1η Σειρά Ασκήσεων

Διογένη Τσολάκου 3170164

Άσκηση 1

α) Υπολογίζοντας τα συνολικά byte των πεδίων κάθε εγγραφής βρίσκουμε το μέγεθος κάθε εγγραφής, το οποίο είναι 113 byte. Ξέρουμε ότι κάθε block έχει μέγεθος 512 byte.

Επομένως κάθε block χωράει 512/113 = 4.53. Επειδή όμως μπορούμε να έχουμε μόνο ακέραιο αριθμό εγγραφών σε ένα block συμπεραίνουμε ότι χωράνε 4 εγγραφές.

β) Από το προηγούμενο ερώτημα ξέρουμε ότι κάθε block χωράει 4 εγγραφές και έχουμε 20000 συνολικά εγγραφές. 20000/4 = 5000. Άρα χρειαζόμαστε 5000 blocks για να αποθηκεύσουμε όλες τις εγγραφές.

γ)

i) Για το rotational delay έχουμε :

(1/2)*(60/2400) = (1/2)*0.025 = 0.0125sec ή 12.5ms

Θεωρούμε ότι 1 sector ισούται με 1 block, άρα σε μία περιστροφή διαβάζουμε 20 blocks. Επομένως για να διαβάσουμε 1 block θέλουμε. Άρα το transfer time είναι :

(1/20)*(60/2400) = 0.05*0.025 = 0.00125sec ή 1.25ms

Για να διαβάσουμε μία εγγραφή θέλουμε χρόνο T :

T = seek time + rotational delay + transfer time = 30 + 12.5 + 1.25 = 43.75ms

Θεωρώντας ότι χρησιμοποιούμε γραμμική αναζήτηση τότε χρειάζεται να προσπελάσουμε 2500 blocks, το οποίο βγαίνει από τον τύπο

((n+2)*(n-1))/2*n, όπου n τα 5000 συνολικά blocks.

43.75*2500 = 109375ms ή 109.375sec ή ~1 λεπτό και 49 δευτερόλεπτα.

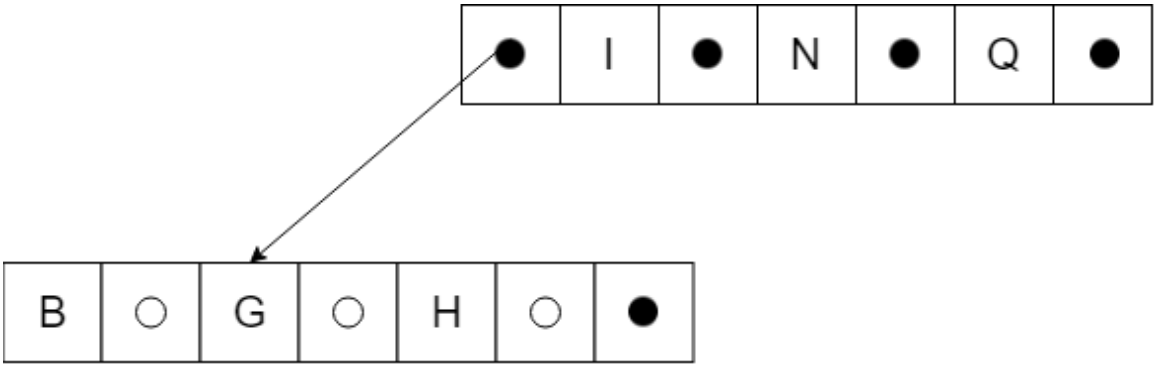
ii) Επειδή τα μπλοκ του αρχείου αποθηκεύονται διάσπαρτα θα πρέπει να ψάξουμε κάθε block για να βρούμε μια εγγραφή. Συνεπώς παίρνουμε τον χρόνο που βρήκαμε στο πρώτο σκέλος για την προσπέλαση ενός block και τον πολλαπλασιάζουμε με το 5000. 43.75*5000 = 218750ms ή 218.75sec ή ~3 λεπτά και 38 δευτερόλεπτα.

