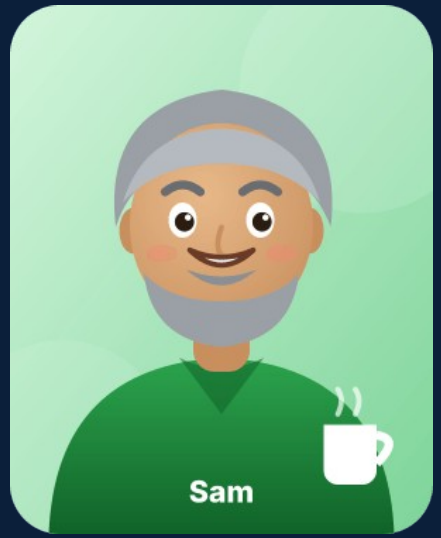
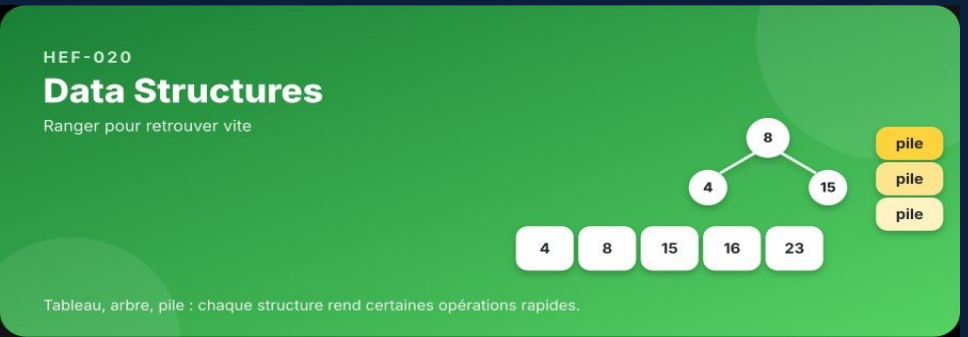
HEF-020 · MISSION 07 / 10

Tas et files de priorité

Quand ce n'est pas le premier arrivé mais le plus urgent qui doit passer d'abord.





Badge : Chef de la priorité

70 min · débutant

L'histoire

Le support de HashBox reçoit des tickets. Certains sont critiques (« le site est en panne »), d'autres mineurs (« une faute de frappe »). Une file classique les traiterait dans l'ordre d'arrivée, ce qui serait dangereux.

« Tu as besoin d'une file de priorité, dit Sam. Et la meilleure façon de la fabriquer, c'est un tas. »



Tu découvres comment obtenir toujours le plus important élément, très vite.

Ta mission

À la fin, je serai capable de…

1. expliquer ce qu'est une file de priorité
2. décrire un tas (heap) et son invariant
3. utiliser heapq en Python
4. trouver les k plus grands éléments avec un tas
5. choisir entre tri complet et tas

Défi d'échauffement

Défi d'échauffement

Aux urgences, les patients sont soignés selon la gravité, pas selon l'ordre d'arrivée. Si 100 patients arrivent au fil de la journée, comment maintiens-tu en permanence la liste de qui soigner ensuite ? Trier à chaque arrivée te paraît-il efficace ?

Le concept

File de priorité

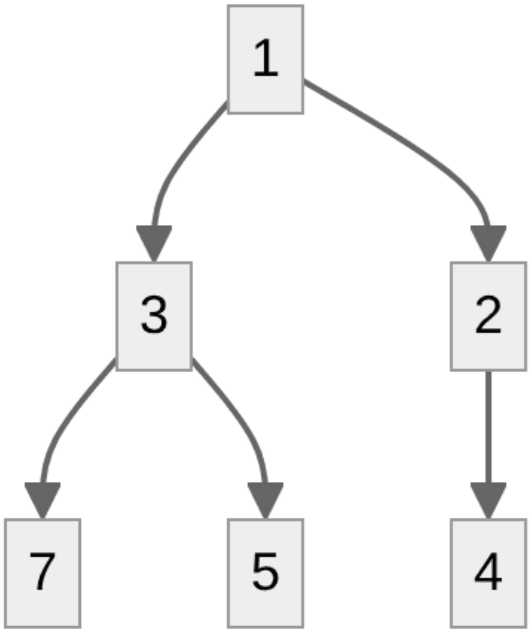
Une **file de priorité** (*priority queue*) permet d'**ajouter** un élément avec une priorité et de **retirer toujours celui de plus haute priorité**.

