Σταθόπουλος Γεώργιος p3170152

Ντούλης Παντελεήμων p3170124

ΜΕΡΟΣ Α

Στο πρώτο μέρος υλοποιήσαμε μία ουρά προτεραιότητας. Επειδή θέλαμε η δομή να είναι αποδοτική η υλοποίηση έγινε με τη χρήση σωρού (εισαγωγή και εξαγωγή μεγίστου σε χρόνο O(logN)). Η υλοποίηση βασίστηκε στην ουρά του εργαστηρίου. Αποτελείται από 5 μεταβλητές: έναν πίνακα τύπου Disk ο οποίος είναι ένας πίνακας με τις ιδιότητες του σωρού, έναν ακέραιο size για το μέγεθος της ουράς, μία μεταβλητή τύπου DiskComparator (υλοποιεί την μέθοδο compare, η οποία συγκρίνει δύο αντικείμενα Disk ανάλογα με τον ελεύθερο χώρο που διαθέτουν) και δύο ακεραίους για την αρχική χωρητικότητα της ουράς και το βήμα επέκτασης σε περίπτωση που γεμίσει. Οι κύριες μέθοδοι της ουράς είναι: η add η οποία προσθέτει ένα στοιχείο (αν η ουρά είναι γεμάτη αυξάνει την χωρητικότητα) και κάνει ανάδυση του στοιχείου για να αποκατασταθεί η ιδιότητα του σωρού, η peek η οποία εμφανίζει το μέγιστο στοιχείο και η getMax() η οποία επιστρέφει και διαγράφει το μέγιστο και φροντίζει να επαναφέρει την ιδιότητα του σωρού με κατάδυση. Για την υλοποίηση των μεθόδων χρησιμοποιήσαμε τις βοηθητικές μεθόδους: swim(αν μετά από μια προσθήκη, ένα στοιχείο έχει μεγαλύτερο κλειδί από τον πατέρα του τότε αντιμεταθέτει το στοιχείο με τον πατέρα του (τα συγκρίνει με ένα αντικείμενο τύπου DiskComparator) και συνεχίζει τις αντιμεταθέσεις ώσπου να μπει στη σωστή θέση), sink(αν κατά τη προσθήκη, ένα στοιχείο έχει μικρότερο κλειδί από τα παιδιά του, βρίσκει το παιδί με το μεγαλύτερο κλειδί και το αντιμεταθέτει με τον πατέρα. Οι αντιμεταθέσεις συνεχίζουν μέχρι το στοιχείο να μπει στη σωστή θέση), swap(χρησιμοποιείται για αντιμετάθεση δύο στοιχείων τύπου Disk) και grow(δημιουργεί έναν νέο πίνακα με μέγεθος του αρχικού συν έναν σταθερό ακέραιο που έχουμε ορίσει αντιγράφει τα στοιχεία στον νέο πίνακα και ενημερώνει τα αναφορές).

ΜΕΡΟΣ Β

Στο δεύτερο μέρος φτιάξαμε ένα πρόγραμμα το οποίο διαβάζει ένα txt αρχείο με ακεραίους στο διάστημα [0,1000000], οι οποίοι αναπαριστούν το μέγεθος των φακέλων, και τους τοποθετεί σε δίσκους με επαρκή χώρο. Το πρόγραμμα αποτελείται από 2 βασικές μεθόδους. Η CreateDisks η οποία παίρνει ως όρισμα το αρχείο με τους ακεραίους και επιστρέφει έναν πίνακα τύπου Disk. Η μέθοδος αυτή διαβάζει το αρχείο για να βεβαιωθεί ότι οι ακέραιοι βρίσκονται στο παραπάνω διάστημα. Αν παραβιάζεται αυτή η συνθήκη το πρόγραμμα τερματίζει. Διαφορετικά

φτιάχνει έναν πίνακα Disk και ξεκινάει να περνάει τις τιμές του αρχείου, με τη σειρά που εμφανίζονται, στη λίστα folders του πρώτου δίσκου που έχει διαθέσιμο χώρο μικρότερο ή ίσο με τον ακέραιο που διάβασε και ταυτόχρονα ενημερώνει τον διαθέσιμο χώρο. Η διαδικασία συνεχίζεται μέχρι να διαβάσει όλους τους ακέραιους. Η δεύτερη βασική μέθοδος είναι η Greedyy η οποία παίρνει ως όρισμα τον πίνακα που επέστρεψε η CreateDisks. Η μέθοδος αυτή χρησιμοποιεί την ουρά προτεραιότητας που υλοποιήσαμε στο Α μέρος. Δημιουργεί μία νέα ουρά, με όρισμα ένα αντικείμενο τύπου DiskComparator, για να μπορεί να συγκρίνει αντικείμενα τύπου Disk. Η μέθοδος διατρέχει τον πίνακα που δημιουργήσαμε, βλέπει πόσοι δίσκοι έχουν χρησιμοποιηθεί και ενημερώνει τις μεταβλητές disks_used και size_of_folders(το συνολικό άθροισμα των ακεραίων του αρχείου). Αφού γίνει αυτό, προσθέτει τα αντικείμενα Disk του πίνακα στην ουρά προτεραιότητας. Τέλος αφαιρεί από την ουρά τα στοιχεία, τα οποία πλέον εμφανίζονται με συγκεκριμένη σειρά (βάση του μεγίστου κλειδιού). Σε αυτή τη περίπτωση εμφανίζει τους δίσκους κατά φθίνουσα σειρά διαθέσιμου χώρου αποθήκευσης (υπεύθυνη για αυτό είναι η κλάση DiskComparator που υλοποιεί την κλάση Comparator και συγκρίνει 2 δίσκους βάση του διαθέσιμου χώρου). Επομένως με την μέθοδο getMax της ουράς τυπώνουμε τους δίσκους με την σειρά που θέλουμε.

