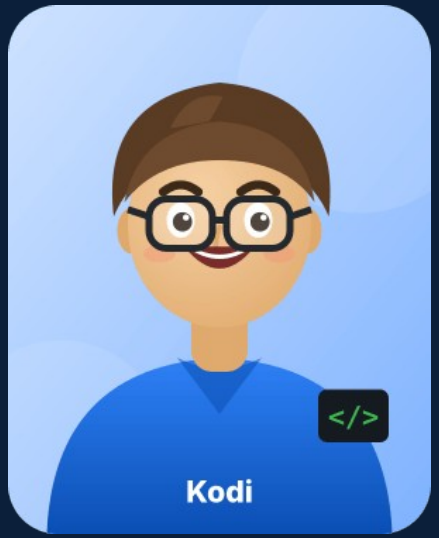
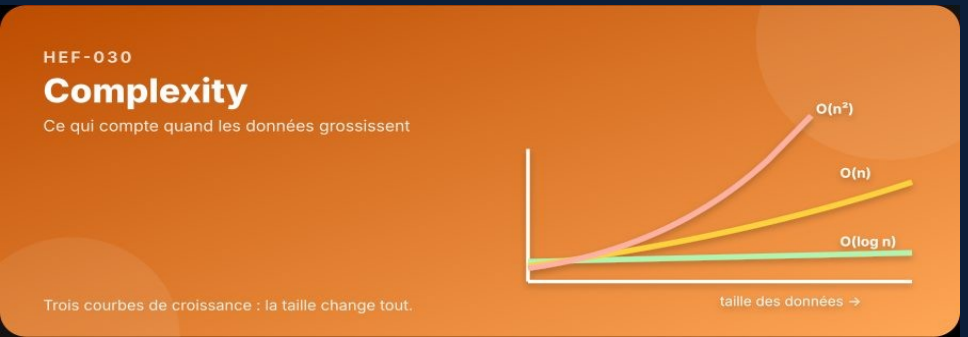
HEF-030 · MISSION 05 / 10

Le coût des structures de données

Chaque structure a ses forces et ses faiblesses. Les connaître évite 90 % des lenteurs.





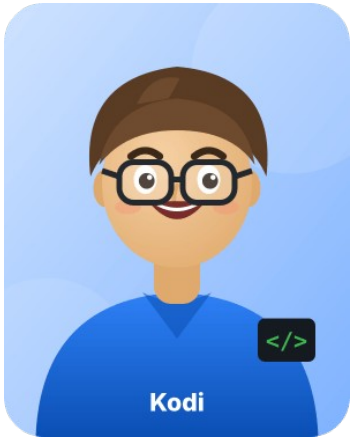
Badge : Connaisseur des coûts

70 min · débutant

L'histoire

Un matin, Maya remarque que la page « mes missions » met 12 secondes à s'afficher pour les membres les plus actifs. Kodi cherche pendant une heure. Sam jette un œil et pointe une ligne :

> *« Tu fais if mission in liste_missions dans une boucle. Tu parcours la liste entière à chaque tour. »*



Tu apprends le tableau des coûts à connaître par cœur pour ne plus tomber dans ce piège.

Ta mission

À la fin, je serai capable de…

1. citer le coût des opérations de base des structures courantes
2. repérer une structure mal choisie dans du code lent
3. expliquer le coût amorti d'un ajout à la fin d'un tableau dynamique
4. choisir une structure à partir de l'opération dominante
5. corriger un code lent en changeant seulement de structure

Défi d'échauffement

Défi d'échauffement

Tu as une liste de 100 000 numéros de membres et tu dois vérifier **1 000 fois** si un numéro y figure. Combien de comparaisons environ si tu parcours la liste à chaque fois ? Et si tu avais un moyen de savoir « présent / absent » en une seule opération ? Quelle structure du module HEF-020 te le permettrait ?

