Structure De Données

CTD 11 : File

Algorithmique

Semestre 2 - 2019-2020

Introduction - Motivation

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

- File d'attente des messages à traiter
- File d'attente des tâches à exécuter
- File d'attente des processus prêts à être exécutés par le système
- File d'attente au Resto-U

Définition informelle

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

File - Description du type

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

File - Abstraction : Opérations

- opérations de construction
  - création d'une file vide
- opérations modifiant l'état d'une file
  - Insérer une valeur en queue d'une file
  - Extraire et retourner la valeur en tête d'une file non vide
- opérations ne modifiant pas l'état d'une file
  - Tester si une file est vide ou non
  - Connaître le nombre d'éléments dans une file
  - Afficher le contenu d'une file

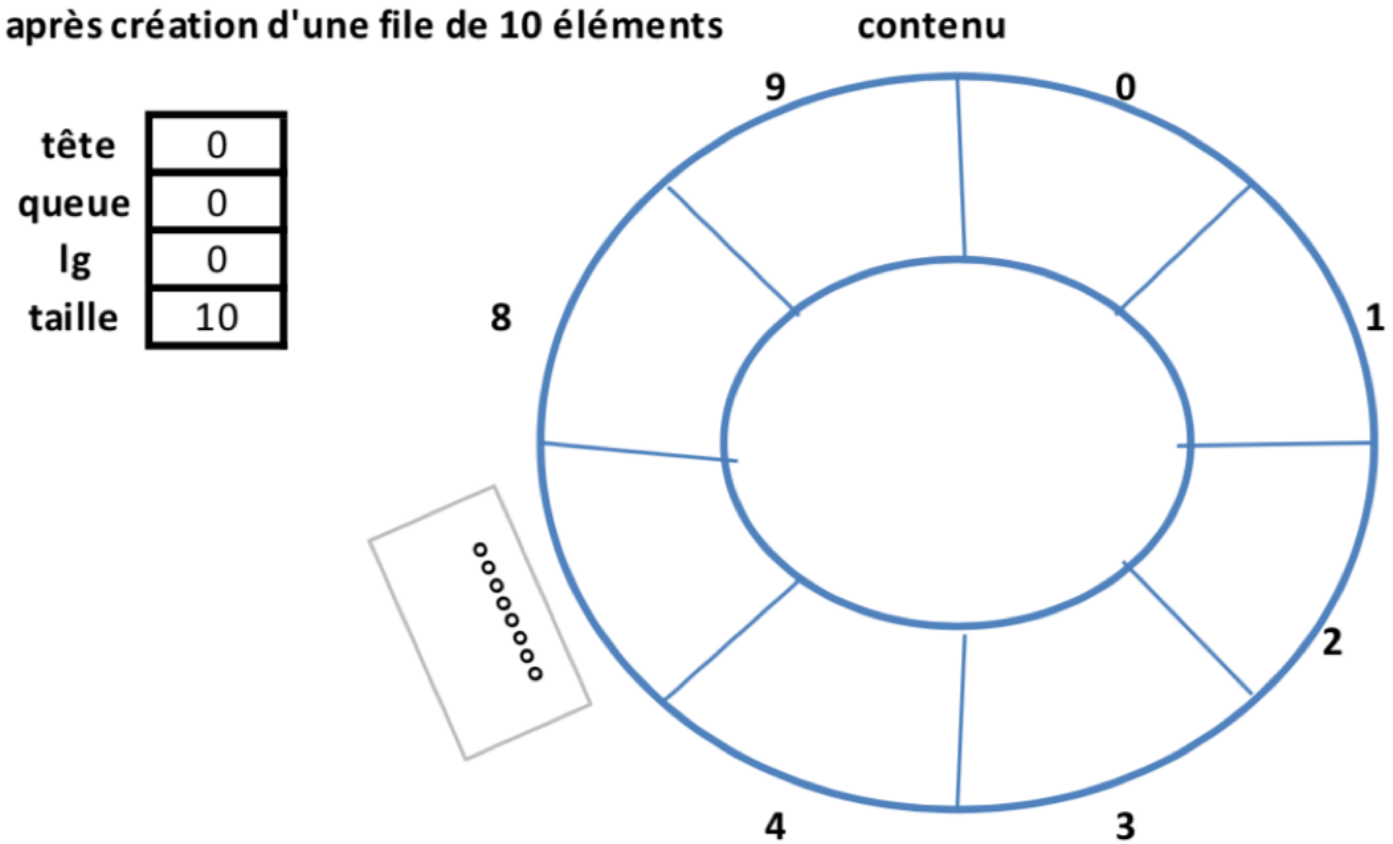
File - Structure de données (implémentations)

