Tables de hachage

Algorithmique et structures de données, 2024-2025

P. Albuquerque (B410) et O. Malaspinas (A401), ISC, HEPIA

2025-02-21

En partie inspirés des supports de cours de P. Albuquerque

Les tables de hachage

Tableau vs Table

Tableau

- Chaque élément (ou valeur) est lié à un indice (la case du tableau).

```
annuaire tab[2] = {
    "+41 22 123 45 67", "+41 22 234 56 78", ...
};
tab[1] == "+41 22 123 45 67";
```

Table

- Chaque élément (ou valeur) est lié à une clé.

```
annuaire tab = {
//  Clé   ,     Valeur
    "Paul",     "+41 22 123 45 67",
    "Orestis", "+41 22 234 56 78",
};
tab["Paul"]    == "+41 22 123 45 67";
tab["Orestis"] == "+41 22 234 56 78";
```

Table

Définition

Structure de données abstraite où chaque *valeur* (ou élément) est associée à une *clé* (ou argument).

On parle de paires *clé-valeur* (*key-value pairs*).

Donnez des exemples de telles paires

Table

Définition

Structure de données abstraite où chaque *valeur* (ou élément) est associée à une *clé* (ou argument).

On parle de paires *clé-valeur* (*key-value pairs*).

Donnez des exemples de telles paires

- Annuaire (nom-téléphone),
- Catalogue (objet-prix),
- Table de valeur fonctions (nombre-nombre),
- Index (nombre-page)
- ...

Table

Opérations principales sur les tables

- Insertion d'élément (`insert(clé, valeur)`), insère la paire `clé-valeur`
- Consultation (`get(clé)`), retourne la `valeur` correspondant à `clé`
- Suppression (`remove(clé)`), supprime la paire `clé-valeur`

Structure de données / implémentation

Efficacité dépend de différents paramètres:

- taille (nombre de clé-valeurs maximal),
- fréquence d'utilisation (insertion, consultation, suppression),
- données triées/non-triées,
- ...

Consultation séquentielle (`sequential_get`)

Séquentielle

- table représentée par un (petit) tableau ou liste chaînée,
- types: `key_t` et `value_t` quelconques, et `key_value_t`

```
typedef struct {
    key_t key;
    value_t value;
} key_value_t;
```

- on recherche l'existence de la clé séquentiellement dans le tableau, on retourne la valeur.

Consultation séquentielle (`sequential_get`)

Implémentation? Une idée?

Consultation séquentielle (`sequential_get`)

Implémentation? Une idée?

```
bool sequential_get(int n, key_value_t table[n], key_t key,
    value_t *value)
{
    int pos = n - 1;
    while (pos >= 0) {
        if (key ==  table[pos].key) {
            *value = table[pos].value;
            return true;
        }
        pos--;
    }
    return false;
}
```

Consultation séquentielle (`sequential_get`)

Implémentation? Une idée?

```
bool sequential_get(int n, key_value_t table[n], key_t key,
    value_t *value)
{
    int pos = n - 1;
    while (pos >= 0) {
        if (key ==  table[pos].key) {
            *value = table[pos].value;
            return true;
        }
        pos--;
    }
    return false;
}
```

Inconvénient?

Consultation séquentielle (`sequential_get`)

Exercice: implémenter la même fonction avec une liste chaînée

Poster le résultat sur matrix.

Consultation dichotomique (`binary_get`)

Dichotomique

- table représentée par un (petit) tableau trié par les clés,
- types: `key_t` et `value_t` quelconques, et `key_value_t`
- on recherche l'existence de la clé par dichotomie dans le tableau, on retourne la valeur,
- les clés possèdent la notion d'ordre (`<`, `>`, `=` sont définis).

Consultation dichotomique (`binary_get`)

Implémentation? Une idée?

