

Linear Hashing

Τμήμα Πληροφορικής & Τηλ/νών, ΕΚΠΑ

Υλοποίηση Συστημάτων Βάσεων Δεδομένων
<http://www.di.uoa.gr/~k18>



Linear vs other Hashing

- Σε αντίθεση με το στατικό κατακερματισμό, τα buckets αυξάνονται καθώς γίνονται εισαγωγές (και μειώνονται καθώς γίνονται διαγραφές – αλλά δεν ενδιαφέρει στα πλαίσια της άσκησης)
- Σε αντίθεση με το extendible hashing (επεκτατό κατακερματισμό), τα buckets προστίθενται 1 τη φορά

Απαραίτητη Πληροφορία

- Στο Linear Hash, η πληροφορία που ενδιαφέρει είναι:
 - Πλήθος των εγγραφών (#recs)
 - Αριθμός των buckets που έχουμε (#buckets)
 - Το επόμενο block προς διάσπαση (split)
 - Παράγοντας φόρτωσης του αρχείου (load factor)
 - Το επίπεδο συνάρτησης κατακερματισμού h (level)
- Ο παράγοντας φόρτωσης υπολογίζεται ως
$$\#recs / (capacity * \#buckets)$$

όπου capacity είναι το πλήθος των εγγραφών που χωράει ένα bucket

Απαραίτητη Πληροφορία (2)

Προσοχή:

- 1) Στον παράγοντα φόρτωσης λαμβάνουμε υπόψη **όλες** τις εγγραφές
- 2) Στον παράγοντα φόρτωσης **δεν** μετράμε τα block υπερχείλισης
- 3) Για το capacity, θεωρούμε ισοδυναμία μεταξύ block και bucket. Άρα, μας ενδιαφέρει πόσες εγγραφές χωράνε σε ένα block.

Εισαγωγή με Linear Hash: Κεντρική ιδέα

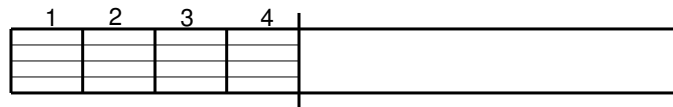
- LH_Insert(Record rec)
 - bucket = hash(rec)
 - Εισαγωγή rec στο bucket, όπως στο στατικό κατακερματισμό
 - Αν load_factor > loadThrs (κατώφλι)
 - new_bucket = Δημιούργησε νέο bucket
 - Για κάθε εγγραφή r στο split bucket
 - bucket1 = hash(r)
 - Εισαγωγή του r στο bucket1, όπως στο στατικό κατακερματισμό

Εισαγωγή με Linear Hash: Παρατηρήσεις

- Όπως φαίνεται και από την προηγούμενη διαφάνεια, η διαδικασία είναι ίδια με το στατικό hashing, μέχρι το σημείο της διάσπασης (split)
- Οι εγγραφές που ξαναμοιράζονται (δηλαδή του split bucket), αφορούν και το αρχικό block αλλά και τα block υπερχείλισης
- Στη διάσπαση, όταν ξανατρέχουμε τη hash(r), αυτή *πρέπει να επιστρέφει* ως bucket, είτε το split (αυτό που διασπάστηκε), είτε το καινούριο (new_bucket) (θα δείξουμε πως γίνεται αυτό)

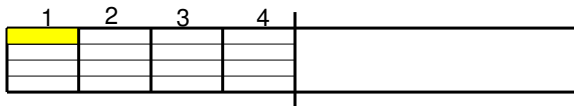
Linear Hash – Παράδειγμα (1)

- Ξεκινάμε με:
 - #bucket = 4
 - capacity = 4
 - split = 1 (η αρίθμηση ξεκινάει από 1)
 - #recs = 0
 - loadThrs = 0.85 (κατώφλι διάσπασης)
 - level = 0

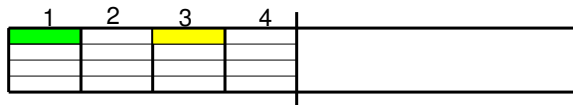


Linear Hash – Παράδειγμα (2)

