Algorithmique

Types de données abstraits

Les types de données abstraits

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Eric Guérin
INSA de Lyon – Département Informatique
3IF

Introduction

- Un type de données abstrait (TDA) est une structure de données **abstraite** (on ne donne pas son implémentation) sur laquelle on peut effectuer des **opérations** qui sont clairement identifiées
- Aucun choix d'implémentation n'est fixé
- En règle générale plusieurs implémentations possibles

Introduction

- Exemple : une liste est un TDA
- Une liste chaînée par contre est une structure de données (façon d'organiser la mémoire)
- On peut implémenter une liste grâce à une liste chaînée mais pas obligatoirement
- Attention au vocabulaire

Introduction

- TDA : pas de complexité car dépend de l'implémentation
- Parfois, chaque opération peut donner une indication de la complexité voulue

Exemples

- Pile : collection d'objets (pas au sens instance de classe) accessible selon une politique LIFO
  - *Last In First Out* (dernier entré premier sorti)
- File : collection d'objets accessible selon une politique FIFO (*First In First Out*)
- File double : mélange de pile et file
- Liste : collection ordonnée d'objets accessible selon la position relative

Exemples

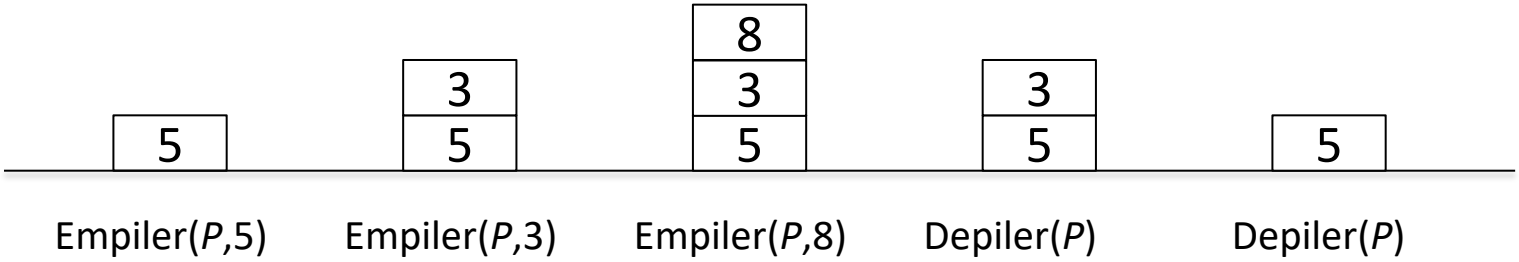
- Vecteur : collection ordonnée d'objets accessible selon la position absolue (indice)
- File de priorité : collection d'objets avec une priorité associée dont le seul accessible est celui de priorité maximum
- Dictionnaire : collection de paires (clé, valeur) avec des opérations de recherche, insertion et suppression basée sur la clé
- Ensemble : collection non ordonnée d'objets sans doublons

La suite

- La suite du cours
  - Développer les TDA les plus connus
  - Plusieurs implémentations
  - Applications

Algorithmique

TDA - Piles



INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

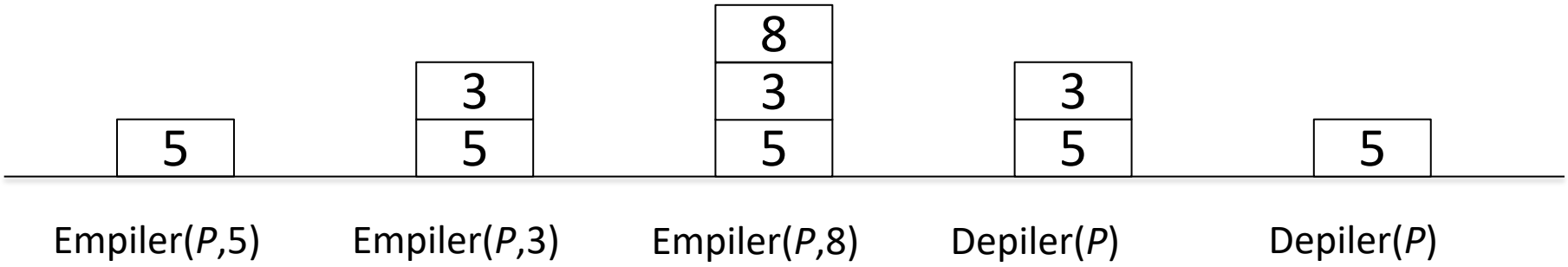
Eric Guérin
INSA de Lyon – Département Informatique
3IF – 2018-2019

