Κατακερματισμός (Hashing)

Ο κατακερματισμός είναι μια τεχνική οργάνωσης ενός αρχείου. Είναι αρκετά δημοφιλής μέθοδος για την οργάνωση αρχείων Βάσεων Δεδομένων, καθώς βοηθάει σημαντικά στην γρήγορη αναζήτηση συγκεκριμένων τιμών ενός κλειδιού. Η βασική ιδέα είναι να καθορίζεται μια συνάρτηση κατακερματισμού (hash function) h(x), η οποία όταν εφαρμοσθεί στο πεδίο κατακερματισμού (hashing field) δίνει τον αριθμό σελίδας στον οποίο έχει αποθηκευθεί ή πρέπει να αποθηκευθεί η εγγραφή.

Παράδειγμα: Έστω h(x) = x mod 10, δηλαδή η τιμή της είναι το υπόλοιπο της διαίρεσης της τιμής του κλειδιού με το 10. Η τιμή 32 δίνει υπόλοιπο 2, άρα η εγγραφή που έχει στο πεδίο hashing την τιμή 32 θα αποθηκευθεί στη σελίδα 2. Προφανώς το 2 που χρησιμοποιούμε εδώ δεν είναι η πραγματική διεύθυνση κάποιας σελίδας στο δίσκο, αλλά κάποια αντιστοίχιση της πραγματικής διεύθυνσης (πχ. 12aaf32acf σε δεκαεξαδικό σύστημα) με μια αναγνωρίσιμη τιμή. Αντιστρόφως, αν ψάχνουμε την εγγραφή που έχει τιμή 45 στο πεδίο hashing, εφαρμόζουμε τη συνάρτηση h(x) και το αποτέλεσμα είναι 45 mod 10 = 5, συνεπώς γνωρίζουμε ότι αν υπάρχει η τιμή 45, αυτή θα βρίσκεται μόνο στη σελίδα με αριθμό 5. Έτσι, χρειάζεται πάντα αναζήτηση μόνο σε μία σελίδα, δηλαδή **απαιτείται μόνο μία προσπέλαση στο δίσκο και μόνο μία μεταφορά των δεδομένων της σελίδας στη μνήμη**.

Στατικό hashing

Η πιο απλή μορφή hashing είναι αυτή που αναφέρθηκε παραπάνω. Υιοθετείται μια συγκεκριμένη συνάρτηση και χρησιμοποιείται για όσο τα δεδομένα αποθηκεύονται με αυτόν τον τρόπο. Προφανώς είναι πιθανό να αποφασίσουμε κάποια στιγμή ότι θέλουμε να την αντικαταστήσουμε με μια διαφορετική συνάρτηση. Αυτό δεν αλλάζει τις ιδιότητες και τα χαρακτηριστικά της συγκεκριμένης μεθόδου.

Η πιο «φυσιολογική» συνάρτηση που μπορεί να χρησιμοποιηθεί είναι η h(x) = x mod N, όπου N ακέραιος. Αυτό συμβαίνει γιατί η συγκεκριμένη συνάρτηση δίνει ακριβώς N σελίδες, από 0 έως N-1, ενώ αν δεν υπάρχει κάποια «ιδιαίτερη» προϋπόθεση οι σελίδες θα δέχονται περίπου ίδιο όγκο εγγραφών. Πχ. για τους Αριθμούς Μητρώου των φοιτητών, περίπου το 1/10 θα καταλήγει σε κάθε αριθμό από 0 έως 9, συνεπώς περίπου 1/10 του συνολικού κάθε φορά πλήθους εγγραφών θα βρίσκεται σε κάθε μία από τις σελίδες. Το ίδιο ισχύει για τους αριθμούς κυκλοφορίας των αυτοκινήτων και άλλες παρόμοιες εφαρμογές. Ωστόσο, μπορεί να υπάρχουν και άλλες συναρτήσεις hashing, με διαφορετική συμπεριφορά, πάντως οι γενικές αρχές δεν αλλάζουν και σε εκείνη την περίπτωση.

Ας δούμε ένα παράδειγμα μιας τέτοιας εναλλακτικής συνάρτησης, για να αποκτήσουμε μια αίσθηση των διαφορετικών επιλογών που έχουμε:

h(x) = άθροισμα τελευταίου και προτελευταίου ψηφίου της τιμής του πεδίου

Μια τέτοια συνάρτηση έχει προφανώς 19 τιμές (0 … 18), καθώς και τα δύο ψηφία μπορεί να είναι 0 (ελάχιστη τιμή του αθροίσματος 0), και τα δύο ψηφία μπορεί να είναι 9 (μέγιστη τιμή του αθροίσματος 18), ή τα δύο ψηφία να έχουν ενδιάμεσες τιμές (τιμές του αθροίσματος 1 έως και 17). Εδώ οι δύο ακραίες τιμές θα συμβούν λιγότερες φορές, η ιδέα πάντως παραμένει ίδια.

