TD : Algorithmes sur les listes

2 heures

Rédigé par Pascal Delahaye

21 décembre 2017

1 RAPPEL : Fonctions opérant sur les listes

Pour la construction d'algorithmes opérants sur des listes, on pourra représenter une liste donnée L par un tableau à une ligne :

$L[0]$	$L[1]$	$L[2]$	$L[3]$	$\ldots$	$L[n-1]$

Représentation d'une liste de taille n

Remarque 1. On retiendra impérativement que les indices des éléments d'une liste varient de 0 à $n-1$.

1.1 Les fonctions de base

En informatique, la définition d'une structure de donnée (ici la structure de LISTE) s'accompagne d'un certain nombre de fonctions de base qui permettent de construire des données ayant cette structure et de les utiliser. Parmi ces fonctions, on distingue les *fonctions de construction*, les *fonctions de sélection* et les *prédicats*.

Fonctions de constructions :

`[]`	représente la liste vide (très utile pour les initialisations !)
`[a,b...,e]`	permet de créer la liste dont les composantes sont `a`, `b`, ..., `e`
`[x]*n`	permet de créer une liste contenant n fois la composante `x`
`L1 + L2`	permet de concaténer deux listes `L1` et `L2`
`L[i] = a`	permet d'affecter la valeur `a` à la composante `L[i]`
`M=L.copy()`	recopie tous les éléments de `L` dans une nouvelle liste nommée `M`
`L.remove(x)`	permet d'éliminer la première valeur `x` de la liste `L`
`L.append(x)`	permet d'ajouter l'élément `x` en queue de liste

Les fonctions suivantes (de sélection et prédicats) sont également valables pour les chaînes de caractères :

Fonctions de sélection	
`len(L)`	renvoie la taille de la liste `L`
`L[k]`	renvoie le (k+1)ème élement de la liste `L` (les listes sont indexées à partir de 0).
`L[i:j:p]`	extrait la sous-liste des éléments d'indice compris entre `i` et `j-1` par pas de `p`.

Prédicat :	
`L1 == L2`	teste l'égalité de deux listes.

Remarque 2. Certaines fonctions précédentes ne sont pas des *fonctions de base* au sens propre car elles ont été construites à partir d'autres fonctions de base. On les a cependant placé dans cette catégorie car elles sont d'utilisation courante.

1.2 Les fonctions complémentaires sur les listes

Les fonctions suivantes sont construites à partir des fonctions de base précédentes.

Modification de la taille :	
`L.clear()`	réinitialise la liste L à la liste vide
`L.insert(i,x)`	permet d'insérer un élément x en position i dans la liste L
`L.pop(i)`	renvoie l'élément d'indice `i` tout en le supprimant de la liste `L`
`L.pop()`	renvoie le dernier élément de la liste `L` tout en le supprimant de `L`
Informations sur la liste	
`L.count(x)`	renvoie le nombre de fois où la valeur x est trouvée dans la liste L
`max(L)`	renvoie l'élément maximum de la liste `L`
`min(L)`	renvoie l'élément minimum de la liste `L`
`e in L`	nous dit si l'élément `e` est contenu dans la liste `L`
Modification de l'ordre	
`L.sort()`	permet de trier la liste L
`L.reverse()`	renvoie la liste inversée des éléments de L

2 Petits Exercices

Soit L une liste donnée.

1. Construire un programme de calcul de la moyenne et deux programmes de calcul de l'écart-type des éléments L en utilisant les formules suivantes :

   Lorsque $x = \{x_i\}_{i\in[\![1,n]\!]}$ on a : $\bar{x} = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} x_i$ et $\sigma = \sqrt{\dfrac{1}{n}\displaystyle\sum_{i=1}^{n}(x_i-\bar{x})^2}$ ou $\sigma = \sqrt{\overline{x^2}-(\bar{x})^2}$

   Construire une liste de taille 100 dont les composantes sont choisies aléatoirement dans [0, 100] et appliquer les fonctions précédentes à cette liste.

