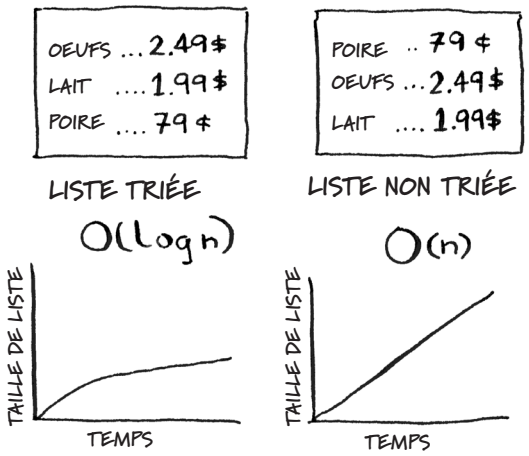
Tables de hachage | 5

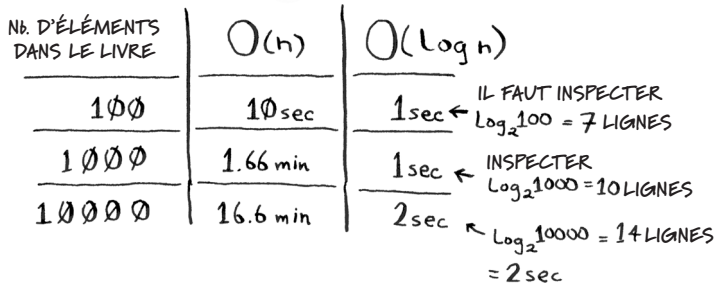
Dans ce chapitre

- Vous découvrirez les tables de hachage, l'une des structures de données de base les plus utiles. Les tables de hachage ont de nombreuses utilisations ; ce chapitre couvre les cas d'utilisation courants.
- Vous découvrirez les rouages des tables de hachage : implémentation, collisions et fonctions de hachage. Cela vous aidera à comprendre comment analyser les performances d'une table de hachage.

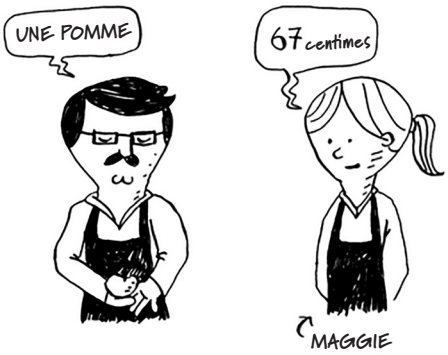
Supposons que vous travailliez dans une épicerie. Lorsqu'un client achète un produit, vous devez rechercher le prix dans un livre. Si le livre n'est pas classé alphabétiquement, cela peut vous prendre beaucoup de temps pour parcourir chaque ligne à la recherche d'une *pomme*. Vous devriez faire une recherche simple à partir du chapitre 1, et vous devriez regarder chaque ligne. Vous vous souvenez combien de temps cela prendrait ? Un temps O(n). Si le livre était classé par ordre alphabétique, vous pourriez lancer une recherche binaire pour trouver le prix d'une pomme. Cela ne prendrait qu'un temps O(log n).



Pour rappel, il y a une grosse différence entre les temps O(*n*) et O(log *n*) ! Supposons que vous puissiez parcourir 10 lignes du livre par seconde. Voici combien de temps la recherche simple et la recherche binaire vous prendraient.



Vous savez déjà que la recherche binaire est particulièrement rapide. Mais en tant que caissier, chercher des choses dans un livre est pénible, même si le livre est trié. Vous pouvez sentir le client s'impatienter pendant que vous recherchez des articles dans le livre. Ce dont vous avez vraiment besoin, c'est d'un acolyte qui a tous les noms et les prix mémorisés. Alors vous n'aurez pas besoin de chercher quoi que ce soit : vous lui demanderez et il vous donnera la réponse instantanément.



Votre amie Maggie peut vous donner le prix en temps O(1) pour n'importe quel article, quelle que soit la taille du livre. Elle est encore plus rapide que la recherche binaire.

Nb. d'éléments dans le livre	Recherche simple O(n)	Recherche binaire O(log n)	Maggie O(1)
100	10 sec	1 sec	INSTANTANÉ
1 000	1.6 min	1 sec	INSTANTANÉ
10 000	16.6 min	2 sec	INSTANTANÉ

Quelle personne merveilleuse ! Comment obtenez-vous une « Maggie » ?

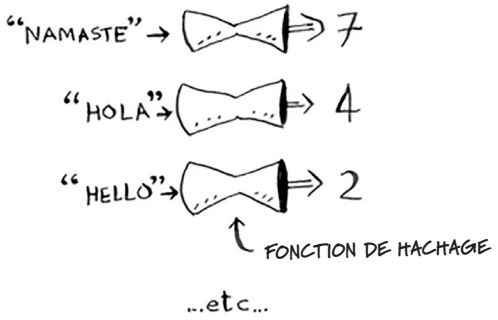
Mettons notre casquette de structure de données. Vous connaissez jusqu'à présent deux structures de données : les tableaux et les listes (je ne parlerai pas des piles car vous ne pouvez pas vraiment « rechercher » quelque chose dans une pile). Vous pourriez implémenter ce livre sous forme de tableau.

(oeufs, 2.49)	(lait, 1.49)	(poire, 0.79)

Chaque élément du tableau est en réalité composé de deux éléments : l'un est le nom d'un type de produit et l'autre est le prix. Si vous triez ce tableau par nom, vous pouvez exécuter une recherche binaire dessus pour trouver le prix d'un article. Ainsi, vous pouvez trouver des éléments en temps O(log *n*). Mais vous voulez trouver des éléments en temps O(1). C'est-à-dire que vous voulez créer une « Maggie ». C'est là qu'interviennent les fonctions de hachage.

