Οργάνωση Αρχείων

Αρχεία

- Τα δεδομένα συνήθως αποθηκεύονται σε αρχεία στο δίσκο
- Η μεταφορά δεδομένων από το δίσκο στη μνήμη και από τη μνήμη στο δίσκο γίνεται σε *μονάδες blocks*

Βασικός στόχος η ελαχιστοποίηση της επικοινωνίας με το δίσκο:

ελαχιστοποίηση του αριθμού των blocks **που μεταφέρονται μεταξύ της πρωτεύουσας (κύριας μνήμης, cache – ενδιάμεση μνήμη – buffers-καταχωρητές) και της δευτερεύουσας αποθήκευσης (δίσκος)**

Αρχεία

Τα δεδομένα συνήθως αποθηκεύονται με τη μορφή εγγραφών

Οι εγγραφές συνήθως περιγράφουν οντότητες (σχέσεις) και τα γνωρίσματά τους

Ένα αρχείο είναι λογικά οργανωμένο σε μια ακολουθία από εγγραφές που μπορεί να βρίσκονται αποθηκευμένες σε πολλές σελίδες (pages) – θα θεωρούμε page = block

Blobs

Εγγραφές

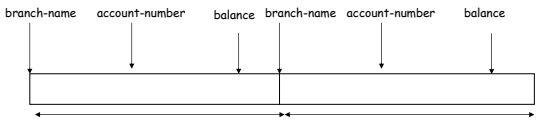
Πως οργανώνονται τα πεδία μέσα σε μία εγγραφή

Εγγραφές σταθερού και μεταβλητού μήκους

```
type film = record
        branch-name: char(22);
        account-number: char(20);
        balance:real;
        end
```

Έστω κάθε char 1 byte - real 8 bytes

Κάθε εγγραφή 50 bytes



Εγγραφές

Γιατί είναι προτιμότερες οι εγγραφές σταθερού μήκους: *εύκολος ο εντοπισμός ενός πεδίου και η διατήρηση πληροφορίας για «άδειες» θέσεις*

Εγγραφές

Πως προκύπτουν οι εγγραφές μεταβλητού τύπου;

Στο σχεσιακό μοντέλο κάθε εγγραφή (πλειάδα) μιας σχέσης περιέχει το ίδιο πλήθος πεδίων (αριθμό γνωρισμάτων). Άρα

- Εγγραφές του ίδιου τύπου αλλά έχουν ένα ή περισσότερα πεδία *μεταβλητού μεγέθους*
- Ανάμεικτο (mixed) αρχείο: εγγραφές διαφορετικού τύπου

Εγγραφές

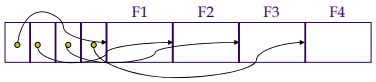
- Αποθήκευση των πεδίων συνεχόμενα, χωρισμένα με διαχωριστές (ειδικούς χαρακτήρες που δεν εμφανίζονται ως δεδομένα)

Εγγραφές

- Χώρο στην αρχή κάθε εγγραφής – πίνακας ακεραίων I[j] όπου j η μετατόπιση (offset) της j-οστής εγγραφής (κρατά την αρχή του j-οστού πεδίου) + τη μετατόπιση του τέλους της εγγραφής

απευθείας πρόσβαση σε οποιαδήποτε πεδίο

καλό χειρισμό της τιμής null



Εγγραφές

- Ως εγγραφές σταθερού μήκους, θεωρώντας το μέγιστο μέγεθος για κάθε εγγραφή

Παράγοντας Ομαδοποίησης

Η μονάδα μεταφοράς μεταξύ δίσκου και μνήμης είναι ένα block δίσκου

Έστω εγγραφές σταθερού μήκους

Όταν B ≥ R περισσότερες από μια εγγραφή ανά block - κάθε εγγραφή σε ένα μόνο block

