ΠΛΗ202 – ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ & ΑΡΧΕΙΩΝ
1η ΑΣΚΗΣΗ

ΠΕΤΡΟΥ ΔΗΜΗΤΡΙΟΣ - 2018030070

Text Editor

1. Σύντομη περιγραφή

Το πρόγραμμα καλείται στο terminal με την εντολή **java mypackage.MyEditor myfile.txt**. (Σε περίπτωση που δεν προσδιοριστεί κάποιο αρχείο γίνεται σχετική ερώτηση με τη δημιουργία κάποιου νέου. Το πρόγραμμα δεν μπορεί να λειτουργήσει χωρίς αρχείο στην έναρξη). Πραγματοποιείται ανάγνωση του αρχείου και δημιουργείται η διπλά συνδεδεμένη λίστα των γραμμών, η οποία επιτρέπει την υλοποίηση της λειτουργικότητας που απαιτείται. Στη συνέχεια ο χρήστης έχει την δυνατότητα να δώσει τις επιθυμητές εντολές. Για την διαδικασία της αναζήτησης όρου απαιτείται απλά να έχει δημιουργηθεί πρώτα το indexing αρχείο. (Το αρχείο δημιουργείται στο ίδιο directory με αυτό του παρεχόμενου αρχείου).

2. Διαδικασία λύσης – παρατηρήσεις

Το πρώτο μέρος δεν ήταν αρκετά περίπλοκο ως προς την συγγραφή κώδικα, ωστόσο απαιτούσε χρόνο για debugging και διαχείριση ακραίων & διακριτών περιπτώσεων.

- Διπλή σύνδεση κόμβων

Για την υλοποίηση της διπλής σύνδεσης στη λίστα, επιλέχθηκε ο κάθε κόμβος (**mypackage.FileLine**) να έχει αναφορά στον προηγούμενο και στον επόμενο. Ενώ, για την δημιουργία της λίστας χρησιμοποιήθηκε η δομή της Java: **java.util.LinkedList** . Η built-in λίστα της Java χρησιμοποιήθηκε ΜΟΝΟ για αποθήκευση στοιχείων και όχι για iteration.

- Ενέργειες πάνω σε γραμμές

Μια μια μεταβλήτη τύπου **FileLine** και ένας **Integer**, μέσα στην κλάση **mypackage.Commander**, «θυμούνται» την τρέχουσα γραμμή και τον αριθμό της αντίστοιχα και τα μεταβάλλουν ανάλογα με τις εντολές μετακίνησης που δίνονται από τον χρήστη (^,$,+,-). Για την προσθήκη νέας γραμμής πριν ή μετά την τρέχουσα (a,t) αφού ελέγξουμε τις ακραίες περιπτώσεις και διαχειριστούμε τα exceptions (πρώτη/τελευταία γραμμή), ανανεώνονται οι αναφορές των γειτονικών κόμβων και με βάση την είσοδο του χρήστη η γραμμή προστίθεται στη λίστα. Για διαγραφή της τρέχουσας γραμμής το σκεπτικό είναι πάνω κάτω το ίδιο, ενημερώνονται οι αναφορές των γειτονικών κόμβων, γίνεται κατάλληλη διαχείριση exceptions και η γραμμή αφαιρείται από την λίστα. Ως επιπλέον λειτουργικότητα έχει προστεθεί η δυνατότητα επεξεργασίας του κειμένου της τρέχουσας γραμμής (εντολή: e). Για την εκτύπωση της τρέχουσας γραμμής και του αριθμού της, χρησιμοποιήθηκαν τα member variables που είχαν δηλωθεί στον Commander.

- Μαζικές ενέργειες

Η καταμέτρηση χαρακτήρων/γραμμών και η εκτύπωση όλων των γραμμών υλοποιήθηκαν με μια απλή διάσχιση της λίστας και διαδοχική καταμέτρηση/εκτύπωση.

- Αποθήκευση αρχείου

Επειδή ο εντοπισμός αλλαγών στη λίστα των γραμμών θα ήταν αρκετά σπάταλος σε πόρους και χρόνο, επιλέχθηκε το αρχείο να πανωγράφεται με βάση όλη τη λίστα των γραμμών στον προορισμό.

- Indexing και δημιουργία αρχείου δεικτοδότησης

Διασχίζοντας τη λίστα των γραμμών, ανά στοιχείο, χωρίζουμε τις λέξεις με τη χρήση της **String.split()** και τις για κάθε μια δημιουργούμε ένα instance της κλάσης **mypackage.WordEntry** το οποίο περιέχει την λέξη και τον αριθμό της γραμμής από την οποία προέκυψε. Όλες οι καταχωρήσεις εισάγονται σε ενα Vector και ταξινομούνται μέσω της στατικής μεθόδου **Collections.sort(Vector<?>)** αφού πρώτα η κλάση WordEntry είχε υλοποιήσει το interface **Comparable** ώστε να παρέχει μέθοδο αλφαβητικής σύγκρισης. Ο κώδικας για τη δημιουργία του αρχείου δεικτοδότησης υλοποιήθηκε με βάση τα όσα συζητήθηκαν στα φροντιστήρια. Η λίστα των WordEntries κόβεται ανά τόσα στοιχεία όσα χωράνε σε μια σελίδα αρχείου, μετατρέπονται μέσω του **mypackage.EntryConverter** σε array από bytes και γράφονται διαδοχικά στο αρχείο μέσω της μεθόδου **mypackage.FilePageAccess.write()** μέχρι να φτάσουμε στο τέλος της λίστας. Και εδώ έχουμε αρκετές περιπτώσεις σφαλμάτων για να διαχειριστούμε, όπως τα εναπομείναντα στοιχεία να είναι λιγότερα από ότι απαιτούνται, η λίστα να είναι άδεια κ.α..

