DIU Enseigner l'informatique au lycée

Notes de cours

Kim Nguyễn

kim.nguyen@lri.fr

Piles et Files

Notions introduites

- Structure de pile (premier entré, dernier sorti)
- Structure de file (premier entré, premier sorti)
- Implémentation via des listes chaînées

Piles

Pile

La pile est un type **abstrait** fondamental en informatique. On dispose de quatre opérations :

`creer_pile()` : crée une pile vide

`est_vide(p)` : teste si une pile `p` est vide

`empiler(p, v)` : ajoute une valeur `v` à une pile `p`

`depiler(p)` : renvoie la valeur se trouvant au sommet de la pile (suppose que `p` est non vide)

En particulier : le type ne permet pas a pirori de connaître la taille de la pile, d'accéder au $n^{\text{ème}}$ élément, de fusionner deux piles, ...

Quelques exemples d'utilisation

- Les calculatrices HP (cf. exercices)
- « bouton arrière » du navigateur Web
- fonction annuler dans un traîtement de texte
- « pile d'appels » des fonctions :
  - la pile d'appel est une zone mémoire
  - lors d'un appel de fonction, le processeur place l'adresse de l'instruction courante au sommet de la pile
  - il « saute » vers l'adresse de la fonction
  - lorsque la fonction est terminée (instruction `return`) le processeur « revient » à l'adresse se trouvant au sommet de la pile.

  Ainsi : si `f` appelle `g` qui appelle `h`, lorsque `h` se termine, on se retrouve dans `g` puis quand `g` se termine on se retrouve dans `f` de nouveau.

Démo

La pile d'appel en Python est affichée lors d'une exception non rattrapée.

Réalisation de la structure de données de file

Quelle structure de données concrète utiliser pour réaliser une pile ?

Réalisation de la structure de données de file

Quelle structure de données concrète utiliser pour réaliser une pile ?
Une liste chaînée !

`creer_pile()` : `Liste()`

`est_vide(p)` : `l.est_vide()` (méthode de la classe `Liste`)

`empiler(p, v)` : `l.ajoute(v)`

`depiler(p)` : `l.retire()`

Toutes ces opérations sont en temps constant.

Réalisation de la structure de données de Pile (suite)

Quelle structure de données concrète utiliser pour réaliser une pile ?

Réalisation de la structure de données de Pile (suite)

Quelle structure de données concrète utiliser pour réaliser une pile ?
Un tableau Python !

`creer_pile()` : `[]`

`est_vide(p)` : `len(t) == 0`

`empiler(p, v)` : `t.append(v)`

`depiler(p)` : `t.pop()`

Toutes ces opérations sont en temps constant (amorti).

Conclusions sur les piles

- Type abstrait très utilisé en informatique
- Très simple à réaliser efficacement (liste chaînée ou tableau redimensionnable)
- De nombreux algorithmes utilisent une seule pile globale
  $\Rightarrow$ une interface impérative est adaptée
- Exemples d'utilisations avancées :
  - construction d'une structure arborescente (e.g. DOM) à partir d'une suite de caractères (e.g. texte en HTML)
  - vérification de bon parenthésage (balises HTML, code source, ...)
  - recherche de composantes fortement connexes d'un graphe

Files

File

La file est un autre type **abstrait** fondamental en informatique. On dispose de quatre opérations :

`creer_file()` : crée une pile vide

`est_vide(f)` : teste si une file `f` est vide

`ajouter(f, v)` : ajoute une valeur `v` à la fin d'une file `f`

`retirer(f)` : renvoie la valeur se trouvant au début de la file (suppose que `f` est non vide)

En particulier : le type ne permet pas a pirori de connaître la taille de la file, d'accéder au $n^{\text{ème}}$ élément, de fusionner deux files, ...

Quelques exemples d'utilisation

- Gestion de file d'attente (centre d'appel . . .)
- Gestion de processus dans un système d'exploitation
  - le processeur ne peut exécuter qu'un programme à la fois
  - tous les programmes en cours d'exécution sont dans une file
  - un programme est sorti de la file, il s'exécute pendant un certain temps
  - le programme est remis en attente bout de file et le programme suivant dans la file est exécuté

Réalisation de la structure de données de file

Quelle structure de données concrète utiliser pour réaliser une pile ?

