Chapitre 4

Conteneurs et fonctionnement des dictionnaires

Sommaire

I Généralités

Nous allons voir dans les grandes lignes comment sont implémentés les principaux conteneurs.

I.1 Définition

Définition 4.1: conteneurs

Les conteneurs sont des objets abstraits qui permettent de stocker des objets informatiques sous forme organisée. Ces conteneurs peuvent être implémentés de différentes façons, et en conséquence, suivre certaines règles. Ces différentes manières de les implémenter entrainent que les actions qui en sont liées ont des complexités en temps et en espace différentes.

Remarque 4.2. *Les conteneurs sont souvent accompagner d'un objet permettant de parcourir tous les éléments, ce qu'on appelle les itérateurs.*

I.2 Propriétés générales

Trois propriétés sont fondamentales pour un conteneur :

- **l'accès :** la manière d'accéder aux différents éléments du conteneur ;
- **le stockage :** la méthode dont sont stockés les différents éléments ;
- **le parcours :** la manière de parcourir les différents éléments du conteneur.

Différentes méthodes peuvent être implémentées (liste non exhaustive) :

- créer un conteneur vide ;
- ajouter, supprimer des objets au conteneur ;
- accéder aux objets du conteneur ;
- connaître le nombre d'objet du conteneur.

II Les principaux conteneurs en informatique

II.1 Liste chaînée

Définition 4.3: liste chaînée

Une liste chainée est un conteneur, éventuellement vide, dont chaque élément ou cellule contient une donnée et l'adresse (mémoire) de la cellule suivante, on dit alors qu'elle pointe sur l'élément suivant. L'accès aux éléments d'une liste se fait de manière séquentielle : chaque élément permet l'accès au suivant (contrairement au tableau dans lequel l'accès se fait de manière directe, par adressage de chaque cellule dudit tableau).

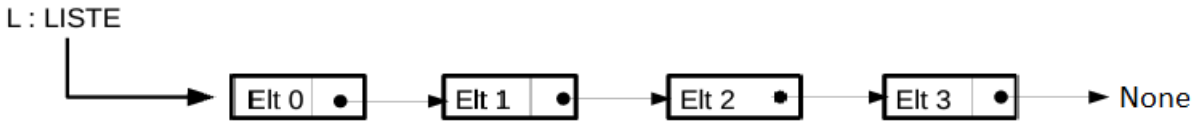


ILLUSTRATION D'UNE LISTE CHAÎNÉE

Cette structure supporte (la plupart du temps) les opérations minimales suivantes :

- création d'une liste vide ;
- ajout, recherche et suppression d'un élément dans la liste ;
- insertion d'un élément dans une position donnée ;
- accès à la cellule de tête ;
- accès au successeur d'une cellule (s'il y a un successeur).

Remarque 4.4. *En python, le type liste n'est pas réellement une liste chaînée, mais plutôt un tableau dont on peut changer la taille (cf sous-sections suivantes). L'accès au successeur n'est pas implémenté pour les listes python, alors qu'on peut accéder à l'élément d'indice k (il faudrait normalement partir de la cellule de tête et réaliser k fois l'opération successeur)*

II.2 Tableau

Définition 4.5: Tableau

Une structure de tableau est un conteneur possédant un nombre fixe d'éléments de même type.

- le nombre et le type des éléments sont définis à la création du tableau (un tableau est donc immuable contrairement à une liste) ;
- chaque composant du tableau est directement accessible en un temps $O(1)$, ce qui suppose que les données soient stockées dans des emplacements mémoire contigus et que l'élément d'adresse T [i] soit directement accessible.

