Types abstraits de données (TAD)

- Un type abstrait de données (Abstract Data Type - ADT) = description d'un ensemble de données
- Un TAD fait une abstraction de la structure de données (structure interne inconnue de l'extérieur)
- Un TAD spécifie:
  - ✓ Le type de données contenues
  - ✓ Une description détaillée des opérations qui peuvent être effectuées sur les données
- Un TAD ne spécifie pas:
  - ☒ La façon dont les données sont stockées
  - ☒ Comment les méthodes sont implémentées

Types abstraits de données (TAD)

Exemple

Modéliser un sac de billes avec un TAD:

- ✓ Le TAD contient des billes
- ✓ Le TAD fournit la possibilité de remettre une bille dans le sac ou d'en prendre une

Types abstraits de données (TAD)

TAD: Sac de billes

Exemple d'implémentation du TAD sac de billes:
Un tissu en coton avec un ruban

Algorithme : Jeu de billes

Le joueur prend une bille du sac de billes et la lance vers la bille cible. Le joueur avec la bille la plus proche du but gagne toutes les billes et remet les billes dans son sac

L'algorithme fait référence au sac des billes uniquement et non pas au tissu en coton

Abstraction: Séparation entre les propriétés du type de données et son implémentation => Modularité : Changer l'implémentation d'un module sans affecter l'implémentation de l'autre

Types abstraits de données (TAD)

TAD: Sac de billes — Interface

Exemple d'implémentation du TAD sac de billes:
Un tissu en coton noué avec un ruban

Exemple d'implémentation du TAD sac de billes:
Un sac en cuir

Exemple d'implémentation du TAD sac de billes:
Un sac plastique avec une fermeture

Class

L'implémentation de l'algorithme 'jeu de billes' est indépendante de l'implémentation du TAD 'Sac de billes'

Types abstraits de données (TAD)

- Spécifier précisément les opérations qui peuvent être exécutées

- Les implémentations sont cachées et peuvent changer facilement

Exemples

données: l'annuaire de téléphone

Operations: chercher, ajouter, effacer …

Piles, Files, and Deques

TAD Piles (Stack)

- Implémentation avec tableau
- Implémentation avec liste simplement chaînée

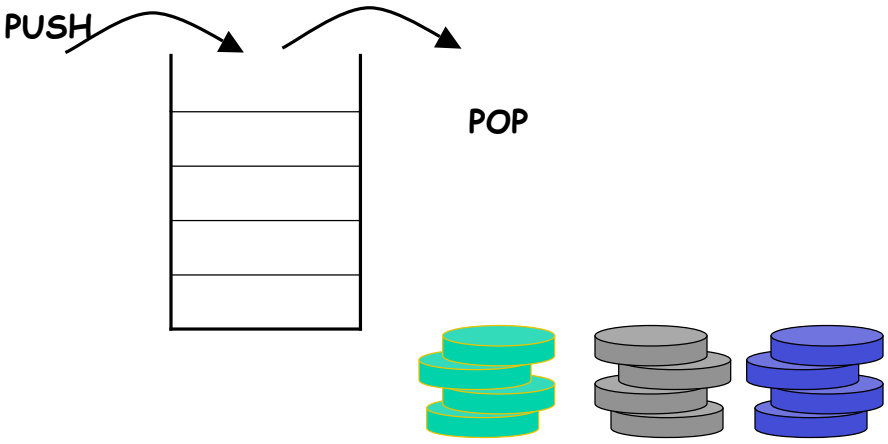
TAD Files (Queue)

- Implémentation avec tableau
- Implémentation avec liste simplement chaînée

TAD Deques (Files à deux bouts)

- Implémentation avec liste doublement chaînée

Piles



Piles

- Une pile contient des objets insérés et retirés selon le principe du dernier rentré, premier sorti (last-in-first-out LIFO)
- Les objets peuvent être insérés à tout moment, mais seulement le dernier (le plus récemment inséré) peut être retiré
- Insérer un objet dans la pile correspond à l'empiler (pushing). Dépiler la pile (popping) correspond au retrait d'un objet.
- Analogie: distributeur de bonbons PEZ®

Le TAD Pile (Stack)

- Méthodes fondamentales:
  - push(e): Insérer l'objet e en top de la pile (Empiler l'objet e)
  - pop(): Retirer l'objet se situant en top de la pile et le retourner (dépiler la pile); une erreur survient lorsque la pile est vide.
- Méthodes secondaires
  - size(): Retourne le nombre d'objets dans la pile
  - isEmpty():Retourne un booléen indiquant si la pile est vide
  - top(): Retourne l'objet du dessus de la pile, sans le retirer; une erreur survient lorsque la pile est vide.