- hash(rec) = 1
- #bucket = 4
- split = 1
- #recs = 1
- level = 0
- loadFctr = $1 / (4 * 4) = 0.0625$
- Αφού loadFctr < 0.85, δεν κάνουμε split



- hash(rec) = 3
- #bucket = 4
- split = 1
- #recs = 2
- level = 0
- loadFctr = $2 / 16 = 0.125 < 0.85 \Rightarrow$ δεν κάνουμε split

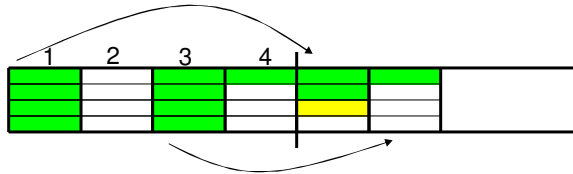


Linear Hash – Παράδειγμα (3)

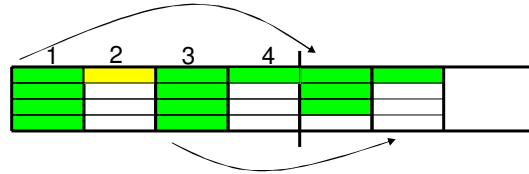
(Αρκετά Βήματα μετά)

- Έστω ότι έχουμε φτάσει στην κατάσταση που φαίνεται παρακάτω (τα βέλη δείχνουν τα block υπερχείλισης του κάθε bucket)

- #bucket = 4
- split = 1
- #recs = 13
- level = 0
- loadFctr = $13 / 16 = 0.8125 < 0.85$

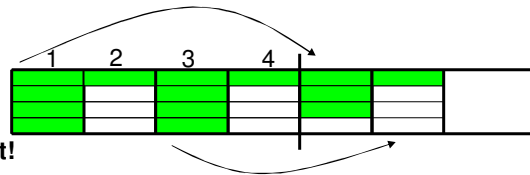


- #bucket = 4
- split = 1
- #recs = 14
- level = 0
- loadFctr = $14 / 16 = 0.8725 > 0.85 \Rightarrow$ Πρέπει να γίνει split!!!



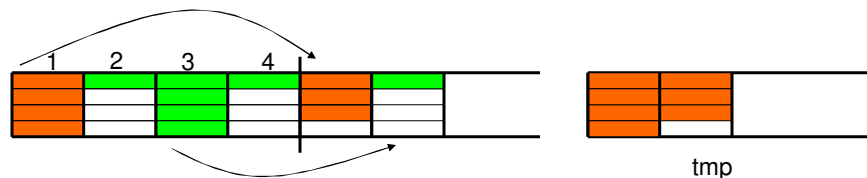
Linear Hash – Παράδειγμα (4)

Σύμφωνα με τα βήματα (slide 5),
πρώτα δημιουργούμε νέο bucket!

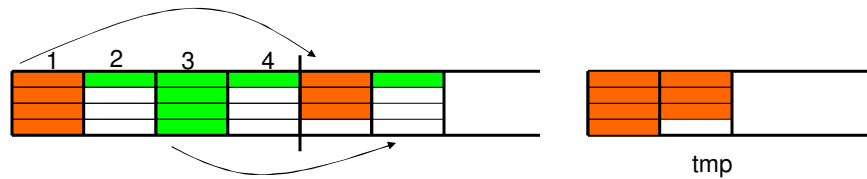


ΠΡΟΣΟΧΗ: Αν πούμε ότι το block 5 γίνεται αυτομάτως bucket, χάνουμε τις εγγραφές που υπάρχουν ήδη εκεί, οι οποίες είναι υπερχείλιση του block 1.

Για να μη χάσουμε τις εγγραφές του bucket, μπορούμε να τις αποθηκεύουμε προσωρινά σε άλλο αρχείο (ένας τρόπος είναι αυτός)



Linear Hash – Παράδειγμα (5)



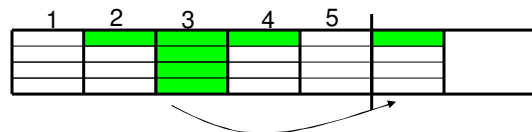
Αφού αντιγράψουμε τις εγγραφές στο tmp, **σβήνουμε / διαγράφουμε** τις εγγραφές από το αρχικό αρχείο.

