- HAI503I: Algorithmique 4 -

Chap. 4 – Tables de hachage

L3 informatique
Université de Montpellier

Les dictionnaires Python

Comment implanter le type `dict` de Python ?

```
>>> NbPattes = {}                               # Dict vide
>>> NbPattes['Humain'] = 2                      # Ajout
>>> NbPattes['Mille␣Pattes'] = 999
>>> NbPattes['Araignee'] = 8
>>> NbPattes['Mille␣Pattes'] = 1000             # Modification
>>> 'Coccinelle' in NbPattes                    # Recherche
False
>>> NbPattes['Mille␣Pattes']
1000
```

La structure de données *dictionnaire*

Définition d'un dictionnaire

- Ensemble de couples (clef, valeur)
- Opérations disponibles :
  - **Création** d'un dictionnaire vide
  - **Insertion** d'un couple
  - Modification d'une valeur $\rightsquigarrow$ Ré-Insertion
  - **Recherche** d'une clef $\rightsquigarrow$ renvoie la valeur ou une erreur

Objectif

Les opérations RECHERCHE et INSERTION doivent être *rapides*

La structure de données *dictionnaire*

Définition d'un dictionnaire

- Ensemble de couples (clef, valeur)
- Opérations disponibles :
  - **Création** d'un dictionnaire vide
  - **Insertion** d'un couple
  - Modification d'une valeur $\rightsquigarrow$ Ré-Insertion
  - **Recherche** d'une clef $\rightsquigarrow$ renvoie la valeur ou une erreur

Objectif

Les opérations RECHERCHE et INSERTION doivent être *rapides*

Hypothèse simplificatrice

- Les clefs sont des entiers
- Théorie : toute donnée est codée en binaire $\rightsquigarrow$ interprétation comme un entier
- Pratique : codage ASCII, Unicode ou transformation quelconque...

Quelles solutions ?

Dictionnaire de n éléments, clef entre 0 et $N-1^{\star}$

Tableau

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

. $\star$ Dans l'exemple de départ : mots de 20 lettres sur un alphabet de taille 128 $\rightsquigarrow$ $N = 128^{20} = 2^{140} \sim 1.4 \times 10^{42}$...

Quelles solutions ?

Dictionnaire de n éléments, clef entre 0 et $N-1^{\star}$

Tableau

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

Liste chaînée

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

. $\star$ Dans l'exemple de départ : mots de 20 lettres sur un alphabet de taille 128 $\rightsquigarrow$ $N = 128^{20} = 2^{140} \sim 1.4 \times 10^{42}$...

Quelles solutions ?

Dictionnaire de n éléments, clef entre 0 et $N - 1^{\star}$

Tableau

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

Arbre binaire de recherche

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

Liste chaînée

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

. $\star$ Dans l'exemple de départ : mots de 20 lettres sur un alphabet de taille 128 $\rightsquigarrow$ $N = 128^{20} = 2^{140} \sim 1.4 \times 10^{42}$...

Quelles solutions ?

Dictionnaire de n éléments, clef entre 0 et $N-1^{\star}$

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

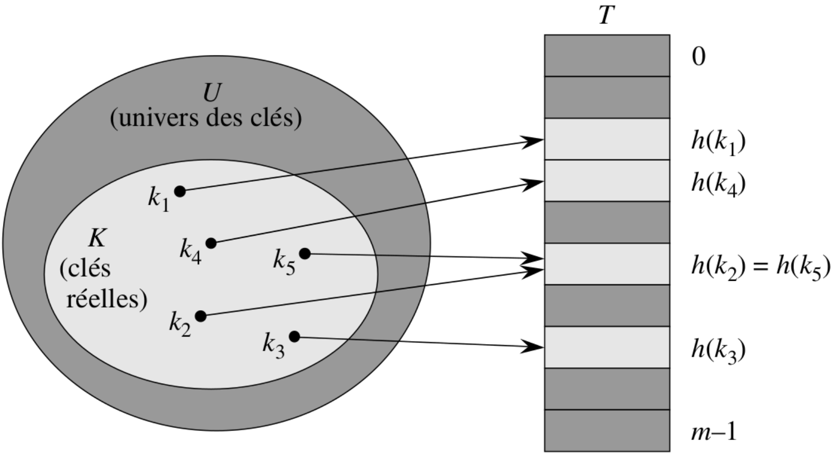
- Taille : n ✓
- CRÉATION : $O(1)$ ✓
- INSERTION : $O(h) \rightsquigarrow O(\log n)$ si équilibré 🙄
- RECHERCHE : $O(h) \rightsquigarrow O(\log n)$ si équilibré 🙄