Applications de Piles

- Applications direct
  - L'historique des pages visitées dans un navigateur web
  - Défaire la séquence écrite dans un éditeur de texte (touche 'supprimer')
  - La chaîne d'appel des méthodes dans la machine virtuelle Java
  - Gestion des emails
- Applications indirect
  - ✓ Une structure de données auxiliaire
  - Une composante d'autres structures de données

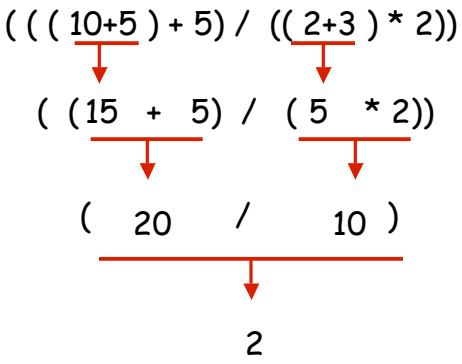
Exemples

Évaluer une expression avec deux piles

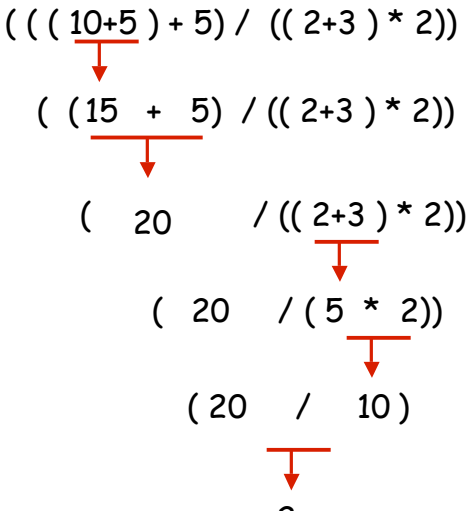
(((10+5) + 5) / ((2+3) * 2))

Comment nous le résolvons ?

Une séquence possible d'opérations



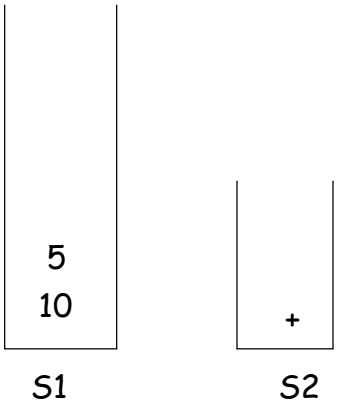
Autre



Avec deux Piles

une pour les opérandes
une pour les opérateurs

(((10+5) + 5) / ((2+3) * 2))



Avec deux Piles

une pour les opérandes
une pour les opérateurs

quand on trouve une parenthèse fermée

((15 + 5) / ((2+3) * 2))
(((10+5) + 5) / ((2+3) * 2))

POP S1 5
POP S2 +
POP S1 10

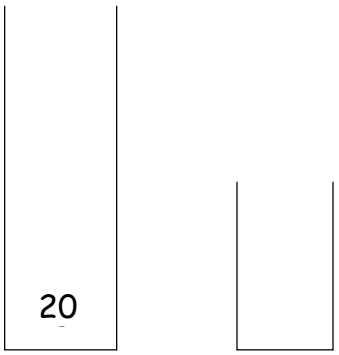
Évaluer: 10 + 5 = 15
PUSH S1 le résultat

Avec deux Piles

une pour les opérandes
une pour les opérateurs

une parenthèse fermée

(20 / ((2+3) * 2))
(((10+5) + 5) / ((2+3) * 2))



