Δομες Δεδομενων 2022-2023

Εργασια 2

Κουσιδης Χρηστος - Κωνσταντακης Αντωνιος

p3200086 - p3200088

Μερος Α)

ΑΤΔ Ουρας Προτεραιοτητας: Για την ουρα προτεραιοτητας χρησιμοποιησαμε το interface του φροντιστηριου(με generics) καθως και την υλοποιηση HeapPriorityQueue, κανοντας αλλαγες όπως αλλαζοντας την μεθοδο add σε insert και προσθετοντας μια μεθοδο getSize.

Disk.java: Εκτος από τις βασικες μεθοδους εισαγωγης/εξαγωγης στοιχειων και μεθοδου εμφανισης του ελευθερου χωρου του δισκου, προσθεσαμε μεθοδο εκτυπωσης δισκου η οποια τυπωνει το id του δισκου, τον ελευθερο του χωρο καθως και τους φακελους από τους οποιους αποτελειται.

Μερος Β)

Για την υλοποιηση του αλγοριθμου χρησιμοποιησαμε μια ουρα προτεραιοτητας DiskList τυπου MaxPQ και μια λιστα folderList τυπου List (Λιστα του φροντιστηριου με generics, με προσθηκη της μεθοδου getSize())

Με την μεθοδο fetchdata που δεχεται το ονομα του txt αρχειου και την λιστα folderList, γραψαμε στην λιστα το μεγεθος κάθε φακελου που μας δινεται σε κάθε γραμμη του txt, χρησιμοποιωντας την κλαση Scanner της java.

Με την μεθοδο fitIntoDisks(που επιστρεφει ένα float με το αθροισμα των μεγεθων των φακελων), τοποθετουμε κάθε φακελο από την λιστα folderList μεσα σε δισκους. Για τον πρωτο φακελο που πρεπει να τοποθετηθει, δημιουργουμε τον πρωτο δισκο,

βαζουμε μεσα το αρχειο και υστερα τοποθετουμε τον δισκο στην

Ουρα Προτεραιοτητας DiskList. Για τα υπολοιπα αρχεια, γινεται ελεγχος αν χωραει στον δισκο με τον περισσοτερο ελευθερο χωρο

(δηλαδη αν το μεγεθος του φακελου είναι μικροτερο ή ισο του diskCheck.peek().getFreeSpace(). Αν είναι, τοτε αφαιρουμε τον δισκο από την Ουρα, προσθετουμε τον φακελο και τον ξαναβαζουμε στην ουρα.

Το diskCheck εχει την τιμη του δισκου της ουρας με την μεγαλυτερη προτεραιοτητα ώστε να μπορουμε να κανουμε τους απαραιτητους ελεγχους.

Κατά την διαδικασια της fitIntoDisks, προστιθεται σε ένα αθροισμα Sum το μεγεθος του φακελου που τοποθετειται σε δισκο, και το τελικο αθροισμα αυτό επιστρεφεται από την μεθοδο. Μετα από αυτή την διαδικασια, οι δισκοι προστιθενται σε ένα Array που περιεχει στοιχεια τυπου Disk και μεσω της printOutput εμφανιζεται το καταλληλο τελικο μηνυμα.

Μερος Γ)

Χρησιμοποιησαμε αλγοριθμο ταξινομησης Quicksort, ο οποιος δεχεται ένα Array τυπου Int καθως και τα ορια του Array.

Ο Αλγοριθμος intquicksort λειτουργει αναδρομικα, χρησιμοποιωντας την intpartition, η οποια ξεκινωντας από το πρωτο στοιχειο, το τοποθετει σε θεση οπου το αριστερο του στοιχειο είναι πιο μικρο, και το δεξι του στοιχειο είναι πιο μεγαλο από το ιδιο(με την χρηση της intswap η οποια αντιμεταθετει τα στοιχεια στις δυο θεσεις που δεχεται). Μετα καλει τον εαυτο του στους επιμερους πινακες με ορια Low – p-1 και p+1 – high αντιστοιχα. Στο τελικο αποτελεσμα, το πρωτο στοιχειο του Array ειναι το μεγιστο του στοιχειο.Ο αλγοριθμος αυτος χρησιμοποιειται σε ένα array που περιεχει τα μεγεθη των φακελων της λιστας folderList την οποια αδειαζουμε πριν την ταξινομηση και ξαναγεμιζουμε μετα την ταξινομηση.