Remarque 4.6. *Les tableaux ont l'avantage qu'on accède plus facilement à un élément que dans une liste chaînée. Par contre, on ne peut rajouter ou enlever un élément comme dans une liste chaînée. L'implémentation des listes en python semble être un hybride des deux conteneurs précédents, car le temps d'accès à une cellule n'est pas proportionnel à son indice.*

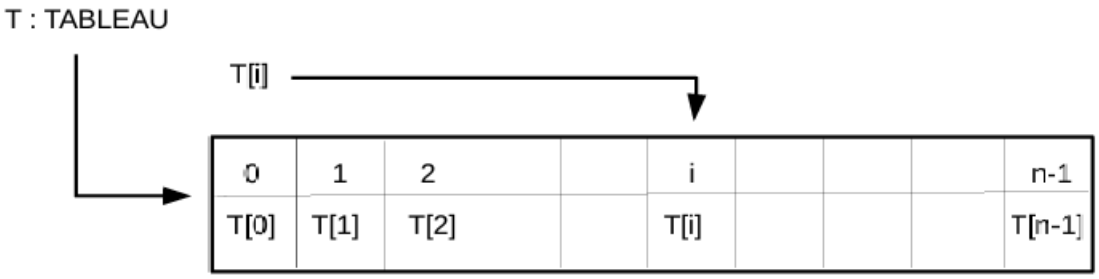


ILLUSTRATION D'UN TABLEAU

II.3 Tableau associatif ou dictionnaire

Définition 4.7

Un tableau associatif ou dictionnaire est un conteneur dont les éléments sont des couples clef-valeur. C'est une application associant un ensemble de clefs à un ensemble de valeurs.

Remarque 4.8. *La recherche d'un élément se fait sur la clef (alors que dans un tableau elle ne se fait pas sur l'indice qui joue le rôle de la clé mais sur les éléments associés aux indices)*

Les tableaux associatifs supportent :

- la création d'une structure vide ;
- l'insertion d'un couple clé-valeur ;
- la suppression d'une clé (et de la valeur associée) ;
- la recherche d'une clé en un temps $O(1)$ en moyenne.

III Implémentation des dictionnaires

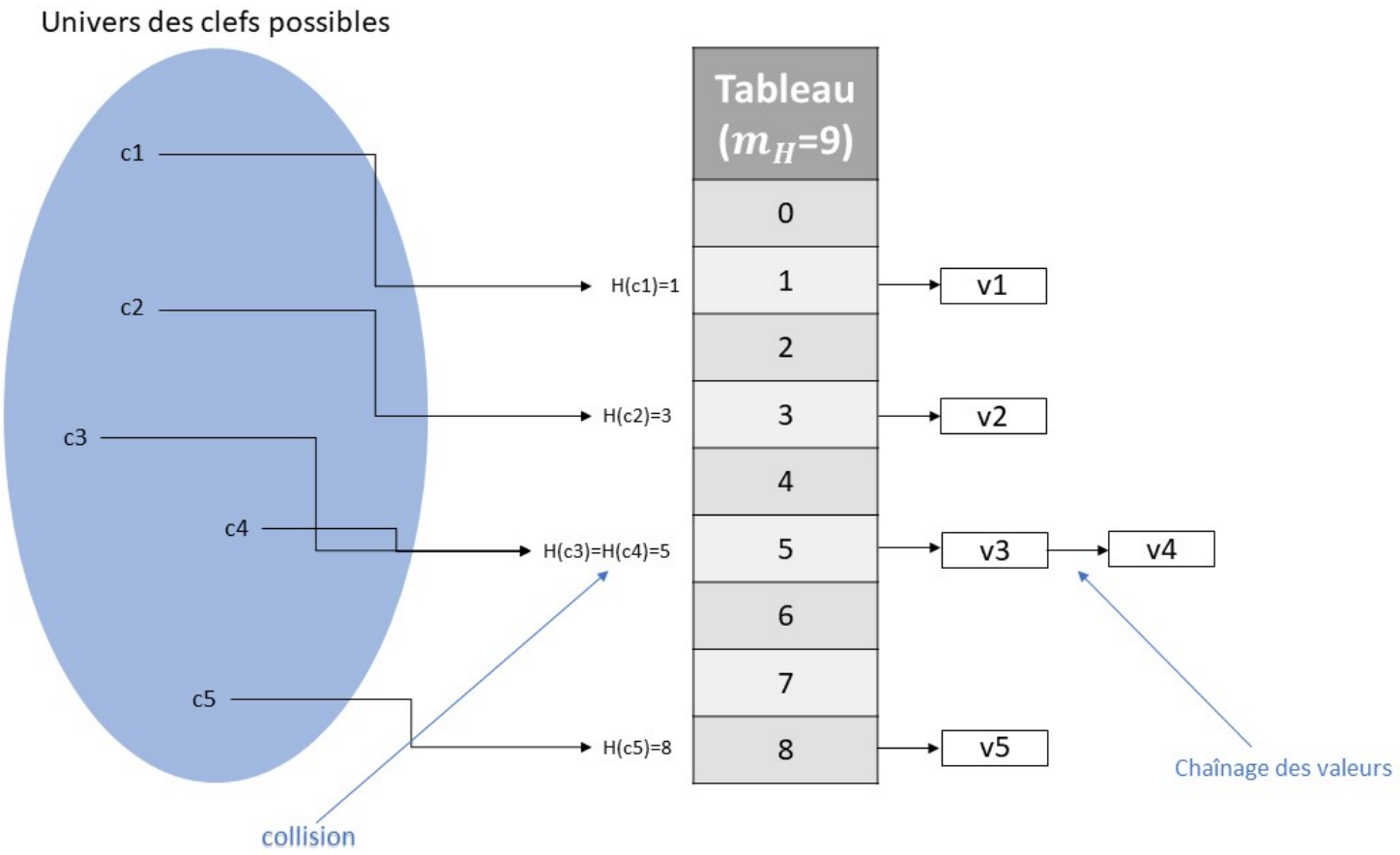
III.1 Fonctionnement et tables de hachage

