ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

# Συστήματα Διαχείρισης και Ανάλυσης Δεδομένων (2022-2023)

Διονύσιος Ρηγάτος (P3200262)

## 1η Σειρά Ασκήσεων

### Άσκηση 1

1) Έχουμε τομέα 512 bytes, άρα:
   Μέγεθος Τομέα * Τομείς/Ιχνος * Κυλίνδρους * Επιφάνειες = 512bytes * 1000 * 10000 * 10 = **51.200.000.000bytes**

2) 60sec * ½ περιστροφή = Rsec * RPM ⇔ 60s * ½ περιστροφή = Rs * 10000 περιστροφές ⇔Rs = 30/10000 ⇔ **R = 0,003sec = 3msec**

3) 1 Block = Μέγεθος Block/Μέγεθος Τομέα = 4096/512 = **8 τομείς**
   Άρα έχουμε:
   (Τομείς ανά Μπλοκ/#Τομεις ανά Ίχνος) * (60/RPM)sec = 8/1000 * 60/10000sec = **0,000048sec = 0,048msec**

4) Το αρχείο καταλαμβάνει 1.000.000 blocks αφού κάθε εγγραφή είναι 1 block. Άρα, ο χρόνος ανάγνωσης 1.000.000 block:

   Time(1.000.000Blocks) = Avg Seek Time + Rotational Delay + 1.000.000*Block Transfer Time = 8msec + 3msec + 1.000.000*0,048msec = **48.011msec = 48,011sec**

5) Εφόσον πρόκειται για primary index, κάθε κλειδί είναι μοναδικό και συνεπώς δεν θα χρειαστεί να προσπελάσουμε πολλαπλούς κόμβους για την εύρεση ενός record. Υποθέτουμε επίσης ότι το φύλλο του ευρετηρίου δεν έχει ενσωματωμένα τα records και θα πρέπει να τα βρούμε με τον pointer. Θα πρέπει, για κάθε κόμβο του δέντρου έως το φύλλο, να βρούμε το block με τον κόμβο του δέντρου (avg seek time + rotational delay) και να το διαβάσουμε (0,048msec) αυτό θα γίνει 3 φορές αφού πρόκειται για B+ Tree με k=3. Τέλος, θα πρέπει για το record που μας ενδιαφέρει, να το εντοπίσουμε στον δίσκο και να το διαβάσουμε. Άρα η διαδικασία θα γίνει k=3 + 1 φορές, 1 φορά για κάθε n, άρα n*(3+1) φορές.

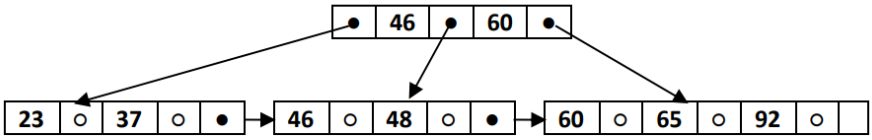
   **f(n) = n*(3+1)*(8ms + 3ms + 0,048ms) = N*44,192ms**

**Άσκηση 2**

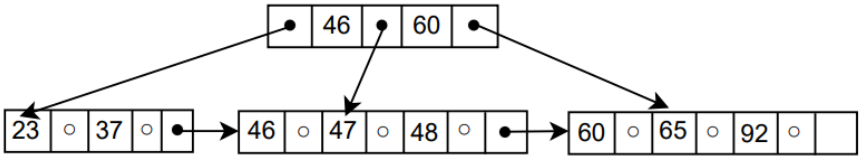
1) Αρχικά, γνωρίζουμε σίγουρα ότι ο optimizer δεν θα χρησιμοποιήσει το non-clustered index αφού είναι βασισμένο στην τιμή που θα κάνουμε update (Halloween problem, optimizer handles it). Επίσης, εφόσον όλες οι εγγραφές ενημερώνονται, το clustered index δεν προσφέρει κάτι στο να βρούμε κάποιες συγκεκριμένες εγγραφές ταχύτερα και συνεπώς δεν επηρεάζει κάτι.. Όμως η εντολή αυτή συνεπάγεται την ενημέρωση του non-clustered ευρετηρίου και συνεπώς **θα επιβραδυνθεί** η εκτέλεση λόγω της διαδικασίας αυτής.
2) Εφόσον πρόκειται για range query πάνω στο κλειδί του clustered ευρετηρίου και γνωρίζουμε ότι στο clustered ευρετήριο τα κλειδιά καθορίζουν τις θέσεις των εγγραφών, τότε η ύπαρξη clustered index στο empid **θα επιταχύνει** το update καθώς εγγραφές με διαδοχικά empids (π.χ. από το 1 έως το 100) θα είναι διαδοχικά γραμμένες στον δίσκο. Το non-clustered ευρετήριο δεν επηρεάζει κάτι.
3) Εδώ έχουμε ενημέρωση τιμών βάσει του departmentid, το οποίο δεν αποτελεί κλειδί κάποιου ευρετηρίου και συνεπώς η ύπαρξη των παραπάνω ευρετηρίων **δεν επηρεάζουν** την ταχύτητα εκτέλεσης της εντολής.

