Tables de hachage

HAI503I – Algorithmique 4

Bruno Grenet

Université de Montpellier – Faculté des Sciences

Les dictionnaires Python

Comment implanter le type `dict` de Python ?

```python
>>> d = {}                                  # Dictionnaire vide
>>> d[1515] = 'Bataille de Marignan'        # Ajout d'élément
>>> d[1492] = 'Colomb en Amérique'
>>> d[1492] = 'Ascension du Mont Aiguille'  # Modification
>>> 1789 in d                               # Recherche
False
>>> d[1492]
'Ascension du Mont Aiguille'
```

La structure de données *dictionnaire*

1515 "Bataille ..."

Version algorithmique

- Ensemble de couples (clef, valeur)
- Opérations disponibles :
  - CRÉATION d'un dictionnaire vide
  - INSERTION d'un couple
  - Modification d'une valeur $\rightarrow$ RÉINSERTION
  - RECHERCHE d'une clef $\rightarrow$ renvoie la valeur ou une erreur

Objectif

Les opérations CRÉATION, INSERTION et RECHERCHE doivent être *rapides* !

Hypothèse simplificatrice

- Les clefs sont des entiers
- Théorie : toute donnée est codée en binaire $\rightarrow$ interprétation comme un entier
- Pratique : on se ramène à des entiers, mais pas forcément de cette façon

Quelles solutions ?

Dictionnaire de n éléments, clef entre 0 et $N-1$

Tableau

- Taille : N
- Création : $O(N)$
- Insertion : $O(1)$
- Recherche : $O(1)$

Liste chaînée

- Taille : n
- Création : $O(1)$
- Insertion : $O(n)$
- Recherche : $O(n)$

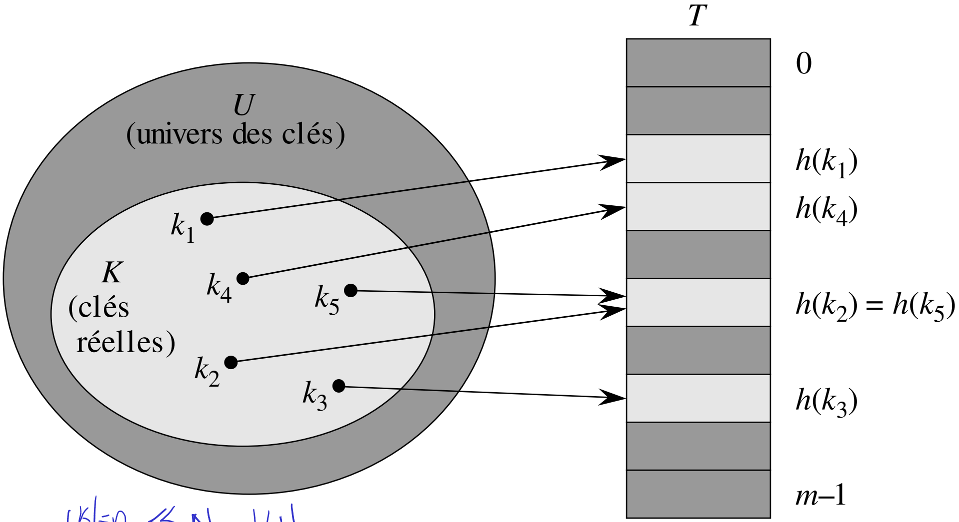
Arbre binaire de recherche

- Taille : n
- Création : $O(1)$
- Insertion : $O(h) \to O(\log n)$ si équilibré
- Recherche : $O(h) \to O(\log n)$ si équilibré

Tas

- Taille : n
- Création : $O(1)$ ou $O(n)$
- Insertion : $O(\log n)$
- Recherche : $O(\log n)$

Tables de hachage



$|K| = n \ll N = |U|$

Source : T. H. Cormen, C. Leiserson, R. Riverst, et C. Stein, Introduction à l'algorithmique. 2ème éd. Dunod, 2004.