POP S1 5
POP S2 +
POP S1 15

Évaluer: 15 + 5 = 20
PUSH S1 le résultat

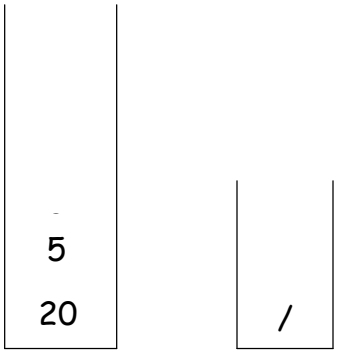
Avec deux Piles

une pour les opérandes
une pour les opérateurs

une parenthèse fermée

(20 / ((2+3) * 2))

(((10+5) + 5) / ((2+3) * 2))



POP S1 3

POP S2 +

POP S1 2

Évaluer 2+ 3 = 5

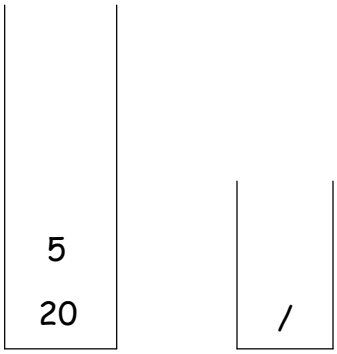
PUSH S1 le résultat

Avec deux Piles

une pour les opérandes
une pour les opérateurs

(20 / (5 * 2))

(((10+5) + 5) / ((2+3) * 2))



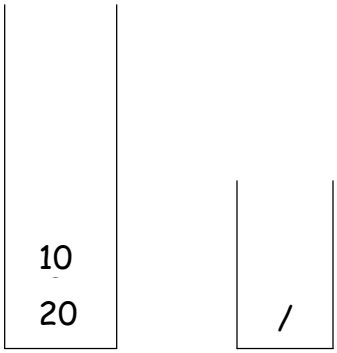
Avec deux Piles

une pour les opérandes
une pour les opérateurs

une parenthèse fermée

(20 / (5 * 2))

(((10+5) + 5) / ((2+3) * 2))



POP S1 2
POP S2 *
POP S1 5

Évaluer 5 * 2 = 10

PUSH S1 le résultat

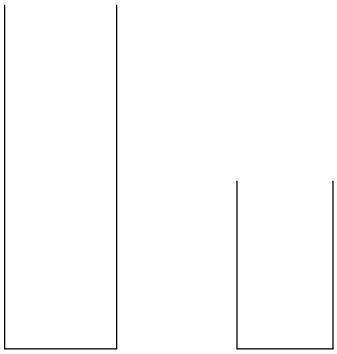
Avec deux Piles

une pour les opérandes
une pour les opérateurs

un parenthèse fermée

(20 / 10)

(((10+5) + 5) / ((2+3) * 2))



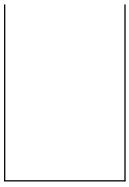
20 / 10= 2

Exemples

Vérifier équilibre des parenthèses

{ [(a+b) -c]/d} { [a+b) -c]/d}

Exemple dans le livre ...



Réalisation d'une pile avec tableau

La pile consiste en un tableau *S* de *N*-élément et une variable entière *t* l'index du «premier» élément dans le tableau *S (top de la pile)*.


```
Algorithm size():
        return t +1
Algorithm isEmpty():
        return (t < 0)
Algorithm top():
        if isEmpty() then
                ERROR
        return S[t]
```

```
Algorithm push(obj):
        if size() = N then
            ERROR
        t ← t + 1
        S[t] ← obj
```

```
Algorithm pop():
        if isEmpty() then
            ERROR
        e ← S[t]
        S[t] ← null
        t ← t-1
        return e
```

Performance et Limitations

- Performance

	Time
size()	O(1)
isEmpty()	O(1)
top()	O(1)
push(obj)	O(1)
pop()	O(1)

