Structure De Données

CTD 11 : Pile

Algorithmique

Semestre 2 - 2019-2020

Introduction
Motivation
Pile
Description du type
Abstraction : Opérations
Structure de données (implémentations)
Implémentation utilisée pour les exercices
Python
Exercices
Parenthésage
Gestion d'une rame

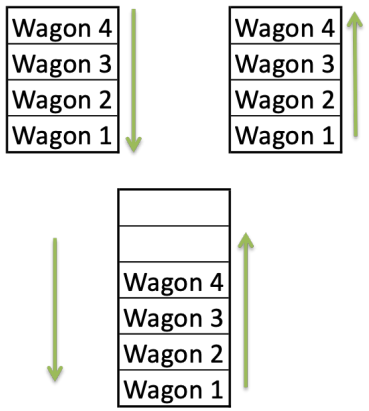
Introduction
Motivation
Pile
Implémentation utilisée pour les exercices
Exercices

Motivation : la structure de données **pile** est omniprésente en algorithmique et dans la vie courante :

- Pile de dossiers ou d'assiettes
- Pile d'exécution
- Calcul en notation postfixée (ou polonaise inverse) : ab + c *
- Tours de Hanoï

Exemples d'utilisation d'une pile

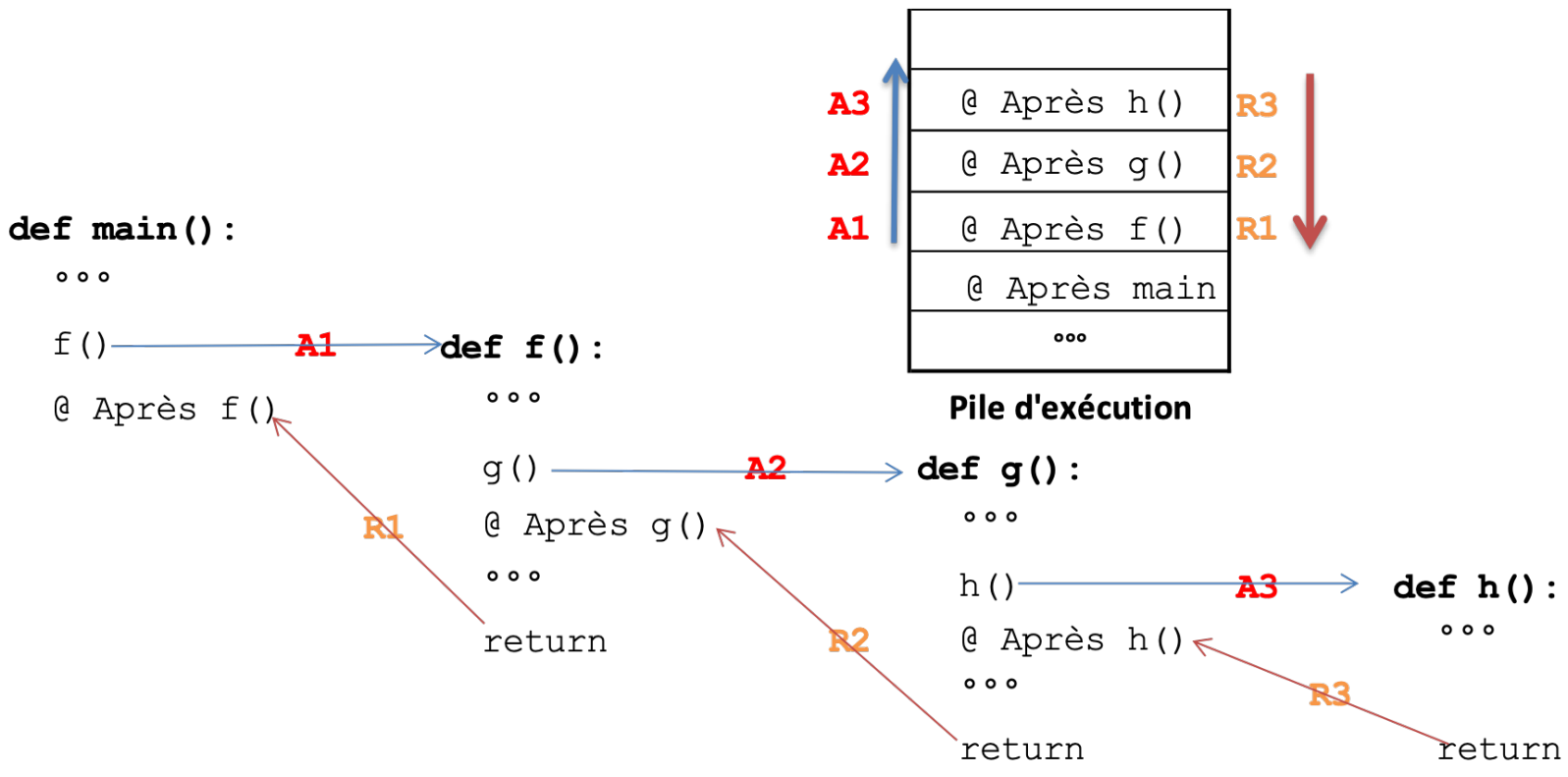
- Arrivée au terminus du métro
  - Dans la voie de retournement, on "empile" les wagons.
  - On les "dépile" ensuite pour avoir la rame prête à repartir dans l'autre sens



