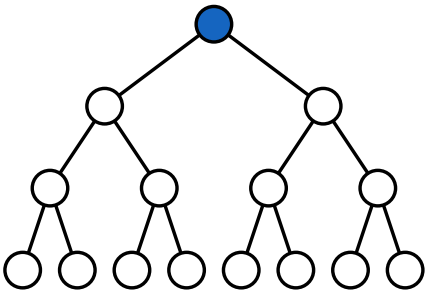
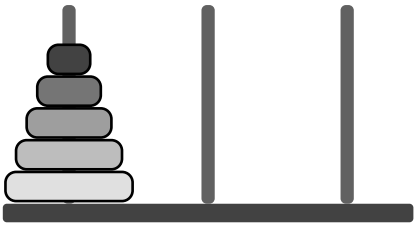
Numérique et Sciences Informatiques





Thème 1 – Structures de données

Terminale

2023-2024

Chapitre 1

Interface et Implémentation - Listes, piles, files

Ce cours présente de nouvelles façons d'organiser et de traiter les données, que l'on appelle des structures de données. On rencontrera, notamment, des structures linéaires comme la liste, la pile et la file, mais également des structures relationnelles telles que les arbres ou les graphes (à voir dans les chapitres suivant). Dans ce chapitre, nous allons commencer par distinguer la structure de données de son implémentation en illustrant les concepts avec le jeu ricochet robot.

Structure de données

Une structure de données est un format pour organiser les données dans le but de les traiter, de les extraire et de les stocker.

Le choix de la structure de données est important dès qu'on souhaite stocker et manipuler efficacement des données. Le choix se fait en fonction du **besoin**. Il est donc important de spécifier son besoin pour choisir la structure de données la plus adéquate.

Pourquoi existe-t-il plusieurs structures de données ?
Le choix de la bonne structure de données permet de gagner beaucoup de temps de calcul. Ainsi, certains programmes ne pourraient pas exister si on utilisait la mauvaise structure de données. En effet, la résolution deviendrait trop longue.

Commençons par les structures de données suivantes :
— Les listes
— Les files FIFO
— Les piles LIFO

Structures linéaires

Ces trois structures de données sont dites **linéaires** : les données sont organisées dans un ordre dans lequel les éléments sont liés les uns après les autres.

On différencie l'interface de l'implémentation. L'interface va spécifier théoriquement la structure de données en proposant des fonctions qu'on peut appliquer sur la structure tandis que l'implémentation et l'utilisation de ces structures avec un langage de programmation. Voici les définitions formelles :

L'interface

L'interface qui est un objet algorithmique spécifiant des méthodes (appelées primitives) pouvant être appliquées sur une structure de données.

L'implémentation

L'implémentation d'une structure de données dans un langage permettant sa mise en œuvre pratique

1.1 Les listes

Les listes regroupent des données de manière à pouvoir y accéder librement (contrairement aux files et aux piles, dont l'accès se fait respectivement en mode FIFO et LIFO). Comme vu l'année dernière, un tableau peut être représenté par une liste.

Par exemple, sur l'image ci-contre, à l'index 4, on lit la valeur 31.

Valeur	**45**	**154**	**58**	**78**	**31**	**5**	**74**
Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**

Comment les manipuler ?

On peut manipuler des listes grâce à des fonctions. En voici les **primitives** :

Insérer	ajoute un élément dans la liste. *Add* en anglais
Retirer	retire un élément de la liste. *Remove* en anglais
La liste est-elle vide ?	renvoie "vrai" si la liste est vide, "faux" sinon. *IsNull* en anglais
Nombre d'éléments dans la liste	renvoie le nombre d'éléments dans la liste. *Length* en anglais

C'est ce qu'on appelle **l'interface**.

