Table de hachage : implémentation

INFORMATIQUE COMMUNE - TP n° 3.1 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ utiliser les listes Python
- ☞ écrire des fonctions en Python
- ☞ utiliser une bibliothèque en l'important correctement
- ☞ expliquer le fonctionnement d'une table de hachage (dictionnaire)

L'objectif de ce TP est de construire une table de hachage «à la main», un équivalent des `dict` Python. Dans ce but, il faut dans un premier temps disposer d'une fonction de hachage adaptée. C'est l'objet de la première partie. La seconde partie se focaliser sur l'implémentation de la table de hachage.

On rappelle sur la figure 1 le principe du dictionnaire (ou table de hachage).

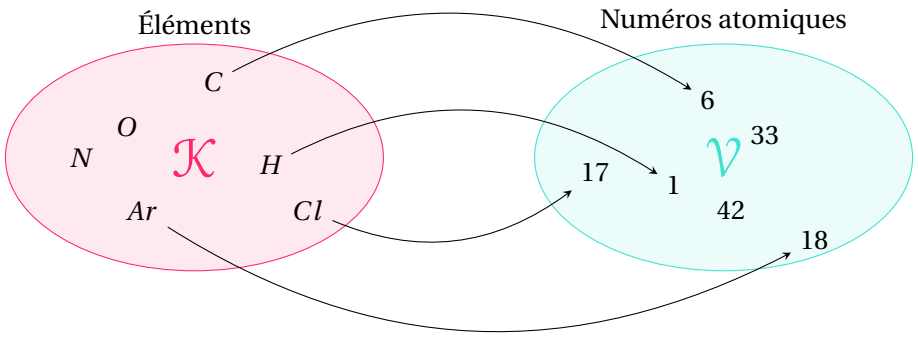

FIGURE 1 – Illustration du concept de dictionnaire : tableau associatif reliant une clef à un numéro atomique.

A Fonctions de hachage et uniformité

R On rappelle que le choix d'une fonction de hachage est délicat et qu'il n'existe pas de méthode pour atteindre l'optimal.

Lorsque les clefs d'une table de hachage sont des chaînes de caractères, il est souvent possible de décomposer une fonction de hachage h en deux étapes :

1. une fonction h_e qui encode la clef d'entrée en un nombre entier (encodage),

2. et une fonction h_c qui compresse ce nombre entier dans l'ensemble des indexes (compression).

Plus formellement, on la fonction de hachage comme une fonction composée :

$$h_e : \mathcal{K} \longrightarrow \mathbb{N} \tag{1}$$
$$h_c : \mathbb{N} \longrightarrow [\![0, n-1]\!] \tag{2}$$

et

$$h = h_c \circ h_e \tag{3}$$

a Encodage

L'idée de l'encodage est de générer un nombre représentant une chaîne de caractère. Deux approches sont considérées :

γ Cette fonction calcule un entier unique pour chaque chaîne de caractères comme suit :

$$\gamma(s) = \sum_{k=0}^{|s|-1} \text{ascii}(s_k) 2^{8k} \tag{4}$$

où $\text{ascii}(s_k)$ est le code ASCII associé au caractère d'indice k de s.

γ_p On dispose des 100 premiers nombres premiers sous la forme d'une variable globale :

```
PRIMES = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
    67, 71, 73, 79, 83, 89, 97, 101, 103,  107, 109, 113, 127, 131, 137, 139,
    149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227,
    229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311,
    313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401,
    409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491,
    499, 503, 509, 521, 523, 541]
```

Cette fonction procède comme suit :

$$\gamma_p(s) = \sum_{k=0}^{|s|-1} \text{PRIMES}[(\text{ascii}(s_k) + k) \bmod 100] \tag{5}$$

P En Python, la fonction `ord` permet d'obtenir le code ASCII associé à un caractère documentation).

A1. À quoi sert le facteur multiplicatif 2^{8k} dans la formule de γ ? Quelle qualité mathématique confère-t-il à la fonction γ ?

A2. Programmer une fonction `gamma(s : string): int` qui calcule $\gamma(s)$.

A3. La fonction γ_p engendre-t-elle des codes tous différents ? Pourquoi ? Que peut-on en conclure sur la fonction γ_p ?

