

# Algorithmique et structures de données

File de priorité - Fast marching

---

Julien Hauret

Lundi 20 février 2022

# Plan de la séance

File de priorité et tri par tas

Tri par tas

TP

L'objectif est de créer une file dont les éléments sont retirés en fonction de la **priorité** (un score) qui leur est attribué.

Il s'agit donc de maintenir une file triée (par ordre de priorité) lors de l'ajout ou de la suppression d'un élément.

## Compromis

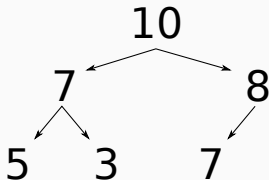
- Accéder en  $O(1)$ , implique une insertion en  $O(N)$
- Insérer en  $O(1)$ , implique un accès en  $O(N)$

## Solution

On fait le choix d'une insertion et d'un accès en  $O(\log N)$ . On utilise pour ce faire une structure d'**arbre binaire** équilibré, i.e. un arbre dans lequel chaque noeud possède au plus 2 fils. On construit l'arbre de sorte que chaque parent a une priorité supérieure à celle de ses fils.

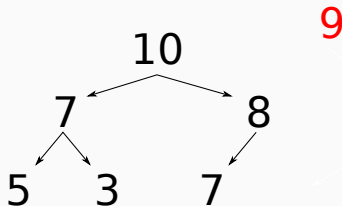
## File de priorité : insertion (push)

Le nouvel élément est inséré dans le sous-arbre de profondeur minimale, puis échangé avec ses parents si sa priorité est supérieure à la leur.



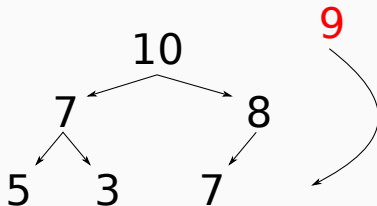
## File de priorité : insertion (push)

Le nouvel élément est inséré dans le sous-arbre de profondeur minimale, puis échangé avec ses parents si sa priorité est supérieure à la leur.



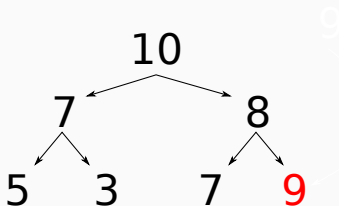
## File de priorité : insertion (push)

Le nouvel élément est inséré dans le sous-arbre de profondeur minimale, puis échangé avec ses parents si sa priorité est supérieure à la leur.



## File de priorité : insertion (push)

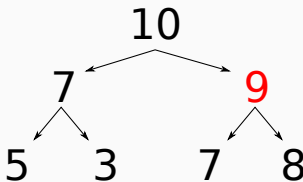
Le nouvel élément est inséré dans le sous-arbre de profondeur minimale, puis échangé avec ses parents si sa priorité est supérieure à la leur.





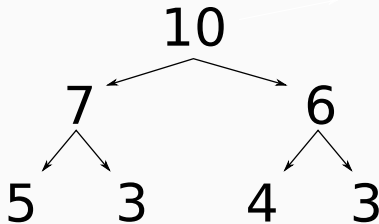
## File de priorité : insertion (push)

Le nouvel élément est inséré dans le sous-arbre de profondeur minimale, puis échangé avec ses parents si sa priorité est supérieure à la leur.



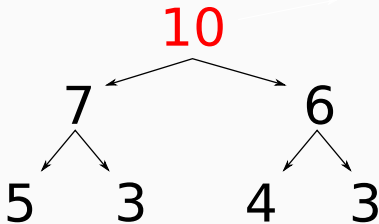
## File de priorité : retrait (pop)

La racine de l'arbre est retirée. L'élément le plus profond est placé à la racine puis échangé avec son fils de priorité maximale si celle-ci est supérieure à la sienne.