δ) Για την δυαδική αναζήτηση έχουμε χρόνο (logn)*block access time καθώς οι εγγραφές είναι ταξινομημένες και το log έχει ως βάση το 2. Το block access time είναι ο χρόνος που βρήκαμε παραπάνω στο ((γ)i). Το n στη συγκεκριμένη περίπτωση είναι τα συνολικά block που περιέχουν τις εγγραφές, τα οποία βρήκαμε πως είναι 5000 στο (β). Άρα έχουμε (log5000)*43.75ms = 12.287*43.75 = 537.55ms ή 0.537sec.

Άσκηση 2

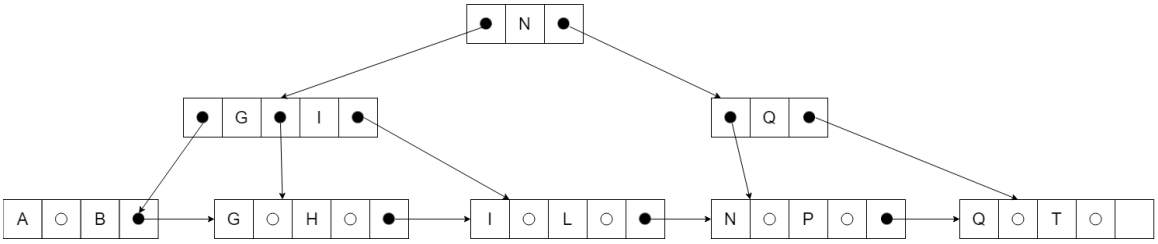
α)

Εισαγωγή G :



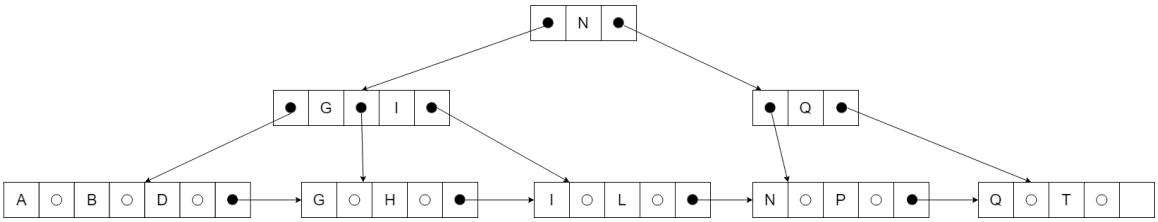
Αναζητούμε αν υπάρχει ήδη η τιμή G στο δέντρο. Το G είναι μικρότερο του I άρα θα πάμε αριστερά. Δεν το βρίσκουμε στο ήδη υπάρχον φύλλο και συνεπώς το εισάγουμε ανάμεσα στο B και H μιας και έχει χώρο.

Εισαγωγή A :



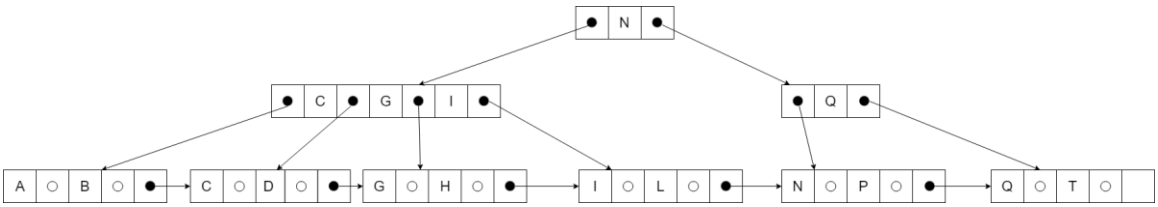
Αναζητούμε αν υπάρχει ήδη η τιμή A. Το A είναι μικρότερο του I άρα θα πάμε αριστερά. Το A είναι μικρότερο του B άρα δεν υπάρχει ήδη. Οπότε μπορούμε να το εισάγουμε. Όμως το παρόν φύλλο είναι γεμάτο άρα η εισαγωγή θα προκαλέσει overflow φύλλου. Για να αντιμετωπιστεί θα σπάσει το φύλλο και το G θα πάει στο αμέσως παραπάνω κόμβο ώστε να υπάρχουν τουλάχιστον 2 κλειδιά ανά φύλλο και θα προκληθεί κι εκεί overflow. Με τον ίδιο τρόπο θα σπάσουμε και τη ρίζα ανεβάζοντας το N και κάνοντας το νέα ρίζα.

