Structures de données
Introduction et listes

Master 1 MISO

Camille Marchet
CNRS, CRIStAL Lille, France

camille.marchet@univ-lille.fr

Préambule - Objectifs de ce cours

- Une meilleure vision de ce que sont des "structures de données" et ce qu'elles impliquent
- Culture G en informatique via différentes notions
- Quelques réflexes en programmation
- Rattacher ces connaissances à la bioinformatique

Introduction - Structures de données ? Dans la vie réelle

Une *structure* (par exemple un meuble, une boite) pour *organiser, ranger* des éléments (matériels ou immatériels)

- Une bibliothèque
- Des placards
- Un classeur
- Un album photo
- Un annuaire
- Un dictionnaire
- Un arbre généalogique

Introduction - Structures de données? En informatique

- idée de poupées russes
- notion d'implémentation ou non
- idée de coût pour différentes opérations

Quelles structures de données connaissez-vous ?

Introduction - Structures de données ? Opérations

Opérations de base lorsqu'un ensemble d'éléments est dans une structure de données :

- Tester si l'ensemble est vide
- Ajouter/supprimer un élément à l'ensemble
- Vérifier si un élément appartient à l'ensemble
- Parcourir les éléments de l'ensemble

Introduction - Structures de données ? Opérations

un paquet initial de k-mers :
ATTA, GTAG, TTGA,
ACCT, CTAA, **CCCC**

structure de données 1
set

structure de données 2
liste triée

[ACCT,ATTA,CCCC,CTAA,GTAG,TTGA]

Introduction - Gestion concrète des structures de données

Deux critères essentiels basés sur les limites physiques d'un ordinateur :

- la place mémoire utilisée
- le nombre d'opérations pour traiter les données (le temps de calcul)

Introduction - La complexité asymptotique

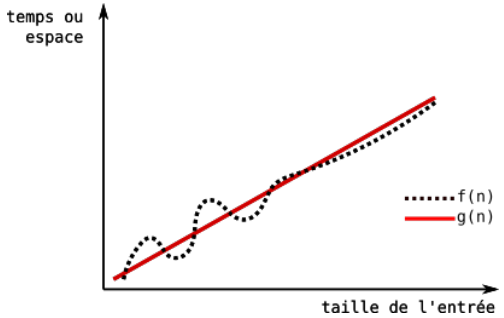
- un programme tourne en des temps différents sur 2 machines, pourtant c'est le même avec le même input, comment en parler ?
- comment anticiper si mon algorithme/ma structure de données va "aller vite" sur mes données (ou prendre de la place) ?

Introduction - La comparaison asymptotique (on dit parfois "complexité")

$\rightarrow$ Donner un ordre de grandeur du temps de calcul/de la place prise en fonction de la taille de l'entrée.

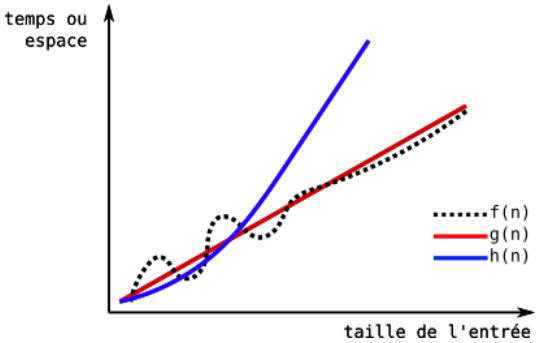
Introduction - La comparaison asymptotique (on dit parfois "complexité")

Fonction g telle qu'**à partir d'un certain seuil,** f **est toujours dominée par** g (intuition : f ne croit pas plus vite que g en l'infini, à une constante multiplicative près).