Παράγοντας ομαδοποίησης (blocking factor), όταν B ≥ R

bfr = ⌊ (B / R) ⌋, όπου B μέγεθος block σε byte και R μέγεθος εγγραφής σε bytes

Δηλαδή, πόσες «ολόκληρες» εγγραφές χωρούν σε ένα block

Εκτεινόμενη καταχώρηση

Εκτεινόμενη και μη εκτεινόμενη καταχώρηση εγγραφών

- **Μη εκτεινόμενη** (unspanned) οργάνωση:οι εγγραφές δεν επιτρέπεται να διασχίζουν τα όρια ενός block
  - Αχρησιμοποίητος χώρος: B - bfr * R bytes ανά block
  - Πιο εύκολη η προσπέλαση
- **Εκτεινόμενη** (spanned) οργάνωση: αποθήκευση μέρους μιας εγγραφής σε ένα block και το υπόλοιπο σε ένα άλλο block - δείκτης στο τέλος του πρώτου τμήματος δείχνει στο block που περιέχει το υπόλοιπο

Εκτεινόμενη καταχώρηση

b: Αριθμός blocks για την αποθήκευση ενός αρχείου r εγγραφών:

$$b = \lceil (r/bfr) \rceil$$

Τοποθέτηση block αρχείου στο δίσκο

συνεχόμενη τοποθέτηση (contiguous allocation) τα block του αρχείου τοποθετούνται σε διαδοχικά blocks του δίσκου

συνδεδεμένη τοποθέτηση (linked allocation) κάθε block του αρχείου περιλαμβάνει ένα δείκτη προς το επόμενο block του αρχείου

Εύκολη επέκταση - πιο αργή ανάγνωση όλου του αρχείου

συστάδες διαδοχικών blocks δίσκου (τμήματα (segments) ή επεκτάματα (extents)

ευρετηριοποιημένη τοποθέτηση (indexed allocation)

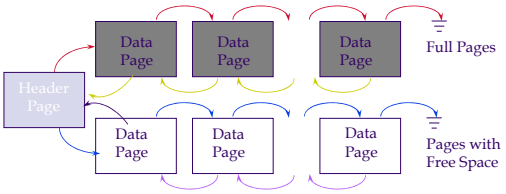
Επικεφαλίδες αρχείων

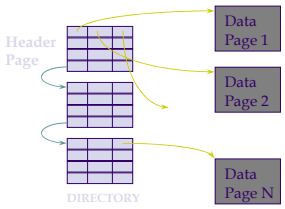
Μια **επικεφαλίδα ή περιγραφέας αρχείου** (file header ή file descriptor) περιέχει πληροφορίες σχετικά με ένα αρχείο που είναι απαραίτητες στα προγράμματα που προσπελαύνουν τις εγγραφές του αρχείου

Πληροφορίες για προσδιορισμό διεύθυνσης των blocks αρχείου στο δίσκο + περιγραφές μορφοποίησης εγγραφών

Αποθηκεύεται στο αρχείο

Θεωρούμε ότι «ξέρουμε» σε ποιο block είναι αποθηκευμένη η i-οστή σελίδα του αρχείου





Αρχείο
από σελίδες
από εγγραφές

Βασικές λειτουργίες:

- Εισαγωγή/διαγραφή/τροποποίηση εγγραφής
- Εντοπισμός μια συγκεκριμένης εγγραφής
- Διάσχιση (scan) όλων των εγγραφών του αρχείου

Οργάνωση αρχείων: πως είναι τοποθετημένες οι εγγραφές ενός αρχείου όταν αποθηκεύονται στο δίσκο

- Αρχεία Σωρού
- Ταξινομημένα Αρχεία
- Κατακερματισμένα Αρχεία

B blocks - R εγγραφές ανά block - D εγγραφή/ανάγνωση - *C* χρόνος επεξεργασίας ανά εγγραφή

D = 15 milliseconds -- *C* = 100 nanoseconds

Αρχεία Σωρού

Αρχείο Σωρού (heap file ή pile file): Οι εγγραφές τοποθετούνται στο αρχείο με τη σειρά που εισάγονται

Μη ταξινομημένο αρχείο

1. Εισαγωγή

2 * D + *C*

2. Αναζήτηση

0.5 * B * (D + R * *C*)

B blocks
R εγγραφές ανά block
D χρόνος μεταφοράς block
C χρόνος επεξεργασίας ανά εγγραφή

3. Διαγραφή εγγραφής

Σημάδι διαγραφής

Περιοδική αναδιοργάνωση

Χρόνος Αναζήτησης + (*C* + D)