3 τρόποι για «διαγραφή» εγγραφών:

- 1) Αν υποθέσουμε ότι έχουμε μετρητή για τις εγγραφές μέσα σε ένα block, μπορούμε να θέσουμε το μετρητή = 0.
- 2) Για κάθε εγγραφή μπορούμε να κρατάμε κάποιο flag (π.χ. 1 bit ή 1 byte – όχι περισσότερο) που να δηλώνει αν η εγγραφή ισχύει ή όχι. Διαγραφή τότε σημαίνει την αλλαγή του flag στην τιμή που δηλώνει ότι η εγγραφή δεν ισχύει.
- 3) Αν σας προβληματίζει ότι τα δεδομένα *φαίνονται*, μπορείτε να θέσετε όλα τα bytes του block σε κάποια τιμή που σας βολεύει.

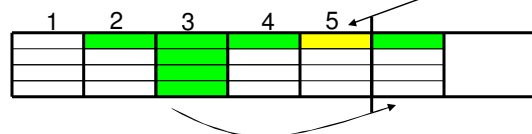
Linear Hash – Παράδειγμα (6)

- Έχω διαγράψει τις εγγραφές μου πλέον.
- Δημιουργώ το νέο bucket

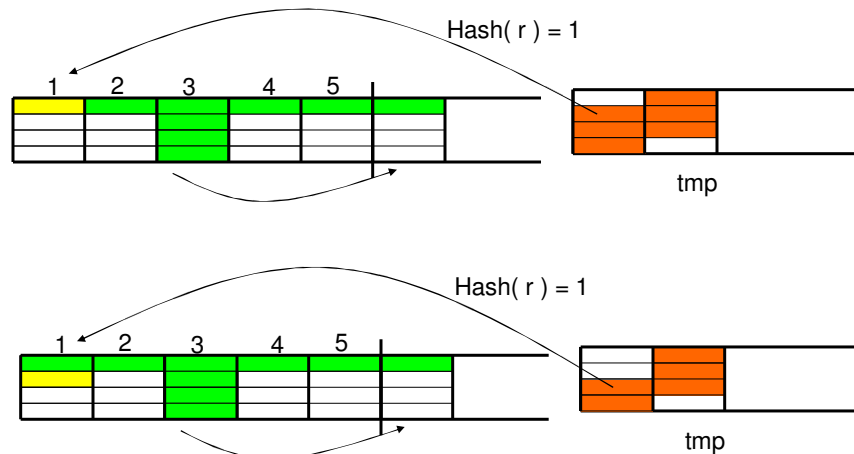


- #bucket = 4 (παραμένει σταθερό!!!!)
- split = 1
- level = 0 (ούτε αυτό αλλάζει)

- Διαβάζω τις εγγραφές από το tmp, μία – μία
- Hash(r) = 5

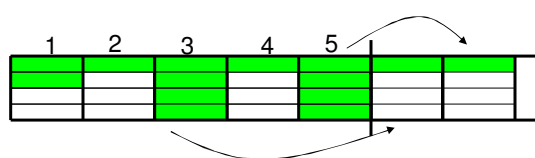


Linear Hash – Παράδειγμα (7)



Και συνεχίζουμε, μέχρι να εισάγουμε όλες τις εγγραφές από το tmp στο αρχικό αρχείο που έχουμε...

Linear Hash – Παράδειγμα (8)



Το νέο bucket 5, το διαχειριζόμαστε σαν κανονικό bucket. Άρα, μπορεί να έχει και αυτό block υπερχείλισης, τα οποία προκύπτουν από τις εγγραφές που επαναφέραμε από το tmp

- Αφού οι εγγραφές ξαναμοιράστηκαν, ενημερώνουμε τις τιμές των πληροφοριών που κρατάμε
- #bucket = 4 (δεν αλλάζει!!!!)
- **split = 2**
- level = 0
- #recs = 14

Ολοκληρώνοντας τη διαδικασία της διάσπασης του αρχικού bucket, πρέπει να αυξήσουμε την τιμή split. Έτσι, την επόμενη φορά θα διασπάσουμε το επόμενο bucket που ακολουθεί.