Le concept

Le tableau des coûts

Structure	Lire par indice	Chercher une valeur	Ajouter	Supprimer
Tableau (`list`)	O(1)	O(n)	fin : O(1) amorti · début/milieu : O(n)	O(n)
Liste chaînée	O(n)	O(n)	début : O(1)	O(1) si on tient déjà le maillon
Pile	:	:	O(1)	O(1)
File (deque)	:	:	O(1)	O(1)
Dictionnaire / ensemble	:	O(1) en moyenne	O(1) en moyenne	O(1) en moyenne
BST équilibré	:	O(log n)	O(log n)	O(log n)

Le concept

Le tableau des coûts (suite)

Structure	Lire par indice	Chercher une valeur	Ajouter	Supprimer
Tas	:	min : O(1)	O(log n)	min : O(log n)

Le coût amorti

Quand un tableau dynamique est plein, Python le **recopie dans un tableau plus grand** (coût O(n)). Mais cela arrive rarement : en répartissant ce coût sur tous les ajouts, l'ajout à la fin coûte **O(1) en moyenne**. On parle de **coût amorti**.

Le coût moyen des tables de hachage

Une table de hachage est O(1) **en moyenne**. Dans le pire cas (beaucoup de collisions), elle dégénère vers O(n). Avec une bonne fonction de hachage, ce cas est très rare.

Le concept

Les erreurs classiques

Code lent	Pourquoi	Version rapide
`if x in liste` dans une boucle	O(n) par test	`ensemble = set(liste)`
`liste.pop(0)` répété	O(n) à chaque retrait	`deque.popleft()`
`liste.insert(0, x)` répété	O(n) à chaque insertion	`deque.appendleft(x)`
`texte += morceau` dans une boucle	recopie répétée	`"".join(morceaux)`
`liste.sort()` après chaque ajout	O(n log n) par ajout	heapq ou insertion triée

Mesurer

On peut vérifier ces coûts avec `time.perf_counter()` ou `timeit`, en doublant n et en regardant comment le temps évolue.

Le concept

Mesurer (suite)

🧠 ***Astuce*** *: en lisant du code lent, cherche* ***d'abord*** *les in, index, count, insert(0, ...), pop(0) et les concaténations dans des boucles.*

Exemple pas à pas

Problème : trouver les missions terminées par un membre qui sont aussi dans le catalogue.
Version lente :

```
import time

catalogue = list(range(20000))
terminees = list(range(0, 40000, 2))

def lente(terminees, catalogue):
    return [m for m in terminees if m in catalogue]    # `in` sur une liste : O(n)

def rapide(terminees, catalogue):
    ensemble = set(catalogue)                          # O(n) une fois
    return [m for m in terminees if m in ensemble]     # O(1) par test

t = time.perf_counter(); a = lente(terminees, catalogue); t1 = time.perf_counter()
    - t
```

Exemple pas à pas (suite)

```
t = time.perf_counter(); b = rapide(terminees, catalogue); t2 = time.perf_counter(
   ) - t
assert a == b
print(round(t1, 3), round(t2, 5))
```

Analyse : si `terminees` a m éléments et `catalogue` n éléments, la version lente coûte O(m × n), la rapide O(m + n). Aucune autre ligne n'a changé : seule la **structure** du catalogue a changé.

À toi, avec aide

Mesure la différence entre `liste.pop(0)` et `deque.popleft()` pour vider une collection de n éléments avec n = 10 000, 20 000, 40 000.

1. Écris le code de mesure.
2. Que se passe-t-il quand n double, pour chaque version ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Construis `list(range(n))` et `deque(range(n))`.

Indice 2

Vide avec une boucle `while` sur chacune.

Indice 3

Compare les temps à n et à 2n.