**Άσκηση 3**

α) Insertions

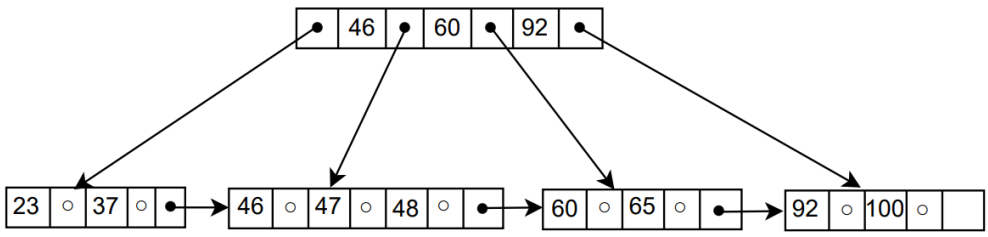



Insert 47



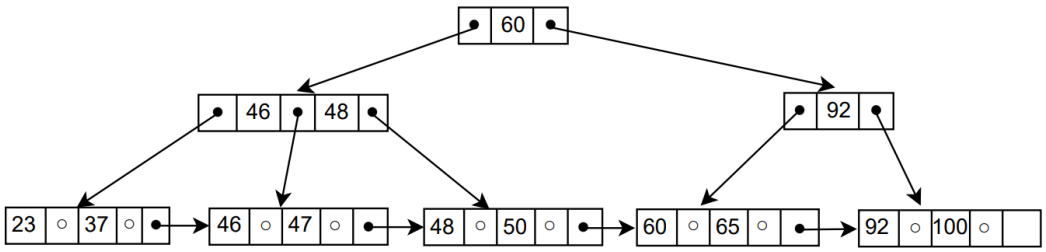

Το 47 είναι μεταξύ των 46 και 60 και συνεπώς θα πάει στο 2ο φύλλο. Επειδή το 2ο φύλλο έχει μόνο n=2 κλειδιά, προσθέτουμε το 47 στη σωστή θέση (μεταξύ του 46 και 48) και έχουμε ένα πλήρες φύλλο (n=3).

Insert 100



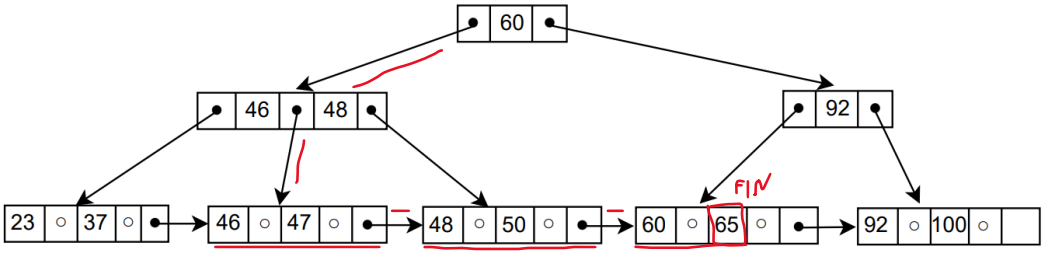
Το 100 είναι μετά το 60 και μετά το 92. Θα το προσθέσουμε στο δεξιότερο φύλλο και θα καταλήξουμε με φύλλο 4ου βαθμού ενώ το δέντρο μας είναι βαθμού n=3. Άρα θα ωθήσουμε το median κλειδί (92) του κόμβου προς τα πάνω (εδώ στην ρίζα) και θα σπάσουμε το φύλλο σε 2 φύλλα, ένα με τιμές <92 και ένα με τιμές >92. Τώρα η ρίζα είναι μεγίστου βαθμού.

