Algorithmique — 1. Structures de données

4. Tables de hachage

Bruno Grenet

https://membres-ljk.imag.fr/Bruno.Grenet/Algorithmique.html

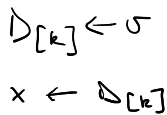
Université Grenoble Alpes – IM²AG
L3 Mathématiques et Informatique

Table des matières

Rappel : le TAD dictionnaire

Définition

- Ensemble de couples (clé, valeur)
- Opérations :
  - NVDICTIONNAIRE()
  - INSÉRER(D, k, v)
  - SUPPRIMER(D, k)
  - RECHERCHER(D, k)



dynamique

Hypothèse simplificatrice : les clés sont des entiers

- Théorie : toute donnée est codée en binaire → interprétation comme un entier
- Pratique : on se ramène à des entiers, mais pas forcément de cette façon

Quelles réalisations ?

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

	taille	nvDictionnaire	insérer	supprimer	rechercher
Tableau					
Liste chaînée					
ABR (si équilibré)					
Tableau dynamique (si trié)					

Quelles réalisations ?

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

	taille	nvDictionnaire	insérer	supprimer	rechercher
Tableau	N	$O(N)$	$O(1)$	$O(1)$	$O(1)$
Liste chaînée					
ABR (si équilibré)					
Tableau dynamique (si trié)					

Quelles réalisations ?

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

	taille	nvDictionnaire	insérer	supprimer	rechercher
Tableau	N	$O(N)$	$O(1)$	$O(1)$	$O(1)$
Liste chaînée	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
ABR (si équilibré)					
Tableau dynamique (si trié)					

Quelles réalisations ?

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

	taille	nvDictionnaire	insérer	supprimer	rechercher
Tableau	N	$O(N)$	$O(1)$	$O(1)$	$O(1)$
Liste chaînée	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
ABR	$O(n)$	$O(1)$	$O(h)$	$O(h)$	$O(h)$
(si équilibré)			$O(\log n)$	$O(\log n)$	$O(\log n)$
Tableau dynamique					
(si trié)					

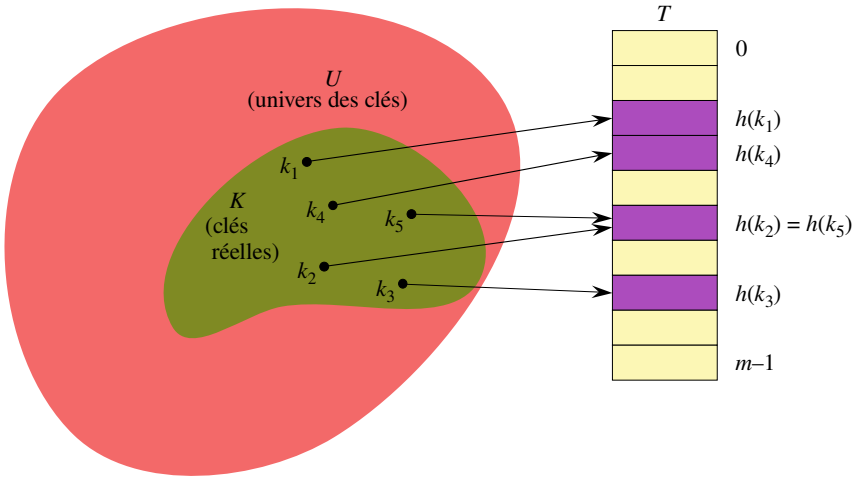
Quelles réalisations ?

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

	taille	nvDictionnaire	insérer	supprimer	rechercher
Tableau	N	$O(N)$	$O(1)$	$O(1)$	$O(1)$
Liste chaînée	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
ABR (si équilibré)	$O(n)$	$O(1)$	$O(h)$ $O(\log n)$	$O(h)$ $O(\log n)$	$O(h)$ $O(\log n)$
Tableau dynamique (si trié)	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$ $O(\log n)$

Table des matières

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. (modif.)

