

INF 431



B. WERNER



Hachage et Arbres

17 février 2010



Déjà vu ?

En INF421 (Olivier Bournez)

Entrevu un TD1

Le chapitre du Poly INF421 est excellent.

But

Table d'associations :

Nom+Prénom \mapsto numéro de sécu

Nom du gateau \mapsto fichier où se trouve la recette

...

fonction partielle

entrée quelconque

Le Tableau

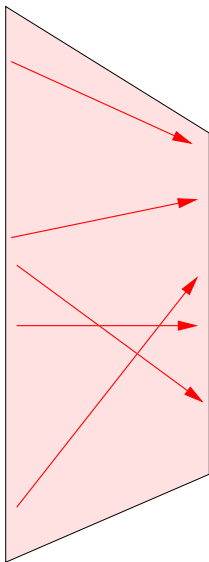
Gilles Dowek

Philippe Baptiste

Olivier Bournez

.

Benjamin Werner



0 →

1 →

.

.

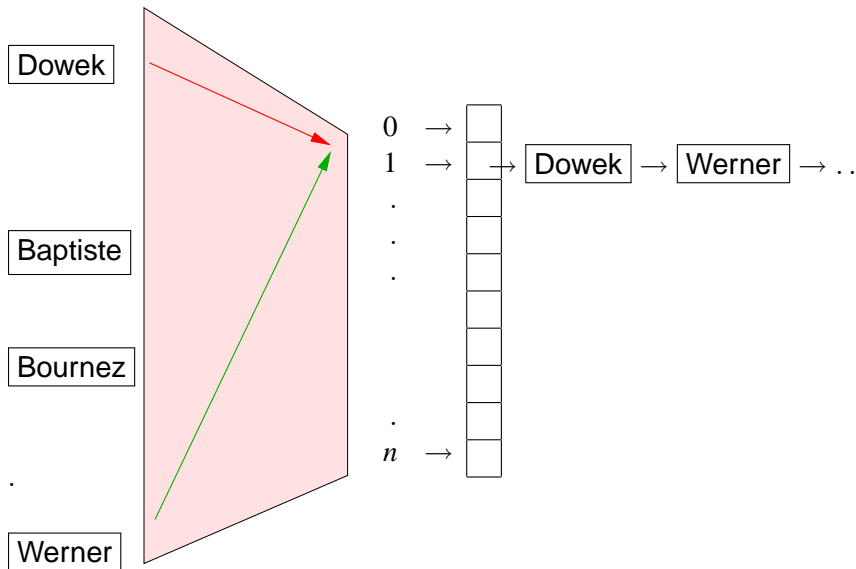
.

.

n →



Collisions



On sait gérer les collisions, mais ça prend du temps.

Qualité du Hachage

Une bonne fonction de hachage doit :

- Avoir des valeurs uniformément réparties sur $[0 \dots n]$
- Être discontinue : Dupont \neq Dupond
- Limiter les collisions
- Être calculable rapidement

Problème de recherche

Variantes :

- Checksums
- Hachage cryptographique

En Java

- Tout objet à une méthode `int hashCode()` et `equals`
- `HashMap` instancie l'interface `Map`

La "clé anglaise" du programmeur.

Mais attention. . .

Ne pas en mettre partout !

(ça prend de la place)

Aussi :

Il faut définir `hashCode` (sauf pour `String`, `Integer`...)

Ne pas oublier `equals`

```
public class Ex {
    public int a,b;
    public Ex(int x, int y){
        a = x; b = y; }

    public static void main(String[] args) {
        Ex e1 = new Ex(3,4);
        Ex e2 = new Ex(3,4);
        System.out.println(e1.hashCode());
        System.out.println(e2.hashCode());
        System.out.println(e2.equals(e1)); }
}
```

donne (par exemple) :

10891203

6718604

false

Dernier point

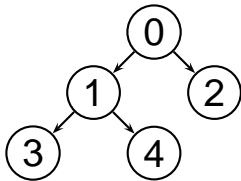
La taille du tableau

- Peut être choisie au départ
- agrandie à la main `rehash`
- ou automatiquement à partir d'un taux de remplissage fixé

Les arbres

Les Arbres

On connaît :



- Une structure mathématique : **simple**
- Des détails informatiques : **plus compliqués**

Mathématique ?

On programme par récursion

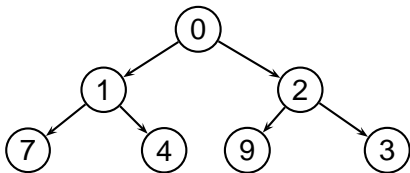
On raisonne par récurrence :

- cas de base : c'est vrai pour les feuilles
- hypothèse de récurrence : c'est vrai pour les sous arbres
- ce qu'il faut montrer : alors c'est vrai pour l'arbre

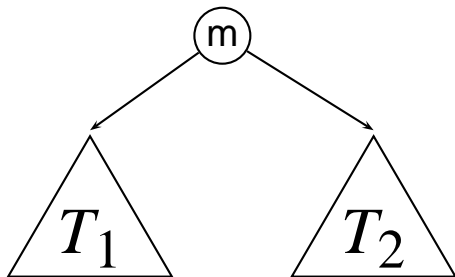
Exemple : tri par tas, sans tableau

Rappel : Un tas partiellement ordonné

Les fils sont plus grands que leur père



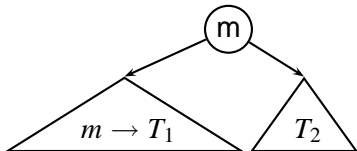
Insertion dans un tas



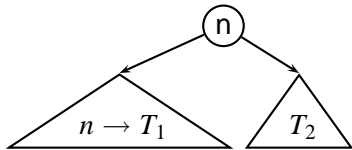
On veut insérer n

On sait insérer dans T_1 et dans T_2 .

Si $m < n$



Si $n \leq m$

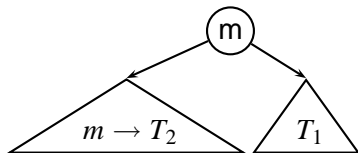


Complexité : hauteur de l'arbre

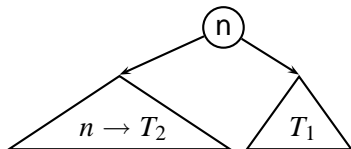
Ah zut. . .

Astuce

Si $m < n$



Si $n \leq m$



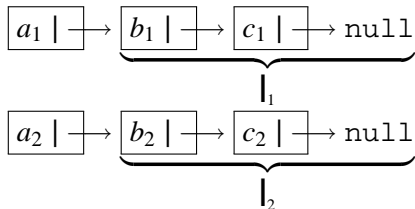
On peut montrer que les arbres restent équilibrés

- Le sous-arbre gauche a 1 ou 0 élément de plus que le sous-arbre droit.
- Les longueurs des différents chemins jusqu'aux feuilles diffèrent au plus de 1.

Cet **invariant** reste préservé par l'insertion.

Insertion en $\Theta(N)$

Fusionner des listes triées



On sait fusionner les sous-listes

Si $a_1 < a_2$ on rend $a_1 \rightarrow \text{merge}(l_1, a_2 \rightarrow l_2)$

Si $a_2 \leq a_1$ on rend $a_2 \rightarrow \text{merge}(a_1 \rightarrow l_1, l_2)$

Complexité : $\Theta(N)$

Trier des listes

- Transformer la liste en tas
- Transformer le tas en liste triée

Un version efficace du tri par insertion $\Theta(n \cdot \log(n))$