Structures de données abstraites

Sandrine Vial
sandrine.vial@uvsq.fr

Septembre 2018

Structures de Données Abstraites

- Mise en œuvre d'un ensemble dynamique
- Définition de données (**structuration**)
- Définition des opérations pour **manipuler** les données.

Quelques structures classiques

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

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 ne peut prendre que l'élément qui est au dessus de la pile (élément le plus récemment inséré).

Une Pile

Opérations Principales

1. Insertion d'un élément dans une pile
2. Suppression d'un élément d'une pile
3. Création d'une pile vide
4. Tester si une pile est vide
5. Quel est l'élément du sommet d'une pile ?
6. ...

Mise en œuvre

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

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 1 La pile est-elle vide ?

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

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 2 La pile est-elle pleine ?

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

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 3 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*
Debut
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 4 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 ;
Debut
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 : $O(1)$

Une File

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

Opérations Principales

1. Insertion d'un élément
2. Suppression d'un élément (le plus ancien de la file)
3. Création d'une file vide
4. Tester si une file est vide
5. Quel est l'élément le plus ancien de la file ?
6. ...

Mise en œuvre

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

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 la plus ancien de la file
    Fin : entier;     Indice du prochain élément à insérer dans la file
}
```

2. A l'aide d'une liste

Mise en œuvre : un tableau

Algorithme 5 La file est-elle vide ?

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

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 6 La file est-elle pleine ?

FilePleine(f : File) : booléen
▷ ***Entrée : f (une file)***
▷ ***Sortie : vrai si la file est pleine, faux sinon.***
Debut
si (f.Début = (f.Fin +1) mod NMAX)
    retourner vrai ;
sinon
    retourner faux ;
fin si
Fin

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 7 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*
Debut
si (FilePleine(f) = faux)
f.T[f.Fin] $\leftarrow$ **elt ;**
f.Fin $\leftarrow$ **(f.Fin + 1) mod NMAX ;**
sinon
Afficher un message d'erreur
fin si
Fin

Complexité : $O(1)$

Mise en œuvre : un tableau

Algorithme 8 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 ;
Debut
si (FileVide(f) = faux)
elt ← f.T[f.Début] ;
f.Début ← (f.Début + 1) mod NMAX ;
retourner elt ;
sinon
Afficher un message d'erreur
fin si
Fin

Complexité : $O(1)$

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 $2i + 1$.

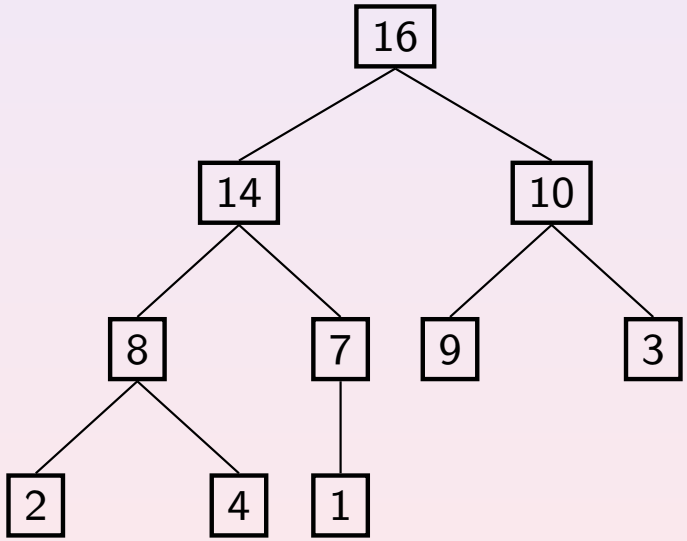
Propriété de tas

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



Un Tas Binaire : les opérations élémentaires

1. **Entasser** : sert à garantir le maintien de la propriété de tas binaire.
2. **Construire_Tas** : transforme un tableau non ordonné en tas binaire.
3. ...

Algorithme 9 Entasser

Entasser(T : tableau, i : entier)

▷ ***Entrée : T (un tableau) et i un indice du tableau***

▷ ***Sortie : l'élément d'indice i est la racine d'un tas binaire***

▷ ***Pré-Conditions : Les éléments d'indices Gauche(i) et Droit(i) sont les racines de deux tas binaires.***

▷ ***Variables locales :***

l,r,max : entier ;

Debut
l ← Gauche(i) ;
r ← Droit(i) ;
si (l ≤ taille (T) et T[l] > T[i])
max ← l ;
sinon
max ← i ;
fin si
si (r ≤ taille (T) et T[r] > T[max])
max ← r ;
fin si
si (max ≠ i)
T[i] ↔ T[max] ;
Entasser(T,max) ;
fin si
Fin

Algorithme 10 Construction d'un tas binaire à partir d'un tableau