4. Τροποποίηση εγγραφής

- εγγραφή μεταβλητού μήκους

5. Σάρωση (scan) Ανάγνωση όλων των εγγραφών

B*(D+R**C*)

6. Ανάγνωση όλων των εγγραφών σε διάταξη

Εξωτερική ταξινόμηση συνήθως μια παραλλαγή της ταξινόμησης με συγχώνευση

Ταξινομημένα Αρχεία

Φυσική διάταξη των εγγραφών ενός αρχείου με βάση την τιμή ενός από τα πεδία του το οποίο λέγεται **πεδίο διάταξης** (ordering field)

Διατεταγμένο ή φυσικό αρχείο

- Αν το πεδίο διάταξης είναι και κλειδί τότε λέγεται και **κλειδί διάταξης**

1. Εισαγωγή

i. Εύρεση της σωστής θέσης της εγγραφής στο αρχείο

ii. Μετακίνηση εγγραφών για να κάνουμε χώρο για την εισαγωγή της

Κατά μέσο όρο μετακίνηση των μισών εγγραφών

Χρόνος αναζήτησης + 2 * (0.5 * B *(D+ R**C*))

B blocks
R εγγραφές ανά block
D χρόνος μεταφοράς block
C χρόνος επεξεργασίας ανά εγγραφή

1. Εισαγωγή (συνέχεια)

- Διατήρηση κάποιου αχρησιμοποίητου χώρου ανά block
- Δημιουργία ενός προσωρινού μη διατεταγμένου αρχείου (αρχείο υπερχείλισης) - κυρίως αρχείο

2. Αναζήτηση εγγραφής (με επιλογή ισότητας)

αποδοτική αν η συνθήκη αναζήτησης είναι στο πεδίο ταξινόμησης

Έστω B blocks, αναζήτηση της εγγραφής με τιμή K στο πεδίο διάταξης

Σημείωση: Υποθέτουμε ότι οι διευθύνσεις των blocks του αρχείου είναι αποθηκευμένες στην επικεφαλίδα του αρχείου

2. Αναζήτηση εγγραφής (συνέχεια)

```
lower := 1; upper := B;
while (upper ≥ lower)
      i := (lower + upper) div 2;
      read block i
      if (K < τιμής διάταξης της πρώτης εγγραφής)
            upper := i - 1;
      else if (K > τιμής διάταξης της τελευταίας εγγραφής)
            lower := i + 1;
      else ...
```

Χρόνος: log B * (D + log R * C)

Συνθήκη πχ., <=

B blocks
R εγγραφές ανά block
D χρόνος μεταφοράς block
C χρόνος επεξεργασίας ανά εγγραφή

3. Διαγραφή εγγραφής

Μετακίνηση εγγραφών

Χρήση σημαδιού διαγραφής

4. Τροποποίηση εγγραφής

5. Ανάγνωση όλων των εγγραφών σε διάταξη

1. Τα δεδομένα αποθηκεύονται σε αρχεία στο δίσκο
2. Για να γίνει η επεξεργασία τους πρέπει να μεταφερθούν στη μνήμη
3. Η μονάδα μεταφοράς από το δίσκο στη μνήμη είναι ένα block
4. Ο χρόνος προσπέλασης (εγγραφής ή ανάγνωσης) ενός block διαφέρει και εξαρτάται από τη θέση του block – δε θα το εξετάσουμε στο μάθημα
5. Μας ενδιαφέρει η ελαχιστοποίηση του I/O (πολυπλοκότητα σε σχέση με blocks)

Ένα αρχείο είναι λογικά οργανωμένο σε μια ακολουθία από εγγραφές

Συνήθως ένα αρχείο ανά (σχήμα) σχέσης και μια εγγραφή αντιστοιχεί σε μια πλειάδα

Μη εκτεινόμενη (unspanned) οργάνωση:

οι εγγραφές δεν επιτρέπεται να διασχίζουν τα όρια ενός block

(-) Αχρησιμοποίητος χώρος

(+) Πιο εύκολη η προσπέλαση

Έστω B μέγεθος block σε byte και R μέγεθος εγγραφής σε bytes

Παράγοντας ομαδοποίησης (blocking factor), όταν B ≥ R

$bfr = \lfloor (B / R) \rfloor$

Πόσες εγγραφές χωρούν σε ένα block

b: Αριθμός blocks για την αποθήκευση ενός αρχείου r εγγραφών:

$b = \lceil (r/bfr) \rceil$

Για κάθε σχέση:

όνομα, αρχείο, δομή αρχείου (πχ αρχείο σωρού)
Όνομα και τύπο για κάθε γνώρισμα
Όνομα ευρετηρίου για κάθε ευρετήριο
Περιορισμοί ακεραιότητας

Για κάθε ευρετήριο:
Δομή (πχ B+ δέντρο) και κλειδιά αναζήτησης

Για κάθε όψη:
Το όνομα και τον ορισμό της

Επίσης, στατιστικά, μέγεθος του buffer pool, δικαιώματα προσπέλασης κλπ.

Ο κατάλογος αποθηκεύεται επίσης ως σχέση

Attr_Cat(attr_name, rel_name, type, position)

attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Αρχείο
από σελίδες (page, block)
από εγγραφές (πλειάδες)

Βασικές λειτουργίες:

- Εισαγωγή/διαγραφή/τροποποίηση εγγραφής
- Εντοπισμός (αναζήτηση) μια συγκεκριμένης εγγραφής με βάση συνθήκη ισότητας ή διαστήματος
- Διάσχιση (scan) όλων των εγγραφών του αρχείου

Βασικός στόχος η *ελαχιστοποίηση του αριθμού* των blocks που μεταφέρονται

Θεωρούμε ότι η πληροφορία για τη θέση στο δίσκο ενός block υπάρχει (π.χ., στην επικεφαλίδα του αρχείου)

Οργάνωση αρχείων: πως είναι τοποθετημένες οι εγγραφές ενός αρχείου όταν αποθηκεύονται στο δίσκο

- Αρχεία Σωρού (δεν υπάρχει διάταξη)
- Ταξινομημένα Αρχεία (διάταξη με βάση κάποιο πεδίο διάταξης)
- Αρχεία Κατακερματισμού

Εσωτερικός Κατακερματισμός (τα δεδομένα είναι στη μνήμη, όπως στις δομές δεδομένων)

Πίνακας κατακερματισμού με M θέσεις - κάδους (buckets)

h: *συνάρτηση κατακερματισμού*

h(k) = i ← Σε ποιο κάδο - τιμή από 0 έως M-1

Πεδίο αναζήτησης - Πεδίο κατακερματισμού

Εξωτερικός Κατακερματισμός (εφαρμογή σε δεδομένα αποθηκευμένα σε αρχεία)

Στόχος

h(k) = i ← Διεύθυνση (αριθμός) block του αρχείου που είναι αποθηκευμένη

Τιμή του πεδίου κατακερματισμού

Η εγγραφή με τιμή στο πεδίο κατακερματισμού k αποθηκεύεται στο i block (κάδο) του αρχείου

h: *συνάρτηση κατακερματισμού*

Ομοιόμορφη κατανομή των κλειδιών στους κάδους (blocks)

- Συνηθισμένη συνάρτηση κατακερματισμού:

h(k) = k mod M

Συχνά M πρώτος

- **Σύγκρουση (collision)**: όταν μια νέα εγγραφή κατακερματίζεται σε μία ήδη γεμάτη θέση
- **Καλή συνάρτηση κατακερματισμού**: κατανέμει τις εγγραφές ομοιόμορφα στο χώρο των διευθύνσεων (ελαχιστοποίηση συγκρούσεων και λίγες αχρησιμοποίητες θέσεις)
- **Ευριστικοί**:

-- αν r εγγραφές, πρέπει να επιλέξουμε το M ώστε το r/M να είναι μεταξύ του 0.7 και 0.9

-- όταν χρησιμοποιείται η mod τότε είναι καλύτερα το M να είναι πρώτος

Επίλυση Συγκρούσεων

1. *Ανοιχτή Διευθυνσιοδότηση* (open addressing): χρησιμοποίησε την επόμενη κενή θέση
2. *Αλυσιδωτή Σύνδεση* (chaining): για κάθε θέση μια συνδεδεμένη λίστα με εγγραφές υπερχείλισης
3. *Πολλαπλός Κατακερματισμός* (multiple hashing): εφαρμογή μιας δεύτερης συνάρτησης κατακερματισμού