- Vérification du parenthésage d'expression : (((a + b) *(c - d)) + ((e – f) / (g + h))) ?
- Visite de sites Internet : on se souvient à chaque fois du précédent pour y retourner lorsqu'on quitte le site courant

Exemples d'utilisation d'une pile

- Appel de fonctions : à chaque appel d'une fonction f, on empile l'adresse de retour (l'instruction qui suit l'appel)
- A chaque return, on dépile une adresse de retour pour revenir après le dernier appel exécuté



Définition informelle

Une **pile** est une structure de données qui mémorise des données/informations selon la politique **dernier arrivé-premier sorti (LIFO : Last In First Out)**

Introduction

Pile
- Description du type
- Abstraction : Opérations

Structure de données (implémentations)

Implémentation utilisée pour les exercices

Exercices

Pile - Description du type

- Nombre d'éléments **variable** (évolution dynamique)
- Eléments de même type
- On ne peut accéder qu'à l'élément en **sommet** de pile

Pile - Abstraction : Opérations

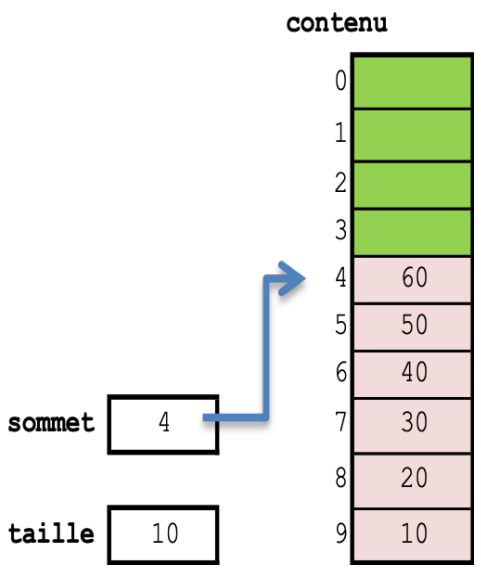
- opérations de construction
  - création d'une pile vide
- opérations modifiant l'état d'une pile
  - Empiler une valeur en sommet d'une pile
  - Dépiler et retourner la valeur en sommet d'une pile non vide
- opérations ne modifiant pas l'état d'une pile
  - Tester si une pile est vide ou non
  - Connaître le nombre d'éléments dans une pile
  - Afficher le contenu d'une pile

Pile - Structure de données (implémentations)

Pour réaliser une pile, diverses implémentations sont possibles :

