PROJET 8

Propriétés spatiales des tables de hachage universelles

sujet proposé par Xavier Erny

xavier.erny@polytechnique.edu

Figure 8.1: Une table de hashage selon DALL-E. . .

1 Table de hachage : présentation

En informatique, il est souvent utile de pouvoir manipuler un ensemble de données (que l'on peut implémenter comme une "List" en Python) pouvant exécuter les opérations de bases suivantes (entre autres...) :

- accès à un élément déjà présent,
- modification d'un élément déjà présent
- ajout d'un nouvel élément.

Dans beaucoup de langages de programmation (notamment parmi ceux plus bas niveau que Python), il y a deux implémentations "naturelles" du type "List" de Python : les tableaux et les listes chaînées (ou

piles). Si on dispose de n données, l'avantage des tableaux est que les opérations "accès à un élément" et "modification d'un élément" sont rapides (le cout est proche de $O(1)$), mais que l'opération "ajout d'un élément" est lente (de l'ordre de $O(n)$), ce qui est exactement l'inverse des listes chaînées. Le but de ce projet est d'étudier un compromis entre les tableaux et les listes chaînées : les tables de hachage.

Dans les trois premières sections du projet, nous définissons formellement une table de hachage comme un tableau contenant des listes chaînées, ce que l'on implémentera en Python comme une List de List. Chaque liste chaînée sera appelée une alvéole. Le nombre d'alvéoles d'une table de hachage donnée est non-modifiable (en particulier, si vos programmes le modifient, alors ils ne seront pas corrects) : vous ne pouvez pas ajouter d'alvéole. Mais vous pouvez ajouter des éléments dans les alvéoles déjà présentes. Autrement dit, vous n'avez pas le droit d'utiliser la commande "append" pour ajouter une alvéole, mais vous avez le droit de l'utiliser pour ajouter un élément dans une alvéole. Ci-dessous, t est une table de hachage implémentée en python à 4 alvéoles : la première contient les éléments 2,8,6, la deuxième est vide, la troisième contient seulement l'élément 1, et la quatrième contient les éléments 5,4,15,9.

```
t = [[2,8,6],
     [],
     [1],
     [5,4,15,9]]
```

Remarque. *Pour créer une table de hachage m à 4 alvéoles (par exemple) initialement vides (sans utiliser la fonction append), vous pouvez utiliser le code ci-dessous*

```
t0 = [[] for i in range(4)]
```

Étant donné un ensemble de données $\mathcal{D}$, pour construire une table de hachage à m alvéoles contenant n données $d_0, ..., d_{n-1} \in \mathcal{D}$, il faut définir ce que l'on appelle une fonction de hachage $h : \mathcal{D} \to [\![0, m-1]\!]$, qui étant donné une donnée d à stocker dans la table, renvoit l'alvéole $h(d)$ dans laquelle d sera insérée (on dit que d est hachée dans l'alvéole $h(d)$). Par exemple, avec l'exemple de la table de hachage t ci-dessus, on a $m = 4$ alvéoles, $\mathcal{D} = \mathbb{N}$ et la fonction de hachage h utilisée vérifie : $h(2) = h(8) = h(6) = 0$, $h(1) = 2$ et $h(5) = h(4) = h(15) = h(9) = 3$.

S1. Écrire un programme qui, étant donné une table de hachage, une fonction de hachage et une donnée, modifie la table en insérant la donnée dans la bonne alvéole. Vous pouvez supposer que les données sont de type Int.
Appliquer votre programme pour insérer la donnée 7 via la fonction de hachage $h(n) = \lfloor n/4 \rfloor$ (où $\lfloor x \rfloor$ est la partie entière de x) dans la table de hachage suivante (et afficher la table ainsi obtenue)

```
[[1,3,2,0],
[6],
[],
[],
[17,18]]
```

S2. Écrire un programme qui, étant donné une table de hachage (implémentée comme expliqué plus haut), une fonction de hachage et une donnée, teste si la donnée est présente dans la table. Vous pouvez supposer que les données sont de type Int.
Vérifier votre programme en testant successivement la présence des données 7, 15 et 18 dans la table obtenue à la fin de **S1**.

