ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τμήμα Μηχανικών Πληροφορικής ΑΤΕΙΘ

ΒΕΛΤΙΩΣΗ ΚΑΙ ΕΞΕΛΙΞΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΑΝΑΘΕΣΗΣ ΠΤΥΧΙΑΚΩΝ ΕΡΓΑΣΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΣΣΑΛΟΝΙΚΗ, ΦΕΒΡΟΥΑΡΙΟΣ 2019

Του φοιτητή
Αντωνίου Αριστοτέλη
Αρ. Μητρώου: 04/2639

Επιβλέπων Καθηγητής
Κ. Αμανατιάδης Δημήτριος

Πρόλογος

Με αφορμή το γεγονός ότι ο υπάρχον αλγόριθμος παρουσίασε σφάλμα κατά την διάρκεια ανάθεσης ομαδικών πτυχιακών εργασιών, κρίθηκε σκόπιμη ή υλοποίηση ενός νέου προγράμματος το οποίο όχι μόνο θα διόρθωνε τυχόν ατέλειες του προηγούμενου, αλλά θα βελτίωνε και θα εξέλυσε τον αλγόριθμο ώστε να είναι πιο αποδοτικός, πιο γρήγορος και θα ανταποκρίνεται στις νέες απαιτήσεις της Επιτροπής Πτυχιακών Εργασιών.

Στόχος της παρούσας πτυχιακής εργασίας είναι η υλοποίηση ενός προγράμματος το οποίο όταν εκτελείται, θα συνδέεται αυτόματα με την ήδη υπάρχουσα Βάση Δεδομένων, θα αντλεί τα δεδομένα που χρειάζεται χωρίς να την επηρεάζει, και αφού δημιουργήσει τοπικά αντίγραφα, θα επιστρέφει ως έξοδο αρχεία CSV τα οποία θα περιέχουν όχι μόνο τα αποτελέσματα, αλλά μια πληθώρα από χρήσιμα δεδομένα.

Περίληψη

Ένα πρόγραμμα αυτοματοποιημένης ανάθεσης πτυχιακών εργασιών θα πρέπει να καλύπτει πλήρως τις απαιτήσεις και τα κριτήρια της Επιτροπής Ανάθεσης Πτυχιακών Εργασιών. Επειδή στο υπάρχων σύστημα εντοπίστηκαν λάθη, κρίθηκε σκόπιμη η εκ νέου δημιουργία του, αξιοποιώντας την εξέλιξη της τεχνολογίας. Με την χρήση μιας σύγχρονης γλώσσας προγραμματισμού, σε συνδυασμό με νεότερες αρχές υλοποίησης, ο παρών αλγόριθμος εξασφαλίζει την ταχύτερη και ασφαλέστερη εκτέλεση του προγράμματος και την ορθότητα των αποτελεσμάτων.

Abstract

The program used by the Thesis Assignment Committee should fully meet the Committee's requirements and criteria. Due to errors in the existing system, it was deemed necessary to re-implement it and to take advantage of technology's evolution. By using a modern programming language, combined with newer implementation principles, the current algorithm ensures a faster and safer program execution, while also guaranteeing a more robust thesis distribution.

Περιεχόμενα

Εισαγωγή

Το παρών έγγραφο είναι ένα εγχειρίδιο χρήσης, σχεδιασμένο να συνοδεύει το πρόγραμμα αυτοματοποιημένης ανάθεσης πτυχιακών εργασιών που υλοποιήθηκε στα πλαίσια πτυχιακής εργασίας για το τμήμα Μηχανικών Πληροφορικής του Αλεξάνδριου Τεχνολογικού Εκπαιδευτικού Ιδρύματος Θεσσαλονίκης. Το πρόγραμμα φτιάχτηκε με σκοπό την χρήση του από την Επιτροπή Ανάθεσης Πτυχιακών Εργασιών και το παρών έγγραφο στοχεύει στο να επεξηγηθούν όσο καλύτερα γίνεται η σκέψη, οι αρχές, και οι τεχνολογίες που χρησιμοποιήθηκαν και διέπουν το πρόγραμμα ανάθεσης πτυχιακών εργασιών.