Consultation dichotomique (`binary_get`)

Implémentation? Une idée?

```
bool binary_get1(int n, key_value_t table[n], key_t key, value_t *value) {
    int top = n - 1, bottom = 0;
    while (top > bottom) {
        int middle = (top + bottom) / 2;
        if (key > table[middle].key) {
            bottom  = middle+1;
        } else {
            top = middle;
        }
    }
    if (key == table[top].key) {
        *value = table[top].value;
        return true;
    } else {
        return false;
    }
}
```

Consultation dichotomique (`binary_get`)

Autre implémentation

```
bool binary_get2(int n, key_value_t table[n], key_t key, value_t *value) {
    int top = n - 1, bottom = 0;
    while (true) {
        int middle = (top + bottom) / 2;
        if (key > table[middle].key) {
            bottom  = middle + 1;
        } else if (key < table[middle].key) {
            top = middle;
        } else {
            *value = table[middle].value;
            return true;
        }
        if (top < bottom) {
             break;
        }
    }
    return false;
}
```

Quelle est la différence avec le code précédent?

Transformation de clé (hashing)

Problématique: Numéro AVS (13 chiffres)

- Format: 106.3123.8492.13

```
Numéro AVS    | Nom
0000000000000 | -------
...           | ...
1063123849213 | Paul
...           | ...
3066713878328 | Orestis
...           | ...
9999999999999 | -------
```

Quelle est la clé? Quelle est la valeur?

Transformation de clé (hashing)

Problématique: Numéro AVS (13 chiffres)

- Format: 106.3123.8492.13

```
Numéro AVS    | Nom
0000000000000 | -------
...           | ...
1063123849213 | Paul
...           | ...
3066713878328 | Orestis
...           | ...
9999999999999 | -------
```

Quelle est la clé? Quelle est la valeur?

- Clé: Numéro AVS, Valeur: Nom.

Nombre de clés? Nombre de citoyens? Rapport?

Transformation de clé (hashing)

Problématique: Numéro AVS (13 chiffres)

- Format: 106.3123.8492.13

```
Numéro AVS    | Nom
0000000000000 | -------
...           | ...
1063123849213 | Paul
...           | ...
3066713878328 | Orestis
...           | ...
9999999999999 | -------
```

Quelle est la clé? Quelle est la valeur?

- Clé: Numéro AVS, Valeur: Nom.

Nombre de clés? Nombre de citoyens? Rapport?

- 10^{13} clés, 10^{7} citoyens, 10^{-5} ($10^{-3}\%$ de la table est occupée) $\Rightarrow$ *inefficace*.
- Pire: 10^{13} entrées ne rentre pas dans la mémoire d'un ordinateur.

Transformation de clé (hashing)

Problématique 2: Identificateurs d'un programme

- Format: 8 caractères (simplification)

```
Identificateur | Adresse
aaaaaaaa       | -------
...            | ...
resultat       | 3aeff
compteur       | 4fedc
...            | ...
zzzzzzzz       | -------
```

Quelle est la clé? Quelle est la valeur?

Transformation de clé (hashing)

Problématique 2: Identificateurs d'un programme

- Format: 8 caractères (simplification)

```
Identificateur | Adresse
aaaaaaaa       | -------
...            | ...
resultat       | 3aeff
compteur       | 4fedc
...            | ...
zzzzzzzz       | -------
```

Quelle est la clé? Quelle est la valeur?

- Clé: Identificateur, Valeur: Adresse.

Nombre de clés? Nombre d'identificateur d'un programme? Rapport?

Transformation de clé (hashing)

Problématique 2: Identificateurs d'un programme

- Format: 8 caractères (simplification)

```
Identificateur | Adresse
aaaaaaaa       | -------
...            | ...
resultat       | 3aeff
compteur       | 4fedc
...            | ...
zzzzzzzz       | -------
```

Quelle est la clé? Quelle est la valeur?

- Clé: Identificateur, Valeur: Adresse.

Nombre de clés? Nombre d'identificateur d'un programme? Rapport?

- $26^8 \sim 2 \cdot 10^{11}$ clés, 2000 identificateurs, 10^{-8} ($10^{-6}\%$ de la table est occupée) $\Rightarrow$ *un peu inefficace*.

Fonctions de transformation de clé (hash functions)

- La table est représentée avec un tableau.
- La taille du tableau est beaucoup plus petit que le nombre de clés.
- On produit un indice du tableau à partir d'une clé:

$$h(key) = n, \quad n \in \mathbb{N}.$$

  En français: on transforme `key` en nombre entier qui sera l'indice dans le tableau correspondant à `key`.

La fonction de hash

- La taille du domaine des clés est beaucoup plus grand que le domaine des indices.
- Plusieurs indices peuvent correspondre à la **même clé**:
  - Il faut traiter les **collisions**.
- L'ensemble des indices doit être plus petit ou égal à la taille de la table.