Formalisation

Clefs

- Univers U des clefs possibles : $U = \{0, \ldots, N-1\}$
- Clefs utilisées : $K \subset U$, de taille n

Table de taille m

- Indices entre 0 et $m-1$
- Une case contient une valeur, voire plusieurs
- Une case peut être vide

Fonction de hachage

- Fonction $h : U \to \{0, \ldots, m-1\}$

INSERTION du couple (k, v) dans la case $T_{[h(k)]}$

Questions à résoudre

Collisions

- Que fait-on si $h(k_1) = h(k_2)$?
  - Plusieurs valeurs dans une case (liste chaînée, etc.)
  - Utiliser une autre case ?
- Est-ce que $h(k_1) = h(k_2)$ arrive souvent ?
  - Comment choisir h ?

Caractéristiques

- Taille : $m \to$ comment choisir m par rapport à n et N ?
- Création : $O(m)$
- Insertion : calcul de $h(k)$ puis insertion en case $h(k) \to$ quelle complexité ?
- Recherche : calcul de $h(k)$ puis recherche dans la case $h(k) \to$ quelle complexité ?

Problématique des fonctions de hachage

Contexte

- Choix d'une fonction $h : \{0, \ldots, N-1\} \to \{0, \ldots, m-1\}$
- Fonction utilisée pour un ensemble de clefs K de taille $n \ll N$

Collisions évitables ?

- Avec $N \gg m$, forcément des collisions $h(k_1) = h(k_2)$!
- Mais on stocke n clefs : si $n \leq m$?
  - Pour un ensemble de clefs, possible de trouver h sans collision
  - Mais… on ne connaît pas les clefs à l'avance !

Problématique

- On veut choisir h avant de connaître les clefs
- On voudrait éviter les collisions entre clefs… sans les connaître !

> Pas le choix : une fonction de hachage doit être choisie aléatoirement !

Modèles aléatoires des fonctions de hachage

On tire h uniformément parmi les fonctions de U dans $\{0, \dots, m-1\}$

Représentation de h

- Pour chaque k, une valeur $h(k) \to$ tableau H de taille N
- Tirage de $h \to$ tirage uniforme et indépendant de chaque $H_{[k]}$ dans $\{0, \dots, m-1\}$

Avantage et inconvénient

- Avantage : très bonnes propriétés probabilistes
  - Pour tout $k_1 \neq k_2$, $\Pr_h[h(k_1) = h(k_2)] = 1/m$
- Inconvénient : totalement **irréaliste** $\to$ tableau de taille N, temps du tirage

Remarques

- Parfois utilisé en théorie car
  - les preuves sont (un peu) simples
  - les résultats obtenus parfois (très) proches du comportement pratique
- Objectif : modèle réaliste avec propriétés proches

Modèle universel des fonctions de hachage

On fixe un ensemble $\mathcal{H}$ de fonctions de hachage et on tire h uniformément dans $\mathcal{H}$

Définition

Un ensemble $\mathcal{H}$ de fonctions de $\{0, \ldots, N-1\}$ dans $\{0, \ldots, m-1\}$ est universel si pour tout $k_1 \neq k_2$, $\Pr_{h\in\mathcal{H}}[h(k_1) = h(k_2)] \leq 1/m$.

Remarques

- Probabilité que deux éléments collisionnent $\leq$ probabilité dans le modèle aléatoire
- L'ensemble de toutes les fonctions est universel… mais irréaliste !
- On *sait* construire des ensembles $\mathcal{H}$ universels réalistes

Ensemble universel *intéressant*

- Ensemble pas trop gros $\to$ représentation de h assez petite
- Tirer uniformément $h \in \mathcal{H}$ doit être efficace
- Calculer $h(k)$ doit être rapide