Fonctions de hachage

Une fonction de hachage est une fonction dans laquelle vous insérez une chaîne (1) et vous récupérez un nombre.

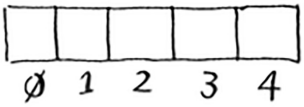


Dans la terminologie technique, nous dirions qu'une fonction de hachage « fait correspondre les chaînes à des nombres ». Vous pourriez penser qu'il n'y a pas de modèle discernable pour le nombre que vous obtenez lorsque vous mettez une chaîne. Mais il y a certaines exigences pour une fonction de hachage :

- Elle doit être cohérente. Par exemple, supposons que vous insériez « pomme » et récupériez « 4 ». Chaque fois que vous entrez « pomme », vous devriez récupérer « 4 ». Sans cela, votre table de hachage ne fonctionnera pas.
- Elle doit faire correspondre des nombres différents à des mots distincts. Par exemple, une fonction de hachage n'est pas bonne si elle renvoie toujours « 1 » pour tous les mots que vous saisissez. Dans le meilleur des cas, chaque mot distinct doit être associé à un nombre différent.

Ainsi, une fonction de hachage convertit les chaînes en nombres. À quoi ça sert ? Eh bien, vous pouvez l'utiliser pour faire votre « Maggie » !

Commencez avec un tableau vide :

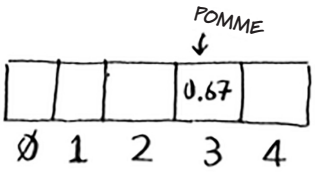


Vous stockerez tous vos prix dans ce tableau. Ajoutons le prix d'une pomme. Introduisez « pomme » dans la fonction de hachage.

---

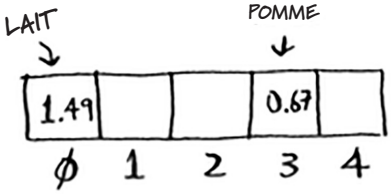
(1) ***Chaîne*** **signifie ici tout type de données — une séquence d'octets.**

La fonction de hachage renvoie « 3 ». Stockons donc le prix d'une pomme à l'indice 3 dans le tableau.

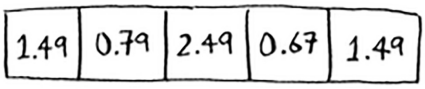


Ajoutons le lait. Introduisez le mot « lait » dans la fonction de hachage.

La fonction de hachage indique « 0 ». Stockons le prix du lait à l'indice 0.

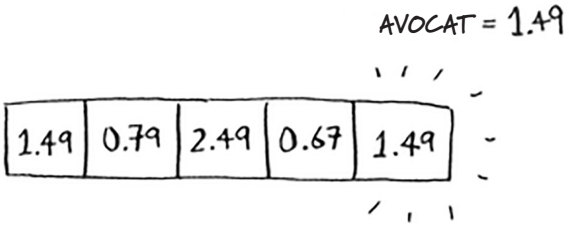


Continuez et finalement tout le tableau sera rempli de prix.



Maintenant, vous demandez : « Hé, quel est le prix d'un avocat ? » Vous n'avez pas besoin de le rechercher dans le tableau. Introduisez simplement « avocat » dans la fonction de hachage.

Elle vous indique que le prix est stocké à l'indice 4. Et bien sûr, c'est là qu'il se trouve.



La fonction de hachage vous indique exactement où le prix est stocké, vous n'avez donc pas à chercher du tout ! Cela fonctionne parce que :

- La fonction de hachage associe systématiquement un nom au même indice. Chaque fois que vous mettez «avocat», vous récupérez le même nombre. Vous pouvez donc l'utiliser la première fois pour trouver où stocker le prix d'un avocat, puis vous pouvez l'utiliser ensuite pour trouver où vous avez stocké ce prix.
- La fonction de hachage associe des indices différents à des chaînes distinctes. « Avocat » correspond à l'indice 4. « Lait » correspond à l'indice 0. Tout correspond à un emplacement différent dans le tableau où vous pouvez stocker les prix.
- La fonction de hachage connaît la taille de votre tableau et ne renvoie que des indices valides. Donc, si votre tableau contient 5 éléments, la fonction de hachage ne renvoie pas 100 ... ce ne serait pas un indice valide dans le tableau.

Vous venez de construire une « Maggie » ! Mettez une fonction de hachage et un tableau ensemble, et vous obtenez une structure de données appelée *table de hachage*. Une table de hachage est la première structure de données que vous apprendrez qui a une logique supplémentaire derrière elle. Les tableaux et les listes sont associés directement en mémoire mais les tables de hachage sont plus intelligentes. Elles utilisent une fonction de hachage pour déterminer intelligemment où stocker les éléments.

Les tables de hachage sont probablement la structure de données complexe la plus utile que vous apprendrez. Elles sont également connues sous le nom de cartes de hachage, cartes, dictionnaires et tableaux associatifs. Et les tables de hachage sont rapides ! Vous vous souvenez de notre discussion sur les tableaux et les listes chaînées au chapitre 2 ? Vous pouvez obtenir un élément d'un tableau instantanément. Et les tables de hachage utilisent un tableau pour stocker les données, elles sont donc tout aussi rapides.