Une bonne fonction de hash

- Distribue uniformément les clés sur l'ensemble des indices.

Fonctions de transformation de clés: exemples

Méthode par troncature

$$h : [0, 9999] \rightarrow [0, 9]$$

$h(key) =$ troisième chiffre du nombre.

```
Key  | Index
0003 | 0
1123 | 2 \
1234 | 3  |-> collision.
1224 | 2 /
1264 | 6
```

Quelle est la taille de la table?

Fonctions de transformation de clés: exemples

Méthode par troncature

$$h : [0, 9999] \rightarrow [0, 9]$$

$h(key) =$ troisième chiffre du nombre.

```
Key  | Index
0003 | 0
1123 | 2 \
1234 | 3  |-> collision.
1224 | 2 /
1264 | 6
```

Quelle est la taille de la table?

C'est bien dix oui.

Fonctions de transformation de clés: exemples

Méthode par découpage

Taille de l'index: 3 chiffres.

```
key = 321 991 24 ->  321
                      991
                    + 24
                    ----
                    1336 -> index = 336
```

Devinez l'algorithme?

Fonctions de transformation de clés: exemples

Méthode par découpage

Taille de l'index: 3 chiffres.

```
key = 321 991 24 ->  321
                      991
                    + 24
                    ----
                    1336 -> index = 336
```

Devinez l'algorithme?

On part de la gauche:

1. On découpe la clé en tranche de longueur égale à celle de l'index.
2. On somme les nombres obtenus.
3. On tronque à la longueur de l'index.

Fonctions de transformation de clés: exemples

Méthode multiplicative

Taille de l'index: 2 chiffres.

`key = 5486 -> key^2 = 30096196 -> index = 96`

On prend le carré de la clé et on garde les chiffres du milieu du résultat.

Fonctions de transformation de clés: exemples

Méthode par division modulo

Taille de l'index: `N` chiffres.

`h(key) = key % N.`

Quelle doit être la taille de la table?

Fonctions de transformation de clés: exemples

Méthode par division modulo

Taille de l'index: `N` chiffres.

`h(key) = key % N.`

Quelle doit être la taille de la table?

Oui comme vous le pensiez au moins `N`.

Traitement des collisions

La collision

`key1 != key2, h(key1) == h(key2)`

Traitement (une idée?)

Traitement des collisions

La collision

`key1 != key2, h(key1) == h(key2)`

Traitement (une idée?)

- La première clé occupe la place prévue dans le tableau.
- La deuxième (troisième, etc.) est placée ailleurs de façon **déterministe**.

Dans ce qui suit la taille de la table est `table_size`.

La méthode séquentielle

Comment ça marche?

- Quand l'index est déjà occupé on regarde sur la position suivante, jusqu'à en trouver une libre.

```
index = h(key);
while (table[index].state == OCCUPIED && table[index].key != key) {
   index = (index + 1) % table_size; // attention à pas dépasser
}
table[index].key = key;
table[index].state = OCCUPIED;
```

Problème?

La méthode séquentielle

Comment ça marche?

- Quand l'index est déjà occupé on regarde sur la position suivante, jusqu'à en trouver une libre.

```
index = h(key);
while (table[index].state == OCCUPIED && table[index].key != key) {
   index = (index + 1) % table_size; // attention à pas dépasser
}
table[index].key = key;
table[index].state = OCCUPIED;
```

Problème?

- Regroupement d'éléments (clustering).

Méthode linéaire

Comment ça marche?

- Comme la méthode séquentielle mais on “saute” de `k`.

```
index = h(key);
while (table[index].state == OCCUPIED && table[index].key != key) {
   index = (index + k) % table_size; // attention à pas dépasser
}
table[index].key = key;
table[index].state = OCCUPIED;
```

Quelle valeur de `k` éviter?

Méthode linéaire

Comment ça marche?

- Comme la méthode séquentielle mais on “saute” de `k`.

```
index = h(key);
while (table[index].state == OCCUPIED && table[index].key != key) {
   index = (index + k) % table_size; // attention à pas dépasser
}
table[index].key = key;
table[index].state = OCCUPIED;
```

Quelle valeur de `k` éviter?