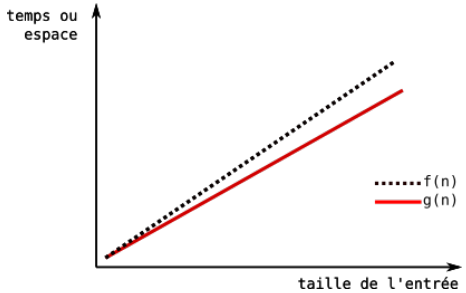
$f(n) = \mathcal{O}(g(n))$ (notation de Landau)

Introduction - La comparaison asymptotique (on dit parfois "complexité")



Introduction - La comparaison asymptotique (on dit parfois "complexité")

Fonction g telle qu'à partir d'un certain seuil, f est toujours dominée par g **à une constante multiplicative près**.



$$f(n) = \mathcal{O}(g(n))$$
$$6x^2 + 3 = \mathcal{O}(x^2)$$

Introduction - La complexité asymptotique

Quelques classes de complexité vues couramment dans la vie de bioinformaticien(ne).

Constante	$\mathcal{O}(1)$	indépendante de la taille de l'entrée
Logarithmique	$\mathcal{O}(log(n))$	ex : recherche dichotomique
Linéaire	$\mathcal{O}(n)$	ex : lire un fichier
Quasi-linéaire	$\mathcal{O}(n \times log(n))$	ex : un bon tri
Quadratique	$\mathcal{O}(n^2)$	ex : alignement de 2 séquences

Au delà, pour de gros jeux de données, c'est très problématique . . . (quadratique : 1 jour pour une entrée de taille 1 $\rightarrow$ 100 jours pour une entrée de taille 10)

Introduction - exercice

Quelle serait la complexité d'un algorithme naïf pour reporter toutes les positions des occurrences d'un mot de taille fixe (par exemple, "ATGGTATA") dans le génome humain ?

Introduction - exercice

Linéaire en la taille du génome : $\mathcal{O}(n)$

TGATAGTAGATATGGTATAATGCCAG .... GATTGA
1 2 n

- identifier les opérations atomiques
- puis combien de fois on les fait en fonction de la taille de l'entrée
- écrire en notation de Landau

Les listes - en Python

```
my_list = ["AGGTA", 1, [89, 'b']]
```

- Notion d'ordre
- Les éléments peuvent être redondants
- Hétérogènes (plusieurs types : entiers, chaines de caractères ...)
- Mutables : un élément peut changer de valeur et de type
- Méthodes `append()`, `pop()`, slicing `l[:3]`, `l[1:6]` ..., opérateur `in`, concaténation

Les listes - en Python. Exercice

En Python, comment

1. ajoute-t-on à la fin d'une liste ?
2. accède-t-on au i-ème élément d'une liste ?
3. accède-t-on du i-ème élément au j-ème élément d'une liste ?
4. connait-on la longueur d'une liste ?

Les listes - en Python. Exercice

Résolution : commençons par créer une liste.
Que contient cette liste `l` :

```
l = [i for i in range(10)]
```

Les listes - en Python. Exercice

Résolution :

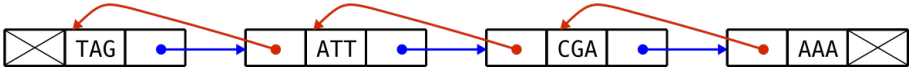
```
 l.append(10)
l += [10]
i=3
j=5
l[i-1:j]
len(l)
```

Les listes - en informatique

- collection finie, ordonnée d'éléments qui se suivent
- structure de données linéaire,
- nombre quelconque d'éléments, y compris nul