A4. Programmer une fonction `gammap(s : string): int` qui calcule $\gamma_p(s)$.

b Compression

Dans un second temps, on cherche à **compresser la valeur encodée dans l'intervalle des index possibles** $[\![0, n-1]\!]$, si n est la taille de la table de hachage.
Une distribution uniforme des clefs dans l'espace d'arrivée peut être obtenue en utilisant des générateurs aléatoires. Les générateurs à congruence linéaires, c'est à dire les fonctions du type $(ax+b) \mod n$ sont de bons candidats pour les fonctions de hachage, pourvu qu'on choisisse bien les constantes a et b du générateur.
On peut choisir :

1. d'utiliser simplement le reste d'une division :

$$h_d : (s, n) \to \gamma(s) \mod n \tag{6}$$

2. d'utiliser une extraction de partie fractionnaire et une multiplication :

$$h_\alpha : (s, n) \to \lfloor n \times \text{frac}(\alpha\gamma(s)) \rfloor \tag{7}$$

$\alpha \in]0,1[$ étant une constante réelle et frac est l'opérateur qui renvoie la partie fractionnaire d'un nombre flottant. Par exemple, frac(42.567) = 0.567. Cette opération peut être réalisée par l'opération `%1` en Python.

A5. Coder les fonctions h_d et h_α en Python. Les paramètres n et α pourront être pris par défaut à 47057 et $\frac{\sqrt{5}-1}{2}$.

A6. Importer tous les mots contenus dans le fichier `"english_words.csv"` dans un liste.
On cherche à tester l'uniformité de la distribution des codes obtenues des fonctions de hachage. On peut facilement vérifier ceci en utilisant le test de Kolmogorov-Smirnov et la bibliothèque scipy et l'instruction :

```
scipy.stats.kstest(codes, "uniform")
#KstestResult(statistic=0.0012179563749926126, pvalue=0.49280753163611735)
```

Si le paramètre `p_value` est plus grand que 0.05, alors la distribution peut être considérée comme uniforme. Le paramètre `statistic` donne une mesure de la distance entre les deux distributions.

A7. Écrire une fonction dont le prototype est `uniform_test(h, table_size)` dont le paramètre `h` est une fonction de hachage et `table_size` la taille de la table de hachage. Cette fonction renvoie le résultat du test de Kolmogorov-Smirnov entre une distribution uniforme et la distribution des codes obtenus avec `h` sur l'ensemble des mots du fichier `"english_words"`. La fonction de `scipy` nécessite un tableau d'entrée Numpy dont les données sont de type `float`.

A8. Observer les résultats de la fonction précédente pour h_d et h_α en utilisant soit γ soit γ_p et faisant varier la taille de la table de hachage. Que pouvez-vous en conclure?

A9. Afficher les histogrammes associés aux différentes distribution de codes obtenues à l'aide de la bibliothèque `matplotlib` et à la fonction `hist`.

B Implémentation d'une table de hachage

On souhaite créer une table de hachage d'après un fichier qui recense les capitales des pays du monde entier. Cette table possède donc des clefs de type `str` (le pays) et des valeurs de type `str` (la capitale).

B1. Écrire une fonction `import_csv()` qui importe les données du fichier `"capitals.csv"`. Cette fonction renvoie une liste de tuples `(pays, capitale)`.

B2. Écrire une fonction de prototype `init_hash_table(elements,table_size)` qui renvoie une table de hachage initialisée d'après le paramètre `elements`. Ce paramètre est la liste de tuples créée à la question précédente.

B3. Écrire une fonction de prototype `get_value(table, table_size, input_key)` qui permet d'accéder à l'élément de clef `input_key` de la table de hachage `table`. Par exemple, `get_value(ht, "Italy"), n` renvoie `"Roma"`.

B4. Créer l'ensemble de toutes les clefs de la table pour lesquelles il existe une valeur, puis parcourir la table à partir de cet ensemble. Les capitales apparaissent-elles dans un ordre quelconque ? Faire apparaître les sous-listes de la table s'il y en a. Combien y-a-t-il de clefs si on utilise h_d ? Même question si on utilise la fonction interne de Python :

```
def hashp(s, table_size=TABLE_SIZE):
  return hash(s) % table_size
```

R Naturellement, si par la suite vous avez besoin d'une table de hachage, il faut utiliser le type `dict` de Python et ne pas réinventer la poudre !