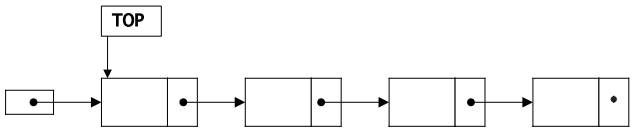
Espace: O(n)

n = taille de le tableau

- Limitations

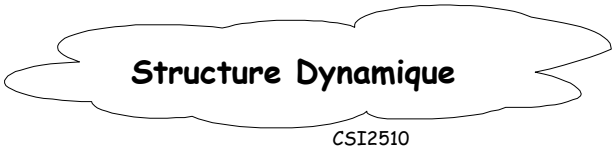
Structure statique

Réalisation d'une Pile à l'aide d'une liste simplement chaînée

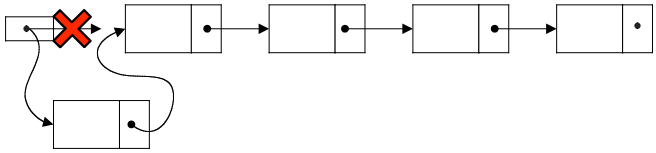


taille = 4

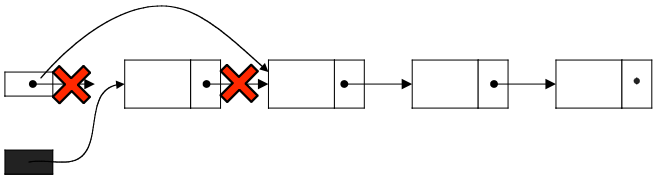
-Liste simplement chaînée avec un variable contenir la taille actuelle de la liste

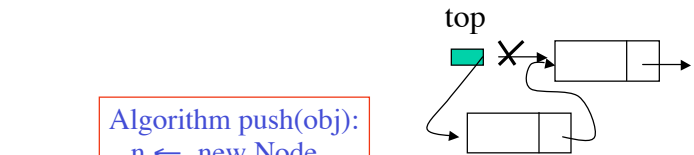


PUSH: Ajouter au devant

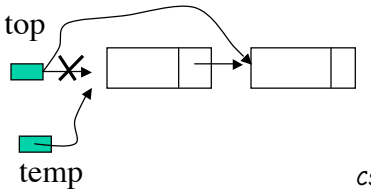


POP: Prendre le première






```
Algorithm push(obj):
  n ← new Node
  n.item ← obj
  n.setNext(top)
  top ← n
  size++
```




```
Algorithm pop():
  if isEmpty() then
            ERROR
  temp ← top.item
  top ← top.getNext()
  size- -
  return temp
```


Performance

Temps:

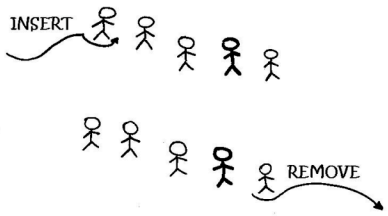
size()	O(1)
isempty()	O(1)
top()	O(1)
push(obj)	O(1)
pop()	O(1)

Espace: Variable

Limitations: ?

La File (The Queue)

La File (Queue)



first-in-first-out (FIFO)

Une file contient des objets insérés et retirés selon le principe du premier rentré, premier sorti (first-in-first-out FIFO)

Les éléments sont enfilés (insérés) du coté arrière et défilés (retirés) du coté avant

Applications de Queues

- Applications directes
  - Les listes d'attentes
  - L'accès aux ressources partagées (ex. imprimer)
  - Multi-programmation
- Applications indirectes
  - Les données auxiliaires structurent pour les algorithmes
  - Le composant d'autres structures de données

Exemple: Palindromes

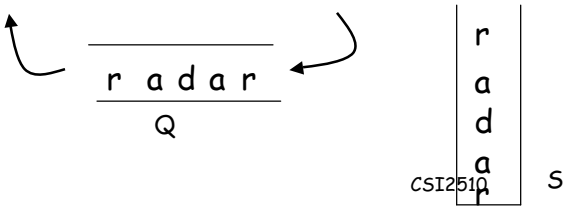
"eye" "madam"

"radar"

"Able was I ere I saw Elba"

Lire la ligne dans une pile et dans une file

Comparer les résultants de la file et la pile

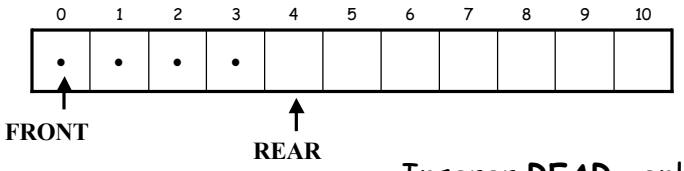


Le TAD File (Queue)

