

Structures de données, IMA S6

Piles et Files

Laure Gonnord

<http://laure.gonnord.org/pro/teaching/>
Laure.Gonnord@polytech-lille.fr

Université Lille 1 - Polytech Lille

Février 2011



Plan

Laure Gonnord (Lille1/Polytech'Lille)

Structures de données IMA S6 Piles et Files

Février 2011

← 2 / 18 →

Piles

1 Piles

2 Files

Introduction

Jusqu'à présent, on a vu :

- Les listes contiguës (tableaux).
 - Les listes chaînées simples, doublement chaînées ...
- ▶ Chacune de ces SD effectue un **compromis** (complexité des opérations, temps, mémoire)
- ▶ On choisit une SD en fonction de **l'application**.

Dans ce cours on va voir les **piles** et les **files** (variantes des listes chaînées).

1 Piles

2 Files

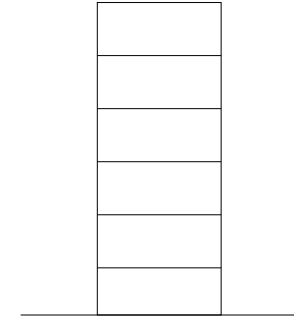
Les piles, pourquoi ?

Exemples d'utilisation :

- pour modéliser des piles (de cartes, de papiers, ...)
 - pour faire des recherches de chemin (labyrinthe, ...)
 - pour gérer la récursivité (cf cours précédent).
- Toutes les fois où l'information importante est la dernière rangée (ou l'avant dernière, ou ...)

Dessin !

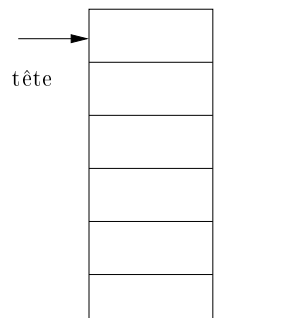
Last In First Out (LIFO)



► La dernière information rangée est la première disponible

Dessin !

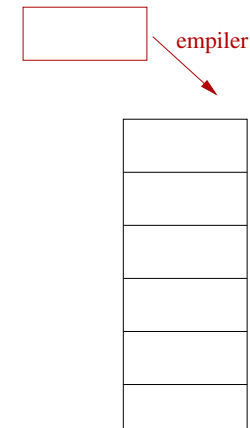
Last In First Out (LIFO)



► La dernière information rangée est la première disponible

Dessin !

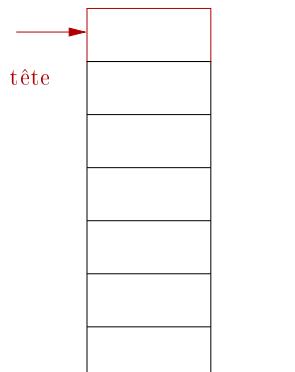
Last In First Out (LIFO)



► La dernière information rangée est la première disponible

Dessin !

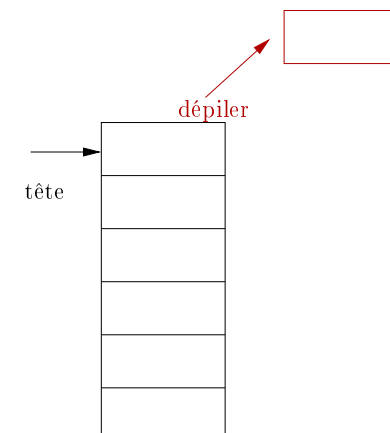
Last In First Out (LIFO)



► La dernière information rangée est la première disponible

Dessin !

Last In First Out (LIFO)



► La dernière information rangée est la première disponible

Le type abstrait Pile

Définition en pseudocode :

pile de entiers

Opérations :

- empiler(P, el) : ajoute l'élément en haut de pile
- dépiler(P, var) : retire l'élément et stocke dans var
- int sommet(P) : regarde l'élément de haut de pile
- bool pileVide(P) : teste si la pile est vide
- initPile(P) : initialise P à vide
- bool pilePleine(P) : teste si la pile (bornée) est pleine

Vers l'idée de bibliothèque / library

Avec ces opérations de base, on peut écrire des algorithmes utilisant les piles **sans connaître quoi que ce soit sur l'implémentation**.

