2η Εργασία στις Δομές Δεδομένων
2019-2020
Υπ. Καθηγητής: Ε. Μαρκάκης
Φοιτητές: Πιπιλίκας Ευάγγελος 3180157 / Αλεξόπουλος Νικόλαος 3180004

Μέρος Α: class MaxPQ

Για την υλοποίηση της Ουράς Προτεραιότητας χρησιμοποιήσαμε τον κώδικα του 6ου εργαστηρίου. Η Ουρά Προτεραιότητας υλοποιείται με χρήση ενός μονοδιάστατου Array (που αναπαριστά σωρό) με αρχικό μέγεθος 4. Εάν η ουρά προτεραιότητας κριθεί πως δεν έχει επαρκή ελεύθερο χώρο και χρειαστεί να προσθέσουμε κάποιο στοιχείο τότε σε αυτή την περίπτωση αντιγράφουμε την αρχική σε μία νέα ουρά προτεραιότητας η οποία είναι μεγαλύτερη σε μέγεθος κατά 4 (**μέθοδος mergeAndGrow()**).

Μέθοδος εισαγωγής στην ουρά προτεραιότητας: insert()

Η μέθοδος insert() αρχικά ελέγχει εάν υπάρχει ελεύθερος χώρος στην ουρά προτεραιότητας για προσθήκη ενός δίσκου. Εάν δεν υπάρχει τότε καλείται η mergeAndGrow() η οποία εκτελεί την διαδικασία που περιγράφηκε πρότινος. Έπειτα προσθέτουμε στο αμέσως επόμενο ελεύθερο κελί της ουράς προτεραιότητας τον Δίσκο. Τέλος καλείται η μέθοδος swim() για την αποκατάσταση της ουράς προτεραιότητας.

Μέθοδος αποκατάστασης της ουράς προτεραιότητας: swim()

Η swim() συγκρίνει επαναληπτικά ένα παιδί με τον γονέα του ελέγχοντας αν ο τελευταίος είναι «μικρότερος» από το παιδί (εδώ εάν ο Δίσκος-γονέας έχει λιγότερο ελεύθερο χώρο από το παιδί) και εάν ισχύει αυτό, τους αντιμεταθέτει. Η διαδικασία αυτή σταματά όταν ο γονέας δεν είναι μικρότερος **ή** ο κόμβος ο οποίος αναδύεται φτάσει στην ρίζα της ουράς προτεραιότητας.

Μέθοδος αποκατάστασης της ουράς προτεραιότητας: sink()

Η sink() αρχικά βρίσκει ποιο είναι το «μεγαλύτερο» παιδί ενός κόμβου. Έπειτα συγκρίνει το συγκεκριμένο (μεγαλύτερο) παιδί με τον γονέα και εάν ο γονέας είναι «μικρότερος» (εδώ, ο γονέας-Δίσκος έχει λιγότερο ελεύθερο χώρο) από το παιδί τους αντιμεταθέτει. Η διαδικασία επαναλαμβάνεται μέχρι ο γονέας να μην έχει κανένα «μεγαλύτερο» παιδί ή η διαδικασία της «βύθισης» ενός στοιχείου να φτάσει ως κάποιο φύλλο.

Μέρος Β: Αλγόριθμος 1

Ο αλγόριθμος 1 καλείται να διασχίσει ένα .txt αρχείο που περιέχει τα μεγέθη (σε MB) φακέλων προς αποθήκευση σε Δίσκους του ενός TB. Για την υλοποίηση της παραπάνω διαδικασίας έχουμε δημιουργήσει μια κλάση με όνομα GreedyImp και η οποία περιέχει τις μεθόδους που χρειαζόμαστε. Για δική μας ευκολία και όπως υποδεικνύεται και στα Hints της εργασίας, η GreedyImp περιέχει τη μέθοδο **inpToArr()** η οποία αποθηκεύει τα περιεχόμενα του .txt αρχείου σε ένα μονοδιάστατο Array. Η μέθοδος **input()** σαρώνει αυτό το Array και για κάθε στοιχείο του καλεί την μέθοδο **chooseDiskAndAddFolder().** Εδώ θα περιγράψουμε το **πώς** χρησιμοποιήσαμε την ουρά προτεραιότητας για την υλοποίηση του αλγορίθμου 1.

