ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ – ΕΡΓΑΣΙΑ 2

Βελισσαρίδης Γιώργος (P3210255)

Μέρος Α – MaxPQ

Έφτιαξα την ουρά προτεραιότητας χρησιμοποιώντας generics, με την προϋπόθεση ο generic τύπος να είναι Comparable, ώστε να μπορούν να εφαρμοστούν πάνω του οι λειτουργίες της ουράς. Για την υλοποίηση των λειτουργιών χρησιμοποίησα heap, ως array του οποίου τα στοιχεία ξεκινούν από τη θέση 1 (αντί για 0, ώστε να είναι εύκολη η αριθμητική δεικτών). Βασίστηκα σε μικρό βαθμό στον κώδικα του εργαστηρίου, αλλά χρησιμοποίησα πιο σύντομο κώδικα για την υλοποίηση των swim και sink. Υλοποίησα επίσης resizing array, όμως όχι με σταθερό increment, αλλά με τις εξής προδιαγραφές: αν ο πίνακας είναι γεμάτος και προστίθεται νέο στοιχείο, το μέγεθός του διπλασιάζεται, ενώ αν, αφού αφαιρεθεί ένα στοιχείο, γίνει κατά 25% γεμάτος, τότε το μέγεθός του μειώνεται στο μισό του.

Μέρος Β – Αλγόριθμος 1

Αρχικά, διαβάζεται το path για ένα σωστά μορφοποιημένο αρχείο κειμένου από τα command line arguments, και δημιουργείται ένα File «input» με αυτό το path. Το input δίνεται ως όρισμα στην στατική μέθοδο TASK_B(), που με τη σειρά της καλεί την στατική μέθοδο readFolderSizes(), που δέχεται επίσης το File input ως όρισμα. Η readFolderSizes() ελέγχει αν το path αντιστοιχεί σε μη κενό αρχείο, αν ναι, μετρά τις γραμμές του και αρχικοποιεί πίνακα ακεραίων “folderSizes” με

μήκος ίσο με τις γραμμές του αρχείου εισόδου. Στη συνέχεια, ελέγχεται κάθε γραμμή του input, και αν αντιστοιχεί σε ακέραιο μεταξύ του 0 και του 1.000.000 προστίθεται στον πίνακα folderSizes, που στο τέλος του διαβάσματος επιστρέφεται και ανατίθεται σε ομώνυμο πίνακα ακεραίων στη μέθοδο TASK_B(). Ο πίνακας αυτός δίνεται ως όρισμα στη στατική μέθοδο greedyBinPacking(), που εκτελεί την ανάθεση φακέλων σε δίσκους. Αρχικά ορίζει ένα max priority queue, και του προσθέτει ένα δίσκο ο οποίος περιέχει τον πρώτο φάκελο. Ακολούθως, διατρέχεται ο πίνακας folderSizes και για κάθε τιμή του (δηλαδή για κάθε μέγεθος φακέλου), αν αυτή είναι μεγαλύτερη από τον ελεύθερο χώρο που πιο άδειου δίσκου στην ουρά προτεραιότητας, τότε τοποθετείται σε νέο δίσκο, ο οποίος με τη σειρά του προστίθεται στην ουρά, ενώ αν η τιμή είναι μικρότερη ή ίση με τον ελεύθερο χώρο που πιο άδειου δίσκου, τότε αυτός ο δίσκος αφαιρείται από την ουρά, του προστίθεται η τιμή και τέλος ξαναπροστίθεται στην ουρά. Στο τέλος της διαδικασίας επιστρέφεται η ουρά προτεραιότητας και ανατίθεται στη μεταβλητή “resultAllocation” στη μέθοδο TASK_B(). Τελικά, η μεταβλητή αυτή δίνεται μαζί με τη μεταβλητή folderSizes ως όρισμα στη στατική μέθοδο printTasksBCResults(), η οποία υπολογίζει το συνολικό μέγεθος των φακέλων στο αρχείο εισόδου σε terabyte και τυπώνει το μέγεθος αυτό, καθώς και τον αριθμό δίσκων που χρησιμοποιήθηκαν στον Αλγόριθμο 1. Αν ο αριθμός φακέλων στο αρχείο εισόδου είναι το πολύ 100, τυπώνονται επίσης οι δίσκοι με τα id και τους περιεχόμενους φακέλους τους, σε σειρά από τον δίσκο με τον περισσότερο ελεύθερο χώρο προς αυτό με το λιγότερο.

Μέρος Γ – Αλγόριθμος Ταξινόμησης Mergesort

Χρησιμοποίησα τον αλγόριθμο ταξινόμησης Mergesort σε φθίνουσα σειρά. Στο αρχείο Sort.java όρισα μία στατική κλάση mergeSort(), οποία δέχεται τον πίνακα ακεραίων προς ταξινόμηση. Εκεί αρχικοποιείται ο βοηθητικός πίνακας ακεραίων και καλείται η στατική μέθοδος innerSort(), που με τη σειρά της κάνει αναδρομικές κλήσεις στον εαυτό της και στη στατική μέθοδο merge() που υλοποιούν την ταξινόμηση με συγχώνευση. Μία ιδιαιτερότητα της υλοποίησης είναι ο έλεγχος στην innerSort(), μετά την ταξινόμηση των δύο υποπινάκων, του αν το στοιχείο στη μεσαία τιμή υπό διερεύνηση είναι μεγαλύτερη από την αμέσως επόμενη, στην οποία περίπτωση δεν χρειάζεται συγχώνευση των δύο υποπινάκων, οπότε παρακάμπτεται, επιταχύνοντας την ταξινόμηση.

