Informatique

2
Dictionnaires et programmation dynamique

TD2-2

Hachage

Sujet inspiré du travail de Bernard SALAMITO
et d'un TD de licence d'informatique de l'université de Nice

Exercice 1: Hachage

Fonction hash

Les dictionnaires sont construits en utilisant un principe de hachage des clés. Ce principe permet de rendre l'accès aux valeurs d'un dictionnaire très rapide. Si le dictionnaire contient n ensembles de clés/valeurs, l'accès à l'un d'eux présente une complexité en moyenne en $O(1)$ et en $O(n)$ dans le pire des cas.

Le hachage est réalisé via une fonction que nous ne développerons pas, mais qui peut être utilisée sous Python avec la commande `hash()`.

Les objets que nous utilisons le plus sont les entiers, flottants, booléens, tuples, chaines de caractères, listes, arrays et dictionnaires

Question 1: Trouver les types hachables sous python parmi les types proposés ci-dessus

La majorité des objets hachables sont les objets non mutables, c'est-à-dire les objets qui ne peuvent pas être modifiés, dont on ne peut pas changer les propriétés une fois qu'ils ont été définis.

Soit les lignes de commande suivantes :

```
import sys
sys.hash_info
M = sys.hash_info.modulus
print(M)
```

Question 2: Comparer M au nombre de valeurs différentes représentables dans le système que vous utilisez (probablement 64 bits)

Les règles de hachage implémentées en Python sont disponibles ici (LIEN) et en voici une capture :

Nous n'irons pas plus loin ici. La suite de ce TD vous fera comprendre l'intérêt du hachage.

CPython implementation detail: Actuellement, le premier utilisé est `P = 2 ** 31 - 1` sur des machines dont les *longs* en C sont de 32 bits `P = 2 ** 61 - 1` sur des machines dont les *longs* en C font 64 bits.

Voici les règles en détail :

- Si `x = m / n` est un nombre rationnel non négatif et `n` n'est pas divisible par `P`, définir `hash(x)` comme `m * invmod(n, P) % P`, où `invmod(n, P)` donne l'inverse de `n` modulo `P`.
- Si `x = m / n` est un nombre rationnel non négatif et `n` est divisible par `P` (mais `m` ne l'est pas), alors `n` n'a pas de modulo inverse `P` et la règle ci-dessus n'est pas applicable ; dans ce cas définir `hash(x)` comme étant la valeur de la constante `sys.hash_info.inf`.
- Si `x = m / n` est un nombre rationnel négatif définir `hash(x)` comme `-hash(-x)`. Si le résultat est `-1`, le remplacer par `-2`.
- Les valeurs particulières `sys.hash_info.inf`, `-sys.hash_info.inf` et `sys.hash_info.nan` sont utilisées comme valeurs de hachage pour l'infini positif, l'infini négatif, ou *nans* (respectivement). (Tous les *nans* hachables ont la même valeur de hachage.)
- Pour un nombre `complexe z`, les valeurs de hachage des parties réelles et imaginaires sont combinées en calculant `hash(z.real) + sys.hash_info.imag * hash(z.imag)`, réduit au modulo `2**sys.hash_info.width` de sorte qu'il se trouve dans `range(-2**(sys.hash_info.width - 1), 2**(sys.hash_info.width - 1))`. Encore une fois, si le résultat est `-1`, il est remplacé par `-2`.

Sujet inspiré du travail de Bernard SALAMITO
et d'un TD de licence d'informatique de l'université de Nice

Hachage d'entiers

Soient des entiers et la fonction de hachage h qui à tout entier n, lui associe son reste dans la division euclidienne par k.

Question 3: Créer la fonction h(n,k)

On souhaite construire un dictionnaire stocké via deux tables, une table de hachage (ou table des indices) et une table de débordement, en utilisant la fonction de hachage h pour k=10 et des clés entières.

En voici un exemple :

Table de hachage

Valeur de hachage ih=h(cle)	Indice débordement id
0	0
1	0
2	1
3	3
4	0
5	4
6	0
7	0
8	0
9	0

Table de débordement

id	Clé	Valeur	Suivant ids
0	None		6
1	12	douze	2
2	32	trente-deux	5
3	43	quarante-trois	6
4	55	cinquante-cinq	0
5	42	quarante-deux	0
6	53	cinquante-trois	0

La clé est hachée afin d'en déterminer l'indice ih dans la table de hachage. Si l'indice de débordement id associé à ih est nul, la clé n'existe pas dans le dictionnaire. Sinon, id indique l'indice dans la table de débordement de la première instance du dictionnaire dont la clé possède la valeur de hachage ih.

La table de débordement contient [None,0] si elle est vide. Le second terme de cette première liste indique l'indice du dernier élément du dictionnaire dans TD. Ensuite, pour chacun de ses éléments, on trouve la clé, la valeur associée, et l'indice suivant ids de la prochaine valeur dans cette table ayant la même valeur de hachage. Si cet indice ids est nul, il n'y a plus d'autres éléments ayant la même valeur de hachage.

