Module EA4 – Éléments d'Algorithmique II

Outils pour l'analyse des algorithmes

Dominique Poulalhon

`dominique.poulalhon@irif.fr`

Université Paris Diderot

L2 Informatique & Math-Info

Année universitaire 2019-2020

Le hachage

III. Résolution des collisions par sondage ou hachage « par adressage ouvert »

Résolution par sondage, ou par adressage ouvert (ou « hachage fermé » (*sic*))

Cette méthode consiste à utiliser directement la table (l'espace d'adressage) pour stocker les éléments, d'où l'appellation courante « par adressage ouvert ». D'autres personnes considèrent au contraire que le fait de se cantonner à l'espace d'adressage est une limite, d'où l'appellation « hachage fermé »... Vu l'ambiguïté des terminologies « hachage ouvert » vs « hachage fermé », je déconseille fortement leur usage.

Principe : si une cellule est occupée, essayer ailleurs !

Problème : comment retrouver ensuite cet « ailleurs » ?

Si `T[h(cle)]` est occupée, on va *sonder* successivement d'autres cases (selon une règle fixée à l'avance, bien sûr), jusqu'à en trouver une libre : pour la clé k, au i^e essai, on teste la case d'indice $h(k, i)$ pour une certaine fonction h fixée. Une éventuelle recherche ultérieure se passera de manière similaire, en sondant la même succession de cases avant de trouver l'élément cherché (ou une case vide si l'élément cherché n'est pas ou plus dans la table).

L'exemple le plus simple est le sondage linéaire : si `T[h(cle)]` est occupée, tester successivement `T[h(cle) + 1]`, `T[h(cle) + 2]`, *etc.* (circulairement, en repartant au début de la table si toutes les cases au-delà de `T[h(cle)]` sont occupées).

Propriétés du hachage avec résolution des collisions par adressage ouvert

Lemme

Le taux de remplissage α d'une table à adressage ouvert est au plus 1.

(forcément, puisque chaque case accueille au plus un élément)

Lemme

Pour tester toutes les cases sans redite, il faut que, pour chaque clé k, la fonction $i \mapsto h(k, i)$ soit une permutation. Dans le cas contraire, les sondages pourraient échouer à trouver une case libre bien qu'il en reste.

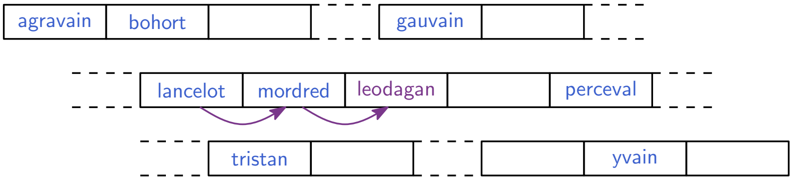
(c'est bien le cas pour le sondage linéaire)

Hachage (par sondage) linéaire

Si `T[h(cle)]` est occupée, tester itérativement `T[h(cle) + 1]`, `T[h(cle) + 2]`, *etc.*
C'est-à-dire que le i^e sondage va tester la case d'indice :

$$h(k, i) = (h(k) + i) \mod m$$

Exemple :



$$h(\text{leodagan}) = 12 = h(\text{lancelot})$$

Hachage (par sondage) linéaire

Cela peut s'écrire :

```
# version "dictionnaire" ie couples (cle, valeur)
def ajouter(table, cle, valeur) :
  for i in range(h(cle), len(table)) :
    if table[i] == None or table[i][0] == cle : break
  else : # si on atteint la fin, on recommence au début
    for i in range(h(cle)) :
      if table[i] == None or table[i][0] == cle : break
  table[i] = (cle, valeur)
```

ou de manière équivalente :

```
def ajouter(table, cle, valeur) :
  k, m = h(cle), len(table)
  for i in range(m) :
    if table[(k+i)%m] == None or table[(k+i)%m][0] == cle :
      break
  table[(k+i)%m] = (cle, valeur)
```

Hachage (par sondage) linéaire

Première version des autres opérations :