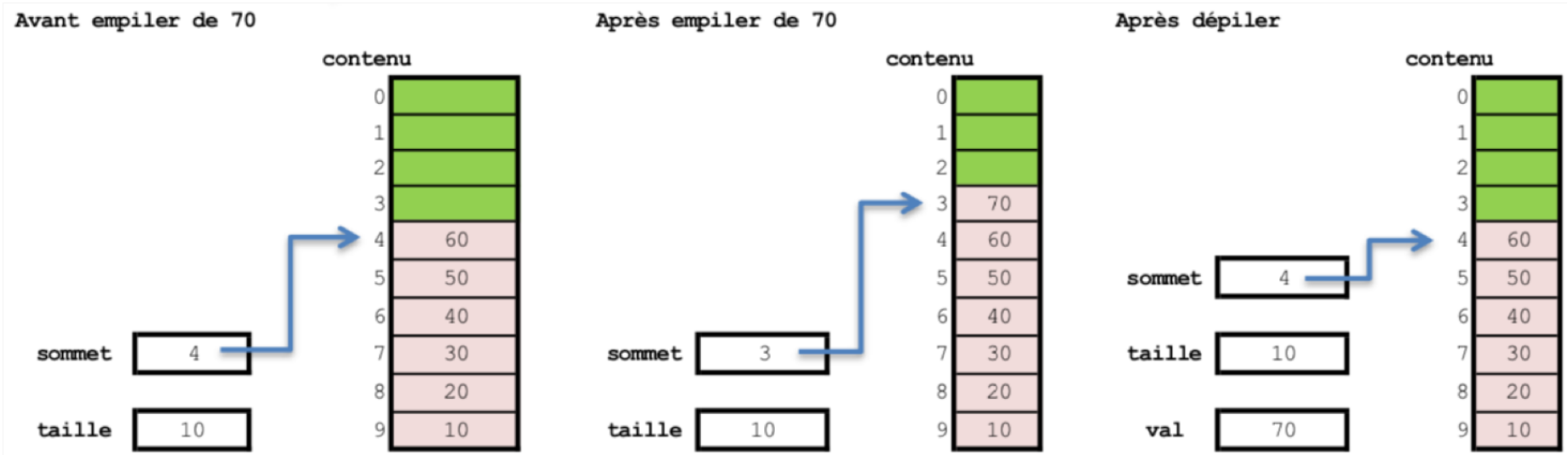
Pile implémentée par un tableau

- Utiliser une zone mémoire contiguë
- Pour gérer une telle pile, il faut :
  - Un tableau qui mémorisera les éléments de la pile (contenu)
  - L'indice de l'élément en sommet de pile (sommet)
  - La taille du tableau réservé pour la pile (taille)
- Pile vide lorsque sommet == taille



Pile implémentée par un tableau

Exemple d'évolution



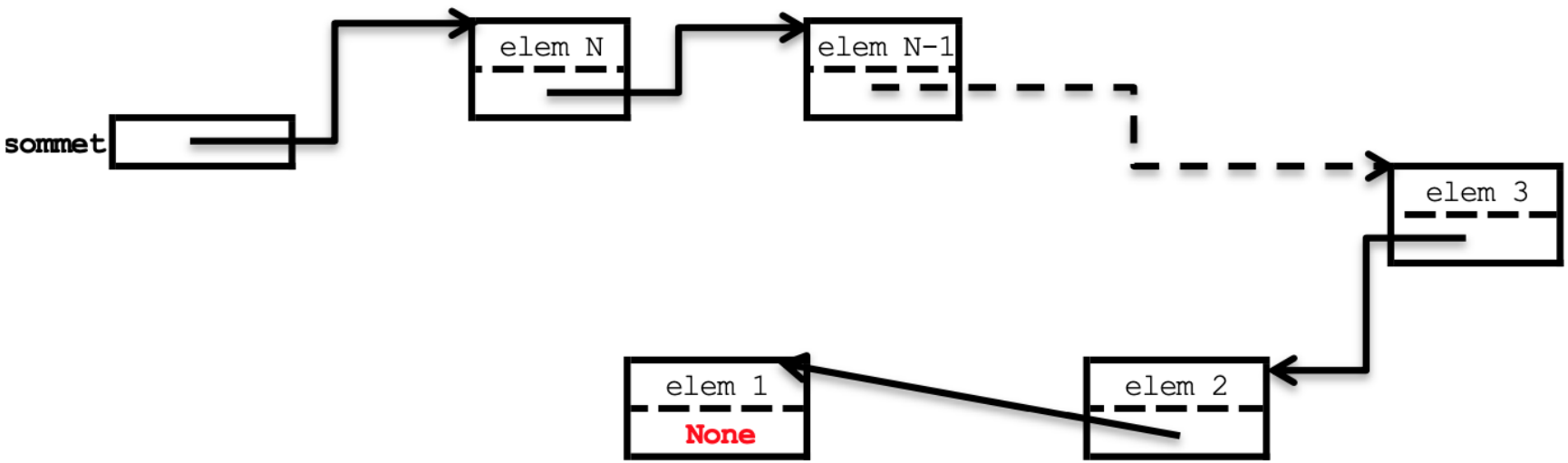
Pile - Structure de données (implémentations)