Στα κεφάλαια που ακολουθούν θα γίνει μια αναφορά τόσο στα θεωρητικά όσο και τεχνικά πλαίσια που αφορούν την υλοποίηση και λογική του προγράμματος. Σε θεωρητικό επίπεδο θα αναλυθούν οι απαιτήσεις, οι περιορισμοί, το περιβάλλον και η διαδικασία λήψης των αποφάσεων του προγράμματος. Σε τεχνικό επίπεδο θα γίνει περιγραφή της γλώσσας προγραμματισμού και του Ολοκληρωμένου Περιβάλλοντος Υλοποίησης που χρησιμοποιήθηκαν και επεξήγηση των τεχνολογιών, μεθοδολογιών και έτοιμων βιβλιοθηκών που χρησιμοποιήθηκαν.

Στο κεφάλαιο «Απαιτήσεις Αλγορίθμου και Υπάρχουσα κατάσταση» θα γίνει μια αναφορά τόσο στους λόγους που οδήγησαν στην ανάγκη βελτίωσης του αλγορίθμου ανάθεσης πτυχιακών εργασιών, όσο και στις βασικές απαιτήσεις της νέας υλοποίησης.

Στο κεφάλαιο «Περιβάλλον υλοποίησης» παρουσιάζονται τα βασικά στοιχεία της γλώσσας προγραμματισμού και των εργαλείων υλοποίησης που χρησιμοποιήθηκαν.

Ακολούθως στο κεφάλαιο «Στάδια ανάπτυξης» θα γίνει μια εκτενή αναφορά στη διαδικασία ανάπτυξης του αλγορίθμου, στα στάδια υλοποίησης, στις κεντρικές ιδέες και στα προβλήματα που αντιμετωπίστηκαν.

Τελευταίο κεφάλαιο είναι το «Ανάλυση Αλγορίθμου». Σε αυτό το κεφάλαιο θα γίνει διεξοδική ανάλυση του αλγορίθμου και επεξήγηση των διαδικασιών αποφάσεων. Σε αυτό το κεφάλαιο γίνεται και παραπομπή σε σημεία του κώδικα τα οποία παρατίθενται στο τέλος του εγγράφου.

Κεφάλαιο 1. Απαιτήσεις Αλγορίθμου και Υπάρχουσα κατάσταση

1.1. Εισαγωγή

Το τμήμα Πληροφορικής διαθέτει ήδη ένα πρόγραμμα για την αυτόματη ανάθεση πτυχιακών εργασιών, το οποίο είχε υλοποιηθεί από έναν φοιτητή στα πλαίσια της πτυχιακής του εργασίας του το 2014. Το πρόγραμμα αυτό όμως παρουσίαζε ένα σημαντικό σφάλμα κατά την ανάθεση ομαδικών εργασιών, με αποτέλεσμα να αναθέτει το ίδιο θέμα σε πάνω από μια ομάδα φοιτητών. Το γεγονός αυτό καθιστούσε το πρόγραμμα αυτό αναξιόπιστο και όταν εντοπίστηκε από την Επιτροπή Ανάθεσης Πτυχιακών Εργασιών κρίθηκε απαραίτητη η δημιουργία ενός νέου προγράμματος ανάθεσης πτυχιακών.

1.2. Βασικοί στόχοι του νέου αλγορίθμου

Με αφορμή το παραπάνω γεγονός κρίθηκε σκόπιμη η εξέλιξη του αλγορίθμου ώστε όχι μόνο να διορθώνει το προαναφερθέν πρόβλημα, αλλά και να καλύπτει νέες απαιτήσεις που είχαν παρουσιαστεί τα τελευταία χρόνια. Πιο συγκεκριμένα οι βασικοί στόχοι του νέου αλγορίθμου είναι οι εξής:

- Σωστή ανάθεση ομαδικών εργασιών.

Ο παραπάνω στόχος είναι αυτονόητος. Κάθε θέμα πτυχιακής εργασίας θα πρέπει να δίνεται σε έναν φοιτητή/ομάδα φοιτητών.

- Ικανότητα εκτέλεσης του αλγορίθμου και χωρίς σύνδεση στη Βάση Δεδομένων.

Αν και η πιθανότητα να μην μπορεί να επιτευχθεί σύνδεση με την Βάση Δεδομένων είναι μικρή (έλλειψη σύνδεσης στο διαδίκτυο ή ο διακομιστής να μην λειτουργεί), κρίθηκε σκόπιμο να καλυφθεί αυτή η πιθανότητα, ώστε η ανάθεση πτυχιακών να μην καθυστερεί.

- Ελαχιστοποίηση της επικοινωνίας προγράμματος – Βάσης Δεδομένων.

Η συνεχής επικοινωνία του προγράμματος με την βάση δεδομένων όχι μόνο καθυστερεί την εκτέλεση του προγράμματος και επιβαρύνει τον διακομιστή, αλλά και μεγιστοποιεί την πιθανότητα σφάλματος λόγο ασταθής σύνδεσης.