Εισαγωγή D :



Αναζητούμε αν υπάρχει ήδη η τιμή D. Είναι μικρότερη του N άρα πάμε αριστερά του, είναι μικρότερη του G άρα συνεχίζουμε αριστερά και βρίσκουμε το αριστερότερο φύλλο, το D δεν υπάρχει και υπάρχει χώρος στο φύλλο άρα το εισάγουμε χωρίς κανένα πρόβλημα.

Εισαγωγή C :



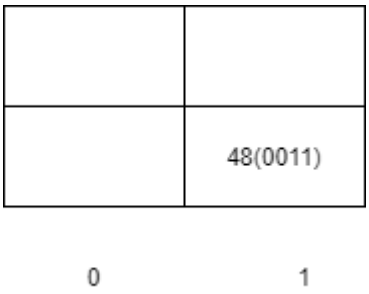
Αναζητούμε αν υπάρχει ήδη η τιμή C. Είναι μικρότερη του N άρα θα πάμε

αριστερά, είναι μικρότερη του G, συνεχίζουμε αριστερά, δεν υπάρχει όμως στο αριστερότερο φύλλο αλλά ούτε υπάρχει χώρος ώστε να εισαχθεί εκεί προκαλώντας overflow. Θα σπάσει το φύλλο σε A, B και C, D και το C θα ανέβει στο αμέσως παραπάνω κόμβο στον οποίο και χωράει.

β) Οι κόμβοι που θα προσπελαστούν είναι ο N, ο C, G, I , ο C, D , ο G, H , ο I, L , και ο N, P. Δηλαδή θα προσπελαστούν 6 κόμβοι για την ανάκτηση όλων των εγγραφών με κλειδί τουλάχιστον ίσο με C και το πολύ ίσο με P (key >= "C" AND key <="P").

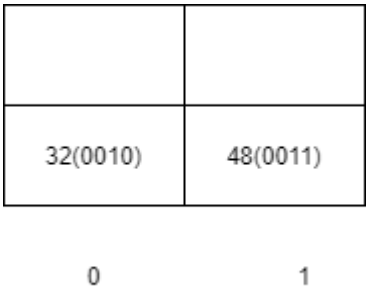
Άσκηση 3

α) 48 mod 15 = 3(0011)



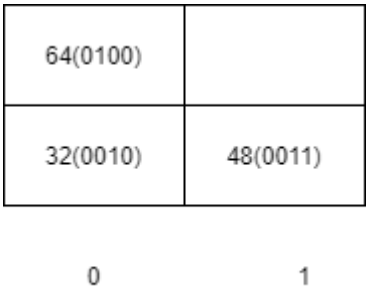
Utilization(1/4) = 25%

32 mod 15 = 2(0010)



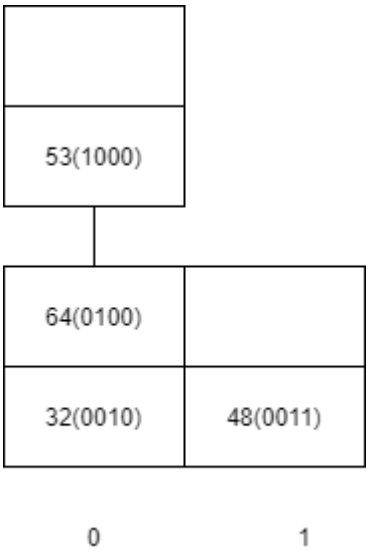
Utilization(2/4) = 50%

64 mod 15 = 4(0100)



