Στάθης Ζάχος Άρης Παγουρτζής

Συναρτήσεις κατακερματισμού (hashing) και δομές λεξικού (dictionary)

(Σημειώσεις)

Εθνικό Μετσόβιο Πολυτεχνείο

Αθήνα 2024

Κεφάλαιο 1

Κατακερματισμός (hashing) και δομή λεξικού (dictionary)

Ας υποθέσουμε ότι έχουμε ένα σύνολο στοιχείων K και θέλουμε να μπορούμε να καταγράφουμε ποια στοιχεία από το K εμφανίζονται σε κάποια δεδομένα εισόδου. Για παράδειγμα, θέλουμε να καταγράψουμε και οργανώσουμε τις λέξεις που εμφανίζονται σε ένα κείμενο, ώστε να μπορούμε να απαντάμε γρήγορα αν κάποια λέξη υπάρχει στο κείμενο ή όχι. Χρειαζόμαστε επομένως μια δομή δεδομένων που να υποστηρίζει αποδοτικά τις λειτουργίες της εισαγωγής στοιχείου (insert) και της εύρεσης στοιχείου (find). Συνήθως (αλλά όχι πάντα) θέλουμε να υποστηρίζεται και η διαδικασία διαγραφής στοιχείου (delete), για παράδειγμα αν κάποια λέξη στο κείμενο που επεξεργαζόμαστε ήταν ορθογραφικά λανθασμένη. Ας σημειωθεί ότι αν οι διαγραφές είναι σπάνιες (όπως συμβαίνει σε πολλές εφαρμογές), η αποδοτικότητα της λειτουργίας αυτής δεν είναι τόσο κρίσιμη.

Ορισμός 1.1. Μια αφηρημένη δομή δεδομένων (ADT) που υποστηρίζει αποδοτικά μία ακολουθία από διαδικασίες Insert, Delete και Find ονομάζεται **λεξικό** (*dictionary*).

Αναφερόμαστε στο σύνολο K ως **σύνολο κλειδιών**, καθώς συχνά αντί να αποθηκεύουμε το ίδιο το στοιχείο, προτιμούμε να χρησιμοποιούμε ένα αναγνωριστικό ή *κλειδί* που προσδιορίζει μοναδικά το στοιχείο.

Υποθέτουμε ότι το πλήθος των στοιχείων που θα εμφανιστούν στην είσοδο είναι $n << |K|$, οπότε η χρήση ενός πίνακα με $|K|$ στοιχεία θα ήταν εξαιρετικά "σπάταλη" σε μνήμη ή ακόμη και πρακτικά ανέφικτη. Ο στόχος είναι να χρησιμοποιηθεί χώρος μνήμης μεγέθους $O(n)$. Θα μπορούσε βέβαια να χρησιμοποιηθεί μια γραμμική δομή (όπως μία συνδεδεμένη λίστα), όμως ο χρόνος των βασικών λειτουργιών θα ήταν $O(n)$, που δεν είναι ικανοποιητικός για μεγάλες τιμές του n. Μια πιο κατάλληλη δομή θα ήταν η χρήση ενός ισορροπημένου δέντρου δυαδικής αναζήτησης, που θα εξασφάλιζε χρόνους εισαγωγής, αναζήτησης, και διαγραφής $O(\log n)$. Όπως θα δούμε στη συνέχεια, μπορούμε να εξασφαλίσουμε πρακτικά χρόνο $O(1)$,

δηλαδή σταθερό, με χρήση *συναρτήσεων κατακερματισμού*:
Η υλοποίηση ενός λεξικού μπορεί να γίνει με μια συνάρτηση[1]

$$h\colon K \to [m],$$

που ονομάζεται **συνάρτηση κατακερματισμού** (*hash function*). Συχνά χρησιμοποιείται και ο όρος **συνάρτηση σύνοψης** (*digest*). Φροντίζουμε ώστε η συνάρτηση h που διαλέγουμε να υπολογίζεται γρήγορα για οποιοδήποτε στοιχείο του συνόλου K, σε χρόνο δηλαδή $O(1)$. Οι συναρτήσεις κατακερματισμού διακρίνονται σε δύο μεγάλες κατηγορίες: κλειστής διευθυνσιοδότησης και ανοιχτής διευθυνσιοδότησης, τις οποίες θα παρουσιάσουμε πιο αναλυτικά στη συνέχεια.