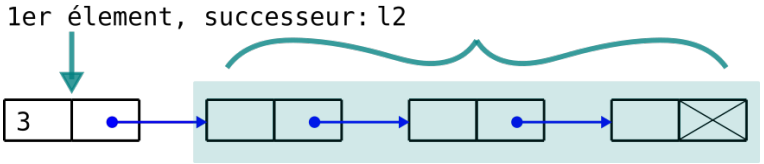
`simplement chaînée`

`doublement chaînée`



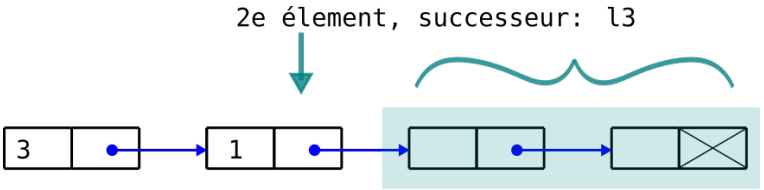
Un exemple de liste (en tant que structure de données)

```
l =[3, 1, 4]
```



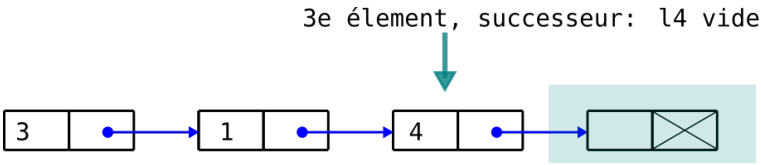
Un exemple de liste (en tant que structure de données)

```
l =[3, 1, 4]
```



Un exemple de liste (en tant que structure de données)

```
l =[3, 1, 4]
```



Une définition récursive !

Définition d'une liste en tant que structure de données **récursive**

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.

Opérations sur les listes

Une structure de données seule est inutile, il faut des fonctions pour l'interroger ou la modifier

- Les listes sont définies par une tête et un reste, il faut pouvoir accéder à ces deux éléments.
- On veut aussi savoir si une liste est vide ou non.

Intro au TP

On va réaliser une classe `List`.
Elle aura les opérations usuelles :

```
l = List()
l
()
l.is_empty()
True
```

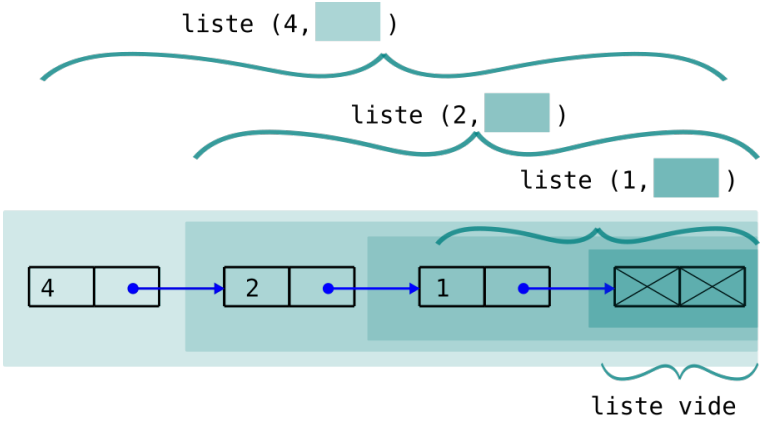
instancier et initialiser une liste, vérifier qu'elle est vide

Intro au TP

```
l3
(2.(1.()))
l3.head()
2
l3.tail()
(1.())
```

représenter les listes, accéder au **premier élément** et au **reste**.

TP listes



dans ce TP : (4.(2.(1.())))
en Python natif : [4, 2, 1]

Frame Title

- Constructeur : `__init__` en Python. C'est lui qu'on appelle quand on crée une nouvelle liste `my_list=List()`.
- Sélecteurs : `my_list.head()` et `my_list.tail()`
- Prédicat : `my_list.is_empty()` ou `my_list.tail().is_empty()`

Sélecteurs Les listes non vides possèdent une tête et un reste. Il nous faut les sélecteurs pour accéder à ces deux composantes.
Prédicat

