



LIFAPSD – Algorithmique, Programmation et Structures de données

Nicolas Pronost



Chapitre 6

Tableau Dynamique

Rappels sur les tableaux

- Qu'est ce que ce programme affiche?

```
int * tab = new int [5];  
cout << sizeof(tab);
```

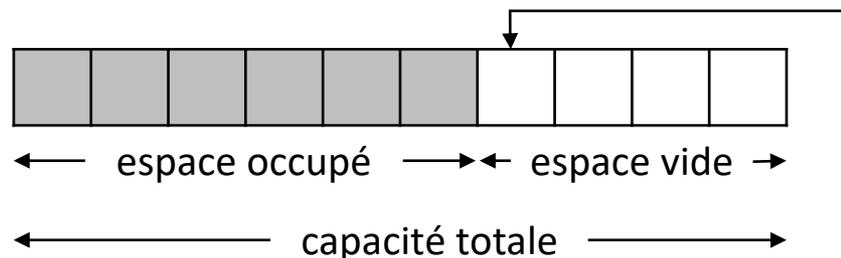
- Peut-on donc écrire la procédure d'affichage suivante?

```
void afficheTableau (const int * tab) {  
    for (unsigned int i = 0; i < sizeof(tab); i++) {  
        cout << tab[i] << " ";  
    }  
}
```

- Que faut-il modifier dans `afficheTableau` pour pouvoir afficher correctement le contenu du tableau?

Rappels sur les tableaux

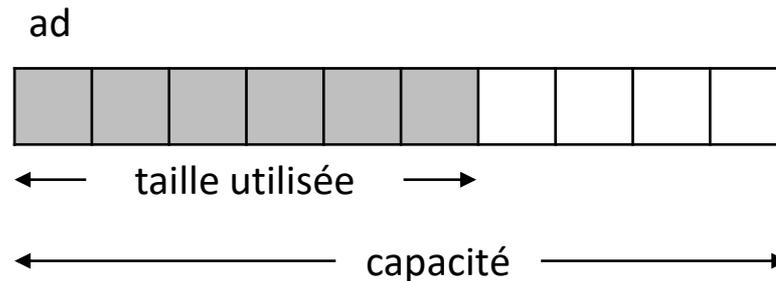
- Les éléments d'un tableau sont stockés de manière contigüe en mémoire
- On y accède directement
 - opérateur [] : calcul du décalage à partir du premier élément grâce à l'arithmétique des pointeurs
- La capacité est connue à la création du tableau
 - `int tab [5]; // création sur la pile`
 - `int * tab = new int [5]; // création sur le tas`
- En pratique, il y a souvent 2 parties distinctes
 - une partie remplie d'éléments
 - une partie 'vide' (i.e. contenu non significatif)



emplacement du
prochain ajout

Tableau Dynamique

- Un tableau dynamique est un **type de donnée abstrait**
- On utilise un tableau dynamique comme un tableau normal (statique) **sans se préoccuper de sa taille**
- L'implémentation du TDA s'occupe d'augmenter et diminuer la taille du tableau quand nécessaire
- L'interface n'en fait rien transparaître, c'est un mécanisme invisible pour l'utilisateur



Module Tableau Dynamique

Module TableauDynamique {un tableau de taille variable}

- **Importer:**

- Module ElementTD

- **Exporter:**

- Type TableauDynamique
 - Constructeur TableauDynamique()
 - Postconditions : réservation d'un tableau de 1 ElementTD sur le tas
 - Constructeur TableauDynamique(taille : entier)
 - Postconditions : réservation d'un tableau de taille ElementTD sur le tas
 - Destructeur ~TableauDynamique()
 - Postconditions : libération de la mémoire utilisée sur le tas
 - ...

Module Tableau Dynamique

- ...
- **Procédure** ajouterElement (e: ElementTD)
 - Postcondition : une copie de e est insérée à la fin du tableau, extension de l'espace mémoire alloué au tableau si nécessaire
 - Paramètre en mode donnée : e
- **Fonction** valeurIemeElement (i: entier positif) : ElementTD
 - Précondition : i est positif et plus petit que le nombre d'éléments
 - Résultat : retourne l'élément d'indice i
 - Paramètre en mode donnée : i
- **Procédure** modifierValeurIemeElement (e: ElementTD, i : entier positif)
 - Précondition : i est positif et plus petit que le nombre d'éléments
 - Postcondition : l'élément d'indice i vaut e
 - Paramètre en mode donnée : e, i
- **Procédure** afficher ()
 - Postcondition : les éléments du tableau sont affichés sur la sortie standard
- **Procédure** supprimerElement (i: entier positif)
 - Précondition : le tableau est non vide
 - Postcondition : l'élément d'indice i est supprimé du tableau
 - Paramètre en mode donnée : i
- **Procédure** insererElement (e: ElementTD, i : entier positif)
 - Précondition : i est positif et plus petit que le nombre d'éléments
 - Postcondition : e est inséré à l'indice i
 - Paramètre en mode donnée : e, i