Κλειστή διευθυνσιοδότηση (closed addressing)

Μέθοδος διαίρεσης και αλυσίδωση

Τυπικό παράδειγμα συνάρτησης κατακερματισμού είναι η συνάρτηση mod (όπου υποθέτουμε ότι $K \subseteq \mathbb{N}$):

$$\forall k \in K: \quad h(k) = k \bmod m \tag{1.1}$$

Ας σημειωθεί ότι η συνάρτηση αυτή κατακερματίζει *ομοιόμορφα* το σύνολο K: αν $K = \{1, \ldots, N\}$ τότε σε κάθε τιμή του πεδίου τιμών απεικονίζονται περίπου $\frac{N}{m} = \frac{|K|}{m}$ στοιχεία του K.
Άλλα παραδείγματα είναι:

(i) η πιο γενική μορφή της μεθόδου διαίρεσης, όπου επιλέγονται 2 αριθμοί $a, b < m$ με $\gcd(a, m) = 1$ και η συνάρτηση h ορίζεται ως εξής:

$$h(k) = a \cdot k + b \bmod m$$

(η επιλογή του a να είναι σχετικά πρώτος με τον m είναι σημαντική ώστε να η συνάρτηση να κατακερματίζει "ομοιόμορφα" το σύνολο K: δύο κλειδιά k_1, k_2 έχουν την ίδια εικόνα ανν έχουν το ίδιο υπόλοιπο $\bmod m$, όπως και στην απλή μορφή της μεθόδου)

(ii) η μέθοδος του πολλαπλασιασμού όπου χρησιμοποιείται το μήκος λέξης του υπολογιστή, έστω w, και ένας πολλαπλασιαστής $a < 2^w$ (a περιττός αριθμός) και η συνάρτηση ορίζεται ως εξής:

$$h(k) = (a \cdot k \bmod 2^w) \operatorname{div} 2^{w-r}$$

[1] Χρησιμοποιούμε τον συμβολισμό $[m] = \{0, \ldots, m-1\}$ και $[m]_+ = \{1, \ldots, m-1\}$ (για φυσικό αριθμό $m > 0$).

δηλαδή αρχικά κρατούνται τα w λιγότερο σημαντικά bits του γινομένου (ώστε να χωρούν σε μια λέξη του υπολογιστή) και από αυτά κρατούνται τα r περισσότερο σημαντικά. Η συνάρτηση αυτή θεωρείται ότι έχει καλή συμπεριφορά στην πράξη και πολύ αποδοτική υλοποίηση. Συναντιέται και στη μορφή

$$h(k) = ((a \cdot k + b) \text{ mod } 2^w) \text{ div } 2^{w-r},$$

όπου $b < 2^w$. Η μορφή αυτή θεωρείται ιδιαίτερα κατάλληλη για τον ορισμό μιας οικογένειας συναρτήσεων κατακερματισμού $h_{a,b}$ που παραμετροποιείται ως προς a και b και έχει (προσεγγιστικά) την ιδιότητα της *καθολικότητας* που θα ορίσουμε αμέσως παρακάτω.