```
Construire_Tas(T : tableau, i : entier)
▷ Entrée : T (un tableau)
▷ Sortie : T est un tas binaire
▷ Variables locales :
    i : entier ;
Debut
pour i de ⌊ longueur(T)/2 ⌋ à 1  faire
    Entasser(T,i) ;
fin pour
Fin
```

Complexité : $O(n)$

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.

Tas = File de Priorité

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

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

Tas = File de Priorité

Algorithme 11 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 ;
Debut
    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 : $O(\log n)$

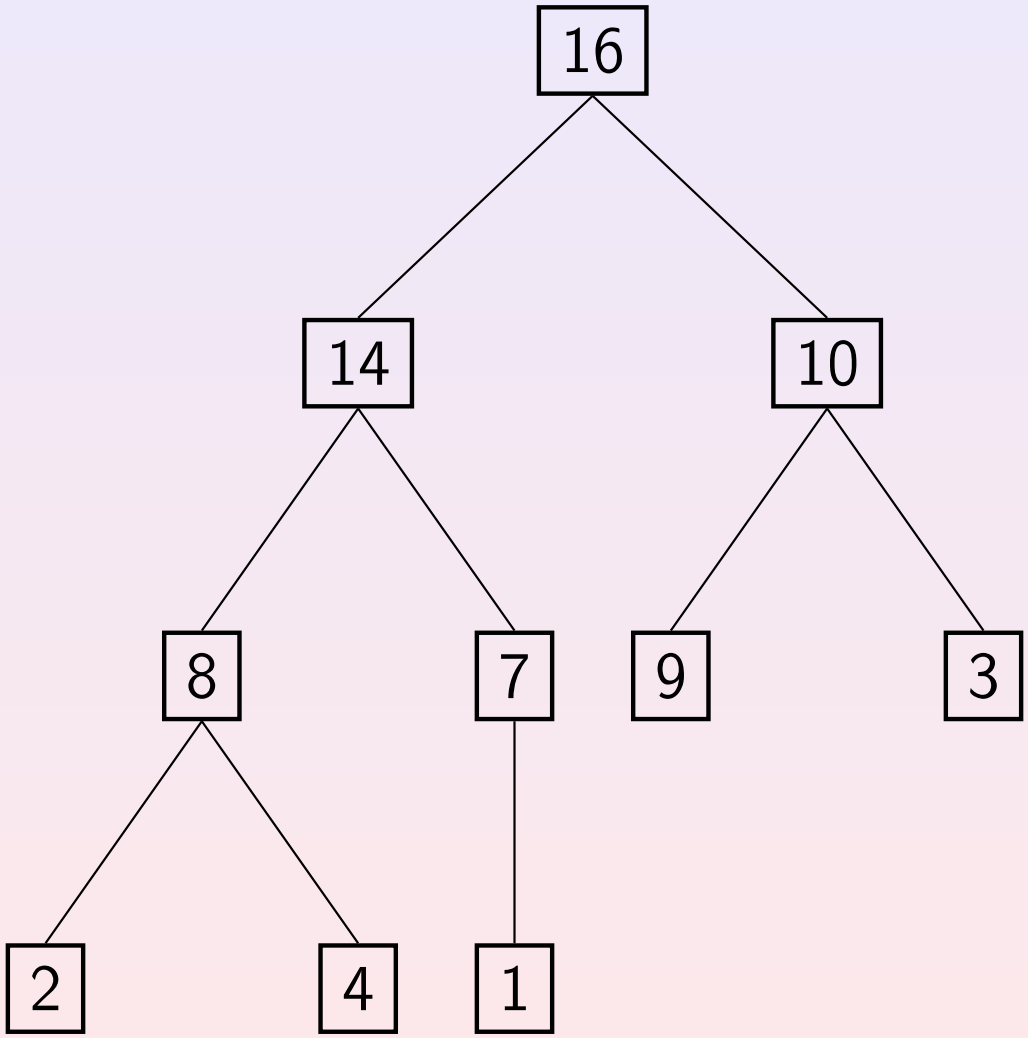
Tas = File de Priorité

Algorithme 12 Insérer un élément dans un tas

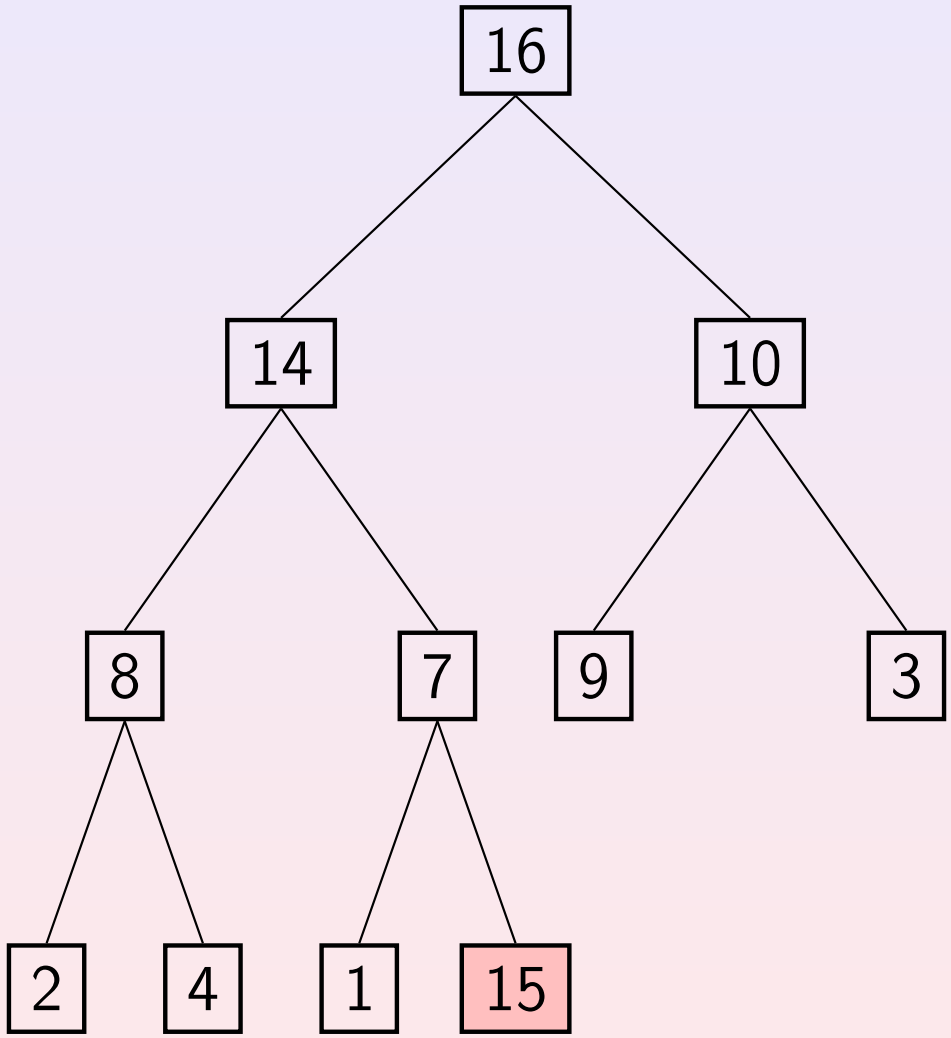
```
Insérer(T : tableau, p :entier)
▷ Entrée : T (un tas binaire)
▷ Sortie : T est un tas binaire contenant l'élément de priorité p
▷ Variables locales :
    i : entier ;
Debut
    Taille(T) ← Taille(T) + 1 ;
    i ← Taille(T) ;
     tant que (i > 1 et T[pere(i)] < p)  faire
        T[i] ← T[Pere(i)]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