Introduction

- Collection dynamique
- Accès par politique LIFO
  - *Last In First Out* = le dernier entré est le premier à sortir
- Des éléments posés les uns sur les autres (empilés) et on ne peut prendre que celui qui se trouve sur le dessus

Introduction

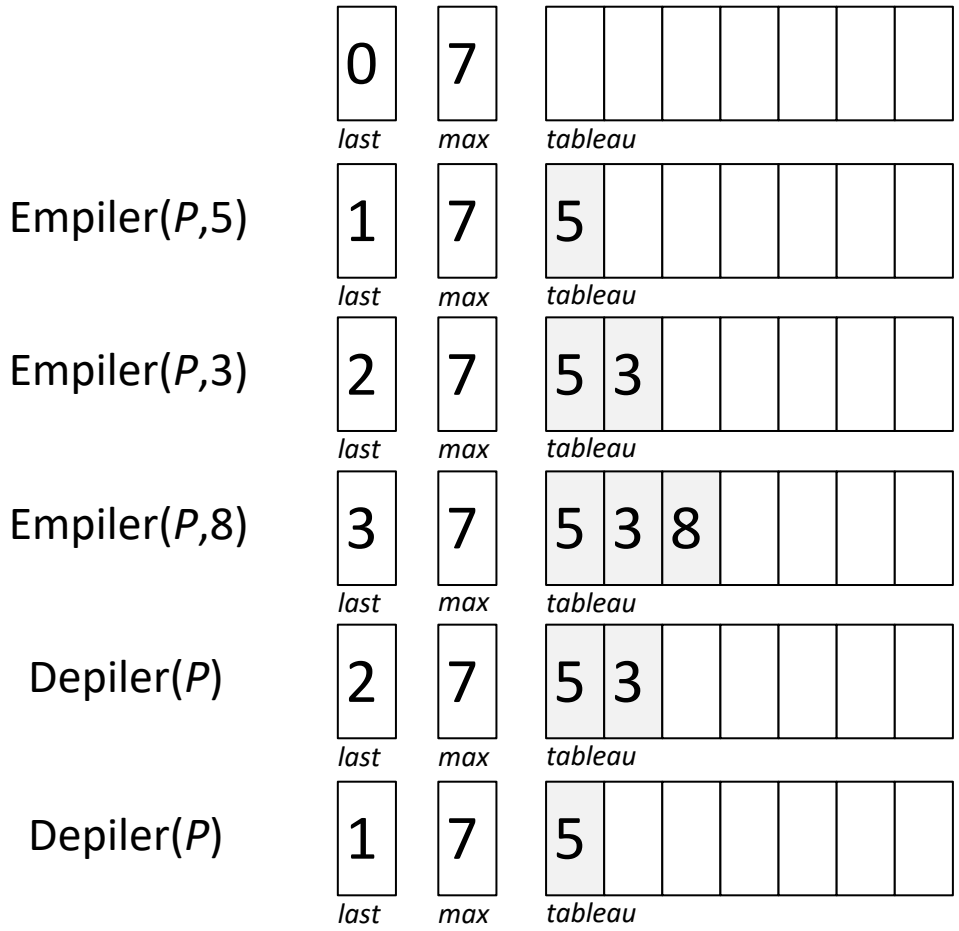
- Opérations
  - `Vide`(P) renvoie un booléen vrai si la pile est vide
  - `Empiler`(P,v) place v sur le sommet de la pile
  - `Depiler`(P) renvoie la valeur du sommet et la retire



