Chapitre 2 : Listes en python

Le type `list` permet d'implémenter les tableaux (de manière dynamique) en python.

1 Qu'est-ce qu'un tableau ?

D'un point de vue algorithmique, un tableau est une structure de données qui permet de stocker une **suite ordonnée** de valeurs `x0,x1,...,xi...,xn−1` : * chaque valeur est repérée par son indice `i` (commençant à 0); * l'accès à une case à partir de son indice `i` se fait en **temps constant**, c'est-à-dire que ce temps ne dépend pas de `i`, ni de la longueur `n` du tableau : il est donc aussi rapide d'accéder à la première valeur qu'à la 200ème (par exemple).

Ceci est possible car dans un tableau : * toutes les cases ont la même taille et occupent donc le même espace mémoire ; * les valeurs sont stockées dans des ***cases contiguës*** de la mémoire de l'ordinateur (appelée RAM : Random Acess Memory) du système informatique.

Le nombre de cases dans un tableau (non dynamique) est fixe.

2 Le type `list` en Python

Python utilise le type `list`pour implémenter les tableaux ; cependant il permet, en plus, au tableau d'adapter ***dynamiquement*** leur taille en fonction des insertions/suppressions d'éléments. Cependant, il est important de comprendre que ces mécanismes bien qu'automatiques représentent un coût temporel.

3 Récapitulatif des opérations permises par le type `list`

Opération	**Syntaxe**	**Complexité**
Création d'une liste vide	`[]`	O(1)
Lecture d'un élément	`x = L[i]`	O(1)
Ecriture d'un élément	`L[i] = x`	O(1)
Ajout d'un élément à la fin	`L.append(x)`	O(n)
Insertion d'un élément	`L.insert(indice,x)`	O(n)
Copie d'une liste	`L2 = L1.copy()`	O(n)
Test d'appartenance	`x in L`	O(n)

4 Optimisation lors des insertions

Python optimise les temps d'exécution en gérant "judicieusement" la capacité des listes. Ces mécanismes permettent de limiter les réallocations intempestives à chaque insertion de valeur : * lors de sa création, une liste a une capacité maximale (nombre de cases disponibles) fixée (arbitraire-

ment par Python). Les insertions (`append()` ou `insert()`) se font “normalement” dans l’espace mémoire alloué pour la liste ; * lorsque la capacité maximale est atteinte, Python opère plusieurs opérations pour agrandir la liste : * création d’une autre liste avec une capacité double (généralement) de la précédente ; * recopie des élements de l’ancienne liste dans la nouvelle liste ; * rajout de l’élément qu’on cherchait à insérer.

Tout cela permet de minimiser les réallocations intempestives de nouvelles listes à chaque insertion lorsque la capacité maximale de la liste est atteinte. Ce mécanisme permet de rendre a peu près constant en moyenne le temps d’exécution pour une insertion à la fin du tableau.