L'implémentation d'une structure de dictionnaire procède généralement de la façon suivante :

1. On détermine un ensemble de clés envisageables (selon le langage). Par exemple, python accepte comme clé la plupart des types d'objets (les entiers, flottants, chaînes de caractères) sauf les listes ;
2. On se donne une famille de fonctions H (appelée **fonction de hachage**) de mon ensemble de clés possibles dans un segment d'entiers $[\![0, m_H - 1]\!]$;
3. Au moment de créer un dictionnaire une fonction de hachage est choisie et un tableau de taille m_H est réservé. On appelle **alvéole** les éléments de ce tableau.
4. Pour insérer un couple clef-valeur, H(clef) est calculé. C'est un entier $i \in [\![0, m_H - 1]\!]$
5. On place dans les alvéoles des pointeurs (adresses) vers des listes chaînées dans lesquelles les couples clefs-valeurs sont ajoutées (ou retranchés) au fur et à mesure.

Remarque 4.9.

- *On pourrait se demander pourquoi on insère pas directement la valeur de la clef dans l'emplacement H(clef). Si on effectue cela, il faut alors que pour tout clef2 différente de clef1, H(clef1) $\neq$ H(clef2), autrement dit que la fonction H soit injective. Mais ceci est totalement impossible car l'univers des clefs possibles est infini. Et quand bien même on se limiterait à des clefs d'une certaine taille mémoire, il serait bien trop grand... Plusieurs clefs doivent avoir la même image par H (valeur de hachage). Lorsqu'on utilise deux clefs ayant la même image par H, on parle de* ***collision****.*
- *Lorsque le nombre de clefs insérées augmente fortement, il est parfois nécessaire de redimensionner le tableau. En effet, pour que le dictionnaire conserve ses bonnes propriétés, il est nécessaire que les listes chaînées ne soient pas trop grandes. Lorsqu'on redimensionne le tableau, on change également de fonction de hachage.*



REPRÉSENTATION D'UNE TABLE DE HACHAGE, LA CLEF ci EST ASSOCIÉ À vi

C'est cette structure que l'on appelle table de hachage. Sa performance dépend de la fonction choisie. On souhaitera en général limiter le nombre de collisions tout en optimisant le taux d'occupation des alvéoles que l'on appelle aussi facteur ou taux de remplissage : $\alpha = \frac{n}{m_H}$ où n est le nombre de clés (ou de couples clé-valeur) insérées.

III.2 Quelques exemples de fonctions de hachage simples

En pratique, les objets qui sont des clefs possibles sont d'abord transformés en entier, puis on utilise cette valeur avec la fonction de hachage. Voici un exemple pour transformer les chaînes de caractères en entiers.

