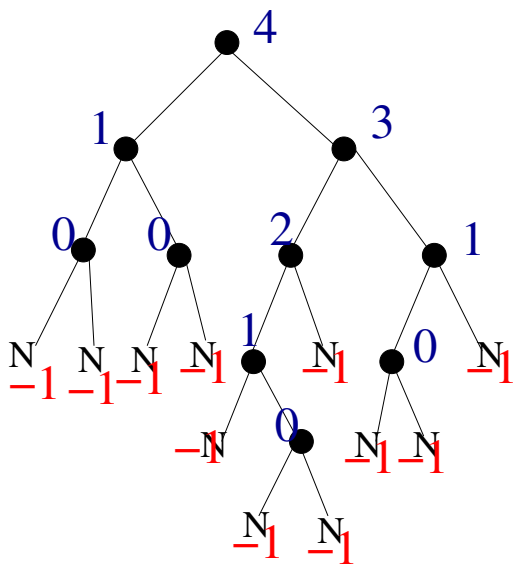
```
hauteurArbre(A : arbreBinaire) : entier
   si A = None alors # condition terminale
      retourner -1
   hG, hD : entier
   hG = hauteurArbre(A->gauche)
   hD = hauteurArbre(A->droit)
   si hG > hD alors retourner hG + 1
   retourner hD + 1
```

Calcul de la hauteur d'un arbre binaire

Procédure

```
hauteurArbre(A : arbreBinaire) : entier
    si A = None alors # condition terminale
        retourner -1
    hG, hD : entier
    hG = hauteurArbre(A->gauche)
    hD = hauteurArbre(A->droit)
    si hG > hD alors retourner hG + 1
    retourner hD + 1
```

Exemple



Arbre complet

Un arbre A de hauteur h est complet si et seulement si

$$N(A) = 2^{h(A)+1} - 1.$$

Comment tester si un arbre est complet

- on ne peut pas tester directement si un arbre binaire est complet
- il n'y a pas de test local (directement sur les nœuds)

Procédure vérifiant si un arbre A est complet

On utilise des procédures déjà définies.

```
estComplet(A : arbreBinaire) : booléen
   n, h : entier
   n = nombreNoeuds(A)
   h = hauteur(A)
   retourner n = 2**(h+1) - 1
```

Plan du CM 6

Évaluation de conditions

Arbres binaires particuliers

Nombre de nœuds et hauteur

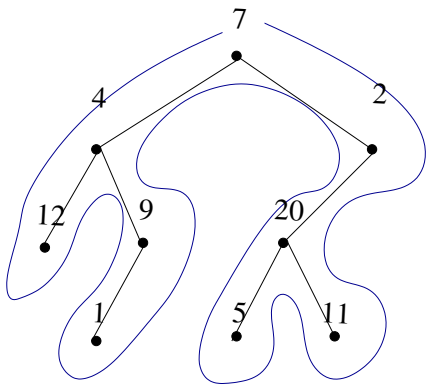
Algorithmes de parcours sur un arbre binaire

Parcours en profondeur

Parcours en profondeur

On part de la racine, on descend le plus à gauche possible et on retourne en arrière pour explorer les autres branches.

Exemple



Ordre de parcours

Chaque nœud est visité trois fois

1. première visite premier passage sur le nœud
2. seconde visite après l'exploration du sous-arbre gauche
3. troisième visite après l'exploration du sous-arbre droit

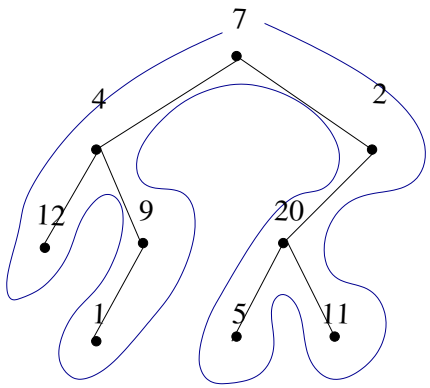
7 4 12 12 12 4 9 1 1 1 9 9 4 7 2 20 5 5 5 20 11 11 11 20 2 2 7

Affichage des nœuds par ordre préfixe

Ordre préfixe

On effectue le traitement (par exemple afficher les valeurs des nœuds) uniquement à la première visite.

