Παρουσίαση για χρήση με το σύγγραμμα, Αλγόριθμοι Σχεδίαση και Εφαρμογές, των M. T. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Κατακερματισμός κούκου

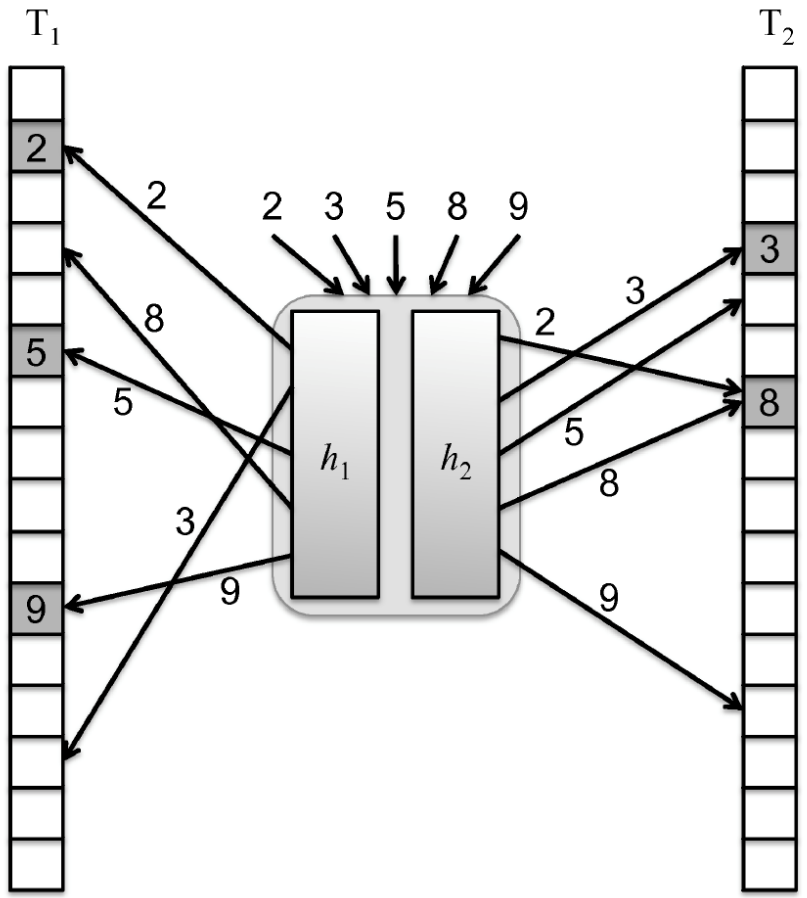
xkcd. http://xkcd.com/1065/. "Shoes." Used with permission under Creative Commons 2.5 License.

Η δύναμη των δύο επιλογών

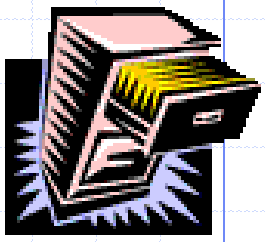
- Στην τεχνική του κατακερματισμού κούκου, χρησιμοποιούμε δύο πίνακες αναζήτησης T_1 and T_2, μεγέθους N, όπου το N είναι μεγαλύτερο από το πλήθος των στοιχείων του πίνακα κατακερματισμού n, κατά τουλάχιστον έναν σταθερό συντελεστή, όπως $N \geq 2n$.
- Χρησιμοποιούμε μία συνάρτηση κατακερματισμού, h_1, για το T_1, και μία άλλη συνάρτηση κατακερματισμού, h_2, για το T_2.
- Για οποιοδήποτε κλειδί, k, υπάρχουν μόνο **δύο πιθανές θέσεις** όπου επιτρέπεται να αποθηκευτεί, είτε στο $T_1[h_1(k)]$ είτε στο $T_2[h_2(k)]$.
- Αυτή η ελευθερία μας επιτρέπει να επιτύχουμε χρόνο O(1) **στη χειρότερη περίπτωση** για αναζήτηση στους πίνακες.

Παράδειγμα κατακερματισμού κούκου

- Κάθε κλειδί από το σύνολο S = {2, 3, 5, 8, 9} έχει δύο πιθανές θέσεις στις οποίες μπορεί να τοποθετηθεί, μία στον πίνακα T_1 και μία στον πίνακα T_2.
- Παρατηρείστε ότι υπάρχει σύγκρουση του 2 και του 8 στον T_2, αλλά δεν υπάρχει πρόβλημα, από την στιγμή που δεν υπάρχει σύγκρουση για το 2 στην εναλλακτική τοποθεσία του στον T_1.



