ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΔΕΚΕΜΒΡΙΟΣ 2019 ΕΡΓΑΣΙΑ 2

ΚΩΤΣΗΣ-ΠΑΝΑΚΑΚΗΣ ΒΑΣΙΛΕΙΟΣ-ΕΚΤΩΡ (Ρ3180094)

ΓΕΩΡΓΙΟΥ ΑΛΕΞΙΟΣ - ΛΑΖΑΡΟΣ (Ρ3180027)

Μέρος Α (Υλοποίηση ουράς προτεραιότητας)

Η ουρά προτεραιότητας που χρησιμοποιήθηκε σε αυτήν την εργασία αποτελεί μια τροποποιημένη μορφή της ουράς προτεραιότητας που παρουσιάστικε στο εργαστήριο του μαθήματος. Διακατέχει τις βασικές μεθόδους μιας ουράς προτεραιότητας **insert()** (τοποθέτηση στοιχείου στην ουρα με βάση το κλειδί του) και **getMax()** (Αφαίρεση του στοιχείου με το μεγαλύτερο κλειδί). Η ανανέωση των στοιχείων της λίστας γίνεται μόνο με τις δυο προαναφερόμενες μεθόδους, με την βοήθεια δύο βοηθητικών μεθόδων. Την **sink()**(κατάδυση στον σορό) και την **swim ()**(ανάδυση στον σορό). Η δομή δεδομένων, φυσικά, τροποποιήθηκε ανάλογα με τις ανάγκες της εργασίας. Προστέθηκε μια μέθοδος επιστροφής του μεγέθους της ουράς, η **getSize()**, μια μέθοδος που, δηλαδή, επιστρέφει τον συνολικό αριθμό των στοιχείων που βρίσκονται μέσα στην ουρά. Η λίστα έχει υλοποιηθεί με την χρήση Generics και με την χρήση ενός **generic comparator**, για την σύγκριση των γειτνιακών στοιχείων της ουράς (πατέρας με γιό).

Μέρος Β (Χρήση ουράς προτεραιότητας στον Αλγόριθμο 1)

Η ουρά προτεραιότητας χρησιμοποιήθηκε για την πιο αποτελεσματική υλοποίηση του αλγορίθμου 1, αντί λίστας η πίνακα. Αρχικά, δημιουργείται μια ουρά προτεραιότητας με όνομα disks, με όρισμα στον Constructor τον **SpaceComparator**, ο οποίος είναι ένας comparator που συγκρίνει τα αντικείμενα τύπου Disk που θα αποθηκεύσουμε στην ουρά ως προς τον ελευθερό τους χώρο. Έπειτα, τοποθετείται με την μέθοδο **insert()** ο πρώτος δίσκος στον οποίο θα μπει σίγουρα το πρώτο αρχείο στην λίστα.Γενικά, επειδή θα προσθεθούν πολλοί δίσκοι, η λογική που μας οδηγεί στην επιλογή χρήσης ουράς προτεραιότητας για αυτό το πρόβλημα είναι πως θέλουμε ο φάκελος να αποθηκευθεί στον δίσκο με τον μεγαλύτερο ελευθερο χώρο. Συνεπώς, χρησιμοποιώντας μια ουρά προτεραιότητας μεγιστοστρεφούς σορού, μπορούμε πάντα να έχουμε αναφορά στο στοιχείο με το μεγαλύτερο κλειδί, στην προκειμένη περίπτωση τον μεγαλύτερο ελεύθερο χώρο, και να του προσθέσουμε τον φάκελο, χωρίς να απαιτείται προσπέλαση του σορού, με την **insert().** Το ενδιαφέρον παρουσιάζεται όταν το πρόγραμμα μπαίνει στον πρώτο επαναλειπτικό βρόγχο for (Ο οποίος εκτελείται τόσες φορές, όσος και ο αριθμός των φακέλων προς αποθήκευση), οπού με την χρήση της μεθόδου **peek()** της ουράς προτεραιότητας ελέγχεται εαν το μέγεθος του φακέλου προς αποθήκευση δεν ξεπερνά τον ελεύθερο χώρο του σκληρού δίσκου που έχει προτεραιότητα στην ουρά. Αν όντως δεν το