- Το πρόγραμμα να μην επηρεάζει την Βάση Δεδομένων.

Για να εξασφαλιστεί μεγαλύτερη ασφάλεια κρίθηκε σκόπιμο το πρόγραμμα να μην κάνει καμία αλλαγή στη βάση δεδομένων. Αυτό εξαλείφει οποιαδήποτε πιθανότητα αλλοίωσης της βάσης δεδομένων και της ανάγκης δημιουργίας αντίγραφών ασφαλείας πριν την εκτέλεση του προγράμματος όπως γινόταν μέχρι τώρα.

- Δικαιότερο μοίρασμα πτυχιακών εργασιών.

Ένα πάρα πολύ σημαντικό κομμάτι του αλγορίθμου είναι η λογική με την οποία γίνεται η ανάθεση των πτυχιακών εργασιών. Υπάρχουν τρία βασικά κριτήρια τα οποία πρέπει να τηρηθούν. Τα κριτήρια αυτά είναι: να δοθούν όσο το δυνατόν περισσότερα θέματα, να γίνουν σεβαστές οι προτιμήσεις του κάθε φοιτητή και να δοθεί προτεραιότητα στους καλύτερους φοιτητές. Όπως είναι εύκολα αντιληπτό, τα κριτήρια αυτά είναι αντικρουόμενα μεταξύ τους και για να γίνει δικαιότερο το μοίρασμα θα πρέπει να βρεθεί μια χρυσή τομή ανάμεσα στα τρία.

Το ήδη υπάρχον πρόγραμμα είναι γραμμένο σε PHP και MySQL. Καθ’ όλη την διάρκεια εκτέλεσης του αλγορίθμου υπάρχει επικοινωνία με την Βάση Δεδομένων, όχι μόνο για να αντλήσει πληροφορίες (SELECT), αλλά και για να κάνει αλλαγές στην Βάση Δεδομένων, ακόμη, ο τρόπος με τον οποίο αναθέτει πτυχιακές δεν είναι ο δικαιότερος δυνατόν. Λαμβάνοντας όλα τα παραπάνω υπ’ όψη ήταν εμφανές ότι ο βέλτιστος τρόπος προσέγγισης αυτής της πτυχιακής θα ήταν η υλοποίηση ενός νέου προγράμματος από την αρχή, χρησιμοποιώντας διαφορετική γλώσσα προγραμματισμού και όχι η προσπάθεια τροποποίησης ή βελτίωσης του ήδη υπάρχοντος κώδικα.

Τέλος είναι απαραίτητο να γίνει εδώ μια αναφορά στην υπάρχουσα Βάση Δεδομένων. Για την υλοποίηση αυτής της πτυχιακής προϋπόθεση ήταν η χρήση της βάσης δεδομένων όπως ήταν, χωρίς μόνιμες τροποποιήσεις ή αλλαγές πινάκων και δεδομένων. Στα επόμενα κεφάλαια θα γίνει αναφορά σε κάποια προβλήματα που αυτό δημιούργησε, όπως και ιδέες για την μελλοντική αντιμετώπιση τους.

Κεφάλαιο 2. Περιβάλλον υλοποίησης

2.1. Εισαγωγή

Το πρόγραμμα που υλοποιήθηκε για της ανάγκες αυτής της πτυχιακής εργασίας έχει γραφτεί εξολοκλήρου σε Java 8 με πλατφόρμα υλοποίησης το IntelliJ. Παρακάτω θα γίνει ξεχωριστή αναφορά στην πλατφόρμα υλοποίησης, στην γλώσσα προγραμματισμού αλλά και στις δυο έτοιμες βιβλιοθήκες που χρησιμοποιήθηκαν για την ανάκτηση και αποθήκευση δεδομένων.

2.2. Ιστορία της Java

Στις αρχές του 1991, η Sun αναζητούσε το κατάλληλο εργαλείο για να αποτελέσει την πλατφόρμα ανάπτυξης λογισμικού σε μικρο-συσκευές.[1] Τα εργαλεία της εποχής ήταν γλώσσες όπως η C++ και η C. Μετά από διάφορους πειραματισμούς προέκυψε το συμπέρασμα ότι οι υπάρχουσες γλώσσες δεν μπορούσαν να καλύψουν τις ανάγκες τους. Ο "πατέρας" της Java, James Gosling, που εργαζόταν εκείνη την εποχή για την Sun, έκανε ήδη πειραματισμούς πάνω στη C++ και είχε παρουσιάσει κατά καιρούς κάποιες πειραματικές γλώσσες (C++ ++, που μετέπειτα ονομάστηκε C#) ως πρότυπα για το νέο εργαλείο που αναζητούσαν στην Sun. Τελικά μετά από λίγο καιρό κατέληξαν με μια πρόταση για το επιτελείο της εταιρίας, η οποία ήταν η γλώσσα Java.