Θυμηθείτε τις λειτουργίες του χάρτη



- get(k): Αν ο χάρτης M περιέχει ένα στοιχείο με κλειδί ίσο με k επιστρέφει την τιμή του, αλλιώς επιστρέφει null
- put(k, v): εισαγωγή στοιχείου (k, v) στον χάρτη M, εάν το κλειδί k δεν υπάρχει στον M επιστρέφει null, αλλιώς επιστρέφει την προηγούμενη τιμή στην οποία αντιστοιχούσε το k
- remove(k): εάν ο χάρτης M έχει ένα στοιχείο με κλειδί k, αφαίρεση του από τον M και επιστρέφει την τιμή του, αλλιώς επιστρέφει null
- size(), isEmpty()

Ψευδό-κώδικας για get και remove

- Οι αλγόριθμοι get και remove είναι απλοί και O(1) στη χειρότερη δυνατή περίπτωση.

- get(k):

  if $T_0[h_0(k)] \neq$ NULL **and** $T_0[h_0(k)]$.key $= k$ **then**
  return $T_0[h_0(k)]$
  if $T_1[h_1(k)] \neq$ NULL **and** $T_1[h_1(k)]$.key $= k$ **then**
  return $T_1[h_1(k)]$
  return NULL

- remove(k):

  if $T_0[h_0(k)] \neq$ NULL **and** $T_0[h_0(k)]$.key $= k$ **then**
  $temp \leftarrow T_0[h_0(k)]$
  $T_0[h_0(k)] \leftarrow$ NULL
  return $temp$
  if $T_1[h_1(k)] \neq$ NULL **and** $T_1[h_1(k)]$.key $= k$ **then**
  $temp \leftarrow T_1[h_1(k)]$
  $T_1[h_1(k)] \leftarrow$ NULL
  return $temp$
  return NULL

Έμπνευση για το όνομα

- Η ονομασία "κατακερματισμός κούκου" προκύπτει από τον τρόπο που εκτελείται η πράξη put(k, v) σ' αυτήν την τεχνική, επειδή μιμείται τις συνήθειες αναπαραγωγής του κούκου.
- Ο κούκος είναι γνωστός για το φαινόμενο του αναπαραγωγικού παρασιτισμού – γεννά τα αβγά του στις φωλιές άλλων πουλιών αφού απομακρύνει ένα αβγό απ' τη συγκεκριμένη φωλιά.

Catesby, Cockoo of Carolina. The bird that is. 18th Century color illustration. Early American bird print. Catesby. Scan of 2 d images in the public domain believed to be free to use without restriction in the US.

Έμπνευση για το όνομα - put

- Ομοίως, αν προκύψει σύγκρουση στην πράξη εισαγωγής στην τεχνική του κατακερματισμού κούκου, τότε απομακρύνουμε το προηγούμενο στοιχείο που υπάρχει στο συγκεκριμένο κελί και εισάγουμε το νέο στη θέση του.
- Αυτό αναγκάζει το στοιχείο που απομακρύνεται να μεταβεί στην εναλλακτική θέση του στον άλλο πίνακα και να τοποθετηθεί εκεί, ενδεχομένως επαναλαμβάνοντας τη διαδικασία απομάκρυνσης με άλλο στοιχείο κοκ.
- Τελικά, είτε βρίσκουμε ένα κενό κελί και σταματάμε, είτε επαναλαμβάνουμε μια προηγούμενη απομάκρυνση, κάτι που υποδεικνύει ότι συντελείται κύκλος απομακρύνσεων.
- Αν ανακαλύψουμε κάτι τέτοιο, τότε διακόπτουμε τη διαδικασία εισαγωγής και επανακατακερματίζουμε όλα τα στοιχεία στους δύο πίνακες χρησιμοποιώντας νέες και, ευελπιστούμε καλύτερες συναρτήσεις κατακερματισμού.