Problématique

Contexte

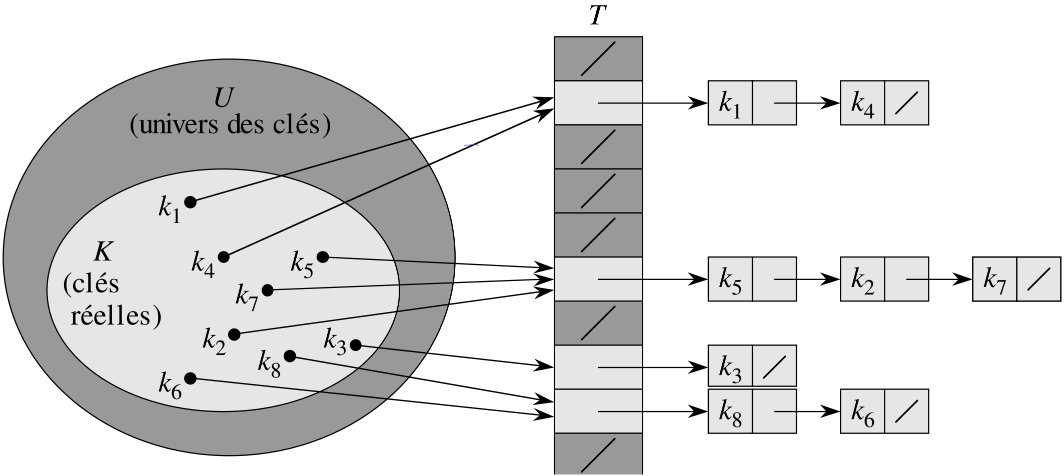
- Table T avec fonction de hachage $h : \{0, \ldots, N-1\} \to \{0, \ldots, m-1\}$
- Ensemble de clefs K

> Que fait-on si $h(k_1) = h(k_2)$ pour deux clefs $k_1 \neq k_2$?

Deux (familles de) solutions

- Mettre plusieurs éléments dans une même case
  - Résolution par *chaînage*
  - *Hachage parfait*
- Trouver une autre case libre : *adressage ouvert*

Résolution par chaînage : principe



Source : T. H. Cormen, C. Leiserson, R. Riverst, et C. Stein, Introduction à l'algorithmique. 2ème éd. Dunod, 2004.

Résolution par chaînage

Chaque case de T contient une liste chaînée

Algorithmes

- RECHERCHE de k :
  - Calcul de $h(k)$ $\to$ $\Theta(1)$
  - Parcours de la liste contenue dans $T_{[h(k)]}$
  - Complexité : $O(\ell(k))$ où $\ell(k)$ est la taille de la liste $T_{[h(k)]}$
- INSERTION de (k, v) :
  - Idem RECHERCHE pour savoir si k est dans le dictionnaire
  - Si k apparaît déjà dans la liste contenue dans $T_{[h(k)]}$, on remplace sa valeur par v
  - Sinon, on ajoute (k, v) à la liste $T_{[h(k)]}$
  - Complexité : $O(\ell(k))$ où $\ell(k)$ est la taille de la liste $T_{[h(k)]}$

Quelle efficacité ?

- Une opération coûte $O(L)$, où $L = \max_{k \in K} \ell(k)$ $\to$ quelle taille maximale ?

Efficacité de la résolution par chaînage

Théorème

Soit T une table de hachage de taille m, avec h tirée uniformément dans un ensemble $\mathcal{H}$ universel. Si T contient n éléments et que les collisions sont résolues par chaînage, l'espérance de la complexité de l'INSERTION et de la RECHERCHE est $O(n/m)$.

↳ $\alpha = \frac{n}{m}$: taux de remplissage

Preuve

Il suffit de regarder le cas où on fait une RECHERCHE infructueuse.

On veut montrer que si $k \notin T$, $\mathbb{E}_h[\ell(k)] = \Theta(n/m)$

On sait que pour $k' \neq k$ $\Pr_h[h(k) = h(k')] \leq 1/m$.

$$\ell(k) = |\{k' \in K : h(k') = h(k)\}|$$

$$\mathbb{E}[\ell(k)] = \sum_{k' \in K} \Big(\underbrace{\Pr[h(k) = h(k')]}_{\leq 1/m} \times 1 + \cancel{\Pr[h(k) \neq h(k')] \times 0} \Big)$$

$$\leq n/m.$$

Bilan sur le chaînage

Complexité

- Complexité espérée de chaque opération : $O(\alpha)$ où $\alpha = \frac{n}{m}$ est le *taux de remplissage*
- Si le taux est autour de 1 : $O(1)$ en moyenne
- Attention : l'espérance du pire cas n'est pas $O(\alpha)$!
  - $\mathbb{E}[\max_k \ell(k)] \neq \max_k \mathbb{E}[\ell(k)]$

La résolution par chaînage marche bien de manière *amortie*, mais certaines opérations peuvent être coûteuses

Pourquoi des listes chaînées ?

- Arbres binaires de recherche ou tas dans chaque case
  - Complexité moyenne en $O(\log \alpha)$
  - Complexité pire cas en $\max_k \log \ell(k)$
- Et pourquoi pas des tables de hachage ?