Comment les manipuler avec python ?

```
# Creer une liste vide
l = []

# Creer une liste non vide
l = ["gauche", 3, 2.3, 7, "droite"]

# Taille de la liste
print(len(l)) # affiche la taille de la liste

# Acceder a un element par son indice
print(l[4]) # affiche "droite"

# Cas particulier
# Premier element
print(l[0]) # affiche "gauche"
# Dernier element
print(l[-1]) # affiche "droite"
print(l[len(l)-1]) # affiche "droite"

# Ajouter un element
l.append("haut")
print(l) # affiche ["gauche", 3, 2.3, 7, "droite", "haut"]

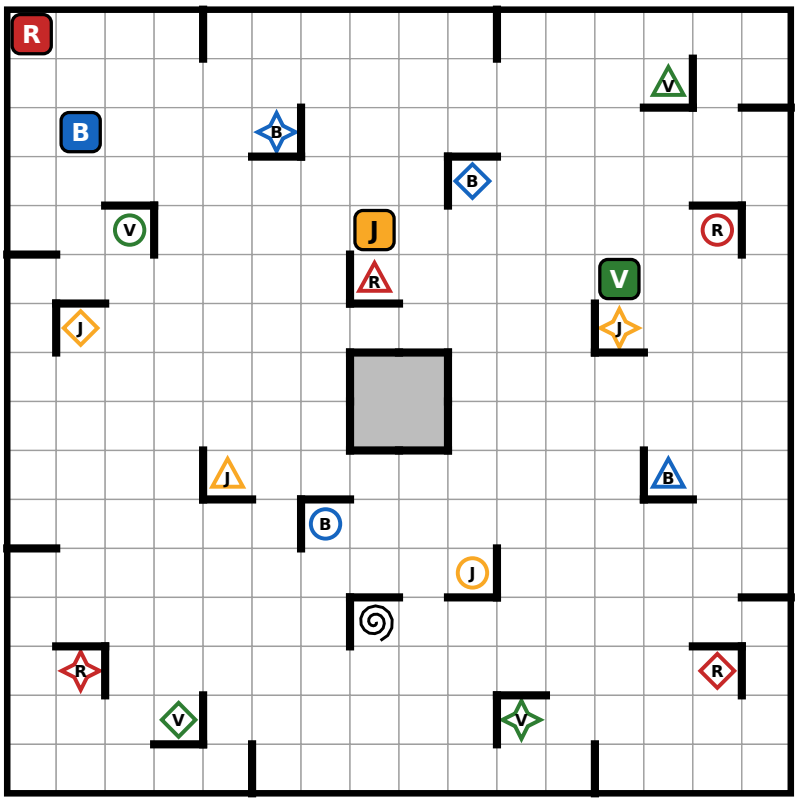
# Supprimer un element
del(l[3])
print(l) # affiche ["gauche", 3, 2.3, "droite", "haut"]
```

C'est ce qu'on appelle **l'implémentation**.

Remarque : on peut accéder par indice avec des indices positifs en partant du début de la liste ou avec des indices négatifs en partant de la fin de la liste.

liste	:	["A",	"B",	"C",	"D",	"E",	"F"]
indice positif	:	0	1	2	3	4	5
indice négatif	:	-6	-5	-4	-3	-2	-1

Exemple avec le jeu ricochet robot



Robots : R rouge, B bleu, J jaune, V vert.
Repères : cercle, triangle, losange, étoile.
La lettre dans le repère donne sa couleur.

Ceci est une grille où peuvent se déplacer les robots. Pour les déplacer on utilise les fonctions "deplacer_B()", "deplacer_R()", "deplacer_J()", "deplacer_V()", respectivement pour les robots bleu, rouge, jaune et vert. Ces fonctions prennent en paramètre "haut", "bas", "gauche" ou "droite". Quand on appelle deplacer_B(), le robot bleu se déplace d'une case dans la direction donnée en paramètre.

Exercice 1 :

```
D = ["haut", "bas", "gauche", "droite"]

deplacer_B(D[1])
deplacer_B(D[1])
deplacer_B(D[3])
```

Sur quel repère se trouve le robot bleu ?

Exercice 2 :

```
D = ["haut", "bas", "gauche", "droite"]

deplacer_J(D[len(D)-2])
deplacer_J(D[len(D)-2])
deplacer_J(D[len(D)-2])
deplacer_J(D[0])
deplacer_J(D[0])
deplacer_J(D[len(D)-1])
```

Sur quel repère se trouve le robot jaune ?

