Structures de données concrètes

Yann Strozecki
yann.strozecki@uvsq.fr

Novembre 2016

Structures de Données Abstraites

- Mise en œuvre d'un ensemble dynamique
- Définition de la structure par les opérations pour **manipuler** les données.
- On donne les propriétés des opérations et leur signature.

Quelques structures classiques

1. Liste
2. Pile
3. File
4. Tables de hachage
5. Tas
6. Files de priorité
7. Arbres

Liste chaînée

Qu'est ce qu'une liste chaînée ?

Une structure linéaire de taille variable.

Une liste simplement chaînée

```
Enregistrement Elément {
    Val      : entier;
    Suivant  : ↑ Elément;
}
```

Une liste est donnée par une référence ou pointeur sur cet élément.

Les différentes implémentations des listes

Différents types de liste :

- simplement chainée
- doublement chainée
- terminée par NIL
- terminée par une autoréférence
- circulaire
- avec sentinelle

Liste non triée : Recherche

Algorithme 1 Recherche dans une liste non triée

```
Recherche(L : ↑ Elément, x : entier) : booléen
▷ Entrées : L (tête de la liste), x (élément recherché)
▷ Sortie : vrai si l'élément x a été trouvé dans la liste L, faux sinon.
▷ Variable Locale
    p : ↑ Element ;
Début
    p ← L ;
    tant que (p ≠ NIL) faire
        si (p.Val ≠ x)
            p ← p.Suivant ;
        sinon
            retourner vrai ;
        fin si
    fin tant que
    retourner faux ;
Fin
```

Attention au cas des listes vides.

Une Pile

Définition

Analogie avec une pile d'assiette :
LIFO (Last In First Out ou **Dernier Arrivé Premier Servi**)

- On ne peut rajouter un élément qu'au dessus de la pile
- On a accès uniquement à l'élément qui est au dessus de la pile (élément le plus récemment inséré).

Mise en œuvre

1. A l'aide d'un tableau (*nombre maximum d'éléments dans la pile fixé*)

Type de données

```
Enregistrement Pile {
    T[NMAX] : entier;
    Sommet : entier;
}
```

2. A l'aide d'une liste

Mise en œuvre : un tableau

Algorithme 2 La pile est-elle vide ?

```
PileVide(p : Pile) : booléen
▷ Entrée : P (une pile)
▷ Sortie : vrai si la pile est vide, faux sinon.
Début
si (p.Sommet = -1)
     retourner  vrai ;
sinon
     retourner  faux ;
fin si
Fin
```

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 3 La pile est-elle pleine ?

```
PilePleine(p : Pile) : booléen
▷ Entrée : P (une pile)
▷ Sortie : vrai si la pile est pleine, faux sinon.
Début
si (p.Sommet = NMAX-1)
     retourner  vrai ;
sinon
     retourner  faux ;
fin si
Fin
```

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 4 Insertion d'un élément

```
Insertion(p : Pile, elt : entier)
▷ Entrée : p (une pile) et elt (un entier)
▷ Sortie : la pile p dans laquelle elt a été inséré
Début
si (PilePleine(p) = faux )
    p.Sommet ← p.Sommet + 1 ;
    p.T[p.Sommet] ← elt ;
sinon
    Afficher un message d'erreur
fin si
Fin
```

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 5 Suppression d'un élément

```
Suppression(p : Pile) : entier
▷ Entrée : p (une pile) e
▷ Sortie : renvoie l'élément qui était au sommet de la pile p et
supprime l'élément de la pile
▷ Variable locale :
    elt : entier ;
Début
si (PileVide(p) =  faux )
    elt ← p.T[p.Sommet] ;
    p.Sommet ← p.Sommet - 1 ;
     retourner elt ;
sinon
    Afficher un message d'erreur
fin si
Fin
```

Complexité : $O(1)$

Stack overflow

- Quand on dépile une pile vide : stack underflow.
- Quand on empile un élément dans une pile pleine : stack overflow.

- Cette dernière erreur arrive fréquemment dans tous les langages de programmation quand on sature la pile d'appels.
- En C, on peut régler la taille de la pile d'appels (NMAX) pour régler ce problème.

Stack overflow

- Quand on dépile une pile vide : stack underflow.
- Quand on empile un élément dans une pile pleine : stack overflow.

- Cette dernière erreur arrive fréquemment dans tous les langages de programmation quand on sature la pile d'appels.
- En C, on peut régler la taille de la pile d'appels (NMAX) pour régler ce problème.
- Exemple d'un programme qui déclenche un stack overflow en C ?