On initialise les tables ainsi :

```
k = 10
TH = [0]*k
TD = [[None,0]]
```

Remarque : il n'est effectivement pas utile de créer la colonne ih dans la table de hachage, on ne créera pas non plus l'indice id dans la table de débordement.

Sujet inspiré du travail de Bernard SALAMITO
et d'un TD de licence d'informatique de l'université de Nice

Question 4: Compléter le dictionnaire ci-dessus à la main en imaginant appeler la fonction insere pour (22,'vingt-deux') et (24,'vingt-quatre')

ATTENTION : pour la suite, vous devez travailler dans le contexte de la gestion informatique des dictionnaires. Autrement dit, utiliser val in TD, len(TH), len(TD) par exemple, coûte O(n) avec n le nombre de clés/valeurs, ce qui est HORRIBLE. Le dictionnaire a pour but d'aller chercher de proche en proche, sur un nombre très restreint de valeurs dans la table de débordement.

Question 5: Créer une fonction insere(cle,valeur) permettant d'insérer un élément dans le dictionnaire en mettant à jour les deux tables. On veillera à écraser une valeur si la clé associée est déjà présente
Question 6: En utilisant la fonction insere, créer les tables proposées ci-dessus

Vérifier :

```
>>> TH
[0, 0, 1, 3, 0, 4, 0, 0, 0, 0]

>>> TD
[[None, 6], [12, 'douze', 2], [32, 'trente deux', 5],
[43, 'quarante trois', 6], [55, 'cinquante cinq', 0],
[42, 'quarante deux', 0], [53, 'cinquante trois', 0]]
```

Remarque : Vous essaierez de remplacer par deux fois (pour revenir à l'état souhaité du dictionnaire) une valeur associée à une clé déjà existante. Exemple : insere(32,'Felipe') puis insere(32,'trente-deux')

Question 7: Créer la fonction recherche(cle) renvoyant le booléen True ou False indiquant si la clé est dans le dictionnaire
Question 8: En supposant que l'on insère n entiers aléatoires dans le dictionnaire, préciser la complexité en temps de la fonction recherche en fonction de k

Pour supprimer des éléments d'un dictionnaire, la suppression et la mise à jour des deux tables coûte relativement cher en temps et n'est donc pas implémentée de la sorte. Une idée consiste à remplacer la valeur associée à la clé à supprimer par None (par exemple) et de ne pas tenir compte des None dans les fonctions précédentes. Dans un objectif pédagogique, et si vous êtes en avance traitez la question suivante. Sinon, sautez là !

Question 9: Proposer une fonction supprime(cle) qui supprime la clé du dictionnaire en supprimant la ligne associée dans la table de débordement en mettant à jour tous les indices qui le nécessitent dans les deux tables.
Vous vérifierez que votre fonction fonctionne.

Exercice 2: Complexité

Question 1: Créer un dictionnaire Dico contenant N=1000 entiers i de 1 à 1000 tel que Dico[i]=i
Question 2: Observer le hachage de Python hash() pour des entiers entre 1 et 1000 et estimer la complexité de la recherche d'une clé dans Dico

Pour rappel, le code suivant permet d'afficher dans la console le temps d'exécution des instructions entre tic et toc :

```
from time import perf_counter as tps
tic = tps()
# instructions
toc = tps()
temps = toc - tic
print(temps)
```

On souhaite étudier le temps de recherche de la clé N dans le dictionnaire afin de se placer dans le pire des cas.

On souhaite étudier les instructions suivantes :

- Instruction 1 : `Val in Dico`
- Instruction 2 : `Val in Dico.keys()`
- Instruction 3 : `Val in L` avec avoir écrit au préalable `L = list(Dico.keys())`

On appelle Ti le temps d'exécution de l'instruction i.

Question 3: Etudier les temps d'exécution Ti pour N en puissance de 10 de 10^3 à 10^8
Remarque : vous pourrez vérifier que hash(10^8) est toujours égal à 10^8

Question 4: Préciser les conditions dans lesquelles obtient-on le pire des cas lors d'une recherche dans un dictionnaire et la complexité associée
Question 5: Préciser la complexité en temps lors de la recherche dans un dictionnaire

Soient :

```
dic1 = {1:1}
dic2 = {1.0:1}
L = [1]
```

On appelle T1 le temps d'accès à un terme de dic1, T2 le temps d'accès à un terme de dic2 et T3 le temps d'accès à un terme de L.
Question 6: Etudier la moyenne du rapport T1/T3 et T2/T3 sur un grand nombre d'essais et préciser l'origine des différences observées

Sujet inspiré du travail de Bernard SALAMITO
et d'un TD de licence d'informatique de l'université de Nice