Tas

- Taille : n ✓
- CRÉATION : $O(1)$ ✓
- INSERTION : $O(\log n)$ 🙄
- RECHERCHE : $O(\log n)$ 🙄

. $\star$ Dans l'exemple de départ : mots de 20 lettres sur un alphabet de taille 128 $\rightsquigarrow$ $N = 128^{20} = 2^{140} \sim 1.4 \times 10^{42}$...

Tables de hachage



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

Formalisation

Table de hachage

- **Clefs**
  - Univers U des clefs possibles : $U = \{0, \dots, N-1\}$
  - Clefs utilisées : $K \subset U$, de taille n
- **Table T de taille m**
  - Indices entre 0 et $m-1$
  - Une case peut être vide ou contenir une valeur, voire plusieurs
- **Fonction de hachage**
  - Fonction $h : U \to \{0, \dots, m-1\}$

Insertion

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

Utilisées de partout :

- En machine : cache, compilateur (variables), disque (table des inodes)
- Pour gérer des bases de données
- En cryptographie, pour la vérification d'intégrité d'un fichier (par exemple, *la fonction MD5*), pour la vérification de mots de passe...

Questions à résoudre

Caractéristiques

- Taille : $m \rightsquigarrow$ comment choisir m par rapport à n et N ?
- Création : on veut $O(m)$
  On ne peut pas explicitement fixer une valeur $h(k)$ pour chaque clef k de U (il y en a trop...), il va falloir une formule ou un algorithme pour calculer $h(k)$!
- Insertion : calcul de $h(k)$ puis insertion en case $h(k) \rightsquigarrow$ quelle complexité ?
- Recherche : calcul de $h(k)$ puis recherche dans la case $h(k) \rightsquigarrow$ quelle complexité ?

Questions à résoudre

Caractéristiques

- Taille : $m \rightsquigarrow$ comment choisir m par rapport à n et N ?
- CRÉATION : on veut $O(m)$
  On ne peut pas explicitement fixer une valeur $h(k)$ pour chaque clef k de U (il y en a trop...), il va falloir une formule ou un algorithme pour calculer $h(k)$!
- INSERTION : calcul de $h(k)$ puis insertion en case $h(k) \rightsquigarrow$ quelle complexité ?
- RECHERCHE : calcul de $h(k)$ puis recherche dans la case $h(k) \rightsquigarrow$ quelle complexité ?

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 ?

Problématique des fonctions de hachage

Contexte

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

Quelques exemples (qui ne marchent pas forcément bien...)

▷ $h(i) = 0$ (pour tout $i \in U$)
▷ $h(i) = i \mod m$
▷ $h(i) = a.i + b \mod m$ (pour a et b à choisir)
▷ $h(i) = 63.i^2 - 12.i + 7 \mod m$ (par exemple...)

Collisions évitables ?

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

Problématique des fonctions de hachage

Contexte

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

Quelques exemples (qui ne marchent pas forcément bien...)

- $\triangleright$ $h(i) = 0$ (pour tout $i \in U$)
- $\triangleright$ $h(i) = i \mod m$
- $\triangleright$ $h(i) = a.i + b \mod m$ (pour a et b à choisir)
- $\triangleright$ $h(i) = 63.i^2 - 12.i + 7 \mod m$ (par exemple...)