Exemple



7 4 12 9 1 2 20 5 11

Algorithme récursif d'affichage

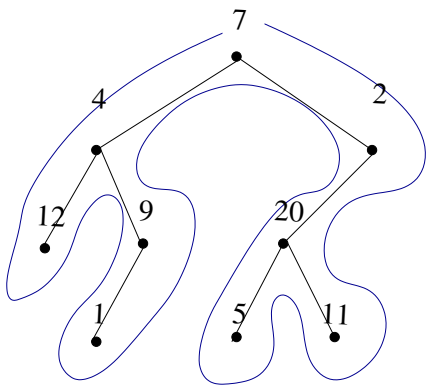
```
affichagePrefixe(A : arbreBinaire)
   si A <> None alors
      afficher A->valeur                    (1)
      affichagePrefixe(A->gauche)           (2)
      affichagePrefixe(A->droite)           (3)
```

Ordre infixe

Ordre infixe

On effectue le traitement à la seconde visite.

Exemple



12 4 1 9 7 5 20 11 2

Algorithme récursif d'affichage

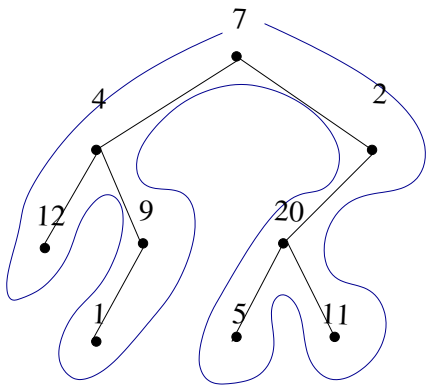
```
affichageInfixe(A : arbreBinaire)
 si A <> None alors
    affichageInfixe(A->gauche)          (2)
    afficher A->valeur                  (1)
    affichageInfixe(A->droite)          (3)
```

Ordre suffixe ou postfixe

Ordre suffixe ou postfixe

On effectue le traitement à la troisième visite.

Exemple



12 1 9 4 5 11 20 2 7

Algorithme récursif d'affichage

```
affichageSuffixe(A : arbreBinaire)
 si A <> None alors
    affichageSuffixe(A->gauche)          (2)
    affichageSuffixe(A->droite)          (3)
    afficher A->valeur                   (1)
```

Parcours en profondeur itératif

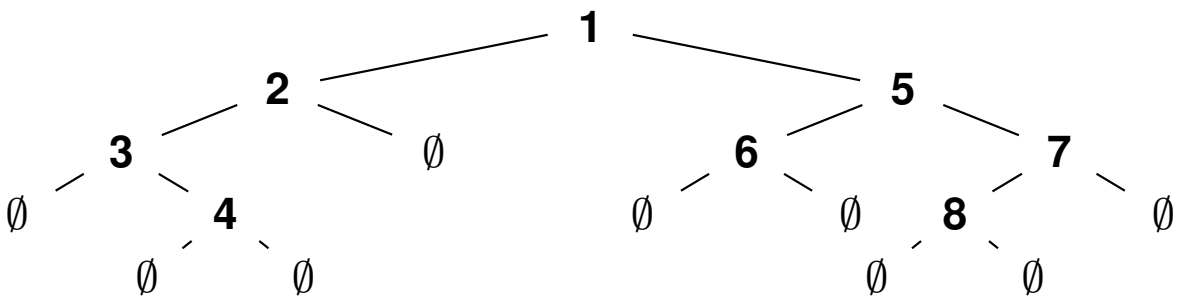
On utilise une pile (de pointeurs sur nœud) pour mémoriser les prochains nœuds à visiter.

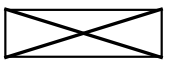
Affichage itératif avec l'ordre préfixe

```
affichagePrefixeIteratif(A : arbreBinaire)
   si A <> None alors
      P : pile ; P = initPile() ; P = empiler(A)

      tant que nonVide(P) faire
         B : pointeur sur noeud ; B = sommet(P); P = depiler(P)

         afficher B->valeur

         si B->droit <> None alors
            P = empiler(P, B->droit)#on commence à droite

         si B->gauche <> None alors# car l'ordre
            P = empiler(P, B->gauche)#est inversé avec la pile
```