ξεπερνά, τότε ακολουθεί η εξής διαδικασία. Ο φάκελος προς αποθήκευση αποθηκεύεται σε αυτόν τον δίσκο, και στην συνέχεια δημιουργείται μια προσωρινή μεταβλητή temp που αποθηκεύει τα περιεχόμενά του, καθώς με την χρήση της **getMax()**, και της insert(), πρέπει να επανεντάξουμε τον πλέον πιο γεμάτο σκληρό δίσκο στην ουρά προτεραιότητας, ουσιαστικά δίνοντας του έτσι καινούργια προτεραιότητα στην ουρά, εφόσον πλέον το κλειδί του έχει μειωθεί. Εαν το ξεπερνά, τότε σημαίνει πως κανένας απο τους δίσκους στην ουρά δεν γίνεται να φιλοξενήσει τον φάκελο, εφόσον ο έλεγχος έγινε στον δίσκο που έχει μέγιστη προτεραιότητα, δηλαδή τον μεγαλύτερο ελεύθερο χώρο. Επομένως, δημιουργείται (με την χρήση ενός Constructor που παίρνει όρισμα folder, βλ Disk.java) ενας νέος σκληρός δίσκος, στον οποίον και αποθηκεύεται αυτός ο λιμνάζων φάκελος, και ύστερα εισάγεται στην ουρά προτεραιότητας στην κατάλληλή του θέση. Αφότου έχουν εξαντληθεί όλοι οι φάκελοι στην λίστα, οι δίσκοι πλέον θα τους έχουν αποθηκεύσει και θα είναι διατεταγμένοι στην ουρά προτεραιότητας με τέτοιον τρόπο ωστε να μπορούμε να τους αφεραίσουμε και να τους τυπώσουμε με την χρήση της **getMax()** έναν προς έναν με φθίνουσα σε ελεύθερο χώρο σειρά, εάν αυτό ζητηθεί.

Βέβαια, αυτός ο τρόπος αποθήκευσης δεν αποτελεί τον πιο αποτελεσματικό και αποδοτικό τρόπο που μπορούμε να σκεφτούμε με την χρήση μιας ουράς προτεραιότητας,

Μέρος Γ (Ταξινόμηση QuickSort)

Η συνάρτηση ταξινόμησης που χρησιμοποιήθηκε για την ταξινόμηση του πίνακα των φακέλων που διαβάστηκαν απο το .txt επιλέχθηκε να είναι η QuickSort, καθώς είναι μια πολύ αποδοτική επιλογή για ταξινόμηση πινάκων, των οποίων ξέρουμε το μέγεθός τους. Η QuickSort σε γενικές γραμμές δουλεύει με την λογική του σπάσιμου του πίνακα σε μικρότερα ταξινομημένα κομμάτια (διαίρει και βασίλευε). Ενα τυχαίο στοιχείο του πίνακα επιλέγεται ως pivot (στην δικιά μας περίπτωση επιλέχθηκε το τελευταίο στοιχείο). Ο δεξής υποπίνακας του pivot θα είναι στοιχεία μικρότερά του, ενώ ο αριστερός μεγαλύτερα του. Αυτή η λογική εφαρμόζεται και στους υποπίνακες αναδρομικά, μέχρις ώτου να καταλήξουμε σε έναν πίνακα στοιχείων ταξινομημένων κατα φθίνουσα σειρά. Χρησιμοποιούμε δύο δείκτες, j και I. Ο I λαμβάνει τιμή (start-1). Ο j δρα σαν μετρητης επαναληπτικου βρόγχου for. Εαν συναντήσει στοιχείο μεγαλύτερο του pivot, τότε ο I προσαυξάνεται κατά 1 και το στοιχείο που δείχνει ο j εναλλάσεται με το στοιχείο που δείχνει το I. Αυτή η διαδικασία συνεχίζεται μέχρις ώτου να δείχνει το j στον pivot, όπου μόλις γίνει αυτό, το pivot εναλλάσεται με το στοιχείο που δείχνει ο I+1 (δηλαδή ένα στοιχείο που είναι σίγουρα μεγαλύτερό του). Με αυτόν τον τρόπο, τα στοιχεία δεξιά του pivot πλέον είναι μικρότερα, και τα αριστερά του μεγαλύτερα. Στην συνέχεια, καλούμε την quicksort για τους υποπίνακες μέχρι να έχουν μείνει πίνακες ενός στοιχείου, πράγμα που σημαίνει πως η ταξινόμιση ολοκληρώθηκε.