Utilization(3/4) = 75%

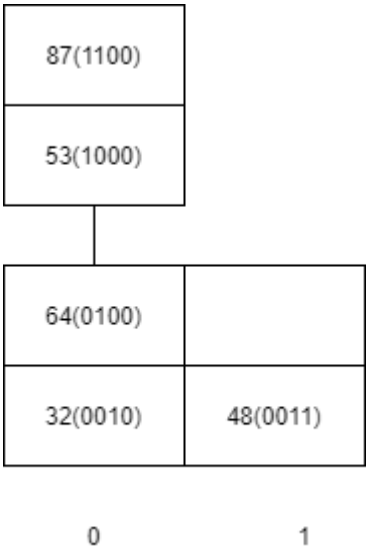
53 mod 15 = 8(1000)



Utilization(4/6) = 66.7%

Βάζουμε overflow chain καθώς το bucket 00 είναι γεμάτο αλλά το utilization δεν ξεπερνάει το 80%.

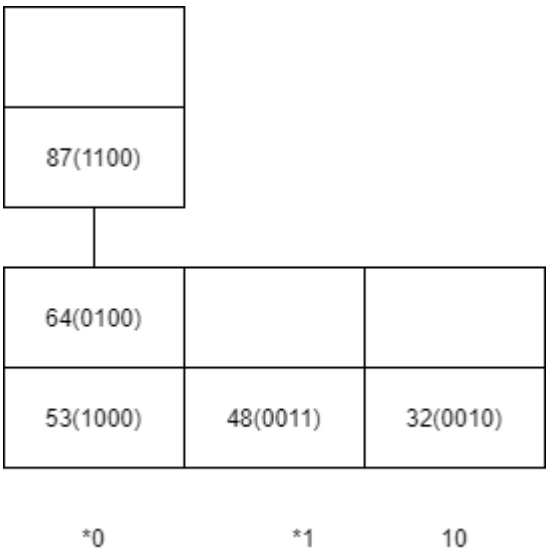
87 mod 15 = 12(1100)



Utilization(5/6) = 83%

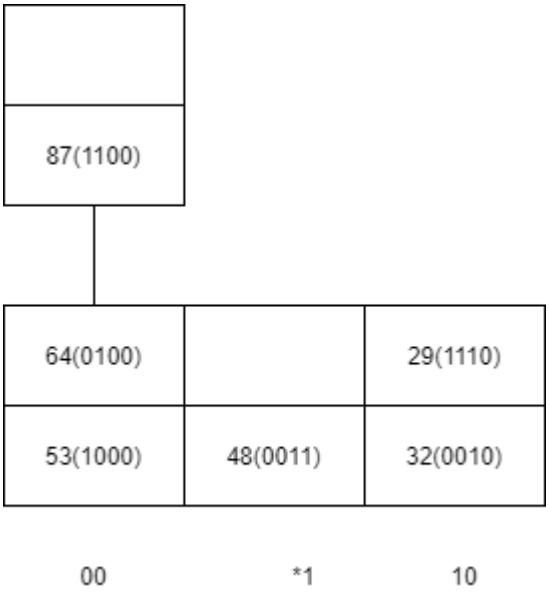
Το βάζουμε στο υπάρχον overflow chain καθώς το bucket 00 είναι γεμάτο αλλά το utilization δεν ξεπερνάει το 80% πριν την εισαγωγή. Όμως με την εισαγωγή του 87 ξεπερνάμε το 80%. Άρα αυξάνουμε το m κατά 1, m = 10, και επειδή το m ξεπερνά το όριο (2^i)-1 = 1, αυξάνουμε και το i κατά 1,

i = 2.



