PARTIE I

LE TYPE ABSTRAIT FILE DE PRIORITÉ

RAPPEL : TYPES ABSTRAITS / STRUCTURES DE DONNÉES

TYPES ABSTRAITS	STRUCTURES DE DONNÉES
PILE, ENSEMBLE, FILE, DICTIONNAIRE, ? FILE DE PRIORITÉ, LISTE PYTHON	tableau, vecteur, tableau dynamique, table de hachage, liste chaînée, arbre binaire
spécification formelle ensemble de données + opérations possibles ≃ interface	manière concrète d'organiser les données ≃ mécanisme

implémentation

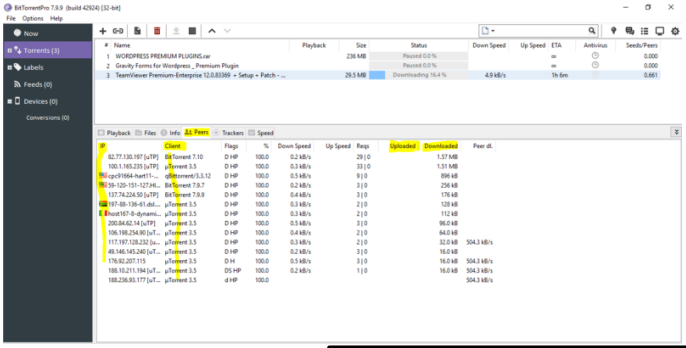
OBJECTIF : Trouver la structure de données qui optimise la complexité asymptotique de chaque opération

C'EST QUOI UNE FILE DE PRIORITÉ ?

EXEMPLE MOTIVATIONNEL

Objectif : Développer un client pour télécharger des torrents

→ on veut choisir rapidement les utilisateurs qui ont le ⊛ de bande passante



Peers

106.198.254.90 : 100 Ko/s

200.84.62.14 : 251 Ko/s

...

Opérations :

→ Utilisateur bande passante max ?

→ Ajouter/supprimer un utilisateur

→ Changer valeur bande passante utilisateur

FILE DE PRIORITÉ (BASIQUE)

On ne s'embêtera pas avec des clés/valeurs

OPÉRATION 1 Créer une file de priorité vide

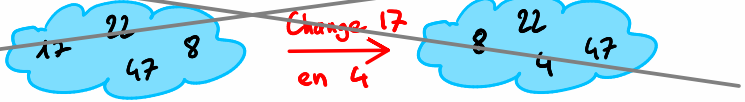
OPÉRATION 2 Ajouter un entier dans la file

OPÉRATION 3 Renvoyer le minimum

On peut remplacer "minimum" par "maximum"

OPÉRATION 4 Supprimer le minimum

~~**OPÉRATION 5** Modifier la valeur d'un entier~~ (enlevé par souci de ~~simplicité~~)



FILE DE PRIORITÉ "ENRICHIE"

On ne verra pas ce cours-ci :

FILE DE PRIORITÉ
Opérations : - Créer une file de priorité vide - Ajouter un couple clé/valeur - Renvoyer la clé de valeur minimale - Supprimer le couple clé/valeur de valeur minimale - Modifier la valeur d'une clé

Application en théorie des graphes :

→ Algorithme de Dijkstra
→ Algorithme de Prim

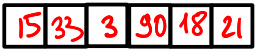
Comment on ferait : Ajouter un dictionnaire en plus de la structure qu'on verra aujourd'hui

PARTIE II

IMPLÉMENTATIONS SOUS-OPTIMALES

PREMIÈRE IDÉE : TABLEAU DYNAMIQUE

tableau

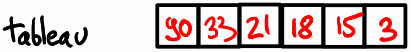


Opérations	Complexité
Créer une file de priorité vide	$O(1)$
Ajouter un entier dans la file	$O(1)^*$
Renvoyer le minimum Il faut parcourir le tableau !	$O(n)$
Supprimer le minimum On échange avec l'élément final et on supprime	$O(n)$ $O(1)$ si on connaît la position du minimum

n = nombre d'éléments

* : complexité amortie

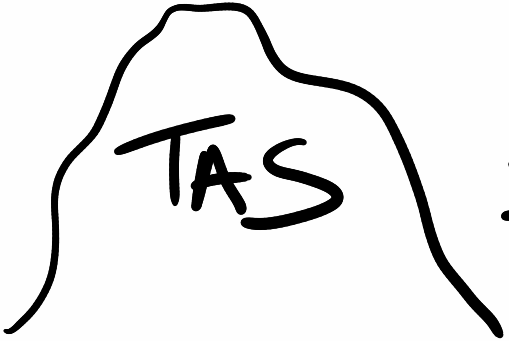
DEUXIÈME IDÉE : TABLEAU DYNAMIQUE TRIÉ (À L'ENVERS)