Παράδειγμα: έχουμε τις εγγραφές ενός πίνακα με τις παρακάτω τιμές του πεδίου hashing:

5, 8, 10, 12, 24, 32, 37, 39, 41, 46, 51, 55, 60, 63, 66, 70

Η συνάρτηση hashing που θα χρησιμοποιήσουμε είναι h(x) = x mod 8. Υποθέτουμε ότι η σελίδα του δίσκου μας γεμίζει με 3 εγγραφές. Προφανώς αυτό είναι αδύνατο σε πραγματικές συνθήκες, καθώς καμία σελίδα δεν είναι τόσο μικρή, ούτε 3 εγγραφές μπορούν να γεμίσουν μια φυσιολογική σελίδα των 1024 bytes. Εδώ προσπαθούμε να εξοικειωθούμε με την ιδέα της κατανομής των εγγραφών με τη χρήση του στατικού hashing.

Οι σελίδες θα είναι από 0 έως 7, γιατί αυτά είναι τα πιθανά υπόλοιπα της διαίρεσης με το 8. Προφανώς

h(5) = 5 (γιατί 5 = 0*8 +5)

h(8) = 0 (γιατί 8 = 1*8 +0)

h(10) = 2 (γιατί 10 = 1*8 +2)

h(12) = 4

κ.ο.κ.

Οι σελίδες λοιπόν παίρνουν την παρακάτω μορφή:

0	8 24 32

1	41

2	10 66

3	51

4	12 60

5	5 13 37

6	46 70

7	55 63

Παρατηρούμε ότι, αν οι τιμές είναι τυχαίες και χωρίς κάποιο συγκεκριμένο κανόνα, οι εγγραφές περίπου ισοκατανέμονται στις σελίδες (κάτι τέτοιο θα γινόταν ακόμα πιο εμφανές αν είχαμε για παράδειγμα 1000 τιμές). Συνεπώς, η χρήση του hashing εξασφαλίζει σχεδόν αυτόματα μια ισορροπημένη αξιοποίηση των διαφορετικών τμημάτων του δίσκου, κάτι που αποτελεί ένα σημαντικό πλεονέκτημα για οποιοδήποτε υπολογιστικό σύστημα.

Πηγαίνοντας αντίστροφα, ας δούμε πώς βρίσκουμε μια τιμή που αναζητούμε ανάμεσα στις αποθηκευμένες. Ας υποθέσουμε λοιπόν ότι αναζητούμε την τιμή 53 του πεδίου hashing. Προφανώς γνωρίζουμε ότι χρησιμοποιείται η συνάρτηση hashing h(x) =x mod 8. Υπολογίζουμε h(53) = 5, γιατί 53 = 6*8 + 5. Μεταφέρουμε στη μνήμη τη σελίδα με διεύθυνση 5 και ψάχνουμε σε αυτήν. Είναι η μόνη σελίδα στην οποία μπορεί να βρίσκεται η συγκεκριμένη τιμή, **αν θεωρήσουμε ότι καμία σελίδα δεν έχει χρειαστεί να «βολέψει» εγγραφές περισσότερες από το όριο της** (δηλαδή περισσότερες από 3 εγγραφές). Κάνουμε την αναζήτηση για την τιμή 53 και διαπιστώνουμε ότι δεν υπάρχει. Η αναζήτηση μας τελειώνει σε αυτό το σημείο. Αντίστοιχα, αν αναζητούμε την τιμή 66, κάνουμε τον ίδιο υπολογισμό: Υπολογίζουμε h(66) = 2, γιατί 66 = 8*8 + 2. Μεταφέρουμε στη μνήμη τη σελίδα

με διεύθυνση 2 και ψάχνουμε σε αυτήν. Κάνουμε την αναζήτηση για την τιμή 66 και διαπιστώνουμε ότι υπάρχει και ακολούθως ξέρουμε ότι θα χρησιμοποιήσουμε αυτή την εγγραφή για να κάνουμε ό,τι προκάλεσε την αναζήτηση.