Implémentation - tableau

- Un tableau avec l'indice de la prochaine place libre
- Empiler
  - On place la valeur dans le tableau
  - On incrémente l'indice
- Dépiler
  - On décrémente l'indice
  - On renvoie la valeur à l'indice
- Vide
  - On teste si l'indice est nul

Implémentation - tableau



Implémentation – liste chaînée

- Tableau : limitation du nombre d'éléments
- Deux solutions
  - Réallocation si dépassement
  - Utilisation d'une liste chaînée
- Empiler
  - Insertion en tête
- Dépiler
  - Suppression en tête
- Vide
  - Test si la liste est vide (pointeur nul par exemple)

Gestion d'erreur

- Que se passe-t-il si on appelle Depiler() sur une pile déjà vide ?
- Plusieurs solutions
  - Depiler() renvoie un booléen qui indique si l'opération s'est bien passée
  - Valeur de retour particulière (bof)
  - Utilisation d'un mécanisme d'erreur du langage cible (exceptions en C++ par exemple)
- Le même problème peut se poser sur l'opération Empiler() dans le cas d'une taille de pile bornée

Exercices

- Donner la structure d'une pile implémentée avec une liste chaînée
- Donner les trois algorithmes des opérations qui permettent de la traiter
- Besoin d'une autre opération ?

Applications

- Vérification de la cohérence d'une chaîne de caractères (parenthèses ouvrantes et fermantes)
- Ouvertures
  - On empile le symbole
- Fermetures
  - On dépile et on compare avec la fermeture
- A la fin, on vérifie que la pile est vide
  - Toutes les parenthèses ont été fermées

Applications

- Evaluation d'une expression post-fixée
  - Feuille de l'expression
    - On empile la valeur
  - Opérateur binaire
    - On dépile deux fois
    - On effectue l'opération
    - On empile le résultat

Applications

Structure *Expression*
- booléen $estvaleur$
- réel $valeur$
- caractère $operateur$
- Expression * $suivant$

Fonction `EvalExpression`($expression$,$diagnostic$)
- **Entrée** : Expression * $expression$
- **Précondition** : $expression$ contient une liste chaînée représentant une expression arithmétique en notation post-fixée.
- **Postcondition** : Renvoie la valeur de l'expression. S'il y a une erreur dans l'expression, $diagnostic$ est positionné à $faux$, sinon à $vrai$.
- **Déclaration** : Expression * $courant$, * tmp
  Pile $pile$
  réel $v1$, $v2$
  booléen $d1$, $d2$

Applications


```
courant ← expression
tant que courant ≠ ∅ faire
    si courant → estvaleur alors
        Empiler(pile, courant → valeur)
    sinon
        Depiler(pile,v1,d1)
        Depiler(pile,v2,d2)
        si non d1 ou non d2 alors
            diagnostic ← faux
            Detruire(pile)
            retourne 0
        si courant → operateur =' +' alors
            Empiler(pile,v1 + v2)
        /* Compléter ici avec d'autres opérateurs */
    courant ← courant → suivant
Depiler(pile,v1,d1)
si d1 alors
    diagnostic ← vrai
    retourne v1
sinon
    diagnostic ← faux
    retourne 0
```

Exercice

- Transformer l'expression suivante en notation post-fixée

((6+3)*(4+3))/3

- Simuler l'algorithme précédent (avec sa pile)