Καθολικός κατακερματισμός (universal hashing). Οι παραπάνω συναρτήσεις, παρ'όλο που έχουν καλές ιδιότητες ομοιομορφίας, είναι δυνατό να δημιουργήσουν μεγάλες συσσωρεύσεις σε συγκεκριμένες θέσεις του πεδίου τιμών αν η κατανομή εισόδου δεν είναι ομοιόμορφη. Για παράδειγμα, αν χρησιμοποιούμε την μέθοδο της διαίρεσης και στην είσοδο έρχονται πάντα κλειδιά που είναι ισότιμα modulo m τότε όλα τα κλειδιά θα απεικονίζονται στην ίδια τιμή, οδηγώντας σε χρόνους αναζήτησης και εισαγωγής τάξης $O(n)$. Ιδανικά θα θέλαμε κάθε κλειδί να απεικονίζεται σε οποιαδήποτε θέση του πεδίου τιμών με πιθανότητα $1/m$, αυτό όμως είναι αδύνατο, δεδομένου ότι ένα βασικό χαρακτηριστικό των συναρτήσεων κατακερματισμού είναι να απεικονίζουν ένα κλειδί στην ίδια πάντα τιμή. Έτσι, για οποιαδήποτε συνάρτηση κατακερματισμού μπορεί ένας 'αντίπαλος' (adversary) να κατασκευάσει μια 'κακή' ακολουθία εισόδου, όπου όλα τα κλειδιά να απεικονίζονται στην ίδια τιμή. Η λύση που προτείνεται για να ξεπεραστεί αυτό το πρόβλημα είναι η τυχαία επιλογή συνάρτησης κατακερματισμού από μια οικογένεια $\mathcal{H}$ που να ικανοποιεί την παρακάτω ιδιότητα.

Ορισμός 1.2. Μια οικογένεια συναρτήσεων $\mathcal{H} : K \to [m]$ λέγεται *καθολική οικογένεια κατακερματισμού* (universal hash family), ή απλώς *καθολική*, αν για δύο οποιαδήποτε διαφορετικά στοιχεία k, k' του συνόλου κλειδιών K, μία τυχαία και ομοιόμορφα επιλεγμένη συνάρτηση της οικογένειας έχει πιθανότητα $\leq 1/m$ να τα απεικονίσει στην ίδια τιμή. Ισοδύναμα,

$$\forall k, k' \in K, |\{h \in \mathcal{H} \mid h(k) = h(k')\}| \leq \frac{|\mathcal{H}|}{m}$$

δηλαδή, για οποιαδήποτε δύο κλειδιά k, k' υπάρχουν το πολύ $|\mathcal{H}|/m$ συναρτήσεις της οικογένειας που τα απεικονίζουν στην ίδια τιμή.

Σημείωση: η οικογένεια $\mathcal{H}$ συνήθως ορίζεται με βάση κάποιες παραμέτρους που παίρνουν τιμές από πεπερασμένα σύνολα, και επομένως είναι πεπερασμένη.

Παράδειγμα 1.1. Η οικογένεια $\mathcal{H} : K \to [m], \mathcal{H} = \bigcup_{a \in [m]_+} h_a$, όπου

$$h_a(k) = a \cdot k \text{ mod } m$$

δεν είναι καθολική, καθώς αν $k \neq k'$, αλλά $k \equiv k' \pmod{m}$ τότε, για οποιοδήποτε a θα ισχύει πάντοτε $h_a(k) = h_a(k')$, επομένως η πιθανότητα να απεικονίζονται τα k, k' που έχουν την παραπάνω ιδιότητα στην ίδια τιμή είναι ίση με 1.

Παρατηρήστε ότι περίπτωση $a = 0$ εξαιρείται, καθώς μια τέτοια συνάρτηση θα απεικόνιζε *όλα* τα κλειδιά στην τιμή 0.

Με εντελώς ανάλογο τρόπο μπορεί να αποδειχθεί ότι η οικογένεια $\mathcal{H} : K \to [m], \mathcal{H} = \bigcup_{a \in [m]_+, b \in [m]} h_{a,b}$, όπου

$$h_{a,b}(k) = (a \cdot k + b) \bmod m$$

δεν είναι καθολική (άσκηση).