Linear Hash – Διάσπαση (1)

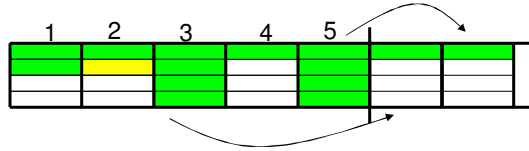
- Η $hash(r)$, πάνω στις εγγραφές του tmp, είπαμε ότι πρέπει να επιστρέφει τιμές είτε split, είτε το νέο bucket. Πώς γίνεται αυτό?
 - Αν η hash εσωτερικά κάνει $\% buckets$, θα επιστρέφει τιμές στο διάστημα $[0, buckets-1]$ -> άρα δε μας κάνει
 - Αντ' αυτού, κάνουμε $\% (2 * buckets)$.
- Η συνάρτηση που κάνει $\% buckets$ είναι η h_i
- Η συνάρτηση που κάνει $\% (2 * buckets)$ είναι η h_{i+1}
- Στη γενική μορφή είναι
$$h_{level} = key \% ((2^{level}) * buckets)$$
όπου το key είναι οποιοσδήποτε θετικός ακέραιος

Linear Hash – Διάσπαση (2)

- **Ερώτηση:** Γιατί να μην κάνω το $buckets = 5$ (αφού προσέθεσα ένα έτσι κι αλλιώς) και να κάνω απλά $\% buckets$? Δε θα δουλέψει αυτό?
- **Απάντηση:** Έστω ότι στο bucket 1 υπήρχαν τα πολλαπλάσια του 4 (π.χ. $bucket = (id \% 4) + 1$). Άρα, στο bucket 1 υπήρχαν οι τιμές: 0, 4, 8, 12, 16, κλπ.
Αν αυξήσω το bucket κατά 1, και κάνω $\% bucket$, έχω:
 - $(0\%5) + 1 \rightarrow 1$ OK
 - $(4\%5) + 1 \rightarrow 5$ OK
 - $(8\%5) + 1 \rightarrow 4$ ΠρόβλημαΆρα, **όχι**, δε δουλεύει με αυτό τον τρόπο

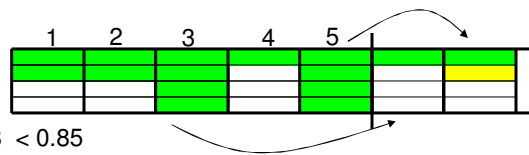
Linear Hash – Παράδειγμα (9)

- #bucket = 4
- split = 2
- #recs = 15
- level = 0
- loadFctr = $15 / (4 * 5) = 0.75 < 0.85$



Προσοχή! Στον υπολογισμό του παράγοντα φόρτωσης χρησιμοποιείται ο πραγματικός αριθμός από buckets

- #bucket = 4
- split = 2
- #recs = 16
- loadFctr = $16 / 20 = 0.8 < 0.85$
- level = 0



- Εγγραφές μπορούν να εισάγονται κανονικά και στο bucket που δημιουργήθηκε, με τον ίδιο ακριβώς τρόπο όπως και σε κάθε άλλο.

Διαχείριση των νέων buckets στην εισαγωγή

- **Ερώτηση:** Αν η τιμή του buckets δεν έχει αλλάξει, πώς γίνεται να εισάγονται εγγραφές και στο 5^ο bucket?
- **Απάντηση:**
 - Γι' αυτό ανά πάσα στιγμή χρειαζόμαστε 2 συναρτήσεις κατακερματισμού: την h_i και την h_{i+1}
 - Για την εισαγωγή μίας εγγραφής rec ακολουθούμε τη διαδικασία:

$$m = h_i(\text{rec})$$

$$\text{Av}(m < \text{split})$$

$$m = h_{i+1}(\text{rec})$$
 - Το m πλέον έχει το bucket που πρέπει να μπει η εγγραφή
 - Οι 2 συναρτήσεις είναι της μορφής που υπάρχουν στο slide 15
 - Οι συναρτήσεις h και η τιμή split πρέπει να είναι συνεπείς ως προς την τιμή που ξεκινάνε, αλλιώς η ανισότητα μπορεί να ισχύει σε λάθος περιπτώσεις..

