Δομές Δεδομένων και Αρχείων- 1η άσκηση

Νίκου Γεώργιος-Νεκτάριος AM:2016030125

Μέρος 1ο:
Αρχικά για την υλοποίηση της διπλά συνδεδεμένης λίστας δημιούργησα την κλάση Node την οποία χρησιμοποίησα στην κλάση DoublyLinkedList για να δημιουργήσω την λίστα.

Στην Main Class βρίσκονται οι βασικές λειτουργίες(κονσόλα κλπ.), οι static μεταβλητές(page size, min word size κλπ.) και καλούνται οι υπόλοιπες κλάσεις για την κάθε ενέργεια. Πιο συγκεκριμένα η main ελέγχει αρχικά αν δίνεται κάποιο αρχείο σαν είσοδος με την εκτέλεση του προγράμματος για να διαβαστεί, αλλιώς ζητάει όνομα για δημιουργία νέου αρχείου.
Έπειτα μέσω BufferedReader ανοίγει το αρχείο και διαβάζει κάθε γραμμή του. Ελέγχει κάθε γραμμή αν ξεπερνάει το όριο του line size, κόβει σε αυτό το όριο όσες το ξεπερνάνε και καλεί την FileRead της DoublyLinkedList ώστε να περαστεί κάθε γραμμή στην λίστα. Τέλος εμφανίζει ένα μήνυμα ώστε να διαλέξει ο χρήστης λειτουργία και με ένα switch statement καλεί την κατάλληλη συνάρτηση/κλάση ανάλογα την επιλογή του χρήστη.

Η κλάση DoublyLinkedList έχει τις βασικές λειτουργίες της διπλά συνδεδεμένης λίστας καθώς και όλες τις συναρτήσεις του πρώτου μέρους. Ειδικότερα για τις μεθόδους:

- size, charCount: επιστρέφουν πλήθος γραμμών και χαρακτήρων για την αντίστοιχη εντολή
- FileRead: η μέθοδος προσθέτει κάθε γραμμή του αρχείου κειμένου στην λίστα
- printList: εκτυπώνει τα περιεχόμενα της λίστας και δέχεται το toggle σαν είσοδο για την εμφάνιση του αριθμού της γραμμής. Η toggle ελέγχεται στην Main Class για να αλλάζει τιμή με το πάτημα της εντολής ‘n’.
- saveFile: διατρέχει την λίστα και την αποθηκεύει στο αρχείο με τις νέες αλλαγές.
- FileIndex: δέχεται ως είσοδο ελάχιστο και μέγιστο μήκος λέξης. Κάνει split τα strings της κάθε γραμμής και αφού απορρίψει τις μικρότερες λέξεις ελέγχει για τις μεγάλες. Έπειτα κόβει τις τελευταίες στο όριο και τις εισάγει μαζί με τον αριθμό γραμμής σε ArrayList τύπου IndexTable όπως και τις υπόλοιπες λέξεις. Έπειτα τις ταξινομεί μέσω του interface Collections και επιστρέφει το ArrayList
- addNewLineAfter, addNewLineBefore, deleteLine: προσθέτουν ή διαγράφουν γραμμή σε σχέση με την τωρινή θέση του δείκτη
- printLine, printCurrentLineNumber: τυπώνουν την τωρινή γραμμή και τον αριθμό της
- editList: έλεγχος του pointer για την τωρινή γραμμή -head, up, down, tail

Μέρος 2ο:
Για την δημιουργία αρχείου δεικτοδότησης δημιουργώ ένα νέο αρχείο με κατάληξη .ndx στην Main. Καλούμε την FileIndex και παίρνουμε ένα ταξινομημένο πίνακα με λέξεις και γραμμές. Διατρέχοντας τον πίνακα παίρνουμε κάθε φορά page_elements στοιχεία, δηλαδή όσα χωράει κάθε σελίδα. Ύστερα μετατρέπει τα στοιχεία αυτά σε bytes(getPageBytes) και τα γράφει(writeFile) στο .ndx αρχείο στην κατάλληλη θέση. Τέλος παίρνει τα bytes της τελευταίας σελίδας ελέγχει αν δεν είναι άδεια, την γράφει και εκτυπώνεται ο αριθμός προσβάσεων.
Για την εκτύπωση του αρχείου δεικτών καλείται η fileIndex και επιστρέφει ένα arrayList. Με for loop διατρέχει τα στοιχεία της και τα εκτυπώνει όπως και τον αριθμό των συνολικών λέξεων.
Για τις serial και binary search στην Main ο χρήστης δίνει την λέξη του και ύστερα καλούνται οι κατάλληλες συναρτήσεις.