Εκ πρώτης όψεως λοιπόν, φαίνεται ότι βρήκαμε μια πολύ καλή λύση για την αποθήκευση του αρχείου μας. Δυστυχώς, όπως σε όλες τις λύσεις, υπάρχουν και προβλήματα. Το πρώτο που διαπιστώνουμε είναι ότι αυτό το σύστημα μπορεί να χειριστεί το πολύ m * N εγγραφές, όπου m οι εγγραφές που χωρούν σε μια σελίδα και N οι σελίδες που προκύπτουν από τη συνάρτηση που χρησιμοποιείται. Στο παράδειγμα παραπάνω, το πλήθος αυτό είναι 3 * 8 = 24. Μετά τις 24 εγγραφές, είναι βέβαιο ότι θα χρειαστεί να βρούμε μια λύση για τις επιπλέον εγγραφές. Αυτό μπορεί να συμβεί και νωρίτερα, αν οι εγγραφές σε μια συγκεκριμένη σελίδα γίνουν περισσότερες από τη χωρητικότητα της, έστω κι αν άλλες σελίδες έχουν χώρο. Πχ. στο παράδειγμα, αν προκύψει μια εγγραφή με τιμή 40 στο πεδίο hashing, θα χρειαστεί να βρούμε πώς θα την αποθηκεύσουμε, καθώς η σελίδα 0 στην οποία κανονικά πρέπει να αποθηκευθεί είναι γεμάτη. Αυτό το πρόβλημα δεν μπορεί να μας το λύσει ο περίσσιος χώρος που έχει αυτή τη στιγμή η σελίδα 1.

Μια λύση που μπορούμε να χρησιμοποιήσουμε είναι να επεκτείνουμε την κάθε σελίδα από τις αρχικές με μια διεύθυνση άλλης σελίδας, στην οποία αποθηκεύουμε τις εγγραφές που πλεονάζουν από την αρχική σελίδα. Ας θυμίσουμε ότι κάθε σελίδα του δίσκου μας κρατάει ένα μικρό τμήμα για τις ανάγκες του συστήματος, γράφοντας εκεί διάφορες πληροφορίες διαχείρισης. Στην περίπτωση μας, θα καταγραφεί εκεί σε μια συγκεκριμένη θέση η διεύθυνση της σελίδας στην οποία «συνεχίζεται» η συγκεκριμένη σελίδα. Αυτή θα είναι συνήθως η πρώτη διαθέσιμη σελίδα στο σύστημα, ωστόσο εμείς θα θεωρούμε πάντα για λόγους απλότητας ότι δεν έχουμε «κατειλημμένες» σελίδες μετά τις σελίδες που χρησιμοποιούμε για την αρχική αποθήκευση (σελίδες 0 έως 7 στην προκειμένη περίπτωση), συνεπώς θα χρησιμοποιήσουμε διαδοχικά τις σελίδες8, 9 , 10 κοκ.

Υποθέτουμε λοιπόν ότι μετά την αρχική αποθήκευση έρχεται μια νέα εγγραφή με τιμή του πεδίου hashing ίση με 40. Επειδή h(40) = 0, η εγγραφή πρέπει να αποθηκευθεί στη σελίδα 0, η οποία είναι γεμάτη. Θα πάμε στην επόμενη διαθέσιμη σελίδα, η οποία είναι η 8. Θα χρειαστεί να καταγράψουμε στη σελίδα 0 ότι η σελίδα στην οποία έχει επεκταθεί η σελίδα 0 είναι η 8:

Με την τροποποίηση που κάναμε στο αρχικό σχήμα αποθήκευσης, λύσαμε κατ' αρχήν το πρόβλημα της υπερχείλισης σελίδων. Απλώς επεκτείνουμε τη σελίδα, καταγράφοντας ποια είναι η σελίδα στην οποία συνεχίζεται η αποθήκευση. Άρα η αναζήτηση μας τώρα θα κάνει την αρχική αναζήτηση στη σελίδα που παραπέμπει η τιμή που υπολογίζει η συνάρτηση hashing και αν δεν την βρίσκει θα εξετάζει αν στη σελίδα αυτή έχει καταγραφεί μια πληροφορία σχετικά με την «επόμενη» σελίδα. Στην περίπτωση μας θα βρει τον αριθμό 8 και θα επιστρέψει στο δίσκο για να πάρει τη σελίδα 8. Αν δεν καταφέρει να βρει την τιμή ούτε σε αυτή τη σελίδα, θα εξετάσει αν υπάρχει καταγραφή επόμενης σελίδας. Στη συγκεκριμένη περίπτωση, θα βρει το σημείο εκείνο κενό (δηλαδή θα βρει μια τιμή null) και θα τερματίσει την αναζήτηση εκεί.