Μέρος Δ – Πειραματική αξιολόγηση των δύο αλγορίθμων

Δημιουργία τυχαίων δεδομένων

Όρισα την κλάση createRandomDataFiles, που αποτελείται από εναλλακτική main. Αρχικά δημιουργείται ο φάκελος data, ως υποφάκελος του φακέλου όπου εκτελείται το πρόγραμμα, μέσω της μεθόδου mkdir() της κλάσης File. Για κάθε μία από τις τιμές N που προσδιορίζονται στις οδηγίες της εργασίας (100, 500, 1000) δημιουργούνται 10 αρχεία κειμένου στον φάκελο data, με ονόματα που προσδιορίζουν την τιμή N, και αρίθμηση από το 1 έως και το 10, χρησιμοποιώντας τις κλάσεις File και FileWriter του πακέτου java.io. Σε καθένα από αυτά τα αρχεία, ανάλογα με το N που του αντιστοιχεί, γράφονται N γραμμές, που καθεμία αποτελείται από έναν ακέραιο μεταξύ 0 και 1.000.000, μέσω ενός αντικειμένου της κλάσης Random και της μεθόδου nextInt().

Σύγκριση αλγορίθμων στα παραγόμενα δεδομένα

Στην κλάση Greedy.java, δημιούργησα τη μέθοδο TASK_D(), η οποία εκτελεί τους αλγορίθμους 1 και 2 για τα 30 παραγόμενα αρχεία. Για τη σωστή της λειτουργία, πρέπει να περαστεί ως όρισμα στο command line το path για τον φάκελο data, όπου αποθηκεύτηκαν τα παραγόμενα αρχεία. Αρχικά, η TASK_D() διαβάζει το input που της περνά η main (βλ. μέρος Β), ελέγχει αν πρόκειται για έγκυρο path σε μη κενό φάκελο και αποθηκεύει καθένα από τα αρχεία κειμένου του φακέλου στον πίνακα File “fileArray”. Καθένα από τα αρχεία αυτά περνά από τη μέθοδο readFolderSizes(), και τα αποτελέσματα αποθηκεύονται στον πίνακα πινάκων ακεραίων “folderSizeArray”. Καθένας από τους πίνακες αυτούς περνά από τη μέθοδο greedyBinPacking(), και τα αποτελέσματα αποθηκεύονται στον πίνακα ουρών προτεραιοτήτων δίσκων “greedyAllocationArray”, υλοποιώντας έτσι τον αλγόριθμο 1 για όλα τα παραγόμενα δεδομένα . Στη συνέχεια οι πίνακες ταξινομούνται και ξαναπερνούν από τη μέθοδο greedyBinPacking(), αλλά αυτή τη φορά τα αποτελέσματα αποθηκεύονται στον πίνακα ουρών προτεραιοτήτων δίσκων “greedyDecrAllocationArray”, υλοποιώντας έτσι τον αλγόριθμο 2 για όλα τα παραγόμενα δεδομένα. Δηλώνονται μετρητές για κάθε N (100, 500, 1000) και κάθε αλγόριθμο που αρχικοποιούνται με 0. Στη συνέχεια εκτελούμε βρόγχο στο fileArray και για κάθε αρχείο κειμένου που περιέχει, ελέγχεται σε ποια τιμή N αυτό αντιστοιχεί με βάση το όνομά του και οι αριθμοί

χρησιμοποιούμενων δίσκων από τους δύο αλγορίθμους κατά την επεξεργασία του συγκεκριμένου αρχείου προστίθεται στους αντίστοιχους μετρητές. Υπολογίζονται, με βάση τους μετρητές, οι μέσες τιμές χρησιμοποιούμενου αριθμού δίσκων για κάθε τιμή N και αλγόριθμο. Τέλος, τυπώνονται τα αποτελέσματα.

Τα αποτελέσματα που πήρα είναι τα εξής:

N = 100: Greedy = 60.5, Greedy-Decreasing = 54.4

N = 500: Greedy = 297.1, Greedy-Decreasing = 259.6

N = 1000: Greedy = 586.6, Greedy-Decreasing = 509.5

Συμπερασματικά, μπορούμε να πούμε ότι ο δεύτερος αλγόριθμος, greedy-decreasing bin packing είναι αποδοτικότερος από τον πρώτο, greedy bin packing, καθώς χρησιμοποιεί λιγότερους δίσκους για την αποθήκευση των ίδιων φακέλων. Η διαφορά μάλιστα μεταξύ τους γίνεται όλο και πιο φανερή για μεγαλύτερες εισόδους.