Dictionnaires et tables de hachage

Cours 04

1 Dictionnaires en Python

1.1 Vocabulaire

Les **dictionnaires** sont une structure de données, qui est une « version alternative » des tableaux : dans un dictionnaire, les indices ne sont pas nécessairement des entiers.

`i`	0	1	2	3	4	5	6	7	8	9
`tab[i]`	1	2	3	5	8	13	21	34	55	89

clés	`"bob"`	`"zoe"`	`"ben"`	`"ada"`	`"moe"`	`"lea"`	`"kit"`	`"bea"`	`"sam"`	`"pam"`
valeurs	14	12	7	19	6	13	10	6	17	9

Les dictionnaires en Python sont :

— mutables : on peut en modifier les valeurs ;

— dynamiques : on peut rajouter des valeurs ;

— on peut même supprimer des valeurs.

1.2 Syntaxe en Python

création d'un dictionnaire vide	`d = {}`
ajout d'une case	`d[nv_cle] = valeur`
modification d'une case	`d[cle] = nv_valeur`
suppression d'une case	`del d[cle]`
longueur	`len(d)`
parcours des clés d'un dictionnaire	`for x in d :`
tester si une clé existe	`cle in d`
tableau des clés d'un dictionnaire	`d.keys()`
tableau des valeurs d'un dictionnaire	`d.values()`
tableau des couples (clé,valeur) d'un dictionnaire	`d.items()`

1.3 Remarques supplémentaires sur les dictionnaires en Python

— Il existe un transtypage des tableaux de couples vers les dictionnaires.
Par exemple, si on a un tableau `t = [("a",1),("b",2),("c",3)]`, `dict(t)` créera le dictionnaire `{'a': 1, 'b': 2, 'c': 3}`.

— Les clés ne peuvent pas prendre des types mutables : les tuples sont autorisés, mais pas les tableaux.

— Les clés peuvent être autre chose que des `string`, et peuvent être de différents types, tout comme les valeurs.

2 Dictionnaires et tables de hachage

Pour implémenter dans la machine les dictionnaires, Python a recours à ce qu'on appelle des tables de hachage.

2.1 Spécification du problème à résoudre

Pour implémenter des dictionnaires, on voudrait :

— pouvoir ajouter une case ;

— pouvoir modifier une case ;

— pouvoir accéder à une case ;

— pouvoir supprimer une case.

— tout cela avec des clés qui ne sont pas des entiers.

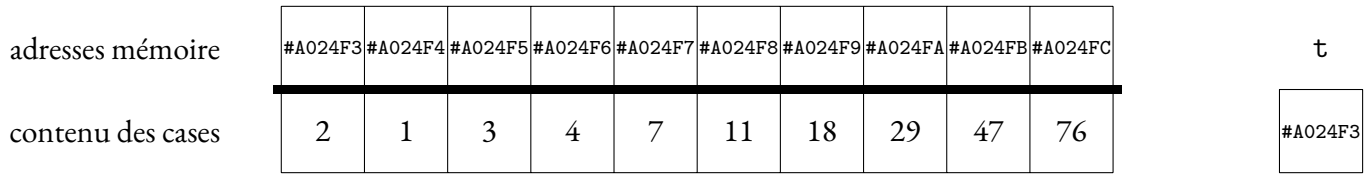
2.2 Et dans le cas des tableaux ?

Cette explication est valide dans le langage C, et tient, dans l'esprit, pour le langage Python.

L'implémentation des tableaux en machine repose sur le fonctionnement de la mémoire. Chaque case mémoire d'une machine dispose d'une adresse, qui est un entier (typiquement sur 64 bits). Un tableau `t` est alors en fait l'adresse mémoire d'une case particulière (par exemple `#A024F3`).