Réalisation de la structure de données de file

Quelle structure de données concrète utiliser pour réaliser une pile ?
Une liste chaînée

Réalisation de la structure de données de file

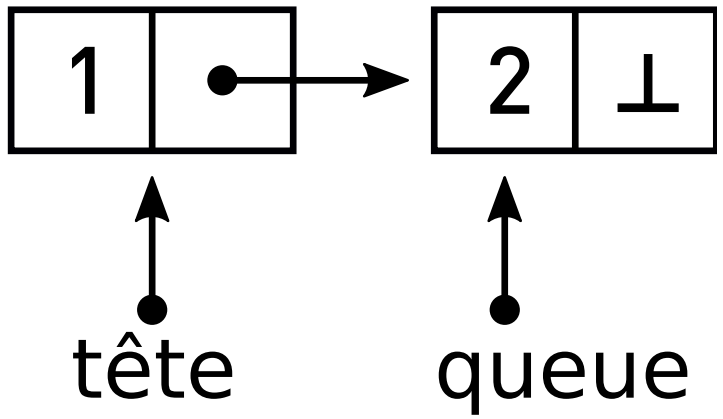
Quelle structure de données concrète utiliser pour réaliser une pile ?
Une liste chaînée customisée !

Réalisation de la structure de données de file

Quelle structure de données concrète utiliser pour réaliser une pile ?
Une liste chaînée customisée !
On garde deux références vers la liste chaînée :

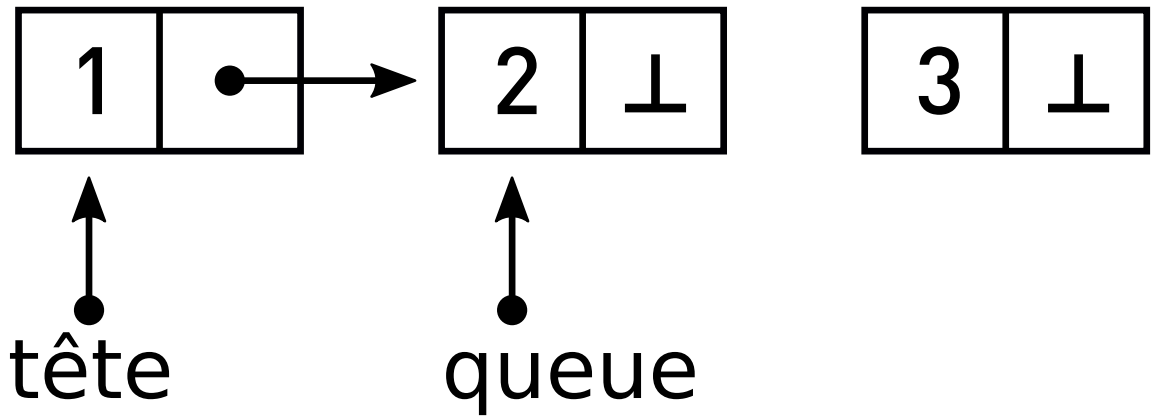
- une référence vers la tête de liste (le début de la file), c'est là qu'on va retirer les éléments (comme dans le cas d'une pile)
- une référence vers la **dernière cellule** de la liste (la queue), c'est là qu'on va ajouter les éléments

Illustration



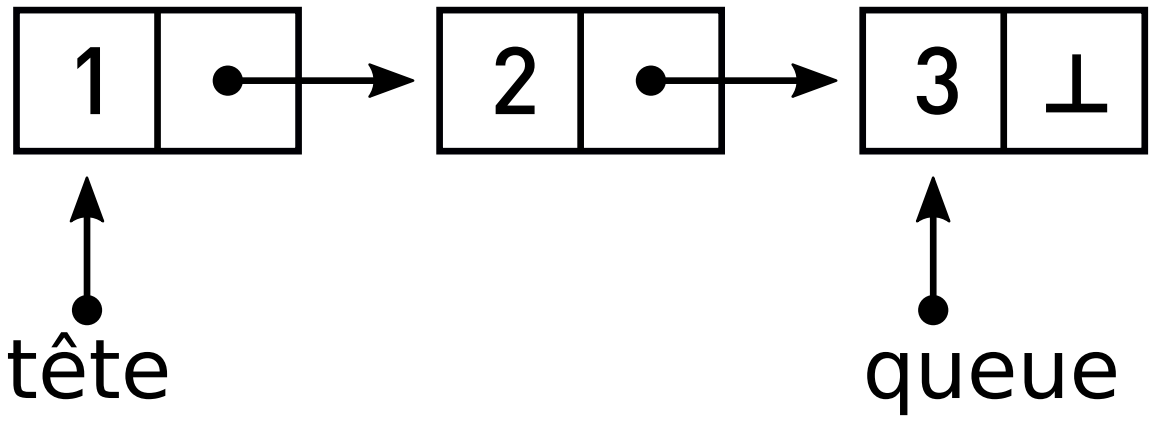
La file initiale contient 1, 2.