Pour réaliser une file, diverses implémentations sont possibles :

File implémentée par un tableau

- Utiliser une zone mémoire contiguë
- Notion de tampon circulaire pour réaliser une file sans avoir à décaler son contenu au gré des ajouts et retraits
- Pour gérer une telle file, il faut :
  - Un tableau pour mémoriser les éléments de la file (contenu).
  - L'indice de l'élément en tête de la file : le prochain qui pourra être retiré (tête).
  - L'indice de l'élément en queue de la file : là où sera stocké le prochain élément (queue).
  - Le nombre d'éléments effectivement mémorisés dans la file (lg)
  - La taille du tableau prévu pour stocker les éléments de la file (taille).
- file vide lorsque lg == 0

File implémentée par un tableau



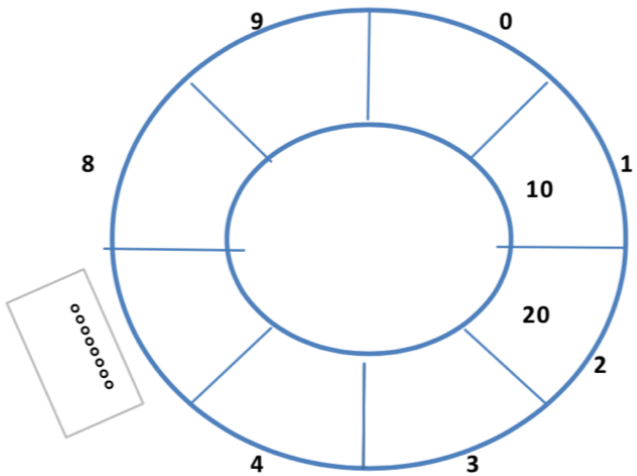
File implémentée par un tableau