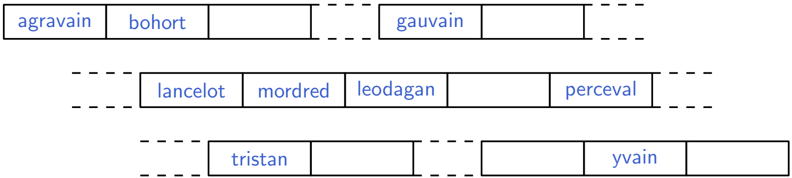
attention, tel quel c'est faux !!

```
def chercher(table, cle) :
  for i in range(h(cle), len(table)) :
    if table[i] == None : return None
    # on s'arrête à la première case vide trouvée :
    # échec de la recherche
    if table[i][0] == cle : return table[i][1]
  # le cas échéant, on recommence au début
  ...

def supprimer(table, cle) : ## (attention, tel quel c'est faux)
  for i in range(h(cle), len(table)) :
    if table[i] == None : return
    if table[i][0] == cle :
      table[i] = None
      return
  # le cas échéant, on recommence au début
  ...
```

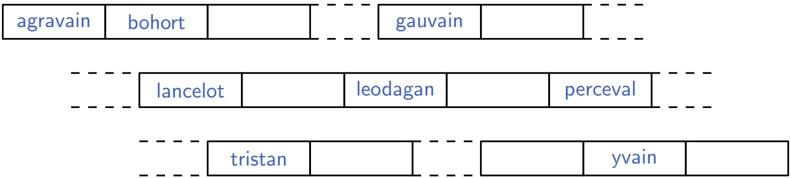
Hachage (par sondage) linéaire

Pourquoi est-ce faux ? Eh bien par exemple, si on supprime `mordred` avant de chercher `leodagan`…



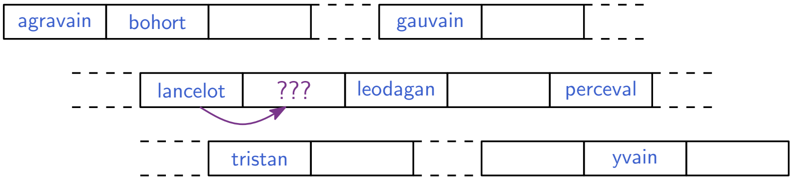
Hachage (par sondage) linéaire

Pourquoi est-ce faux ? Eh bien par exemple, si on supprime `mordred` avant de chercher `leodagan`…



Hachage (par sondage) linéaire

Pourquoi est-ce faux ? Eh bien par exemple, si on supprime `mordred` avant de chercher `leodagan`...



GLOUPS !!!

Hachage (par sondage) linéaire

Lorsqu'un élément est retiré de la table, il est important de laisser une marque pour que les recherches ultérieures tiennent compte du fait que la case a un jour été occupée (ce qui a pu provoquer la poursuite des sondages lors d'une insertion).

```
def supprimer(table, cle) :
  for i in range(h(cle), len(table)) :
    if table[i] == None : return
    if table[i][0] == cle :
      # la case n'est pas vidée, mais libérée
      table[i][1] = None
      return
  # le cas échéant, on recommence au début
  ...
```

(j'ai choisi de coder les cases vraiment vides par `None`, et les cases libérées par (`cle`, `None`) où `cle` est la clé ayant un jour occupé la case. Tout autre type de marque peut faire l'affaire, mais il faut différencier les cases vides depuis toujours et les cases libérées)

Hachage (par sondage) linéaire

Cela modifie donc un peu la recherche :

```
def chercher(table, cle) :
  for i in range(h(cle), len(table)) :
    if table[i] == None : return None
    if table[i][0] == cle : return table[i][1]
  # le cas échéant, on recommence au début
  ...
```

mais aussi l'ajout :

```
def ajouter(table, cle, valeur) :
  for i in range(h(cle), len(table)) :
    if table[i] == None or table[i][1] == None :
    # pas tout à fait suffisant (pb si cle est déjà dans table)
      table[i] = (cle, valeur)
      return
  # le cas échéant, on recommence au début
  ...
```

Complexité de la résolution par adressage ouvert

Hypothèse de hachage uniforme (forte) : pour une clé aléatoire, chacune des $m!$ permutations a la même probabilité $\frac{1}{m!}$ d'apparaître comme suite de sondages.