Vous n'aurez probablement jamais à implémenter vous-même des tables de hachage. Tout bon langage informatique aura une implémentation pour les tables de hachage. Python a des tables de hachage ; on les appelle des *dictionnaires*. Vous pouvez créer une nouvelle table de hachage en utilisant la fonction `dict` :

```
>>> book = dict()
```

`book` est une nouvelle table de hachage. Ajoutons quelques prix à la variable `book` :

```
>>> book["apple"] = 0.67
>>> book["milk"] = 1.49
>>> book["avocado"] = 1.49
>>> print book
{'avocado': 1.49, 'apple': 0.67, 'milk': 1.49}
```

une pomme coûte 67 centimes.

le lait coûte $1.49.

Plutôt facile ! Demandons maintenant le prix d'un avocat :

```
>>> print book["avocado"]
1.49
```

Le prix d'un avocat

Une table de hachage a des clés et des valeurs. Dans la table de hachage `book`, les noms des produits sont les clés et leurs prix sont les valeurs. Une table de hachage associe les valeurs aux clés.

Dans la section suivante, vous verrez quelques exemples où les tables de hachage sont vraiment utiles.

EXERCICES

Il est important que les fonctions de hachage renvoient systématiquement la même sortie pour la même entrée. Si elles ne le font pas, vous ne pourrez pas trouver votre article après l'avoir mis dans la table de hachage !

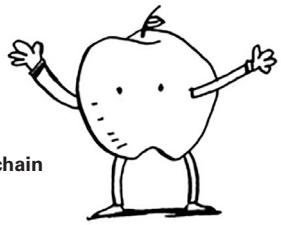
Lesquelles de ces fonctions de hachage sont cohérentes ?

5.1 `f(x) = 1` — **Renvoie « 1 » pour toutes les entrées**

5.2 `f(x) = rand()` — **Renvoie un nombre aléatoire à chaque fois**

5.3 `f(x) = next_empty_slot()` — **Renvoie l'indice du prochain emplacement**

5.4 `f(x) = len(x)` — **Utilise la longueur de la chaîne comme indice**



Cas d'usage

Les tables de hachage sont utilisées partout. Cette section va vous montrer quelques cas de leur utilisation.

Utilisation de tables de hachage pour les recherches

Votre téléphone dispose d'un répertoire pratique intégré. Chaque nom est associé à un numéro de téléphone.

BADE MAMA → 581 660 9820
ALEX MANNING → 484 234 4680
JANE MARIN → 415 567 3579

Supposons que vous vouliez créer un annuaire téléphonique comme celui-ci. Vous associez les noms des personnes à des numéros de téléphone. Votre répertoire téléphonique doit avoir cette fonctionnalité :

- Ajouter le nom d'une personne et le numéro de téléphone associé à cette personne.
- Saisir le nom d'une personne et obtenir le numéro de téléphone associé à ce nom.

C'est un cas d'utilisation parfait pour les tables de hachage !
Les tables de hachage sont l'outil idéal quand vous voulez

- créer une association d'une chose à une autre
- rechercher quelque chose

Construire un annuaire téléphonique est assez facile. Tout d'abord, créez une nouvelle table de hachage :

```
>>> phone_book = dict()
```

Soit dit en passant, Python a un raccourci pour créer une nouvelle table de hachage. Vous pouvez utiliser deux accolades :

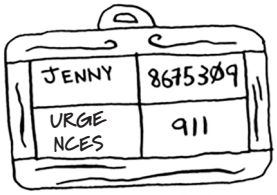
```
>>> phone_book = {}
```
Identique à phone_book = dict()

Ajoutons les numéros de téléphone de certaines personnes dans cet annuaire :

```
>>> phone_book[«jenny»] = 8675309
>>> phone_book[«urgences»] = 911
```

Il n'y a rien de plus ! Maintenant, supposons que vous vouliez trouver le numéro de téléphone de Jenny.
Il suffit de passer la clé au tableau de hachage :

```
>>> print phone_book[«jenny»]
8675309
```
Numéro de téléphone de Jenny



UNE TABLE DE HACHAGE POUR CRÉER UN RÉPERTOIRE TÉLÉPHONIQUE

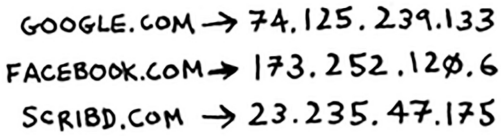
Imaginez si vous deviez le faire en utilisant un tableau à la place.

Comment feriez-vous ? Les tables de hachage facilitent la modélisation d'une relation d'un élément à un autre.

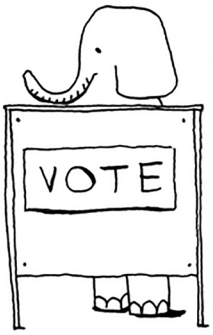
Les tables de hachage sont utilisées pour les recherches à une échelle beaucoup plus grande. Par exemple, supposons que vous vous rendiez sur un site Web comme http://adit.io. Votre ordinateur doit traduire adit.io en une adresse IP.

ADIT.IO → 173.255.248.55

Pour tout site Web sur lequel vous vous rendez, l'adresse Web doit être traduite en une adresse IP.

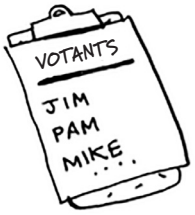


Wow, associer une adresse Web à une adresse IP ? Cela ressemble à un cas d'utilisation parfait pour les tables de hachage ! Ce processus est appelé *résolution DNS*. Les tables de hachage sont un moyen de fournir cette fonctionnalité.



Empêcher les doublons en entrée

Supposons que vous dirigiez un bureau de vote. Naturellement, chaque personne ne peut voter qu'une seule fois. Comment s'assurer qu'elle n'a pas déjà voté ? Quand quelqu'un vient voter, vous demandez son nom complet. Ensuite, vous le comparez à la liste des personnes qui ont voté.