S3. Écrire un programme qui, étant donné une List d'entiers compris entre 0 et 19, construit la table de hachage à 5 alvéoles associée à la fonction de hachage $h(n) = \lfloor n/4 \rfloor$, telle que chaque donnée ne soit présente qu'une fois dans la table (même si la donnée est présente en double dans la List). Il ne sera pas nécessaire de vérifier que les entiers de la List sont compris entre 0 et 19.

Utiliser votre programme pour constuire la table (que vous afficherez) otenue par insertion successives des données suivante (dans cet ordre) : 5, 1, 7, 6, 5, 9, 15, 0, 18.

Dans toute la suite, on supposera que les tables de hachages ne contiennent pas de doublon.

La recherche d'une donnée d dans une table de hachage a l'avantage que vous n'avez pas besoin de rechercher parmi toutes les données stockées, mais seulement parmi celles qui sont hachées dans l'alvéole $h(d)$. C'est donc sans surprise que les complexités moyennes des algorithmes sur les tables de hachage dépendent du facteur de remplissage $\alpha = n/m$ (avec n le nombre de données stockées et m le nombre d'alvéoles). Dans une table de hachage "bien construite", toutes les alvéoles ont une taille proche de α. Quand m est petit (par exemple $m = 1$ et $\alpha \gg 1$), on ne profite pas de la structure de la table de hachage pour optimiser le temps d'exécution des programmes. Inversement, si $m \gg n$ (et donc $\alpha \ll 1$), il y a beaucoup d'alvéoles vides, ce qui est un gachis de ressources en espace.

Le but du projet est d'étudier des méthodes de construction de tables de hachage qui ont de bonnes propriétés en moyenne.

2 Hypothèse de hachage uniforme simple

Dans toute cette section, nous noterons $h : \mathcal{D} \to [\![0, m-1]\!]$ la fonction de hachage (déterministe), et $D_0, ..., D_{n-1}$ les données (v.a. sur $\mathcal{D}$) qui sont insérées successivement dans une table de hachage à m alvéoles initialement vide. Soit T la table de hachage ainsi obtenue. Pour $0 \leq j \leq m-1$, nous noterons N_j la taille de l'alvéole j:

$$N_j = |\{0 \leq i \leq n-1 \ : \ h(D_i) = j\}|.$$

Définition. *Nous dirons que $(T, D_0, ..., D_{n-1})$ satisfait l'hypothèse de hachage uniforme simple (HHUS) si les variables $h(D_i)$ ($0 \leq i \leq n-1$) sont i.i.d. de loi $\mathcal{U}([\![0, m-1]\!])$, et si, avec probabilité 1, les variables D_i ($0 \leq i \leq n-1$) sont distinctes deux-à-deux.*

En théorie, la seconde condition de l'HHUS n'est pas nécessaire, mais elle permet d'éviter des difficultés techniques sans grande importance dans les questions de cette section.

T1. Supposons que $D_0, ..., D_{n-1}$ sont i.i.d. de loi $\mathcal{U}([0, 1[)$, et $h : d \in [0, 1[\mapsto \lfloor dm \rfloor$. Montrer que l'HHUS est vérifiée.

S4. La table de hachage obtenue à **T1** est aléatoire (car les données sont des v.a.). Écrire un programme qui simule cette table de hachage aléatoire avec $m = 5$ alvéoles en y insérant $n = 1000$ données i.i.d. de loi $\mathcal{U}([0, 1[)$. Utiliser votre programme pour simuler une telle table, puis afficher la taille de ses alvéoles à l'aide d'un histogramme.

T2. Supposons que l'HHUS est vérifiée. Pour $0 \leq i, j \leq n-1$, donner la probabilité que les données D_i et D_j soient hachées dans la même alvéole. Donner la loi de N_j ($0 \leq j \leq m-1$).
Indication: Pour la deuxième partie de la question, commencer par montrer que, pour tout $0 \leq j \leq m-1$,

$$N_j = \sum_{i=0}^{n-1} 1_{\{h(D_i)=j\}}.$$

Dans les deux questions suivantes, nous considérons l'algorithme de recherche de données de **S2**. Pour estimer le nombre de comparaisons que doit effectuer cet algorithme pour tester si une donnée d est présente dans la table T, il y a deux scénarios :

- soit d n'est pas dans la table, alors le nombre de comparaisons est le nombre total de données insérées dans l'alvéole $h(d)$ (car il faut parcourir toute l'avéole pour être sûr que d n'est pas présente),

- soit d est dans la table, alors le nombre de comparaisons est le nombre de données qui ont été insérées dans l'alvéole $h(d)$ avant d, ce qui inclut d (car une fois que d est trouvée, on peut arrêter la recherche).