Αυτή η διαδικασία δημιουργεί βεβαίως άλλα προβλήματα. Το προφανές είναι ότι, αν δεν υπάρχει κάποιος τρόπος να μείνουν οι τιμές ταξινομημένες, είναι πιθανόν να χρειαστεί να φέρουμε ολόκληρη την αλυσίδα των σελίδων μέχρι να βρούμε ή να μην βρούμε την τιμή που ψάχνουμε. Έτσι, είναι δύσκολο να υπολογίσουμε και τον χρόνο που θα μας χρειάζεται για μια μέση αναζήτηση.

Επεκτατός κατακερματισμός (extendible hashing)

Μέρος των προβλημάτων που προκύπτουν από την παραπάνω εφαρμογή του hashing οφείλεται στη **στατικότητα** που έχει η μέθοδος. Ορίζονται αρχικά οι σελίδες που διατίθενται από το σύστημα και ακολούθως πρέπει να βρεθούν λύσεις για την υπερχείλιση, οι οποίες όμως δεν μπορούν να κάνουν κάτι σχετικά με τις αρχικές σελίδες, παρά μόνο να τις επεκτείνουν. Θα ήταν πιο βολικό ένα σχήμα όπου θα μπορούσαμε να δίνουμε επί μέρους λύσεις επέκτασης μόνο όπου υπάρχει ανάγκη. Αυτό κάνει ο **επεκτατός κατακερματισμός** (που σημαίνει κατά λέξη «κατακερματισμός που μπορεί να επεκτείνεται»).

Στην περίπτωση αυτή, διατηρούμε μια διαφορετική διευθυνσιοδότηση από αυτήν που είδαμε στην αρχή. Η βασική μας αρχή είναι ότι **ξεκινάμε με όσες σελίδες χρειαζόμαστε και όχι με όσες προκύπτουν από τις τιμές του πεδίου hashing**. Όταν χρειαζόμαστε περισσότερο χώρο σε κάποια σελίδα, **διασπούμε αυτή τη σελίδα σε δύο νέες, διατηρώντας όμως ανέγγιχτες τις υπόλοιπες**.

Παράδειγμα: έχουμε τις εγγραφές ενός πίνακα με τις παρακάτω τιμές του πεδίου hashing:

5, 8, 10, 12, 24, 31

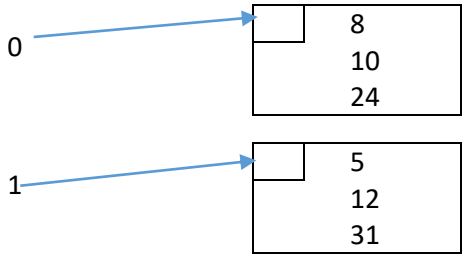
Ισχύουν τα ίδια αντικειμενικά δεδομένα με το πρώτο μας παράδειγμα, δηλαδή h(x) = x mod 8 και σε κάθε σελίδα του δίσκου χωρούν 3 εγγραφές. Στην περίπτωση μας, καλυπτόμαστε αρχικά με μόλις 2 σελίδες. Ουσιαστικά πρέπει να αποφασίσουμε σε ποια από τις 2 αυτές σελίδες θα ενσωματωθεί η κάθε μία από τις σελίδες 0, 1, 2, …,7. Θα μπορούσαμε να πούμε ότι στην «νέα» σελίδα 0 θα πάνε οι ζυγές σελίδες (0, 2, 4, 6) και στην «νέα» σελίδα 1 θα πάνε οι μονές σελίδες (1, 3, 5, 7). Αυτός είναι βέβαια ένας πρακτικός τρόπος, ο οποίος δεν μπορεί να εφαρμοστεί εύκολα από το λογισμικό του συστήματος μας. Ας δούμε μια λύση που ταιριάζει πολύ καλύτερα με τη δυαδική φύση των υπολογισμών ενός οποιουδήποτε συστήματος:

Αν χρειαζόμαστε δύο σελίδες, μπορούμε να δώσουμε τις διευθύνσεις 0 και 1. Αν μετατρέψουμε τις αρχικές διευθύνσεις στα δυαδικά τους ισοδύναμα, μπορούμε να πούμε ότι στη σελίδα 0 θα πάνε όλες οι σελίδες που η διεύθυνση τους έχει πρώτο δυαδικό ψηφίο το 0 και στη σελίδα 1 θα πάνε όλες οι σελίδες που η διεύθυνση τους έχει πρώτο δυαδικό ψηφίο το 1. Ας δούμε τι προκύπτει, υπολογίζοντας τα δυαδικά ισοδύναμα των σελίδων 0 έως 7:

Αρχική διεύθυνση	Δυαδικό ισοδύναμο
0	**0**00
1	**0**01
2	**0**10
3	**0**11
4	**1**00
5	**1**01
6	**1**10
7	**1**11