Παράδειγμα 1.2. Η οικογένεια $\mathcal{H} : K \to [m], \mathcal{H} = \bigcup_{a \in [m]_+, b \in [m]} h_{a,b}$, όπου

$$h_{a,b}(k) = ((a \cdot k + b) \bmod p) \bmod m$$

όπου $p > |K|$ είναι πρώτος αριθμός *είναι* καθολική. Η απόδειξη υπάρχει στην εργασία των Carter και Wegman [5] που πρότειναν την οικογένεια αυτή.

Παράδειγμα 1.3. Η οικογένεια $\mathcal{H} : [m^2] \to [m], \mathcal{H} = \bigcup_{a_H, a_L \in [m] \times [m]}, h_{a_H, a_L}$, με m πρώτο αριθμό, όπου αναπαριστώντας κάθε κλειδί $k \in [m^2]$ με μοναδικό τρόπο ως ζεύγος $k_H, k_L \in [m] \times [m] : k = k_H \cdot m + k_L$ ορίζουμε:

$$h_{a_H, a_L}(k) = a_H k_H + a_L k_L \bmod m$$

είναι καθολική, καθώς αν $k \neq k'$ θα διαφέρουν τουλάχιστον σε ένα από τα ζεύγη (k_H, k'_H), (k_L, k'_L). Χ.β.γ. έστω ότι $k_H \neq k'_H$. Τότε:

$$\begin{aligned} h_{a_H, a_L}(k) = h_{a_H, a_L}(k') \Leftrightarrow \quad & a_H k_H + a_L k_L \equiv a_H k'_H + a_L k'_L \pmod{m} \\ \Leftrightarrow \quad & a_H (k_H - k'_H) \equiv a_L (k'_L - k_L) \pmod{m} \\ \Leftrightarrow \quad & a_H \equiv a_L (k'_L - k_L)(k_H - k'_H)^{-1} \pmod{m} \end{aligned}$$

Επειδή $k_H \neq k'_H$ και m πρώτος αριθμός, η τιμή $(k_H - k'_H)^{-1} \pmod{m}$ ορίζεται και είναι μοναδική. Επομένως για κάθε $a_L \in [m]$ υπάρχει μοναδικό $a_H \in [m]$ ώστε τα k, k' να απεικονίζονται στην ίδια τιμή, άρα συνολικά υπάρχουν m συναρτήσεις από τις $|\mathcal{H}| = m^2$ για τις οποίες συμβαίνει αυτό, οπότε η πιθανότητα είναι $1/m$.

Σημείωση: ο ορισμός της οικογένειας μπορεί να επεκταθεί ώστε να έχει πεδίο ορισμού οποιοδήποτε σύνολο K, θεωρώντας $r \in \mathbb{N} : |K| \leq m^r$. Τότε μπορούμε να αναπαραστήσουμε μοναδικά κάθε κλειδί $k \in K$ με r αριθμούς από το $[m]$, έστω $k_0, \ldots, k_{r-1}$ που αντιστοιχούν στα 'ψηφία' του k στο m-αδικό σύστημα αρίθμησης. Επιλέγοντας τιμές $a_0, \ldots, a_{r-1} \in [m]$ και ορίζοντας

$$h_{a_0, \ldots, a_{r-1}}(k) = \Sigma_{i=0}^{r-1} a_i k_i \bmod m$$

προκύπτει μια καθολική οικογένεια κατακερματισμού $\mathcal{H} : K \to [m]$. Η απόδειξη είναι εντελώς ανάλογη και αφήνεται ως άσκηση.