Stack overflow

- Quand on dépile une pile vide : stack underflow.
- Quand on empile un élément dans une pile pleine : stack overflow.

- Cette dernière erreur arrive fréquemment dans tous les langages de programmation quand on sature la pile d'appels.
- En C, on peut régler la taille de la pile d'appels (NMAX) pour régler ce problème.
- Exemple d'un programme qui déclenche un stack overflow en C ?

Retourner une pile

Algorithme 6 Retourner abstrait

```
RetournerPile(p : Pile) : pile
▷ Entrée : P (une pile)
▷ Sortie : Q (une pile)
 Début
 tant que (EstVide(P) = faux)  faire
     Insertion(Suppression(P),Q)
 fin tant que
 Fin
```

Algorithme 7 Retourner concret

```
RetournerPile(p : Pile) : pile
▷ Entrée : P (une pile donnée par un tableau)
▷ Sortie : P (une pile)
 Début
 pour i = 0 à P.sommet/2  faire
     P.T[i] ↔ P.T[P.sommet - i]
 fin pour
 Fin
```

Retourner une pile

Algorithme 8 Retourner abstrait

```
RetournerPile(p : Pile) : pile
▷ Entrée : P (une pile)
▷ Sortie : Q (une pile)
 Début
 tant que (EstVide(P) = faux)  faire
     Insertion(Suppression(P),Q)
 fin tant que
 Fin
```

Algorithme 9 Retourner concret

```
RetournerPile(p : Pile) : pile
▷ Entrée : P (une pile donnée par un tableau)
▷ Sortie : P (une pile)
 Début
 pour i = 0 à P.sommet/2  faire
     P.T[i] ↔ P.T[P.sommet - i]
 fin pour
 Fin
```

Une File

- Comment modéliser une file d'attente ?
- Comment modéliser un buffer qui stocke les entrées claviers ?

Définition

Analogie avec une file d'attente :
FIFO (First In First Out ou **Premier Arrivé Premier Servi**)

- On rajoute un élément à la fin de la file
- On supprime l'élément qui est en tête de file.

Une File

- Comment modéliser une file d'attente ?
- Comment modéliser un buffer qui stocke les entrées claviers ?

Définition

Analogie avec une file d'attente :
FIFO (First In First Out ou **Premier Arrivé Premier Servi**)

- On rajoute un élément à la fin de la file
- On supprime l'élément qui est en tête de file.

Mise en œuvre

1. A l'aide d'un tableau (*nombre maximum d'éléments dans la file fixé*)

Type de données

```
Enregistrement File {
    T[NMAX] : entier;
    Début : entier;  Indice de l'élément le plus ancien de la file
    Taille : entier;  Taille de la file, l'indice du prochain élément à
insérer est début + taille
}
```

2. A l'aide d'une liste

Mise en œuvre : un tableau

Algorithme 10 La file est-elle vide ?

```
FileVide(f : File) : booléen
▷ Entrée : F (une file)
▷ Sortie : vrai si la file est vide, faux sinon.
Début
si (f.Taille = 0)
     retourner  vrai ;
sinon
     retourner  faux ;
fin si
Fin
```

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 11 La file est-elle pleine ?

```
FilePleine(f : File) : booléen
▷ Entrée : f (une file)
▷ Sortie : vrai si la file est pleine, faux sinon.
Début
si (f.Taille = NMAX)
    retourner vrai ;
sinon
    retourner faux ;
fin si
Fin
```

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 12 Insertion d'un élément

```
Insertion(f : File, elt : entier)
▷ Entrée : f (une file) et elt (un entier)
▷ Sortie : la file f dans laquelle elt a été inséré
Début
si (FilePleine(f) = faux )
    fin = f.Debut + f.taille mod NMAX ;
    f.T[fin] ← elt ;
    f.Taille++ ;
sinon
    Afficher un message d'erreur
fin si
Fin
```

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 13 Suppression d'un élément

```
Suppression(f : File) : entier
▷ Entrée : f (une file)
▷ Sortie : renvoie l'élément le plus ancien de la file f et supprime
l'élément de la file
▷ Variable locale :
      elt : entier ;
 Début
 si (FileVide(f) =  faux )
      elt ← f.T[f.Début] ;
      f.Taille = f.Taille - 1 ;
      f.Début ← (f.Début + 1) mod NMAX
       retourner elt ;
 sinon
      Afficher un message d'erreur
 fin si
 Fin
```

Complexité : $O(1)$

Une File de Priorité

Definition

Un ensemble dans lequel chaque élément possède une valeur et une priorité.

