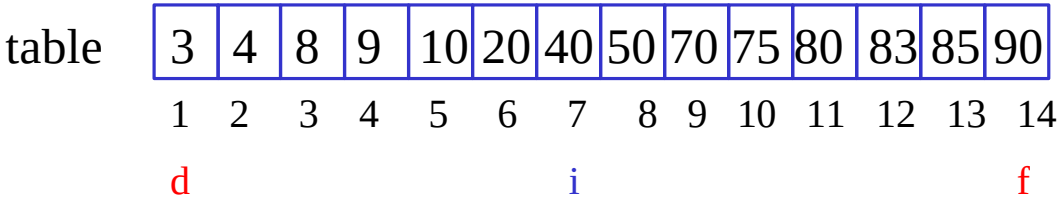
Méthodes de hachage

1. Introduction
2. Hachage ouvert
3. Hachage fermé
4. Implémentation des fonctions

Recherche dichotomique

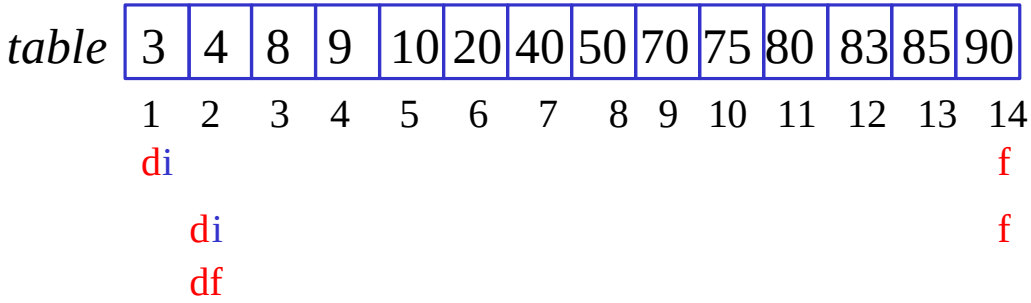


4 ?

```
fonction ELEMENT (x, table, d, f ) ;
début
    si (d = f) alors
        si (x = table [d ]) alors retour (vrai)
        sinon retour (faux)
    sinon {
        i ← ⌊(d+f ) / 2⌋ ;
        si (x > table [i]) alors
            retour (ELEMENT (x, table, i+1, f ))
        sinon retour (ELEMENT (x, table, d, i))
    }
fin
```

Temps (ELEMENT sur *table* [1 … *n*]) = O (log *n*)

Recherche par interpolation



4 ?

fonction ELEMENT (*x*, *table*, *d*, *f*) ;
début
si (*d* = *f*) **alors**
si (*x* = *table* [*d*] **alors retour** (vrai)
sinon retour (faux)
sinon {

$$i \neg d + \left\lfloor \frac{x - table[d]}{table[f] - table[d]} \right\rfloor * (f - d);$$

si (*x* > *table* [*i*] **alors**
retour (ELEMENT (*x*, *table*, *i*+1, *f*))
sinon retour (ELEMENT (*x*, *table*, *d*, *i*))
}
fin

Idée : Établir une relation entre un élément et l'adresse
à laquelle il est rangé en mémoire

Théorème : Si les éléments de *table* [1 … *n*] et *x* sont choisis uniformément dans un intervalle [*a*,*b*], le temps moyen d'exécution de la recherche par interpolation est O (log log *n*)

Hachage

Type abstrait « dictionnaire »

Ensembles avec les opérations principales

ELEMENT (x, A)
AJOUTER (x, A)
ENLEVER (x, A)

} temps constant en moyenne

Implémentation non adapté au classement.