Si son nom figure sur la liste, c'est que cette personne a déjà voté et doit être refoulée ! Sinon, vous ajoutez son nom à la liste et vous la laissez voter. Supposons maintenant que beaucoup de personnes soient venues voter et que la liste des personnes qui ont voté soit vraiment longue.

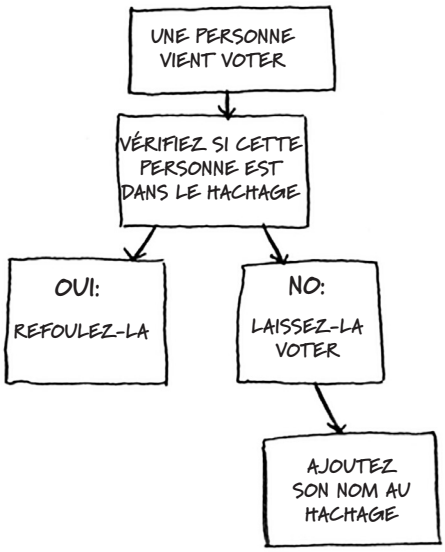
À chaque fois qu'un nouveau votant se présente, il faut scanner cette immense liste pour voir s'il a déjà voté. Mais il y a un meilleur moyen : utilisez un hachage ! Tout d'abord, faites un hachage pour garder une trace des personnes qui ont voté :

```
>>> voted = {}
```

Lorsqu'une nouvelle personne vient voter, vérifiez si elle est déjà dans le hachage :

```
>>> value = voted.get("tom")
```

La fonction `get` renvoie la valeur si « tom » est dans la table de hachage. Sinon, elle renvoie `None`. Vous pouvez l'utiliser pour vérifier si quelqu'un a déjà voté !



Voici le code :

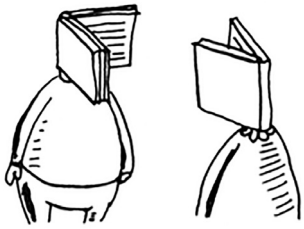
```
voted = {}

def check_voter(name):
if voted.get(name):
print "refoulez-le !"
else:
voted[name] = True
print "laissez-le voter !"
```

Testons-le plusieurs fois :

```
>>> check_voter("tom")
laissez-le voter !
>>> check_voter(« mike »)
laissez-le voter !
>>> check_voter(« mike »)
refoulez-le !
```

La première fois que Tom entrera, ceci affichera, «laissez-le voter !» Ensuite, Mike entre, et il s'affiche, « laissez-le voter ! » Ensuite, Mike essaie d'y aller une deuxième fois, et il s'affiche, « refoulez-le ! ».

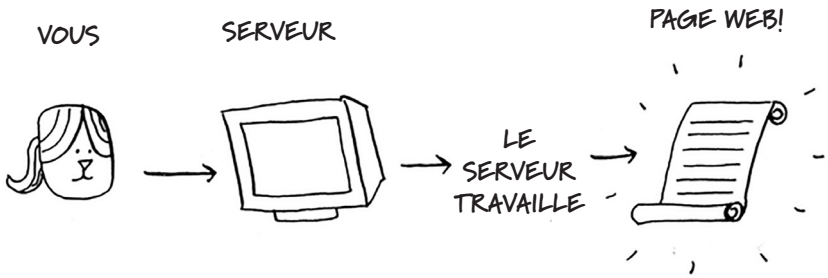
N'oubliez pas que si vous stockiez ces noms dans une liste de personnes ayant déjà voté, cette fonction finirait par devenir très lente, car elle devrait effectuer une recherche simple sur toute la liste. Mais vous stockez plutôt les noms dans une table de hachage, et une table de hachage vous indique instantanément si le nom de cette personne figure ou non dans la table. La détection de doublons est donc très rapide avec une table de hachage.



Utiliser des tables de hachage comme cache

Un dernier cas d'utilisation : la mise en cache. Si vous travaillez sur un site Web, vous avez peut-être déjà entendu parler de la mise en cache comme une bonne chose à faire. Voici l'idée. Supposons que vous visitiez facebook.com :

1. Vous faites une demande au serveur de Facebook.
2. Le serveur réfléchit une seconde et propose la page Web à vous envoyer.
3. Vous recevez une page Web.

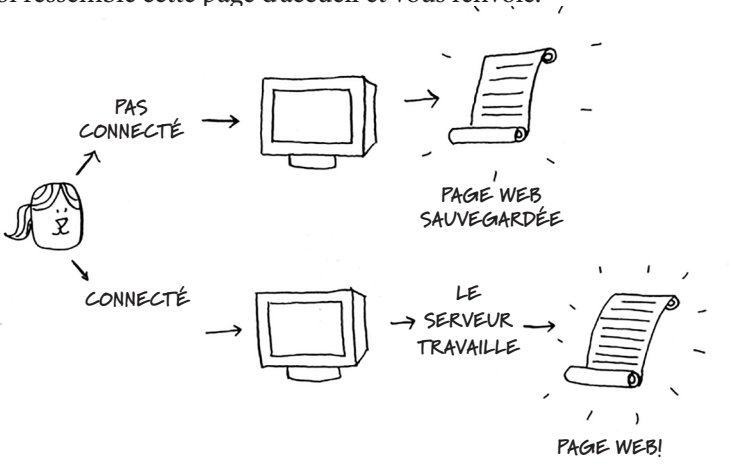


Par exemple, sur Facebook, le serveur peut collecter toutes les activités de vos amis pour vous les montrer. Il faut quelques secondes pour collecter toute cette activité et vous la présenter. Ces quelques secondes peuvent sembler longues en tant qu'utilisateur. Vous pourriez penser : « Pourquoi Facebook est-il si lent ? » D'un autre côté, les serveurs de Facebook doivent servir des millions de personnes, et ces quelques secondes s'additionnent pour chacun d'eux. Les serveurs de Facebook travaillent vraiment dur pour produire toutes ces pages Web. Existe-t-il un moyen de rendre Facebook plus rapide et de réduire la charge de travail de ses serveurs en même temps ?