Πίνακες κατακερματισμού και μέθοδος αλυσίδωσης. Στη συνέχεια θα θεωρούμε ότι έχει επιλεγεί μία συνάρτηση h από μια καθολική οικογένεια κατακερματισμού. Για την αποθήκευση των κλειδιών δημιουργούμε ένα πίνακα A (hash table). Κάθε στοιχείο $A[i]$ του πίνακα δείχνει σε μια λίστα η οποία περιέχει εκείνα τα κλειδιά $a \in K$ για τα οποία ισχύει $h(a) = i$. Συνεπώς για να κάνουμε *Insert*, *Delete* και *Find* ένα στοιχείο a, αρκεί να ψάξουμε μόνο τη λίστα στην οποία δείχνει το στοιχείο $A[h(a)]$.

Η μέθοδος αποθήκευσης στοιχείων με ίδια τιμή κατακερματισμού σε λίστα λέγεται **κατακερματισμός με αλυσίδωση** (*hashing with chaining*) και είναι αρκετά αποδοτικός όταν το πλήθος των εισαγόμενων στοιχείων n είναι ανάλογο του m, καθώς τότε το αναμενόμενο πλήθος κλειδιών ανα θέση του πίνακα είναι $O(1)$. Συχνά όμως ισχύει $n >> m$ και στη χειρότερη περίπτωση, είναι πιθανόν μετά από n εισαγωγές στοιχείων, να έχουμε μια λίστα μήκους σχεδόν n (δηλαδή σχεδόν όλα τα στοιχεία να έχουν πάει στην ίδια θέση του πίνακα). Σε αυτή την περίπτωση, αν χρειαστεί να εκτελέσουμε n φορές τις διαδικασίες *Delete* ή *Find*, ο χρόνος που απαιτείται είναι $O(n^2)$. Αν όμως φροντίσουμε ώστε η εκλογή της συνάρτησης h να εξασφαλίζει μια όσο το δυνατό ομοιόμορφη κατανομή των στοιχείων στις λίστες, έτσι ώστε να μην υπάρχει συσσώρευση στοιχείων σε μια λίστα, τότε ο χρόνος αναζήτησης μπορεί να καλυτερεύσει σημαντικά.

Έστω ότι εισάγονται στον πίνακα n στοιχεία. Ονομάζουμε **παράγοντα φόρτου** (*load factor*) την τιμή $\alpha = \frac{n}{m}$, που εκφράζει το αναμενόμενο (μέσο) μήκος λίστας αν υποθέσουμε ότι τα στοιχεία κατανέμονται ομοιόμορφα στις θέσεις του πίνακα. Τη στιγμή που εισάγεται το i-οστό στοιχείο, η λίστα στην οποία θα μπει θα έχει αναμενόμενο μήκος $\frac{i-1}{m} < \alpha$ — αυτό είναι και το πλήθος δοκιμών που θα χρειαστούν τόσο για την εισαγωγή όσο και για την αναζήτηση του στοιχείου αυτού. [2] Συνεπώς το αναμενόμενο πλήθος δοκιμών που θα κάνει η Find φράσσεται από την τιμή $1 + \alpha$ (συνυπολογίζουμε και την τελευταία ανεπιτυχή δοκιμή, στην περίπτωση που το στοιχείο δεν υπάρχει). Με πιο προσεκτική ανάλυση, η αναζήτηση όταν υπάρχει το στοιχείο κοστίζει περίπου (αναμενόμενη τιμή) $\frac{1+\alpha}{2}$. Επομένως ο αναμενόμενος χρόνος για τις λειτουργίες εισαγωγής, αναζήτησης και διαγραφής είναι $O(1 + \alpha)$.