Opérations fondamentales

- **Insérer** : insère un nouvel élément dans l'ensemble.
- **Maximum** : renvoie l'élément de plus grande priorité.
- **Extraire_Max** : supprime de l'ensemble et renvoie l'élément de plus grande priorité.

Un tas permet de modéliser une file de priorité.

Un tas binaire

Definition

Un tableau T qui peut être vu comme un arbre A.

1. La **racine** du tas est en $T[1]$
2. Sachant l'indice i d'un élément (un **nœud**) du tas :
   - **Pere(i)** : élément d'indice $\lfloor \frac{i}{2} \rfloor$.
   - **Gauche(i)** : élément d'indice $2i$.
   - **Droit(i)** : élément d'indice $2i + 1$.

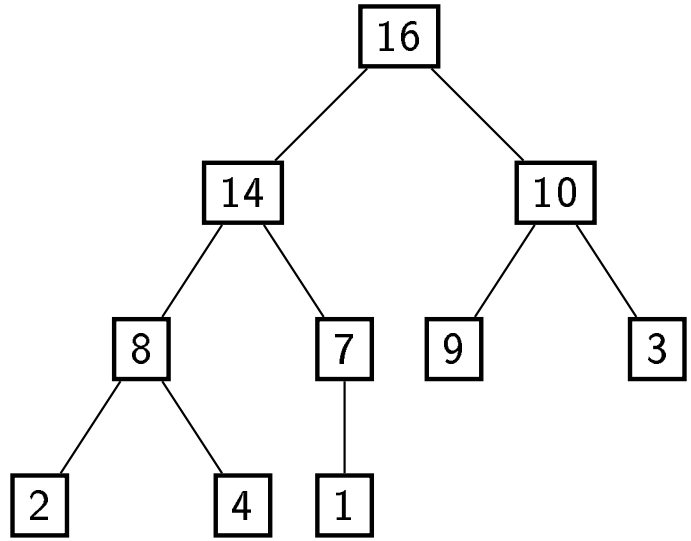
Propriété de tas

Pour chaque nœud i autre que la racine

$$T[Pere(i)] \geq T[i]$$

Un tas binaire

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1



Tas = File de Priorité

Le plus grand élément du tas : $T[1]$.

Maximum a une complexité de $\Theta(1)$.

Tas = File de Priorité

Algorithme 14 Extraire l'élément maximum

```
Extraire_Max(T : tableau) : entier
▷ Entrée : T (un tas binaire)
▷ Sortie : T est un tas binaire sans l'élément maximum et retourne
l'élément maximum
▷ Variables locales :
    max : entier ;
Début
    si (taille(T) < 1)
        Afficher un message d'erreur ;
    fin si
    max ← T[1] ;
    T[1] ← T[taille(T)] ;
    taille(T) = taille(T) - 1 ;
    Entasser(T,1) ;
    retourner max ;
Fin
```