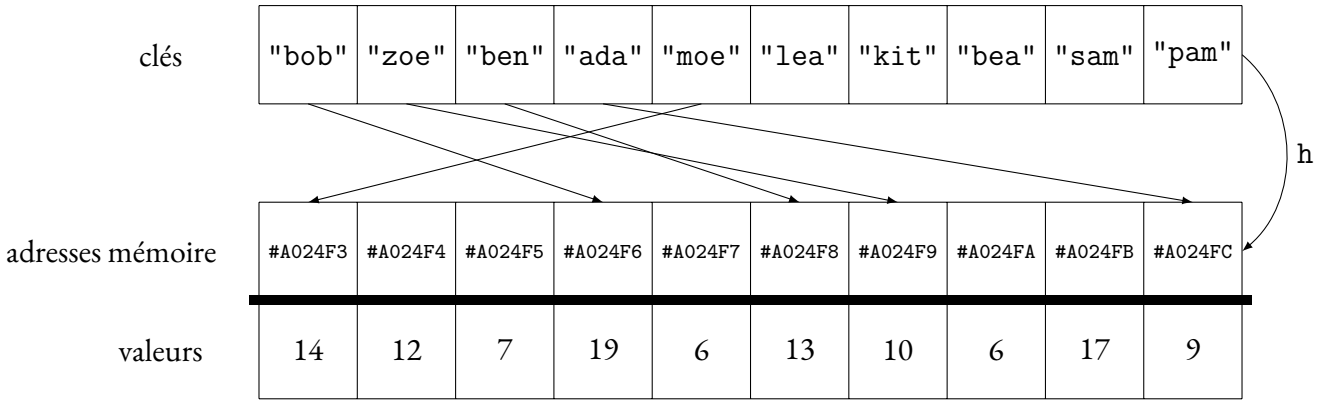
Pour obtenir *le contenu* d'une case du tableau, on demande alors `t[0]` : la machine comprend qu'elle doit donner le contenu situé à la case dont l'adresse mémoire vaut `#A024F3` + 0. De manière générale, si on demande `t[k]`, la machine renvoie le contenu situé à la case d'adresse `#A024F3` + `k`. Pour connaître la case à manipuler, la machine fait une simple addition : c'est ce qu'on appelle l'arithmétique des pointeurs.

Cette technique ne peut pas marcher directement dans le cas des dictionnaires, car il n'est pas possible de faire de l'arithmétique avec tout type de clé possible et imaginable.



2.3 Tables de hachage

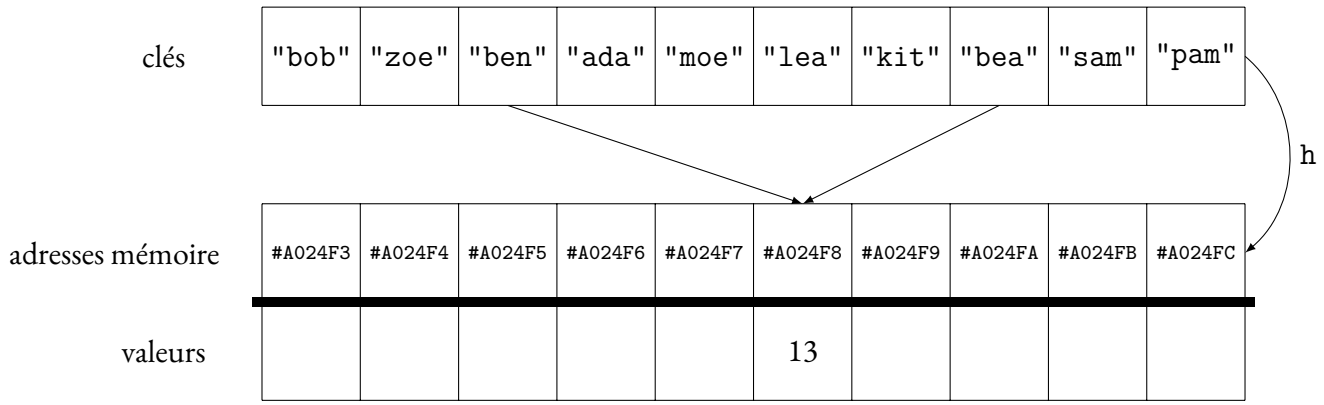
Pour implémenter un dictionnaire, une idée est alors de tout de même considérer un dictionnaire comme un tableau, en construisant une fonction qui transforme les clés en entiers :



La fonction `h` est appelée **fonction de hachage**. Pour que nos calculs soient efficaces, on a besoin que le calcul d'une image par `h` se fasse en $O(1)$.