Αν μας αρκούν 2 σελίδες, αποφασίζουμε ότι όποια διεύθυνση έχει πρώτο δυαδικό ψηφίο το 0 θα πάει στη σελίδα 0 και όποια διεύθυνση έχει πρώτο δυαδικό ψηφίο το 1 θα πάει στη σελίδα 1. Πρακτικά, όποια εγγραφή βγάζει τιμή hashing ίση με 0, 1, 2, ή 3 κατανέμεται στη σελίδα 0 και όποια εγγραφή βγάζει τιμή hashing ίση με 4, 5, 6, ή 7 κατανέμεται στη σελίδα 1.

Τιμή πεδίου	**Τιμή hashing**	**Τιμή δυαδικού ισοδυνάμου**
5	5	101
8	0	000
10	2	010
12	4	100
24	0	000
31	7	111

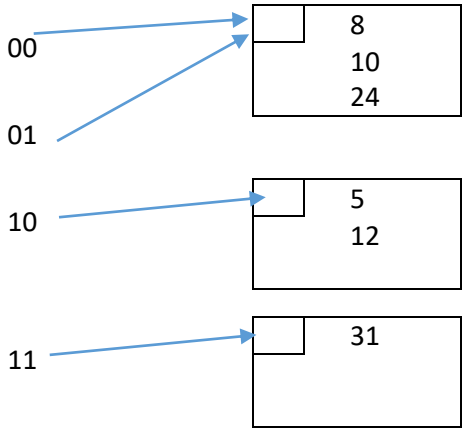


Ας πούμε ότι η επόμενη εγγραφή που προκύπτει είναι η 30. Υπολογίζουμε h(30) = 6, άρα αυτή πρέπει να κατανεμηθεί στη σελίδα που δείχνει η τιμή 1. Θα χρειαστεί όμως να επεκτείνουμε τη σελίδα αυτή, γιατί είναι ήδη γεμάτη. Άρα δεν καλυπτόμαστε πλέον από το ένα ψηφίο που χρησιμοποιούμε. Η λύση είναι να χρησιμοποιήσουμε και το δεύτερο ψηφίο, με αποτέλεσμα να προκύψουν 4 διαφορετικές τιμές. Αναπαράγουμε εδώ τον πίνακα με τα δυαδικά ισοδύναμα, τονίζοντας τώρα τα δύο πρώτα ψηφία:

Αρχική διεύθυνση	Δυαδικό ισοδύναμο
0	**00**0
1	**00**1
2	**01**0
3	**01**1
4	**10**0
5	**10**1
6	**11**0
7	**11**1

Τώρα λοιπόν θα χρησιμοποιούμε 4 δείκτες, **χωρίς όμως να είναι απαραίτητη η χρήση 4 σελίδων**. Θα χρειαστεί όπως είπαμε να διασπάσουμε τη σελίδα 1 στις σελίδες 10 και 11 και να κατανείμουμε και πάλι τις εγγραφές, αλλά η 0 δεν χρειάζεται διάσπαση. Συνεπώς, οι νέοι δείκτες 00 και 01 θα δείχνουν και οι δύο στην ίδια σελίδα.

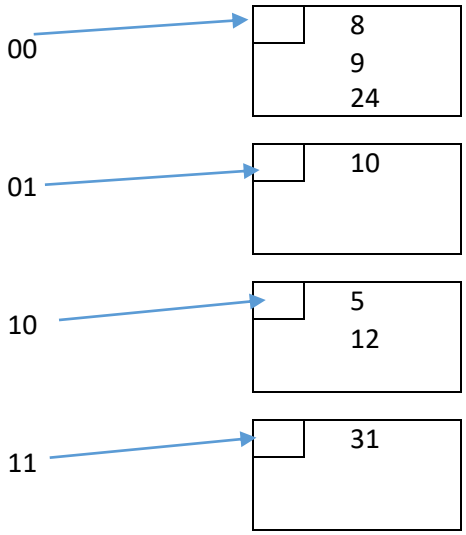
Πώς θα κατανεμηθούν οι τιμές των εγγραφών της σελίδας 1 στις δύο νέες σελίδες 10 και 11; Θα χρειαστεί να δούμε τι συμβαίνει με τις τιμές hashing. Από τον παραπάνω πίνακα βλέπουμε ότι οι τιμές 4 και 5 αντιστοιχούν στη σελίδα 10 και οι τιμές 6 και 7 στη σελίδα 11:



Τι συμβαίνει αν χρειαστεί να αναζητήσουμε μια εγγραφή; Ας πούμε ότι αναζητούμε την τιμή 16. Υπολογίζουμε h(16) = 16 mod 8 = 0. Το δυαδικό ισοδύναμο είναι 000, άρα παίρνοντας τα δυο πρώτα ψηφία βλέπουμε ότι πρέπει να πάρουμε τη σελίδα στην οποία δείχνει η τιμή 00. **Δεν μας ενδιαφέρει ότι στην ίδια σελίδα δείχνει και ένας άλλος δείκτης**.