Exemple d'évolution

Avant insérer de 30

tête	1
queue	3
lg	2
taille	10

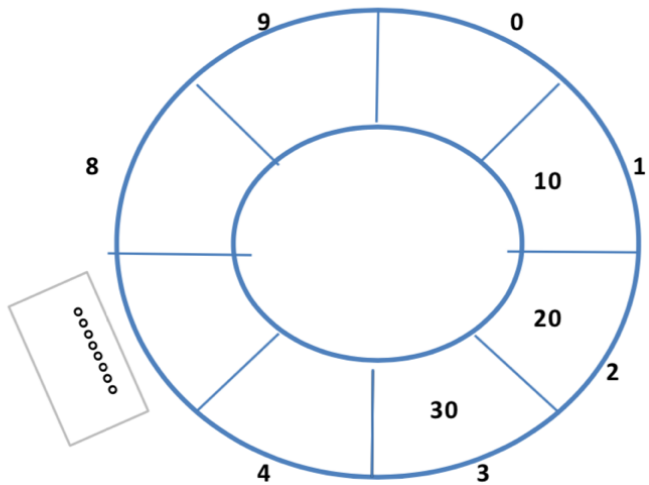
contenu



Après insérer de 30

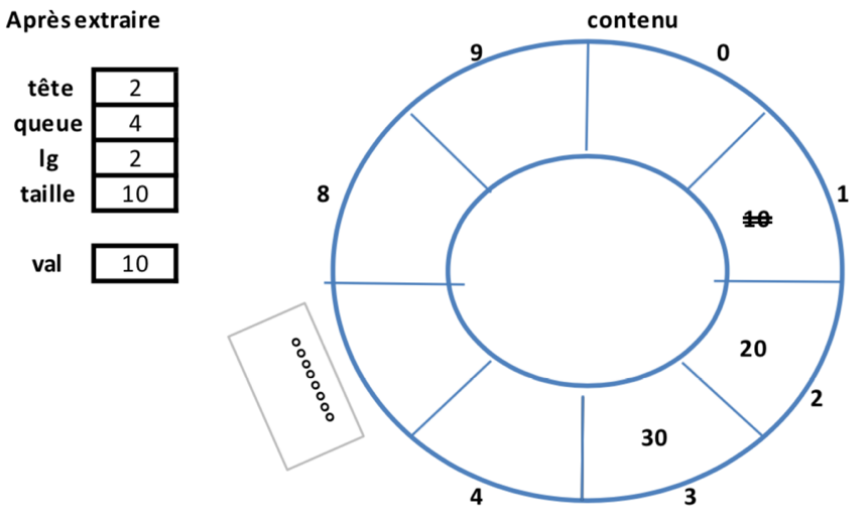
tête	1
queue	4
lg	3
taille	10

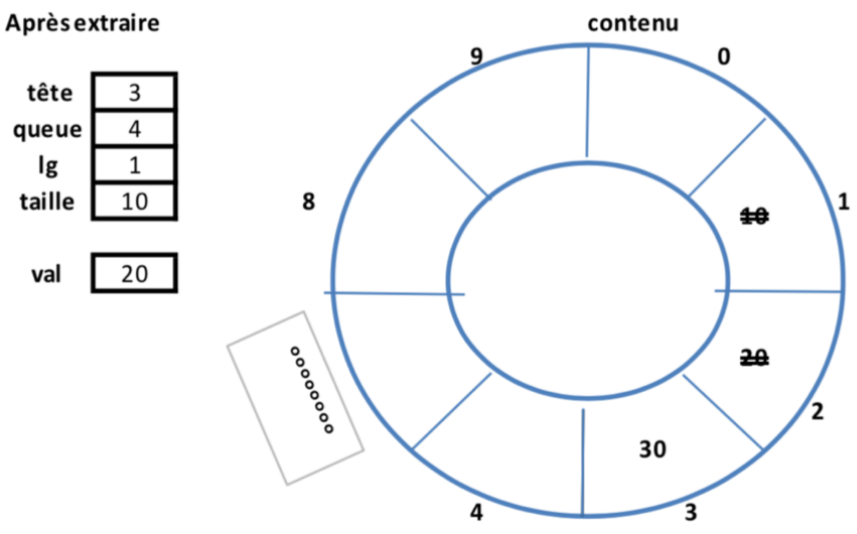
contenu



File implémentée par un tableau

Exemple d'évolution





File - Structure de données (implémentations)

File implémentée en utilisant une liste chainée de cellules

- Séquence logiquement ordonnée de cellules, afin de retrouver l'ordre des insertions de données :
  - Chaque cellule contient une information
  - Chaque cellule permet d'accéder ("pointe") sa suivante, c'est à dire l'information qui a été ajoutée après elle dans la file

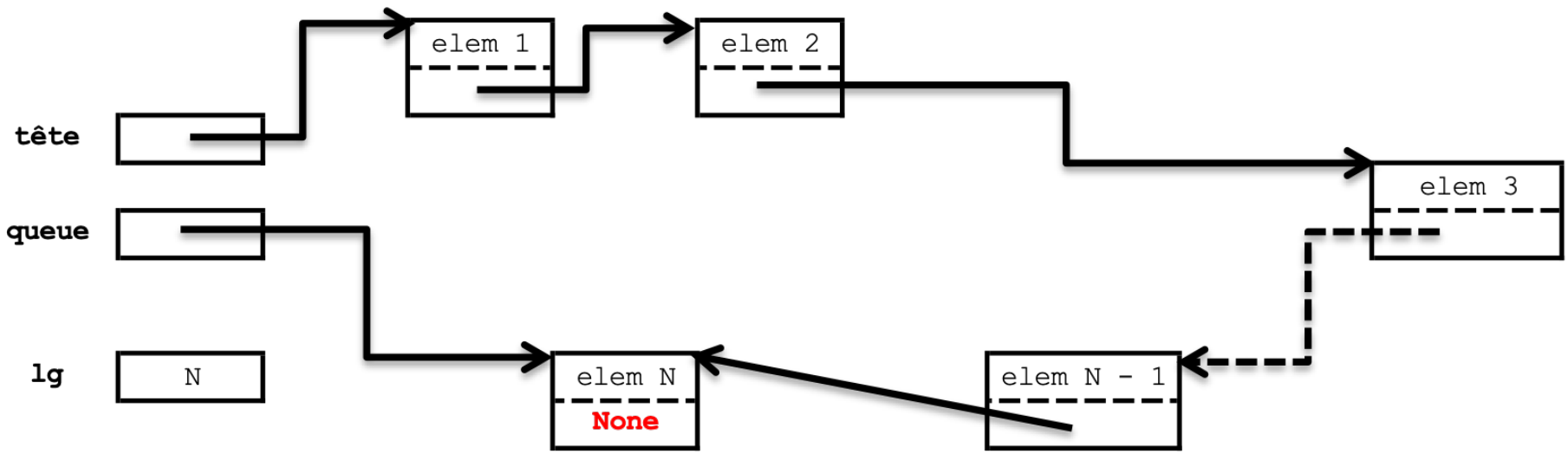
File - Structure de données (implémentations)