Parcours en profondeur itératif

Exécution sur un exemple

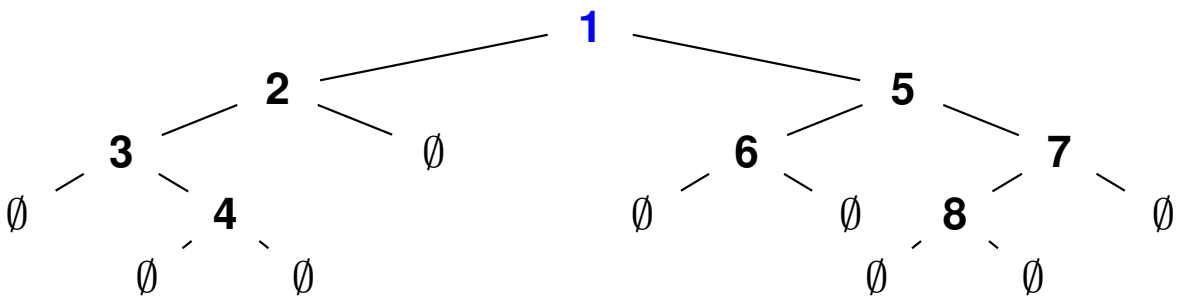


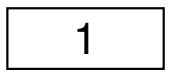


Au début la pile est vide

Parcours en profondeur itératif

Exécution sur un exemple

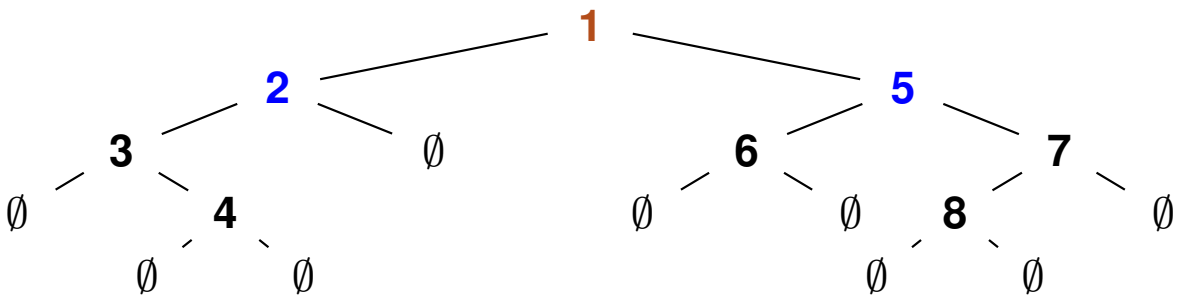


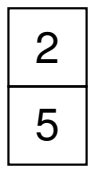


- on empile 1

Parcours en profondeur itératif

Exécution sur un exemple

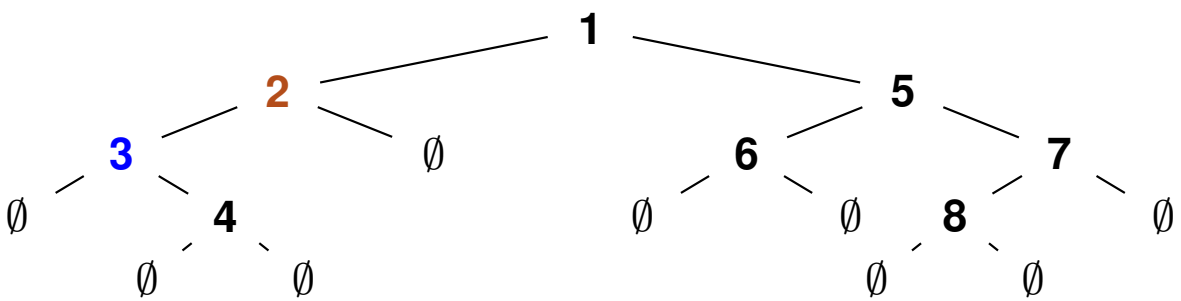


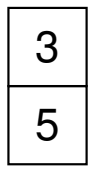


- on affiche 1
- on dépile 1
- on empile 5
- on empile 2

Parcours en profondeur itératif

Exécution sur un exemple