Κάδος: μια συστάδα από συνεχόμενα blocks του αρχείου

h(k) = i ← Σχετική διεύθυνση του κάδου (ποιος κάδος του αρχείου)

Τιμή του πεδίου κατακερματισμού

Ένας πίνακας που αποθηκεύεται στην επικεφαλίδα του αρχείου μετατρέπει τον αριθμό κάδου στην αντίστοιχη διεύθυνση block

0	διεύθυνση 1ου block του κάδου στο δίσκο
1	διεύθυνση 1ου block του κάδου στο δίσκο
2	διεύθυνση 1ου block του κάδου στο δίσκο
…	…
M-1	διεύθυνση 1ου block του κάδου στο δίσκο

Συγκρούσεις - αλυσιδωτή σύνδεση - εγγραφές υπερχείλισης ανά κάδο

1. **Ανάγνωση όλου του αρχείου (scan)**

   Έστω ότι διατηρούμε κάθε κάδο γεμάτο κατά 80% άρα ένα αρχείο με μέγεθος B blocks χρειάζεται 1.25 B blocks

   $1.25 * B * (D + R * C)$

2. **Αναζήτηση**

   Συνθήκη ισότητας και μόνο ένα block ανά κάδο: $D + R * C$

   Αν συνθήκη περιοχής (διαστήματος): scan!

Κόστος: μεταφορά blocks (I/O)

	Σωρός	Ταξινομημένο	Κατακερματισμένο
Ανάγνωση του αρχείου	B	B	1.25B
Αναζήτηση με συνθήκη ισότητας	0.5 B	logB	1
Αναζήτηση με συνθήκη περιοχής	B	logB + ταιριάσματα	1.25 B
Εισαγωγή	2	αναζήτηση + B	2
Διαγραφή	αναζήτηση + 1	αναζήτηση + B	αναζήτηση + 1

Πρόβλημα:

Έστω M κάδους και r εγγραφές ανά κάδο - το πολύ M * r εγγραφές (αλλιώς μεγάλες αλυσίδες υπερχείλισης)

Δυναμικός κατακερματισμός

Δυναμικός Εξωτερικός Κατακερματισμός

- Δυαδική αναπαράσταση του αποτελέσματος της συνάρτησης κατακερματισμού, δηλαδή ως μια ακολουθίας δυαδικών ψηφίων
- Κατανομή εγγραφών με βάση την τιμή των αρχικών *(ή τελικών)* ψηφίων

Δυναμικός Εξωτερικός Κατακερματισμός

- Το αρχείο ξεκινά με ένα μόνο κάδο
- Μόλις γεμίσει ένας κάδος διασπάται σε δύο κάδους με βάση την τιμή του 1ου δυαδικού ψηφίου των τιμών κατακερματισμού -- δηλαδή οι εγγραφές που το πρώτο ψηφίο της τιμής κατακερματισμού τους είναι 1 τοποθετούνται σε ένα κάδο και οι άλλες (με 0) στον άλλο
- Νέα υπερχείλιση ενός κάδου οδηγεί σε διάσπαση του με βάση το αμέσως επόμενο δυαδικό ψηφίο κοκ

Δυναμικός Εξωτερικός Κατακερματισμός

Έτσι δημιουργείται μια δυαδική δενδρική δομή που λέγεται **κατάλογος** (dirtectory) ή **ευρετήριο** (index) με δύο ειδών κόμβους

- εσωτερικούς: που καθοδηγούν την αναζήτηση
- εξωτερικούς: που δείχνουν σε ένα κάδο

Δυναμικός Εξωτερικός Κατακερματισμός (Παράδειγμα)

Χρήση των τελευταίων bits της δυαδικής αναπαράστασης

1	000001
4	000100
5	000101
7	000111
10	001010
12	001100
15	001111
16	010000
19	010011
21	010101
32	100000
13	001101
20	010100