Supposons que vous ayez une nièce qui n'arrête pas de vous poser des questions sur les planètes. « À quelle distance se trouve Mars de la Terre ? » « À quelle distance est la Lune ? » « À quelle distance est Jupiter ? » À chaque fois, il faut faire une recherche sur Google et

lui donner une réponse. Cela prend quelques minutes. Maintenant, supposons qu'elle demande toujours : « À quelle distance est la Lune ? » Très vite, vous vous souviendrez que la Lune est à 384 400 kilomètres. Vous n'auriez pas à chercher sur Google ... vous vous souviendriez simplement et répondriez. C'est ainsi que fonctionne la mise en cache : les sites Web mémorisent les données au lieu de les recalculer.

Si vous êtes connecté à Facebook, tout le contenu que vous voyez est conçu spécialement pour vous. Chaque fois que vous allez sur facebook.com, ses serveurs doivent réfléchir au contenu qui vous intéresse. Mais si vous n'êtes pas connecté à Facebook, vous voyez la page de connexion. Tout le monde voit la même page de connexion. On demande sans cesse la même chose à Facebook : « Donnez-moi la page d'accueil lorsque je suis déconnecté ». Il cesse donc de faire fonctionner le serveur pour décider à quoi ressemble la page d'accueil. Au lieu de cela, il mémorise à quoi ressemble cette page d'accueil et vous l'envoie.

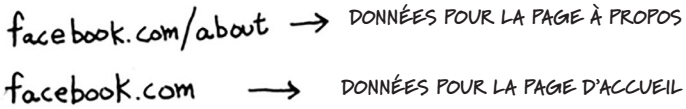


C'est ce qu'on appelle la *mise en cache*. Cela a deux avantages :

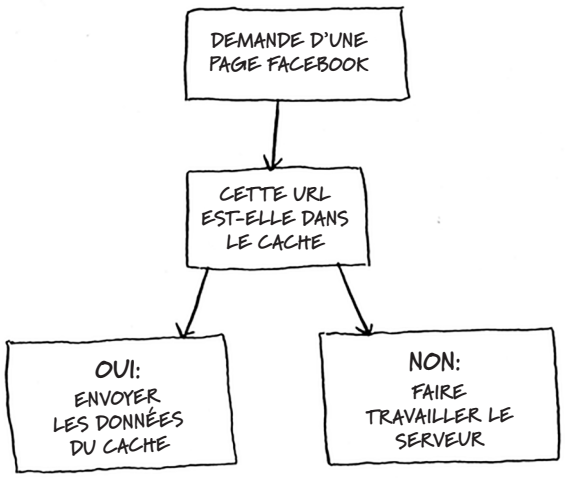
- Vous obtenez la page Web beaucoup plus rapidement, tout comme lorsque vous mémorisez la distance de la Terre à la Lune. La prochaine fois que votre nièce vous le demandera, vous n'aurez pas à rechercher sur Google. Vous pourrez répondre instantanément.
- Facebook doit faire moins de travail.

La mise en cache est un moyen courant d'accélérer les choses. Tous les grands sites Web utilisent la mise en cache. Et ces données sont mises en cache dans un hachage !

Facebook ne se contente pas de mettre en cache la page d'accueil. Il met également en cache la page *À propos*, la page *Contact*, la page *Termes et conditions*, et bien plus encore. Il a donc besoin d'une correspondance entre l'URL de la page et les données de la page.



Lorsque vous visitez une page sur Facebook, il vérifie d'abord si la page est stockée dans la table de hachage.



En voici le code :

```
cache = {}

def get _ page(url):
  if cache.get(url):
    return cache[url]
else:
    data = get _ data _ from _ server(url)
    cache[url] = data
    return data
```

Renvoie les données mises en cache

Enregistre ces données dans votre cache d'abord

Ici, vous faites fonctionner le serveur uniquement si l'URL n'est pas dans le cache. Avant de renvoyer les données, cependant, vous les enregistrez dans le cache. La prochaine fois que quelqu'un demandera cette URL, vous pourrez envoyer les données du cache au lieu de demander au serveur de faire le travail.

À retenir

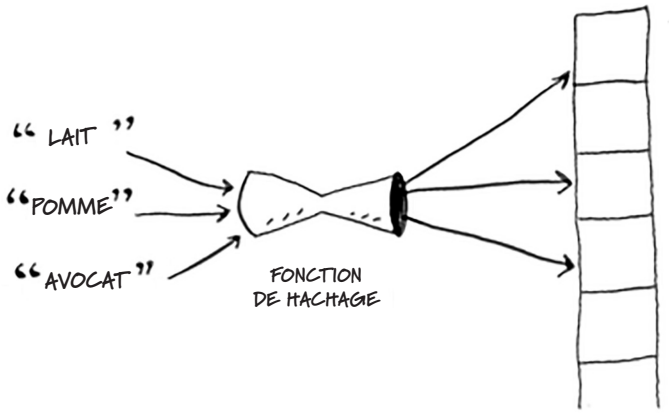
Pour récapituler, les hachages sont bons pour

- modéliser des relations d'une chose à une autre
- détecter des doublons
- mettre en cache/mémoriser les données au lieu de faire travailler votre serveur

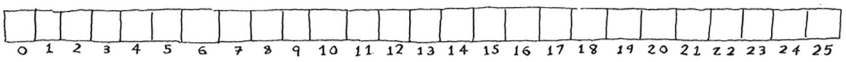
Collisions

Comme je l'ai expliqué plus tôt, la plupart des langages possèdent des tables de hachage. Vous n'avez pas à savoir écrire les vôtres. Donc, je ne parlerai pas trop des détails des tables de hachage. Mais vous vous souciez néanmoins de la performance ! Pour comprendre les performances des tables de hachage, vous devez d'abord comprendre ce que sont les collisions. Les deux sections suivantes traitent des collisions et des performances.

