

Οικονομικό Πανεπιστήμιο Αθηνών, Τμήμα Πληροφορικής  
Μάθημα: Δομές Δεδομένων  
Ακαδημαϊκό έτος: 2019–20  
Μέλη: Ελένη Σαζώνη-3160153, Αριστείδης Χρονόπουλος-3160194

## Εργασία 2

### Μέρος Α:

Για το πρώτο μέρος τις εργασίας, όπου έπρεπε να υλοποιήσουμε μια ουρά προτεραιότητας με τη χρήση σωρού, κρατήσαμε την υλοποίηση του εργαστηρίου. Η υλοποίηση καλύπτει τις παρακάτω απαραίτητες μεθόδους με την χρήση διάφορων βοηθητικών μεθόδων.

`public void add(Disk item)`: προσθέτει αντικείμενα τύπου Disk στην ουρά προτεραιότητας

`public Disk peek()`: ανακτά το στοιχείο με τη μεγαλύτερη προτεραιότητα, χωρίς να το αφαιρεί

`public Disk getMax()`: αφαιρεί και επιστρέφει το στοιχείο με μεγαλύτερη προτεραιότητα

### Μέρος Β:

[\(Greedy.java\)](#) > [java Greedy folders.txt](#)

Στο δεύτερο μέρος της εργασίας, για την υλοποίηση του άπληστου αλγορίθμου της εκφώνησης, χρησιμοποιήσαμε την ουρά προτεραιότητας του πρώτου μέρους. Πιο συγκεκριμένα, η υλοποίηση βρίσκεται στην κλάση Greedy.java και κάνει χρήση της κλάσης Disk.java και MaxPQ του πρώτου μέρους. Αρχικά, διαβάζουμε τα αρχεία προς αποθήκευση από το αρχείο “folders.txt” και τους αποθηκεύουμε σε έναν πίνακα folders. Ξεκινώντας από το πρώτο αρχείο, το αποθηκεύουμε στον πρώτο δίσκο και ύστερα κάθε δίσκος που δημιουργείται μπαίνει σε μια ουρά προτεραιότητας, με την μεγαλύτερη προτεραιότητα να έχει ο δίσκος με τον περισσότερο ελεύθερο χώρο. Για να ελέγχουμε τον ελεύθερο χώρο κάθε δίσκου χρησιμοποιούμε την μέθοδο getFreeSpace() της κλάσης Disk ενώ για να πάρουμε τον δίσκο με την μεγαλύτερη προτεραιότητα, δηλαδή τον περισσότερο ελεύθερο χώρο, χρησιμοποιούμε την getMax() μέθοδο της MaxPQ. Ουσιαστικά κάθε φορά που θέλουμε να προσθέσουμε κάποιο αρχείο, ανακτούμε τον δίσκο με την μεγαλύτερη χωρητικότητα και αν δεν χωράει στον συγκεκριμένο δημιουργούμε έναν καινούριο δίσκο και τον προσθέτουμε στην ουρά προτεραιότητας.

Όσον αφορά γενικότερα την κλάση Greedy, αποτελείται από δύο μεθόδους την readInputData(), η οποία διαβάζει τα δεδομένα μας από αρχείο και την Greedy1(int[] folders) η οποία δέχεται σαν όρισμα τα δεδομένα που διαβάσαμε και αποθηκεύσαμε σε πίνακα και υλοποιεί τον άπληστο αλγόριθμο που περιγράψαμε παραπάνω.

### Μέρος Γ:

[\(Sort.java\)](#) > [java Sort folders.txt](#)

Για το τρίτο μέρος της εργασίας κληθήκαμε να υλοποιήσουμε έναν αλγόριθμο ταξινόμησης, προκειμένου να βελτιστοποιηθεί η αποθήκευση των αρχείων σε δίσκους, αυτή τη φορά με ταξινομημένα τα δεδομένα μας. Για τον λόγο αυτό, διαλέξαμε τον αλγόριθμο με συγχώνευση - Mergesort. Ο αλγόριθμος Mergesort είναι ένας αλγόριθμος Διαίρει και Βασίλευε και βασίζεται σε αναδρομικές κλήσεις. Ο αλγόριθμος βασίζεται στην υπόθεση ότι ο πίνακας μήκους ένα είναι ταξινομημένος. Επομένως διαιρεί συνεχώς τον δεδομένο πίνακα σε υποπίνακες έως ότου φτάσει σε μεμονωμένα στοιχεία. Στη συνέχεια συγχωνεύει τους υποπίνακες επαναληπτικά μέχρι να φτάσει στο επιθυμητό αποτέλεσμα, το οποίο είναι ο τελικός ταξινομημένος πίνακας. Πιο συγκεκριμένα στην δική μας υλοποίηση στην κλάση Sort.java έχουμε δύο μεθόδους την merge και την sort. Η sort δέχεται σαν είσοδο έναν πίνακα και δύο δείκτες left και right, οι οποίοι υποδεικνύουν τα δύο άκρα του πίνακα. Με αναδρομικές κλήσεις διαχωρίζει τον πίνακα σε μικρότερους υποπίνακες και κάνοντας χρήση της μεθόδου merge συγχωνεύει τους υποπίνακες, δίνοντας τον τελικό ταξινομημένο.