4 εγγραφές ανά κάδο

Δυναμικός Εξωτερικός Κατακερματισμός

Αλγόριθμος αναζήτησης

```
h := τιμή κατακερματισμού
t  := ρίζα του δέντρου
i := 1
while (t εσωτερικός κόμβος)
        if (i-οστό bit του h είναι 0)
                t := αριστερά του t
        else t := δεξιά του t
     i := i +1
```


Δυναμικός Εξωτερικός Κατακερματισμός

- *Που αποθηκεύεται ο κατάλογος*

  στη μνήμη, εκτός αν είναι πολύ μεγάλος

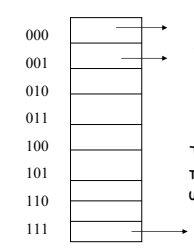
  τότε στο δίσκο – οπότε θα απαιτούνται επιπρόσθετες προσπελάσεις

- Δυναμική επέκταση αλλά *μέγιστος αριθμός* επιπέδων (το πλήθος των δυαδικών ψηφίων της συνάρτησης κατακερματισμού)
- Ισοζύγιση
- Συνένωση κάδων (δυναμική συρρίκνωση)

Επεκτάσιμος Εξωτερικός Κατακερματισμός

Extendible hashing

Ο **κατάλογος** είναι ένας πίνακας με 2^d διευθύνσεις κάδων (**d: *ολικό βάθος* του καταλόγου**)

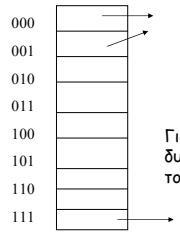


Κάδος για τις εγγραφές με τιμές κατακερματισμού που αρχίζουν από (ή τελειώνουν σε) 000

Τα πρώτα (ή τα τελευταία) d ψηφία της τιμής κατακερματισμού χρησιμοποιούνται ως δείκτης στον πίνακα

Επεκτάσιμος Εξωτερικός Κατακερματισμός

Δε χρειάζεται ένας διαφορετικός κάδος για κάθε μία από τις 2^d θέσεις - μπορεί η θέση του πίνακα να δείχνει στη διεύθυνση του ίδιου κάδου αν αυτές χωράνε σε ένα κάδο



Κάδος για τις εγγραφές με τιμές κατακερματισμού που αρχίζουν από 00

Για κάθε κάδο, **τοπικό βάθος d'** ο αριθμός των δυαδικών ψηφίων στα οποία βασίζεται η χρήση του κάδου

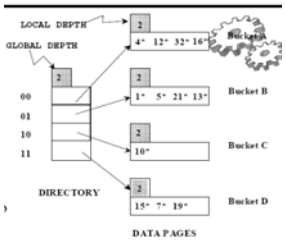
Επεκτάσιμος Εξωτερικός Κατακερματισμός (Παράδειγμα)

Χρήση των τελευταίων bits της δυαδικής αναπαράστασης

1	000001
4	000100
5	000101
7	000111
10	001010
12	001100
15	001111
16	010000
19	010011
21	010101
32	100000
13	001101

4 εγγραφές ανά κάδο

Επεκτάσιμος Εξωτερικός Κατακερματισμός (Παράδειγμα)



Χρήση των τελευταίων bits της δυαδικής αναπαράστασης

1	000001
4	000100
5	000101
7	000111
10	001010
12	001100
15	001111
16	010000
19	010011
21	010101
32	100000
13	001101

Επεκτάσιμος Εξωτερικός Κατακερματισμός

Η τιμή του d μπορεί να αυξάνεται (μέχρι $2^κ$, κ: αριθμός δυαδικών ψηφίων της τιμής κατακερματισμού) ή να μειώνεται

- **Αύξηση της τιμής του d**

  Όταν ένας κάδος με τιμή d' = d υπερχειλίσει

  Διπλασιασμός του πίνακα

  Δε χρειάζεται rehash (επανα-κερματισμό), διασπάμε κάθε κάδο

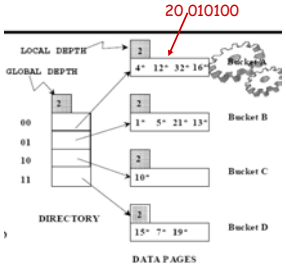
- **Μείωση της τιμής του d**

  Επίσης, κάθε φορά μόνο τον κάδο που υπερχείλισε

  Όταν για όλους τους κάδους d' < d

  Μείωση του μεγέθους του πίνακα στο μισό

Επεκτάσιμος Εξωτερικός Κατακερματισμός (Παράδειγμα)