RDV sur le Gitlab

`https://gitlab-etu.fil.univ-lille1.fr/m1-miso/sd-listes/`

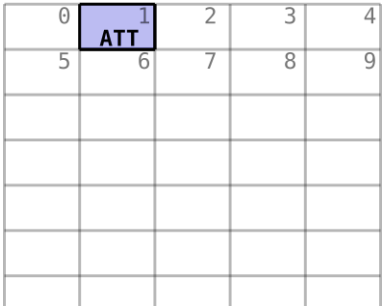
- git = gestionnaire de version
- clone ou fork du dépôt
- documentation git `https://www.cristal.univ-lille.fr/TPGIT/`
- si besoin d'accès au gitlab hors du campus, VPN : `https://sciences-technologies.univ-lille.fr/informatique/stock-pages-speciales/documentation-vpn`

Rendu du tp : une archive (`.tar.gz` ou `.zip` du code + vos réponses dans un `README.md`)

Les listes - Comment sont-elles implémentées ?

Via une structure de données : un tableau dynamique

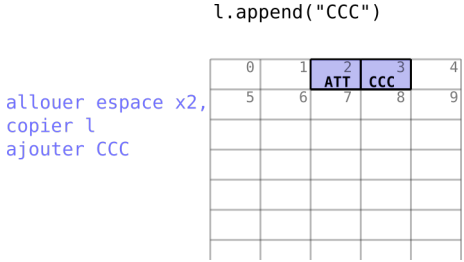
```
l = ["ATT"]
```



Les listes - Comment sont-elles implémentées ?

Via une structure de données : un tableau dynamique

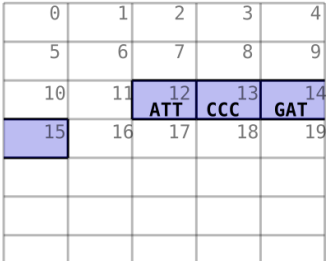
Tableau dynamique → ré-allouer de l'espace quand on arrive à la limite de la taille du tableau, faire une copie du tableau dans ce nouvel espace



Les listes - Comment sont-elles implémentées ?

```
l += ["GAT"]
```

allouer espace x2,
copier l
ajouter GAT



NB: précédemment j'ai utilisé `append("CCC")`, ici `+=["GAT"]`

Les listes - Comment sont-elles implémentées ?

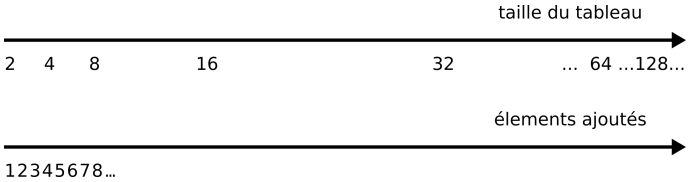
```
l += ["GGA", "ATT"]
```

allouer espace x2,
copier l
ajouter GGA

ajouter ATT

ou `l.extend(["GGA", "ATT"])`

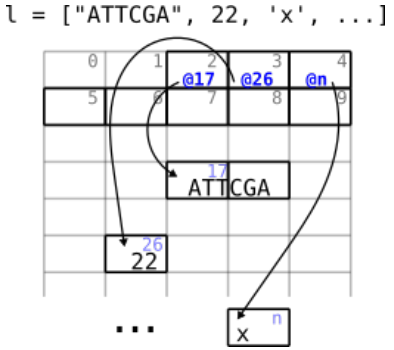
Les listes - Complexités



- $\mathcal{O}(n)$ complexité amortie pour construire la liste
- Ajout en temps constant dans le cas général
- Accès aux éléments en temps constant (`my_list[3]`)

Les listes - le passage par *référence*

- Les listes sont hétérogènes : on peut mettre dans une liste des éléments de différents types (entier, caractère, flottant, . . .)
- La liste ne contient en fait pas directement les éléments mais des références vers ces éléments