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

Tentative de choix de fonctions de hachage

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

Représentation de h

- Pour chaque $k \in U$, une valeur $h(k) \rightsquigarrow$ tableau H de taille N
- Tirage de $h \rightsquigarrow$ tirage uniforme et indépendant de chaque $h(k)$ dans $\{0, \ldots, m-1\}$

Avantage et inconvénient

- *Avantage* : on a bien pour tout $k_1 \neq k_2$, $\Pr[h(k_1) = h(k_2)] = 1/m$
- *Inconvénient* : **totalement irréaliste $\rightsquigarrow$ tableau de taille N ! ! !**

Modèles aléatoires des fonctions de hachage

Tentative de choix de fonctions de hachage

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

Représentation de h

- Pour chaque $k \in U$, une valeur $h(k) \rightsquigarrow$ tableau H de taille N
- Tirage de $h \rightsquigarrow$ tirage uniforme et indépendant de chaque $h(k)$ dans $\{0, \ldots, m-1\}$

Avantage et inconvénient

- *Avantage* : on a bien pour tout $k_1 \neq k_2$, $\Pr[h(k_1) = h(k_2)] = 1/m$
- *Inconvénient* : **totalement irréaliste** $\rightsquigarrow$ **tableau de taille** N **! ! !**

Nécessité d'un autre modèle

- Avec les mêmes (bonnes) propriétés probabilistes
- Mais, avec qui soit implémentable en temps raisonnable

Modèle universel des fonctions de hachage

Tentative 2 de choix de 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 $U = \{0, \ldots, N-1\}$ dans $\{0, \ldots, m-1\}$ est **universel** si pour tout $k_1 \neq k_2 \in U$, lorsqu'on tire uniformément h dans $\mathcal{H}$ on a $\Pr[h(k_1) = h(k_2)] \leq 1/m$.

Remarques

- Probabilité de collision $\leq$ probabilité dans le modèle aléatoire
- On *connait* des ensembles $\mathcal{H}$ universels réalistes *(cf Partie 3...)*

Ensemble universel *intéressant*

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

Collisions

Contexte

- Table T avec fonction de hachage $h : U = \{0, \dots, N-1\} \to \{0, \dots, m-1\}$ choisie dans un ensemble universel
- Ensemble de clefs $K \subseteq U$
- Il y a **collision** entre les clefs k_1 et k_2 si $h(k_1) = h(k_2)$

Remarque

- Si la table est suffisamment grande, avec bonne probabilité, il n'y aura pas de collision.
- Mais on ne peut pas être complétement sûr de les éviter...

Collisions

Contexte

- Table T avec fonction de hachage $h : U = \{0, \ldots, N-1\} \to \{0, \ldots, m-1\}$ choisie dans un ensemble universel
- Ensemble de clefs $K \subseteq U$
- Il y a **collision** entre les clefs k_1 et k_2 si $h(k_1) = h(k_2)$

Remarque

- Si la table est suffisamment grande, avec bonne probabilité, il n'y aura pas de collision.
- Mais on ne peut pas être complétement sûr de les éviter...

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 $\leq \frac{1}{2}$.

Collisions

Contexte

- Table T avec fonction de hachage $h : U = \{0, \ldots, N-1\} \to \{0, \ldots, m-1\}$ choisie dans un ensemble universel
- Ensemble de clefs $K \subseteq U$
- Il y a **collision** entre les clefs k_1 et k_2 si $h(k_1) = h(k_2)$