Μέρος Δ (Σύγκριση Αλγορίθμων 1 και 2 με τυχαία δεδομένα)

Φτιάχτηκε μια κλάση με main η οποία παράγει 10 txt αρχεία με **N** (Όπου N μπορεί να είναι 100, 500 και 1000) τυχαίους αριθμούς από το 1 μέχρι το 1,000,000 σε κάθε γραμμή του κειμένου χρησιμοποιώντας την κλάση Random της java, με όνομα FolderGenerator. Τα αρχεία αποθηκεύονται στο φάκελο "data" και σε subfolders μέσα με όνομα το αντίστοιχο N. Για την υλοποίηση της γεννήτριας, χρησιμοποιήθηκε η κλάση Random της java.util. Φτιάχνουμε μέσα στην κλάση μία στατική μέθοδο **generate()** που παίρνει σαν όρισμα τον αριθμό των τυχαίων αριθμών που θέλουμε να φτιάξουμε μέσα σε ένα αρχείο. Μέσα στην μέθοδο, κάνουμε τα εξής:

1) Φτιάχνουμε ενα αντικείμενο rand τύπου Random για να έχουμε τις μεθόδους που χρειαζόμαστε.

2) Φτιαχνουμε έναν πίνακα απο String με όνομα **folders**, η οποία είναι η λίστα που θα επιστραφεί στο τέλος απο την μέθοδο.

3) Μπαίνουμε στον επαναληπτικό βρόγχο **for**, ο οποίος θα τρέξει τόσες φορές όσο το length του πίνακα **folders**. Για κάθε iteration, το I-οστό στοιχείο του folders θα πάρει την τιμή ενός τυχαίου int (που θα έχει μετατραπεί σε τύπο String με την βοήθεια της μεθόδου **valueOf**, για ευκολεία αργότερα στην εγγραφή) απο το 1 μέχρι το 1.000.000, με την χρήση της μεθόδου **nextInt()** του rand (ορισμά το άνω όριο 1.000.001 στην nextInt(), και προσθέτουμε το 1, ώστε να μην υπάρχει ενδεχόμενο να δημιουργηθεί int ίσος με 0, γιατί δεν έχει κανένα νόημα σαν φάκελος).

4) Επιστρέφουμε τον πλέον γεμάτο με ακέραιους πίνακα **folders**

Μετά, μέσα στην main της FolderGenerator, τρέχουμε ένα πρόγραμμα εγγραφής των δεδομένων που θα δημιουργήσουμε απο την generate(), 10 φορές για κάθε N(Στην προκειμένη περίπτωση 10 * αριθμός των N (100, 500, 1000), δηλαδή 30 φορές. Ωστόσο θα μπορούσαν να ήταν και περισσότερες, με τις κατάλληλες τροποποιήσεις στον πηγαίο κωδικα, αλλά δεν ασχολούμαστε με αυτό στην εργασία αυτήν).

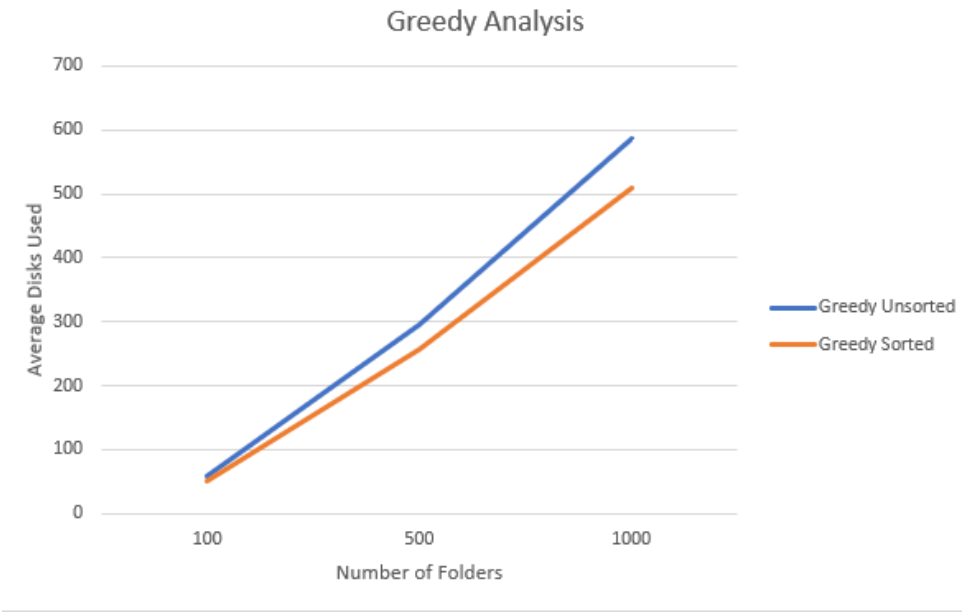
Αφού τρέξει το generator των τυχαίων αριθμών, μπορούμε να τρέξουμε την main των αλγορίθμων η οποία θα διαβάσει τα txt αρχεία από τα γνωστά directory και θα τα περάσει στην δομή της ουράς προτεραιότητας και τρέχοντας τους υλοποιημένους αλγορίθμους (Greedy-Unsorted και Greedy-Sorted) θα παράγει το μέσο όρο των δίσκων που χρειάζονται για κάθε N, ώστε να γίνει μια πρόχειρη ανάλυση/σύγκριση των δυο αλγορίθμων. Αυτός ο αλγόριθμος βρίσκεται στην κλάση **AlgorithmCompare**, και περιέχει τα εξής βήματα:

1) Φτιάχνουμε 3 αντικείμενα τύπου **File (dir1,dir2,dir3)** με όρισμα στον constructor τους το path στο directory απο το οποίο περιμένουμε να τα διαβάσουμε. Στη συνέχεια, φτιαχνουμε μία λίστα **dirs** με όλα τα 3 αντικείμενα στη σειρά μέσα της, που θα μας βοηθήσει στην προσπέλαση, και μία βοηθητική λίστα **Nlist**, που θα δρά σαν δείκτης που θα ενημερώνει τον χρήστη για ποιό directory γίνονται οι συγκρίσεις του αλγορίθμου

2) Μπαίνουμε στον πρώτο επαναληπτικό βρόγχο **for**, ο οποίος εκτελείται τόσες φορές όσα και τα αντικείμενα τύπου File (3 δηλαδή). Σε αυτόν τον βρόγχο αρχικοποιούνται 2 ακέραιες μεταβλητές, **sumOfDisks1** και **sumOfDisks2**, που στο τέλος θα έχουν την τιμή των συνολικών δίσκων που χρειάζονται 10 αρχεία απο συνολικά N φακέλους τυχαίου μεγέθους ο καθένας σε καθέναν απο τους 2 αλγόριθμους (sumOfDisks1 για τον αλγόριθμο 1, και sumOfDisks2 για τον αλγόριθμο 2).

3) Μέσα στην for, εμφολεύουμε μια δεύτερη for, η οποία τρέχει τόσες φορές, όσα και τα αρχεία που βρίσκονται στο τρέχων directory (με την βοήθεια της μεθόδου **listFiles()**, που επιστρέφει πίνακα με τα υπάρχοντα αρχεία στο directory με αλφαβητική σειρά). Το μόνο που μένει τώρα είναι η συνεχής ανανέωση των μεταβλητών sumOfDisks, αυξάνοντάς τους με το αποτέλεσμα που επιστρέφουν οι αλγόριθμοι 1 και 2 στον καθένα αντίστοιχα, για κάθε αρχείο στο τρέχων directory. Για αυτόν τον λόγο, στους αλγόριθμους 1 και 2, βάλαμε να επιστρέφουν και τον αριθμό των δίσκων που χρησιμοποιούνται, πέρα απο την παρουσίαση των αποτελεσμάτων που ζητείται στο Μέρος Β και Μέρος Γ.

4) Τέλος, εκτυπώνουμε με κατάλληλο formatting τον μέσο όρο των δίσκων που αξιοποιήθηκαν για τον κάθε αλγόριθμο που χρησιμοποιήθηκε (δηλαδή sumOfDisks/10). Η καμπύλη σχέσης του μέσου όρου των δίσκων που χρειάστηκαν και του αριθμού των φακέλων που ήταν αποθηκευμένοι σε κάθε αρχείο μπορεί να φανεί παρακάτω.



N	100	500	1000
Greedy Unsorted Average Disks	58	295	587
Greedy Sorted Average Disks	52	257	510

Προφανώς, αν ταξινομήσουμε τους φακέλους από το μεγαλύτερο στο μικρότερο σε μέγεθος πριν ξεκινήσουμε τον αλγόριθμο τότε θα χρειαστούμε λιγότερους δίσκους για την αποθήκευση των φακέλων (Αυτό όπως προαναφέρθηκε, γίνεται με την χρήση της QuickSort στα πλαίσια της εργασίας). Ενώ χωρίς ταξινόμηση, ο συνολικός αριθμός απαιτούμενων σκληρών δίσκων κατα πάσα πιθανότητα θα είναι μεγαλύτερος. Βέβαια το κόστος της ταξινόμησης πρέπει να αναγνωρίζεται για την επιλογή του κατάλληλου αλγορίθμου σε κάθε περίπτωση. Γενικά όσο μεγαλώνει το N αυξάνει και η διαφορά του μέσου ορού δίσκων των δυο μεθόδων αλλά φαίνεται να είναι εξαρτωμένη από τον αριθμό των φακέλων περίπου 0.07 * N.