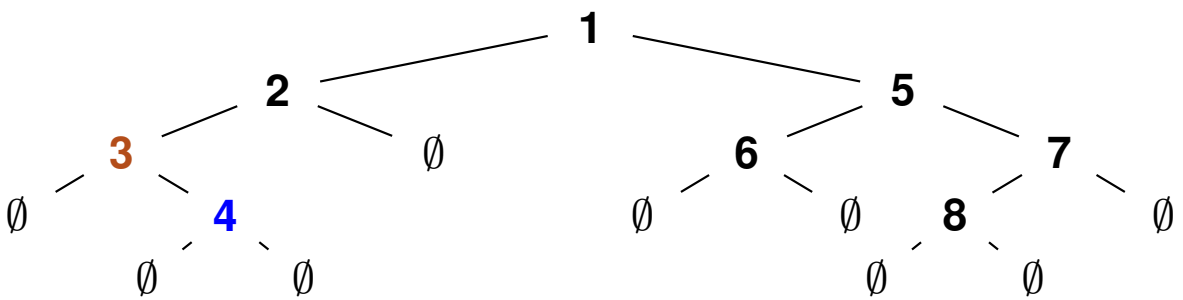




- on affiche 2
- on dépile 2
- on empile 3

Parcours en profondeur itératif

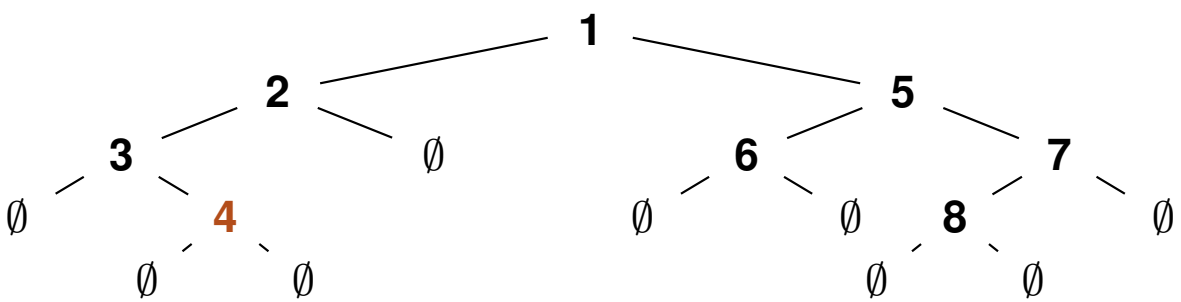
Exécution sur un exemple



- on affiche 3
- on dépile 3
- on empile 4

Parcours en profondeur itératif

Exécution sur un exemple

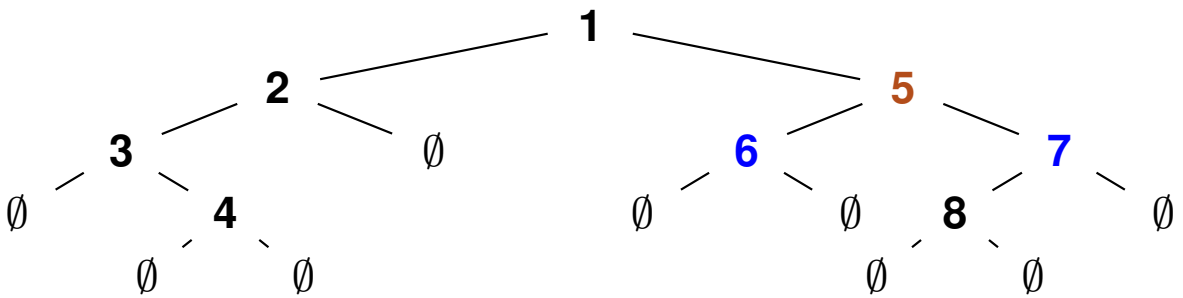


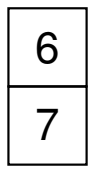
5

- on affiche 4
- on dépile 4

Parcours en profondeur itératif

Exécution sur un exemple

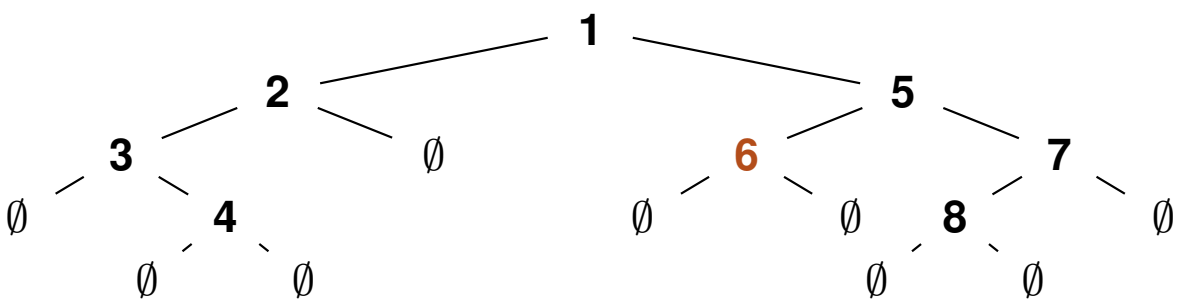




- on affiche 5
- on dépile 5
- on empile 7
- on empile 6

Parcours en profondeur itératif