Η ιδέα για τη χρήση της ουράς προτεραιότητας δίνεται από την περιγραφή του αλγορίθμου Greedy. Εφόσον ένας φάκελος χωράει σε κάποιο από τα ήδη δημιουργημένα αντικείμενα Δίσκου τότε τον αποθηκεύουμε σε αυτόν τον Δίσκο με τον **περισσότερο** ελεύθερο χώρο. Συνεπώς εκμεταλλευόμαστε την ιδιότητα της ουράς προτεραιότητας να έχει ως ρίζα της τον Δίσκο αυτόν. Έτσι μπορούμε να ελέγχουμε άμεσα το **αν** υπάρχει δίσκος που να χωρά τον εκάστοτε φάκελο και ταυτόχρονα ο Δίσκος αυτός να έχει τον περισσότερο ελεύθερο χώρο(δίχως να σαρώνουμε, εναλλακτικά, μία αταξινόμητη ή άναρχη λίστα ή έναν πίνακα ώστε να βρούμε τον Δίσκο με τον περισσότερο ελεύθερο χώρο. Ειδικά εάν δεν υπάρχει κάποιος τέτοιος Δίσκος τότε ο αλγόριθμός μας δεν θα ήταν καθόλου οικονομικός.).

Η μέθοδος **chooseDiskAndAddFolder()** αρχικά ελέγχει αν η ουρά προτεραιότητας είναι άδεια. Εάν είναι, δημιουργεί το πρώτο αντικείμενο Δίσκου, του προσθέτει τον πρώτο φάκελο και καλεί την μέθοδο **insert()** της ουράς προτεραιότητας. Αν η ουρά δεν είναι άδεια τότε ελέγχει αν ο Δίσκος στη ρίζα της, έχει αρκετό ελεύθερο χώρο για τον φάκελο προς αποθήκευση. Αν ναι, τότε τον προσθέτει

και στη συνέχεια επειδή πιθανότατα έχει διαταραχθεί η ιδιότητά της ουράς προτεραιότητας καλεί την μέθοδο **sink()** για την ρίζα.

Η class GreedyImp έχει μια static (class variable) μεταβλητή, την readCorrectly, η οποία ελέγχει εάν το .txt αρχείο είναι ορθό.

Ο αλγόριθμος "τρέχει" από την κλάση Greedy η οποία περιέχει την μέθοδο main. Η main δημιουργεί ένα αντικείμενο της GreedyImp στον κατασκευαστή της οποίας στέλνει ως παράμετρο το μονοπάτι του .txt αρχείου με τα μεγέθη των φακέλων. Αρχικά καλείται η **inpToArr()** για τη δημιουργία του πίνακα με τα περιεχόμενα του .txt αρχείου. Εφόσον αυτή γίνει σωστά (readCorrectly == true) καλείται η **input()** και η **output()** για την εμφάνιση των αποτελεσμάτων όπως περιγράφονται στην εκφώνηση του **Μέρους Β.**

<u>Μέρος Γ: Αλγόριθμος Ταξινόμησης QuickSort</u>

Βασιζόμενοι στην κλάση GreedyImp η οποία διαβάζει το .txt αρχείο και το αποθηκεύει σε έναν μονοδιάστατο πίνακα, δημιουργήσαμε την κλάση Sort στην οποία θα κάνουμε pass τον πίνακα αυτόν και η οποία θα τον ταξινομήσει σε φθίνουσα σειρά.

Όπως και πριν λοιπόν περνάμε στην GreedyImp το .txt αρχείο με τα μεγέθη των φακέλων και μέσω της inpToArr() δημιουργούμε ένα μονοδιάστατο Array με τα περιεχόμενά του .txt. Έπειτα περνάμε το Array στην κλάση Sort και εκεί εφαρμόζουμε την μέθοδο ταξινόμησης QuickSort με βάση τον κώδικα που περιέχεται στις διαφάνειες του μαθήματος. Η QuickSort πραγματοποιείται με χρήση του **τελευταίου στοιχείου** του πίνακα (a[r]) το οποίο η μέθοδος **partition()** τοποθετεί στην σωστή του θέση. Έπειτα μέσω αναδρομικής κλήσης της μεθόδου **qSort()** πρώτα για τον υποπίνακα από την αρχή του μέχρι την προηγούμενη από τη θέση που τοποθετήθηκε το εκάστοτε pivot και έπειτα για τον υποπίνακα από το pivot μέχρι και το τελευταίο στοιχείο .