Complexité : $O(\log n)$

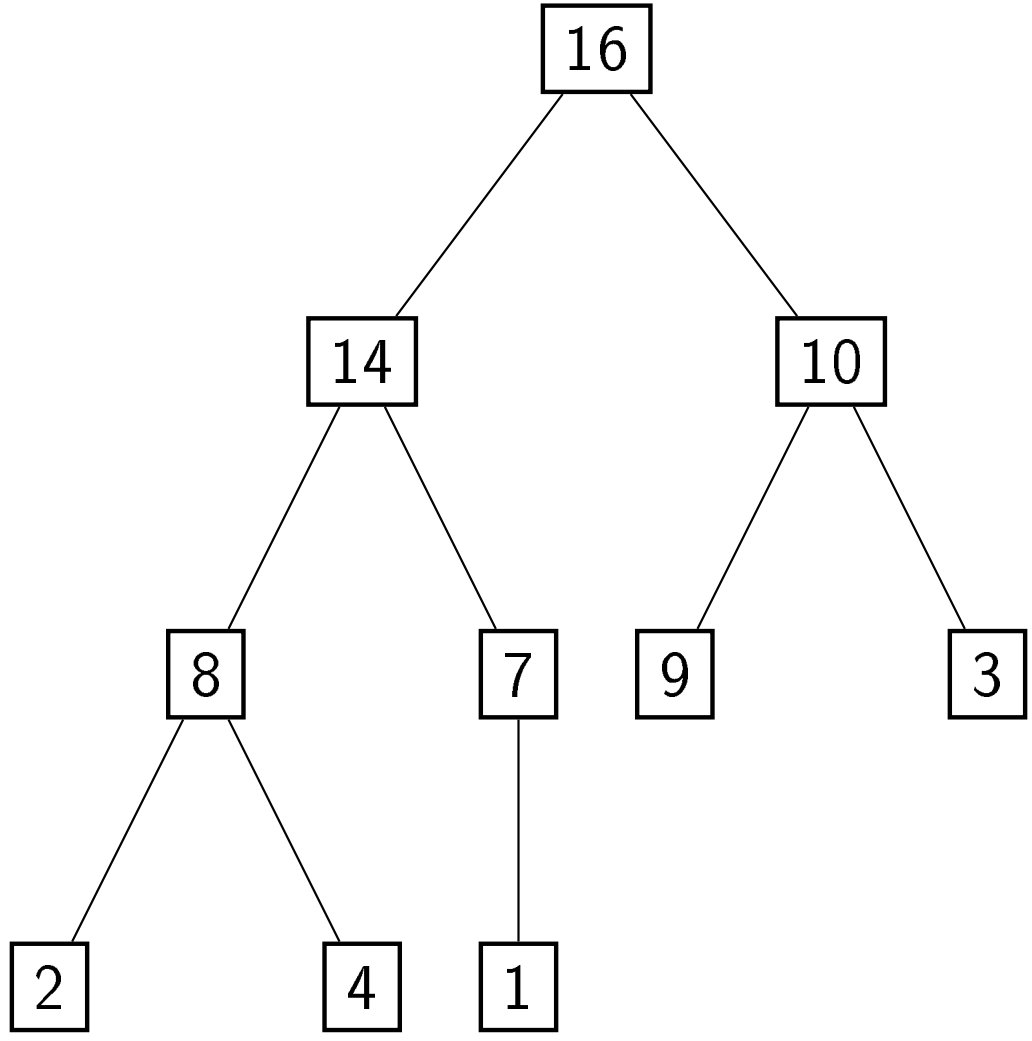
Tas = File de Priorité

Algorithme 15 Insérer un élément dans un tas

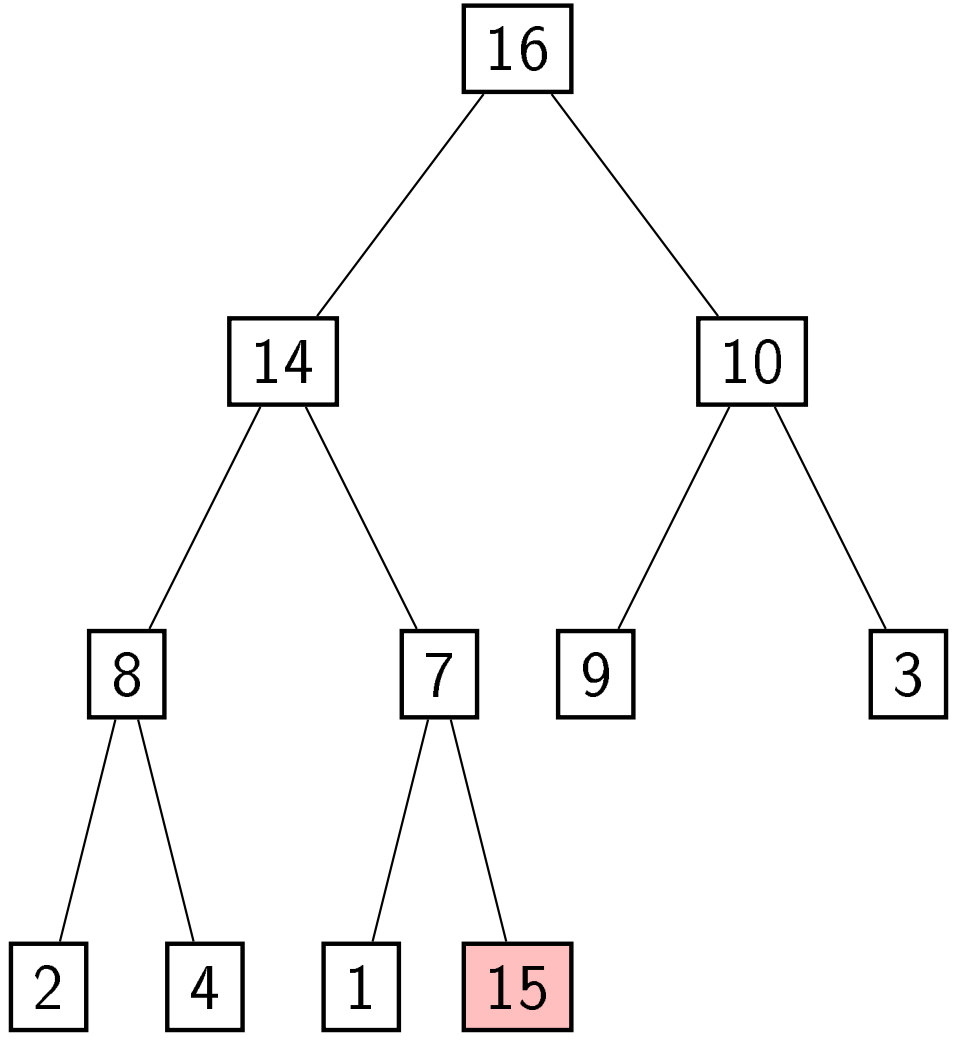
```
Insérer(T : tableau, p :entier)
▷ Entrée : T (un tas binaire)
▷ Sortie : T est un tas binaire contenant l'élément de priorité p
▷ Variables locales :
    i : entier ;
Début
    Taille(T) ← Taille(T) + 1 ;
    i ← Taille(T) ;
     tant que (i > 1 et T[pere(i)] < p)  faire
        T[i] ← T[Pere(i)] ;
        i ← Pere(i) ;
    T[i] ← p ;
Fin
```