Exécution sur un exemple

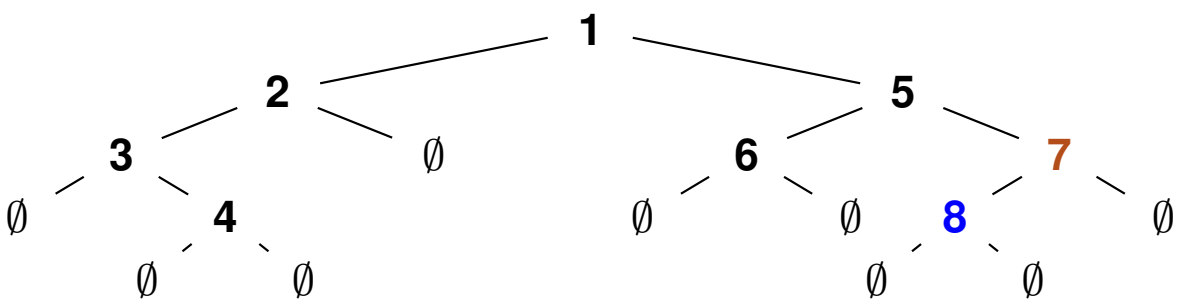


7

- on affiche 6
- dépile 6

Parcours en profondeur itératif

Exécution sur un exemple

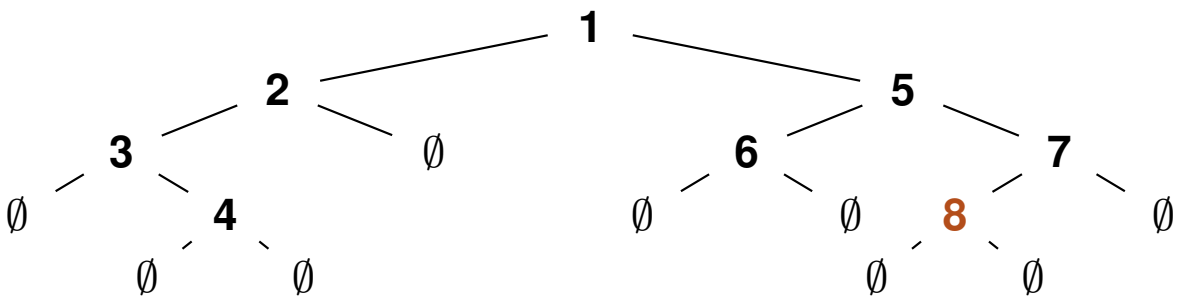


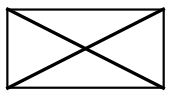
8

- on affiche 7
- on dépile 7
- on empile 8

Parcours en profondeur itératif

Exécution sur un exemple





- on affiche 8
- on dépile 8
- la pile est vide, on s'arrête

Parcours en largeur

On définit un algorithme itératif.

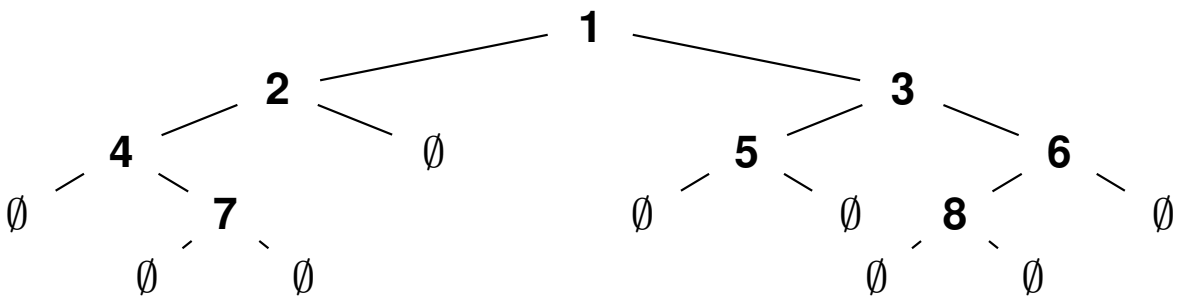
On utilise une file (de nœuds) pour mémoriser les prochains nœuds à visiter.