Insert 50

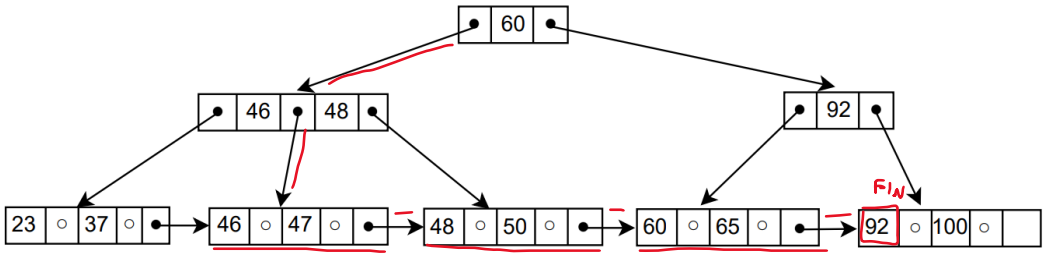



Το 50 βρίσκεται μεταξύ των 46 και 60, και μετά το 48. Επομένως το εισαγάγουμε στο 2ο φύλλο, το οποίο γίνεται 4ου βαθμού και συνεπώς πρέπει να το σπάσουμε. Ωθούμε το median στοιχείο (48) προς τα πάνω (ρίζα) και σπάμε το φύλλο σε 2 φύλλα, ένα με τιμές <48 και ένα με τιμές >48. Όμως εδώ η ρίζα γίνεται 4ου βαθμού κόμβος και συνεπώς πρέπει να σπάσει. Ωθούμε το median στοιχείο της ρίζας (60) προς τα πάνω και έχουμε μια καινούργια ρίζα-κόμβο με ένα μόνο στοιχείο (το 60). Επίσης, σπάμε τον κόμβο μας σε 2, έναν με στοιχεία <60 και έναν με στοιχεία >60 και καταλήγουμε με το παραπάνω τελικό δέντρο.

β) Προσπελάσεις αναζήτησης




i) Η διαδρομή έχει τονιστεί με κόκκινο. Θα προσπελαστούν 3 κόμβοι-φύλλα και θα σταματήσουμε αφού βρήκαμε το κλειδί 65 και γνωρίζουμε ότι είναι μοναδικό. Εν τέλει προσπελάσαμε 5 κόμβους κατά μήκος της διαδρομής.




ii) Η διαδρομή έχει τονιστεί με κόκκινο. Θα προσπελαστούν 4 κόμβοι-φύλλα και θα σταματήσουμε όταν βρούμε κλειδί >65 (το 92). Ο λόγος που δεν σταματήσαμε όταν

βρήκαμε το 1ο 65 είναι επειδή μπορεί να υπάρχουν κι άλλα 65 στον επόμενο κόμβο και συνεπώς δεν πρέπει να σταματήσουμε μέχρι να βρούμε μια τιμή μεγαλύτερη από αυτό. Εν τέλει προσπελάσαμε 6 κόμβους κατά μήκος της διαδρομής.

**Άσκηση 4**

Για την σχέση, χωρίς ευρετήριο, χρειαζόμαστε 1000/5 = 200 blocks αφού κάθε block χωράει 5 εγγραφές.