Ψευδό-κώδικας για put

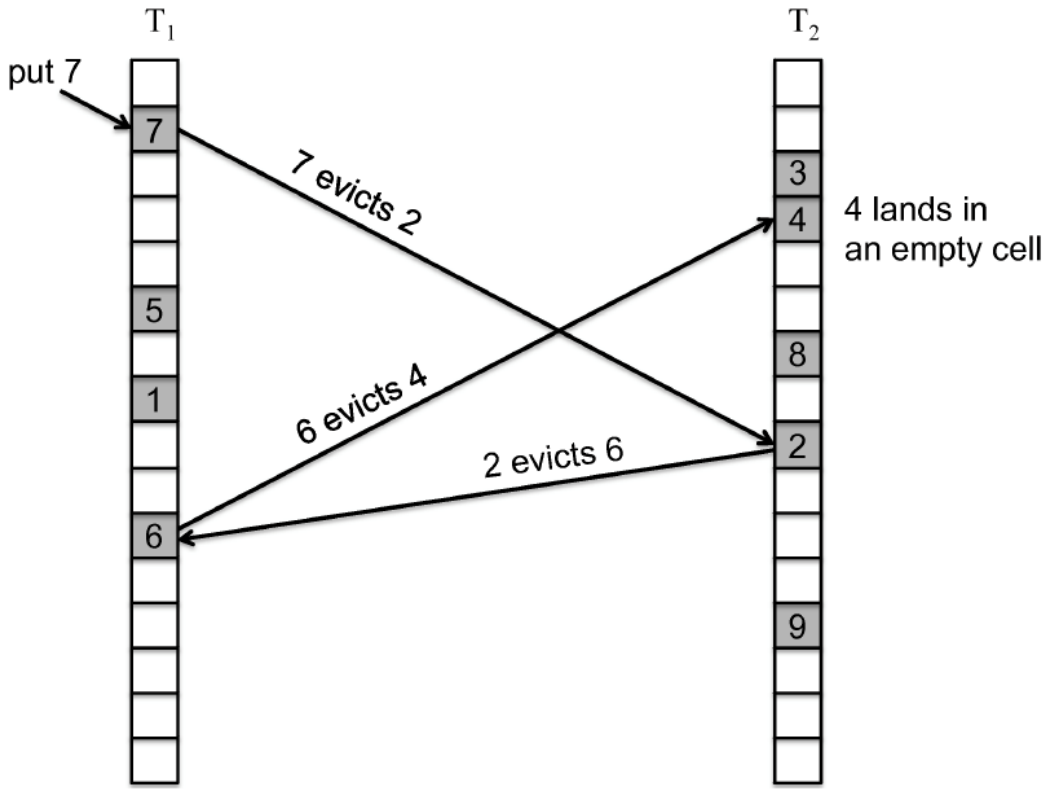
- put(k, v):

if $T_0[h_0(k)] \neq$ NULL **and** $T_0[h_0(k)]$.key $= k$ **then**
  $T_0[h_0(k)] \leftarrow (k, v)$
  return
if $T_1[h_1(k)] \neq$ NULL **and** $T_1[h_1(k)]$.key $= k$ **then**
  $T_1[h_1(k)] \leftarrow (k, v)$
  return
$i \leftarrow 0$
repeat
  if $T_i[h_i(k)] =$ NULL **then**
    $T_i[h_i(k)] \leftarrow (k, v)$
    return
  $temp \leftarrow T_i[h_i(k)]$
  $T_i[h_i(k)] \leftarrow (k, v)$ // cuckoo eviction
  $(k, v) \leftarrow temp$
  $i \leftarrow (i + 1) \bmod 2$
until a cycle occurs
Rehash all the items, plus (k, v), using new hash functions, h_0 and h_1.

- Σημειώστε ότι ο παραπάνω ψευδό-κώδικας έχει μία συνθήκη, που εντοπίζει την ύπαρξη κύκλου στην σειρά εισαγωγών. Υπάρχουν πολλοί τρόποι διαμόρφωσης αυτής της συνθήκης. Για παράδειγμα, θα μπορούσαμε να μετρήσουμε τον αριθμό των επαναλήψεων γι' αυτό το βρόχο και να εξετάσουμε αν υπάρχει κύκλος αν ξεπεράσουμε ένα συγκεκριμένο όριο όπως n ή log n.