T3. Supposons que l'HHUS est vérifiée. Soit D une v.a. sur $\mathcal{D}$ qui est presque sûrement différente de toutes les D_i $(0 \leq i \leq n-1)$. Quelle est l'espérance du nombre de comparaisons de données qui devront être faîtes par l'algorithme de **S2** qui teste si D est présente dans la table H ?
Indication : commencer par donner le nombre exact de comparaisons (qui est aléatoire) en utilisant l'indication de **T2**.

T4. Supposons que l'HHUS est vérifiée. Soit I une v.a. sur $[\![0, n-1]\!]$. Quelle est l'espérance du nombre de comparaisons qui devront être faîtes par l'algorithme de **S2** qui teste si D_I est présente dans la table H ?
Indication : même indice que pour **T3**.

3 Méthode de hachage universel

Étant donné un ensemble $\mathcal{D}$ et un entier $m \geq 1$, on dit qu'une collection finie de fonctions $h_j : \mathcal{D} \to [\![0, m-1]\!]$ $(0 \leq j \leq q-1)$ est une collection de fonctions de hachage universelle si, pour tout $d \neq d'$, le nombre de fonctions h_j $(0 \leq j \leq q-1)$ telles que $h_j(d) = h_j(d')$ est inférieur à q/m.

Dans toute cette section, nous noterons $h_0, ..., h_{q-1}$ une collection de fonctions de hachage universelle (déterministe), et $d_0, ..., d_{n-1}$ les données (éléments déterministes de $\mathcal{D}$) qui sont insérées successivement dans une table de hachage à m alvéoles initialement vide, où la fonction de hachage utilisée est h_I avec I v.a. de loi $\mathcal{U}([\![0, q-1]\!])$ (c'est la même fonction h_I qui est utilisée pour hacher toutes les données d_i dans la table). Nous noterons T la table ainsi obtenue. Nous supposons que les données d_i $(0 \leq i \leq n-1)$ sont deux-à-deux distinctes. Pour $0 \leq j \leq m-1$, nous notons encore N_j la taille de l'alvéole j

$$N_j = |\{0 \leq i \leq n-1 \ : \ h_I(d_i) = j\}|.$$

Nous appellerons méthode de hachage universel, cette méthode de construction.

T5. Soit $d \in \mathcal{D}$ différente des données $d_0, ..., d_{n-1}$. Montrer que l'espérance du nombre de comparaisons qui devront être faîtes par l'algorithme qui teste si d est présente dans la table H est inférieure au facteur de remplissage $\alpha = n/m$.

T6. Soit $i \in [\![0, n-1]\!]$. Montrer que l'espérance du nombre de comparaisons qui devront être faîtes par l'algorithme qui teste si d_i est présente dans la table H est inférieure à $1 + \alpha$.

Application avec des données de $(\mathbb{Z}_p)^l$

Dans la suite, pour tout entier $p \geq 2$, nous notons $\mathbb{Z}_p$ (aussi couramment noté $\mathbb{Z}/p\mathbb{Z}$) l'ensemble des classes d'équivalence pour la relation de congruence modulo p : deux entiers sont équivalents si et seulement s'ils ont le même reste dans la division euclidienne par p. On admettra que si p est un nombre premier, alors $\mathbb{Z}_p$ (muni de l'addition et de la multiplication habituelles) est un corps.

T7. (facultative). Soient $l \in \mathbb{N}^*$, p un nombre premier et $\mathcal{D} = (\mathbb{Z}_p)^l$. Pour $a \in \mathbb{Z}_p$, soit

$$h_a : d = (d_0, ..., d_{l-1}) \in \mathcal{D} \mapsto \sum_{j=0}^{l-1} d_j a^j \in \mathbb{Z}_p.$$

Étant donné deux éléments distincts b, c de $\mathcal{D}$, montrer qu'il y a au plus $l-1$ éléments $a \in \mathbb{Z}_p$ tels que $h_a(b) = h_a(c)$.