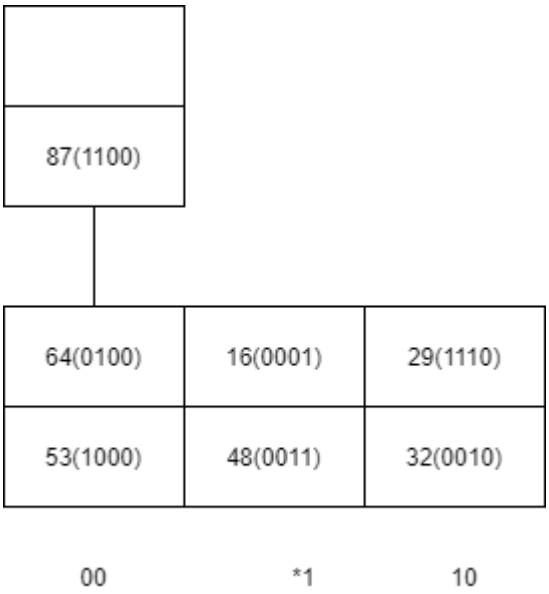
Utilization(5/8) = 62.5%

29 mod 15 = 14(1110)



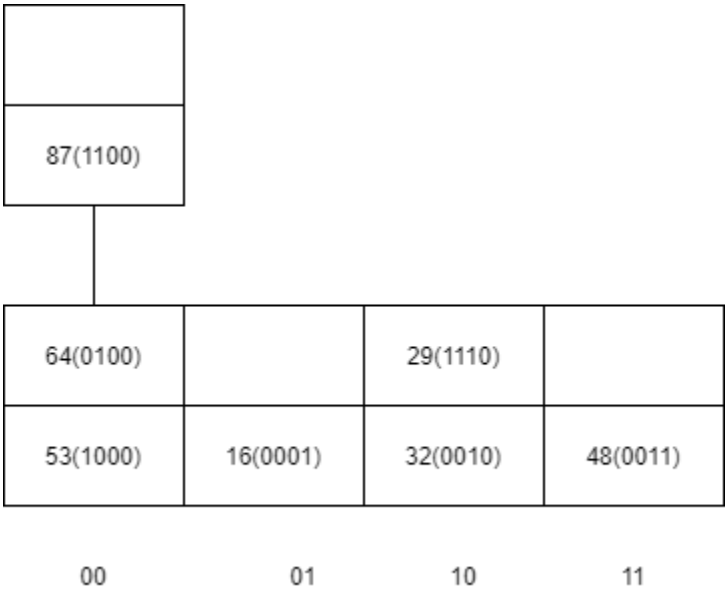
Utilization(6/8) = 75%

16 mod 15 = 1(0001)



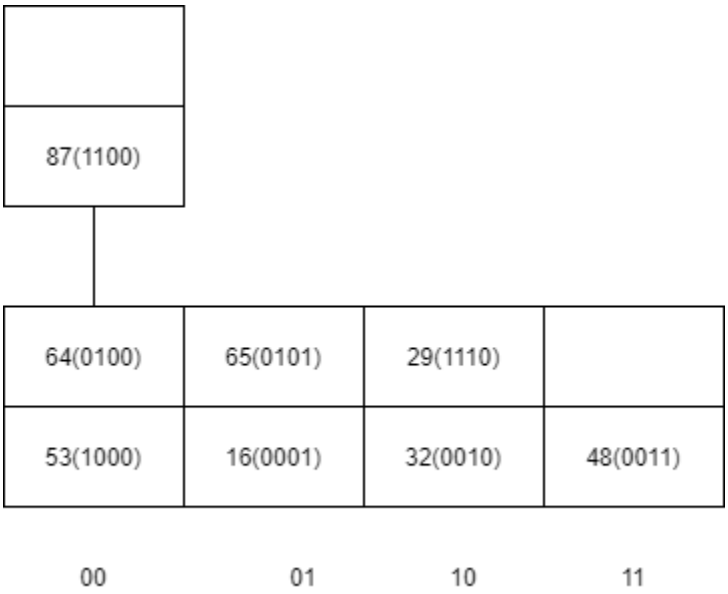
Utilization(7/8) = 87.5%

Το utilization ξεπερνά το 80% άρα πρέπει να προσθέσουμε ένα bucket και να αυξήσουμε το m κατά 1. Άρα το m = 11.



Utilization(7/10) = 70%

65 mod 15 = 5(0101)



Utilization(8/10) = 80%

Το utilization ισούται με 80% οπότε πρέπει να προσθέσουμε ένα bucket, να αυξήσουμε το m κατά 1, m = 100 και επειδή το m ξεπερνά το όριο (2^i)-1 = 3, να αυξήσουμε το i κατά 1, i = 3.