Παράδειγμα σειράς απομακρύνσεων

- Μία σειρά απομακρύνσεων μήκους 3:



Οι μεγάλες σειρές απομακρύνσεων είναι σπάνιες

Lemma 6.1: *The probability that there is a possible sequence of evictions of length L between a cell, x, and a cell, y, in $T_0 \cup T_1$, is at most $1/(2^L N)$.*

Proof: The proof is by induction. For the base case, $L = 1$, note that there is a possible length-1 eviction sequence between x and y if and only if we choose x and y as the two possible locations for some item. The hash functions h_0 and h_1 effectively choose 1 cell each out of N possible cells in T_0 and T_1, respectively. Thus, the probability that a particular item chooses both x and y as its two locations is at most $1/N^2$, and the probability that ***any*** of the n items in our set, S, chooses both x and y is at most

$$\sum_{(k,v)\in S} \frac{1}{N^2} = \frac{n}{N^2} \leq \frac{1}{2} \cdot \frac{1}{N},$$

because of our assumption that $N \geq 2n$. Note that the probability that there is a possible eviction sequence of length 1 between x and y is 0 if both x and y are in the same table, T_i, but this probability is clearly bounded by $1/(2N)$. So this completes the proof for the base case, $L = 1$.

Οι μεγάλες σειρές απομακρύνσεων είναι σπάνιες

For the inductive step, $L \geq 2$, let us assume the claim is true for possible eviction sequences of length $(L-1)$. For there to be a length-L eviction sequence from x to y, there has to be a possible length-$(L-1)$ eviction sequence between x and some cell, z, and a possible length-1 eviction sequence between z and y. By induction, for a given cell, z, the probability there is a length-$(L-1)$ eviction sequence between x and z is at most $1/(2^{L-1}N)$. Likewise, from the above discussion for the base case, there is a length-1 eviction sequence between z and y with probability at most $1/(2N)$. Thus, the probability that there is a length-L eviction sequence between x and y is at most

$$\begin{aligned}\sum_z \frac{1}{2^{L-1}N} \cdot \frac{1}{2N} &= \sum_z \frac{1}{2^L N^2} \\ &\leq \frac{N}{2^L N^2} \\ &= \frac{1}{2^L N},\end{aligned}$$

since there are only N candidates for z, because it has to be in either T_0 or T_1, depending, respectively, on whether y is in T_1 or T_0. ■

Ο αναμενόμενος αριθμός των πιθανών απομακρύνσεων

Say that two keys k and l are in the same "bucket" if there is a sequence of evictions (of any length) between a possible cell for k and a possible cell for l. Thus, for k and l to be in the same bucket, there has to be an eviction sequence between one of the cells $T_0[h_0(k)]$ or $T_1[h_1(k)]$ and one of the cells $T_0[h_0(l)]$ or $T_1[h_1(l)]$. Note that there are 4 such possible sequences, depending on where we stop and end, since we are ignoring the direction that a sequence of evictions can take. Then, by Lemma 6.1, and summing across all possible lengths, the probability that k and l are in the same bucket is bounded by

$$4\sum_{L=1}^{\infty}\frac{1}{2^L N} = \frac{4}{N}\sum_{L=1}^{\infty}\frac{1}{2^L}$$
$$= \frac{4}{N}.$$

Note that, by this notion of a "bucket," the running time for performing an insertion in a cuckoo table is certainly bounded by the number of items that map to the same bucket as the item being inserted, so long as we don't cause a rehash. Thus, the expected time to insert an item with key k in this case is bounded by

$$\sum_{\text{keys in } S}\frac{4}{N} = 4n/N \leq 2.$$

In other words, the expected running time for performing a put(k, v) operation for a cuckoo table is $O(1)$, assuming this insertion does not cause a rehash.

Η απόδοση του κατακερματισμού κούκου

- Μπορούμε να αποδείξουμε ότι από τη στιγμή που οι μεγάλες σειρές απομακρύνσεων είναι σπάνιες η πιθανότητα να χρειαστεί επανακατακερματισμός είναι πολύ μικρή.
- Έτσι, ο επιμερισμένος χρόνος για την εισαγωγή μίας εγγραφής στον πίνακα κούκου είναι O(1).
- Έτσι ο πίνακας κατακερματισμού κούκου επιτυγχάνει χρόνο O(1) για αναζητήσεις και διαγραφές και O(1) για εισαγωγές.
- Αυτό συμβαίνει κυρίως επειδή για οποιοδήποτε στοιχείο υπάρχουν ακριβώς δύο πιθανές θέσεις στις οποίες μπορεί να βρίσκεται ένα στοιχείο σε έναν πίνακα κατακερματισμού, γεγονός που δείχνει τη δύναμη της ύπαρξης δύο επιλογών.