Correction

- Version post-fixée

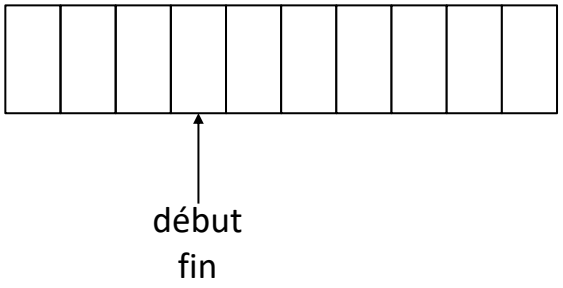
  6 3 + 4 3 + * 3 /

- Exécution (états successifs de la pile)
  - 6
  - 3 6
  - 9
  - 4 9
  - 3 4 9
  - 7 9
  - 63
  - 3 63
  - 21

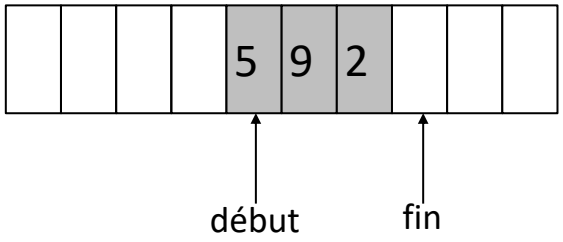
Algorithmique

TDA – Files et files doubles

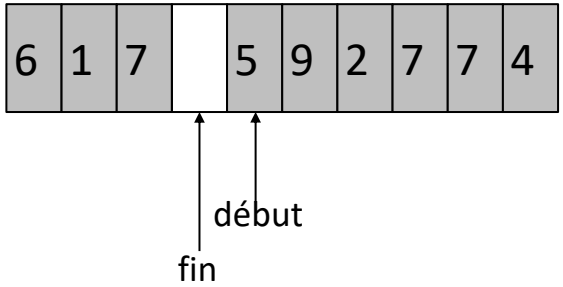
File vide



File contenant 3 éléments



File pleine



INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Eric Guérin
INSA de Lyon – Département Informatique
3IF – 2018-2019

Introduction

- Collection dynamique
- Accès par politique FIFO
  - *First In First Out* = le premier entré est le premier à sortir
- Des éléments sont mis les uns derrières les autres comme dans une file d'attente
- Le premier à être arrivé est le premier servi

Introduction

- Opérations (file simple)
  - `Vide`(F) renvoie un booléen vrai si la file est vide
  - `Enfiler`(F,v) place v dans la file
  - `Defiler`(F) renvoie la prochaine valeur et la retire

Implémentation – tableau circulaire

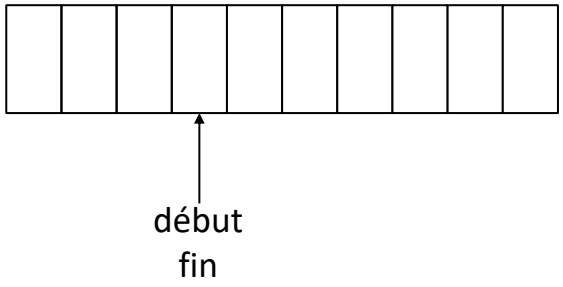
- Un peu plus complexe que la pile
- 2 indices
  - Début
  - Fin
- Enfiler incrémente l'indice de fin
- Défiler incrémente l'indice de début
- Gestion circulaire
  - Quand l'indice devient supérieur à la taille du tableau, on le met à zéro

Implémentation – tableau circulaire

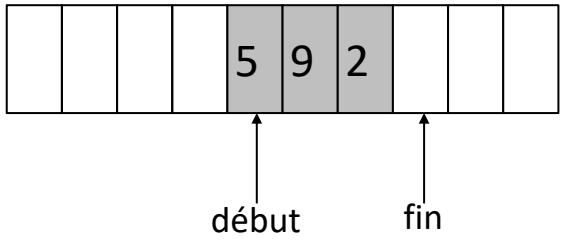
- Codage de la file vide
  - Les indices sont égaux
  - Donc l'indice de fin indique nécessairement la prochaine case à remplir
- Donc une case minimum sera toujours vide
  - Sinon comment différencier la file pleine de la file vide ?