Κλάση IndexTable: χρησιμοποιείται για την δημιουργία πίνακα δεικτοδότησης

- compareTo: ταξινομεί τον πίνακα δεικτοδότησης βάσει της λέξης

Κλάση FilePageAccess: εγγραφή/ανάγνωση μιας σελίδας από/στο αρχείο

- writeFile: γράφει τα bytes της σελίδας στην σωστή θέση με την seek στο .ndx αρχείο μέσω RandomAccessFile
- readFile: διαβάζει page_size bytes από το αρχείο και τα μετατρέπει σε indexTable δεδομένα μέσω της συνάρτησης toIndex(κλάση Index Converter) και επιστρέφει arrayList από indexTable

Κλάση IndexConverter: μετατροπή μεταξύ indexTable και bytes

- getPageBytes: μετατρέπει page_elements στοιχεία του IndexTable για να γεμίσει μια σελίδα σε bytes και τα επιστρέφει
- toIndex: μετατρέπει μια σελίδα από bytes σε arrayList από IndexTable και την επιστρέφει

Κλάση Search: ελέγχει σειριακά τα στοιχεία μιας σελίδας αν είναι ίσα με την λέξη που αναζητείται και επιστρέφει τον αριθμό των γραμμών

Κλάση SerialSearch: για κάθε μια από τις σελίδες παίρνει έναν πίνακα IndexTable(καλώντας την readFile) και καλεί την search η οποία επιστρέφει τις τιμές από τις γραμμές, αν υπάρχει η λέξη. Τέλος εκτυπώνει το αποτέλεσμα και τον αριθμό των προσβάσεων.

Κλάση BinarySearch: σαν αρχική σελίδα αναζήτησης(μεσαία σελίδα) παίρνει την μισή από το σύνολο των λέξεων. Έπειτα σε ένα while loop ανάλογα την τιμή του ret γίνεται η αντίστοιχη ενέργεια πχ. πρόσβαση άνω ή κάτω μισού και έξοδος. Μέσα στο loop ελέγχω για διάφορες περιπτώσεις για να γίνει η πρόσβαση στην σωστή σελίδα.
Αρχικά συγκρίνουμε για σελίδες γεμάτες με την λέξη που αναζητήθηκε όπου την κρατάμε και αποθηκεύουμε τον αριθμό των γραμμών. Ύστερα αναζητάμε προς τα πίσω και μόλις δεν την βρίσκουμε άλλο αναζητούμε από την σελίδα που κρατήσαμε και έπειτα.
Επιπλέον ελέγχουμε αν τη τελευταία σελίδα που κάναμε πρόσβαση πρόκειται να την επισκεφθούμε ξανά,σε περίπτωση που η λέξη βρίσκεται ανάμεσα στην τελευταία λέξη της μιας σελίδας και την πρώτη της επόμενης(για να μην γίνει infinite loop). Τέλος ελέγχεται να μην κάνουμε access σελίδα πριν/μετά την πρώτη/τελευταία σελίδα. Για την δυαδική αναζήτηση κάθε φορά αλλάζουμε το first ή last page ανάλογα την πρόσβαση άνω ή κάτω μισού και παίρνουμε τον μέσο όρο τους. Πχ για να κάνουμε πρόσβαση στο κάτω μισό θέτω last page = middle _page-1 και παίρνω τον μέσο όρο μεταξύ first page και last page. Αντίστοιχα για το πάνω μισό θέτω first page = middle_page + 1 και παίρνω τον μέσο όρο. Επίσης στην περίπτωση που η λέξη που αναζητείται βρίσκεται πχ στο τελευταίο στοιχείο της σελίδας γίνεται πρόσβαση και στις επόμενες σελίδες διαδοχικά για να βρεθούν όλες τις τιμές. Τέλος εκτυπώνεται το αποτέλεσμα και ο αριθμός προσβάσεων.