Διαχείριση των νέων buckets στην εισαγωγή (2) - Παράδειγμα

- Έστω ότι έρχεται μια νέα εγγραφή rec που το key είναι 20. Δηλαδή, $h'(rec) = 20$.
- Έχουμε:

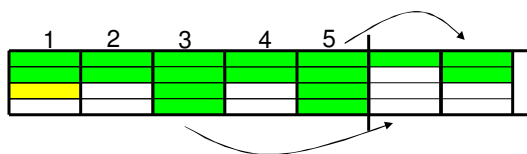
$$20 \% (2^0 * 4) + 1 = 20 \% 4 + 1 = 1$$
 Επειδή $1 < split = 2$

$$20 \% (2^1 * 4) + 1 = 4 + 1 = 5$$
 Άρα μία εγγραφή με $h'(rec) = key = 20$, θα μπει στο bucket 5. Αντίθετα, αν $h'(rec) = 24 \rightarrow bucket = 1$
- Η h' επιστρέφει θετικούς ακεραίους, χωρίς να περιορίζεται σε κάποιο εύρος τιμών

Linear Hash – Παράδειγμα (10)

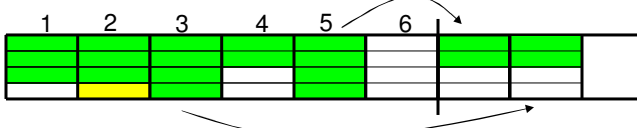
(Πολλά Βήματα μετά)

- #bucket = 4
- split = 2
- #recs = 18
- level = 0
- loadFctr = $18 / 20 = 0.9 > 0.85 \Rightarrow$ γίνεται πάλι διάσπαση!
 - Στην 17^η εισαγωγή, ήταν $17 / 20 = 0.85$. Αφού δεν ξεπερνάει το threshold, δεν γίνεται split
- Το bucket που θα διασπαστεί είναι, όπως πάντα, αυτό που δείχνει το split. Έτσι, παρ' ότι το block 6 είναι υπερχείλιση του bucket 3, διάσπαση θα γίνει στο bucket 2!
- Αυτό φυσικά σημαίνει ότι δεν πρέπει να χάσουμε τις εγγραφές του block 6, που είναι υπερχείλιση του 3! Ανεξάρτητα τίνος bucket είναι το block υπερχείλισης, τα data δεν πρέπει να χαθούν!
- Το να μεταφέρω τις εγγραφές του bucket 2 στο tmp δεν αρκεί για να μη χάσω τις εγγραφές μου από το bucket 3!



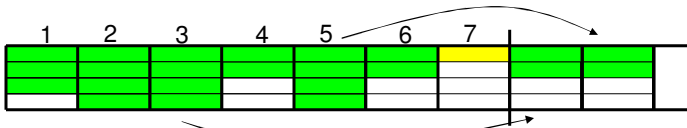
Linear Hash – Παράδειγμα (11)

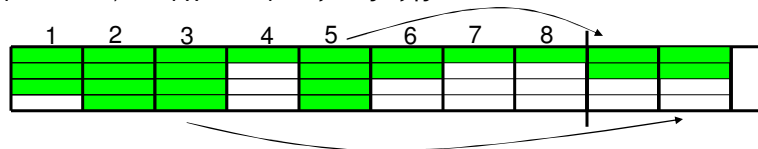
(Μερικά Βήματα ακόμα)

- #bucket = 4
 - split = 3
 - #recs = 21
 - level = 0
 - loadFctr = $21 / 24 = 0.875 > 0.85 \Rightarrow$ διάσπαση!
- 
- Το bucket 6 δεν έχει πάρει εγγραφές μέχρι στιγμής (έτυχε)
 - Το overflow block του bucket 3 έχει μεταφερθεί στο τέλος (και έχει προστεθεί 1 εγγραφή)
 - Το bucket 3 είναι το επόμενο προς διάσπαση. Δεδομένου ότι ξεπεράσαμε το κατώφλι, κάνουμε διάσπαση στο block 3. Κατόπιν το split γίνεται 4.
 - Προσοχή και πάλι στο ότι το block που θα γίνει το bucket 7 είναι overflow block του bucket 5 και όχι του block που θα διασπαστεί! Και πάλι πρέπει να «σώσουμε» τα δεδομένα πριν το 7ο block μετατραπεί σε bucket