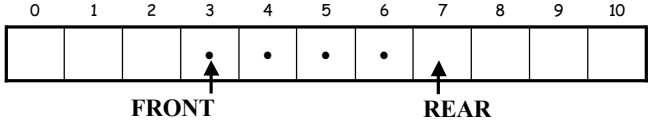
- Méthodes fondamentales:
  - **enqueue(o):** Insérer l'objet o à l'arrière de la file
  - **dequeue():** Retirer l'objet qui est au début de la file et le retourner; une erreur survient lorsque la file est vide

- Méthodes secondaires:
  - **size():** Retourne le nombre d'objets dans la file
  - **isEmpty():** Retourne un booléen indiquant si la file est vide
  - **front():** Retourne l'objet qui est au début de la file sans le retirer; une erreur survient lorsque la file est vide

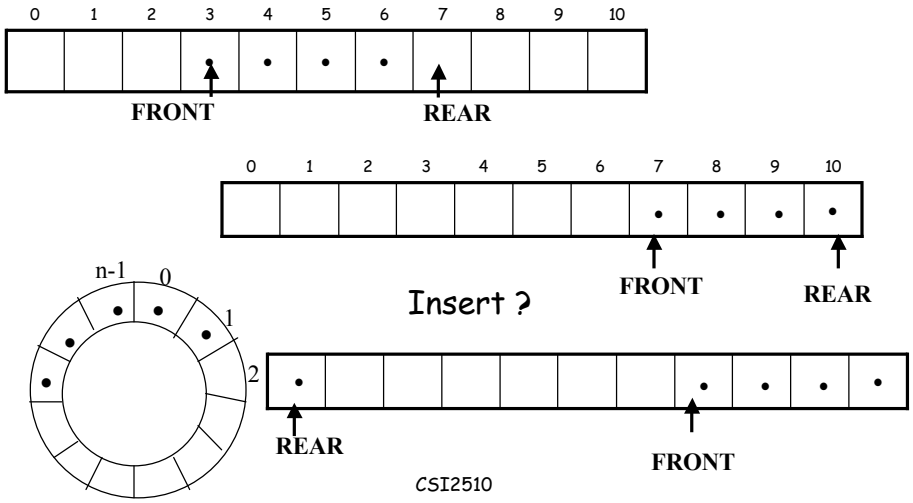
Implementation d'une file avec tableau



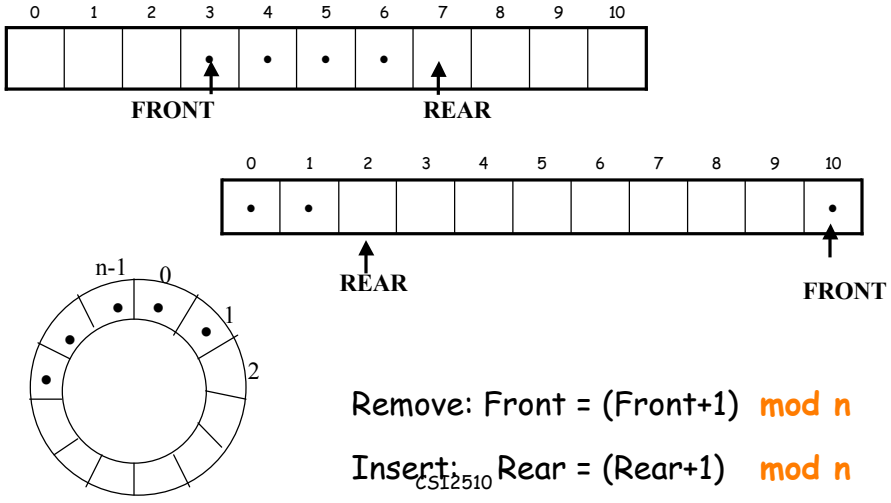
Inserer **REAR** enlever de: **FRONT**



Implementation d'une file avec tableau



Implementation d'une file avec tableau



Remove: Front = (Front+1) mod n

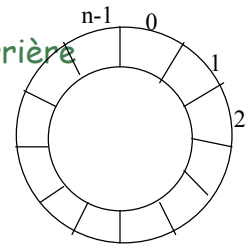
Insert: Rear = (Rear+1) mod n

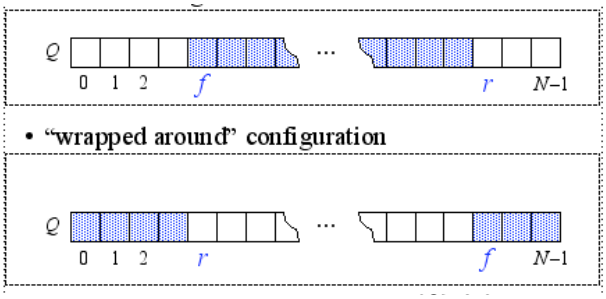
• configuration circulaire ("wrapped around")
• Une taille fixé au début

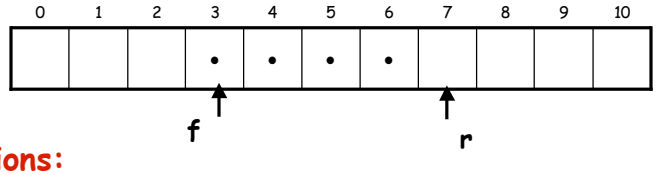
•La file est composée d'un tableau *Q* de *N* éléments et de deux variables entières:

-*f*, l'index de l'élément du devant