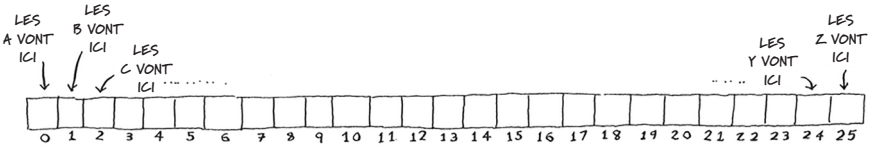
Tout d'abord, je vous ai dit un mensonge. Je vous ai dit qu'une fonction de hachage associe toujours des emplacements différents du tableau à des clés distinctes.

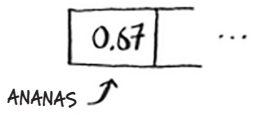


En réalité, il est presque impossible d'écrire une fonction de hachage qui fasse cela. Prenons un exemple simple. Supposons que votre tableau contienne 26 emplacements.



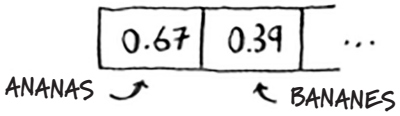
Et votre fonction de hachage est vraiment simple : elle attribue un emplacement dans le tableau de manière alphabétique.



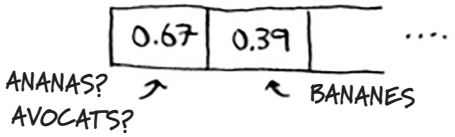


Peut-être que vous entrevoyez déjà le problème. Vous voulez inscrire le prix des ananas dans votre hachage. Le premier emplacement vous est attribué.

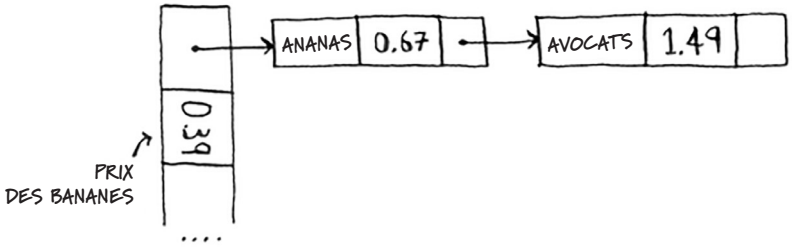
Ensuite, vous voulez mettre le prix des bananes dans le hachage. Vous obtenez le deuxième emplacement.



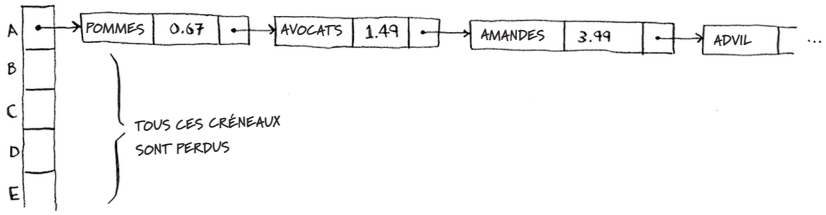
Jusque-là tout va bien ! Mais maintenant, vous voulez mettre le prix des avocats dans votre hachage. Vous obtenez à nouveau le premier emplacement.



Hélas ! Les ananas occupent déjà cet emplacement ! Que faire ? C'est ce qu'on appelle une *collision* : deux clés ont été affectées au même emplacement. C'est un problème. Si vous stockez le prix des avocats à cet emplacement, vous écraserez le prix des ananas. Ensuite, la prochaine fois que quelqu'un demandera le prix des ananas, il obtiendra le prix des avocats à la place ! Les collisions sont mauvaises et vous devez les éviter. Il existe de nombreuses façons de gérer les collisions. Le plus simple est de faire comme ceci : si plusieurs clés correspondent au même emplacement, démarrez une liste chaînée à cet emplacement.



Dans cet exemple, « ananas » et « avocat » correspondent au même emplacement. Vous commencez donc une liste chaînée à cet emplacement. Si vous avez besoin de connaître le prix des bananes, cela reste rapide. Si vous avez besoin de connaître le prix des ananas, c'est un peu plus lent. Vous devez rechercher dans cette liste chaînée pour trouver «ananas». Si la liste chaînée est petite, ce n'est pas grave : vous devrez rechercher parmi trois ou quatre éléments. Mais supposons que vous travailliez dans une épicerie où vous ne vendez que des produits commençant par la lettre A.



Minute ! La table de hachage entière est totalement vide à l'exception d'un emplacement. Et cet emplacement a une liste chaînée géante ! Chaque élément de cette table de hachage se trouve dans la liste chaînée. C'est aussi mauvais que de tout mettre dans une liste chaînée dès le départ. Cela va ralentir votre table de hachage.

Il y a deux leçons ici :

- *La fonction de hachage est vraiment importante.* Votre fonction de hachage envoyait toutes les clés sur un unique emplacement. Idéalement, votre fonction de hachage devrait associer les clés de manière uniforme sur tout le hachage.
- Si ces listes chaînées deviennent longues, cela ralentit considérablement votre table de hachage. Mais elles ne seront pas longues si vous utilisez *une bonne fonction de hachage* !