Éviter les collisions

Si la table est suffisamment grande, il n'y a pas de collision (avec bonne probabilité)

Lemme

Si $m = n^2$, et h est tirée uniformément dans un ensemble universel $\mathcal{H}$, alors la probabilité qu'il existe deux clefs $k_1 \neq k_2$ telles que $h(k_1) = h(k_2)$ est $\leq \frac{1}{2}$.

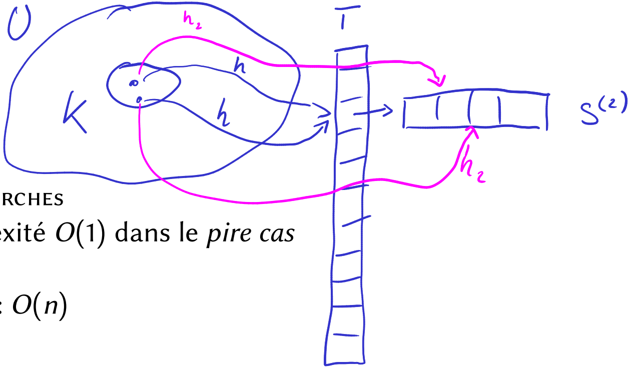
C = v.a. qui désigne le nb total de collisions

$$C = \sum_{\substack{k_1 < k_2 \\ k_1, k_2 \in K}} C_{k_1,k_2} \quad \text{où } C_{k_1,k_2} = \begin{cases} 1 & \text{si } h(k_1) = h(k_2) \\ 0 & \text{sinon} \end{cases}.$$

$$\mathbb{E}[C] = \sum_{k_1<k_2} \mathbb{E}[C_{k_1,k_2}] = \sum_{k_1<k_2} \Pr[h(k_1) = h(k_2)] \leq \sum_{k_1<k_2} 1/m = \frac{1}{m}\binom{n}{2} = \frac{n(n-1)}{2n^2} \leq 1/2$$

Inégalité de Markov : $\Pr[X \geqslant \lambda \mathbb{E}[X]] \leq 1/\lambda$ $\qquad \Pr[C \geqslant 1] \leq 1/2$.

Le hachage parfait



Objectif

- Un table de hachage *statique*
  - Insertion de n couples
  - puis uniquement des Recherches
- Chaque Recherche de complexité $O(1)$ dans le *pire cas*
- Taille totale de la table : $O(n)$
- Temps de création de la table : $O(n)$

Principes

- Une table principale T avec fonction de hachage h, de taille $m = n$
- Chaque case $T_{[i]}$ contient une table de hachage *secondaire* $S^{(i)}$
- La table $S^{(i)}$ est de taille m_i, avec fonction de hachage $h_i : U \to \{0, \dots, m_i - 1\}$
- Chaque case de $S^{(i)}$ ne peut contenir qu'un élément (pas de chaînage)
- Clef k en case $S^{(i)}_{[j]}$ où $i = h(k)$ et $j = h_i(k)$ $\to$ Recherche en $O(1)$

Création de la table

Algorithme

1. Choisir $m = n$ et tirer $h : U \to \{0, \ldots, m-1\}$ dans un ensemble universel $\mathcal{H}$
2. Calculer tous les *hachés* $h(k_i), 1 \leq i \leq n$
3. Pour $j = 0$ à $m - 1$:
4. $\quad m_j \leftarrow$ nombre de clefs k_i telles que $h(k_i) = j$
5. $\quad$ Créer une table $S^{(j)}$ de taille m_j^2
6. $\quad$ Tirer $h_j : U \to \{0, \ldots, m_j^2\}$ dans un ensemble universel
7. $\quad$ Insérer dans $S^{(j)}$ tous les couples (k_i, v_i) tq $h(k_i) = j$
8. $\quad$ En cas de collision, goto 5.

Lemme

L'espérance du temps de calcul de l'algorithme de création est $O(n)$

Taille totale de la table

Théorème

L'espérance de la taille totale d'une table de hachage parfaite est $\mathbb{E}\left[\sum_{j=0}^{n} m_j^2\right] \leq 2n$

Bilan sur le hachage parfait

Complexités

- INSERTION en complexité *amortie* $O(1) \to$ création complète de la table en temps $O(n)$
- RECHERCHE en temps $O(1)$ *à tous les coups*
- Mémoire nécessaire : $O(n) \to$ pas de perte de place

Au delà des tables statiques

- Comment gérer des INSERTIONS et RECHERCHES imbriquées ?
  - Même résultat !
- Comment gérer des suppressions ?
  - Plus subtile
  - Idées proches des tableaux dynamiques

Principe

Si la case pour insérer (k, v) est occupée, trouver une autre case !