Théorème

dans une table à adressage ouvert, taux de remplissage $\alpha < 1$, et hachage supposé uniforme, le nombre moyen de sondages pour une recherche infructueuse est au plus $\frac{1}{1-\alpha}$.

(preuve à suivre)

Théorème (*admis*)

sous les mêmes hypothèses, le nombre moyen de sondages pour une recherche réussie est au plus $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$.

Donc sous ces hypothèses, à taux de remplissage fixe, les accès ont un coût moyen constant.

Complexité de la résolution par adressage ouvert

Théorème

dans une table à adressage ouvert, taux de remplissage $\alpha < 1$, et hachage supposé uniforme, le nombre moyen de sondages pour une recherche infructueuse est au plus $\frac{1}{1-\alpha}$

Démonstration

Lors d'une recherche infructueuse, on sonde successivement des cases occupées avant de s'arrêter sur une case vide. Notons S *le nombre de sondages nécessaires (qui est une variable aléatoire), et considérons la probabilité que* $S > k$.

$S > 1$ *signifie que la première case sondée est l'une des* n *cases occupées parmi* m, *donc* $\mathbb{P}(S > 1) = \frac{n}{m}$.

$S > 2$ *signifie que la première case sondée est occupée, et que la deuxième, distincte de la première, est l'une des* $n-1$ *autres cases occupées, parmi* $m-1$, *donc* $\mathbb{P}(S > 2) = \frac{n}{m} \cdot \frac{n-1}{m-1}$.

Plus généralement, on obtient $\mathbb{P}(S > k) = \dfrac{n}{m} \cdot \dfrac{n-1}{m-1} \cdots \dfrac{n-k-1}{m-k-1}$.

On remarque que $\frac{n}{m} \geqslant \frac{n-i}{m-i}$ *car* $n(m-i) \geqslant m(n-i)$ *dès que* $n \leqslant m$, *ce qui est le cas.*

Donc $\mathbb{P}(S > k) \geqslant \alpha^k$.

Pour obtenir l'espérance (i.e. la valeur moyenne) de S, *on se sert ensuite de la formule*

$$\mathbb{E}(S) = \sum_{k=1}^{n} k \cdot \mathbb{P}(S = k) = \sum_{k=1}^{n} \mathbb{P}(S \geqslant k) = \sum_{k=0}^{n} \mathbb{P}(S > k),$$

qu'on peut majorer par la somme de la série géométrique $\sum \alpha^k = \dfrac{1}{1-\alpha}$.

Problème

Le sondage linéaire permet seulement m séquences de sondage différentes... donc on est très très loin de l'hypothèse de hachage uniforme !

Constatation

À l'expérience, on observe un phénomène de clusterisation qui diminue rapidement les performances du hachage : les éléments s'agglutinent en gros amas, rendant les accès dans les zones concernées très lents puisqu'ils nécessitent souvent de parcourir une grande partie de l'amas.

Idée

Utiliser *deux* fonctions de hachage h_1 et h_2, et sonder successivement

$$h(k, i) = h_1(k) + i \cdot h_2(k) \mod m$$

utiliser *deux* fonctions de hachage h_1 et h_2, et sonder successivement

$$h(k, i) = h_1(k) + i \cdot h_2(k) \mod m$$

Lemme

la suite $i \mapsto h(k, i)$ *est une permutation si et seulement si :*

$h_2(k)$ *est premier avec* m

Comment assurer cette propriété ?

- avec m premier et $h_2(k) < m$ quelconque
  — *parfait si* n *est connu à l'avance (et donc* m *aussi)*
- avec $m = 2^p$ et $h_2(k)$ impair
  — *si des redimensionnements sont nécessaires*

on obtient alors $\Theta(m^2)$ séquences de sondage...
c'est encore loin de $m!$, mais nettement meilleur que m