Table de hachage

table dont les indices sont dans [0 .. B-1]

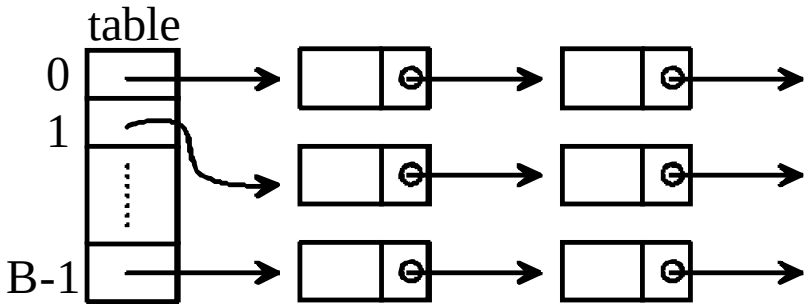
Fonction de hachage

h : éléments $\rightarrow$ [0 .. B-1] non injective en général

Résolution des collisions

Hachage ouvert : avec listes, triées ou non.
Hachage fermé : linéaire, quadratique, aléatoire, uniforme, double, ...

Hachage ouvert



- Liens explicites
- Taille variable

```
fonction h (x : mot) : indice ;
début
        somme : = 0 ;
        pour i ¬ 1 à longueurmaxi faire
            somme ¬ somme + ord (x [i]) ;
        retour (somme mod B) ;
fin
```

```
int hash (char * s) {
    char * p ;
    unsigned  h = 0, g ;
    for (p = s ;  * p ;   p++) {
         h = (h<< 4) + (* p) ;
         if (g = h  &  0xf0000000) {
            h =  h ^ (g >> 24) ;
            h =  h ^ g ;
         }
    }
    return (h % PRIME) ;
}
```

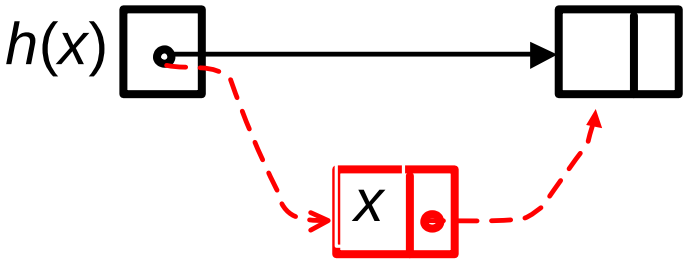
Voir Aho, Sethi, Ullman, *Compilers,* Addison-Wesley, 1986

```
        const B = {constante ad hoc} ;
        type  liste = ↑ cellule ;
              cellule = record
                    elt : élément ;
                    suivant : liste
              end  ;
        dictionnaire = array [0 .. B-1] of  liste ;

fonction  VIDER  () : dictionnaire ;
début
        pour i ¬  0 à B-1  faire A [ i ] ¬ NULL ;
        retour A ;
fin

fonction  ELEMENT (x élément,  A dictionnaire) : booléen ;
début   p ¬  A [ h(x) ] ;
        tantque  p ≠ NULL  faire
              si p->elt = x  alors retour  vrai
              sinon p ¬  p->suivant ;
        retour  faux ;
fin
```

fonction AJOUTER (*x* élément, *A* dictionnaire) ;
début
    AJOUTERLISTE (*x*, *h*(*x*)) ;
    retour *A* ;
fin



fonction AJOUTER (*x* élément, *A* dictionnaire) ;
début
    si non ELEMENT (*x*, *A*) **alors {**
        i ¬ *h* (*x*) ;
        p ¬ *A* [*i*] ;
        allouer *A* [*i*] ;
        A [*i*]->*elt* ¬ *x* ;
        A [*i*]->*suivant* ¬ *p* ;
    }
    retour *A* ;
fin