À toi, avec aide · réponse

Réponse

```
import time
from collections import deque

def vider_liste(n):
    l = list(range(n))
    t = time.perf_counter()
    while l:
        l.pop(0)
    return time.perf_counter() - t

def vider_deque(n):
    d = deque(range(n))
    t = time.perf_counter()
    while d:
        d.popleft()
    return time.perf_counter() - t

for n in (10000, 20000, 40000):
    print(n, round(vider_liste(n), 4), round(vider_deque(n), 4))
```

Réponse (suite)

La version liste est O(n²) au total : le temps est multiplié par environ 4 quand n double. La version deque est O(n) : il est multiplié par environ 2.

À toi, en solo

Voici une fonction lente. Trouve **tous** les coûts cachés, puis réécris-la avec les bonnes structures.

```
def noms_populaires(commentaires):       # commentaires : liste de (nom, texte)
    resultat = []
    for nom, texte in commentaires:
        compte = 0
        for n, t in commentaires:
            if n == nom:
                compte += 1
        if compte >= 3 and nom not in resultat:
            resultat.append(nom)
    return resultat
```

À toi, en solo · réponse

Pistes de correction

Coûts cachés : la boucle interne recompte pour chaque commentaire ($O(n^2)$) ; `nom not in resultat` parcourt une liste ($O(k)$).

```
def noms_populaires(commentaires):
    comptes = {}
    for nom, _ in commentaires:
        comptes[nom] = comptes.get(nom, 0) + 1
    return [nom for nom, c in comptes.items() if c >= 3]

coms = [("a", "x"), ("b", "y"), ("a", "z"), ("a", "w"), ("b", "q")]
print(noms_populaires(coms))   # ['a']
```

Nouveau coût : $O(n)$. Les noms sont dans l'ordre de première apparition (les dictionnaires conservent l'ordre d'insertion en Python).

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
`in` sur une liste dans une boucle	O(n²) caché	Convertir en ensemble
Concaténer des textes dans une boucle	Recopies répétées	`"".join(liste)`
Supposer O(1) partout	Mauvaise estimation	Apprendre le tableau des coûts
Conclure sans mesurer	Intuition trompeuse	Mesurer avec n puis 2n

Mini-boss

Mini-boss

Audit de performance. Voici cinq fragments de code. Pour chacun : coût actuel, coût possible, correction, test comparatif.

1. `if x in liste_d_exclus` dans une boucle de 100 000 lignes.
2. Construction d'un gros texte avec `texte += ligne` pour 50 000 lignes.
3. File d'attente implémentée avec `liste.pop(0)`.
4. Dédoublonnage avec `if x not in resultat: resultat.append(x)`.
5. Classement recalculé par `sorted(...)` après chaque nouveau score reçu.

Rends un tableau (problème · coût solution · gain mesuré).

Mini-boss · réponse

Pistes de correction

Corrections : (1) `set` ; (2) `"".join` ; (3) `deque` ; (4) un ensemble `vus` en parallèle ; (5) un tas ou une insertion triée avec `bisect.insort`. Pour chacun, une mesure avant/après sur n = 10 000 et 20 000 montre l'écart. Le tableau doit aussi dire pour quelle taille de données la correction n'est plus nécessaire.

Quiz éclair

Quiz éclair

1. Chercher une valeur dans un dictionnaire coûte en moyenne :

A. O(n)
B. O(n^2)
C. O(1)

2. `liste.pop(0)` coûte :

A. O(n)
B. O(1)
C. O(log n)

3. Le coût amorti d'un `append` en Python est :

A. O(n)

Quiz éclair

Quiz éclair (suite)

B. O(1)

C. O(n^2)

4. 🔁 *Révision mission 03* : Un `while i < n: i *= 2` fait environ :

A. n tours

B. n^2 tours

C. $\log_2$ n tours

5. 🔁 *Révision mission 02* : O(n^2) signifie que si n est multiplié par 10, le travail est multiplié par :

A. environ 100

B. 10

C. 1 000

Quiz éclair · réponse

Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

Je réfléchis

1. Quelle structure avais-tu tendance à utiliser « par défaut » ? Est-ce toujours le bon choix ?
2. Quel est le coût caché le plus fréquent selon toi dans le code des débutants ?