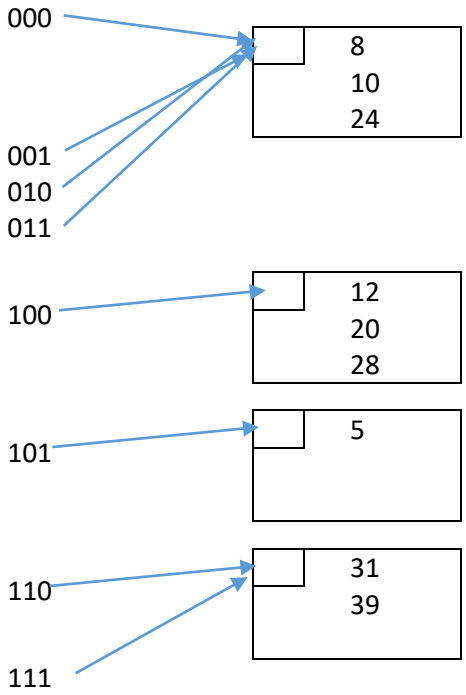
Το ίδιο συμβαίνει και στην άλλη περίπτωση. Ας πούμε ότι αναζητούμε την τιμή 23. Υπολογίζουμε h(23) = 23 mod 8 = 7. Το δυαδικό ισοδύναμο είναι 111, άρα παίρνοντας τα δυο πρώτα ψηφία βλέπουμε ότι πρέπει να πάρουμε τη σελίδα στην οποία δείχνει η τιμή 11. **Δεν υπάρχει κάποια διαφορά από την προηγούμενη περίπτωση**.

Αν προκύψει ανάγκη να διασπαστεί η σελίδα στην οποία δείχνουν τα 00 και 01, δεν χρειάζεται να κάνουμε κάτι στους δείκτες. Απλώς οι τιμές hashing 0 και 1 θα πάνε στη σελίδα που δείχνει το 00 και οι τιμές hashing 2 και 3 θα πάνε στη σελίδα που δείχνει το 01. Πχ. αν προκύψει η τιμή 9, η νέα μορφή των σελίδων θα είναι:



Ας επιστρέψουμε στην προηγούμενη κατάσταση, όπου οι δείκτες 00 και 01 δείχνουν στην ίδια σελίδα (δηλαδή πριν την εισαγωγή του 9). Τι συμβαίνει αν χρειαστούμε νέα διάσπαση σε κάποια από τις σελίδες 10 ή 11; Ας υποθέσουμε ότι έρχονται οι τιμές 20, 28 και 39. Οι 20 και 28 πρέπει να μπουν στη σελίδα που δείχνει το 10 και η 39 στην 11. Η 10 πρέπει να διασπασθεί και πάλι, άρα χρειάζεται και νέα επέκταση των ψηφίων από 2 σε 3. Η 10 θα

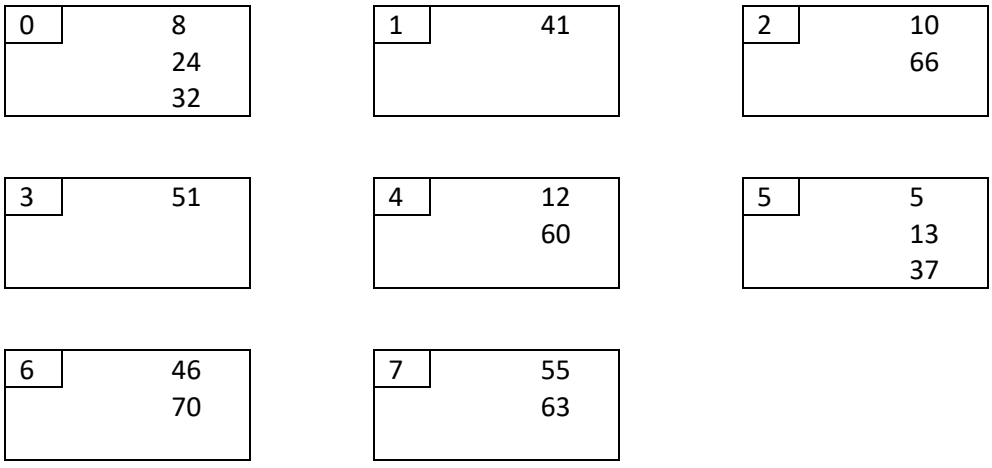
διασπασθεί στην 100 και στην 101. Και οι υπόλοιπες διευθύνσεις θα γίνουν τριψήφιες, **αλλά αυτό δεν θα επηρεάσει τις σελίδες**.