Pile implémentée en utilisant une liste chainée de cellules

- Séquence logiquement ordonnée de cellules, afin de retrouver l'ordre des empilements de données :
  - Chaque cellule contient une information
  - Chaque cellule permet d'accéder ("pointe") à la précédente, c'est à dire l'élément en dessous dans la pile

Pile implémentée en utilisant une liste chainée de cellules

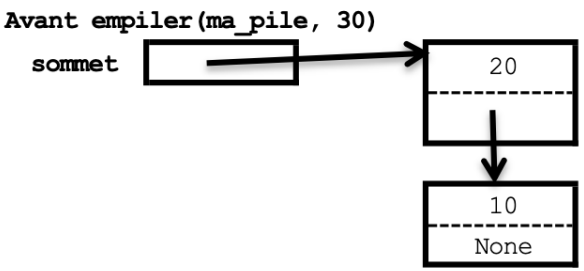
- Variable sommet qui "pointe" la cellule en sommet de pile, qui "pointe" celle qui est dessous . . . et ainsi de suite
- Valeur pointeur particulière NULL, nil ou None pour indiquer qu'il n'y a plus de cellule dessous

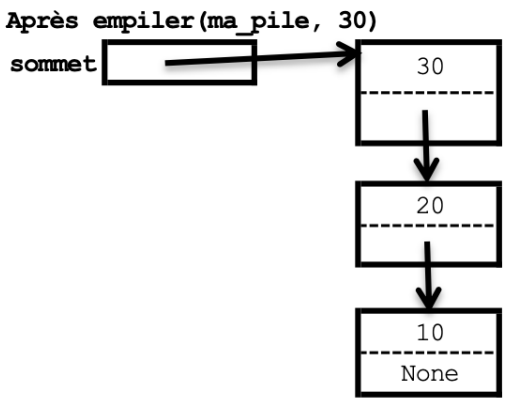


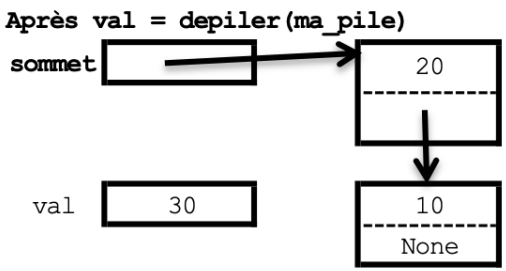
- Pile vide lorsque sommet == None (ne "pointe" rien)

Pile implémentée en utilisant une liste chainée de cellules

Exemple d'évolution







Comparaison Tableau / Liste chainée

- Tableau ou liste chainée : même coût d'exécution
- Occupation mémoire nécessaire
  - Tableau
    - Zone mémoire allouée dès la déclaration de la pile
    - Place perdue tant que la pile n'est pas suffisamment remplie
    - Réallocation d'une zone mémoire lorsque pleine
  - Liste chainée
    - S'étend dynamiquement
    - Pointeur dans chaque cellule (information de service)
    - Occupation mémoire plus importante