File implémentée en utilisant une liste chainée de cellules

- Variable tête qui "pointe" la cellule en tête de la file, qui "pointe" celle qui est après . . . et ainsi de suite
  - Valeur pointeur particulière None, NULL ou nil pour indiquer qu'il n'y a plus de cellule après
- Variable queue qui "pointe" la dernière information insérée dans la file, pour optimiser l'ajout d'une nouvelle information (sans avoir à parcourir toute la file pour retrouver la dernière cellule)
- Variable lg qui comptabilise le nombre d'éléments dans la file (pour ne pas avoir à le recalculer à chaque fois)
- file vide lorsque tête == None (ne "pointe" rien) ou aussi en utilisant lg == 0

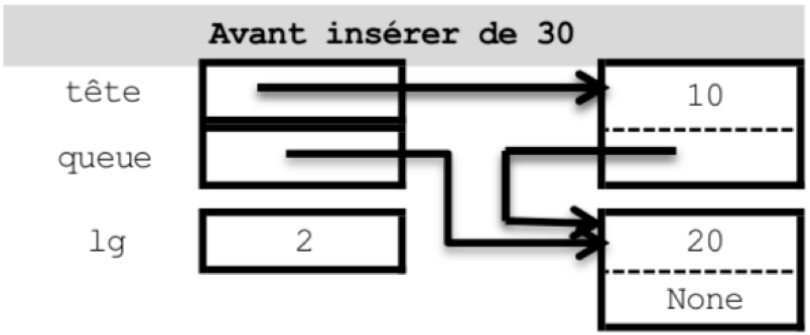
File - Structure de données (implémentations)

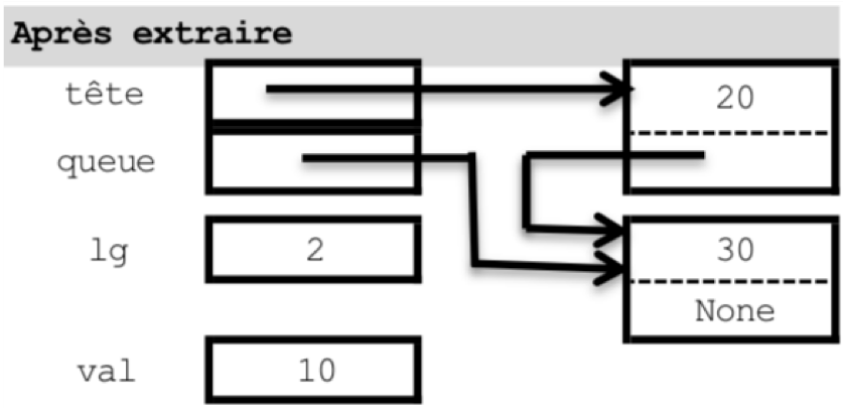
File implémentée en utilisant une liste chainée de cellules

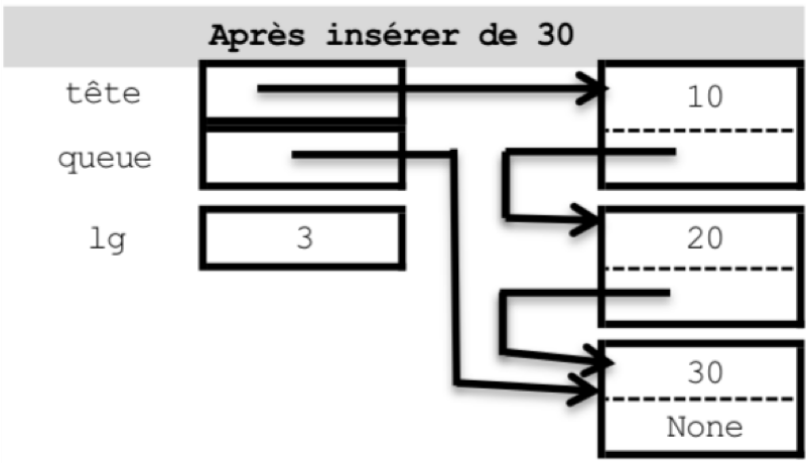


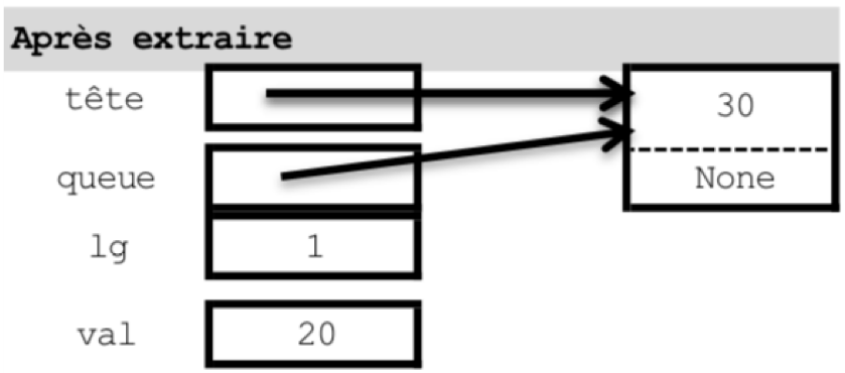
File implémentée en utilisant une liste chainée de cellules