Opérations	Complexité
Créer une file de priorité vide	$O(1)$
Ajouter un entier dans la file Il faut décaler les entiers comme dans un tri insertion	$O(n)$
Renvoyer le minimum C'est le dernier !	$O(1)$
Supprimer le minimum	$O(1)$

n = nombre d'éléments

PARTIE III



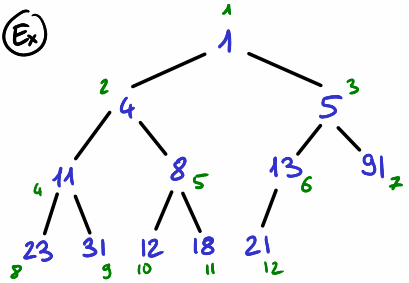
BINAIRES

TAS BINAIRE : DÉFINITION

Définition

tas binaire min = arbre binaire
- presque complet (rempli à tous les niveaux sauf éventuellement le dernier niveau, rempli de la gauche jusqu'à un certain point)
- où chaque noeud a une valeur plus petite que celles de ses enfants

Ex



En machine, sous forme de tableau

	1	2	3	4	5	6	7	8	9	10	11	12
tableau	1	4	5	11	8	13	91	23	31	12	18	21

Astuce

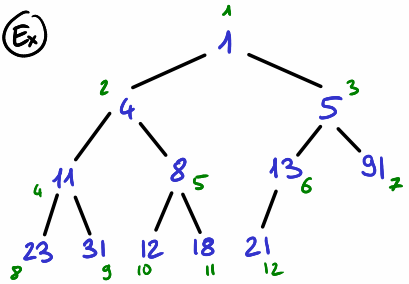
Si les indices du tableau vont de 1 à n,
- un noeud en position i a son enfant gauche en position $2i$
- un noeud en position i a son enfant droit en position $2i+1$
- un noeud en position i a son parent en position $\lfloor i/2 \rfloor$

TAS BINAIRE : DÉFINITION

Définition

tas binaire min = arbre binaire
- presque complet (rempli à tous les niveaux sauf éventuellement le dernier niveau, rempli de la gauche jusqu'à un certain point)
- où chaque noeud a une valeur plus petite que celles de ses enfants

Ex



Vu qu'il est presque complet, la hauteur de l'arbre vaut $\lfloor \log_2(\text{nb d'éléments}) \rfloor$

Preuve Un arbre presque complet de hauteur h avec n noeuds satisfait

$$1 + 2 + 2^2 + \dots + 2^{h-1} < n \leq 1 + 2 + 2^2 + \dots + 2^h$$

soit

$$2^h - 1 < n \leq 2^{h+1} - 1$$

ou

$$2^h \leq n < 2^{h+1}$$

et donc $h \leq \log_2(n) < h+1$. ∎

OPÉRATION 1 : CRÉATION D'UNE FILE VIDE

Dans notre tête

Dans la mémoire de l'ordinateur

tableau

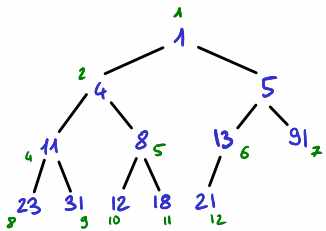
Création d'une file de priorité vide :