Σημείωση: το compile του αρχείου Greedy.java γίνεται από cmd. Αν δεν περάσει το compile με την εντολή javac Greedy.java, χρησιμοποιήστε την εντολή javac Greedy.java –Xlint:unchecked ενώ για να τρέξει πρέπει να δώσουμε και το μονοπάτι για το txt αρχείο. (java Greedy path_to_file/filename.txt)

ΜΕΡΟΣ Γ

Στο τρίτο μέρος χρησιμοποιήσαμε τον αλγόριθμο ταξινόμησης quicksort. Ο αλγόριθμος παίρνει ως ορίσματα έναν πίνακα ακεραίων και τα όρια του πίνακα στα οποία θέλουμε να τον εφαρμόσουμε. Σαν στοιχείο pivot επιλέξαμε τα τελευταίο στοιχείο του πίνακα. Η υλοποίηση έγινε με αναδρομή. Σε κάθε κλήση της μεθόδου το pivot μπαίνει στη σωστή θέση (έστω α[i]) και γίνεται αναδρομική κλήση στους δύο υποπίνακες (α[p] έως α[i-1] και α[i+1] έως α[r], όπου p,r τα όρια του αρχικού πίνακα). Ο αλγόριθμος χρησιμοποιεί και την μέθοδο partition. Ο ρόλος της είναι ο εξής: σαρώνει τον πίνακα από τα αριστερά (με έναν δείκτη i) μέχρι να βρει στοιχείο >= του pivot και από τα δεξιά (με έναν δείκτη j) μέχρι να βρει στοιχείο <= του pivot. Στη συνέχεια αντιμεταθέτει τα στοιχεία και συνεχίζει μέχρι να συναντηθούν οι δείκτες ή να περάσει ο j τον i. Όταν γίνει αυτό τοποθετεί το pivot εκεί που σταμάτησε ο i. Η partition χρησιμοποιεί τις βοηθητικές μεθόδους less (για σύγκριση δύο ακεραίων) και exch (για αντιμετάθεση). Στη πράξη η μέθοδος quicksort είναι η πιο γρήγορη και ευρέως χρησιμοποιούμενη μέχρι σήμερα καθώς τρέχει σε χρόνο:

για average case O(NlogN) και worst case O(N^2), ωστόσο σπάνια συναντάμε το worst case.

ΜΕΡΟΣ Δ

Στο μέρος Δ φτιάξαμε 2 κλάσεις. Η πρώτη είναι η File_Creation η οποία δημιουργεί τα δεδομένα πάνω στα οποία κάνουμε τις συγκρίσεις. Υπεύθυνη γι' αυτό είναι η μέθοδος f_creation η οποία παίρνει σαν όρισμα τον αριθμό των ακεραίων που θέλουμε να έχει κάθε αρχείο. Αρχικά φτιάχνουμε ένα αντικείμενο τύπου Random. Επειδή όταν δημιουργείτε κάθε αρχείο το όνομα που παίρνει είναι Filei όπου i δείκτης από 1-30 (1-10 για txt αρχεία με 100 ακεραίους το καθένα, 11-20 με 500, 21-30 με 1000) αρχικοποιήσαμε μια μεταβλητή τύπου int με τιμή 1. Αν το N είναι 500 τότε η μεταβλητή παίρνει την τιμή 11 ώστε τα αρχεία που θα δημιουργηθούν να είναι τα Filei για i από 11-20. Όμοια αν N=1000 τότε η τιμή γίνεται 21. Ορίζουμε ένα αντικείμενο τύπου PrintWriter(writer) το οποίο θα μας βοηθήσει στη δημιουργία των αρχείων. Μ ένα for loop των 10 επαναλήψεων (όσα τα αρχεία με 100,500,1000 ακέραιους) φτιάχνουμε 10 αρχεία (εντολή writer = new PrintWriter("C:/Users/user/Desktop/3170152_3170124/data/File" +k+".txt", "UTF-8")).Με ένα δεύτερο for loop περνάμε τις τιμές των ακεραίων στο αρχείο. Για να γίνει αυτό δημιουργήσαμε μία μεταβλητή τύπου int στην οποία σε κάθε επανάληψη του δεύτερου for περνάμε μία τυχαία τιμή στο διάστημα [0,1000000] μέσω της εντολής x=rand.nextInt(1000001);. Τέλος με την εντολή writer.println(x); γράφουμε την τιμή της μεταβλητής x στο txt αρχείο. Στη main καλούμε την μέθοδο f_creation 3 φορές με παραμέτρους 100,500,1000 αντίστοιχα και δημιουργούνται τα 30 txt αρχεία.