Το πρόγραμμα τρέχει κανονικά και περνάει χωρίς λάθη από τον compiler. Η σειριακή αναζήτηση γίνεται εξαντλητικά και δεν σταματάει μόλις ξεπεραστεί η τιμή που αναζητούμε όταν δεν υπάρχει.

<u>Πείραμα</u>

testfile1.txt: αρχείο μέγεθος 17 kB, 130 σελίδες
1.α)Σειριακή αναζήτηση: Rectors: 39 προσβάσεις, laboratories: 93 προσβάσεις, Technical: 47 προσβάσεις, Venetian: 54 προσβάσεις
β)Δυαδική αναζήτηση: Rectors: 7 προσβάσεις, laboratories: 6 προσβάσεις, Technical: 8 προσβάσεις, Venetian: 6 προσβάσεις

2.α)Σειριακή αναζήτηση: 130 προσβάσεις
β)Δυαδική αναζήτηση: 6 προσβάσεις

---

testfile_x2.txt: αρχείο μέγεθος 33 kB, 259 σελίδες
1.α)Σειριακή αναζήτηση: Rectors: 77 ,laboratories: 185, Technical: 92, Venetian: 107
β)Δυαδική αναζήτηση: Rectors: 8, laboratories: 8, Technical: 11,Venetian: 7

2.α)Σειριακή αναζήτηση: 259 προσβάσεις
β)Δυαδική αναζήτηση: 7 προσβάσεις

---

testfile_x5.txt: αρχείο μέγεθος 81 kB, 646 σελίδες
1.α)Σειριακή αναζήτηση:Rectors: 189, laboratories: 459, Technical:228, Venetian: 265
β)Δυαδική αναζήτηση: Rectors: 11, laboratories: 11, Technical: 22,Venetian: 10

2.α)Σειριακή αναζήτηση: 646 προσβάσεις
β)Δυαδική αναζήτηση: 9,83 προσβάσεις

---

testfile_x10.txt: αρχείο μέγεθος 162 kB, 1292 σελίδες
1.α)Σειριακή αναζήτηση: Rectors: 378, laboratories: 918, Technical: 456, Venetian: 530
β)Δυαδική αναζήτηση: Rectors: 12. laboratories: 11, Technical: 38, Venetian: 14

2.α)Σειριακή αναζήτηση: 1292 προσβάσεις
β)Δυαδική αναζήτηση: 10,16 προσβάσεις

---

testfile_x1000.txt: αρχείο μέγεθος 15,7 MB, 129200 σελίδες
1.α)Σειριακή αναζήτηση: Rectors: 37602, laboratories: 91602, Technical: 45402, Venetian: 52802
β)Δυαδική αναζήτηση: Rectors: 210, laboratories: 210, Technical: 3206, Venetian: 806

2.α)Σειριακή αναζήτηση: 129200 προσβάσεις
β)Δυαδική αναζήτηση: 17,43 προσβάσεις

Η δυαδική μέθοδος αναζήτησης φαίνεται ξεκάθαρα πιο αποδοτική σε σχέση με την σειριακή. Όσο αυξάνεται το μέγεθος των αρχείων που αναζητούνται οι προσβάσεις της σειριακής αναζήτησης αυξάνονται γραμμικά ενώ της δυαδικής ακολουθούν λογαριθμική αύξηση.
Για ανεπιτυχή σειριακή αναζήτηση ο αριθμός των προσβάσεων είναι ίσος με τον αριθμό των σελίδων του αρχείου καθώς γίνεται εξαντλητική σειριακή αναζήτηση. Αντίθετα για ανεπιτυχή δυαδική αναζήτηση ο αριθμός των προσβάσεων παραμένει παρόμοιος με τον αντίστοιχο των επιτυχών αναζητήσεων και αυξάνεται λογαριθμικά.

Πηγές

https://www.java2novice.com/data-structures-in-java/linked-list/doubly-linked-list : συμβουλεύτηκα για την δημιουργία του node και της διπλά συνδεδεμένης λίστας και έκανα αρκετές τροποποιήσεις έπειτα
https://www.baeldung.com/java-binary-search : πήρα από το Iterative Impl παράδειγμα την ιδέα για την υλοποίηση της δυαδικής αναζήτησης θέτοντας first page = mid+1 ή last page = mid – 1
https://stackoverflow.com/ + αναζήτηση google : για βοήθεια στο γράψιμο της Java και σε απορίες