2.4 Difficultés autour des fonctions de hachage

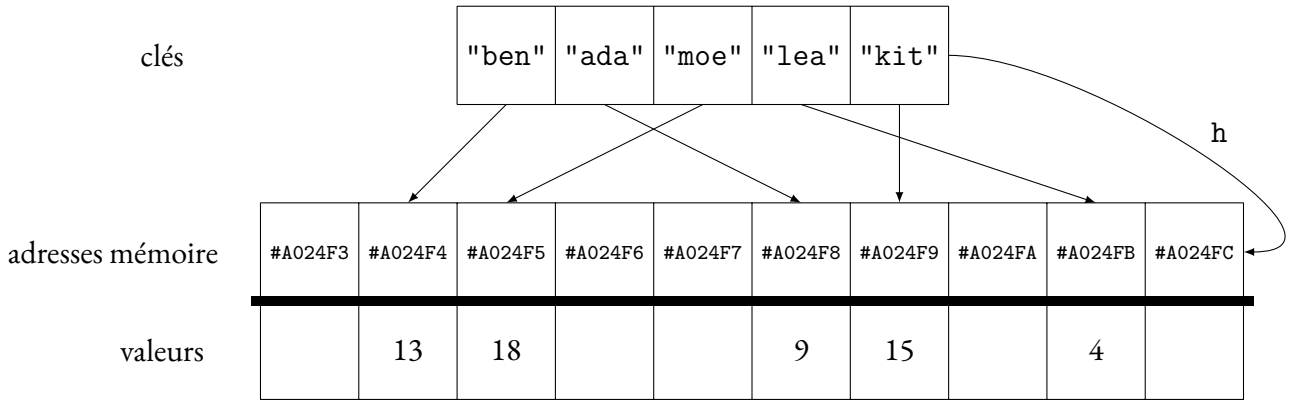
Générer des fonctions de hachage est une tâche très difficile, car il faut remplir beaucoup de contraintes en le moins de temps possible. En fait, c'est une tâche *trop* difficile, et les fonctions de hachage peuvent ne pas être injectives.



Dans ce cas, notre implémentation d'un dictionnaire est ratée : nos deux clés n'ont pas des résultats indépendants.

2.4.1 Première solution : avoir beaucoup plus de cases remplissables que de clés

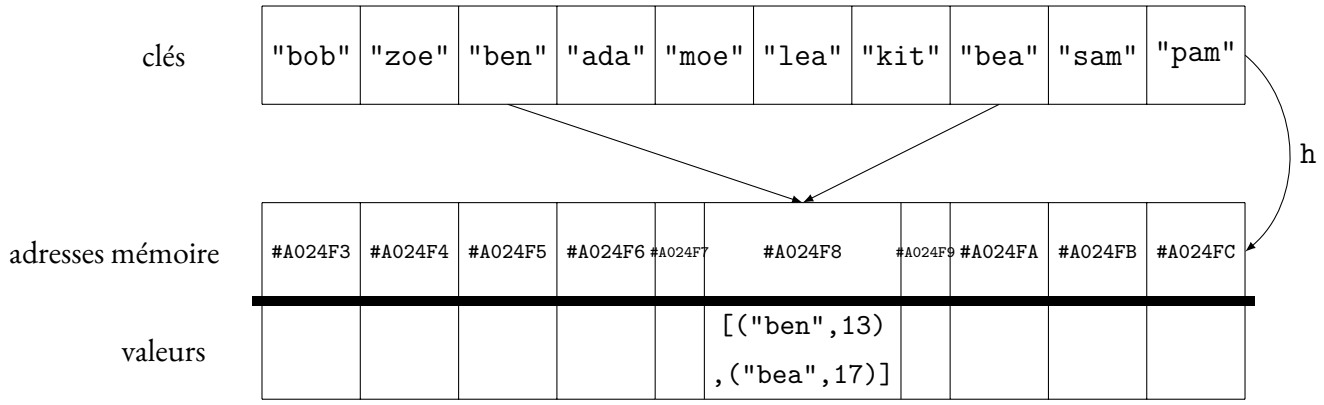
On peut décorréler l'espace des clés et l'espace mémoire alloué au dictionnaire pour qu'ils n'aient pas la même taille.



On diminue alors la probabilité de collisions. Mieux encore, si on a créé une collision, on peut, en peu de temps, générer une nouvelle fonction de hachage qui n'aura pas de collision. En notant `n` le nombre de clés et `m` le nombre de cases mémoire réservées, on appelle facteur de charge le rapport `n/m`. Plus le facteur de charge est important, plus la mémoire est utilisée efficacement, mais plus il y a de risques de collision. En pratique, on choisit généralement un facteur de charge de ~ 0.75.

2.4.2 Deuxième solution : stocker des listes

Plutôt que de ne stocker qu'une seule valeur par case, on peut stocker une liste de valeurs : s'il y a collision, on a alors les différents résultats possibles.



On peut se dire qu'on a déplacé le problème de l'implémentation des dictionnaires un cran plus bas. L'avantage est que, si notre fonction de hachage n'est pas trop mauvaise, on a considérablement réduit la taille des nouvelles tables de hachage à construire.

2.4.3 En pratique

En pratique, les deux solutions sont combinées : on a à la fois plus d'espace que nécessaire, et de la gestion de collision par sous-tables de hachage.

Concernant les fonctionnalités à implémenter :

— l'ajout de cases à un dictionnaire se fait en ajoutant des cases mémoire allouées, et en modifiant la fonction de hachage ;

— la suppression de cases se fait en modifiant la fonction de hachage. Si on a alloué « trop » d'espace mémoire, on peut aussi désallouer de la mémoire. Il faut cependant faire attention à des phénomènes de seuil.