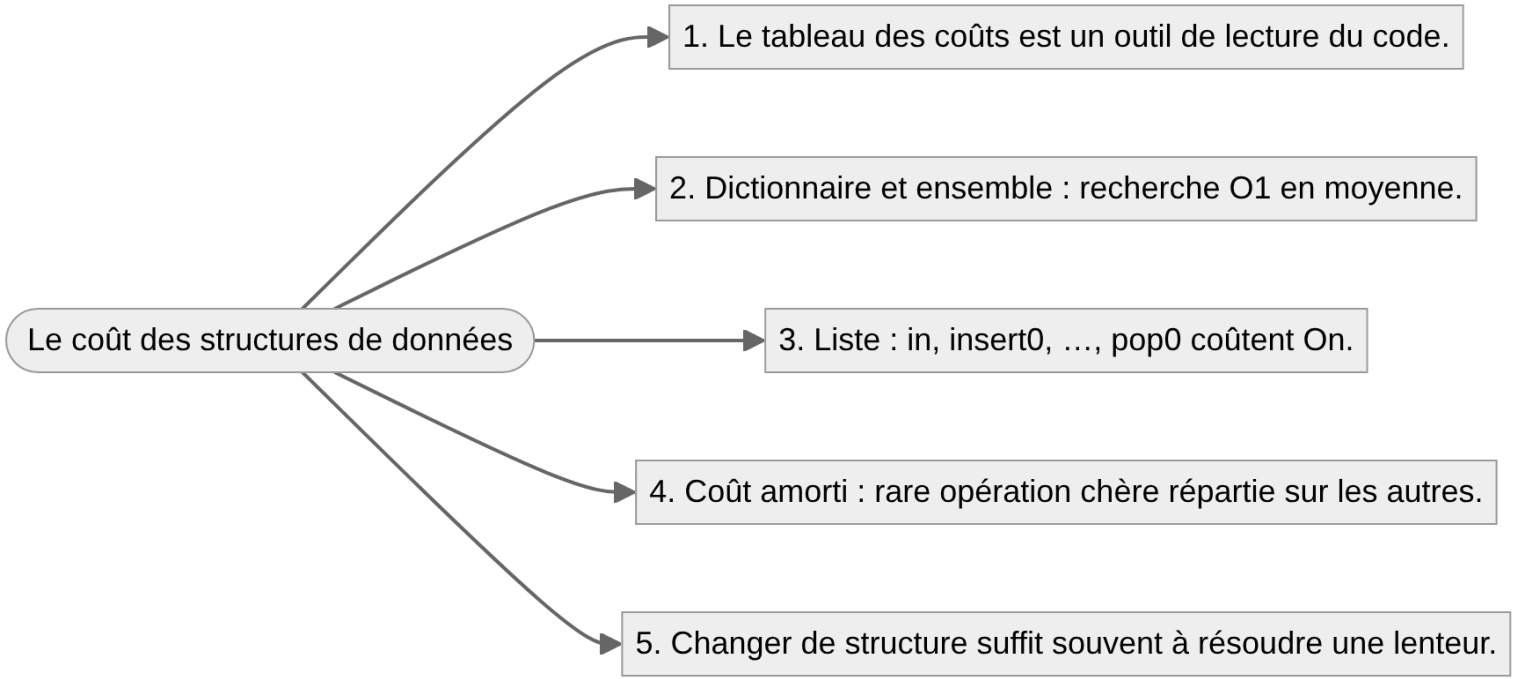
À retenir

À retenir

1. Le tableau des coûts est un outil de lecture du code.
2. Dictionnaire et ensemble : recherche O(1) en moyenne.
3. Liste : `in`, `insert(0, …)`, `pop(0)` coûtent O(n).
4. Coût amorti : rare opération chère répartie sur les autres.
5. Changer de structure suffit souvent à résoudre une lenteur.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Connaisseur des coûts

Prochaine mission : Recherche et tri : le grand écart