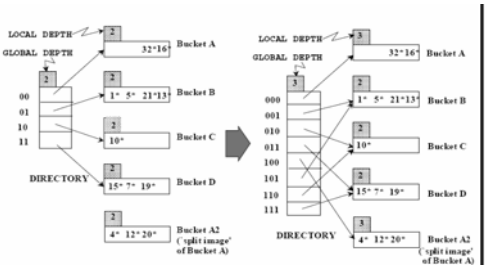
1	000001
4	000100
5	000101
7	000111
10	001010
12	001100
15	001111
16	010000
19	010011
21	010101
32	100000
13	001101

Διάσπαση -> Ολικό βάθος 3

Επεκτάσιμος Εξωτερικός Κατακερματισμός

1	000001
4	000100
5	000101
7	000111
10	001010
12	001100
15	001111
16	010000
19	010011
21	010101
32	100000
13	001101
20	010100

4 12 32 16 20 -> διάσπαση



Γραμμικός Κατακερματισμός

Θέλουμε να αποφύγουμε τη χρήση καταλόγου +

Διπλασιασμό μεγέθους του καταλόγου

Σημείωση: διατηρούμε λίστες υπερχείλισης

Χρησιμοποιεί μια οικογένεια από συναρτήσεις κατακερματισμού

$h_0(k)$, $h_1(k)$, …, $h_n(k)$

Κάθε συνάρτηση *διπλάσιους κάδους* από την προηγούμενη:

$h_0(k) = k \bmod M$, $h_1(k) = k \bmod 2M$, $h_2(k) = k \bmod 4M$, …,

$h_j(k) = k \bmod 2^j M$

Όταν συμβαίνει η πρώτη υπερχείλιση ενός κάδου, πάμε στην επόμενη συνάρτηση μέχρι να διασπαστούν όλοι οι κάδοι με αυτήν τη συνάρτηση

ΠΡΟΣΟΧΗ: δε διασπάμε τον κάδο που υπερχειλίζει, αλλά έναν-έναν τον κάδο με τη σειρά!

Αρχικά:

Βήμα Διάσπασης (ποια συνάρτηση χρησιμοποιούμε) αρχικά **j = 0**:

Πλήθος Διασπάσεων (στο τρέχον βήμα) αρχικά **n = 0**,

j -> ποια συνάρτηση χρησιμοποιούμε

n -> ποιο κάδο διασπάμε

Έστω αρχικά M κάδους αριθμημένους από 0 έως M - 1 και

αρχική συνάρτηση κατακερματισμού

$h_0(k) = k \bmod M$

Όταν συμβεί μια υπερχείλιση σε έναν οποιοδήποτε κάδο, **ο κάδος 0** χωρίζεται σε δύο κάδους: τον αρχικό κάδο 0 και ένα νέο κάδο M στο τέλος του αρχείου με βάση την συνάρτηση $h_1(k) = k \bmod 2M$

Βήμα Διάσπασης (ποια συνάρτηση χρησιμοποιούμε) j = 1

Πλήθος Διασπάσεων n = 1

Συνεχίζουμε γραμμικά, διασπώντας με τη σειρά τους κάδους 1, 2, 3, …

μέχρι να διασπαστούν όλοι οι «παλιοί» κάδοι

μια μεταβλητή n («Πλήθος Διασπάσεων») κρατάει ποιος κάδος έχει σειρά για διάσπαση

Βήμα διάσπασης (ποια συνάρτηση χρησιμοποιούμε) j = 1:

Πλήθος Διασπάσεων **n = m -1** :

Όταν συμβεί μια υπερχείλιση σε έναν οποιοδήποτε κάδο,

ο κάδος m -1 χωρίζεται σε δύο κάδους: τον αρχικό κάδο m - 1 και ένα νέο κάδο m + k - 1 στο τέλος του αρχείου με βάση την συνάρτηση $h_1(k) = k \bmod 2M$