Complexité : $O(\log n)$

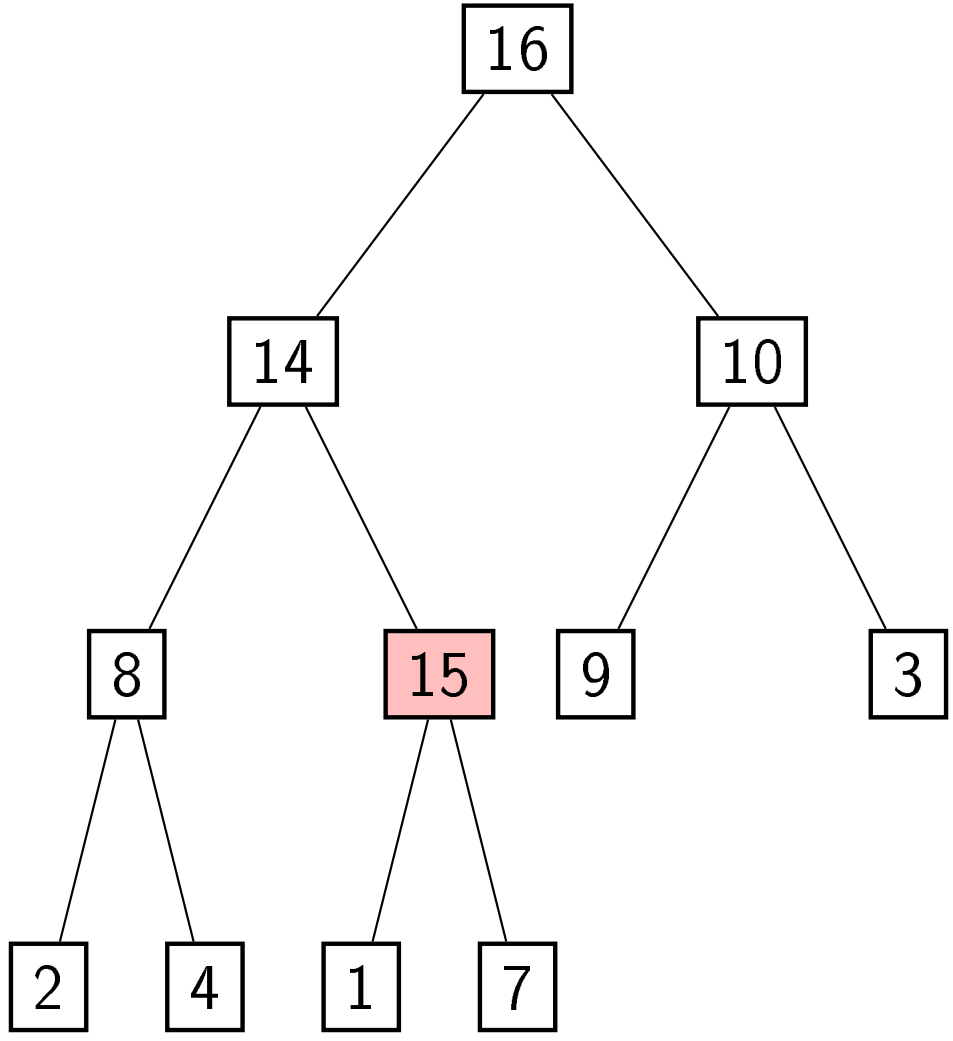
Exemple d'insertion : insertion du nœud 15