-*r*, l'index de l'élément suivant celui de l'arrière







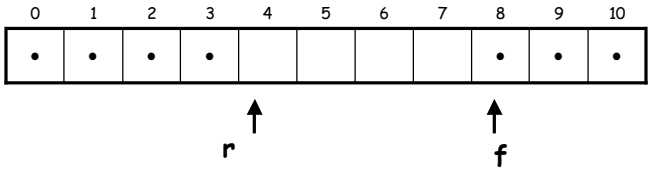
Questions:

Que veut dire *f* = *r*?

La File est vide

Comment calculer le nombre d'éléments dans la file ?

CSI2510



(N - f + r) mod N

Dan le exemple:
(11 - 8 + 4) mod 11 = 7


```
Algorithm size():
        return (N - f + r) mod N

Algorithm isEmpty():
        return (f = r)

Algorithm front():
        if isEmpty() then
                ERROR
        return Q[f]
```

```
Algorithm dequeue():
        if isEmpty() then
                ERROR
        temp ← Q[f]
        Q[f] ← null
        f ← (f + 1) mod N
        return temp

Algorithm enqueue(o):
        if size = N - 1 then
                ERROR
        Q[r] ← o
```

Performance

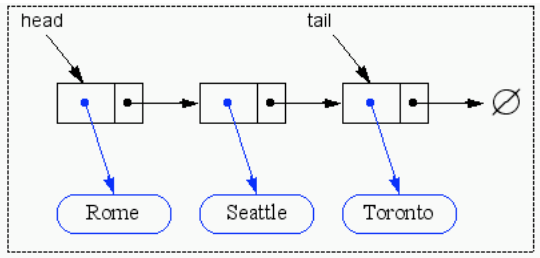
temps:

size()	O(1)
isempty()	O(1)
front()	O(1)
enqueue(o)	O(1)
dequeue()	O(1)

espace: O(N)

Réalisation d'une File à l'aide d'une liste simplement chaînée

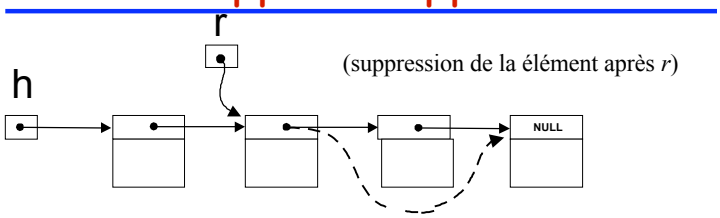
Nœuds connectés en chaîne par de liens (links)



La tête de la liste (head) est le début de la file, la queue de la liste (tail) constitue l'arrière de la file.

Pourquoi pas le contraire?