T8. Soient p un nombre premier et $\mathcal{D} = (\mathbb{Z}_p)^2$. Donner un entier m et une collection de fonctions de hachage universelle.

S5. Écrire un programme qui, étant donné une List d'éléments de $(\mathbb{Z}_{11})^2$ (un élément de $(\mathbb{Z}_{11})^2$ pourra être implémenté par une List de longueur 2 ou par un couple), construit une table de hachage par une méthode de hachage universelle. Utiliser votre programme pour construire la table (puis afficher la) contenant les données suivantes

```
dS5 = [[5,8],[0,0],[3,1],[10,5],[6,2],[1,5]]
```

Contrôle du nombre de collisions

Par définition, les collisions dans une table de hachage sont les paires de données hachées dans une même alvéole. L'ensemble $\mathcal{C}$ des collisions est formellement décrit de la façon suivante

$$\mathcal{C} = \{\{k,l\} \subseteq [\![0, n-1]\!] \ : \ k \neq l \text{ et } h_I(d_k) = h_I(d_l)\}.$$

On notera X le nombre de collisions dans la table T.

T9. Supposons $m \geq n^2$. Montrer que l'espérance du nombre de collisions X dans la table T est strictement inférieure à $1/2$. En déduire que la probabilité qu'il n'y ait aucune collision est strictement supérieure à $1/2$.
Indication : On pourra exploiter (et réarranger la somme)

$$X = \sum_{\{k,l\} \subseteq [\![0,n-1]\!]} 1_{\{k,l\} \in \mathcal{C}}.$$

T10. Si $m \geq n^2$, étant donné n données $d_0, ..., d_{n-1}$, on cherche une fonction de hachage h parmi $h_0, ..., h_{q-1}$ qui n'a aucune collision. Pour ça, on en choisit une au hasard (uniformément) parmi les fonctions $h_0, ..., h_{q-1}$ jusqu'à en trouver une sans collision (si la fonction choisie a une collision, il faut en rechoisir une aléatoirement indépendamment des tirages précédents). Donner la loi du nombre de tirages que vous devrez faire avant de trouver une fonction de hachage sans collision (le résultat dépend de la probabilité d'avoir une collision : $\mathbb{P}(X \geq 1)$, que l'on ne cherchera pas à expliciter). Montrer que la probabilité que le nombre de tirages soit strictement supérieur à k, est inférieure à 2^{-k}.

S6. En déduire un programme qui, étant donné une List de $n \leq 14$ éléments de $(\mathbb{Z}_{211})^2$, construit une table de hachage à $m = 211$ alvéoles sans aucune collision (i.e. toutes les alvéoles sont de taille 0 ou 1).
Utiliser votre programme pour construire une table sans collision contenant les données suivantes

```
dS6 = [[5,8],[0,0],[3,1],[10,5],[6,2],[1,5],[4,2],[5,7],[3,5],[6,9],[0,2]]
```

L'avantage de la table de hachage construite à **S6** (qui suit la méthode de **T10**) est que toutes les alvéoles sont de tailles 0 ou 1. L'inconvénient est l'espace gâché : le facteur de remplissage est $1/n$ qui peut être proche de zéro. Dans la section suivante, nous verrons comment exploiter **T10** sans avoir un facteur de remplissage qui tend vers zéro quand n tend vers l'infini.

4 Méthode de hachage parfait

L'idée du hachage parfait est de construire une table de hachage dont les alvéoles sont des tables de hachage sans collision. Il y a donc deux niveaux de hachage. La fonction de hachage pour la table principale sera choisie selon une méthode de hachage universel (comme décrit dans la section

précédente). Pour chaque alvéole j, il faut choisir une fonction de hachage $h^{[j]}$ pour garantir l'absence de collision (en suivant les questions **T10** et **S6**).

Pour construire une table de hachage selon la méthode du hachage parfait, il faut connaître dès le départ l'ensemble des données à stocker. Une fois la table construite, il n'est pas possible d'insérer a posteriori de nouveaux éléments tout en garantissant l'absence de collisions au second niveau de hachage.