Formalisation

Clés

Univers des clés possibles : $U = \{0, \ldots, N-1\}$

Ensemble des clés utilisées : $K \subset U$, de taille n

Tableau de taille m

- Indices entre 0 et $m-1$
- Une case peut
  - être vide
  - contenir une, ou plusieurs, valeurs

Fonction de hachage

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

$\mathcal{T} = (T, h) \to$ couple (k, v) stocké en 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 ?

Complexités

- Taille : $m \rightarrow$ choix de m par rapport à n et N ?
- nvDictionnaire : $O(m)$
- insérer / supprimer / rechercher :
  1. calcul de $h(k)$
  2. insérer / supprimer / rechercher en case $h(k)$

$\rightarrow$ Coût du calcul de $h(k)$? Coût des opérations dans une case ?

Table des matières

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 clés 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 clés : si $n \leq m$?
  - Pour un ensemble de clés donné, il y a forcément une fonction h sans collision
  - Mais… on ne connaît pas les clés à l'avance !

Problématique

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

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

Modèle idéalisé des fonctions de hachage

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

Avantage et inconvénient

- Avantage : très bonnes propriétés probabilistes
  - Ex.: pour tout $k_1 \neq k_2$, $\Pr_h[h(k_1) = h(k_2)] = 1/m$
- Inconvénient : totalement **irréaliste** $\rightarrow$ comment tirer et stocker h ?

Représentation de h

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

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 $h : \{0, \ldots, N-1\} \to \{0, \ldots, m-1\}$ 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 idéalisé
- L'ensemble de toutes les fonctions est universel… mais irréaliste !
- On veut (et 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

Un exemple d'ensemble universel : le hachage multiplicatif

Définition

Soit $\mathcal{H}_p^{N,m} = \{h_{a,b} : 0 < a < p, 0 \leq b < p\}$ où $p > N$ est premier et

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

Efficacité

- Représentation de $h_{a,b}$: $(a,b,p) \to$ taille : $O(\log N)$ bits
- Tirage aléatoire de $h_{a,b}$: tirage de $a \in \{1,\dots,p-1\}$ et $b \in \{0,\dots,p-1\}$
- Calcul de $h_{a,b}(k)$ en $O(1)$ opérations sur les entiers

($O(\log N \log\log N)$ opérations sur les bits)

Théorème

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

Outil : système linéaire *modulo p*

Lemme

Soit $k_1 \neq k_2$ et $u \neq v$ dans $\{0, \dots, p-1\}$, alors il existe un unique couple (a, b) dans $\{0, \dots, p-1\}$ tel que $\begin{cases} u \equiv_p ak_1 + b \\ v \equiv_p ak_2 + b \end{cases}$ $(*)$

Preuve

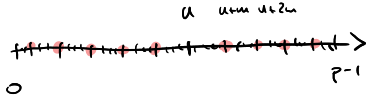
$(*) \iff \begin{cases} u \equiv_p ak_1 + b \\ u - v \equiv_p a(k_1 - k_2) \end{cases} \iff \begin{cases} u \equiv_p ak_1 + b \\ a \equiv_p (u-v)(k_1-k_2)^{-1} \end{cases} \iff \begin{cases} b \equiv_p u - (u-v)(k_1-k_2)^{-1} k_1 \\ a \equiv_p (u-v)(k_1-k_2)^{-1} \end{cases}$

Corollaire

Les ensembles $\{(a,b) : 0 \leq a < p \\ 0 \leq b < p\}$ et $\{(u,v) : 0 \leq u, v < p, u \neq v\}$

sont en bijection.