### Μέρος Δ:

[\(Experiment.java\)](#) > [java Experiment](#)

Στο τέταρτο και τελευταίο μέρος της εργασίας, προκειμένου να διαπιστώσουμε ποιος αλγόριθμος είναι καλύτερος, φτιάξαμε τυχαία δεδομένα εισόδου για διάφορες τιμές πλήθους αρχείων, των οποίων το μέγεθος είναι από 0 έως 1000000 TB. Συνολικά δημιουργήσαμε 10 txt αρχεία για πλήθος φακέλων N = 100, 10 txt αρχεία για πλήθος φακέλων N = 500 και 10 txt αρχεία για πλήθος αρχείων N = 1000.

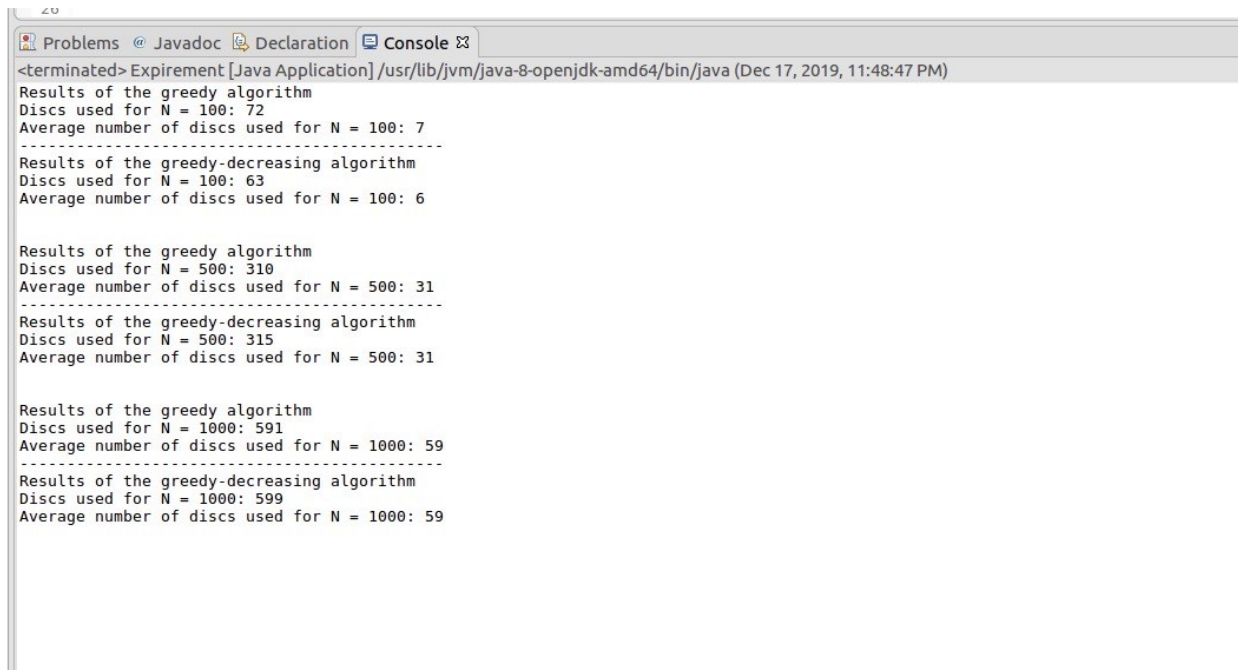
Για την διαδικασία αυτή, φτιάξαμε την μέθοδο createFile(int lines, String name) η οποία δέχεται ως είσοδο των αριθμό των φακέλων, δηλαδή τις γραμμές που θα έχει το txt αρχείο και ένα όνομα για το αρχείο που θα δημιουργηθεί. Για την τυχαία δημιουργία των φακέλων χρησιμοποιούμε την κλάση Random του πακέτου java.util και συγκεκριμένα με την .nextInt(1000) παράγουμε τυχαίους αριθμούς από το 0 έως το 1000 και παράλληλα τα γράφουμε στο αρχείο μας.

Για να τρέξουμε τους αλγορίθμους από τα προηγούμενα ερωτήματα, δημιουργήσαμε δύο μεθόδους runAlgorithm1( ) και runAlgorithm2( ). Τα τρέχουμε ξεχωριστά για κάθε τιμή του N.

### Συμπεράσματα υλοποίησης:

Αρχικά η υλοποίηση με ταξινόμηση φαίνεται να βγάζει καλύτερα αποτελέσματα, όμως καθώς το N αυξάνεται παρατηρούμε ότι είναι καλύτερος ο άπληστος αλγόριθμος χωρίς τον ταξινομημένο πίνακα.

### Ενδεικτικά αποτελέσματα:



```
<terminated> Expiration [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Dec 17, 2019, 11:48:47 PM)
Results of the greedy algorithm
Discs used for N = 100: 72
Average number of discs used for N = 100: 7
-----
Results of the greedy-decreasing algorithm
Discs used for N = 100: 63
Average number of discs used for N = 100: 6

Results of the greedy algorithm
Discs used for N = 500: 310
Average number of discs used for N = 500: 31
-----
Results of the greedy-decreasing algorithm
Discs used for N = 500: 315
Average number of discs used for N = 500: 31

Results of the greedy algorithm
Discs used for N = 1000: 591
Average number of discs used for N = 1000: 59
-----
Results of the greedy-decreasing algorithm
Discs used for N = 1000: 599
Average number of discs used for N = 1000: 59
```