Nous notons m le nombre d'alvéoles de la table de hachage, $d_0, ..., d_{n-1}$ les données à stocker, et N_j le nombre de données hachées dans l'alvéole j. Avant d'insérer les données dans la table à m alvéoles, il faut choisir le nombre d'alvéoles M_j de l'avéole j (car chaque alvéole est une table de hachage) : les nombres M_j ne sont pas modifiables.

Le but de la question suivante est d'écrire un programme qui calcule pour chaque alvéole j, le nombre de données N_j qui seront hachées dans cet alvéole.

S7. Écrire un programme qui, étant donné un entier m, une List de données et une fonction de hachage à valeurs dans $[\![0, m-1]\!]$, renvoit une List l de taille m telle que $l[j]$ est le nombre de données hachées dans l'alvéole j (votre programme n'est pas censé construire de table de hachage).
Afficher sous la forme d'un histogramme la taille des alvéoles de la table construite à **S5** (sans reconstruire la table).

Dans le contexte du hachage parfait, on connait à l'avance le nombre de données N_j stockées dans chaque alvéole j. Le principe de la méthode de hachage parfait est de garantir l'absence de collision au sein des alvéoles. Une condition nécessaire évidente pour avoir cette propriété est "pour tout $0 \leq j \leq m-1$, $M_j \geq N_j$" (par le principe des tiroirs). Toutefois, si les fonctions de hachage sont choisies parmi une collection universelle $\mathcal{H}$, le choix de $M_j = N_j$ ne permet pas de garantir qu'il existe une fonction $h^{[j]} \in \mathcal{H}$ telle que $h^{[j]}$ n'a pas de collision pour les N_j données à insérer dans l'alvéole j (qui contient M_j alvéoles). Il faut donc choisir des tailles d'alvéoles M_j apprioriées.

Conformément à ce qui a été montré à la question **T10**, nous choisirons d'implémenter l'alvéole j par une table de hachage de taille $M_j = N_j^2$ associée à une fonction de hachage $h^{[j]}$ choisie de telle sorte à ce qu'il n'y ait aucune collision dans la table de hachage de l'alvéole j. La taille totale de la table de hachage est alors $\sum_{j=0}^{m-1} N_j^2$.

On peut penser qu'il y a un gachis d'espace important (car l'alvéole j est de taille N_j^2 pour stocker N_j données). En particulier, si toutes les n données sont hachées dans la même alvéole de la table principale, alors la taille totale est n^2, et le facteur de remplissage est $1/n$ comme dans la question **S6**. Les deux questions suivantes montrent que la taille totale de la table est "souvent" d'ordre n.

T11. En supposant que $m \geq n$, montrer que l'espérance de la taille totale (i.e. la somme des tailles des alvéoles) de la table construite par hachage parfait est strictement inférieure à $2n$.
Indication: Montrer que $N_j^2 = N_j + 2X_j$ avec X_j le nombre de collisions dans la table de hachage principal venant de l'alvéole j.

T12. En supposant que $m \geq n$, montrer que, pour tout $0 < \varepsilon < 1$, il existe un entier $t(\varepsilon)$ tel que, avec probabilité supérieure à $1 - \varepsilon$, la taille totale de la table de hachage ne dépasse pas $nt(\varepsilon)$.

S8. Donner une valeur approchée de l'espérance de la taille totale de la table de hachage à $m = 11$ alvéoles construite par la méthode de hachage parfait (la fonction de hachage principale est choisie uniformément parmi les fonctions h_a définies à **T7** avec $p = 11$ et $l = 2$) via la liste de données `lEx` définie ci-dessous (qui sont des éléments de $\mathbb{Z}_{11}$). On garantira que l'erreur commise est inférieure 4, avec probabilité supérieure à 0.5.
Indication : démontrer (et exploiter) que la variance de la taille totale de la table est inférieure à $m \cdot n^4$ (c'est une borne relativement grossière)

```
lEx = [[5,8],[0,0],[3,1],[10,5],[6,2],[1,5],[4,7],[2,2],[10,7],[5,4]]
```

S9. Tracer un histogramme de la répartition empirique de la taille totale de la table de hachage (avec les mêmes paramètres que dans la question **S8**). Vous pouvez prendre un échantillon de taille 10000.