Le tas (heap)

Un **tas-min** est un arbre binaire où **chaque parent est plus petit (ou égal) que ses enfants**. Le plus petit élément est donc toujours à la **racine**.

Le concept

Le tas (heap) (suite)



Le concept

Le tas (heap) (suite)

Description du schéma : un tas-min. La racine (1) est inférieure à ses enfants (3 et 2). Chaque nœud est plus petit que ses enfants : 3 < 7 et 5, 2 < 4.

L'invariant rend le minimum accessible tout de suite. À l'ajout et au retrait, on **répare** localement le tas en échangeant des éléments le long d'un chemin, ce qui coûte de l'ordre de la hauteur de l'arbre, soit environ $\log_2(n)$.

On stocke le tas dans un **tableau** : les enfants de l'élément `i` sont aux positions `2i+1` et `2i+2`.

`heapq` en Python

```
import heapq

tas = []
heapq.heappush(tas, 5)
heapq.heappush(tas, 1)
```

Le concept

`heapq` en Python (suite)

```
heapq.heappush(tas, 3)
print(heapq.heappop(tas))   # 1 (le plus petit)
print(tas[0])               # 3 (le prochain)
```

Pour des priorités : on pousse des tuples `(priorité, élément)`. Le plus petit nombre sort en premier ; pour avoir « le plus urgent d'abord », utilise 1 pour le plus urgent.

Les k meilleurs éléments

Pour obtenir les 3 plus grands parmi 1 million, trier tout coûte cher. Un tas de taille 3 suffit : `heapq.nlargest(3, donnees)`.

🧠 ***Astuce*** *: tas = « je veux seulement le plus petit (ou le plus grand), tout le temps, pendant que des éléments arrivent ».*

Exemple pas à pas

Exemple pas à pas

Problème : traiter des tickets par urgence, `1` = critique.

```
import heapq

tickets = []
heapq.heappush(tickets, (3, "faute de frappe"))
heapq.heappush(tickets, (1, "site en panne"))
heapq.heappush(tickets, (2, "paiement refusé"))

while tickets:
    urgence, texte = heapq.heappop(tickets)
    print(urgence, texte)
# 1 site en panne
# 2 paiement refusé
# 3 faute de frappe
```

Exemple pas à pas (suite)

Si deux tickets ont la même urgence ? Python compare alors le texte, ce qui marche pour des textes mais pas pour des objets quelconques. Astuce : ajouter un compteur d'arrivée `(urgence, numero, ticket)` pour garantir un ordre stable et éviter la comparaison des objets.

```
import heapq, itertools

compteur = itertools.count()
tas = []
for urgence, texte in [(2, "b"), (1, "a"), (2, "c")]:
    heapq.heappush(tas, (urgence, next(compteur), texte))
print([heapq.heappop(tas)[2] for _ in range(3)])   # ['a', 'b', 'c']
```

À toi, avec aide

Un serveur reçoit `[7, 2, 9, 4, 1, 8, 3]`. Utilise un tas pour afficher les **3 plus petits** nombres dans l'ordre sans trier toute la liste.
Puis donne la liste après avoir fait seulement `heapify` : que contient `tas[0]` ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

`heapq.heapify(liste)` transforme une liste en tas sur place.

Indice 2

`heapq.heappop` retire le plus petit.

Indice 3

Le premier élément du tableau d'un tas-min est le minimum.

À toi, avec aide · réponse

Réponse

```
import heapq

donnees = [7, 2, 9, 4, 1, 8, 3]
tas = list(donnees)
heapq.heapify(tas)
print(tas[0])                                   # 1
print([heapq.heappop(tas) for _ in range(3)])   # [1, 2, 3]
print(heapq.nsmallest(3, donnees))              # [1, 2, 3]
```

À toi, en solo

Écris `top_k(flux, k)` qui renvoie les `k` plus grands éléments d'une suite de nombres **en ne gardant jamais plus de k éléments en mémoire** (le flux peut être très long).