- Une valeur où `table_size` est multiple de `k`.

Cette méthode répartit mieux les regroupements au travers de la table.

Méthode du double hashing

Comment ça marche?

- Comme la méthode linéaire, mais `k = h2(key)` (variable).

```
index = h(key);
while (table[index].state == OCCUPIED && table[index].key != key) {
   index = (index + h2(k)) % table_size; // attention à pas dépasser
}
table[index].key = key;
table[index].state = OCCUPIED;
```

Quelle propriété doit avoir `h2`?

Exemple

```
h2(key) = (table_size - 2) - key % (table_size -2)
```

Méthode pseudo-aléatoire

Comment ça marche?

- Comme la méthode linéaire mais on génère `k` pseudo-aléatoirement.

```
index = h(key);
while (table[index].state == OCCUPIED && table[index].key != key) {
    index = (index + random_number) % table_size;
}
table[index].key = key;
table[index].state = OCCUPIED;
```

Comment s'assurer qu'on va bien retrouver la bonne clé?

Méthode pseudo-aléatoire

Comment ça marche?

- Comme la méthode linéaire mais on génère `k` pseudo-aléatoirement.

```
index = h(key);
while (table[index].state == OCCUPIED && table[index].key != key) {
    index = (index + random_number) % table_size;
}
table[index].key = key;
table[index].state = OCCUPIED;
```

Comment s'assurer qu'on va bien retrouver la bonne clé?

- Le germe (seed) de la séquence pseudo-aléatoire doit être le même.
- Le germe à choisir est l'index retourné par `h(key)`.

```
srand(h(key));
while {
    random_number = rand();
}
```

Méthode quadratique

- La fonction des indices de collision est de degré 2.
- Soit $J_0 = h(key)$, les indices de collision se construisent comme:

```
J_i = J_0 + i^2 % table_size, i > 0,
J_0 = 100, J_1 = 101, J_2 = 104, J_3 = 109, ...
```

Problème possible?

Méthode quadratique

- La fonction des indices de collision est de degré 2.
- Soit $J_0 = h(key)$, les indices de collision se construisent comme:

```
J_i = J_0 + i^2 % table_size, i > 0,
J_0 = 100, J_1 = 101, J_2 = 104, J_3 = 109, ...
```

Problème possible?

- Calculer le carré peut-être “lent”.
- En fait on peut ruser un peu.

Méthode quadratique

```
J_i = J_0 + i^2 % table_size, i > 0,
J_0 = 100
          \
           d_0 = 1
          /        \
J_1 = 101           Delta = 2
          \        /
           d_1 = 3
          /        \
J_2 = 104           Delta = 2
          \        /
           d_2 = 5
          /        \
J_3 = 109           Delta = 2
          \        /
           d_3 = 7
          /
J_4 = 116
-------------------------------------
J_{i+1} = J_i + d_i,
d_{i+1} = d_i + Delta, d_0 = 1, i > 0.
```

Méthode de chaînage

Comment ça marche?

- Chaque index de la table contient un pointeur vers une liste chaînée contenant les paires clés-valeurs.

Un petit dessin

Méthode de chaînage

Exemple

On hash avec la fonction `h(key) = key % 11` (`key` est le numéro de la lettre de l'alphabet)

U	N	E	X	E	M	P	L	E	D	E	T	A	B	L	E
10	3	5	2	5	2	5	1	5	4	5	9	1	2	1	5

Comment on représente ça? (à vous)

Méthode de chaînage

Exemple

On hash avec la fonction `h(key) = key % 11` (`key` est le numéro de la lettre de l'alphabet)

U	N	E	X	E	M	P	L	E	D	E	T	A	B	L	E
10	3	5	2	5	2	5	1	5	4	5	9	1	2	1	5

Comment on représente ça? (à vous)