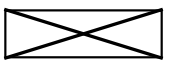
Affichage des nœuds avec un parcours en largeur

```
affichageLargeur(A : arbreBinaire)
   si A <> None alors
      F: file ; F = initFile() ; F = enfiler(F,A)

      tant que nonVide(F) faire
          B : pointeur sur noeud ; B = tete(F) ; F = defiler(F)
          afficher B->valeur

          si B->gauche <> None alors
             F = enfiler(F,B->gauche)

          si B->droit <> None
             F = enfiler(F,B->droit)
```

Parcours en largeur

Exécution sur un exemple

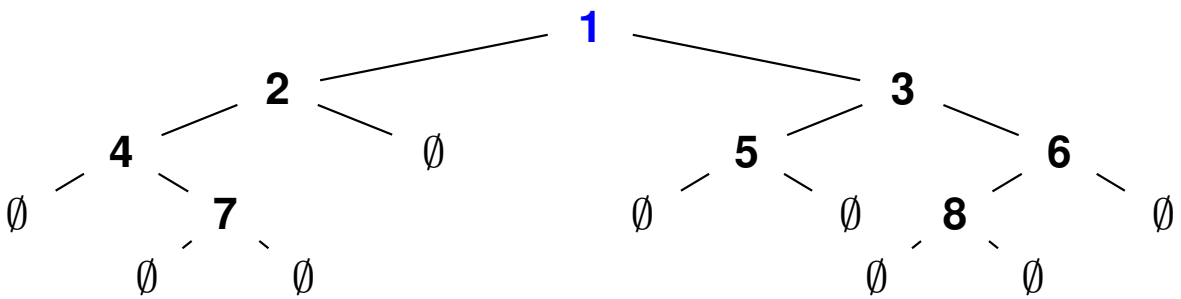


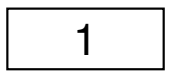


- au début la file est vide

Parcours en largeur