Introduction

Pile

Implémentation utilisée pour les exercices
    Python

Exercices

Implémentation utilisée pour les exercices - Python

- On utilise la liste Python pour implémenter la pile
- Une pile est vide si la liste qui la représente est vide
- L'élément en sommet de pile correspond au premier élément de la liste
- Empiler une valeur revient à l'insérer en début de la liste
- Depiler dans une pile non vide retourne le premier élément de la liste, après l'avoir supprimé de la liste

Implémentation utilisée pour les exercices - Python

```python
# creer_pile: retourne une pile vide
def creer_pile_vide():
    return []

# nombre d'elements dans la pile
def nombre_elements(une_pile):
    return len(une_pile)

# pile_vide: retourne True si la pile une_pile
# est vide
def pile_vide(une_pile) :
    return nombre_elements(une_pile) == 0


# empiler: ajoute une_valeur en sommet de la
# pile une_pile
def empiler(une_pile, une_valeur):
    une_pile.insert(0, une_valeur)
```

Implémentation utilisée pour les exercices - Python

```
1  # depiler: supprime et retourne la valeur en
2  # sommet de la pile une_pile.
3  # Une exception est déclenchée si la pile est
4  # vide
5  def depiler(une_pile):
6    assert not pile_vide(une_pile),
7           "depiler␣interdit␣sur␣une␣pile␣vide"
8    sommet = une_pile[0]
9    del une_pile[0]
10   return sommet
11
12 # affiche à l'écran le contenu de la pile
13 # une_pile, en commen ant à partir de la
14 # valeur en sommet.
15 def afficher(une_pile):
16   print(une_pile)
```

Implémentation utilisée pour les exercices - Python

```
1  # fonction de test
2  def test():
3    pile=creer_pile_vide()
4    for i in range(1, 11):
5      empiler(pile, i)
6    print("nb element dans la pile = ",
7                          nombre_elements(pile))
8    afficher(pile)
9    print("Vider la pile")
10   i=1
11   while not pile_vide(pile):
12     val = depiler(pile)
13     print('i=',i,' val=',val,"reste ds pile=",
14                              nombre_elements(pile))
15     i += 1
16   print("Exception pour un depiler en trop")
17   val = depiler(pile)
18 test()
```

Introduction

Pile

Implémentation utilisée pour les exercices

Exercices
- Parenthésage
- Gestion d'une rame

Exercices - Parenthésage

Enonce

On utilise une chaîne de caractères pour stocker une expression mathématique telle que :

```
[{p*(q/(r-s))}/t] - [(x + y)  {(a/b)*c}] + d
```

On souhaite vérifier que les trois types de parenthésage, c'est-à-dire les parenthèses, les crochets et les accolades, sont équilibrés.
Ainsi, l'expression ci-dessus a un parenthésage correct, alors que les expressions `[a  (b*c])`, `a  {b*c]` et `(a  (b*c)` ont un parenthésage incorrect.
Pour simplifier, on ne vérifie pas le placement correct des arguments des opérateurs. Par exemple, on considère que l'expression `(a*)-[bc]` a un parenthésage correct.
Ecrivez une fonction `verifier_parenthesage(expression)` qui retourne `OK` si `expression` a un parenthésage correct, `Erreur` sinon.

Exercices - Gestion d'une rame

Exemple

Entrée	Sortie
[{p*(q/(r-s))}/t] - [(x + y) - {(a/b)*c}] + d	OK
[a - (b*c]	Erreur
a - {b*c)	Erreur
a - [b*c}	Erreur
(a - (b*c)	Erreur

Exercices - Gestion d'une rame

Enoncé

- Ecrire une fonction `retourner_rame` qui simule l'arrivée d'une rame à une station terminus.
- La rame sera représentée par un tableau d'entiers positifs.
- Le retournement de la rame consistera à empiler les wagons jusqu'au dernier, puis à les dépiler pour reconstituer la nouvelle rame. Cette nouvelle rame sera retournée par la fonction.
- Ecrire une fonction de `test` qui simule le parcours d'une rame entre deux stations terminus et qui affiche la composition de la rame au départ de chacune d'elles.