Preuve du théorème



$h_{a,b}(k) = \big((ak+b) \bmod p\big) \bmod m$

Théorème (réécrit)

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

Preuve

- Si $k_1 \neq k_2$, $(ak_1+b) \bmod p \neq (ak_2+b) \bmod p$
- Soit $\begin{cases} u = ak_1+b \bmod p \\ v = ak_2+b \bmod p \end{cases}$. On a $h_{a,b}(k_1) = h_{a,b}(k_2) \iff u \equiv_m v$
- Or il y a bijection entre $\{(a,b) : \begin{smallmatrix} 0<a<p \\ 0\leq b<p \end{smallmatrix}\}$ et $\{(u,v) : 0 \leq u,v < p, u \neq v\}$

  Donc $\Pr_{a,b}[h_{a,b}(k_1)=h_{a,b}(k_2)] = \Pr_{a,b}[u \equiv_m v] = \Pr_{u,v}[u \bmod m = v \bmod m]$
- Si u est fixé, $\#\{v \in \{0,\dots,p-1\} : u \neq v \text{ et } u \equiv_m v\} = \lfloor p/m \rfloor \leq \frac{p-1}{m}$

  $\Rightarrow \Pr_{u,v}[u \equiv_m v] \leq \frac{p-1}{m} \cdot \frac{1}{p-1} \leq 1/m$

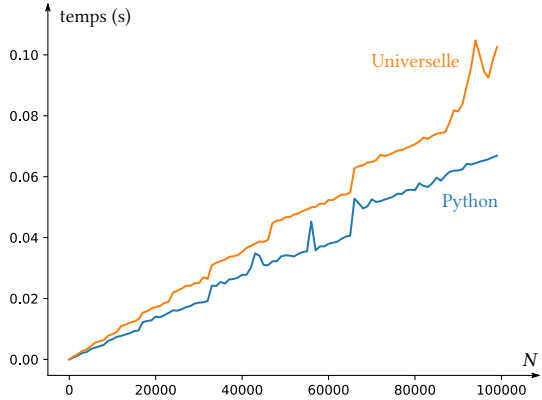
Bilan sur la famille universelle

Utilisation de la famille

- Création du dictionnaire
  - tirage aléatoire de a et b
  - tirage de $p > N$ premier
- Stockage : tableau + entiers a, b, p

Abus de langage

Fonction de hachage universelle :
Fonction tirée uniformément dans une famille universelle de fonctions de hachages



Autres familles universelles

- $h_a(k) = (ak \bmod 2^w) \operatorname{div} 2^{w-\ell}$
- $h_{\vec{c}}(k) = ((\sum_i c_i k^i) \bmod p) \bmod m$

quasi-universelle
fortement universelle

Table des matières

Problématique

Contexte

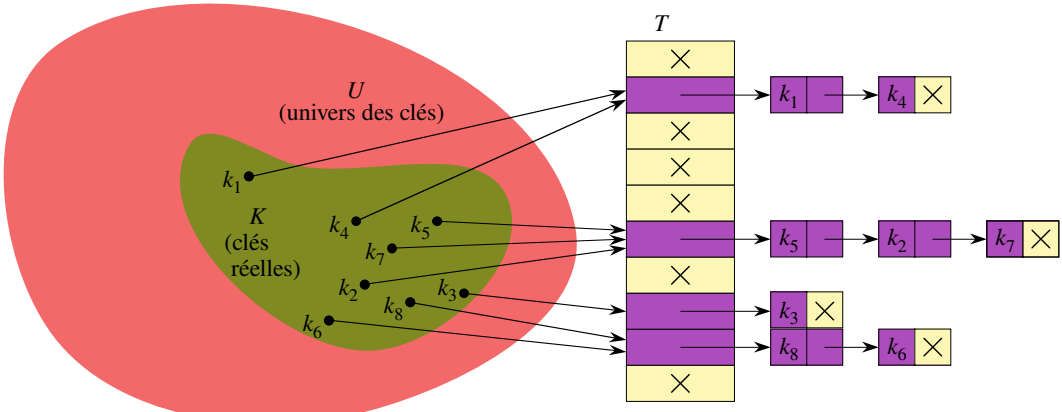
- Table $\mathcal{T}$ avec fonction de hachage $h : \{0, \ldots, N-1\} \to \{0, \ldots, m-1\}$
- Ensemble de clés K

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

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. (modif.)