Η **partition()**, έχοντας δεδομένο ότι το pivot είναι το τελευταίο στοιχείο του εκάστοτε πίνακα και θέλοντας να γίνει φθίνουσα ταξινόμηση, σαρώνει τον πίνακα πρώτα από αριστερά προς τα δεξιά μέχρι να βρεθεί ένα στοιχείο, **μικρότερο ή ίσο** του pivot **(***while (a[++i] > pvt);***)**. Στη συνέχεια σαρώνει από τα δεξιά προς τα αριστερά τον πίνακα μέχρι να βρεθεί στοιχείο **μεγαλύτερο ή ίσο** του pivot**(***while (pvt > a[--j])***)**. Εάν το πρώτο στοιχείο **ΔΕΝ** βρίσκεται «δεξιότερα» του δεύτερου τότε τα δύο στοιχεία αυτά αντιμετατίθενται. Τέλος, μόλις τελειώσει η σάρωση, αντιμετατίθεται το pivot με το στοιχείο στη θέση i και πλεόν το pivot βρίσκεται στη σωστή του θέση. Με τον τρόπο αυτό κάθε στοιχείο αριστερά του pivot είναι μεγαλύτερο από αυτό και κάθε στοιχείο δεξιά του είναι μικρότερο, χωρίς αυτά να είναι απαραίτητα ταξινομημένα.

Αφού επιστραφεί η τιμή του i στην qSort καλείται αναδρομικά η συνάρτηση qSort για τον υποπίνακα από το αριστερότερο μέχρι πριν το στοιχείο στη θέση i και για τον υποπίνακα από τη θέση i+1 μέχρι το δεξιότερο στοιχείο. Έτσι ταξινομούνται και τα υπόλοιπα στοιχεία στους ολοένα και μικρότερους σε μέγεθος υποπίνακες.