Η κλαση Sort λειτουργει με τροπο Πανομοιοτυπο της Greedy με την μονη διαφορα να είναι η χρηση του αλγοριθμου ταξινομησης. Για αυτό τον λογο ολες οι μεθοδοι που χρησιμοποιει είναι οι public μεθοδοι της Greedy. (π.χ. Greedy.fitIntoDisks(), Greedy.fetchdata())

Μερος Δ)

Για το μερος Δ δημιουργησαμε τις μεθοδους:

RandomFill: Δεχεται το πληθος των γραμμων που θα περιεχει το αρχειο, καθως και 2 Arrays τυπου int. Με την χρηση της κλασης Random της java δημιουργησαμε επαναληπτικα ακεραιους από

0 εως 1.000.000 και γεμισαμε τα 2 Arrays με τα ιδια N στοιχεια.

FileWrite: Δεχεται το ονομα του αλγοριθμου που χρησιμοποιειται, ένα array που περιεχει τα στοιχεια που θα μπουν σε κάθε γραμμη του αρχειου, τον αριθμο των στοιχειων/γραμμων, καθως και το νουμερο του αρχειου που δημιουργειται(αναλογα με το ποιο δοκιμαστικο αρχειο είναι, από 1-10)

Η μεθοδος συνθετει το ονομα του txt αρχειου που θα δημιουργηθει,το δημιουργει, και με την χρηση ενός BufferedWriter γραφει σε κάθε γραμμη του το αντιστοιχο στοιχειο του Array. Τελος επιστρεφει το ονομα του αρχειου που δημιουργησε.

StartProcessing: Δεχεται το πληθως των γραμμων που θα εχει το κάθε αρχειο καθως και το νουμερο του αρχειου αυτου(1-10)

Καλει την randomFill και επιστρεφει τους γεματους πινακες,

Ταξινομει τον πινακα του αλγοριθμου 2 με την Sort.intquicksort, και υστερα γραφει τα καταλληλα αρχεια για τους δυο αλγοριθμους, επιστρεφοντας τα ονοματα των αρχειων.

Για το κάθε ονομα καλει την Greedy.fetchdata ωστε να γεμισει τις καταλληλες ουρες/λιστες και να βαλει σε δισκους τους φακελους που λαβαμε από τα txt με την χρηση της Greedy.fitIntoDisks.

Τελος επιστρεφει ένα array int[2] με το πληθος των δισκων που χρησιμοποιησε ο κάθε αλγοριθμος σε κάθε κελι του.

Main: Δημιουργουμε 2 Array int[2], Τα Sum και Temp.

Με την βοηθεια του Temp γεμιζουμε επαναληπτικα το Sum με το συνολικο αθροισμα των δισκων για κάθε N (100, 500, 1000) από τα 10 αρχεια που δημιουργησαμε για κάθε τιμη του N και υπολογιζουμε τον μεσο ορο δισκων που χρησιμοποιησε ο κάθε αλγοριθμος. Τελος τυπωνει για κάθε N το καταλληλο μηνυμα.

Συμπερασμα: Από το πρωτο κιολλας πειραμα χρησης των δυο αλγοριθμων, προκυπτει ότι η χρηση του Αλγοριθμου 2 (Sort.java) χρησιμοποιει κατά μεσο ορο λιγοτερους δισκους για την αποθηκευση των ιδιων φακελων από ότι ο Αλγοριθμος 1 (Greedy.java). Μετα από πολλες χρησεις το πειραμα παραγει σταθερα αποτελεσματα και συνεπως αποδεικνυει ότι ο Αλγοριθμος 2 είναι πιο αποδοτικος για την αποθηκευση μεγαλου πληθους φακελων σε δισκους.