NSI Terminale - Structure de données

Résumé : structure de données liste

qkzk

2020/06/24

La structure de donnée liste

Objectifs

La structure de données linéaire **liste**.

- de *définir* la structure de données liste. Pour cela nous allons nous concentrer sur ses méthodes,
- de *manipuler* cette structure de données,
- d'appréhender la notion de *mutabilité* des listes (elles peuvent changer),
- d'appréhender la *complexité* de la manipulation des listes,
- de comprendre que ce qui est appelé `List` *en Python n'est pas une liste au sens commun du terme.*

La structure de donnée

Qu'est-ce qu'une liste ?

- Intuitivement. Une liste est une collection finie d'éléments qui se suivent. C'est donc une structure de données *linéaire*.
- Une liste peut contenir un nombre quelconque d'éléments y compris nul (la liste vide).

Obtenir une définition formelle

Prenons une liste comme par exemple '$\ell_1 = [3, 1, 4]$'. C'est une liste à trois éléments.

- la liste 'ℓ_1' possède un premier élément '3' qu'on nommera élément de *tête*,
- et que vient après cet élément de tête la liste '$\ell_2 = [1, 4]$' des éléments qui suivent, liste qu'on nommera *reste*.

Peut être répété pour la liste 'ℓ_2' qui est donc constituée :

- d'un élément de *tête* : '1',
- et d'un *reste* : '$\ell_3 = [4]$'.

La liste 'ℓ_3' est donc constituée :

- d'un élément de *tête* : '4',
- et d'un *reste* : '$\ell_4 = []$'.

'ℓ_4' étant vide : pas d'élément de tête, ne peut donc pas être décomposée comme nous venons.

Si on convient d'utiliser la notation '(x, ℓ)' pour désigner le couple constitué de l'élément 'x' de tête, et du reste 'ℓ' d'une liste, on peut alors écrire :

$$\text{'}\ell_1 = (3, (1, (4, [\,])))\text{'}$$

Définition

Une *liste* d'éléments d'un ensemble 'E' est

- soit la liste vide
- soit un couple '(x, ℓ)' constitué d'un élément '$x \in E$' et d'une liste 'ℓ' d'éléments de 'E'.

Les listes peuvent donc être vues comme *des structures de données récursives.*

Primitives sur les listes

Primitives Les *opérations primitives* d'une structure de donnée sont les opérations minimales qui permettent de définir la structure et de lui donner les méthodes attendues.

Primitives sur les listes

Une structure de donnée **liste** doit implémenter :

1. **La construction** à partir d'une liste vide ou à partir d'un couple tête (élément) et reste (liste).
2. **La sélection** de l'élément de tête ou du reste.
3. **Prédicat.** on doit pouvoir répondre à la question : "cette liste est-elle vide ?"

Liste vs tableaux

Qu'est ce qui différencie les listes des tableaux ?

les *tableaux* sont aussi des structures linéaires. Ils contiennent un nombre prédéfini d'emplacements mémoire (qu'on peut parfois modifier).

- On accède en temps constant à une valeur.
- Insérer un élément demande de décaler tous les suivants. C'est long.

Python utilise des tableaux dynamiques qu'il appelle `list` (parce qu'ils sont mutables).

Ce ne sont ni des "listes" ni des "tableaux" tels qu'on vient de les décrire.

Complexité

Pourquoi implémenter plusieurs structures ? Après tout, on peut *tout faire* avec des listes Python !

Parce que l'efficacité est fondamentale. Certaines structures sont plus adaptées à certains problèmes.

Accéder à un élément : tableau

Pour accéder à l'élément 2 du tableau

```
T = ['a', 'b', 'c']
```

1. On se rend à l'adresse où débute `T`
2. On se déplace de deux positions. On lit : `'c'`

Le temps est constant : Accéder se fait en complexité $O(1)$.

Accéder à un élément : liste

Pour accéder à l'élément 2 de la liste

```
L = ('a', ('b', ('c', [])))
```

1. On se rend à l'adresse où débute `L`
2. On suit le lien jusque l'adresse de la queue du premier élément
3. On suit le lien jusque l'adresse de la queue du second élément
4. On lit la valeur de la tête : `'c'`

Le temps est linéaire : Accéder se fait en complexité $O(n)$.

Une classe `Liste` minimale en Python

Implémenter une structure minimale

Pour implémenter une structure de donnée il faut :

1. avoir décrit les primitives
2. décrire les méthodes qui seront à construire plus tard
3. Utilise la programmation objet pour créer un nouveau *type* de données

Construction de la liste

Voici une manière **construire** :

```
class Liste: # avec un e !!!
  def __init__(self, *args):
    if len(args) == 0:
      self.__contenu = None
    else:

      self.__contenu = {
        "tete": args[0],
        "queue": args[1]
      }
```

```
>>> # Un exemple :
>>>
>>> l = Liste()
>>> g = Liste(1, l)
>>> h = Liste(2, g)
>>> isinstance(l, Liste)
True
```

Sélecteurs

```
  # toujours dans la classe List

  def tete(self):
    if self.__contenu is None:
      return None
    else:
      return self.__contenu["tete"]

  def queue(self):
    if self.__contenu is None:
      return None
    else:
      return self.__contenu["queue"]
```

```
>>> # Exemple
>>>
>>> g.tete()
1
>>> h.queue()
<__main__.Liste object at 0x7f66fbdecf70>
>>> isinstance(h, Liste)
True # c'est bien un objet de type liste
```

Prédicat : *est vide*

```
  # toujours dans la classe liste
  def est_vide(self):
    if self.__content is None:
      return True
    else:
      return False
```

```
>>> # Exemple
>>>
>>> h.est_vide()
False
>>> l.est_vide()
```

Méthodes utiles

Notre structure de donnée est incomplète

Il manque des méthodes pratiques : `longueur`, `index`, `ajouter`, `supprimer`.