Exécution sur un exemple

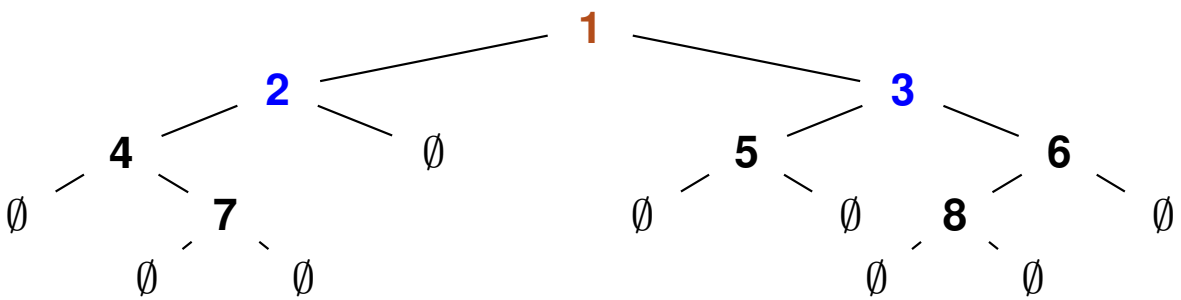




- on enfile 1

Parcours en largeur

Exécution sur un exemple

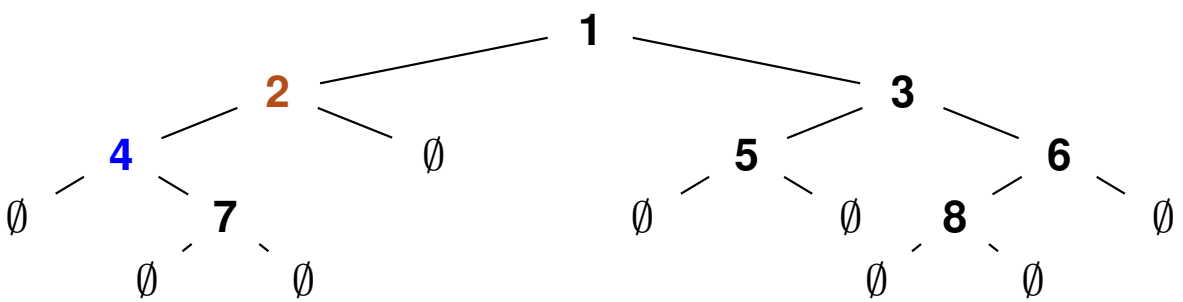


2	3

- on affiche 1
- on défile 1
- on enfile 2
- on enfile 3

Parcours en largeur

Exécution sur un exemple

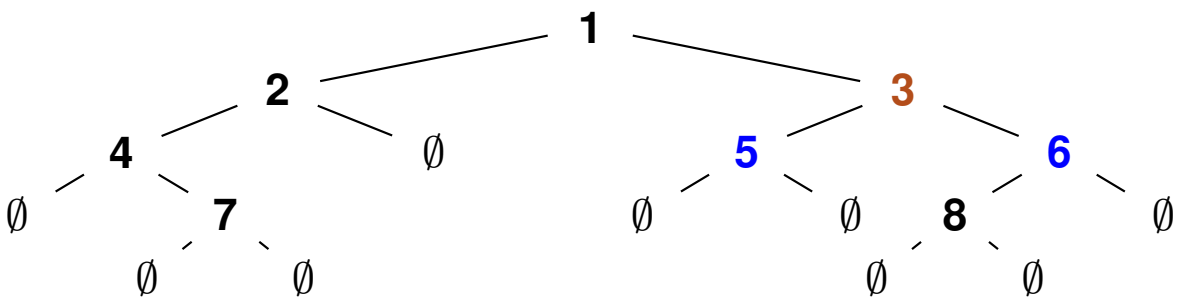


3	4

- on affiche 2
- on défile 2
- on enfile 4

Parcours en largeur

Exécution sur un exemple

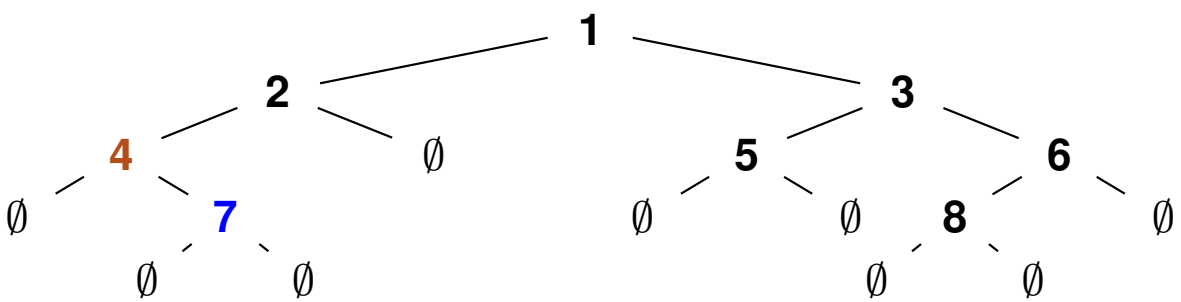


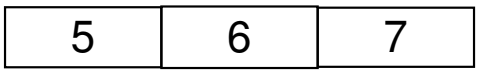
4	5	6

- on affiche 3
- on défile 3
- on enfile 5
- on enfile 6

Parcours en largeur

Exécution sur un exemple

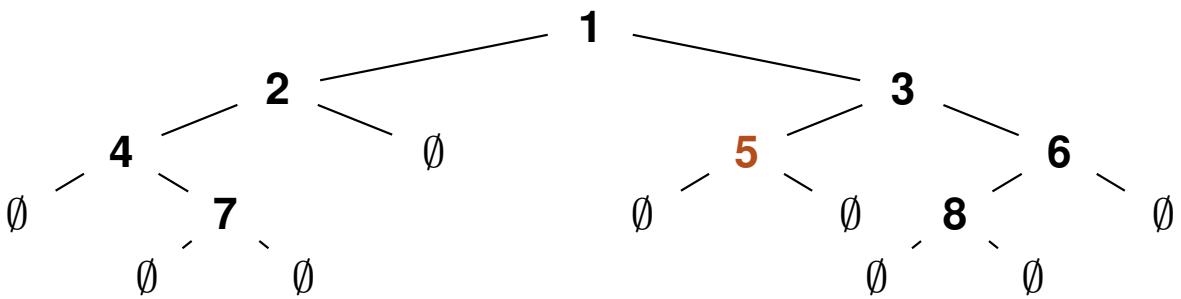