Module Tableau Dynamique

- **Implémentation:**

- **Type** TableauDynamique = Classe
 - ad : pointeur sur ElementTD
 - capacite : entier
 - taille_utilisee : entier
 - **Constructeur** TableauDynamique()
 - Début
 - ad ← **réserve** tableau [1..1] de ElementTD
 - capacite ← 1
 - taille_utilisee ← 0
 - Fin
 - **Destructeur** ~TableauDynamique()
 - Début
 - libère** ad
 - capacite ← 0
 - taille_utilisee ← 0
 - Fin
 - **Etc.**
- Fin classe

FinModule TableauDynamique

Mise en œuvre en C++

- On peut définir un ElementTD par exemple comme un entier (TD/TP)

```
ElementTD.h
#ifndef _ELEMENT_TD
#define _ELEMENT_TD

typedef int ElementTD;

void afficheElementTD (ElementTD e);
/* Preconditions : aucune */
/* Post-conditions : affichage de e sur la sortie standard */

#endif
```

```
ElementTD.cpp
#include <iostream>
#include "ElementTD.h"

void afficheElementTD (ElementTD e) {
    std::cout << e;
}
```

Mise en œuvre en C++

TableauDynamique.h

```
#ifndef _TAB_DYN
#define _TAB_DYN

#include "ElementTD.h"

class TableauDynamique {
public:
    unsigned int capacite;
    unsigned int taille_utilisee;
    ElementTD * ad;

    TableauDynamique ();
    TableauDynamique (unsigned int taille);
    ~TableauDynamique ();

    void ajouterElement (ElementTD e);
    ElementTD valeurIemeElement (unsigned int indice) const;
    void modifierValeurIemeElement (ElementTD e, unsigned int indice);
    void afficher () const;
    void supprimerElement (unsigned int indice);
    void insererElement (ElementTD e, unsigned int indice);
};

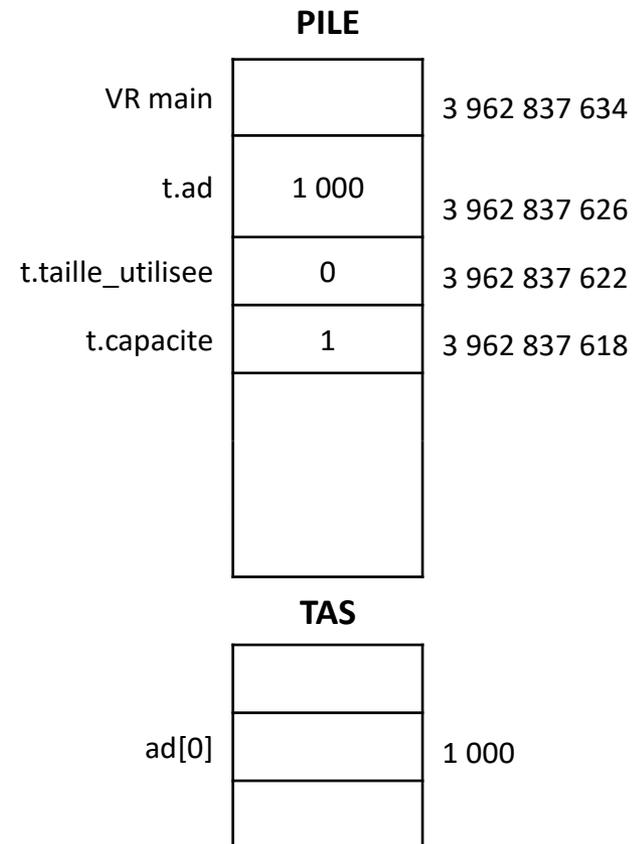
#endif
```

Création d'un tableau dynamique

- A la création (appel au constructeur), on alloue de la place sur le tas

```
TableauDynamique::TableauDynamique () {  
    ad = new ElementTD [1];  
    capacite = 1;  
    taille_utilisee = 0;  
}
```

```
int main () {  
    TableauDynamique t;  
    return 0;  
}
```