Είναι φανερό ότι αυτή η μέθοδος μας δίνει τη δυνατότητα να επεκτείνουμε τις σελίδες μόνο για τις περιπτώσεις που αυτό είναι απαραίτητο. Για τις υπόλοιπες σελίδες, μια τέτοια επέκταση απλώς δημιουργεί επιπλέον γραμμές στον πίνακα της διευθυνσιοδότησης, με τους νέους δείκτες να δείχνουν όπου έδειχνε και ο παλιός.

Γραμμικός κατακερματισμός (Linear hashing)

Μια άλλη ιδέα για δυναμική επέκταση του hashing είναι να επεκτείνουμε τις σελίδες δυναμικά, χωρίς να χρειάζεται κάποιος κατάλογος, με τιμές και διευθύνσεις που δείχνουν στις σελίδες του δίσκου. Ας επιστρέψουμε στο αρχικό παράδειγμα:

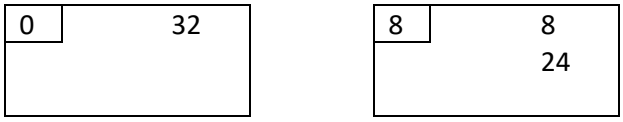


Ας υποθέσουμε ότι η επόμενη τιμή του πεδίου hashing που χρειάζεται να αποθηκευθεί είναι η 45. Υπολογίζουμε h(45) = x mod 8 = 5. Βλέπουμε ωστόσο ότι η σελίδα 5 είναι γεμάτη.

Χρειάζεται λοιπόν να δημιουργήσουμε μια σύνδεση με άλλη σελίδα, η οποία θα αποτελεί την επέκταση της 5 και θα είναι μια υπερχείλιση.

Εφ' όσον χρειάστηκε η επέκταση μιας σελίδας, συμπεραίνουμε ότι σύντομα θα προκύπτει ανάγκη επέκτασης και άλλων σελίδων. Δημιουργούμε λοιπόν χώρο ως εξής:

Διασπούμε την πρώτη σελίδα (δηλαδή την 0) σε δύο, **όχι όμως με την ίδια συνάρτηση**. Αντίθετα, **για αυτές μόνο τις σελίδες**, χρησιμοποιούμε τη συνάρτηση με διπλάσιο N, δηλαδή την $h_1(x) = x \bmod 16$. Η επόμενη σελίδα είναι η 8. Για την σελίδα που διασπάστηκε, πρέπει να εξετάσουμε αν οι τιμές που βρίσκονταν εκεί πηγαίνουν στη σελίδα 0 ή στη σελίδα 8. Αν το υπόλοιπο της διαίρεσης με το 8 ήταν 0, το υπόλοιπο διαίρεσης με το 16 θα είναι είτε 0 είτε 8. Οι τιμές που έχουμε στη σελίδα 0 λοιπόν κατανέμονται ως εξής:

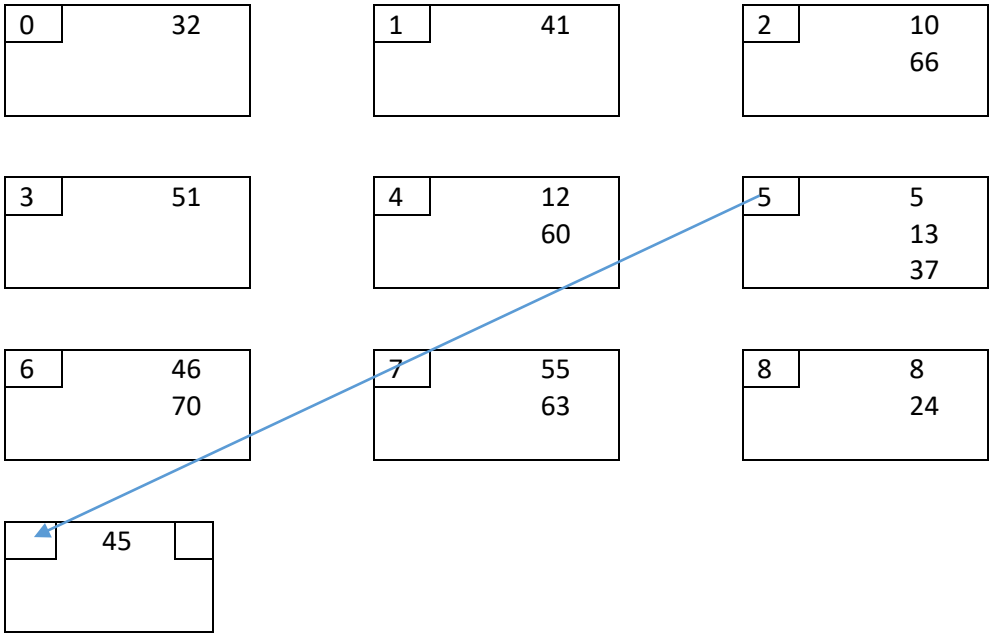


Χρειάζεται και μια ακόμη πρόβλεψη: πρέπει να ξέρουμε σε ποιες σελίδες χρησιμοποιείται η συνάρτηση x mod 8 και σε ποιες η x mod 16. Για αυτό το σκοπό, δημιουργούμε έναν μετρητή n, ο οποίος μετράει τι διασπάσεις. Αυτός ο μετρητής ξεκινάει από το 0 και σε κάθε διάσπαση σελίδας αυξάνεται κατά 1. Κάθε φορά που χρειάζεται να αποθηκεύσουμε μια τιμή, κάνουμε τα ακόλουθα βήματα:

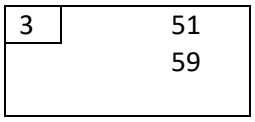
1. Χρησιμοποιούμε την αρχική συνάρτηση x mod 8 και βρίσκουμε τη σελίδα στην οποία πρέπει να αποθηκευθεί η τιμή.
2. Μετά κοιτάμε την τιμή του μετρητή.
   - Αν είναι μικρότερη από την σελίδα που υπολογίσαμε, προχωράμε κανονικά
   - Αν δεν είναι, αυτό σημαίνει ότι η σελίδα την οποία υπολογίσαμε έχει διασπαστεί.

     Στην περίπτωση αυτή, χρησιμοποιούμε την νέα συνάρτηση x mod 16.

Συνεχίζοντας το παράδειγμα μας, μετά την εισαγωγή του 45, και τη διάσπαση της σελίδας 0, η κατάσταση έχει ως εξής:



Ο δείκτης έχει την τιμή 1. Υποθέτουμε ότι έρχεται η τιμή 59. Υπολογίζουμε h(59) = 59 mod 8 =3. Ο δείκτης έχει την τιμή 1, άρα η σελίδα 3 δεν έχει ακόμα διασπαστεί. Η τιμή μπαίνει κανονικά στη σελίδα 3:



Η επόμενη τιμή είναι 40. h(40) =40 mod 8 = 0. Άρα η τιμή πρέπει να πάει στη σελίδα 0. Ο μετρητής είναι 1, άρα η πρώτη σελίδα έχει διασπαστεί και αυτή είναι η 0. Επιστρέφουμε λοιπόν και ξανακάνουμε τον υπολογισμό με τη νέα συνάρτηση h(40) = 40 mod 16 = 8. Άρα η τιμή πρέπει να πάει στη σελίδα 8.

Όταν χρειαστεί να διασπαστεί και η σελίδα 5, η οποία έχει και μια σελίδα υπερχείλισης, η νέα σελίδα θα είναι η 5+8 = 13. Εκεί θα χρειαστεί να δούμε για κάθε τιμή αν η συνάρτηση x mod 16 μας δίνει αποτέλεσμα 5 ή 13. Στο συγκεκριμένο παράδειγμα, θα κάνουμε τον υπολογισμό για τις τιμές 5, 13, 37 και 45. Οι 5 και 37 δίνουν αποτέλεσμα 5 και οι 13 και 45 δίνουν αποτέλεσμα 13. Άρα η νέα μορφή θα είναι (χωρίς υπερχείλιση)

Όταν σιγά-σιγά διασπαστούν όλες οι σελίδες, χρησιμοποιούμε για όλες τη νέα συνάρτηση x mod 16 και μηδενίζουμε τον μετρητή. Λογικά, ο χώρος είναι πλέον αρκετός για να συνεχίσουμε για πολύ περισσότερες τιμές. Αν χρειαστούμε και πάλι επεκτάσεις κάνουμε την ίδια διαδικασία, χρησιμοποιώντας και τη συνάρτηση x mod 32, διπλασιάζοντας ακόμη μια φορά το χώρο.