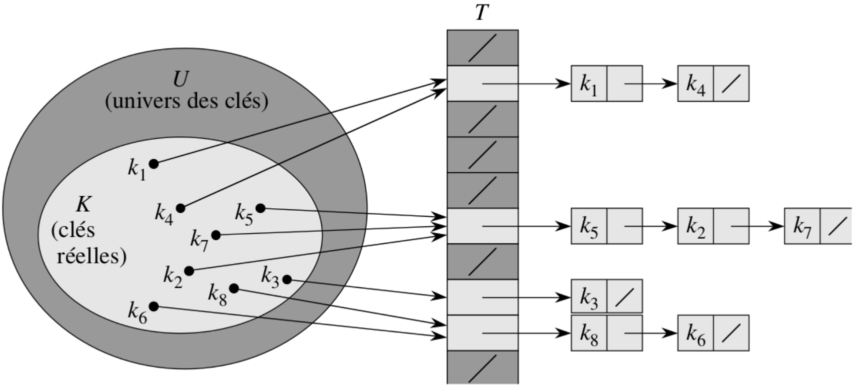
Remarque

- Si la table est suffisamment grande, avec bonne probabilité, il n'y aura pas de collision.
- Mais on ne peut pas être complétement sûr de les éviter...

Deux exemples de (familles de) solutions

- Plusieurs éléments dans une même case : *résolution par chaînage*
- 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 $h(k)$
  - Parcours de la liste contenue dans $T_{[h(k)]}$, retour de (k, v) ou NULL si k n'apparaît pas dans la liste.
  - Complexité : $O(\ell(k))$ où $\ell(k)$ est la taille de la liste $T_{[h(k)]}$
- INSERTION de (k, v) :
  - 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 $L = \max_{k \in K} \ell(k) \rightsquigarrow$ quelle taille maximale en moyenne ?

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 RECHERCHE et de INSERTION est en $O(n/m)$.

Complexité

- Complexité espérée : $O(\alpha)$ où $\alpha = \frac{n}{m}$ est le *taux de remplissage*
- Si le taux est autour de 1 : $O(1)$ en moyenne
- La résolution par chaînage marche bien en moyenne, mais certaines opérations peuvent être coûteuses (si on a une grande liste...)

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 ?

Principe

Idée

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 case $h_m(k)$
- Condition : pour tout k, $\{h_1(k), \ldots, h_m(k)\}$ est une **permutation** de $\{0, \ldots, m-1\}$

Algorithmes

- Recherche : explorer $T_{[h_1(k)]}$, $T_{[h_2(k)]}$, ...
  - si on trouve $k \rightsquigarrow$ gagné
  - si on trouve une case vide $\rightsquigarrow$ k n'est pas dans T
- Insertion : explorer jusqu'à trouver une case vide

Adressage ouvert : constructions et performance

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

Quelques possibilités pratiques

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

Hypothèse (théorique...)

Pour tout k, $\{h_1(k), \dots, h_m(k)\}$ est une permutation aléatoire

Théorème

Sous l'hypothèse précédente, si le facteur de remplissage est $\alpha = n/m < 1$, l'espérance du nombre de cases visitées pour une Recherche infructueuse ou une Insertion est $\leq \frac{1}{1-\alpha}$.

Bilan sur l'adressage ouvert

Principe de base

- Une seule table principale, un seul élément par case
- Si une case est occupée, *aller ailleurs*, plusieurs possibilités pour ça...

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)	$\leq 1,387$	$\leq 2,559$

Bilan sur l'adressage ouvert

Principe de base

- Une seule table principale, un seul élément par case
- Si une case est occupée, *aller ailleurs*, plusieurs possibilités pour ça...

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)	$\leq 1,387$	$\leq 2,559$

Un exemple supplémentaire : *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 déplace (k', v') dans la case $h^{2}(k')$
  - Et récursivement...
- Et ça marche !