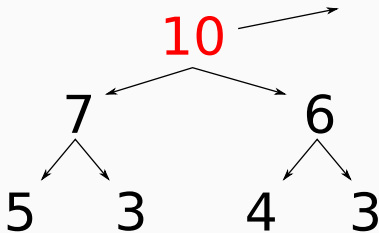
## File de priorité : retrait (pop)

La racine de l'arbre est retirée. L'élément le plus profond est placé à la racine puis échangé avec son fils de priorité maximale si celle-ci est supérieure à la sienne.



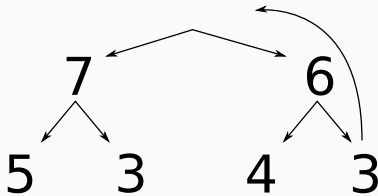
## File de priorité : retrait (pop)

La racine de l'arbre est retirée. L'élément le plus profond est placé à la racine puis échangé avec son fils de priorité maximale si celle-ci est supérieure à la sienne.



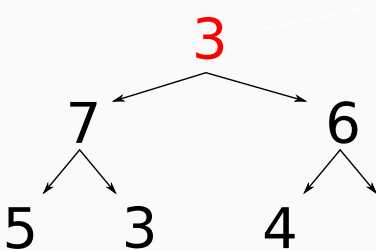
## File de priorité : retrait (pop)

La racine de l'arbre est retirée. L'élément le plus profond est placé à la racine puis échangé avec son fils de priorité maximale si celle-ci est supérieure à la sienne.



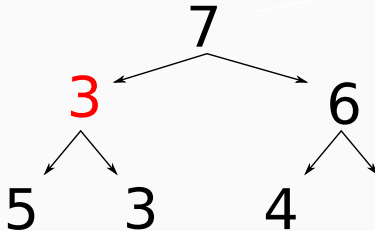
## File de priorité : retrait (pop)

La racine de l'arbre est retirée. L'élément le plus profond est placé à la racine puis échangé avec son fils de priorité maximale si celle-ci est supérieure à la sienne.



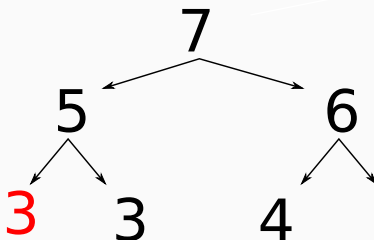
## File de priorité : retrait (pop)

La racine de l'arbre est retirée. L'élément le plus profond est placé à la racine puis échangé avec son fils de priorité maximale si celle-ci est supérieure à la sienne.



## File de priorité : retrait (pop)

La racine de l'arbre est retirée. L'élément le plus profond est placé à la racine puis échangé avec son fils de priorité maximale si celle-ci est supérieure à la sienne.

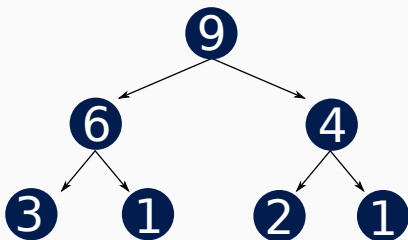




# Stockage

La file de priorité est stockée dans un tableau, construit de la façon suivante :

En commençant à l'indice 1, les fils du nœud  $i$  sont placés aux indices  $2i$  et  $2i + 1$ .

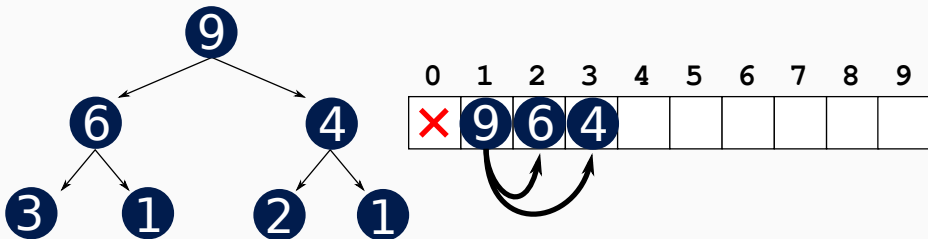


0	1	2	3	4	5	6	7	8	9
×	9								

# Stockage

La file de priorité est stockée dans un tableau, construit de la façon suivante :