	65(0101)	29(1110)		87(1100)
53(1000)	16(0001)	32(0010)	48(0011)	64(0100)

000 *01 *10 *11 100

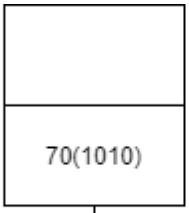
Utilization(8/10) = 80%. Το utilization ισούται πάλι με 80%, οπότε αυξάνουμε το m κατά 1, m = 101, δεν ξεπερνά όμως το όριο (2^i)-1 = 5 άρα δεν αυξάνουμε το i.

		29(1110)		87(1100)	
53(1000)	16(0001)	32(0010)	48(0011)	64(0100)	65(0101)

000 001 *10 *11 100 101

Utilization(8/12) = 66.7%

70 mod 15 = 10(1010)



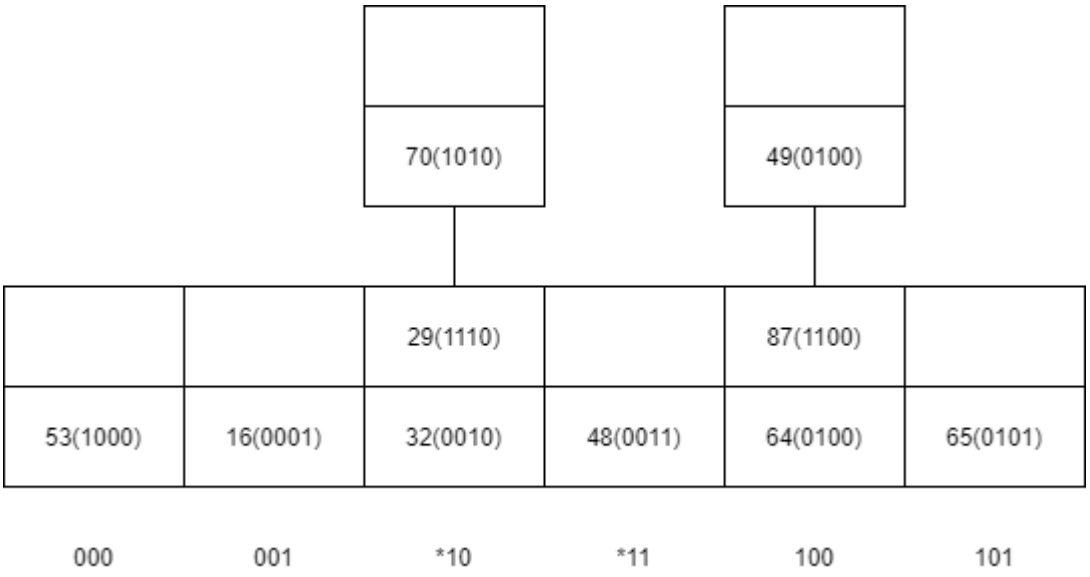
		29(1110)		87(1100)	
53(1000)	16(0001)	32(0010)	48(0011)	64(0100)	65(0101)

000 001 *10 *11 100 101

Βάζουμε overflow chain μιας και το bucket *10 δεν έχει χώρο αλλά δεν ξεπερνάμε το 80% του utilization.

Utilization(9/14) = 64.2%

49 mod 15 = 4(0100)



Με παρόμοιο τρόπο με την εισαγωγή του 70 βάζουμε overflow chain.
Utilization(10/16) = 62.5%

β)

i) Αφού έχουμε ευρετήριο το κόστος αναζήτησης είναι 1 I/O, όμως επειδή έχουμε δύο overflow pages έχουμε συνολικά 3 I/O.

ii) Αφού δεν έχουμε ευρετήριο και και δεν υπάρχει ταξινόμηση των εγγραφών δεν μπορούμε να κάνουμε δυαδική αναζήτηση και αρκούμαστε στην γραμμική αναζήτηση, συνεπώς πρέπει να ψάξουμε τον μισό πίνακα κατακερματισμού.