Formellement

- m fonctions de hachage $h_1, \ldots, h_m$
  - 1^{er} essai : INSERTION en case $h_1(k)$
  - $2^{\text{ème}}$ essai : INSERTION en case $h_2(k)$
  - ...
  - $m^{\text{ème}}$ essai : INSERTION en cas $h_m(k)$
- Condition : pour tout k, $\{h_1(k), \ldots, h_m(k)\}$ est une *permutation* de $\{0, \ldots, m-1\}$

Algorithmes

- RECHERCHE : explorer $T_{[h_1(k)]}, T_{[h_2(k)]}, \ldots$
  - si on trouve $k \rightarrow$ gagné !
  - si on trouve une case vide $\rightarrow$ k n'est pas dans T
- INSERTION : explorer jusqu'à trouver une case vide

Constructions d'adressage ouvert

Construire les m fonctions à partie d'une (ou deux) fonctions de hachage

Quelques possibilités pratiques

- Sondage linéaire : $h_i(k) = (h(k) + i) \bmod m$
- Sondage quadratique : $h_i(k) = (h(k) + ai^2 + bi) \bmod m$ *(bien choisir a et b !)*
- Sondage binaire : $h_i(k) = h(k) \oplus i$ *(si $m = 2^\ell$)*
- Double hachage : $h_i(k) = (h^{(1)}(k) + ih^{(2)}(k)) \bmod m$ *(conditions sur $h^{(1)}$ et $h^{(2)}$)*
- …

Analyse de l'adressage ouvert

Hypothèse : pour tout k, $\{h_1(k), \ldots, h_m(k)\}$ est une permutation aléatoire

Théorème

Si le facteur de remplissage est $\alpha = n/m < 1$, l'espérance du nombre de cases visitées pour une RECHERCHE infructueuse est $\leq \frac{1}{1-\alpha}$.

Bilan sur l'adressage ouvert

Idée de principe

- Une seule table principale, un seul élément par case
- Si une case est occupée, aller ailleurs !
- Plusieurs solutions pour *aller ailleurs*

Complexité espérée (modèle aléatoire)

	$\alpha = \frac{1}{2}$	$\alpha = \frac{9}{10}$
Insertion ou Recherche infructueuse : $\frac{1}{1-\alpha}$	2	10
Recherche réussie : $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ (admis)	$\leq 1,387$	$\leq 2,559$

Pour aller plus loin : *hachage du coucou*

- Deux fonctions de hachage $h^{(1)}$ et $h^{(2)}$ *deux emplacements possibles par clef*
- Insertion de (k, v) :
  - Insertion en case $h^{(1)}(k)$
  - Si la case contenait (k', v'), on le déplace à son autre emplacement
  - Et récursivement…
- Et ça marche !

Conclusion sur la résolution des collisions

Les collisions sont inévitables !

Deux familles de résolutions

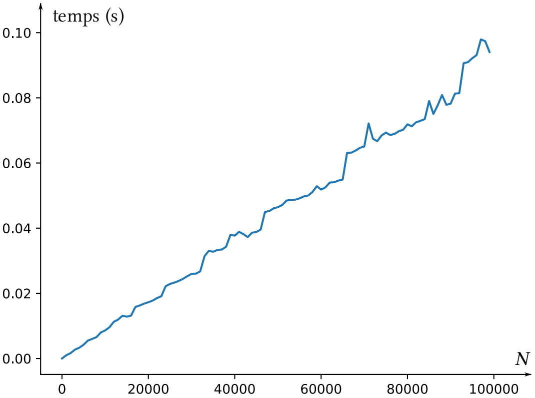
- Chaînage, hachage parfait, ...
  - Gérer les collisions en mettant plusieurs éléments par case
  - Complexité liée au nombre maximal d'éléments par case et à la structure de données
- Adressage ouvert
  - Gérer les collisions en cherchant une autre case libre
  - Complexité liée au nombre de cases à inspecter