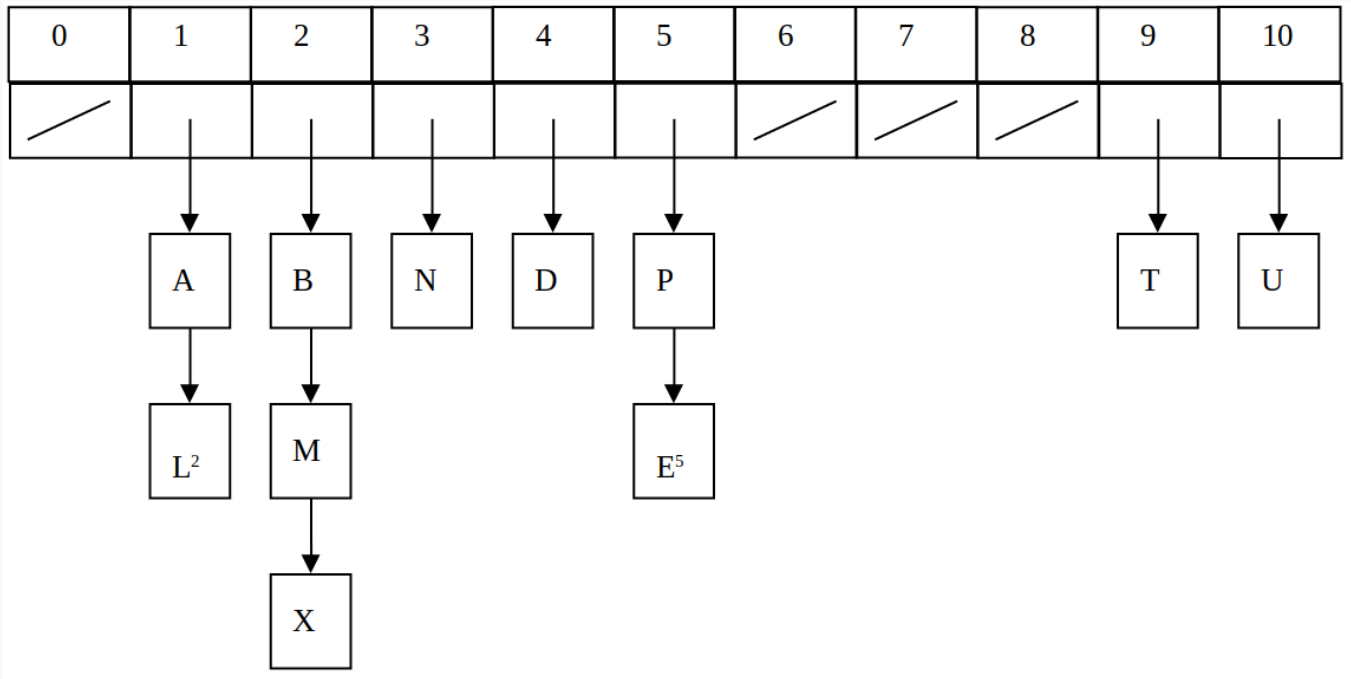

Figure 1: La méthode de chaînage

Méthode de chaînage

Avantages:

- Si les clés sont grandes l'économie de place est importante (les places vides sont `NULL`).
- La gestion des collisions est conceptuellement simple.
- Pas de problème de regroupement (clustering).

Exercice 1

- Construire une table à partir de la liste de clés suivante: `R, E, C, O, U, P, A, N, T`
- On suppose que la table est initialement vide, de taille $n = 13$.
- Utiliser la fonction $h1(k) = k \mod 13$ où k est la k-ème lettre de l'alphabet et un traitement séquentiel des collisions.

Exercice 2

- Reprendre l'exercice 1 et utiliser la technique de double hachage pour traiter les collisions avec

$$h_1(k) = k \mod 13,$$
$$h_2(k) = 1 + (k \mod 11).$$

- La fonction de hachage est donc $h(k) = (h(k) + h_2(k))\%13$ en cas de collision.

Exercice 3

- Stocker les numéros de téléphones internes d'une entreprise suivants dans un tableau de 10 positions.
- Les numéros sont compris entre 100 et 299.
- Soit N le numéro de téléphone, la fonction de hachage est

$$h(N) = N \mod 10.$$

- La fonction de gestion des collisions est

$$C_1(N, i) = (h(N) + 3 \cdot i) \mod 10.$$

- Placer 145, 167, 110, 175, 210, 215 (mettre son état à occupé).
- Supprimer 175 (rechercher 175, et mettre son état à supprimé).
- Rechercher 35.
- Les cases ni supprimées, ni occupées sont vides.
- Expliquer se qui se passe si on utilise?

$$C_1(N, i) = (h(N) + 5 \cdot i) \mod 10.$$