Temps des opérations

Hachage ouvert

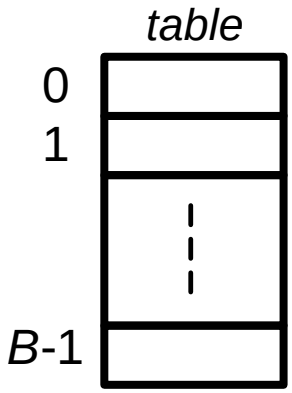
- initialisation (VIDER) : O (*B*)
- ajout : temps constant (après test d 'appartenance)
- appartenance
- suppression

$$O(1+\frac{n}{B})$$

si les événements "*h* (*x*) = *i*" sont équiprobables

Création d'une table de *n* éléments : $O(n(1+\frac{n}{B}))$

Hachage fermé



- liens implicites : gain de place
- taille limitée
- ré-allocation en cas de débordement

Re-hachage

$h_0(x) = h(x), h_1(x), \ldots h_i(x), \ldots$

où $h_i(x)$ dépend généralement de x et de i

Suppression d 'un élément

distinction entre « vide » et « disponible »

facteur de charge : n / B où n = nombre d 'éléments

Hachage linéaire

Re-hachage : $h_i(x) = (h(x) + i)$ **mod** B

0	FORWARD
1	disponible
2	THEN
3	vide
4	vide
5	FOR
B-1= 6	TO

EXEMPLE

$h(x) = (\text{ord}(c) - \text{ord}(\text{'a'})) \bmod B$
où c première lettre de x.

AJOUTER ('BEGIN')
AJOUTER ('FOR')
AJOUTER ('FUNCTION')
AJOUTER ('FORWARD')
AJOUTER ('THEN')
ENLEVER ('FUNCTION')
ENLEVER ('BEGIN')
AJOUTER ('TO')

```
            const        B = {constante ad hoc} ;
                       vide =  {constantes particulières  } ;
                 disponible = {distinctes des éléments   } ;
                              {mais de même type         } ;
            type  dictionnaire = array  [0… B-1]  of  éléments ;

fonction  VIDER  () : dictionnaire ;
début
        pour  i  ¬  0  à  B-1  faire
                A  [ i]  ¬  vide ;
        retour A ;
fin

fonction  POSITION (x  élément,  A  dictionnaire) : indice ;
/* calcule la seule position possible où ajouter x dans A */
début  i  ¬  0 ;
  tantque  (i < B)  et  A [h_i (x)] ≠ x  et  A [h_i (x)] ≠ vide  et  A [h_i (x)] ≠ disponible
        faire  i  ¬  i + 1 ;
  retour (h_i (x)) ;
fin
```

```
fonction  POSITIONE (x élément,  A  dictionnaire) : indice ;
début  hi ¬ h (x) ; dernier ¬ (hi + B-1)  mod  B ;
        tantque  hi ≠ dernier  et   (A [hi] Ï {x, vide})  faire
                hi ¬ (hi + 1)  mod  B  ;
        retour  (hi)
fin

fonction  ELEMENT (x  élément,  A  dictionnaire) : boolean ;
début
        si  A [POSITIONE (x, A)] = x  retour  vrai ;
        sinon  retour  faux ;
fin

fonction ENLEVER (x  élément,  A  dictionnaire) : dictionnaire ;
début
        i ¬ POSITIONE (x, A) ;
        si A [ i ] =  x  alors  A [ i ] ¬  disponible ;
        retour A ;
fin
```

```
fonction  POSITIONA (x  élément,  A dictionnaire) : indice ;
début  hi ¬ h (x) ;  dernier := (hi + B-1)  mod  B ;
        tantque  hi ≠ dernier  et  (A [hi] Ï {x, vide, disponible})  faire
                hi ¬ (hi + 1)  mod  B  ;
        retour  hi ;
fin

fonction  AJOUTER  (x  élément,  A dictionnaire) : dictionnaire ;
début
        i  ¬  POSITIONA (x, A) ;
        si  A [ i ] Î {vide, disponible}  alors
                A [ i ]  ¬  x ;
                retour A ;
        sinon si A [ i ]  ≠ x  alors
                erreur ( " table pleine " ) ;
fin
```