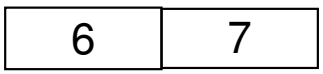


- on affiche 4
- on défile 4
- on enfile 7

Parcours en largeur

Exécution sur un exemple

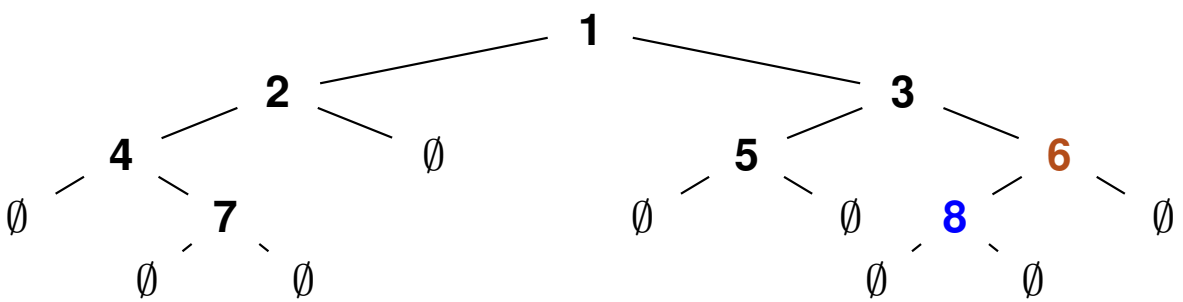


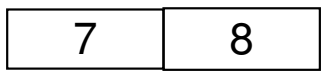


- on affiche 5
- on défile 5

Parcours en largeur

Exécution sur un exemple

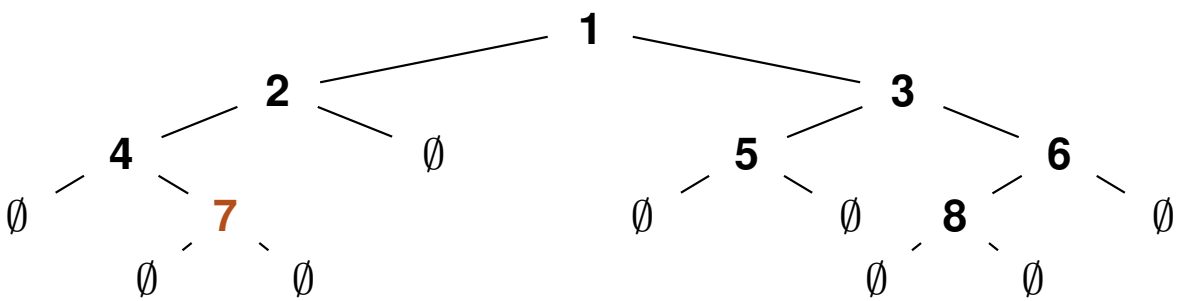




- on affiche 6
- on défile 6
- on enfile 8

Parcours en largeur

Exécution sur un exemple

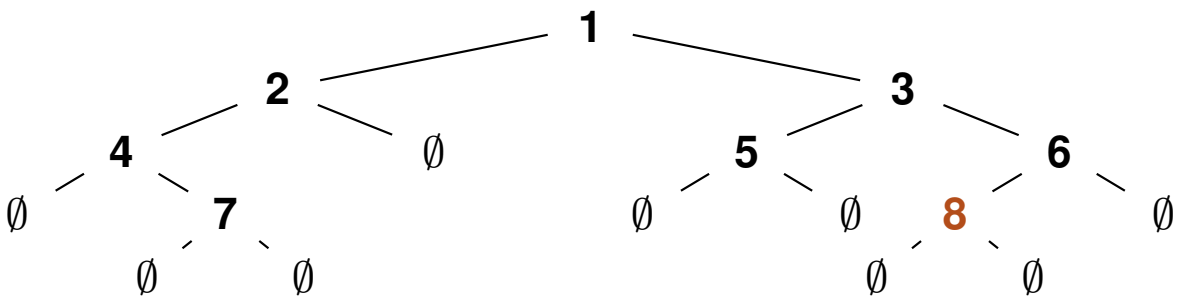


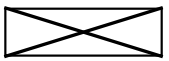
8

- on affiche 7
- on défile 7

Parcours en largeur

Exécution sur un exemple





- on affiche 8
- on défile 8
- la file est vide, on s'arrête