Illustration



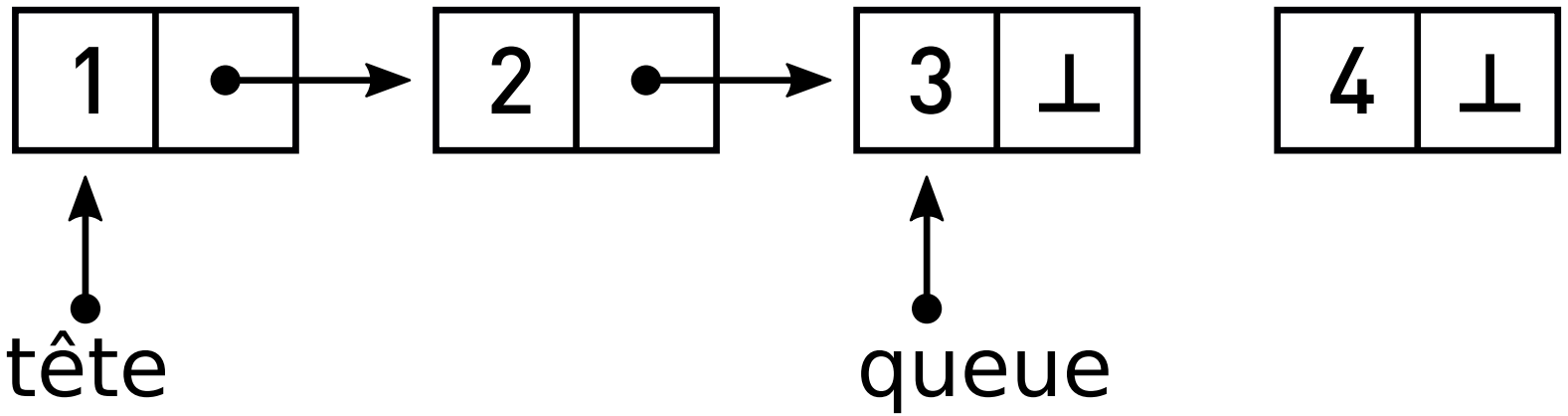
Ajout de 3 en fin de file.

Illustration



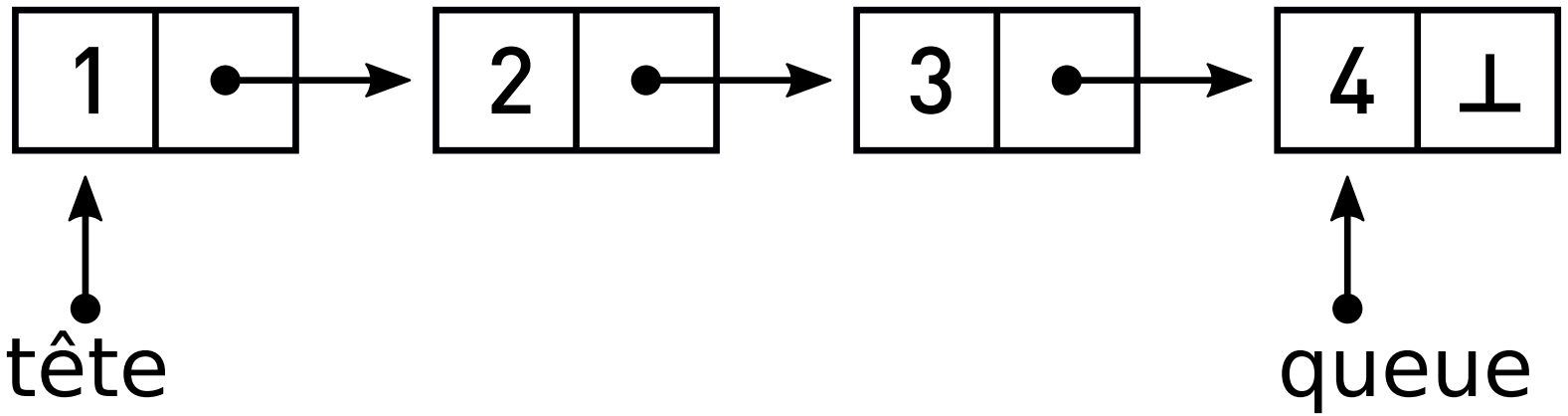
Chaînage et mise à jour de la référence vers la queue.