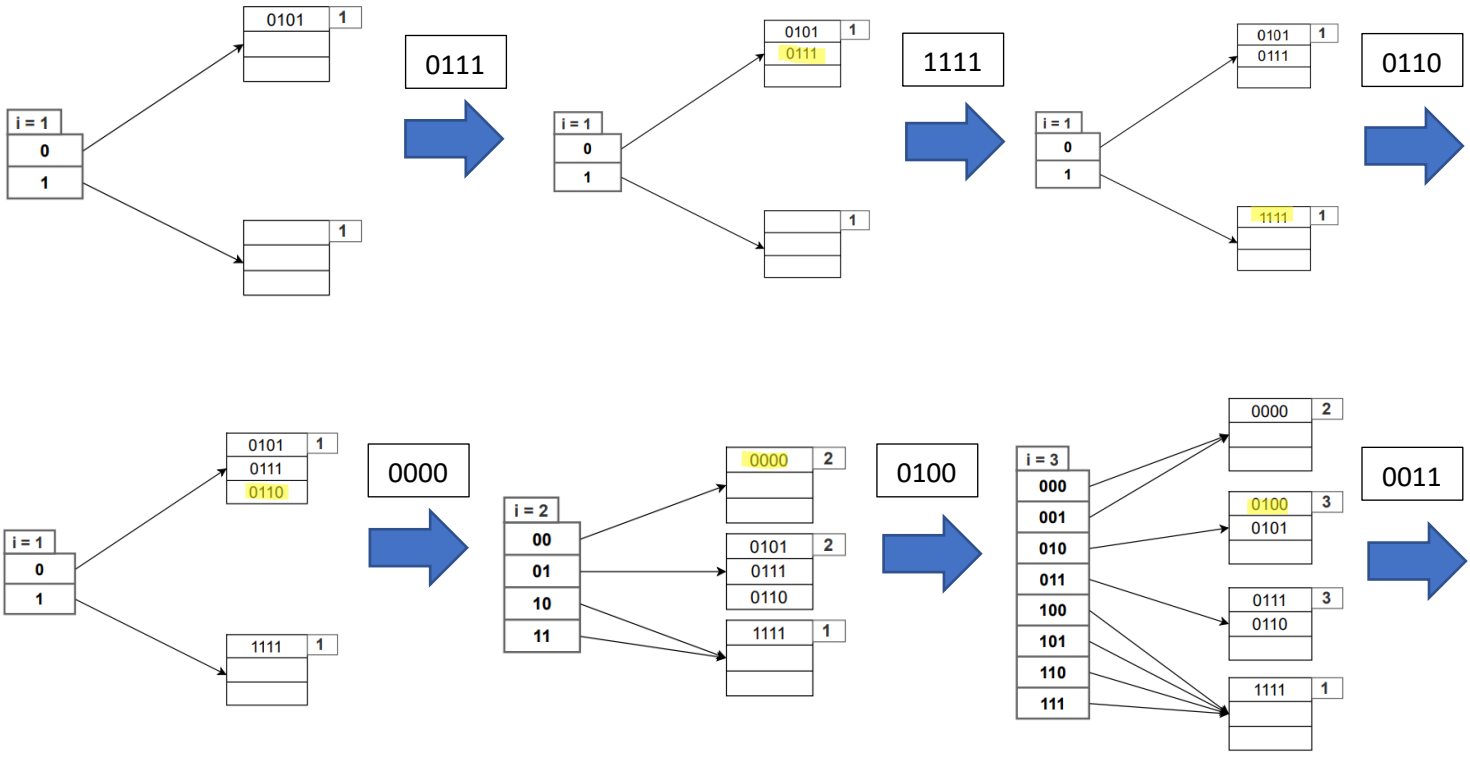
Επιπροσθέτως, κάθε κόμβος του δέντρου θα αντιστοιχεί σε ένα block και συνεπώς κάθε κόμβος θα έχει έως και 10 keys και 11 pointers. Κάθε φύλλο θα έχει 10 pointers προς ένα block δεδομένων με εγγραφές και 1 pointer προς το επόμενο φύλλο, άρα θα χρειαστούμε κατ' ελάχιστον 200/10 = 20 κόμβους (blocks) για τα φύλλα μας.