Exercice 3 :

```
D = ["haut", "bas", "gauche", "droite"]

deplacer_V(D[-1])
deplacer_V(D[1])
deplacer_V(D[-3])
deplacer_V(D[len(D)-3])
deplacer_V(D[4-3])
```

Sur quel repère se trouve le robot vert ?

Exercice 4 :

```
D = ["haut", "bas", "gauche", "droite"]
a = 2
b = 3
deplacer_V(D[a])
deplacer_V(D[1])
a = 1
deplacer_V(D[a])
deplacer_V("droite")
deplacer_V(D[b])
deplacer_V(D[0])
deplacer_V(D[b-a])
```

Sur quel repère se trouve le robot vert ?

Exercice 5 :

```
D = []
if not D :
    print("D est vide !")

D.append("bas")
D.append("droite")
deplacer_R(D[0])
D[0] = "droite"
deplacer_R(D[0])
D.extend(["haut", "bas", "gauche"])
deplacer_R(D[3])
deplacer_R(D[-2])
del(D[1])
if not D :
    print("D est vide !")
else :
    deplacer_R(D[len(D)-2])
    deplacer_R(D[0])
```

Sur quel repère se trouve le robot rouge ?

Exercice 6 :

```
D = ["haut", "bas", "gauche", "droite"]

deplacer_V(D[2])
for i in range(1,10) :
    deplacer_V(D[1])
deplacer_V(D[-2])
```

Sur quel repère se trouve le robot vert ?

Exercice 7 :

```
D = ["haut", "droite", "bas", "gauche"]

a = 5
while a > 0:
    deplacer_J(D[1])
    deplacer_J(D[-2])
    a -= 1

for i in range(3):
    deplacer_J(D[i])
```

Sur quel repère se trouve le robot jaune ?

Exercice 8 :

```
D = [["haut", "bas"], ["gauche", "droite"]]

deplacer_R(D[0][1])
for i in range(13):
    deplacer_R(D[1][1])
```

Sur quel repère se trouve le robot rouge ?

À votre tour de faire jouer les autres. Mettez-vous par groupe de 3 et faites une liste d'instructions pour déplacer un robot. Les autres groupes devront réussir à trouver sur quel repère le robot se trouve à la fin de votre programme.

1.2 Les files FIFO et les piles LIFO

Les files sont adaptées pour stocker des éléments suivant une séquentialité. Les files FIFO suivent la politique "premier entré, premier sorti" (*first in, first out* en anglais). Les premiers éléments ajoutés à la file seront les premiers à en être retirés ; tandis que les LIFO suivent la règle "dernier entré, premier sorti" (*last in, first out* en anglais).

Comment les manipuler ?

Enfiler	ajoute un élément dans la file. Le terme anglais correspondant est enqueue.
Défiler	renvoie le prochain élément de la file, et le retire de la file. *dequeue* en anglais.
La file est-elle vide ?	renvoie " vrai " si la file est vide, " faux " sinon.
Nombre d'éléments dans la file	renvoie le nombre d'éléments dans la file