Complexité : $O(\log n)$

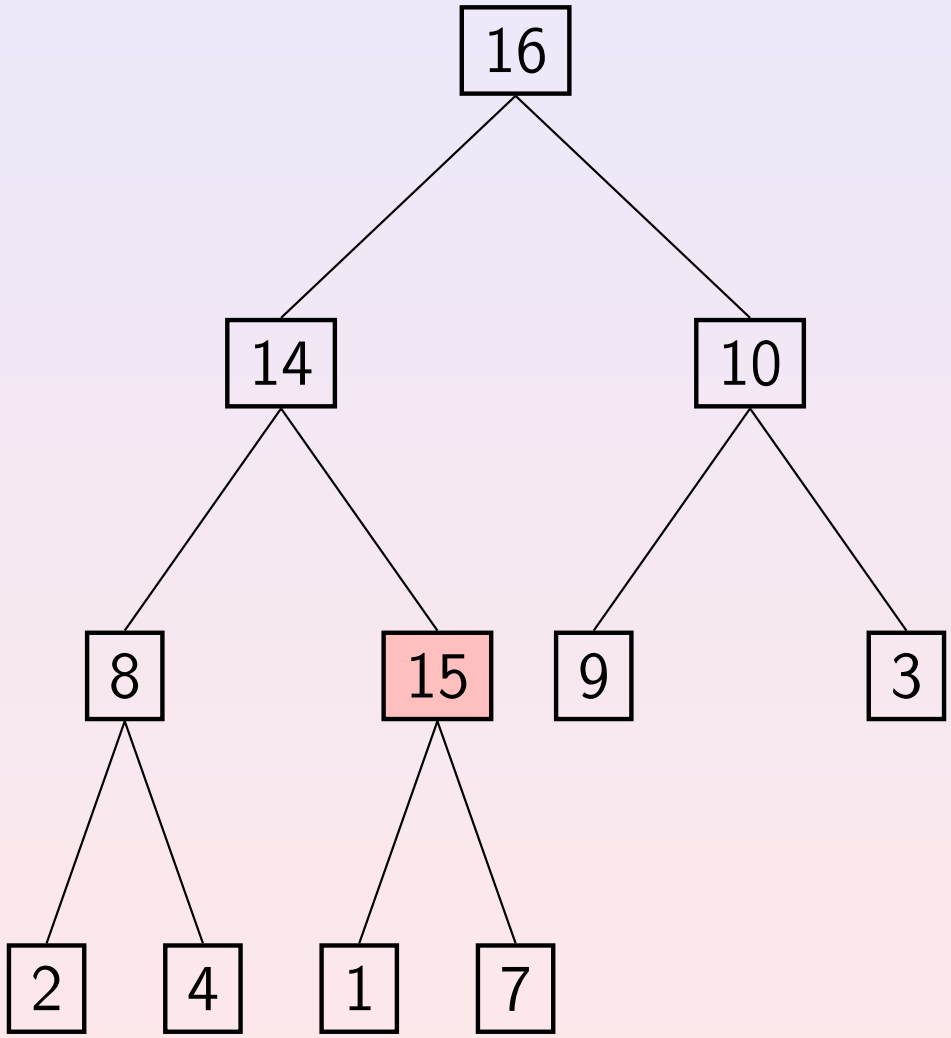
Exemple d'insertion : insertion du nœud 15



Exemple d'insertion : insertion du nœud 15



Exemple d'insertion : insertion du nœud 15



Exemple d'insertion : insertion du nœud 15