Les fonctions de hachage sont importantes. Une bonne fonction de hachage créera très peu de collisions. Alors, comment choisir une bonne fonction de hachage ? Nous allons le voir dans la section suivante !

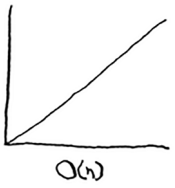
Performance

Vous avez commencé ce chapitre à l'épicerie. Vous vouliez construire quelque chose qui vous donnerait instantanément les prix des produits. Effectivement, les tables de hachage sont vraiment rapides.

	CAS MOYEN	PIRE DES CAS
RECHERCHER	O(1)	O(n)
INSÉRER	O(1)	O(n)
SUPPRIMER	O(1)	O(n)

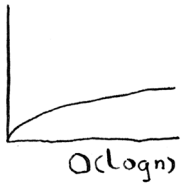
PERFORMANCE DES TABLES DE HACHAGE

Dans le cas moyen, les tables de hachage prennent un temps O(1) pour tout. O(1) est appelé *temps constant.* Vous n'avez encore jamais vu le temps constant. Cela ne veut pas dire instantané. Cela signifie que le temps nécessaire restera le même, quelle que soit la taille de la table de hachage. Par exemple, vous savez qu'une recherche simple prend un temps linéaire.



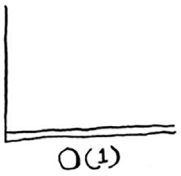
TEMPS LINÉAIRE
(RECHERCHE SIMPLE)

La recherche binaire est plus rapide - elle prend un temps logarithmique :



TEMPS LOGARITHMIQUE
(RECHERCHE BINAIRE)

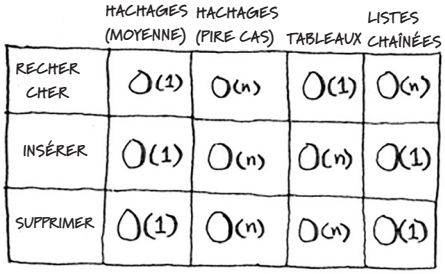
Rechercher quelque chose dans une table de hachage prend un temps constant.



TEMPS CONSTANT
(TABLE DE HACHAGE)

Vous voyez comment c'est représenté par une ligne horizontale ? Cela signifie que peu importe que votre table de hachage contienne 1 élément ou 1 milliard d'éléments — extraire quelque chose de cette table de hachage prendra le même temps. En fait, vous avez déjà vu le temps constant. Sortir un élément d'un tableau prend un temps constant. La taille de votre tableau n'a pas d'importance ; il faut le même temps pour obtenir un élément. Dans le cas moyen, les tables de hachage sont vraiment rapides.

Dans le pire des cas, une table de hachage prend O(n) — temps linéaire — pour tout, ce qui est vraiment lent. Comparons les tables de hachage aux tableaux et aux listes.



Regardez le cas moyen pour les tables de hachage. Les tables de hachage sont aussi rapides que les tableaux pour la recherche (obtenir une valeur à un certain indice). Et elles sont aussi rapides que les listes chaînées pour les insertions et les suppressions. C'est le meilleur des deux mondes ! Mais dans le pire des cas, les tables de hachage sont toutes lentes. Il est donc important que vous n'atteigniez pas les pires performances avec les tables de hachage. Et pour ce faire, vous devez éviter les collisions. Pour éviter les collisions, vous avez besoin

- d'un faible facteur de charge
- d'une bonne fonction de hachage

Note

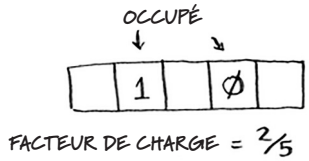
Avant de commencer la section suivante, sachez que cette lecture n'est pas obligatoire. Je vais parler de la façon d'implémenter une table de hachage, mais vous n'aurez jamais à le faire vous-même. Quel que soit le langage de programmation que vous utilisez, une implémentation de tables de hachage sera intégrée. Vous pouvez utiliser la table de hachage intégrée et supposer qu'elle aura de bonnes performances. La section suivante vous donne un aperçu de ce qui se passe sous le capot.

Facteur de charge

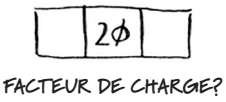
Le facteur de charge d'une table de hachage est facile à calculer.

$$\frac{\text{NOMBRE D'ÉLÉMENTS DANS LA TABLE DE HACHAGE}}{\text{NOMBRE TOTAL D'EMPLACEMENTS}}$$

Les tables de hachage utilisent un tableau pour le stockage, vous comptez donc le nombre d'emplacements occupés dans un tableau. Par exemple, la table de hachage suivante a un facteur de charge de $^2/_5$, ou 0,4.

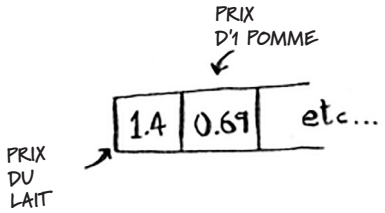


Quel est le facteur de charge de cette table de hachage ?



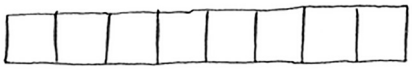
Si vous avez répondu 1/3, vous avez raison. Le facteur de charge mesure le nombre d'emplacements vides restants dans votre table de hachage.

Supposons que vous ayez besoin de stocker le prix de 100 produits dans votre table de hachage et que votre table de hachage ait 100 emplacements. Dans le meilleur des cas, chaque élément aura son propre emplacement.