Linear Hash – Παράδειγμα (12)

(Μερικά Βήματα ακόμα)

- #bucket = 4
 - split = 4
 - #recs = 24
 - level = 0
 - loadFctr = $24 / 28 = 0.85714 > 0.85 \Rightarrow$ διάσπαση!
- 
- Κάνουμε διάσπαση στο 4ο bucket, σύμφωνα με την τιμή που έχει το split
 - Το overflow block του bucket 3 πρέπει να μεταφερθεί ξανά, για να μην χάσουμε τα δεδομένα του, όπως κάναμε και τις δύο τελευταίες φορές.
 - Έτσι, αφού γίνει διάσπαση και του bucket 4 (και ανακατανομή των δεδομένων του) το αρχείο θα μοιάζει ως εξής:



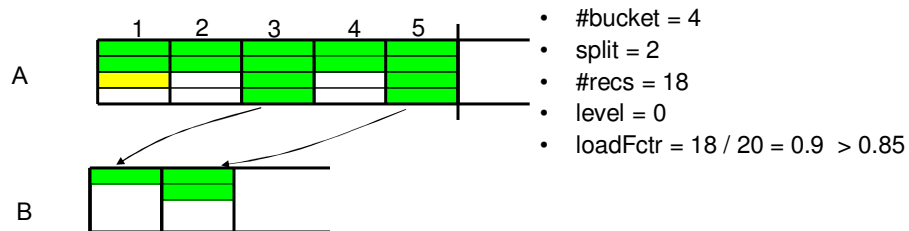
Linear Hash – Παράδειγμα (13)

- Με την προηγούμενη διαδικασία, κάναμε διάσπαση και του τελευταίου αρχικού bucket. Δηλαδή, και τα 4 αρχικά buckets έχουν διασπαστεί από 1 φορά το καθένα
- Όταν φτάσουμε στο σημείο όπου γίνει split και το τελευταίο bucket (για το συγκεκριμένο level), τότε:
 - Το level αυξάνει κατά 1
 - Το split αρχίζει πάλι από την αρχή (split = 1)
- Μετά την 1^η φορά, το split θα πρέπει να πάρει τιμές 1,2,3, ..., 7, 8 οπότε και το level αυξάνει ξανά και το split αρχίζει πάλι από το 1 κ.ο.κ.

Linear Hash – 2 αρχεία

- Αν εξαιρέσουμε το tmp αρχείο, που χρησιμοποιούνταν για την αντιγραφή των εγγραφών του bucket προς διάσπαση, τα αρχικά block ενός bucket, αλλά και τα block υπερχείλισης, ήταν στο ίδιο αρχείο
- Όπως φάνηκε και από την «εικονική» εκτέλεση, αυτό μας ανάγκαζε να μεταφέρουμε block μέσα στο ίδιο αρχείο, για να μη χάσουμε τα δεδομένα
- Αντ' αυτού, μπορούμε να έχουμε 2 αρχεία: 1 για τα βασικά block ενός bucket και 1 με τα block υπερχείλισης όλων των buckets

Linear Hash – 2 αρχεία - Παράδειγμα



Όπως φαίνεται στην περίπτωση αυτή, μπορώ να κάνω απλά allocate ένα νέο block στο αρχείο A, χωρίς να χρειαστεί να μετακινήσω πρώτα το overflow του bucket 3 (όπως συνέβη με το 1 αρχείο). Αυτομάτως το νέο block γίνεται το bucket 6!

Για να διασπάσω τις εγγραφές του bucket 2, μπορώ ξανά να χρησιμοποιήσω το tmp αρχείο. Και σε αυτή την περίπτωση, αν υπάρχουν block υπερχείλισης στο αρχείο B, πρέπει να τα λαμβάνω υπόψη.