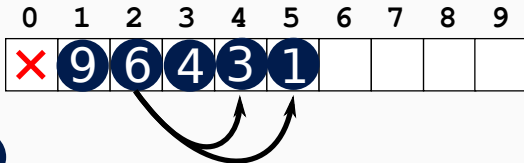
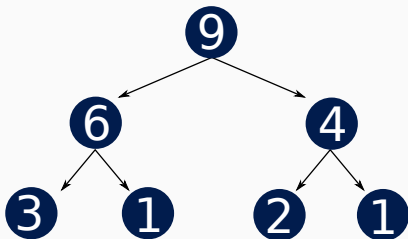
En commençant à l'indice 1, les fils du nœud  $i$  sont placés aux indices  $2i$  et  $2i + 1$ .



# Stockage

La file de priorité est stockée dans un tableau, construit de la façon suivante :

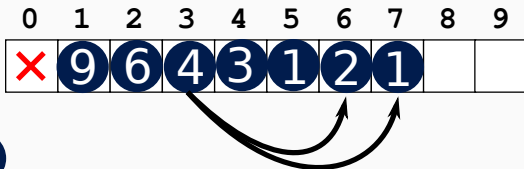
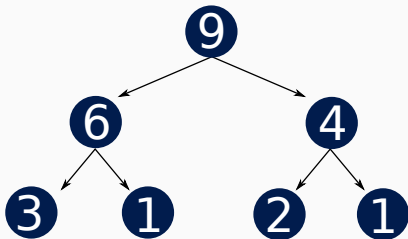
En commençant à l'indice 1, les fils du nœud  $i$  sont placés aux indices  $2i$  et  $2i + 1$ .



# Stockage

La file de priorité est stockée dans un tableau, construit de la façon suivante :

En commençant à l'indice 1, les fils du nœud  $i$  sont placés aux indices  $2i$  et  $2i + 1$ .



# Plan de la séance

File de priorité et tri par tas

Tri par tas

TP

# HeapSort

**HeapSort** remplit une file de priorité et puis retire les éléments un par un.

```
void HeapSort(std::vector<double> &v){  
    FilePriorite f;  
    for(int i=0; i<v.size(); i++){  
        f.push(v[i]);  
    }  
    for(int i=0; i<v.size(); i++){  
        v[i] = f.pop();  
    }  
}
```

# Conclusion

HeapSort est un tri en  $O(N \log N)$  **dans tous les cas**. Cependant en comparaison à QuickSort, il utilise plus de mémoire et est plus long en moyenne.

En pratique, QuickSort est le tri le plus utilisé.

- Tri :  $O(N \log N)$
- Recherche dans un tableau trié :  $O(\log N)$
- Recherche dans un tableau non trié :  $O(N)$



# Plan de la séance

File de priorité et tri par tas

Tri par tas

TP

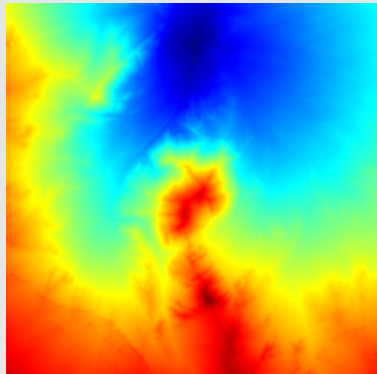
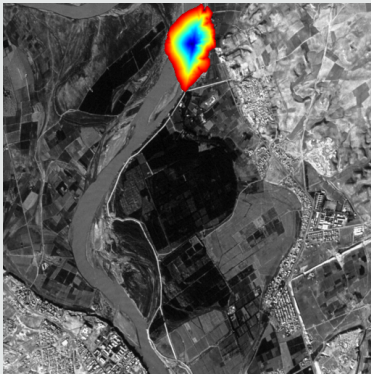
## TP

Deux parties :

- Implémentation d'une file de priorité
- Application au fast marching

# Fast marching

## Calcul rapide de cartes de distances



# Fast marching

## Calcul de plus court chemin