$\rightarrow$ Dans les deux cas : complexité liée au nombre de collisions

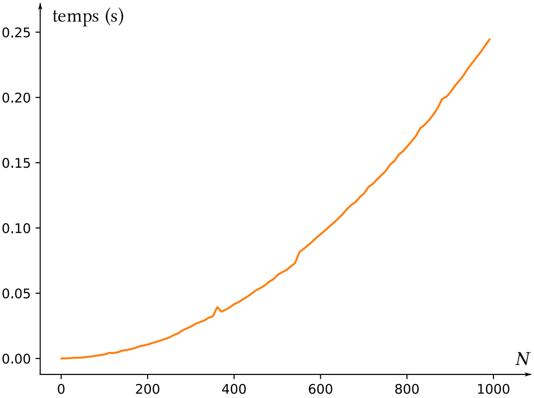
Cas des dictionnaires Python

- Fonction de hachage pas aléatoire ! $h(i) = i \bmod (2^{61} - 1)$ si i est un entier
- Résolution des collisions par adressage ouvert
  - Ordre de parcours des cases un peu complexe
- Solution théoriquement faible, à peu près correcte en pratique

Analyse des temps de calcul

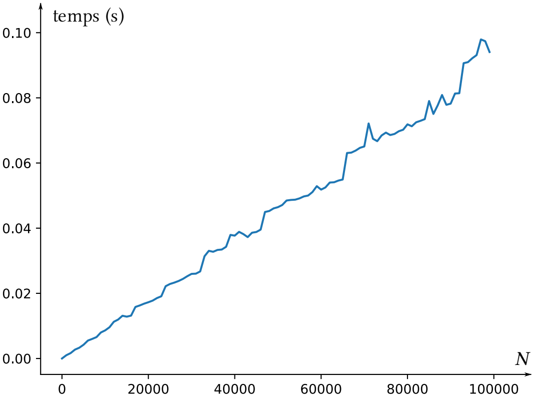



```python
d = {}
for i in range(N):
  d[randrange(2**61*N**2)] = i
```

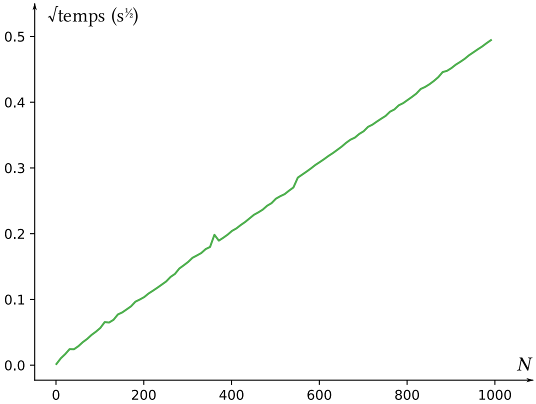



```python
d = {}
for i in range(N):
  d[(2**61-1)*randrange(N**2)] = i
```

Analyse des temps de calcul




```
d = {}
for i in range(N):
  d[randrange(2**61*N**2)] = i
```




```
d = {}
for i in range(N):
  d[(2**61-1)*randrange(N**2)] = i
```

Objectif

Rappel de la définition

Un ensemble $\mathcal{H}$ de fonctions de $\{0, \ldots, N-1\}$ dans $\{0, \ldots, m-1\}$ est universel si pour tout $k_1 \neq k_2$, $\Pr_{h\in\mathcal{H}}[h(k_1) = h(k_2)] \leq 1/m$.

Contraintes sur $\mathcal{H}$

- Suffisamment grand pour avoir une probabilité $\leq 1/m$
- Suffisamment petit pour savoir représenter $h \in \mathcal{H}$ avec une place raisonnable
- Suffisamment *simple* pour savoir tirer $h \in \mathcal{H}$ en temps raisonnable

$\mathcal{H}$ de taille polynomiale en N

- Nombre de couples de clefs possibles $\binom{N}{2} \to$ au moins autant de fonctions h
- Représentation d'une fonction h en $O(\log N)$ bits $\to$ similaire à une clef
- Tirage aléatoire en $O(\log N) \to$ équivalent au calcul de $h(k)$

