

Listes doublement chaînées et tables de hachage

Algorithmes et structures de données, 2025-2026

P. Albuquerque (B410) et O. Malaspinas (A401), ISC, HEPIA
2026-02-18

En partie inspiré des supports de cours de P. Albuquerque

Quel but ?

- Permet de retrouver rapidement un élément.
- Utile pour la recherche de plus court chemin dans des graphes.
- Ordonnancement de processus par degré de priorité.

Comment ?

- Les implémentations les plus efficaces se basent sur les tableaux.
- Possibles aussi avec des listes chaînées.

Les listes triées

Quelle structure de données dans notre cas ?

Une liste chaînée bien sûr (oui, c'est pour vous entraîner) !

```
typedef struct _element { // chaque élément
    int data;
    struct _element *next;
} element;
typedef element* sorted_list; // la liste
```

Fonctionnalités

```
// insertion de val
sorted_list sorted_list_push(sorted_list list, int val);
// la liste est-elle vide?
bool sorted_list_is_empty(sorted_list list); // list == NULL
// extraction de val (il disparaît)
sorted_list sorted_list_extract(sorted_list list, int val);
// rechercher un élément et le retourner
element* sorted_list_search(sorted_list list, int val);
```

L'insertion (1/3)

Trois cas

1. La liste est vide.

L'insertion (1/3)

Trois cas

1. La liste est vide.

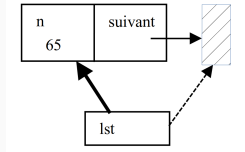


Figure 1 : Insertion dans une liste vide, `list == NULL`.

L'insertion (1/3)

Trois cas

1. La liste est vide.

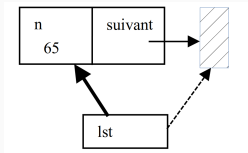


Figure 1 : Insertion dans une liste vide, `list == NULL`.

```
sorted_list sorted_list_push(sorted_list list, int val) {  
    if (sorted_list_is_empty(list)) {  
        list = malloc(sizeof(*list));  
        list->data = val;  
        list->next = NULL;  
        return list;  
    }  
}
```

L'insertion (2/3)

2. L'insertion se fait en première position.

L'insertion (2/3)

2. L'insertion se fait en première position.

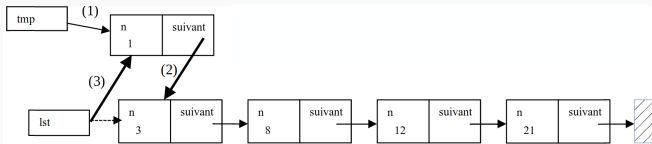


Figure 2 : Insertion en tête de liste, `list->data >= val`.

L'insertion (2/3)

2. L'insertion se fait en première position.

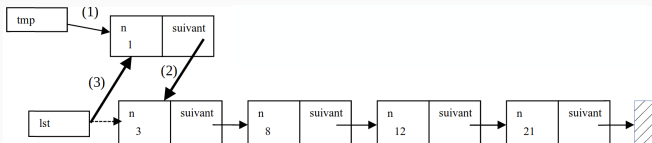


Figure 2 : Insertion en tête de liste, `list->data >= val`.

```
sorted_list sorted_list_push(sorted_list list, int val) {  
    if (list->data >= val) {  
        element *tmp = malloc(sizeof(*tmp));  
        tmp->data = val;  
        tmp->next = list;  
        list = tmp;  
        return list;  
    }  
}
```

L'insertion (3/3)

3. L'insertion se fait sur une autre position que la première.

L'insertion (3/3)

3. L'insertion se fait sur une autre position que la première.

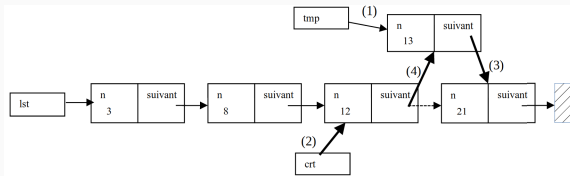


Figure 3 : Insertion sur une autre position, $list \rightarrow data < val$.