Conclusion sur la résolution des collisions

Les collisions sont inévitables

Deux familles de résolutions vues ici

- **Chaînage :**
  - 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

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

Conclusion sur la résolution des collisions

Les collisions sont inévitables

Deux familles de résolutions vues ici

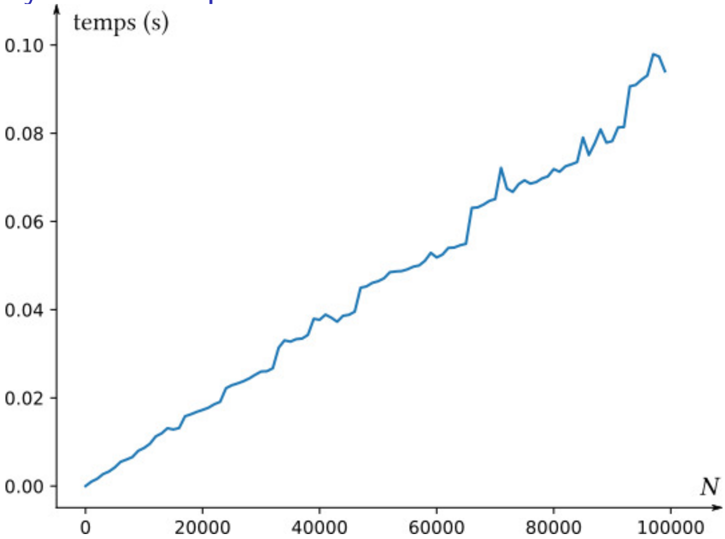
- **Chaînage :**
  - Gérer les collisions en mettant plusieurs éléments par case
  - Complexité liée au nombre maximal d'éléments par case et à la structure de données
- **Adressage ouvert :**
  - Gérer les collisions en cherchant une autre case libre
  - Complexité liée au nombre de cases à inspecter

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

Cas des dictionnaires Python

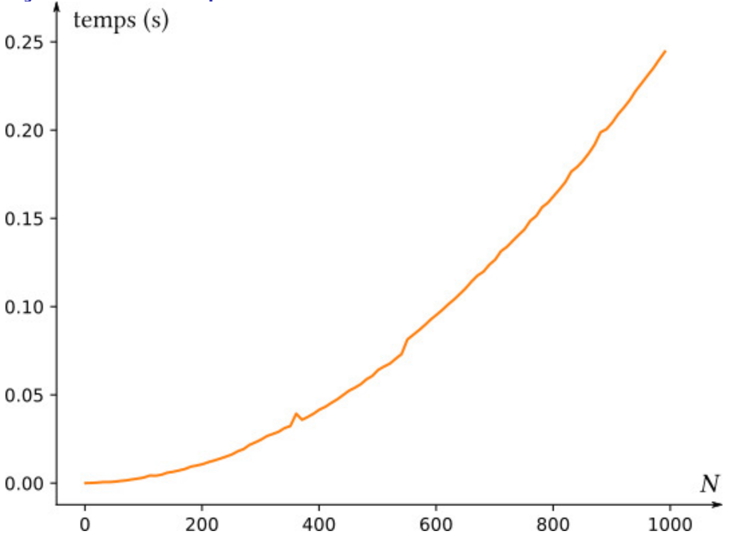
- Fonction de hachage pas aléatoire $\rightsquigarrow h(i) = i \bmod (2^{61} - 1)$
- Résolution des collisions par adressage ouvert
  - Ordre de parcours des cases un peu complexe
- Solution théoriquement faible, à peu près correcte en pratique

Analyse des temps de calcul



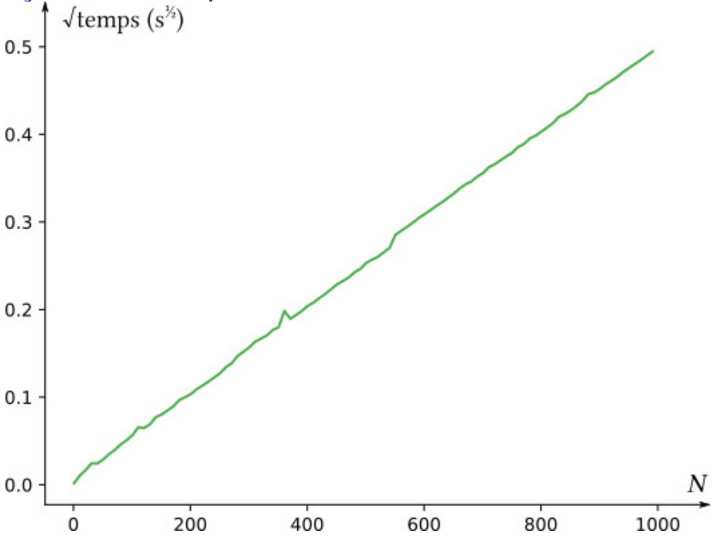

```
d={}
for i in range(N):
    d[randrange(2**61*N**2)]=i
```