Illustration



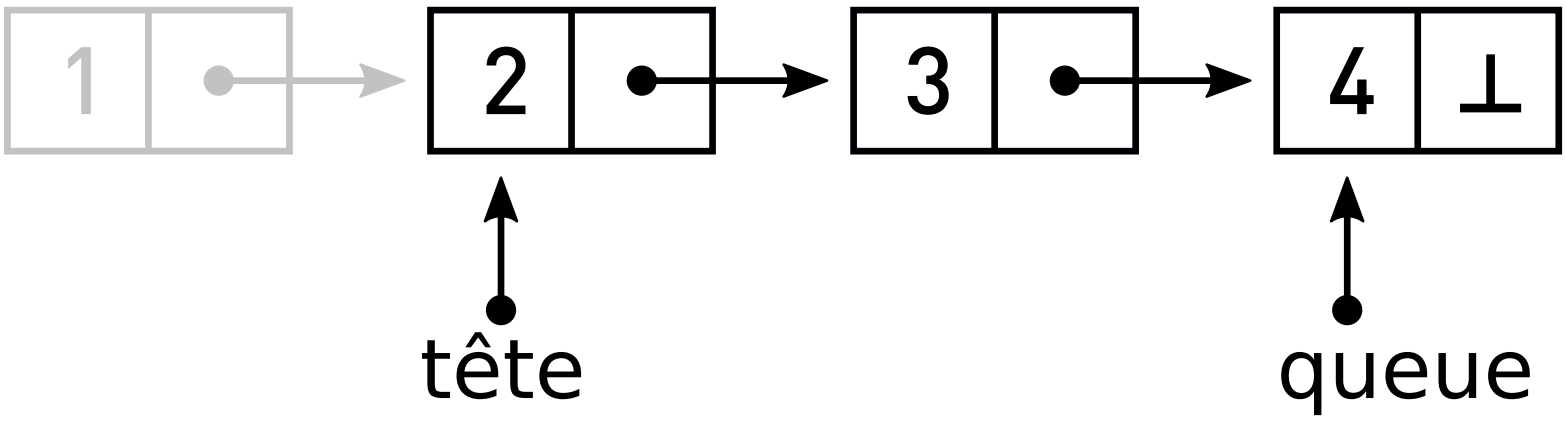
Ajout de 4 en fin de file.

Illustration



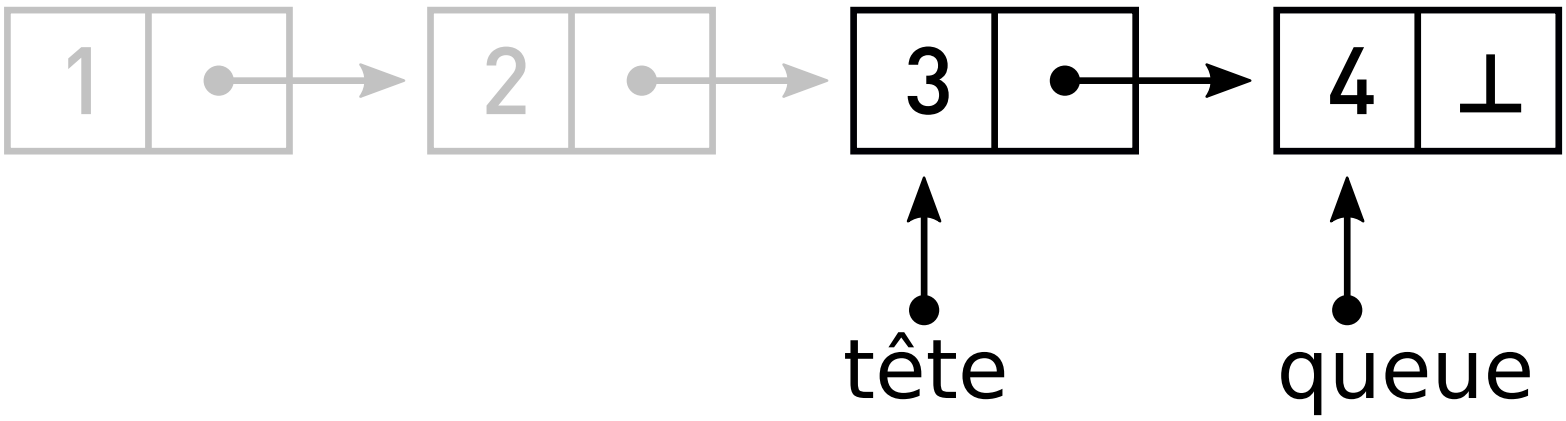
Chaînage et mise à jour de la référence vers la queue.