Linear Hash – Εναλλακτικές (1)

- Όταν έχω διασπάσει όλα τα block της γύρας που είμαι τώρα (level), είπαμε ότι αυξάνω το level κατά 1 και αρχίζω από την αρχή.
- Η αύξηση του level κατά 1, σημαίνει ότι χρησιμοποιώ πια την h_{i+1} αντί για την h_i (οπότε η νέα h_{i+1} είναι η παλιά h_{i+2})
- Τέλος, ισχύει ότι $h_{i+1} = 2^{(i+1)} * \text{buckets} = 2 * 2^i * \text{buckets} = 2^i * (2 * \text{buckets})$, δηλαδή σαν να χρησιμοποιώ απλά διπλάσιο αριθμό από buckets. Αυτό ισχύει για όλα τα i , και άρα αρκεί να διπλασιάζω τα buckets όποτε θα αύξανα το level. Με τον τρόπο αυτό είναι σαν να έχω πάντα level = 0 οπότε το αγνοώ.

Linear Hash – Εναλλακτικές (2)

- Αντί να μεταφέρω τα δεδομένα του «προς διάσπαση bucket» σε ένα tmp αρχείο, μπορώ να χρησιμοποιήσω ένα block που θα είναι στην κύρια μνήμη όπου:
 - Διαβάζω το αρχικό block του bucket στο buffer αυτό
 - Διαγράφω τις εγγραφές του αρχικού block
 - Μοιράζω τις εγγραφές από το buffer
 - Ακολουθώ τα block υπερχείλισης και για κάθε ένα
 - Διαβάζω το block υπερχείλισης στο buffer
 - Διαγράφω τις εγγραφές του αντίστοιχου block υπερχείλισης
 - Μοιράζω τις εγγραφές από το buffer

Linear Hash – Εναλλακτικές (3)

- Είτε χρησιμοποιώ 1 αρχείο, είτε δύο, είναι πιθανό να δημιουργούνται «τρύπες» ή αλλιώς «ορφανά» block.
- Είπαμε ότι μετά τη διάσπαση, η εισαγωγή (με εξαίρεση την αλλαγή στη hash function) γίνεται όπως στο στατικό κατακερματισμό
- Με τον «κλασσικό» τρόπο, αυτό σημαίνει ότι δεσμεύονται νέα block στο τέλος του αρχείου ως block υπερχείλισης, αγνοώντας τα ήδη υπάρχοντα.
- Τα block που πριν ήταν block υπερχείλισης αλλά μετά τη διάσπαση δεν ανήκουν σε κανένα bucket (επειδή δεσμεύτηκαν νέα), τα ονομάζουμε «ορφανά».
- Μια εναλλακτική (και πιο σωστή) πρακτική είναι να επαναχρησιμοποιούνται τα «ορφανά» block και να δεσμεύεται νέο block, όταν δεν υπάρχουν άλλα «ορφανά» block.
- Αν και θέλουμε να επαναχρησιμοποιούμε τέτοια block, είναι **λάθος** να σαρώνουμε κάθε φορά το αρχείο για να τα εντοπίσουμε!
- Αντί για συνεχή σάρωση, μπορούμε να κρατάμε στη μνήμη ποια είναι αυτά τα block και να τα χρησιμοποιούμε όποτε χρειάζεται (αντί για νέο allocate)

Linear Hash – Εναλλακτικές (4)

- Στο στατικό κατακερματισμό, στο 1^ο block (header) μπορούσαμε να κρατάμε ένα πίνακα (table ή hashtable) από ακεραίους, που να δείχνει για το i-οστό bucket ποιο είναι το αρχικό του block (αντί για απ' ευθείας αντιστοιχία i-οστού bucket με i-οστό block)
- Αντίστοιχο πίνακα μπορούμε να διατηρούμε και στο Linear Hashing, *όμως*:
 - Πλέον δεν ισχύει ο περιορισμός ότι θα υπάρχουν μέχρι N buckets, αφού συνεχώς προστίθενται νέα
- Αυτό σημαίνει ότι πρέπει να υποστηρίζετε οσοδήποτε μεγάλο πλήθος από buckets
 - Αυτό μπορεί να γίνει με «block» υπερχείλισης, αλλά για το header. Δηλαδή, αν δεν χωράνε άλλα στοιχεία του table στο header block, δεσμεύετε νέο block όπου θα συνεχίζει το header
- Αξίζει να αναφερθεί ότι η εναλλακτική αυτή **δεν** έχει μεγάλη πρακτική εφαρμογή για το Linear Hashing, σε αντίθεση με το Static Hashing