Résolution par chaînage

Chaque case de T contient une liste chaînée

Algorithmes

RECHERCHER$(\mathcal{T}, k)$:
1. Calcul de $h(k)$
2. Rechercher k dans $T_{[h(k)]}$

INSÉRER$(\mathcal{T}, k, v)$:
1. Calcul de $h(k)$
2. Insérer (k, v) dans $T_{[h(k)]}$

SUPPRIMER$(\mathcal{T}, k)$:
1. Calcul de $h(k)$
2. Supprimer k de $T_{[h(k)]}$

- Complexités : $O(\ell(k))$ où $\ell(k)$ est la taille de la liste $T_{[h(k)]}$

Quelle efficacité ?

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

Efficacité de la résolution par chaînage

Théorème

Soit $\mathcal{T} = (T, h)$ où $\#T = m$ et h est universelle[1]. Si $\mathcal{T}$ contient n éléments et que les collisions sont résolues par chaînage, l'espérance de la complexité de l'insertion et de la recherche est $O(n/m)$.

Preuve Sans perte de généralité, on regarde une recherche infructueuse ($k \notin \mathcal{T}$)

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

↳ $\ell(k) = \#\{k' \in \mathcal{T} : h(k) = h(k')\}$

$$\mathbb{E}_h[\ell(k)] = \sum_{k' \in \mathcal{T}} \left(1 \times \underbrace{\mathbb{P}_r[h(k) = h(k')]}_{\leq 1/m} + 0 \times \mathbb{P}_r[h(k) \neq h(k')]\right) \leq n/m .$$

Espérance du coût de la recherche : $\mathbb{E}[\Theta(\ell(k))] = \Theta(n/m)$.

[1] « h tirée uniformément dans une famille universelle de fonctions de hachage »

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 $O(\alpha)$! $\quad \mathbb{E}[\max_k \ell(k)] \neq \max_k \mathbb{E}[\ell(k)]$

- La résolution par chaînage est efficace *en moyenne*
- Certaines opérations sont parfois coûteuses

Pourquoi des listes chaînées ?

- Chaque case contient un ensemble de (clé,valeur) $\rightarrow$ un dictionnaire par case !
  - Tas ou ABR équilibré $\rightarrow$ complexité moyenne $O(\log \alpha)$
  - Et pourquoi pas des tables de hachage ? *hachage parfait*
- Intérêt des listes chaînées :
  - Simplicité
  - Suffisant si $\alpha = O(1)$

L'adressage ouvert

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

Formellement

- m fonctions de hachage $h_1, \ldots, h_m$
- INSÉRER$(\mathcal{T}, k, v)$:
  1. $i \leftarrow 1$
  2. Tant que $i \leq m$ et $T_{[h_i(k)]}$ n'est pas vide : $i \leftarrow i + 1$
  3. Si $i \leq m$: $T_{[h_i(k)]} \leftarrow (k, v)$
  4. Sinon : erreur « la table est pleine »
- Condition : pour tout k, $\{h_1(k), \ldots, h_m(k)\}$ est une *permutation* de $\{0, \ldots, m-1\}$

Avantage

- Tableau T standard, pas de liste chaînée / ABR / etc. dans les cases

Constructions d'adressage ouvert

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 : $h_i(k) = (h(k) + ai^2 + bi) \bmod m$ *(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 $h^{(2)}$)*
- …

Algorithmes

- RECHERCHER : explorer $T_{[h_1(k)]}$, $T_{[h_2(k)]}$, …
  - si on trouve $k \rightarrow$ gagné !
  - si on trouve une case vide $\rightarrow$ k n'est pas dans T
- INSÉRER : explorer jusqu'à trouver une case vide
- SUPPRIMER : RECHERCHER puis vider la case + complications

Analyse de l'adressage ouvert

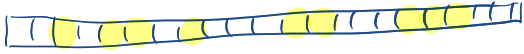
Hypothèse : pour tout k, $\{h_1(k), \ldots, h_m(k)\}$ est une permutation aléatoire

Théorème

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

Preuve

× Hypothèse ⇒ on tire des indices aléatoirement jusqu'à tomber sur une case vide



× $\mathbb{E}_{m,n} = \mathbb{E}\left[\#\text{ cases visitées}\right]$ α

$$\mathbb{E}_{m,n} = 1 \times \frac{m-n}{m} + \left(1 - \frac{m-n}{m}\right)\left(1 + \mathbb{E}_{m-1,n-1}\right) = 1 + \alpha\,\mathbb{E}_{m-1,n-1}$$

⟶ Par récurrence, on montre $\mathbb{E}_{m,n} \leq \frac{1}{1-\alpha}$.