L'insertion (3/3)

3. L'insertion se fait sur une autre position que la première.

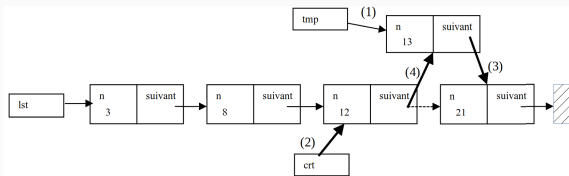


Figure 3 : Insertion sur une autre position, $\text{list} \rightarrow \text{data} < \text{val}$.

```
sorted_list sorted_list_push(sorted_list list, int val) {
    element *tmp = malloc(sizeof(*tmp));
    tmp->data = val;
    element *crt = list;
    while (NULL != crt->next && val > crt->next->data) {
        crt = crt->next;
    }
    tmp->next = crt->next;
    crt->next = tmp;
    return list;
}
```

L'extraction (1/3)

Trois cas

1. L'élément à extraire n'est **pas** le premier élément de la liste.

L'extraction (1/3)

Trois cas

1. L'élément à extraire n'est **pas** le premier élément de la liste.

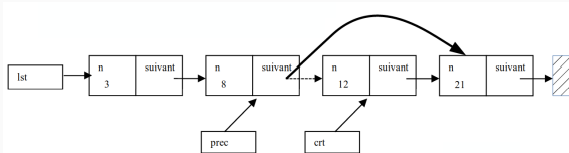


Figure 4 : Extraction d'un élément qui n'est pas le premier.

L'extraction (1/3)

Trois cas

1. L'élément à extraire n'est **pas** le premier élément de la liste.

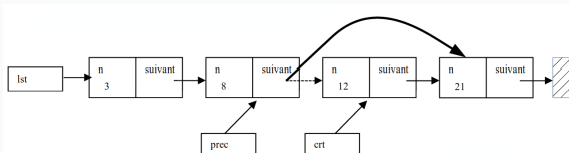


Figure 4 : Extraction d'un élément qui n'est pas le premier.

```
sorted_list sorted_list_extract(sorted_list list, int val) {  
    element *prec = *crt = list; // needed to glue elements together  
    while (NULL != crt && val > crt->data) {  
        prec = crt;  
        crt = crt->next;  
    }  
    if (NULL != crt && prec != crt && crt->data == val) { // glue things together  
        prec->next = crt->next;  
        free(crt);  
    }  
    return list;  
}
```

L'extraction (2/3)

-
2. L'élément à extraire est le premier élément de la liste.

L'extraction (2/3)

2. L'élément à extraire est le premier élément de la liste.

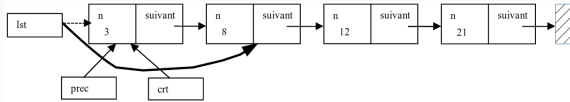


Figure 5 : Extraction d'un élément qui est le premier.

L'extraction (2/3)

2. L'élément à extraire est le premier élément de la liste.

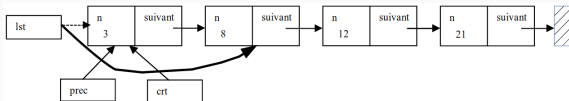


Figure 5 : Extraction d'un élément qui est le premier.

```
sorted_list sorted_list_extract(sorted_list list, int val) {  
    element *prec = *crt = list; // needed to glue elements together  
    while (NULL != crt && val > crt->data) {  
        prec = crt;  
        crt = crt->next;  
    }  
    // glue things together  
    if (NULL != crt && crt->data == val && prec == crt) {  
        list = list->next;  
        free(crt);  
    }  
    return list;  
}
```

L'extraction (3/3)

3. L'élément à extraire n'est **pas** dans la liste.
- La liste est vide.
 - La valeur est plus grande que le dernier élément de la liste.
 - La valeur est plus petite que la valeur de crt.

L'extraction (3/3)

3. L'élément à extraire n'est **pas** dans la liste.

- La liste est vide.
- La valeur est plus grande que le dernier élément de la liste.
- La valeur est plus petite que la valeur de crt.

On retourne la liste inchangée.

L'extraction (3/3)

3. L'élément à extraire n'est **pas** dans la liste.

- La liste est vide.
- La valeur est plus grande que le dernier élément de la liste.
- La valeur est plus petite que la valeur de crt.

On retourne la liste inchangée.

```
sorted_list sorted_list_extract(sorted_list list, int val) {  
    element *prec = *crt = list; // needed to glue elements together  
    while (NULL != crt && val > crt->data) {  
        prec = crt;  
        crt = crt->next;  
    }  
    if (NULL == crt || crt->data != val) { // val not present  
        return list;  
    }  
}
```

La recherche

```
element* sorted_list_search(sorted_list list, int val);
```

- Retourne NULL si la valeur n'est pas présente (ou la liste vide).
- Retourne un pointeur vers l'élément si la valeur est présente.