- Allouer un tableau dynamique vide

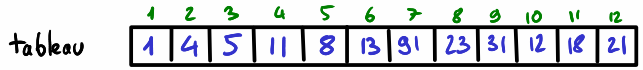
Complexité : $O(1)$

OPÉRATION 2 : AJOUT D'UN ÉLÉMENT

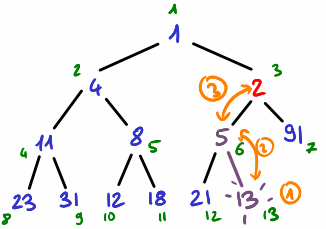
Dans notre tête

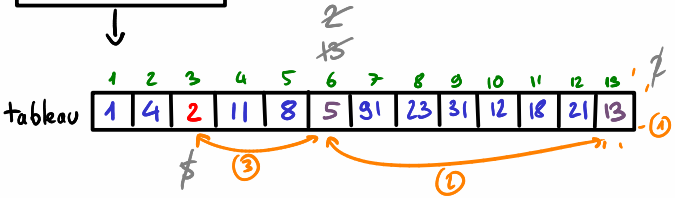


Dans la mémoire de l'ordinateur



Ajout de 2





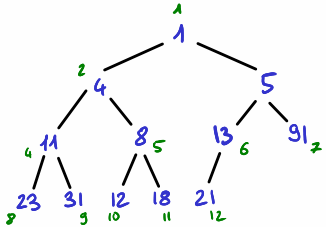
Instructions :

- Placer x à la fin de tableau
- Si x est plus grand que son parent, échanger les 2 nombres
- Ainsi de suite, jusqu'à que x soit à la racine ou plus petit que son parent

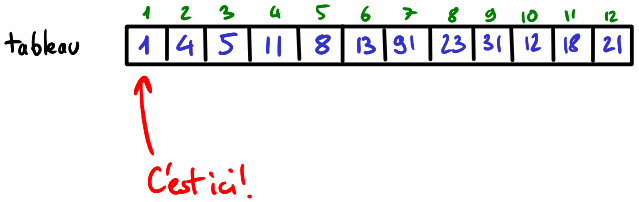
Complexité (amortie) : $O(\log(\text{nb éléments}))$

OPÉRATION 3 : MINIMUM

Dans notre tête



Dans la mémoire de l'ordinateur



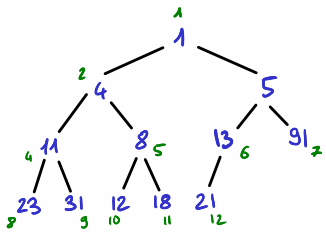
Instructions :

- Renvoyer le premier élément du tableau

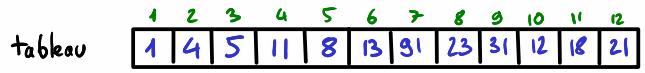
Complexité : $O(1)$

OPÉRATION 4 : SUPPRIMER LE MINIMUM

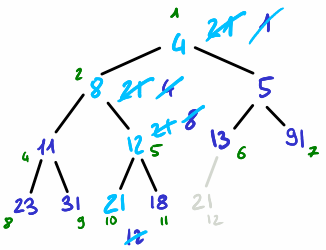
Dans notre tête

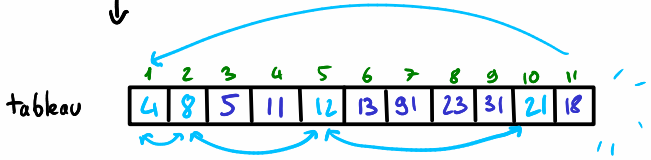


Dans la mémoire de l'ordinateur



Suppression du minimum





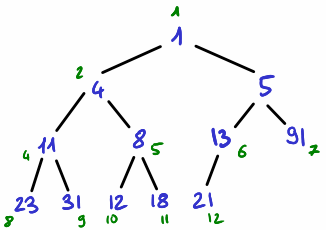
Instructions :

- Écraser la première valeur du tableau par la dernière (on la notera y)
- Supprimer le dernier élément du tableau
- Tant que y est plus grand que ses enfants, le descendre d'un cran en l'échangeant avec son plus petit enfant.

Complexité : $O(\log(\text{nb éléments}))$

EN RÉSUMÉ

Dans notre tête



Dans la mémoire de l'ordinateur

	1	2	3	4	5	6	7	8	9	10	11	12
tableau	1	4	5	11	8	13	91	23	31	12	18	21

Opérations	Complexité
Créer une file de priorité vide	$O(1)$
Ajouter un entier dans la file On le rajoute comme une feuille et on le fait remonter dans l'arbre	$O(\log(n))^*$
Renvoyer le minimum	$O(1)$
Supprimer le minimum On remplace la valeur de la racine par celle de la dernière feuille et on la fait descendre	$O(\log(n))$

n = nombre d'éléments

$*$: complexité amortie