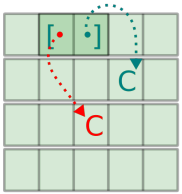
Les listes - exercice

1. Sans la console, savez-vous ce que donne
   ```
   l = [["C"] *2] *3
   print(l)
   ```
2. Avec la console, que se passe-t-il quand on fait:
   ```
    l = [["C"] *2] *3
    l[0][0] = "A"
   print(l)
   ```

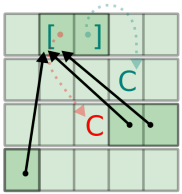
Indice : utiliser `id(l[0]), id(l[1]), id(l[2])` pour comprendre

Les listes - exercice

[["C"] *2] *3



[["C"] *2] *3

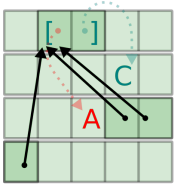


[['C','C'],['C','C'],['C','C']]

id(l[0])
= id(l[1])
= id(l[2])

Les listes - exercice

l[0][0] = 'A'



[['A','C'],['A','C'],['A','C']]

id(l[0])
= id(l[1])
= id(l[2])

Les listes - la copie en Python

```
l = [["ACC"] *2] *3
```
ou
```
a=b
```
sont des copies "faibles" (on garde la même référence pour plusieurs objets)

Pour faire une copie forte :
```
import copy
b = copy.copy(a)
```

Les listes - exercice

Quelle différence entre :

```
g = ["AAT"]
m = ["CCC", "GAG"]
n = m
m = m + g
```

et

```
g = ["AAT"]
m = ["CCC", "GAG"]
n = m
m += g
```

Les listes - en bioinformatique

- `vector` en C++ (attention un seul type)
  $\rightarrow$ un bloc de base utilisé partout

Récap listes - Qu'a-t-on vu ?

- Implémentation et complexité des listes en Python
- Tableau dynamique
- Notion de complexité amortie
- Passage par référence et ses implications quand on programme

Un dernier point sur list versus array

Array dans numpy ressemble aux listes.
On peut remarquer que :

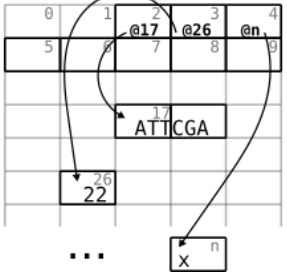
- `array` n'enregistre que des éléments du même type
- à nombre d'élements équivalent `array` est plus rapide pour les accès et prend moins de place en RAM

Ce sont deux tableaux dynamiques, alors pourquoi ?

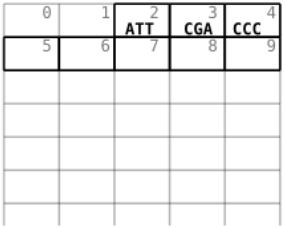
Un dernier point sur list versus array

`array` = tableau dense avec variables d'un même type
Le nom de la structure est plus cohérent avec son implémentation !

```
l = ["ATTCGA", 22, 'x', ...]
```



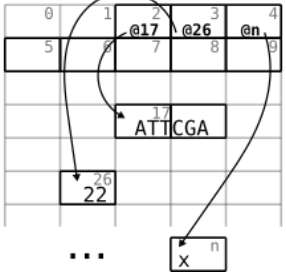

```
t = numpy.array(["ATT", "CGA", "CCC", ...])
```

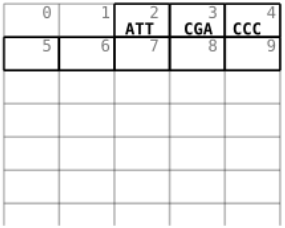


Un dernier point sur list versus array

→ notion de localité importante pour les performances

```
l = ["ATTCGA", 22, 'x', ...]   t = numpy.array(["ATT", "CGA", "CCC", ...])
```





Récap structures de données de type liste en Python

Qu'a-t-on vu ?

- Structures hétérogènes/homogènes, types et tailles des types
- Notion de localité
- Limites de la connaissance de la comparaison asymptotique