Ανακατακερματισμός (Rehashing). Είναι συνηθισμένο να μην γνωρίζουμε από πριν τον πληθικό αριθμό που μπορεί να έχει το σύνολό μας. Στην περίπτωση αυτή διαλέγουμε μια τιμή m για τον πίνακα *hash* και όταν ο αριθμός των στοιχείων γίνει μεγαλύτερος από m, δημιουργούμε ένα καινούργιο πίνακα-στήλη μεγέθους $2m$ και με rehashing (δηλαδή ορίζοντας μια νέα συνάρτηση με πεδίο τιμών αυτή τη φορά το $[2m]$), βάζουμε τα στοιχεία στον καινούργιο πίνακα, καταστρέφοντας τον παλιό. Όταν τα στοιχεία γίνουν περισσότερα από $2m$, δημιουργούμε ένα άλλο πίνακα μεγέθους $4m$ κ.ο.κ. Είναι σαφές ότι κάθε φορά η κατάλληλη επιλογή της συνάρτησης κατακερματισμού παίζει σπουδαίο ρόλο προκειμένου να διατηρήσουμε τους

[2] Η εισαγωγή μπορεί να γίνει και πιο γρήγορα, σε 1 βήμα, αν βάζουμε κάθε νέο στοιχείο στην αρχή της λίστας. Αυτό φυσικά αυξάνει το κόστος αναζήτησης των στοιχείων που είναι ήδη στη λίστα και έτσι ο μέσος χρόνος αναζήτησης παραμένει ίδιος.

χρόνους προσπέλασης μικρούς.

Παράδειγμα 1.4. Έστω ότι το σύνολό μας αποτελείται από ακεραίους που μπορούν να πάρουν τιμές στο διάστημα $[0, r], r > n$. Τότε αν χρησιμοποιήσουμε τη συνάρτηση $h(a) = a \bmod m$, όπου m το μέγεθος του τρέχοντα πίνακα-στήλη έχουμε τα παρακάτω: Έστω ότι εισάγουμε τους αριθμούς $1, 5, 8, 3, 9, 6$. Αρχικά επιλέγουμε $m = 2$ και έχουμε :

$$\begin{array}{ll} 0: & \\ 1: & 1, 5 \end{array}$$

Σε αυτό το σημείο επιλέγουμε $m = 4$:

$$\begin{array}{ll} 0: & 8 \\ 1: & 1, 5 \end{array} \qquad \begin{array}{ll} 2: & \\ 3: & 3 \end{array}$$

Τέλος με $m = 8$ προκύπτει το παρακάτω:

$$\begin{array}{ll} 0: & 8 \\ 1: & 1, 9 \\ 2: & \\ 3: & 3 \end{array} \qquad \begin{array}{ll} 4: & \\ 5: & 5 \\ 6: & 6 \\ 7: & \end{array}$$

Ανοιχτή διευθυνσιοδότηση (open addressing)

Η μέθοδος αυτή χρησιμοποιείται όταν το πλήθος των εισαγωγών n είναι μικρότερο ή ίσο των διαθέσιμων θέσεων m. Τότε μπορεί να αποφευχθεί η χρήση λίστας με τον εξής τρόπο:

Φυλάσσεται το πολύ ένα στοιχείο σε κάθε θέση του πίνακα. Αν η θέση $A[h(a)]$ όπου πρέπει να αποθηκευθεί το στοιχείο a είναι κατειλημμένη, τότε διερευνάται αν η επόμενη θέση είναι ελεύθερη, κ.ο.κ. μέχρις ότου βρεθεί ελεύθερη θέση. Για την αναζήτηση (Find) και τη διαγραφή, ελέγχεται πρώτα η θέση $A[h(a)]$ και αν είναι κατειλημμένη από στοιχείο διαφορετικό του a τότε διερευνάται η επόμενη θέση, κ.ο.κ. μέχρις ότου είτε βρεθεί το στοιχείο είτε βρεθεί κενή θέση χωρίς να έχει βρεθεί το στοιχείο, το οποίο σημαίνει ότι το στοιχείο δεν υπάρχει στο λεξικό (αλλιώς θα έπρεπε να είχε μπει στην κενή θέση).

