Le Thu, 09 Jul 2020 13:25:10 +0200,

"Thirot Jean-Louis" [<Jean-Louis.Thirot@ac-rennes.fr>](mailto:Jean-Louis.Thirot@ac-rennes.fr) a écrit :

le problème est que je suis paumé !je n'ai pas compris ce qu'on fait de

cette fiche phrase : Distinguer la recherche d’une valeur dans une

liste et dans un dictionnaire.

Bonjour Jean-Louis,

personnellement, j'aurai présenté ceci non pas comme une recherche, mais

comme une lecture (même si au final, il y a une forme de recherche):

Distinguer la lecture d'une valeur dans une liste et dans un dictionnaire.

\* Dans le cas d'une liste à base de tableau, la lecture est immédiate

grâce à l'indice : L[i] , avec une complexité d'accès en O(1).

\* Dans le cas d'une liste chaînée, la lecture est séquentielle puisqu'il

faut parcourir la liste depuis le début pour atteindre la position de

l'élément contenant la valeur à lire , avec une complexité en O(n).

\* Dans le cas d'un dictionnaire, la lecture se fait en deux étapes :

le condensé de la clé (hash), une fois calculé, permet de déterminer le

triplet (hash, référence clé, référence valeur) puis de lire la valeur.

Dans l'idéal, ce triplet est obtenu immédiatement par le hash qui sert

d'indice dans la table de hachage. Dans le pire des cas lorsqu'il y a

beaucoup de collisions (et que la fonction de hachage n'est pas bonne),

le triplet est reporté ailleurs... On a ainsi une complexité qui varie

entre O(1) et O(n).

J'ai eu un doute à la lecture de ton message.  
  
Non pas tout à fait : si tu as des insertions "régulières" (avec tout le flou que cela suppose), la dichotomie va t'obliger :  
- soit à re-trier à chaque fois ta liste  
- soit si c'est implémenté par un tableau indexé, à effectuer une dichotomie pour l'insertion (rapide) puis à décaler n/2 termes en moyennes pour insérer à proprement parler (pas rapide)  
- soit si c'est implémenté par une liste chaînée à effectuer un parcours en O(n) pour insérer au bon endroit.  
  
Dans ce cas l'arbreBR équilibré est un meilleur compromis (comme mentionné au préalable, cette analyse de complexité n'a de sens que sur des grosses structures, sinon privilégier la compréhension du code :-)  
(Et la table de hash est encore meilleure du point de vue complexité)  
  
JdV

je pense qu'on est tous en accord sur le sens de cette phrase.

merci à tout le monde

reste que la recherche dans la liste de tuple se fait en O(1) et là je suis perplexe.

Le 09/07/20, **Julien de Villèle** <julien.de-villele@ac-orleans-tours.fr> a écrit :

|  |  |
| --- | --- |
| Dans le Froidevaux-Godel-Soria au chapitre 9 le terme "*recherche*" est assimilé à "*recherche associative*". On y considère donc qu'une recherche est la recherche d'une valeur associée à une clef donnée. Le premier exemple qui y est donné est celui d'une liste de 2-uplets où la clef est l'élément d'indice 0 de chaque 2-uplet.  Donc l'exemple du bouquin est cohérent si on retient cette interprétation de *recherche* comme *recherche associative*.  Le 09/07/2020 à 13:25, Thirot Jean-Louis a écrit :  le problème est que je suis paumé !  je n'ai pas compris ce qu'on fait de cette fiche phrase : *Distinguer la recherche d’une valeur dans une liste et dans un dictionnaire.*  déjà : cherche une valeur dans un dict c'est pas si trivial. un truc du genre :  def find\_key(v,d):      for k, val in d.items():          if v == val:              return k      return "Clé n'existe pas"  ou [c for c,val in d.items() if val==v] (mais c'est kifkif)  sauf que justement, ça c'est bien moins efficace qu'un bête lst.index(v)  du coup, le truc c'est le nombre de page du bouquin comme a proposé Guillaume.  ok, on a un dico : {livre1:nbpages1,livre2:nbpages2 etc...}  et on va vite trouver le nombre de pages du bouquin souhaité  mais on compare à quoi ? parce que la représentation en liste de cette situation c'est quoi ? une liste de tuple (bouquin,nbpages) ? alors oui c'est sur ça va pas rigoler pour trouver le bon bouquin.....  for elem in lst :      if elem[0]==bouquin:         return elem[1]  à comparer avec  dico[bouquin]  J'ai capté ?  Le 09/07/20, **Antoine Meyer** [<antoine.meyer@u-pem.fr>](mailto:antoine.meyer@u-pem.fr) a écrit :   |  | | --- | | Effectivement ce n'est pas limpide. Ces histoires de listes auraient bien besoin d'une clarification officielle (je ne sais pas si ça se fait).  Voilà une autre façon, plus élémentaire, d'entendre ce bout du programme, où il n'est pas question *du tout* de types abstraits, et seulement un peu de complexité. Je vise plutôt la compréhension basique des types **concrets** list et dict en Python, dans l'espoir d'amener les élèves à utiliser l'un ou l'autre à bon escient.  Si vos élèves maîtrisent parfaitement les trois points suivants après la terminale, c'est déjà pas mal sur ce sujet-là à mon avis.  Mes deux cents.  --  Antoine  1) Chercher une **valeur** dans une liste quelconque (entendre : non triée) demande toujours un parcours exhaustif. Il y a plein de façons de l'écrire, toutes en O(n).  Syntaxes / techniques à enseigner (parmi d'autres) :  elem in lst  # est-ce dedans ?  lst.index(elem)  # où est-ce ?  for e in lst:  # parcours à la main      if e == elem: ...  (Note : dans le chapitre algo on aborde la recherche par dichotomie sur une liste triée, c'est donc assez cohérent de bien faire maîtriser la recherche exhaustive avant...)  (Re-note : si vous utilisez une liste chaînée plutôt qu'un tableau, c'est kif-kif, la recherche demande un parcours exhaustif, donc O(n) itou.)  2) Chercher une **clé** particulière dans un dictionnaire est en temps *presque* constant (comprendre : O(1) en "moyenne" mais pas au pire) grâce à la magie des tables de hachage : on calcule le hash, on cherche à l'emplacement correspondant dans la table, et sauf malchance c'est fini.  Syntaxes à enseigner :  clé in dico  # est-ce dedans ?  dico[cle]  # valeur associée ?  dico.get(cle, defaut)  # pour les cadors  3) Chercher une **valeur** dans un dictionnaire, outre que c'est assez rarement utile, revient à itérer sur le dictionnaire tout entier, comme sur une bête liste, et donc ça ne change rien ou pas grand chose.  Techniques possibles :  elem in dico.values()  # est-ce dedans ?  for v in dico.values():  # parcours à la main      if v == elem:          ... | |