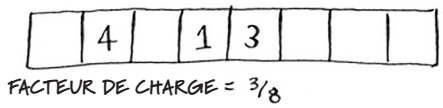


Cette table de hachage a un facteur de charge de 1. Et si votre table de hachage n'a que 50 emplacements ? En ce cas, elle a un facteur de charge de 2. Il n'y a aucun moyen que chaque élément ait son propre emplacement, car il n'y a pas assez d'emplacements ! Avoir un facteur de charge supérieur à 1 signifie que vous avez plus d'éléments que d'emplacements dans votre table. Lorsque le facteur de charge commence à augmenter, vous devez ajouter plus d'emplacements à votre table de hachage. C'est ce qu'on appelle le redimensionnement. Par exemple, supposons que vous ayez cette table de hachage qui devient assez pleine.

Vous devez redimensionner cette table de hachage. D'abord, vous créez un nouveau tableau qui est plus grand. La règle d'or consiste à créer un tableau deux fois plus grand.



Vous devez maintenant réinsérer tous les éléments dans cette nouvelle table de hachage en utilisant la fonction de hachage :

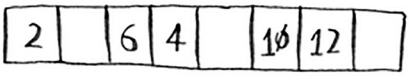


Cette nouvelle table a un facteur de charge de $^3/_8$. Beaucoup mieux ! Avec un facteur de charge inférieur, vous aurez moins de collisions et votre table fonctionnera mieux. Une bonne règle empirique est de redimensionner lorsque votre facteur de charge devient supérieur à 0,7.

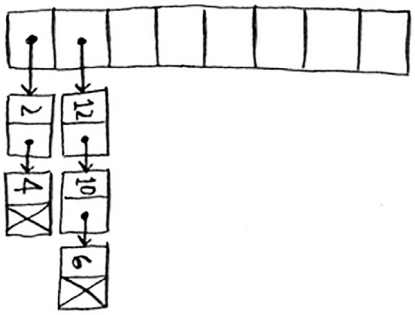
Vous pensez peut-être : « Cette entreprise de redimensionnement prend beaucoup de temps ! » Et c'est juste. Le redimensionnement est coûteux et vous ne voulez pas redimensionner trop souvent. Mais en moyenne, les tables de hachage prennent un temps O(1) même avec un redimensionnement.

Une bonne fonction de hachage

Une bonne fonction de hachage distribue uniformément les valeurs dans le tableau.



Une mauvaise fonction de hachage regroupe les valeurs et produit beaucoup de collisions.



Qu'est-ce qu'une bonne fonction de hachage ? C'est quelque chose dont vous n'aurez jamais à vous soucier : seuls des vieillards avec de grandes barbes, assis dans des pièces sombres, s'en inquiètent. Si vous êtes vraiment curieux, jetez un coup d'œil à la fonction SHA (il y en a une courte description dans le dernier chapitre). Vous pouvez l'utiliser comme fonction de hachage.

EXERCICES

Il est important que les fonctions de hachage aient une bonne distribution. Elles doivent répartir les éléments aussi largement que possible. Le pire des cas est une fonction de hachage qui associe tous les éléments au même emplacement dans la table.

Supposons que vous ayez ces quatre fonctions de hachage qui fonctionnent avec des chaînes :

A. Renvoyer « 1 » pour toutes les entrées.

B. Utiliser la longueur de la chaîne comme indice.

C. Utiliser le premier caractère de la chaîne comme indice. Ainsi, toutes les chaînes commençant par un 'a' sont hachées ensemble, et ainsi de suite.

D. Associer chaque lettre à un nombre premier : a = 2, b = 3, c = 5, d = 7, e = 11, et ainsi de suite. Pour une chaîne, la fonction de hachage est la somme de tous les caractères modulo de la taille du hachage. Par exemple, si votre taille de hachage est 10 et que la chaîne est « bag », l'indice est 3 + 2 + 17 % 10 = 22 % 10 = 2.

Pour chacun de ces exemples, quelles fonctions de hachage fourniraient une bonne distribution ? Supposons une taille de table de hachage de 10 emplacements.

5.5 Un répertoire où les touches sont des noms et les valeurs sont des numéros de téléphone. Les noms sont les suivants : Esther, Ben, Bob et Dan.

5.6 Une correspondance entre la taille d'une pile et sa puissance. Les tailles sont A, AA, AAA et AAAA.

5.7 Une correspondance entre des titres de livres et leurs auteurs. Les titres sont Maus, Fun Home et Watchmen.

À retenir

Vous n'aurez presque jamais à implémenter vous-même une table de hachage. Le langage de programmation que vous utilisez doit vous fournir une implémentation. Vous pouvez utiliser les tables de hachage de Python et supposer que vous obtiendrez les performances des cas moyens : temps constant.

Les tables de hachage sont une structure de données puissante car elles sont si rapides et elles vous permettent de modéliser les données d'une

manière différente. Vous découvrirez peut-être bientôt qu'on les utilise tout le temps.

- Vous pouvez créer une table de hachage en combinant une fonction de hachage avec un tableau.
- Les collisions sont mauvaises. Vous avez besoin d'une fonction de hachage qui minimise les collisions.
- Les tables de hachage ont des fonctions de recherche, d'insertion et de suppression très rapides.
- Les tables de hachage sont utiles pour modéliser les relations d'un élément à un autre.
- Une fois que votre facteur de charge est supérieur à 0,7, il est temps de redimensionner votre table de hachage.
- Les tables de hachage sont utilisées pour la mise en cache des données (par exemple, avec un serveur Web).
- Les tables de hachage sont idéales pour détecter les doublons.