Illustration



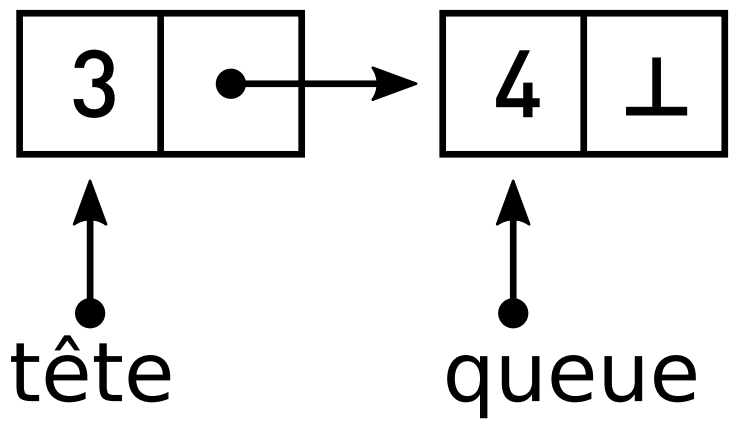
On retire 1, mise à jour de la référence vers la tête.

Illustration



On retire 2, mise à jour de la référence vers la tête.

Illustration



La file finale. Les cellules non référencées sont désallouée par le GC de Python.

Implémentation en Python

- Une liste chaînée de cellules encapsulée dans un objet `File`
- Deux attributs `queue` et `tete`
- Attention : cas particulier de la file vide

Implémentation en Python (suite)

```
from liste import Cellule

class File:

    def __init__(self):
        self.tete = None
        self.queue = None
```

Il va falloir maintenir l'invariant que `tete` et queue sont tous les deux à `None` ou pointent tous les deux vers une cellule valide.

Implémentation en Python (suite)

```
def ajouter(self, v):
    if self.queue is None:
        self.queue = Cellule(v, None)
        self.tete = self.queue
    else:
        self.queue.suivant = Cellule(v, None)
        self.queue = self.queue.suivant
```

- Si la liste chaînée est vide, on crée une unique cellule qui est à la fois la tête et la queue.
- Sinon on crée une nouvelle cellule de queue, on l'enchaîne à l'ancienne queue et on met à jour la référence vers la queue.

Implémentation en Python (suite)

```
def retirer(self):
    v = self.tete.valeur
    if self.tete.suivant is None:
        self.queue = None
    self.tete = self.tete.suivant
    return v
```

- Dans tous les cas, on prend la valeur de la tête de la liste (supposée non vide) puis on fait pointer la tête vers sa cellule suivante.
- Mais si la liste ne contient qu'une cellule, alors seulement dans ce cas on change aussi la queue pour la rendre vide.

Implémentation en Python (suite)

```
def est_vide(self):
    return self.tete is None
```

Inutile de tester queue, l'invariant nous garantit qu'il vaut `None` si la liste est vide.
Toutes les opérations sont en **temps constant**.

Conclusions sur les files

- autre exemple de structure linéaire utilisée en informatique
- premier exemple pour lequel l'implémentation pour laquelle un tableau (redimensionnable) ne convient pas.
- utilisée dans divers algorithmes
  - parcours de graphes
  - ordonnancement de processus
- existent en version plus sophistiquées (files de priorité)