Implémentation – tableau circulaire

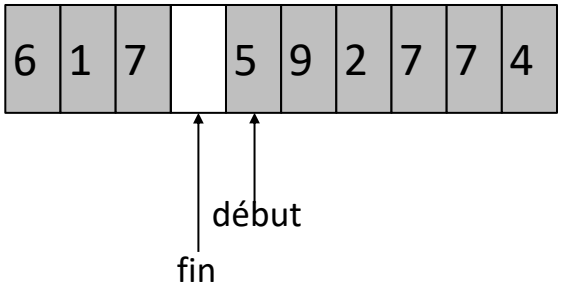
File vide



File contenant 3 éléments



File pleine



Implémentation – tableau circulaire

- Algorithmes donnés dans le polycopié

Implémentation – liste chaînée

- Assez triviale
- La liste chaînée doit permette
  - L'insertion en fin
  - La suppression en tête
- Liste simplement chaînée avec pointeur de fin
- Laissé en exercice

Applications

- Dès lors que l'on veut garder la trace de l'ordre d'arrivée
- Exemples :
  - File d'impression
  - Programmation réseau
  - Traitement de messages

File double

- Généralise la file simple et la pile
- *Double ended queue (deque)*
- Permet d'effectuer des opérations d'ajout et de suppression en début et en fin
- Opérations
  - `InsererDebut()` permet d'insérer au début
  - `InsererFin()` permet d'insérer en fin
  - `SupprimerDebut()` permet de supprimer le premier élément
  - `SupprimerFin()` permet de supprimer le dernier élément

File double

- Implémentation
  - Liste chaînée double avec ou sans sentinelle
  - Toutes les opérations sont en temps constant

Algorithmique

TDA – Listes et vecteurs

Listes et vecteurs

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Eric Guérin
INSA de Lyon – Département Informatique
3IF – 2018-2019

Liste

- Généralisation des piles, files, et files doubles
- Accès à un élément de manière séquentielle par des pointeurs
- Possibilité d'insérer, supprimer, remplacer un élément
- Possibilité de trouver le premier et le dernier de la liste
- Possibilité de trouver le suivant et le précédent d'un élément

Liste

Liste
Vecteur

- Ce TDA est très adapté à une implémentation en liste chaînée, d'où la confusion souvent entre les deux
- La liste des opérations oblige à faire une liste doublement chaînée
- Dans certains langages, on ne considère pas un pointeur sur l'élément mais un itérateur
  - Les deux notions sont très proches sémantiquement
- D'autres implémentations possibles

Vecteur

- Problème de la liste
  - Pour accéder à un élément, il faut parcourir tous les précédents ou tous les suivants
- Vecteur = accès à un élément par son rang (indice)

Vecteur

- Les opérations
  - `ElementPosition`(V , i) renvoie le i^{ème} élément
  - `RemplacerPosition`(V , i , valeur) remplace la valeur du i^{ème} élément
  - `InsererPosition`(V , i , valeur) insère valeur à la ième position
  - `SupprimerPosition`(V , i) supprime le i^{ème} élément
  - `Taille`(V) renvoie la taille du vecteur

Vecteur

- Parfois les deux premières opérations peuvent se représenter avec l'utilisation des crochets
  - `ElementPosition`(V , i) équivalent à V[i]
  - `RemplacerPosition`(V , i , valeur) équivalent à V[i] <- valeur
- Attention à la confusion avec les tableaux !
- En C++, la classe std::vector permet de faire cela

Vecteur

- Applications
  - Le vecteur est une généralisation du tableau dynamique
  - La tableau dynamique ne sait ni insérer à une position, ni supprimer
  - Utilisation dans le même genre de contexte