- Σειριακή αναζήτηση στο αρχείο

Έχοντας τον όρο με βάση τον οποίο θέλουμε να βρούμε καταχωρήσεις ξεκινάμε διαβάζοντας μία-μία τις σελίδες του αρχείου και μετατρέποντας τις απο byte σε λίστα. Ύστερα μετά την μετατροπή σε κάθε λίστα στοιχείων αναζητούμε σειριακά καταχωρήσεις που να μας ενδιαφέρουν. Όταν βρούμε μια τέτοια καταχώρηση την αντιγράφουμε σε μια άλλη λίστα. Η σειριακή αναζήτηση διακόπτεται όταν πλέον βρισκόμαστε σε σελίδα που περιέχει λέξεις αλφαβητικά μεγαλύτερες αυτής που αναζητούμε.

- Δυαδική αναζήτηση στο αρχείο

Έχοντας τον όρο με βάση τον οποίο θέλουμε να βρούμε καταχωρήσεις ξεκινάμε διαβάζοντας την μεσαία σελίδα του αρχείου. Παίρνουμε την πρώτη και την τελευταία λέξη της σελίδας και συνεχίζουμε με διάφορες συνθήκης. Ελέγχουμε αρχικά αν ο όρος μας είναι μεταξύ αυτών των δύο λέξεων, αν είναι τότε αναζητούμε σειριακά στην current σελίδα και η αναζήτηση τελειώνει. Αν ο όρος μας είναι μικρότερος, λεξικογραφικά από το πρώτο στοιχείο τότε η αναζήτηση συνεχίζεται στο αριστερό μισό του array από σελίδες. Αντίθετα αν όρος μας είναι μεγαλύτερος από το τελευταίο στοιχείο της σελίδας η αναζήτηση συνεχίζεται στο δεξί μισό του array από σελίδες. Η έκφραση «η αναζήτηση συνεχίζεται» σημαίνει ότι ο αλγόριθμος ξανατρέχει με την ίδια λογική σε πιο μικρό range σελίδων. Αφού βρεθούν καταχωρήσεις σε μία σελίδα ελέγχονται και οι γειτονικές της υπό όρους για τυχόν ξεχασμένες ή «κομμένες» καταχωρήσεις.

Αποτελέσματα πειράματος

Όνομα αρχείου	**Indexing File**		**Προσβάσεις στο δίσκο (Serial/Binary)**			
	Bytes	**Pages**	“Rectors”	“laboratories”	“Technical”	“Venetian”
testfile1.txt	15.616	121	95/8	67/8	113/6*	122/Error*
testfile_x2.txt	31.104	242	190/8	134/7	226/7*	243/Error*
testfile_x5.txt	77.568	605	474/10	334/11	564/6*	606/Error*
testfile_x10.txt	155.136	1211	948/10	668/10	1128/6*	1212/Error*
testfile_x1000.txt	15.504.128	121125	94800/6*	66800/9	112800/6*	121126/Error*

*Ο αλγόριθμος του binary search δεν λειτούργησε όπως αναμενόταν. Στα μεγάλα αρχεία χάνει καταχωρήσεις ενώ παρουσιάζει πρόβλημα και στην αναζήτηση καταχωρήσεων που τυχαίνει να βρίσκονται σε ακραίες σελίδες.

Τα στατιστικά των αναζητήσεων στα μικρά κυρίως αρχεία, στα οποία η δυαδική αναζήτηση λειτουργεί καλά, δείχνουν την σημαντική διαφορά που υπάρχει ανάμεσα στις προσβάσεις στο δίσκο αφού τα δύο αποτελέσματα διαφέρουν κατά μέσο όρο περί τα 84 disk accesses λιγότερα στο binary search.

Η ανεπιτυχής αναζήτηση έδωσε τα παρακάτω αποτελέσματα ανά αρχείο (μέση τιμή, στρογγυλοποιημένο):

Όνομα αρχείου	Προσβάσεις στο δίσκο (Serial/Binary)
testfile1.txt	55/4
testfile_x2.txt	103/7
testfile_x5.txt	456/9
testfile_x10.txt	640/14*
testfile_x1000.txt	30400/12*

*Τα αποτελέσματα της δυαδικής αναζήτησης για το μεγάλο αρχείο δεν είναι ακριβή.