Η σύγκριση των 2 αλγορίθμων γίνεται από την κλάση Comparison, η οποία αποτελείται από 2 βασικές μεθόδους. Η μέθοδος disk_count παίρνει ως όρισμα επιστρέφουμε έναν πίνακα Disk και επιστρέφει έναν ακέραιο (ο αριθμός των δίσκων που χρησιμοποιήθηκαν για την αποθήκευση των φακέλων). Μέσω ενός for loop (αριθμός επαναλήψεων = στοιχεία του πίνακα που δίνεται ως όρισμα) βρίσκουμε τους δίσκους στους οποίους έχει αποθηκευτεί κάποιος φάκελος (εντολή if(x[i].getFreeSpace()<1000000)) και τον συνολικά αριθμό τους. Η δεύτερη μέθοδος, η alg_compare, παίρνει ως παράμετρο έναν ακέραιο (αριθμός των ακεραίων σε κάθε αρχείο). Η μέθοδος αυτή διαβάζει το κάθε ένα από τα 10 αρχεία για συγκεκριμένη τιμή του N (Filei όμοια τεχνική με τον τρόπο που τα δημιουργήσαμε. Αν N=500 i=11 κ.ο.κ.). Για τον πρώτο αλγόριθμο καλεί τη μέθοδο CreateDisks της κλάσης Greedy (εξήγηση στο Β μέρος) η οποία επιστρέφει έναν πίνακα Disk. Στη συνέχεια αποθηκεύει σε μία μεταβλητή sum1 το άθροισμα των δίσκων που χρησιμοποίησε ο πρώτος αλγόριθμος για να αποθηκεύσει τους φακέλους (μέσω της disk_count). Όσον αφορά τον δεύτερο αλγόριθμο καλούμε την μέθοδο CreateDisks

της κλάσης Sort (ίδια μέθοδος με την CreateDisks της κλάσης Greedy αλλά χρησιμοποιούμε την μέθοδο quicksort, δηλαδή οι φάκελοι μπαίνουν κατά φθίνουσα σειρά). Σε μία μεταβλητή sum2 αποθηκεύουμε το άθροισμα των δίσκων που χρησιμοποίησε ο δεύτερος αλγόριθμος για να αποθηκεύσει τους φακέλους (μέσω της μεθόδου disk_count). Τέλος διαιρώντας τα sum1,sum2 με το 10 (όσα είναι και τα αρχεία για συγκεκριμένη τιμή του N) βρίσκουμε τον μέσο όρο των δίσκων που χρησιμοποιήθηκαν. Στη main καλώντας την alg_compare με N=100,500,1000 συγκρίνουμε τον μέσο όρο των δίσκων που χρησιμοποίησαν οι 2 αλγόριθμοι για 10 αρχεία με 100,500,1000 ακέραιους αντίστοιχα.

Από τα αποτελέσματα των συγκρίσεων συμπεραίνουμε ότι ο δεύτερος αλγόριθμος χρησιμοποιεί πάντα λιγότερους δίσκους σε σύγκριση με τον πρώτο. Συγκεκριμένα για τα δεδομένα που χρησιμοποιήσαμε: για N=100 ο πρώτος αλγόριθμος χρησιμοποίησε κατά μέσο όρο 54.7 δίσκους ενώ ο δεύτερος 51.5, για N=500 ο πρώτος 270.9 και ο δεύτερος 261.4 ενώ για N=1000 ο πρώτος 528.1 και ο δεύτερος 507.3. Παρατηρούμε ότι όσο αυξάνεται το N τόσο αυξάνεται και η διαφορά των δίσκων που χρησιμοποιούν οι αλγόριθμοι. (για N=100 η διαφορά είναι 3.2,για N=500 είναι 9.5, για N=1000 είναι 20.8). Άρα για μεγαλύτερες τιμές του N γίνεται πιο εύκολα ξεκάθαρο ότι ο δεύτερος αλγόριθμος είναι αρκετά πιο αποδοτικός.

Σημείωση: Αν με την εντολή javac Comparison.java δεν γίνει compile χρησιμοποιήστε την εντολή javac Comparison.java –Xlint:unchecked σε cmd. Επίσης επειδή τα txt αρχεία βρίσκονται σε διαφορετικό φάκελο από το αρχείο Comparison.java χρησιμοποιούμε το αντίστοιχο path (File file = new File("C:/Users/user/Desktop/3170152_3170124/data/File" +k+".txt") για να τα διαβάσουμε. Επομένως πρέπει να βρίσκονται σε αυτή τη θέση για να μπορεί να τα βρει η main.