Hachage quadratique

Re-hachage : $h_i(x) = (h(x) + i^2)$ **mod** B

fonction POSITION (*x* élément, *A* dictionnaire) : indice ;
début
    hi ¬ *h* (*x*) ; *inc* ¬ 1 ;
    tantque (*inc* < *B*) **et** (*A* [*hi*] Ï {*x*, vide, ? }) **faire**
        hi ¬ (*hi* + *inc*) **mod** *B* ;
        inc ¬ *inc* + 2 ;
    retour *hi* ;
fin

• seule la moitié de la table est examinée par re-hachage
→ utiliser la suite :
    h (*x*), *h* (*x*) + 1, *h* (*x*) - 1, *h* (*x*) + 4, *h* (*x*) - 4, …
    avec *B* premier.

Hachage double

Re-hachage : $h_i(x) = (h(x) + i\,g(x))$ **mod** B

- B premier et $1 \leq g(x) \leq B - 1$
- ou B premier avec chacun des $g(x)$

pour examen de toute la table par re-hachage.

fonction POSITION (*x* élément, *A* dictionnaire) : indice ;
début
    hi ¬ *h* (*x*) ;
    inc ¬ *g* (*x*) ;
    dernier ¬ (*hi* + (*B*-1)* *inc*) **mod** *B* ;
    tantque (*hi* ≠ *dernier*) **et** (*A* [*hi*] Ï {*x*, vide, ? }) **faire**
        hi ¬ (*hi* + *inc*) **mod** *B* ;
    retour *hi* ;
fin

Hachage aléatoire

Re-hachage **:** $h_i(x) = (h(x) + d_i)$ **mod** B,
$d_1, d_2, \ldots, d_{B-1}$ permutation aléatoire de (1,..., B-1)

Génération des d_i par « décalage ».

- choix de $k \in (1,\ldots, B-1)$

- $d_i + 1 = \begin{cases} 2 \,.\, d_i & \text{si } 2 \,.\, d_i \leq B-1 \\ (2 \,.\, d_i - B) \oplus k & \text{sinon} \end{cases}$

Exemple $B = 8$ $k = 3 = 11_2$

$d1 = 5_{10} = 101_2$
$d2 = 1_{10} = 001_2$
$d3 = 2_{10} = 010_2$
$d4 = 4_{10} = 100_2$
$d5 = 3_{10} = 011_2$
$d6 = 6_{10} = 110_2$
$d7 = 7_{10} = 111_2$

Temps des opérations

Hachage fermé (aléatoire)

table contenant n éléments

- initialisation (VIDER) : O (B)
- probabilité d'avoir i comparaisons :

$$\frac{n}{b} \cdot \frac{n-1}{B-1} \cdot \ldots \frac{n-i+1}{B-i+1} \cong \left(\frac{n}{B}\right)^i$$

- coût d'un ajout réussi

$$\leq 1 + \sum_{i=1}^{\infty} \left(\frac{n}{B}\right)^i = \frac{1}{1-n/B}$$

- création d'une table à n éléments ($n \leq B$)

C_n = coût moyen d'un ajout

$$C_n = \frac{1}{n} \sum_{k=0}^{n-1} \frac{1}{1-k/B} \sim \frac{1}{n/B} \cdot \log_e \frac{1}{1-n/B} - 1/B$$

n/B	50 %	80 %	90 %
C_n	1,39	2,01	2,56

Tableau associatif

F tableau associatif : nombre fini d'indices de type quelconque
⇔ *F* fonction de domaine fini
⇔ *F* représentable par l'ensemble
E = { (*x*, *F* [*x*]) / *x* Î domaine de *F* }

Opérations

- INIT (*F*) rendre *F* [*x*] non défini pour tout *x*
- DEFINIR (*F*, *x*, *y*) poser *F* [*x*] = *y*
- CALCULER (*F*, *x*) = { *y* si *F* [*x*] = *y* ; nul sinon

Implémentation

- représenter *E* par hachage sur le domaine de *F*

TRADUCTEUR

Traducteur mot-à-mot