Tables de hachage

Algorithmique et structures de données, 2022-2023

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

2023-02-24

En partie inspirés des supports de cours de P. Albuquerque

Rappel sur les tables de hachage (1/N)

Définition? Qui se souvient?

Rappel sur les tables de hachage (1/N)

Définition? Qui se souvient?

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

Rappel sur les tables de hachage (1/N)

Définition? Qui se souvient?

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)
- ...

Rappel sur les tables de hachage (1/N)

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`

Transformation de clé (hashing)

- Format: 106.3123.8492.13

```
Numéro AVS     | Nom
000000000000   | -------
...            | ...
106312384921 3 | Paul
...            | ...
306671387832 8 | Orestis
...            | ...
999999999999   | -------
```

- Nombre de numéros » nombre d'entrées.

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: exemple

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: exemple

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

- 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;
```

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)

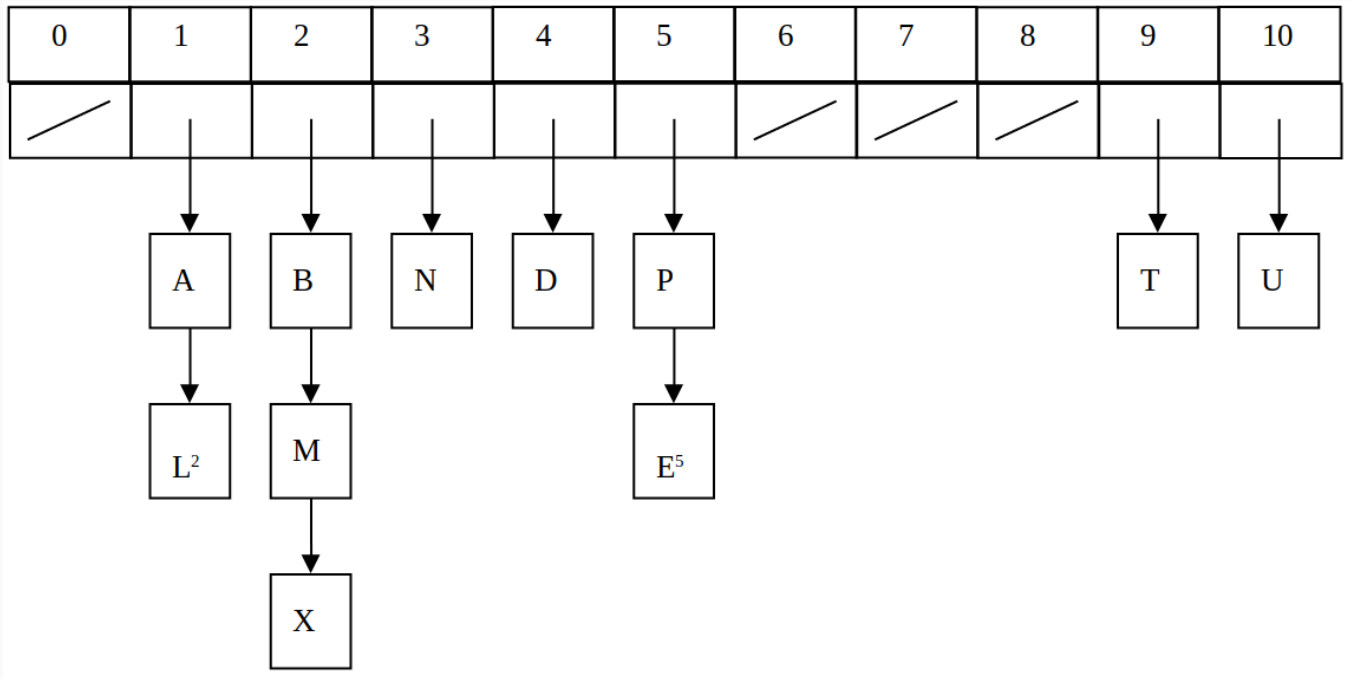

Figure 1: La méthode de chaînage

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?

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

Préambule

- On considère pas le cas du chaînage en cas de collisions.
- L'insertion est construite avec une forme du type

```
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;
```

- Gestion de l'état d'une case *explicite*

```
typedef enum {EMPTY, OCCUPIED, DELETED} state;
```

L'insertion

Pseudocode?

L'insertion

Pseudocode?

```
insert(table, key, value) {
    index = hash de la clé;
    index =
        si "index" est déjà "occupé"
        et la clé correspondante n'est pas "key"
        alors gérer la collision;

    changer l'état de la case "index" à "occupé";
    changer la valeur de la case "index" à "value";
}
```

La suppression

Pseudocode?

La suppression

Pseudocode?

```
value_t remove(table, key) {
    index = hash de la clé;
    tant que l'état de la case n'est pas "vide"
        si "index" est "occupé" et la clé est "key"
            changer l'état de la case à "supprimé"
        sinon
            index = rehash
}
```

La recherche

Pseudocode?

La recherche

Pseudocode?

```
bool search(table, key, value) {
    index = hash de la clé;
    tant que l'état de la case n'est pas "vide"
        si "index" est "occupé" et la clé est "key"
            retourner vrai
        sinon
            index = rehash
}
```

Écrivons le code!

- Mais avant:
  - Quelles sont les structures de données dont nous avons besoin?
  - Y a-t-il des fonctions auxiliaires à écrire?
  - Écrire les signatures des fonctions.

Écrivons le code!

- Mais avant:
  - Quelles sont les structures de données dont nous avons besoin?
  - Y a-t-il des fonctions auxiliaires à écrire?
  - Écrire les signatures des fonctions.

Structures de données

Écrivons le code!

- Mais avant:
  - Quelles sont les structures de données dont nous avons besoin?
  - Y a-t-il des fonctions auxiliaires à écrire?
  - Écrire les signatures des fonctions.

Structures de données

```
typedef enum {empty, deleted, occupied};
typedef ... key_t;
typedef ... value_t;
typedef struct _cell_t {
    key_t key;
    value_t value;
    state_t state;
} cell_t;
typedef struct _hm {
    cell_t *table;
    int capacity;
    int size;
} hm;
```

Écrivons le code!

Fonctions auxiliaires

Écrivons le code!

Fonctions auxiliaires

```c
static int hash(key_t key);
static int rehash(int index, key_t key);
static int find_index(hm h, key_t key);
```

Signature de l'API

Écrivons le code!

Fonctions auxiliaires

```
static int hash(key_t key);
static int rehash(int index, key_t key);
static int find_index(hm h, key_t key);
```

Signature de l'API

```
void hm_init(hm *h, int capacity);
void hm_destroy(hm *h);
bool hm_set(hm *h, key_t key, value_t *value);
bool hm_get(hm h, key_t key, value_t *value);
bool hm_remove(hm *h, key_t key, value_t *value);
bool hm_search(hm h, key_t key);
void hm_print(hm h);
```

Live code session!

0. Offered to you by ProtonVPN[1]!

[1] The fastest way to connect to BBB!

Live code session!

0. Offered to you by ProtonVPN[1]!

1. Like the video.
2. Subscribe to the channel.
3. Use our one time voucher for ProtonVPN: `PAULISAWESOME`.
4. Consider donating on our patreon.

[1] The fastest way to connect to BBB!