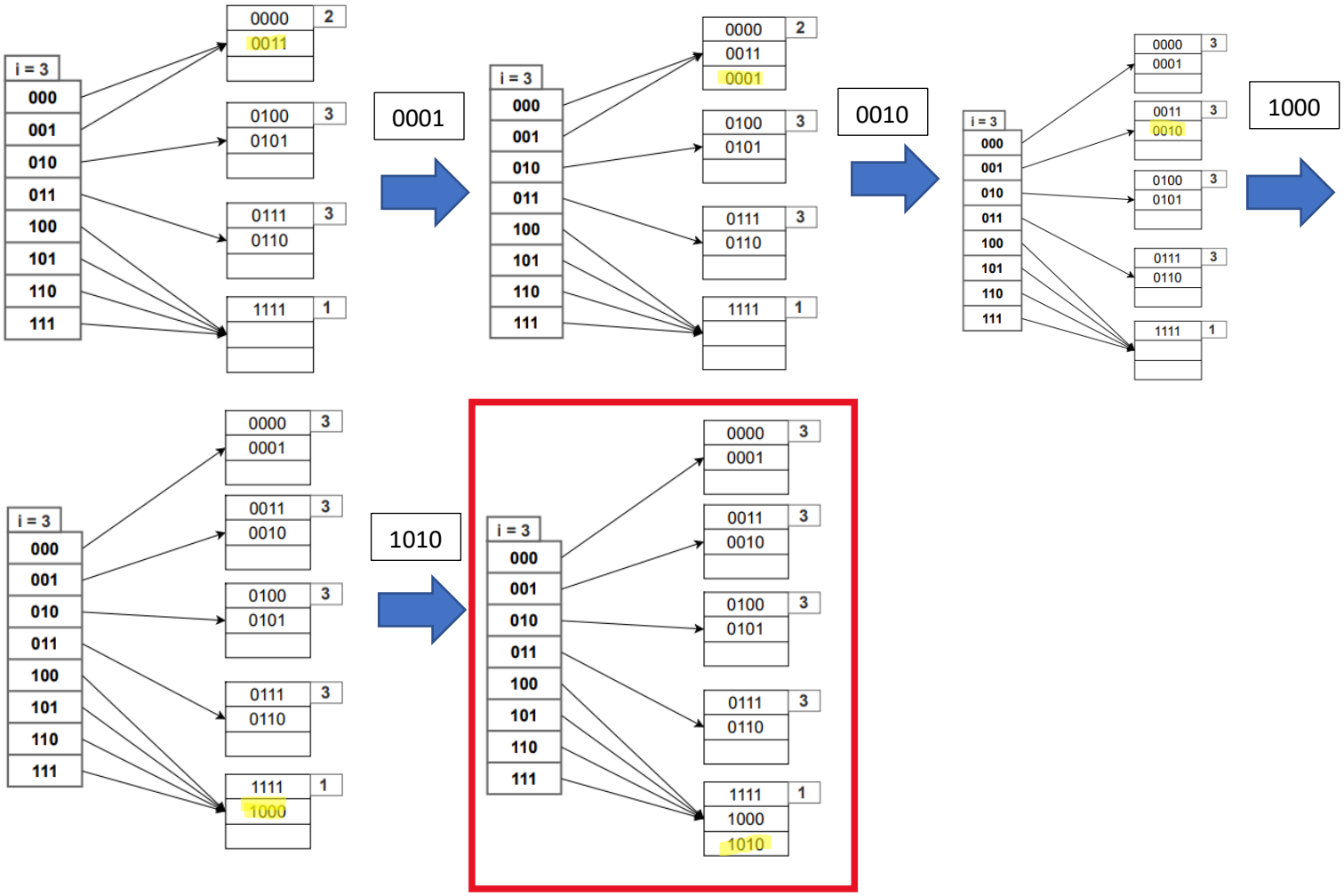
Όμως, πρέπει να λάβουμε υπόψιν και τους ενδιάμεσους κόμβους/pointers του δέντρου. Αφού έχουμε 20 φύλλα, χρειαζόμαστε τουλάχιστον 2 κόμβους που θα δείχνουν σε αυτά (π.χ. 10 pointers ο καθένας). Επίσης, χρειαζόμαστε μια ρίζα για το δέντρο μας, άρα άλλον 1 κόμβο με 2 pointers προς τους προαναφερόμενους 2 κόμβους που δείχνουν στα φύλλα. Άρα καταλήγουμε με ένα δέντρο ύψους 3.

Εν τέλει έχουμε: 200 + 20 + 2 + 1 = **223 blocks κατ' ελάχιστο**.

**Άσκηση 5**

α)






β) Για να διπλασιαστεί το ευρετήριο θα πρέπει να γεμίσουμε ένα από τα buckets βάθους i. Εδώ i = 3 και συνεπώς η εισαγωγή διπλότυπων τιμών στους κουβάδες βάθους i, (π.χ. εισαγωγή 0010 και 0011) θα οδηγούσε σε διάσπαση και συνεπώς αύξηση του i από 3 σε 4 (άρα 2^4 = 16 τιμές από 2^3 = 8 στο ευρετήριο). Άρα 2 τιμές αρκούν για τον διπλασιασμό του ευρετηρίου.

**Άσκηση 6**

Έστω ένας κάδος βάθους i με N εγγραφές. Έστω καινούργια εγγραφή η οποία αντιστοιχεί σε αυτόν τον κάδο, δηλαδή τα i bits των εγγραφών του κάδου και της καινούργιας εγγραφής είναι ίδια. Εφόσον η εγγραφή αυτή δεν χωράει στον κάδο, ο κάδος υπερχειλίζει και σπάει σε δύο καινούργιους κάδους, βάθους i+1. Αναζητούμε λοιπόν την πιθανότητα οι N+1 (προυπάρχουσες + καινούργια) εγγραφές να έχουν τα i+1 σημαντικότερα bits κοινά. Γνωρίζουμε ήδη πως οι εγγραφές έχουν τα i σημαντικότερα bits κοινά, άρα ψάχνουμε την πιθανότητα το i+1-στο σημαντικότερο bit να είναι ίδιο παντού. Κάθε bit έχει **½** πιθανότητα να πάρει μια τιμή, έχουμε **2** τιμές (0 ή 1) και θέλουμε να είναι ίδιο **N+1** φορές.

Άρα καταλήγουμε στην πιθανότητα: **2*(½)^(N+1) = (½)^N**