Idée : garde un tas-min de taille `k` ; si un nouvel élément est plus grand que le minimum du tas, remplace-le. Teste : `top_k([5, 1, 9, 3, 7, 8, 2], 3)` → `[7, 8, 9]` (dans l'ordre croissant).

À toi, en solo · réponse

Pistes de correction

```
import heapq

def top_k(flux, k):
    tas = []
    for x in flux:
        if len(tas) < k:
            heapq.heappush(tas, x)
        elif x > tas[0]:
            heapq.heapreplace(tas, x)
    return sorted(tas)

print(top_k([5, 1, 9, 3, 7, 8, 2], 3))   # [7, 8, 9]
print(top_k(range(1000000), 3))          # [999997, 999998, 999999]
```

Mémoire : k éléments. Chaque élément coûte au plus de l'ordre de log k opérations.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Croire que le tas est entièrement trié	Seule la racine est garantie	Ne lire que `tas[0]` et utiliser `heappop`
Tas-min vs tas-max	`heapq` est un tas-min	Pour un max, pousser les valeurs négatives
Comparer des objets sans ordre	`TypeError` en cas d'égalité de priorité	Ajouter un compteur dans le tuple
Modifier un élément déjà dans le tas	L'invariant se casse	Retirer puis réinsérer

Mini-boss

Mini-boss

Construis un **planificateur de tâches** pour HashBox.

1. `Planificateur` avec `ajouter(nom, priorite)` et `suivante()` (la plus urgente, et à priorité égale la plus ancienne) ;
2. `changer_priorite(nom, p)` : approche « paresseuse » (on ajoute une nouvelle entrée et on ignore l'ancienne à la sortie) ;
3. Teste : 5 tâches, une qui change de priorité, une qui est annulée.

Mini-boss · réponse

Pistes de correction

```
import heapq, itertools

class Planificateur:
    def __init__(self):
        self._tas = []
        self._actuel = {}                    # nom -> (priorite, numero)
        self._compteur = itertools.count()

    def ajouter(self, nom, priorite):
        numero = next(self._compteur)
        self._actuel[nom] = (priorite, numero)
        heapq.heappush(self._tas, (priorite, numero, nom))

    changer_priorite = ajouter

    def annuler(self, nom):
        self._actuel.pop(nom, None)

    def suivante(self):
```

Mini-boss · réponse

Pistes de correction (suite)

```
        while self._tas:
            priorite, numero, nom = heapq.heappop(self._tas)
            if self._actuel.get(nom) == (priorite, numero):
                del self._actuel[nom]
                return nom
        return None

p = Planificateur()
p.ajouter("sauvegarde", 3); p.ajouter("panne", 1); p.ajouter("mail", 2)
p.changer_priorite("mail", 0); p.annuler("sauvegarde")
print(p.suivante(), p.suivante(), p.suivante())   # mail panne None
```

Quiz éclair

Quiz éclair

1. Dans un tas-min, la racine est :

A. le plus petit élément
B. le plus grand
C. au hasard

2. Une file de priorité retire toujours :

A. le premier arrivé
B. l'élément de plus haute priorité
C. le dernier arrivé

3. `heapq` en Python est un :

A. tas-max

Quiz éclair

Quiz éclair (suite)

B. arbre de recherche
C. tas-min

4. 🔁 *Révision mission 06* : Dans un BST, le sous-arbre de gauche contient :

A. des valeurs plus petites
B. des valeurs plus grandes
C. des valeurs au hasard

5. 🔁 *Révision mission 05* : Un ensemble contient :

A. des doublons
B. des éléments sans doublons
C. des couples clé-valeur

Quiz éclair · réponse

Réponses

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

Je réfléchis

Je réfléchis

1. Dans quelles situations de ta vie les priorités changent-elles en cours de route ?
2. Pourquoi trier toute la liste à chaque nouvelle arrivée serait-il un gaspillage ?

À retenir

1. File de priorité : toujours le plus important d'abord.
2. Tas : arbre où parent ≤ enfants, stocké dans un tableau.
3. `heapq` = tas-min en Python.
4. Un tas de taille k donne les k meilleurs sans tout trier.
5. Ajouter un compteur évite de comparer des objets.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Chef de la priorité

Prochaine mission : Graphes : modéliser les relations