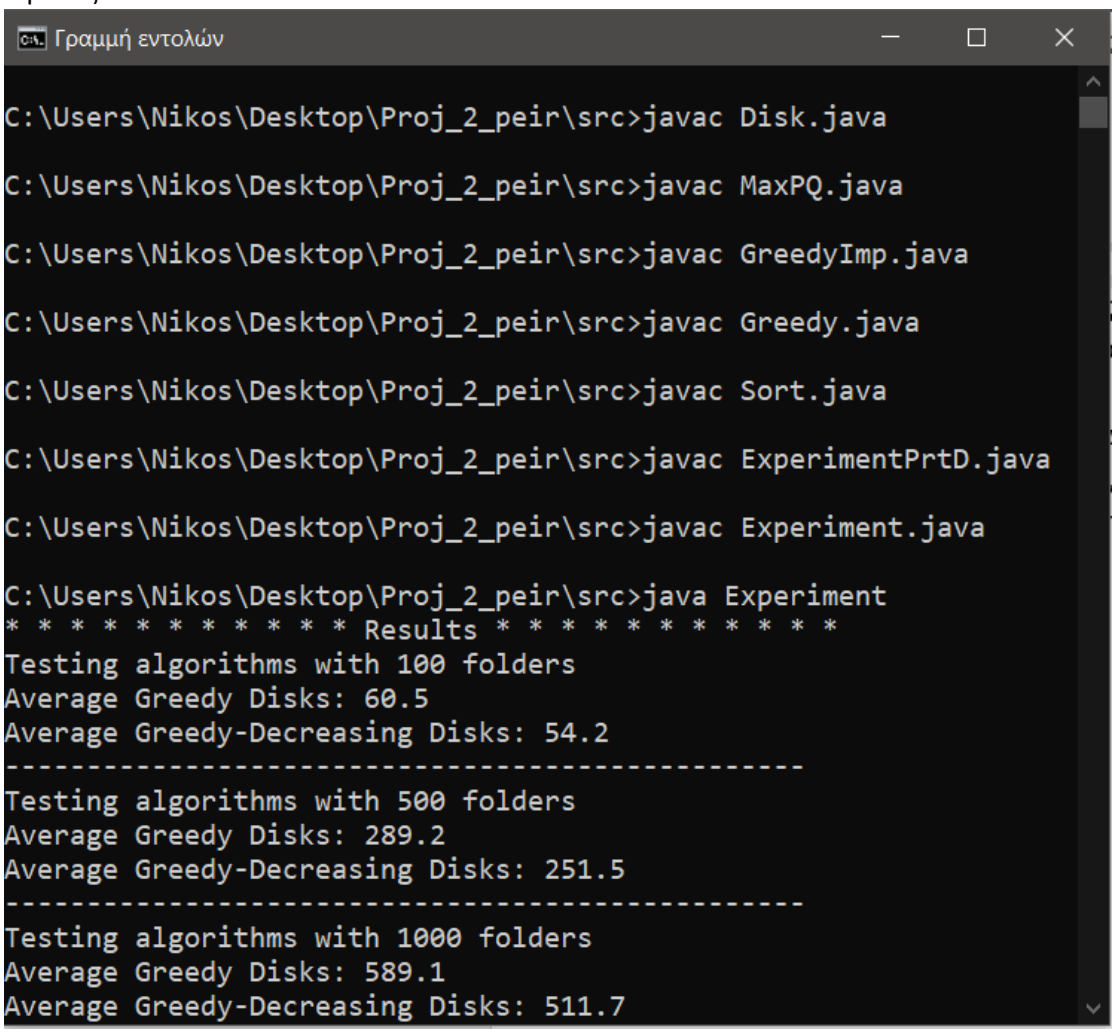
<u>Μέρος Δ: Πείραμα και Αξιολόγηση</u>

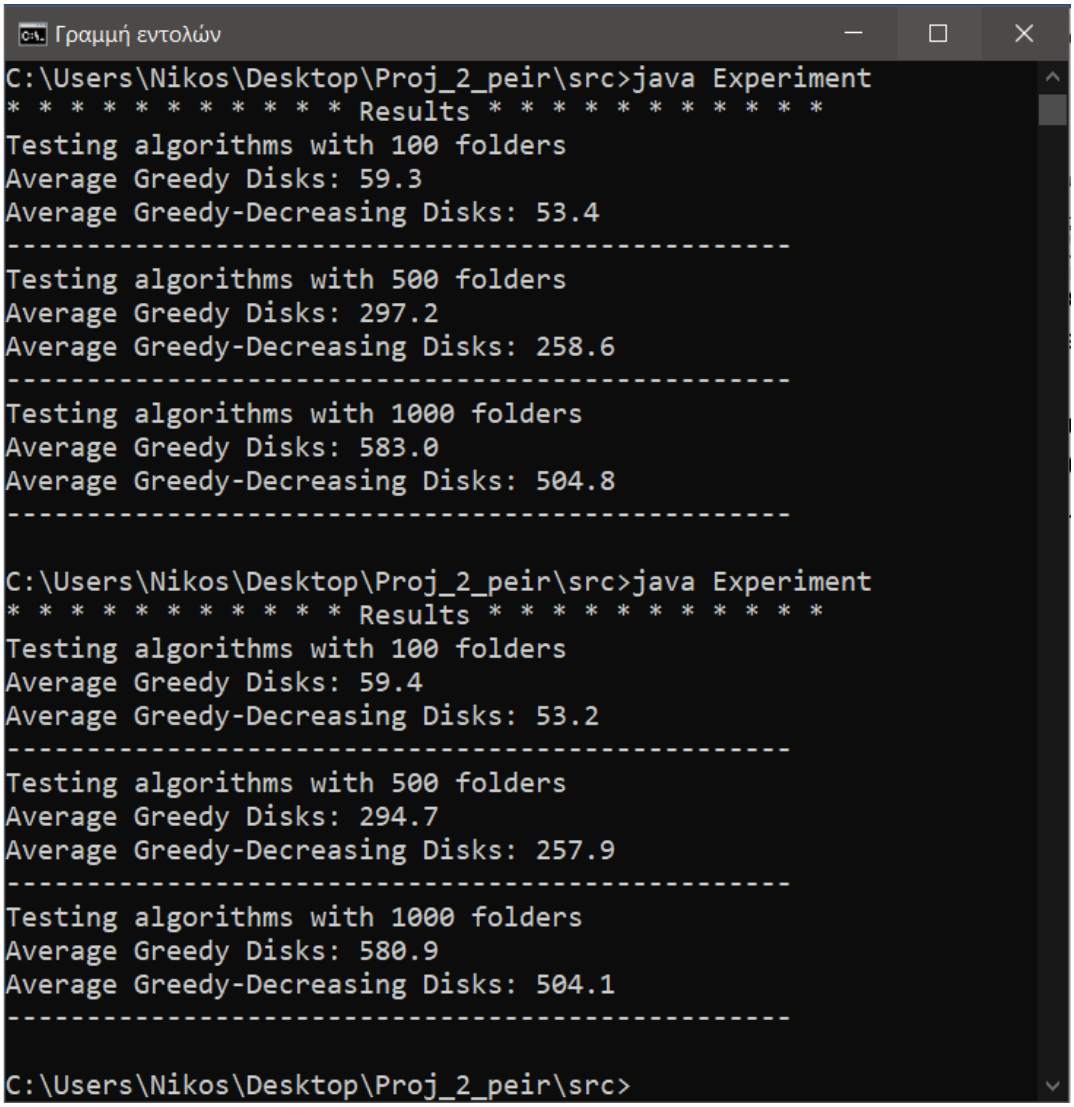
Η υλοποίηση της σύγκρισης της απόδοσης των δύο αλγορίθμων γίνεται στην κλάση ExperimentPrtD με τις ανάλογες υπολογιστικές μεθόδους. Ακολουθώντας τις οδηγίες της εκφώνησης αρχικά γράψαμε την μέθοδο **createFiles()** η οποία απλώς δημιουργεί 30 κενά .txt αρχεία στα οποία θα «γράψει» η μέθοδος **writeFile()** στη συνέχεια.
Η **writeFile()** δέχεται ως όρισμα το πλήθος των μεγεθών φακέλων που θα γραφούν στο αρχείο, έστω n, και με χρήση της κλάσης **Random** γράφει στα .txt αρχεία n τυχαίους αριθμούς $\in [0, 1000000]$.