Analyse des temps de calcul




```
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[(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 $k_1 \neq k_2$ fixés, $\Pr[h(k_1) = h(k_2)] \leq 1/m$.

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

On veut $\mathcal{H}$ **de taille polynomiale en** N

- Nombre de couples de clefs possibles $\binom{N}{2}$ $\rightsquigarrow$ on demande au moins autant de fonctions h, donc **on veut** $|\mathcal{H}| \geq \binom{N}{2}$
- **Représentation d'une fonction** h **en** $O(\log N)$ **bits** $\rightsquigarrow$ similaire à une clef
- **Tirage aléatoire en** $O(\log N)$ $\rightsquigarrow$ 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

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

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 : $|\mathcal{H}_p^{N,m}| = p(p-1) > N^2$

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

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

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 $p \geq N$)

Preuve du théorème

Rappel de la définition de $h_{a,b}$

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

$$h_{a,b} : \left| \begin{array}{ccc} \{0, \dots, N-1\} & \rightarrow & \{0, \dots, m-1\} \\ k & \rightarrow & ((ak+b) \bmod p) \bmod m \end{array} \right.$$

où p est un nombre premier $> N$

Lemme : système linéaire *modulo p*

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

Rappel de la définition de $h_{a,b}$

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

$$h_{a,b} : \left| \begin{array}{ccc} \{0,\dots,N-1\} & \to & \{0,\dots,m-1\} \\ k & \to & ((ak+b) \bmod p) \bmod m \end{array} \right.$$

où p est un nombre premier $> N$

Lemme : système linéaire *modulo p*

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 $v = ak_2 + b$.

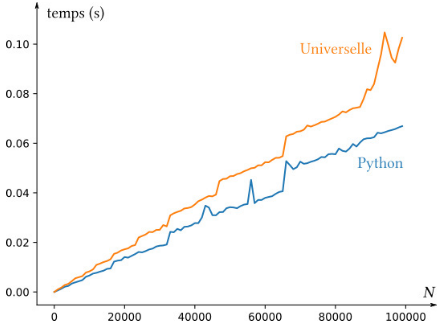
Théorème (réécrit)

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

Bilan sur la famille universelle

Utilisation de la famille

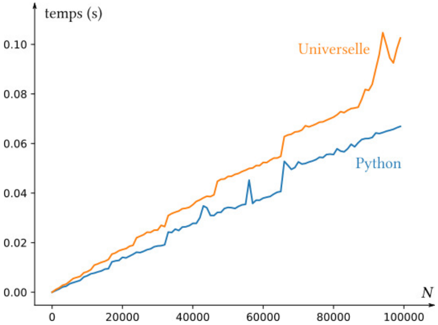
- CRÉATION du dictionnaire : tirage aléatoire de a et b
- Complexité du calcul de $h_{a,b}(k) = ((ak+b) \bmod p) \bmod m$
  - Additions, multiplications, divisions d'entiers $\leq p^2$: $O(\log^2 p) = O(\log^2 N)$
  - Taille d'une clef $\rightsquigarrow$ $O(\log N)$



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 $\rightsquigarrow$ $O(\log N)$



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 (ex : filtres de Bloom)
- Constructions pratiques inspirées de la théorie

Gestion des collisions

- Chaînage $\rightsquigarrow$ complexité amortie $O(1)$ dans le modèle universel
- Adressage ouvert $\rightsquigarrow$ complexité amortie $O(1)$ dans le modèle aléatoire
- D'autres méthodes existent. Ex. le hachage parfait$\rightsquigarrow$ complexité pire cas $O(1)$ dans le modèle universel

Construction de familles universelles

- $h_{a,b}(k) = (((ak + b) \bmod p) \bmod m)$ fournit une famille universelle
- D'autres familles existent...

Dans les langages de programmation

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