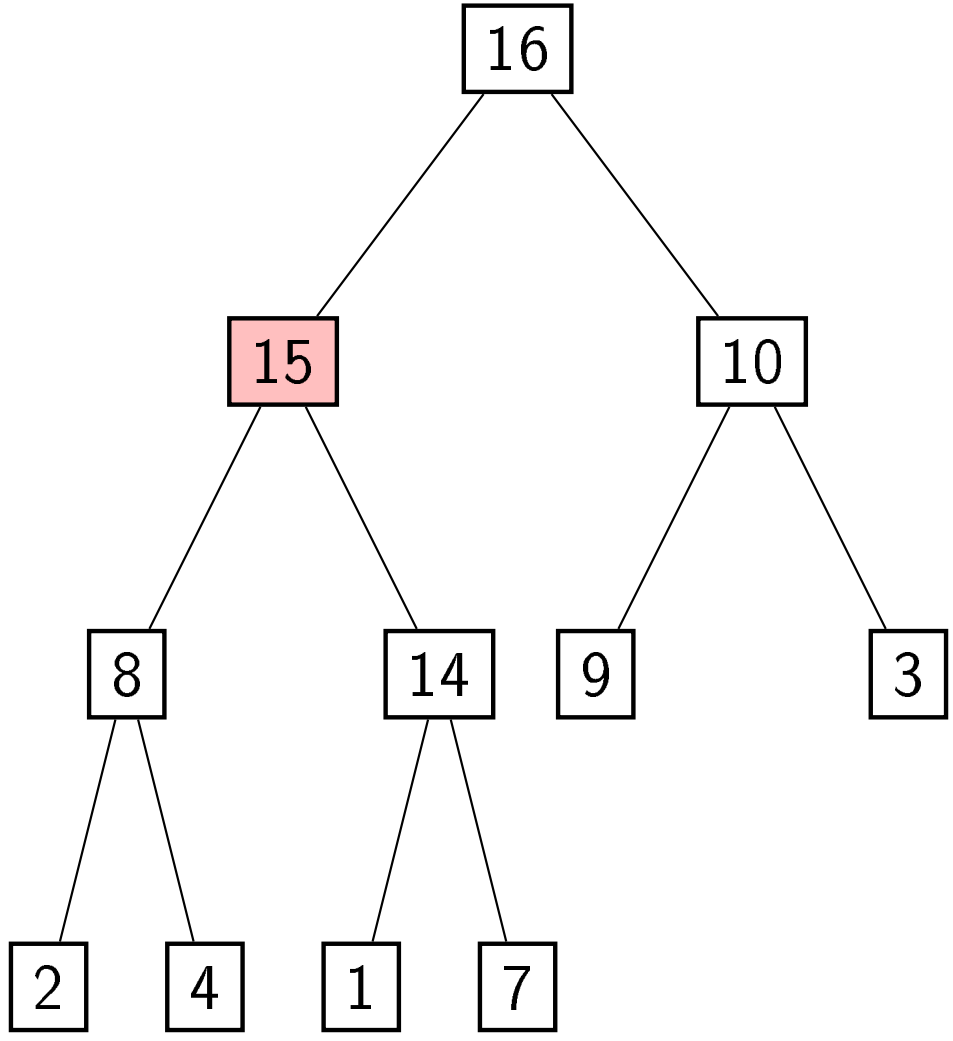
Exemple d'insertion : insertion du nœud 15



Exemple d'insertion : insertion du nœud 15



Exemple d'insertion : insertion du nœud 15



Utilisation des files de priorité

- Simulation à évènement discret
- Algorithme de Huffman (fusion des arbres)
- Pleins d'algos en réseau et système (scheduler) ...