2. La syntaxe `len(L)` permet de déterminer la longueur de la liste L.
   Proposez un programme effectuant la même opération (on pourra utiliser la fonction pop()).

3. La syntaxe `e in L` permet de tester si l'élément e est contenu dans la liste L.
   Proposez un programme effectuant la même opération.

4. La syntaxe `L.count(x)` permet de compter le nombre de fois où apparaît x dans la liste L.
   Proposez un programme effectuant la même opération.

5. La syntaxe `L.reverse()` permet d'inverser "en place" l'ordre des éléments de la liste L.
   Proposez un programme effectuant la même opération.

6. La syntaxe `L1 == L2` permet de tester l'égalité des deux listes L1 et L2.
   Proposez un programme effectuant la même opération.

7. Les syntaxes `max(L)` et `min(L)` permettent d'extraire le maximum et le minimum de la liste L.
   Proposez un programme effectuant l'une de ces deux opérations.

Remarque 3. Les algorithmes de tri seront abordés à une nouvelle occasion...

3 Recherche d'un mot dans une liste ou une chaine de caractères

On considère une phrase "P" un mot "M" sous la forme de deux chaînes de caractères.
On dira que le mot "M" est contenu dans la phrase "P" lorsque la chaîne M est incluse dans la chaine P.
On souhaite ici rédiger un programme permettant de déterminer si le mot "M" est contenu ou non dans la phrase "P" et de compter combien de fois le mot apparaît dans "P".

Exercice : 1
(∗) Construire un tel programme en utilisant la fonction d'extraction d'une sous-chaine `P[i:j:p]` et la fonction permettant de comparer l'égalité de deux chaines `M1 == M2`.

Exercice : 2
(∗∗) On souhaite construire un tel programme en utilisant l'algorithme suivant :

- On note p la longueur de P et m la longueur de M.
- On crée une liste L destinée à stocker tous les indices i tels que $M = [P[i],\ P[i+1],\ \ldots,\ P[i+m-1]]$. Cette liste est initialisée à [].
- A l'aide d'une boucle "for", on recherche tous les indices $i \leq p-m$ tels que $P[i] = M[0]$.
  On stocke ces indices dans une liste `Indices`.
- Pour chacun de ces indices "i", on teste si $M = [P[i],\ P[i+1],\ \ldots,\ P[i+m-1]]$. Si c'est le cas, on ajoute i à la liste L.
- La longueur de L nous donne alors le nombre d'occurences du mot "M" dans la phrase "P" et les valeurs $L[i]$ nous donne les positions du mot "M" dans la phrase "P".

1. Construction de sous-procédures :

   (a) Construire une procédure `position1(P,M)` qui renvoie la liste L contenant les indices "i" tels que $P[i] = M[0]$ parmi les composantes $P[0],\ \ldots, P[p-m]$.

   (b) Construire une procédure `test(P,M,i)` qui teste l'égalité $M = [P[i],\ P[i+1],\ \ldots,\ P[i+m-1]]$.
   On interdit ici l'utilisation de la commande `M == P[i:i+m:1]`.

2. En déduire une procédure qui répond au problème posé.

Remarque 4. A l'aide de la fonction `L.remove(i)`, on peut modifier l'algorithme en testant pour tout i de L, l'égalité `M[k] = P[i+k]` pour tout $k \in [\![0, m-1]\!]$. On élimine alors l'indice `i` de la liste L dès que cette égalité n'est pas vérifiée.

Remarque 5. **Polymorphisme :**
En Python, comme les chaînes de caractères, les listes sont des itérables. Celui signifie que l'on peut en général utiliser les mêmes fonctions sur les unes et sur les autres. Vérifiez que vos deux algorithmes précédents fonctionnent correctement si on les appliquent à des listes d'entiers au lieu de chaînes de caractères.