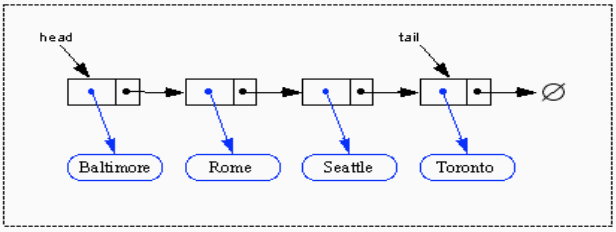
Rappel : Suppression



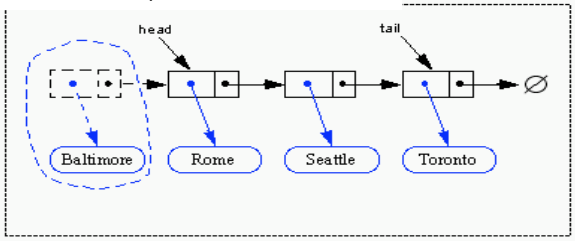
Premier élément (facile)	h ← h.getNext()
Élément après *r* (facile)	w ← r.getNext() r.setNext(w)
Élément à r (difficile)	• Utiliser un pointeur à l'élément précédant, ou • Échanger les contenus de l'élément à *r* avec les contenus de l'élément suivent, et effacer l'élément après *r*. **Très difficile si *r* indique dernier élément!

CSI2510

Retirer l'élément de tête



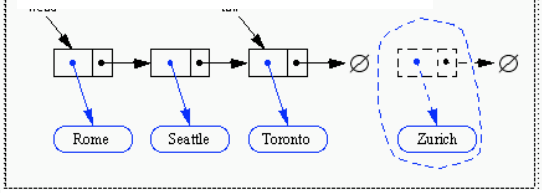
Avancez la référence de la tête



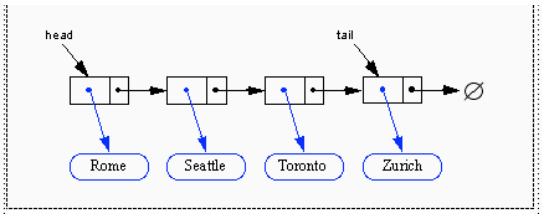
Insérer un élément à la tête est tout aussi facile

Insérer un élément à la queue

Créez un nouveau nœud



Enchaînez-le et déplacez la référence à la queue



Comment retirer l'élément d queue?

Performance

temps:

size()	O(1)
isempty()	O(1)
front()	O(1)
enqueue(o)	O(1)
dequeue()	O(1)

Espace: Variable

Si nous savons en avance une limites supérieur raisonnable pour le nombre d'éléments dans la file, alors

⇨ Tableau

Autrement

⇨ Listes

Plus général TAD: Files à deux bouts (Deque)

Une file à double têtes ou Deque (Double-ended queue) est une généralisation des types files et piles. Les insertions et les suppressions d'éléments dans une Deque peuvent s'effectuer aux deux bouts avant et arrière

•Méthodes fondamentales:

- insertFirst(e): Insérer l'objet e au début de la Deque
- insertLast(e): Insérer l'objet e à l'arrière de la Deque
- removeFirst(): Supprimer et retourner l'objet qui est au début de la Deque *
- removeLast(): Supprimer et retourner l'objet qui est à l'arrière de la Deque*

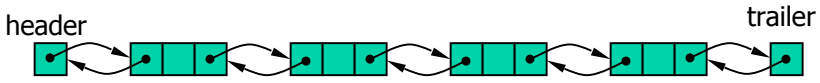
•Méthodes secondaires:

- getfirst(): Retourne l'objet qui est au début de la Deque sans le retirer*
- getlast(): Retourne l'objet qui est a l'arrière de la Deque sans le retirer*
- size(): Retourne le nombre d'objets dans la Deque
- isEmpty(): Retourne un booléen indiquant si la Deque est vide

* Erreur si Deque vide

Réalisation des Deques à l'aide d'une liste doublement chaînée

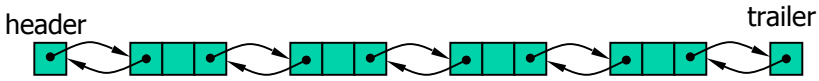
Effacer l'élément de queue d'une liste simplement chaînee ne peut pas être fait en un temps constant