Exemple d'évolution









Implémentation utilisée pour les exercices - Python

- On utilise la liste Python pour implémenter la file
- Une file est vide si la liste qui la représente est vide
- L'élément en tête de la file correspond au premier élément de la liste
- Insérer une valeur revient à l'insérer en fin de la liste (append)
- Extraire d'une file non vide retourne le premier élément de la liste, après l'avoir supprimé de la liste

Implémentation utilisée pour les exercices - Python

```python
# creer_file: retourne une file vide
def creer_file_vide():
    return []

# nombre d'elements dans la file
def nombre_elements(une_file):
    return len(une_file)

# file_vide: retourne True si la file une_file est vide
def file_vide(une_file) :
    return nombre_elements(une_file) == 0

# inserer: ajoute une_valeur en queue/fin de la file une_file
def inserer(une_file, une_valeur):
    une_file.append(une_valeur)
```

Implémentation utilisée pour les exercices - Python

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

Implémentation utilisée pour les exercices - Python

```
1  def test():
2    file=creer_file_vide()
3    for i in range(1,11):
4      inserer(file, i)
5    print('taile␣de␣la␣file␣apres␣remplissage␣=',
6                  nombre_elements(file))
7    afficher(file)
8    print("vider␣la␣file")
9    i=1
10   while not file_vide(file):
11     val = extraire(file)
12     print('i␣=', i, '␣val=', val,
13     'nouvelle␣taille␣=', nombre_elements(file))
14     i += 1
15   # Une exception pour un extraire en trop
16   print('Exception␣pour␣un␣extraire␣en␣trop')
17   val = extraire(file)
18 test()
```

Exercices - Temps d'attente de processus

Enonce

- Un processus est un programme en exécution. Lorsque plusieurs processus sont lancés en même temps, tous ne peuvent pas ou ne doivent pas être exécutés immédiatement (processeurs occupés ou heure souhaitée pour le début du processus).
- Ceux dont le traitement est différé sont placés dans une file d'attente en attendant leurs moments d'exécution (qui, heure souhaitée pour son début, durée).
- L'heure dans le système est décrite par un nombre entier, représentant le nombre de nanosecondes depuis une heure fixe.
- Pour apprécier la qualité du système, on souhaite calculer les temps d'attente des processus, en s'intéressant au traitement de la file d'attente des processus.

Exercices - Temps d'attente de processus

Enonce

- Chaque processus est caractérisé par un tuple qui comporte trois valeurs : `designation` (une chaîne de caractères), `heure_debut` (entier) et `duree` (entier).
- La simulation de l'exécution des processus consistera à faire avancer l'heure du système, `heure_systeme`, exprimée en nanosecondes, en fonction des caractéristiques des processus qui vont être exécutés.
- Ecrire une fonction `temps_attente(heure_systeme , file_processus)` qui, considérant que l'heure du système vaut `heure_systeme`, traite la file d'attente `file_processus` des processus en attente d'exécution.

Exercices - Temps d'attente de processus

Enonce

- Le traitement consistera à afficher pour chaque processus sa désignation, son heure de début demandée, son heure effective de début d'exécution et son retard (différence entre son heure de début demandée et l'heure à laquelle il a commencé à être exécuté).
- A la fin de cet affichage, l'heure du système après exécution des processus sera affichée.
- Cette fonction retourne la nouvelle heure du système.

Exercices - Temps d'attente de processus

Exemple

Si `file_processus` contient les demandes suivantes :

```
[ ("p2922", 2951, 47), ("p2923", 3010, 25),
  ("p2924", 3025, 49), ("p2925", 3044, 27),
  ("p2926", 3132, 34), ("p2927", 3103, 21) ]
```

L'appel `temps_attente(2921, file_processus)` doit produire :

```
(p2922, 2951, 2951, 0)
(p2923, 3010, 3010, 0)
(p2924, 3025, 3035, 10)
(p2925, 3044, 3084, 40)
(p2926, 3132, 3132, 0)
(p2927, 3103, 3166, 63)

Heure systeme:  3187
```