Hachage multiplicatif

Définition

Soit $\mathcal{H}_p^{N,m} = \{h_{a,b} : 0 < a < p, 0 \leq b < p\}$ la famille de fonctions définies par

$$h_{a,b} : \begin{array}{ccc} \{0,\ldots,N-1\} & \to & \{0,\ldots,m-1\} \\ k & \mapsto & ((ak+b) \bmod p) \bmod m \end{array}$$

où p est un nombre premier $> N$

Représentation et tirage aléatoire

- Tirage aléatoire de $h_{a,b}$: tirage de $a \in \{1,\ldots,p-1\}$ et $b \in \{0,\ldots,p-1\}$
- Représentation de $h_{a,b}$: (a, b, p)
- Taille : $|\mathcal{H}_p^{N,m}| = p(p-1) > N^2$

Théorème

La famille $\mathcal{H}_p^{N,m}$ est universelle (pour tout N, m et $p \geq N$)

Outil : système linéaire *modulo p*

Lemme

Soit $k_1 \neq k_2$ et $u \neq v$ dans $\mathbb{Z}/p\mathbb{Z}$, alors il existe un unique couple $a, b \in \mathbb{Z}/p\mathbb{Z}$ tel que $u = ak_1 + b$ et $v = ak_2 + b$.

Preuve du théorème

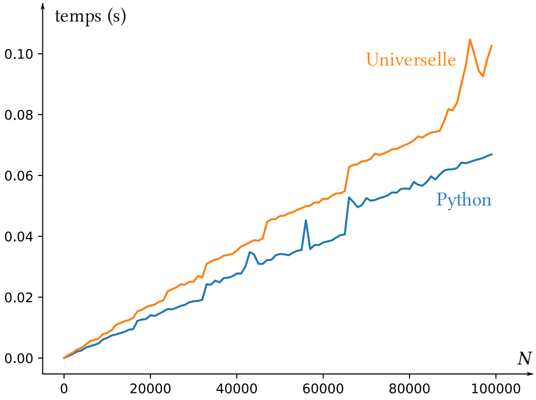
Théorème (réécrit)

Pour tout $k_1 \neq k_2$, $\Pr_{a,b}[h_{a,b}(k_1) = h_{a,b}(k_2)] \leq 1/m$

Bilan sur la famille universelle

Utilisation de la famille

- Création du dictionnaire : tirage aléatoire de a et b
- Complexité du calcul de $h_{a,b}(k) = ((ak + b) \bmod p) \bmod m$
  - Additions, multiplications, divisions d'entiers $\leq p^2$: $O(\log^2 p) = O(\log^2 N)$
  - Taille d'une clef $\to O(\log N)$



Autres familles universelles

- $h_a(k) = (ak \bmod 2^w) \operatorname{div} 2^{w-\ell}$ — quasi-universelle (*cf* TD)
- $h_{\vec{c}}(k) = ((\sum_i c_i k^i) \bmod p) \bmod m$ — fortement universelle

Conclusion sur les tables de hachage

Tables de hachage

- Structure de données très efficace, et très répandue
- Autres structures dérivées des tables de hachage (filtres de Bloom, etc.)
- Constructions pratiques inspirées de la théorie

Gestion des collisions

- Chaînage $\to$ complexité amortie $O(1)$ dans le modèle universel
- Hachage parfait $\to$ complexité pire cas $O(1)$ dans le modèle universel
- Adressage ouvert $\to$ complexité amortie $O(1)$ dans le modèle aléatoire
  - Difficile : même résultat dans le modèle (fortement-)universel

Construction de familles universelles

- $h_{a,b}(k) = (((ak + b) \bmod p) \bmod m)$ fournit une famille universelle
- Construction d'autres familles universelles
- Meilleures garanties : familles fortement universelles

Pour aller plus loin

Fonctions de hachage

- Utiles au delà des tables de hachage (empreinte numérique, etc.)
- Riche théorie, basée sur les probabilités
- Hachage d'autres objets (chaînes de caractères, graphes, ...)
- Autre type de fonctions de hachage : fonctions de hachage cryptographiques

Dans les langages de programmation

- Tables de hachages souvent proposées (dictionnaires)
- Fonctions de hachage non aléatoires
- Comportement souvent bon en pratique, mais possibles mauvaises surprises

Et en pratique

- Fonctions de hachages utilisées partout !
- Applications *critiques* $\to$ utilité de la théorie