Pour réaliser une deque, nous utilisons une liste doublement chaînée avec des nœuds spéciaux pour l'avant (header) et l'arrière (trailer)

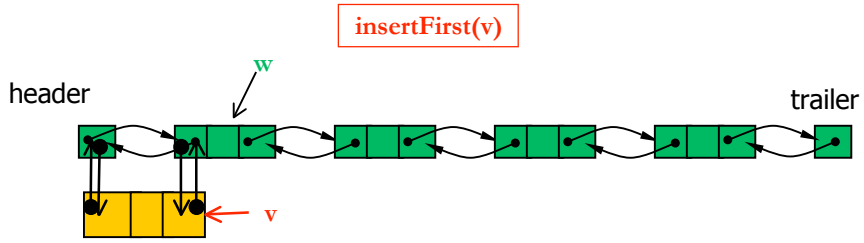
•Le nœud header est placé avant le premier élément de la liste. Sa référence 'suivant' est valide tandis que sa référence 'précédent' est null

•Le nœud trailer est placé après le dernier élément de la liste. Sa référence 'suivant' est null tandis que sa référence 'précédent' est valide



NOTE: Les nœuds header et trailer sont des sentinelles ou nœuds "bidons" parce qu'ils ne contiennent pas d'éléments.

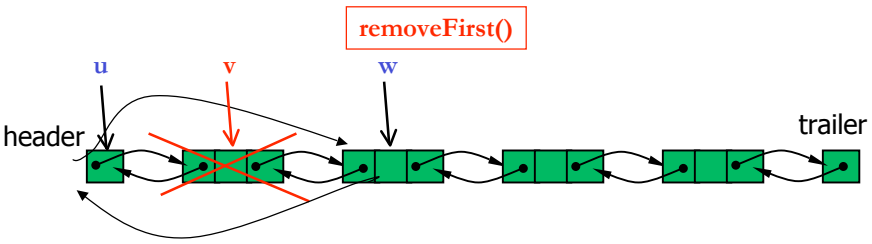
insertFirst(v):



w← header.getNext()
v.setNext(w)
v.setPrev(header)
w.setPrev(v)
header.setNext(v)
Size ← size+1

CSI2510

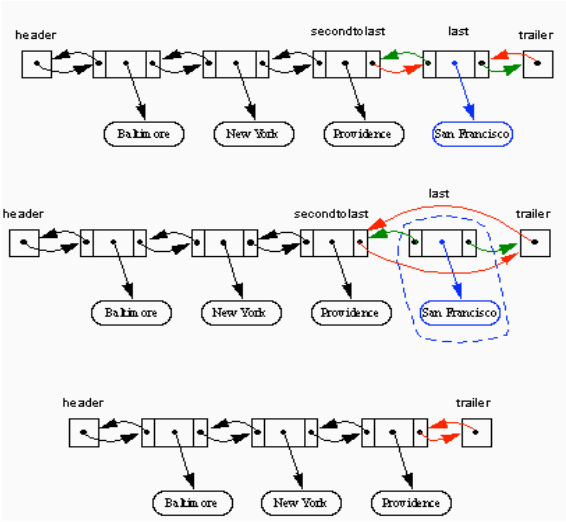
removeFirst():



u← v.getPrev()
w← v.getNext()
w.setPrev(u)
u.setNext(w)
v.setPrev(null)
v.setNext(null)
size← size-1

CSI2510

visualisons de code de removeLast().



Avec cet réalisation, toutes les méthodes ont un temps d'exécution constant (c'est-à-dire O (1))!

Réalisation de piles et de files à l'aide de Deques

Piles avec Deques:

Méthodes de Pile	Implémentation en Deque
isEmpty()	isEmpty()
top()	getLast()
push(e)	insertLast(e)
pop()	removeLast()
size()	size()

Files avec Deques:

Méthodes de File	Implémentation en Deque
isEmpty()	isEmpty()
front()	getFirst()
enqueue(e)	insertLast(e)
dequeue()	removeFirst()
size()	size()