Transformation d'une chaîne de caractères en entier

On propose ici un exemple simple de pré-traitement des chaînes de caractères avant hachage. Nous allons nous servir du codage ASCII qui est donné sous Python par la fonction `ord(c)` qui renvoie le code ASCII sur 8 bits du caractère c. Si `ch` est une chaîne de n caractères du code ASCII , on associe à `ch` la valeur : $\sum_{k=0}^{n-1} \text{ord}(\text{ch}[k])256^k$. La fonction ainsi définie est bijective de l'ensemble des chaînes de caractères du code ASCII à valeurs dans $\mathbb{N}$.

```python
def chaine_en_entier(ch):
        r=0
        for k in ch:
                r=256*r+ord(k)
        return r
```

Par exemple, la chaine "Lycee Marceau" renvoie l'entier 605890819975307780371850811019 7.

Première fonction de hachage

C'est la fonction la plus simple, pour une table avec m alvéoles on choisit

$$H : c \mapsto c \bmod m$$

La qualité de ce hachage dépend fortement du choix de m si les clés ont une répartition qui n'est pas aléatoire. En pratique on choisit des nombres premiers éloignés des puissances de 2.

Seconde fonction de hachage

Cette méthode consiste à construire des fonctions de hachage de la façon suivante :

- on se donne un réel (ou un flottant) $\theta \in]0,1[$ et un entier $m > 0$.
- on définit

  $$H : e \in \mathbb{N}^* \mapsto \lfloor m \times d(e \times \theta) \rfloor \in [\![0, m-1]\!]$$

  où $d : x \in \mathbb{R} \mapsto x - \lfloor x \rfloor \in [0,1[$. Pour que cette fonction répartit uniformément les clefs, il faudrait prendre θ irrationnel. Mais comme tout représentant des réels est un décimal, il faut que sa représentation soit de la forme $\frac{p}{q}$ avec q très grand. En effet, si $\theta = \frac{p}{q}$ et $e = b \times q + r$ avec $0 \leq r < q$, alors $d(\theta e) = d(\theta r)$ ce qui implique qu'il n'y a que q valeurs possibles.

III.3 Opérations des dictionnaires en python

Opération	Résultat
`D={}`	affecte à la variable D un dictionnaire vide
`D={'a':1,'b':23}`	affecte à la variable D un dictionnaire donc les couples clefs-valeurs sont 'a'-1 et 'b'-23
`D['a']`	renvoie la valeur associée à la clef `'a'`
`D['a']=32`	affecte à la clef `'a'` la valeur entière 32
`del D['a']`	supprime l'association clef-valeur liée à la clef `'a'` dans D
`x in D, x not in D`	teste si x est une clef de D ou non
`list(D), set(D)`	renvoie la liste (resp. l'ensemble) des clefs de D
`D.keys(), D.values()`	permet de créer un itérateur sur les clefs ou les valeurs (pour les boules itératives)
`D.items()`	permet de créer un itérateur sur les couples clefs-valeurs
`D.copy()`	permet de copier D (sans dépendance)
`D={k:k**2 for k in range(5)}`	renvoie `{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}`

III.4 Exercices

Exercice 1. *Écrire une fonction* `occurrence(L)` *qui prend en argument une liste L et renvoie un dictionnaire où les clefs sont les éléments de L et les valeurs le nombre d'occurrence de la clef dans L.*

Exercice 2. *Écrire une fonction* `occurrence2(texte)` *qui prend en argument une chaîne de caractères* `texte` *et renvoie un dictionnaire où les clefs sont les lettres de* `texte` *et les valeurs le nombre d'occurrence de la clef dans texte.*

Exercice 3. *Écrire une fonction* `diviseurs(L)` *qui prend en argument une liste L d'entiers naturels et qui renvoie le dictionnaire dont les clefs sont les éléments de L et les valeurs la liste des diviseurs entiers naturels de la clef.*

Exercice 4. *On suppose que M est la matrice (liste de listes) d'adjacence d'un graphe dont les m sommets sont numérotés de 0 à* $m-1$*. Écrire une fonction python* `adj(M)` *qui prend cette matrice et renvoie un dictionnaire dont les clefs sont les sommets et les valeurs une liste contenant les voisins du sommet clef.*