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 idéalisé)

	$\alpha = \frac{1}{2}$	$\alpha = \frac{9}{10}$
INSÉRER ou RECHERCHER infructueux : $\frac{1}{1-\alpha}$	2	10
RECHERCHER réussi : $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ (admis)	$\leq 1,39$	$\leq 2,56$

Pour aller plus loin : *hachage du coucou*

- Deux fonctions de hachage $h^{(1)}$ et $h^{(2)}$ *deux emplacements possibles par clé*
- INSÉRER$(\mathcal{T}, k, v)$:
  - Mettre (k, v) en case $T_{[h^{(1)}(k)]}$
  - 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 !

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

Dans les deux cas

- Complexité liée au nombre de collisions $\rightarrow$ à minimiser !
- Si table trop remplie : nombreuses collisions
  $\rightarrow$ combiner avec des tableaux dynamiques

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

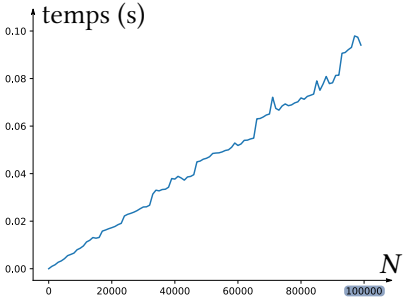
- Résultats présentés :
  - Chaînage : complexité espérée $O(1)$ dans le modèle universel
  - Adressage ouvert : complexité espérée $O(1)$ dans le modèle idéalisé
- D'autres résultats :
  - Hachage parfait : complexité pire cas $O(1)$ dans le modèle universel
  - Adressage ouvert : même résultat dans le modèle (fortement-)universel *difficile*

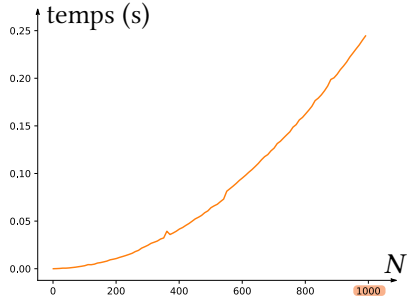
Construction de familles universelles

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

Exemple 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



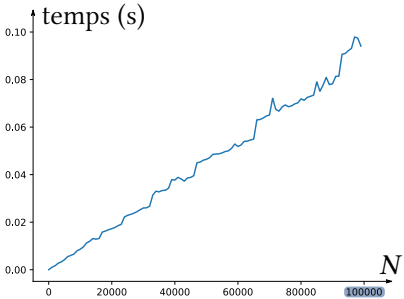



```
d = {}
for i in range(N):
  d[randrange(2**61*N**2)] = i
```

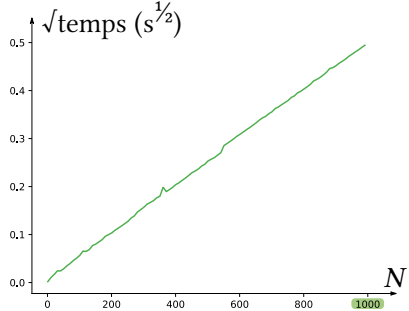
```
d = {}
for i in range(N):
  d[(2**61-1)*randrange(N**2)] = i
```

Exemple 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




```
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
```

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* $\rightarrow$ utilité de la théorie

Conclusion sur les structures de données

Types abstraits de données

- Version algorithmique des types
- Construction hiérarchique
- Dynamique versus statique

Les TAD étudiés

- Tableau, liste, arbre binaire
- File, Pile, File de priorité (tas)
- Tableau dynamique
- Dictionnaire : ABR et tables de hachage

Quelques concepts rencontrés

- Complexité *amortie*
- Structure probabiliste et *espérance* de complexité