Η Java ήταν μία γλώσσα που διατηρούσε μεγάλη συγγένεια με την C++. Παρόλα αυτά είχε πολύ πιο έντονο αντικειμενοστρεφή (object oriented) χαρακτήρα σε σχέση με την C++ και χαρακτηριζόταν για την απλότητα της. Η επίσημη εμφάνιση της Java στη βιομηχανία της πληροφορικής έγινε το Μάρτιο του 1995 όταν η Sun την ανακοίνωσε στο συνέδριο Sun World 1995. Ο πρώτος μεταγλωττιστής (compiler) της ήταν γραμμένος στη γλώσσα C από τον James Gosling.

Στις 13 Νοεμβρίου του 2006 η Java έγινε πλέον μια γλώσσα ανοιχτού κώδικα και στις 27 Απριλίου 2010 η εταιρία λογισμικού Oracle Corporation ανακοίνωσε ότι μετά από πολύμηνες συζητήσεις ήρθε σε συμφωνία για την εξαγορά της Sun Microsystems και των τεχνολογιών (πνευματικά δικαιώματα/ πατέντες) που η δεύτερη

[1] https://el.wikipedia.org/wiki/Java

είχε στην κατοχή της ή δημιουργήσει. Η συγκεκριμένη συμφωνία θεωρείται σημαντική για το μέλλον της Java και του γενικότερου οικοσυστήματος τεχνολογιών γύρω από αυτή μιας και ο έμμεσος έλεγχος της τεχνολογίας και η εξέλιξη της περνάει σε άλλα χέρια.

2.3. Χαρακτηριστικά της Java

Η Java είναι δύο πράγματα: γλώσσα προγραμματισμού και πλατφόρμα. Η Java σαν γλώσσα προγραμματισμού χαρακτηρίζεται από τις παρακάτω έννοιες.[2]

- Απλή

Στόχος της ομάδας της Sun που ανέπτυξε την Java, ήταν μια γλώσσα εύκολη στην χρήση, που δεν απαιτεί πολλή εξάσκηση και εκπαίδευση. Οι περισσότεροι προγραμματιστές στις μέρες μας δουλεύουν είτε με την C είτε με την C++. Έτσι, μολονότι η C++ δεν ήταν η κατάλληλη για το αρχικό σχέδιο, η Java σχεδιάστηκε βάσει της C++, με σκοπό να γίνει όσο το δυνατόν περισσότερο κατανοητή.

Η Java παραλείπει πολλά από τα σπανίως χρησιμοποιούμενα και δυσκολονόητα χαρακτηριστικά της C++, που δεν ωφελούν και πολύ την ευελιξία της γλώσσας. Προστέθηκαν διεργασίες, όπως η αυτόματη συλλογή των "σκουπιδιών" (automatic garbage collection), διευκολύνοντας τον προγραμματισμό σε Java. Μια κοινή πηγή πολυπλοκότητας της C++ και της C είναι η διαχείριση της

[2] http://www.islab.demokritos.gr/gr/html/ptixiakes/kostas-aris_ptyxiakh/Phtml/java.htm

μνήμης. Με την καινούργια διεργασία της αυτόματης συλλογής "σκουπιδιών", που συνιστάται από την περιοδική αποδέσμευση της μνήμης που δεν χρησιμοποιείται, μεγάλο μέρος από την δουλεία των προγραμματιστών αυτοματοποιείται και μειώνονται τα bugs.

Ένα πλεονέκτημα της Java που οφείλεται στην απλότητα της είναι ότι το μέγεθος των απαραίτητων εργαλείων. Ο Java interpreter και οι βασικές βιβλιοθήκες είναι μικρές και ο κώδικάς της Java είναι τόσο περιορισμένος σε μέγεθος που μπορεί άνετα να τρέξει σε οποιαδήποτε μικρή μηχανή και να κατέβει από το δίκτυο.