Η παραπάνω μέθοδος λέγεται **γραμμική διερεύνηση** (*linear probing*) αλλά δεν λειτουργεί καλά όταν n είναι συγκρίσιμο με το m λόγω του ότι δημιουργεί μεγάλες συστοιχίες συνεχόμενων θέσεων οι οποίες αυξάνουν την πιθανότητα κατάληψης της επόμενης ελεύθερης θέσης κ.λπ. Η μέθοδος περιγράφεται τυπικά μέσω μιας *συνάρτησης διερεύνησης θέσης* με ορίσματα το κλειδί a και τον αύξοντα αριθμό βήματος $i = 0, 1, \ldots$, που καθορίζει σε κάθε βήμα i ποια θέση του πίνακα θα διερευνηθεί:

$$pos_l(a, i) = h(a) + i \bmod m, \quad i = 0, 1, \ldots$$

Μια καλύτερη μέθοδος είναι η **τετραγωνική διερεύνηση** (*quadratic probing*) όπου αν η θέση $A[h(a)]$ είναι κατειλημμένη, διερευνάται η εξής σειρά θέσεων:

$$pos_q(a,i) = h(a) + c \cdot i + c' \cdot i^2 \bmod m, \quad i = 0, 1, \ldots$$

όπου i είναι ο αύξων αριθμός θέσης που διερευνάται, και c, c' σταθερές. Και αυτή η μέθοδος όμως μπορεί να δημιουργήσει συστοιχίες, τις λεγόμενες *δευτερεύουσες συστοιχίες*.

Μια ακόμη καλύτερη μέθοδος, που πρακτικά παράγει μια αρκετά τυχαία σειρά θέσεων, καθορίζει την απόσταση διερεύνησης σαν συνάρτηση του κλειδιού a, αποφεύγοντας τη δημιουργία συστοιχιών. Η μέθοδος λέγεται **διπλός κατακερματισμός** (*double hashing*) και χρησιμοποιεί δύο διαφορετικές συναρτήσεις κατακερματισμού για τον καθορισμό της θέσης διερεύνησης:

$$pos_{dh}(a,i) = h_1(a) + i \cdot h_2(a) \bmod m, \quad i = 1, 2, \ldots$$

Ενώ η γραμμική και η τετραγωνική διερεύνηση δημιουργούν μόνο m διαφορετικές δυνατές ακολουθίες θέσεων διερεύνησης (η αρχική θέση καθορίζει και όλες τις υπόλοιπες), ο διπλός κατακερματισμός, με λίγη προσοχή στην επιλογή των δύο συναρτήσεων, δημιουργεί περίπου m^2 διαφορετικές ακολουθίες διερεύνησης, γεγονός που τον κάνει να συμπεριφέρεται πρακτικά πολύ παρόμοια με τον ομοιόμορφο κατακερματισμό (μολονότι σε αυτόν θα είχαμε επιλογή κάθε θέσης ομοιόμορφα, επομένως θα υπήρχαν $m!$ ακολουθίες διερεύνησης).

Αποδεικνύεται ότι αν υποθέσουμε ομοιόμορφο κατακερματισμό (κάτι που προσεγγίζεται όπως είπαμε από τον διπλό κατακερματισμό), αν $\alpha = n/m < 1$ είναι ο παράγοντας φόρτου, το κόστος εισαγωγής και αναζήτησης (επιτυχούς ή μη) φράσσονται από την τιμή $\frac{1}{1-\alpha}$: η πιθανότητα εύρεσης κενής θέσης κατά την εισαγωγή ή την αναζήτηση είναι $1 - \alpha$ (θεωρώντας ότι η συνάρτηση θέσης επιλέγει θέσεις σχεδόν τυχαία και ανεξάρτητα από τις προηγούμενες δοκιμές) και επομένως ο αναμενόμενος αριθμός δοκιμών είναι $1/(1-\alpha)$ (αποδεικνύεται με χρήση βασικής θεωρίας πιθανοτήτων). Μια πιο προσεκτική ανάλυση δείχνει ότι ο χρόνος *επιτυχούς αναζήτησης* είναι πολύ καλύτερος: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$. [3]