Vecteur

- Implémentation
- Deux choses contradictoires
  - Accès aux éléments par un rang entier (indice)
  - Ajout/suppression n'importe où dans la collection
- Liste chaînée
  - Bien pour ajout/suppression (encore que discutable car il faut trouver l'élément i avant de pouvoir ajouter ou supprimer)
- Tableau extensible
  - Bien pour l'accès par un indice

Vecteur

- Exercices
- Donner les complexités en temps des opérations de base sur un vecteur
  - Implémenté par une liste chaînée
  - Implémenté par un tableau
- Imaginer une implémentation construite par une liste chaînée de tableaux extensibles
  - Quid de la gestion des ajouts et suppressions ?

Files de priorité

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Eric Guérin
INSA de Lyon – Département Informatique
3IF – 2018-2019

Introduction

- Interface très simple, on veut pouvoir ajouter des éléments avec une priorité, et seulement extraire celui dont la priorité est maximum
- Opérations
  - `Inserer`(FP , p) : insère l'élément de priorité p
  - `Maximum`(FP) : renvoie l'élément de priorité maximum
  - `ExtraireMaximum`(FP) : renvoie l'élément de priorité maximum et le supprime

Applications

- Gestion de ressources dont la priorité est variable
  - En opposition avec la file d'attente où c'est l'ordre d'arrivée qui prime
- Gestion de processus d'un système d'exploitation
- Dans des algorithmes, très utilisé pour augmenter les performances : maintenir une structure de données qui permet d'accéder à l'élément max sans avoir à trier ou rechercher cet élément maximum
  - Exemple : Dijkstra

Implémentations

- Tableau trié par ordre croissant
  - Simple, mais peu efficace
  - Ajout : temps linéaire
  - Maximum/extraction : temps constant
- Tableau non trié, c'est l'inverse
  - Ajout : temps constant
  - Maximum/extraction : temps linéaire
- Liste chaînée
  - Même genre de complexités

Implémentations

- Tas binaire
- Implémentation la plus commune car
  - Ajout en log(n)
  - Maximum en temps constant
  - Extraction en log(n)
- Cf chapitre sur les arbres

Dictionnaires

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Eric Guérin
INSA de Lyon – Département Informatique
3IF – 2018-2019

Introduction

- Ensemble dynamique d'objets avec une clé
- Comparaison de clé possible
- Optionnel
  - Valeur associée à la clé
  - Un parle alors d'association entre une clé et sa valeur (une paire)

Introduction

- Opérations
  - `RechercherDico`(D , k) retourne un pointeur p sur l'élément tel quel que p → cle = k . Si la clé n'existe pas, retourne le pointeur nul
  - `InsererDico`(D , e) insère e dans le dictionnaire. Si un élément possède déjà la clé de e, mise à jour de la valeur
  - `SupprimerDico`(D , k) supprime l'élément dont la clé est k. Rien n'est fait si k n'est pas présent

Applications

- En général deux buts dans l'utilisation d'un dictionnaire
  - Optimiser les coûts d'accès et de modification
  - Optimiser l'espace de stockage
- Applications nombreuses
  - Table des symboles d'un compilateur
    - Clé = nom du symbole
    - Valeur = type, *etc.*
  - Table de noms de domaines (DNS)
    - Clé = nom de domaine
    - Valeur = IP

Implémentations

- Liste chaînée
  - Chaque cellule contient une paire (clé,valeur)
  - Structure de donnée peu adaptée
- Tableau trié
  - Recherche en log(n) (dichotomie)
  - Insertion/suppression, temps linéaire
  - Par contre : pas de perte de place

Implémentations

- Arbre binaire de recherche
  - Implémentation très répandue
  - Temps log(n) pour toutes les opérations
  - Mémoire : gaspillage proportionnel au nombre d'éléments
- Table de hachage
  - Très performant (temps constant)
  - Nécessite un tableau