Remarque normalement les détails d'implémentation sont dans la doc de la bibliothèque.

Utilisation d'une pile

Exos classiques :

- Écrire (en utilisant les **piles**) un algorithme qui dit si une chaîne de caractères donnée est bien parenthésée.
- La calculette en notation polonaise inversée (cf **TD**)

Implémentation d'une pile - 1

On a le choix :

- Implémentation par liste contiguë
 - Implémentation par liste chaînée
- commençons par la liste contiguë

Implémentation d'une pile - 2

On parle de **type concret**. Déclaration d'une pile d'entiers :

Structure *Pile*

index : Entier

tab : tableau[MAX] d'entiers

FStruct

► écrivons les fonctions de base :

- empiler(P, el) : ajoute l'élément en haut de pile
- dépiler(P, var) : retire l'élément et stocke dans var
- int sommet(P) : regarde l'élément de haut de pile
- bool pileVide(P) : teste si la pile est vide
- initPile(P) : initialise P à vide
- bool pilePleine(P) : teste si la pile (bornée) est pleine

Implémentation d'une pile - 3

Avantages des listes chaînées :

- pas de limite sur le nb d'éléments
- moins de place mémoire

Sinon, n'apporte rien au niveau algo !

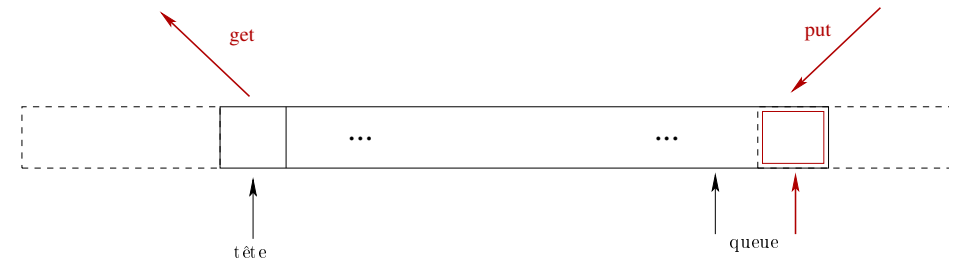
► Implémentation par listes contiguës.

Description - Utilisation

SD dans laquelle les éléments sont utilisés dans l'ordre de leur rangement (FIFO)

1 Piles

2 Files



- stock de données périssables
- files d'attente de supermarché
- gestion de ressource partagée.

Le type abstrait File (fifo) - interface

Définition en pseudocode :

fifo de entiers

Opérations :

- `initFifo(F)` : initialise F à vide
- `bool isFifoEmpty(F)` : teste si la file est vide
- `bool isFifoFull(F)` : teste si la file (bornée) est pleine
- `put(F, el)` : range en queue de file
- `get(F, var)` : retire l'élément de début de file et stocke dans var
- `int first(F)` : regarde l'élément de début de file

Implémentation d'une file - 1

Implem par liste chaînée :

- Fortement dynamique mais sans estimation de taille max
- Get et first : accès aisé
- Put et last : accès coûteux
- ▶ Liste chaînée avec un **pointeur de queue**.

Structure *Fifo*

head : **pointeur de Cellule**
tail : **pointeur de Cellule**

FStruct

- ▶ Implémentons `initFifo`, `first` et `get`

Implémentation d'une file - 2

Implem par liste contiguë :

- Taille peu variable mais max connue
- Put et last : accès en queue aisé avec **indice dernier**
- First : accès en tête, donc ok.
- Get : comment gérer la tête variable ?
 - compactage systématique : cher
 - indice premier avec un espace libre devant
 - solution : **boucler sur l'espace**

► Implémentons `initFifo`, `isfifoEmpty`, `isfifoFull`, `first`, `put` et `get`

Implémentation d'une file - 3

Alors, liste ou tableau ? ► Comme d'habitude, cela dépend !