La recherche

```
element* sorted_list_search(sorted_list list, int val);
```

- Retourne NULL si la valeur n'est pas présente (ou la liste vide).
- Retourne un pointeur vers l'élément si la valeur est présente.

```
element* sorted_list_search(sorted_list list, int val) {  
    // search for element smaller than val  
    element* pos = sorted_list_position(list, val);  
    if (NULL == pos && val == list->data) {  
        return list; // first element contains val  
    } else if (NULL != pos && NULL != pos->next  
        && val == pos->next->data) {  
        return pos->next; // non-first element contains val  
    } else {  
        return NULL; // well... val's not here  
    }  
}
```

La recherche

La fonction `sorted_list_position`

```
element* sorted_list_position(sorted_list list, int val);
```

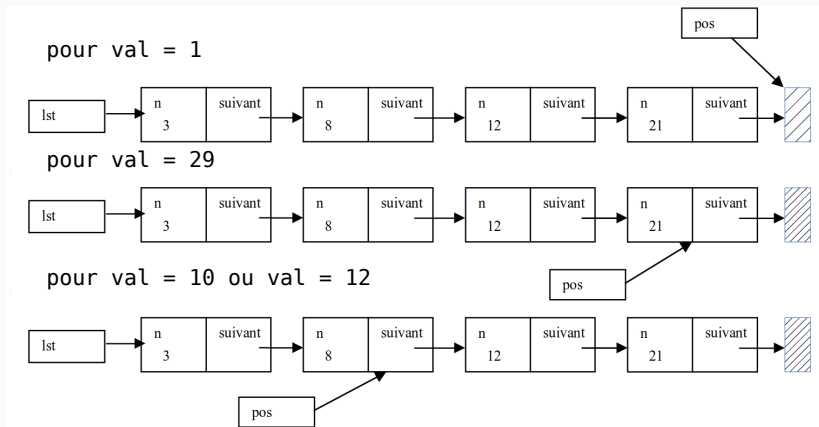


Figure 6 : Trois exemples de retour de la fonction `sorted_list_position()`.

Exercice : implémenter

```
element* sorted_list_position(sorted_list list, int val);
```

Exercice : implémenter

```
element* sorted_list_position(sorted_list list, int val);

element* sorted_list_position(sorted_list list, int val) {
    element* pos = list;
    if (sorted_list_is_empty(list) || val <= list->data) {
        pos = NULL;
    } else {
        while (NULL != pos->next && val > pos->next->data) {
            pos = pos->next;
        }
    }
    return pos;
}
```

Complexité de la liste chaînée triée

L'insertion ?

Complexité de la liste chaînée triée

L'insertion ?

$$\mathcal{O}(N).$$

L'extraction ?

Complexité de la liste chaînée triée

L'insertion ?

$$\mathcal{O}(N).$$

L'extraction ?

$$\mathcal{O}(N).$$

La recherche ?

Complexité de la liste chaînée triée

L'insertion ?

$$\mathcal{O}(N).$$

L'extraction ?

$$\mathcal{O}(N).$$

La recherche ?

$$\mathcal{O}(N).$$

Listes doublement chaînées

Liste doublement chaînée

Application : navigateur ou éditeur de texte

- Avec une liste chaînée :
 - Comment implémenter les fonctions back et forward d'un navigateur ?
 - Comment implémenter les fonctions undo et redo d'un éditeur de texte ?

Liste doublement chaînée

Application : navigateur ou éditeur de texte

- Avec une liste chaînée :
 - Comment implémenter les fonctions back et forward d'un navigateur ?
 - Comment implémenter les fonctions undo et redo d'un éditeur de texte ?

Pas possible.

Solution ?

Liste doublement chaînée

Application : navigateur ou éditeur de texte

- Avec une liste chaînée :
 - Comment implémenter les fonctions back et forward d'un navigateur ?
 - Comment implémenter les fonctions undo et redo d'un éditeur de texte ?

Pas possible.

Solution ?

- Garder un pointeur supplémentaire sur l'élément précédent et pas seulement le suivant.

Liste doublement chaînée

Application : navigateur ou éditeur de texte

- Avec une liste chaînée :
 - Comment implémenter les fonctions back et forward d'un navigateur ?
 - Comment implémenter les fonctions undo et redo d'un éditeur de texte ?

Pas possible.