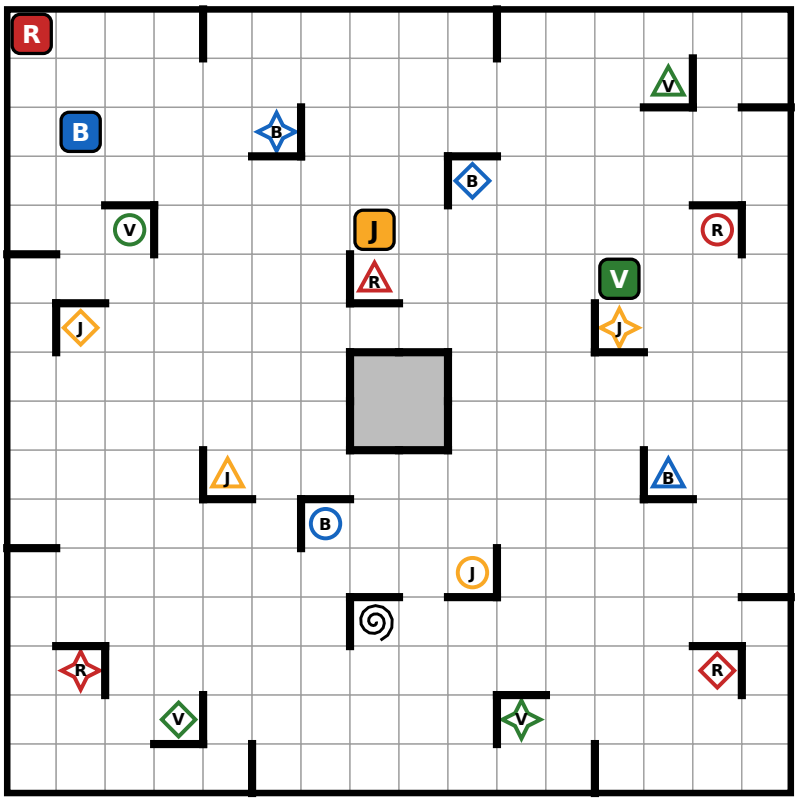
Comment les manipuler avec python ?

En utilisant des listes !

```
1
2   # Creer une file FIFO vide
3   fifo = []
4
5   # Creer une pile LIFO vide
6   lifo = []
7
8   # Enfiler les elements dans la file FIFO
9   fifo.append("gauche") # ajoute l'element en fin de liste
10  fifo.append("droite")
11  fifo.append("haut")
12  fifo.append("gauche") # pareil que append
13
14  # Enfiler les elements dans la pile LIFO
15  lifo.append("gauche")
16  lifo.append("droite")
17  lifo.append("haut")
18  lifo.append("bas")
19
20  # Defiler le prochain element
21  print(fifo.pop(0)) # retire et affiche le prochain element de la file
22  print(lifo.pop(-1)) # retire et affiche le prochain element de la pile
23
24  # Si la file n'est pas vide, retire puis affiche le prochain element
25  if len(fifo) > 0:
26      print(fifo.pop(0))
27
28  # Meme methode pour la pile
29  if len(lifo) > 0:
30      print(lifo.pop(-1))
31
32
33  # Compte le nombre d'elements dans la file
34  print(len(fifo))
35  # Meme methode pour la pile
36  print(len(lifo))
```

Les affichages sont-ils les mêmes en fonction de la structure utilisée ?

Exemple avec le jeu ricochet robot



Robots : R rouge, B bleu, J jaune, V vert.
Repères : cercle, triangle, losange, étoile.
La lettre dans le repère donne sa couleur.

Reprenons le même exemple que précédemment. Cette fois vous devez envoyer les instructions avec une file dans un premier temps, puis avec une pile dans un deuxième temps.

Exercice 1 :

```
fifo = []
fifo.append("bas")
fifo.append("bas")
fifo.append("droite")

while len(fifo) > 0:
    d = fifo.pop(0)
    deplacer_B(d)
```

Sur quel repère se trouve le robot bleu ?

Exercice 2 :

```
fifo = []
fifo.append("haut")
fifo.append("haut")
fifo.append("gauche")
fifo.append("gauche")
fifo.append("gauche")

while len(fifo) > 0:
    d = fifo.pop(0)
    deplacer_V(d)
```

Sur quel repère se trouve le robot vert ?

Exercice 3 :

```
# Creer une pile LIFO vide
lifo = []

for i in range(2):
    lifo.append("bas")
for i in range(4):
    lifo.append("droite")
for i in range(5):
    lifo.append("bas")
for i in range(2):
    lifo.append("gauche")

while len(lifo) > 0:
    d = lifo.pop(-1)
    deplacer_J(d)
```

Sur quel repère se trouve le robot jaune ?

A votre tour de faire jouer les autres !

Bravo, vous savez maintenant expliquer les principes théoriques d'une structure de données, en préciser l'interface et implémentation en python les listes, les piles et les files !