Έπειτα ορίζουμε 2 μεθόδους (**testingGreedy()** και **testingGreedyDecreasing()**) οι οποίες επιτελούν το τρέξιμο των αλγορίθμων 1 και 2 αντίστοιχα και οι οποίες επιστρέφουν τον ακέραιο αριθμό των Δίσκων που απαίτησε η κάθε μια για την αποθήκευση **όλων** των φακέλων.

Τέλος, η μέθοδος **experiment()** δημιουργεί και **επιστρέφει** έναν float δισδιάστατο πίνακα με τα αποτελέσματα των δύο αλγορίθμων. Αναλυτικότερα, στο συγκεκριμένο παράδειγμα θέλουμε να τρέξουμε τους αλγορίθμους και να συγκρίνουμε τα αποτελέσματά τους για 100, 500 και 1000 μεγέθη φακέλων προς αποθήκευση. Δηλαδή 3 περιπτώσεις. Άρα ο πίνακας αποτελείται από 3 γραμμές και 2 στήλες. Στα κελία του πίνακα, αποθηκεύουμε τους συνολικούς δίσκους, **κατά μέσο όρο,** που απαίτησε ο αλγόριθμος 1 (στα δέκα διαφορετικά .txt αρχεία), στη στήλη 1, και ο αλγόριθμος 2 (επίσης στα δέκα διαφορετικά .txt αρχεία), στη στήλη 2 για κάθε τιμή του N (αριθμός μεγεθών φακέλων). **Παράδειγμα:** Αν ο αλγόριθμος 1 απαίτησε και για τα 10 .txt αρχεία που περιέχουν 100 μεγέθη 150 Δίσκους κατά μέσο όρο και ο αλγόριθμος 2 129 Δίσκους κατά μέσο όρο τότε το κελί με συντεταγμένες (0,0), ξεκινώντας την αρίθμηση από το 0, θα περιέχει το στοιχείο 150 και το κελί με συντεταγμένες (0,1) το στοιχείο 129.

Η εκτέλεση του πειράματος γίνεται στην κλάση **Experiment.java** όπου δηλώνεται ο πίνακας με τα διαφορετικά N (εδώ N = {100, 500, 1000} και από όπου καλείται η παραπάνω μέθοδος experiment(). Η experiment() επιστρέφει στην κλάση Experiment τον δισδιάστατο πίνακα με τα αποτελέσματα και τα εμφανίζει. Παραθέτουμε μερικά στιγμιότυπα από την εκτέλεση του πειράματος:





Παρατηρούμε σε κάθε περίπτωση πως ο αλγόριθμος 2, Greedy-Decreasing απαιτεί σημαντικά λιγότερους Δίσκους Αποθήκευσης σε σχέση με τον αλγόριθμο 1, ως αποτέλεσμα της ταξινόμησης κατά φθίνουσα σειρά των μεγεθών των φακέλων που θέλουμε να αποθηκεύσουμε στους δίσκους.

Συγκεντρωτικά:

	α/α	Αλγόριθμος 1	Αλγόριθμος 2
1Η δοκιμή	N = 100	60.5	54.2
	N = 500	289.2	251.5
	N = 1000	589.1	511.7
2Η δοκιμή	N = 100	59.3	53.4
	N = 500	297.2	258.6
	N = 1000	583.0	504.8
3Η δοκιμή	N = 100	59.4	53.2
	N = 500	294.7	257.9
	N = 1000	580.9	504.1