Solution ?

- Garder un pointeur supplémentaire sur l'élément précédent et pas seulement le suivant.
- Cette structure de données est la **liste doublement chaînée** ou **doubly linked list**.

Liste doublement chaînée

Liste doublement chaînée

Exercices

- Partir du dessin suivant et par **groupe de 5**

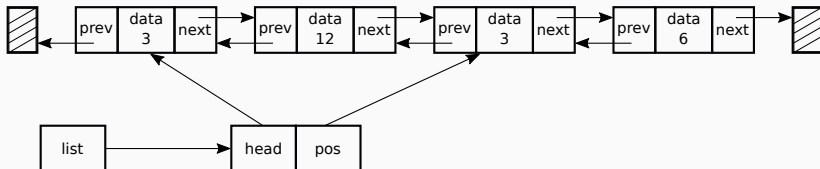


Figure 7 : Un schéma de liste doublement chaînée d'entiers.

- Écrire les structures de données pour représenter la liste doublement chaînée dont le type sera `dll` (pour `doubly_linked_list`)

Liste doublement chaînée

2. Écrire les fonctionnalités de création et consultation

```
// crée la liste doublement chaînée
dll dll_create();

// retourne la valeur à la position actuelle dans la liste
int dll_value(dll list);

// la liste est-elle vide?
bool dll_is_empty(dll list);

// pos est-il le 1er élément?
bool dll_is_head(dll list);

// pos est-il le dernier élément?
bool dll_is_tail(dll list);

// data est-elle dans la liste?
bool dll_is_present(dll list, int data);

// affiche la liste
void dll_print(dll list);
```

Liste doublement chaînée

3. Écrire les fonctionnalités de manipulation

```
// déplace pos au début de la liste  
dll dll_move_to_head(dll list);  
// déplace pos à la position suivante dans la liste  
dll dll_next(dll list);  
// déplace pos à la position précédente dans la liste  
dll dll_prev(dll list);
```

Liste doublement chaînée

4. Écrire les fonctionnalités d'insertion

```
// insertion de data dans l'élément après pos  
dll dll_insert_after(dll list, int data);  
// insertion de data en tête de liste  
dll dll_push(dll list, int data);
```

5. Écrire les fonctionnalités d'extraction

```
// extraction de la valeur se trouvant dans l'élément pos  
// l'élément pos est libéré  
int dll_extract(dll *list);  
// extrait la donnée en tête de liste  
int dll_pop(dll *list);  
// vide la liste  
void dll_destroy(dll *list);
```


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";
```

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

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)
- ...

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^{-6} ($10^{-4}\%$ 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 bien plus petit que le nombre de clés possibles.
- 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

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.

$\text{key} = 5486 \rightarrow \text{key}^2 = 30096196 \rightarrow \text{index} = 96$

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.

$$h(\text{key}) = \text{key} \% N.$$

Quelle doit être la taille de la table ?

Fonctions de transformation de clés : exemples

Méthode par division modulo

Taille de l'index : N chiffres.

$$h(\text{key}) = \text{key} \% N.$$

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 $k = h_2(\text{key})$ (variable).

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

Quelle propriété doit avoir h_2 ?

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 $h(\text{key}) = \text{key} \% 11$ (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)

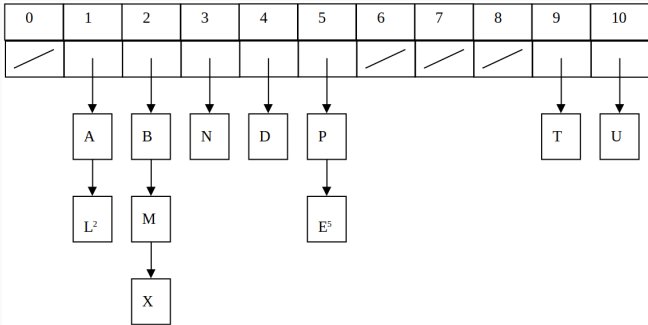
Méthode de chaînage

Example

On hash avec la fonction $h(\text{key}) = \text{key} \% 11$ (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

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 $h_1(k) = k \bmod 13$ 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

$$\begin{aligned}h_1(k) &= k \bmod 13, \\h_2(k) &= 1 + (k \bmod 11).\end{aligned}$$

- La fonction de hachage est donc $h(k) = (h_1(k) + h_2(k)) \% 13$ 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

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

- La fonction de gestion des collisions est

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

- 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 ?

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