Δηλαδή, σε κάθε υπερχείλιση χωρίζουμε όλους τους κάδους με τη σειρά ξεκινώντας από τον πρώτο κάδο

Συνεχίζουμε …

Όλοι οι κάδοι έχουν διασπαστεί όταν: n = M

Τότε έχουμε 2M κάδους

Όταν **n = M**,

μηδενίζουμε το n, n = 0

και για οποιαδήποτε νέα διάσπαση εφαρμόζουμε την

$h_2(k) = k \bmod 4M$

Διασπώντας πάλι τον κάδο 0, 1, … κ.τ.λ

Γενικά βήμα διάσπασης **j** (j = 0, 1, 2, …)

$h_j(k) = k \bmod 2^j M$,
και την $h_{j+1}(k)$ για διασπάσεις

32
9
44
31
5
25
7
11
36
14
18
10
35
30

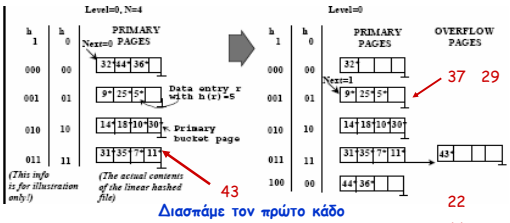
Κάθε κάδος 4 εγγραφές

Αρχικά 4 κάδους

ΠΡΟΣΟΧΗ: Δε χρησιμοποιούμε τη δυαδική αναπαράσταση

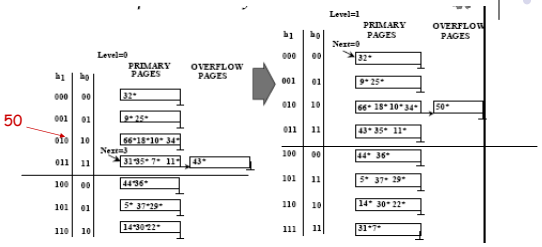
$h_0(k) = k \bmod 4$
$h_1(k) = k \bmod 8$

Για μη διασπασμένους κάδους: παλιά συνάρτηση
Για διασπασμένους κάδους: νέα συνάρτηση



Διασπάμε τον πρώτο κάδο

Βήμα διάσπασης 0 (χρήση h_0)
Πλήθος διασπάσεων = 0



Βήμα διάσπασης 0 (χρήση h_0)
Πλήθος διασπάσεων = 0

Αναζήτηση Εγγραφής (γενικά)

Τι χρειάζεται να ξέρουμε για να βρεθεί ο κάδος της εγγραφής k που ψάχνουμε;

- ποια συνάρτηση χρησιμοποιούμε (δηλαδή, το j)
- σε ποια διάσπαση βρισκόμαστε (δηλαδή το n)

Έστω ότι είμαστε στο βήμα j,

Τότε θα πρέπει να κοιτάξουμε είτε το

$h_j(k)$ αν ο κάδος δεν έχει διασπαστεί

ή το

$h_{j+1}(k)$ αν έχει διασπαστεί

Πως θα ελέγξουμε αν ο κάδος έχει διασπαστεί ή όχι

Αναζήτηση Εγγραφής

Δύο περιπτώσεις ο κάδος στον οποίο είναι (1) έχει ή (2) δεν έχει διασπαστεί

Κρατάμε μια μεταβλητή το πλήθος n των διασπάσεων

Έστω n ο αριθμός διασπάσεων και ότι αναζητούμε το k.

βρίσκεται στον κάδο **$h_0(k)$**

τότε αν **$n \leq h_0(k)$** ο κάδος δεν έχει διασπαστεί

ενώ αν **$n > h_0(k)$** ο κάδος έχει διασπαστεί και εφαρμόζουμε την $h_1(k)$

Αλγόριθμος Αναζήτησης

j : βήμα διάσπασης n : πλήθος διασπάσεων στο βήμα j

```
if (n = 0)
    then m := h_j(k);
else {
    m := h_j(k);
    if (m < n) then m := h_{j+1}(k)
}
```

σημαίνει ότι ο κάδος έχει διασπαστεί