Ajout dans un tableau dynamique

- Que fait-on quand on n'a plus de place pour ajouter un nouvel élément?
 - C'est-à-dire quand `taille_utilisee` vaut `capacite`
- On doit choisir de combien d'emplacements on veut augmenter le tableau en mémoire
- En CM, TD et TP, on choisit de **doubler la capacité** quand on n'a plus de place pour ajouter un nouvel élément
 - Exercice en TD sur l'extension avec un nombre constant d'emplacements
- En commençant avec une capacité de 1, la capacité est toujours une puissance de 2

Ajout dans un tableau dynamique

- Création d'un tableau dynamique



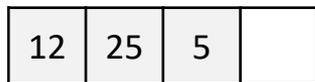
- Ajout de l'élément 12



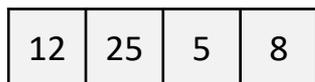
- Ajout de l'élément 25



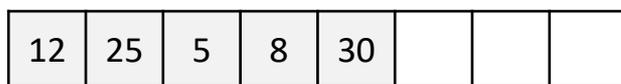
- Ajout de l'élément 5



- Ajout de l'élément 8



- Ajout de l'élément 30



Pas de place pour ajouter 25 !

- création d'un nouveau tableau 2x plus grand
- recopie de l'ancien dans le nouveau
- ajout de la valeur 25

Pas de place pour ajouter 5 !

- création d'un nouveau tableau 2x plus grand
- recopie de l'ancien dans le nouveau
- ajout de la valeur 5

Pas de place pour ajouter 30 !

- création d'un nouveau tableau 2x plus grand
- recopie de l'ancien dans le nouveau
- ajout de la valeur 30

Ajout dans un tableau dynamique

```
void TableauDynamique::ajouterElement (ElementTD e) {
    if (taille_utilisee == capacite) { // tableau plein, on double la capacité
        ElementTD * temp = ad;
        ad = new ElementTD [2*capacite];
        capacite *= 2;
        for (unsigned int i = 0; i < taille_utilisee; i++) ad[i] = temp[i];
        delete [] temp;
    }
    ad[taille_utilisee] = e;
    taille_utilisee++;
}
```

Destruction d'un tableau dynamique

```
TableauDynamique::~~TableauDynamique () {  
    if (ad != nullptr) {  
        delete [] ad;  
        ad = nullptr;  
    }  
    capacite = 0;  
    taille_utilisee = 0;  
}
```

Etude des coûts

	Tableau statique	Tableau dynamique
Accès / Modification	temps constant $O(1)$	temps constant $O(1)$
Recherche	temps linéaire $O(n)$	temps linéaire $O(n)$
Ajout/Insertion	<i>impossible</i>	?
Suppression	<i>impossible</i>	?

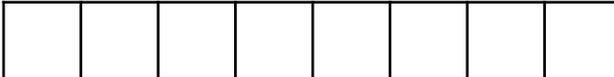
Coût de l'ajout

- Stratégies d'ajout dans un tableau dynamique
 - Doubler la capacité du tableau
 - Augmenter la capacité d'un nombre constant d'emplacements
 - Etc.
- En doublant la capacité :

$$2^0 = 1$$


$$2^1 = 2$$


$$2^2 = 4$$


$$2^3 = 8$$


$$2^4 = 16$$


Extension $k \Rightarrow$ capacité du tableau = 2^k

Coût de l'ajout

- Soit n le nombre d'éléments du tableau (= `taille_utilisee`)
- On sait qu'il existe $p > 0$ tel que $2^{p-1} < n \leq 2^p < 2n$
- A l'ajout du $n^{\text{ème}}$ élément, on a le coût total (ajouts et recopies d'éléments):

$$C = n + \sum_{k=0}^{p-1} 2^k = n + 2^p - 1 < n + 2^p < 3n$$

- Le terme n car il faut ajouter les n éléments
 - Le terme sommé pour toutes les recopies aux moments des extensions du tableau
- Coût amorti = $\frac{\text{coût total}}{n} \leq \frac{3n}{n} = 3$

Suppression d'un élément

- Après avoir doublé la capacité du tableau plusieurs fois et supprimé des éléments, le rapport entre la taille utilisée et la capacité peut devenir petit
 - On a beaucoup d'emplacements vides, donc inutiles en mémoire
- Stratégie de libération de la mémoire : si la taille utilisée passe en dessous du tiers de la capacité, on utilise un tableau deux fois plus petit
 - On alloue un nouveau tableau de capacité/2 éléments sur le tas
 - On recopie tous les éléments sauf celui à supprimer dans le nouveau tableau
 - On libère la mémoire de l'ancien tableau
- Permet de libérer beaucoup de mémoire, tout en restant flexible à des ajouts juste après une suppression qui libère