Κατακερματισμός κούκου. Τέλος, μια αρκετά διαφορετική μέθοδος που λέγεται *κατακερματισμός κούκου* (*cuckoo hashing*) και προτάθηκε πριν 20 έτη περίπου από τους Pagh και Rodler [4] έχει αποδειχθεί ιδιαίτερα αποτελεσματική καθώς εξασφαλίζει *σταθερό χρόνο αναζήτησης* (2 δοκιμές μόνο!). Και σε αυτή τη μέθοδο χρησιμοποιούνται δύο διαφορετικές συναρτήσεις κατακερματισμού h_1, h_2 αλλά με τελείως διαφορετικό τρόπο απ'ότι στον διπλό κατακερματισμό. Συγκεκριμένα, δημιουργούνται δύο πίνακες m θέσεων. Κάθε στοιχείο μπαίνει στην θέση $h_1(a)$

[3] Η ανάλυση έχει κάποιο βαθμό δυσκολίας και αφήνεται ως άσκηση. Ξεκινήστε αποδεικνύοντας ότι κατά την εισαγωγή του i-οστού στοιχείου ο αναμενόμενος αριθμός δοκιμών μέχρι να βρεθεί κενή θέση είναι $m/(m-i)$.

του πρώτου πίνακα και αν αυτή είναι κατειλημμένη “διώχνει” το στοιχείο a' που βρίσκεται εκεί[4], το οποίο μπαίνει στη θέση $h_2(a')$ του δεύτερου πίνακα, όπου αν υπάρχει στοιχείο a'' “διώχνεται” και επαναλαμβάνεται η παραπάνω διαδικασία για το a'', κ.ο.κ. μέχρις ότου κάποιο στοιχείο να μπει σε ελεύθερη θέση ή να δημιουργηθεί κύκλος. Στην τελευταία περίπτωση πρέπει να γίνει rehashing, επιλέγοντας νέες συναρτήσεις κατακερματισμού. Στην μέθοδο αυτή η εισαγωγή μπορεί να κοστίζει χρονικά λίγο περισσότερο, όμως η αναζήτηση (και η διαγραφή επομένως) γίνεται σε 2 το πολύ δοκιμές: το στοιχείο a, αν βρίσκεται στην δομή θα βρίσκεται είτε στη θέση $h_1(a)$ του πρώτου πίνακα είτε στη θέση $h_2(a)$ του δεύτερου.

Σημείωση: σε όλες τις μεθόδους ανοιχτής διευθυνσιοδότησης, αν ο πίνακας γεμίσει πρέπει να αυξηθεί το μέγεθος (π.χ. να διπλασιαστεί το m) και να γίνει ανακατακερματισμός (rehashing).

[4] Ο κούκος βγάζει τα αυγά άλλων πουλιών από τη φωλιά τους για να βάλει τα δικά του, από όπου και το όνομα της μεθόδου.

Βιβλιογραφία

[1] Knuth, Donald E.: The Art of Computer Programming: Volume 3: Sorting and Searching (2nd edition), Addison-Wesley, 1998.

[2] Thomas Cormen, Charles Leiserson, Ronald Rivest and Cliff Stein, “Introduction to Algorithms”, 3rd edition, MIT Press, 2009.

[3] C.L. Liu. Στοιχεία Διακριτών Μαθηματικών (απόδοση στα Ελληνικά: Κ. Μπους και Δ. Γραμμένος). Πανεπιστημιακές Εκδόσεις Κρήτης, 2003.

[4] Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms 51, pp. 122-–144 (2004).

[5] Larry Carter and Mark N. Wegman: Universal Classes of Hash Functions, J. Comput. Syst. Sci. 18(2), pp. 143–154, (1979), https://doi.org/10.1016/0022-0000(79)90044-8.