- Αντικειμενοστρεφής

Λέγοντας ότι μία γλώσσα προγραμματισμού είναι αντικειμενοστρεφής, εννοούμε η τεχνική σχεδιασμού ενός προγράμματος συγκεντρώνεται σε αντικείμενα. Ένα αντικείμενο είναι ο συνδυασμός δεδομένων, διαδικασιών και λειτουργιών με βασική ιδιότητα την απόκρυψη του συνδυασμού αυτού. Το κάθε αντικείμενο, δηλαδή, αντιμετωπίζεται σαν ένα "μαύρο κουτί". Τα αντικείμενα δεν είναι ανεξάρτητα μεταξύ τους, αλλά βρίσκονται σε σχέση αλληλεξάρτησης με τα υπόλοιπα. Υπάρχει η έννοια της κληρονομικότητας μεταξύ των αντικειμένων, δηλαδή ένα αντικείμενο μπορεί να κληρονομήσει δεδομένα από άλλα αντικείμενα.

Οι γλώσσες αντικειμενοστρεφή προγραμματισμού είναι γλώσσες υψηλού επιπέδου, αφαιρετικές, αποτελεσματικές, γρήγορες και χρησιμοποιούνται για την δημιουργία μεγάλων και σημαντικών εφαρμογών. Οι αντικειμενοστρεφής ευκολίες της Java είναι ίδιες με αυτές της C++, με επεκτάσεις από την Objective C.

- Συμβατή με Δίκτυα

Η Java έχει μια μεγάλη βιβλιοθήκη από ρουτίνες για την επιτυχημένη συνεργασία με τα πρωτόκολλα HTTP και FTP. Κατ’ αυτόν τον τρόπο, οι δικτυακές συνδέσεις δημιουργούνται ευκολότερα από ότι με την C ή την C++. Τα προγράμματα σε Java μπορούν να έχουν πρόσβαση μέσω δικτύου σε αντικείμενα , με την ίδια άνεση που ένας χρήστης προσπελάυνει ένα τοπικό σύστημα αρχείων.

- Σταθερή

Η Java προορίζεται για την σύνταξη προγραμμάτων που θα είναι αξιόπιστα από όλες τις πλευρές. Δίνεται έμφαση στον από νωρίς έλεγχο για πιθανά προβλήματα και στον έλεγχο σε πραγματικό χρόνο και στην εξάλειψη καταστάσεων που προκαλούν λάθη.

Η μεγαλύτερη διαφορά μεταξύ Java και C/C++ είναι το γεγονός ότι η Java έχει ένα μοντέλο δεικτών που εξαφανίζει την πιθανότητα της επαναχρησιμοποίησης της μνήμης και την καταστροφή των δεδομένων. Αντί για αριθμητικούς δείκτες (pointer arithmetic), η Java έχει πραγματικούς πίνακες (true arrays). Οι προγραμματιστές της Java δεν έχουν να φοβηθούν την ακούσια (ή μη) τροποποίηση της μνήμης, γιατί δεν υπάρχουν δείκτες (pointers). Εξάλλου, τα προγράμματα σε Java δεν μπορούν να αποκτήσουν μη εγκεκριμένη πρόσβαση στην μνήμη.

- Ασφαλής

Η Java προορίζεται για χρήση σε ανοικτά, δικτυωμένα περιβάλλοντα. Γι' αυτό το λόγο, ιδιαίτερη προσοχή έχει δοθεί στην ασφάλεια που παρέχει η γλώσσα. Η Java επιτρέπει την κατασκευή προγραμμάτων ελεύθερων από ιούς και η τροποποίηση τους είναι αδύνατη. Οι τεχνικές πιστοποίησης ταυτότητας βασίζονται στην ασύμμετρη κρυπτογραφία.

Υπάρχει μεγάλη σχέση μεταξύ του τρόπου διαχείρισης της μνήμης και της παρεχόμενης ασφάλειας. Αλλαγές στην σημασιολογία των δεικτών της μνήμης κάνουν αδύνατη την μη έγκυρη πρόσβαση στα δεδομένα της μνήμης ή της πρόσβασης των δεδομένων των αντικειμένων. Με αυτόν τον τρόπο καταπολεμούνται οι περισσότεροι ιοί.

- Ουδέτερη της Υποκείμενης Αρχιτεκτονικής

Η Java έχει σχεδιαστεί για να υποστηρίζει δικτυακές εφαρμογές. Ένα δίκτυο, όμως, αποτελείται από ποικιλία διαφορετικών συστημάτων, με διαφορετικές CPU και

λειτουργικά συστήματα. Για να μπορούν οι Java εφαρμογές να εκτελούνται παντού στο δίκτυο, το πρόγραμμα Java πρέπει να περάσει από δύο διαδικασίες ώστε να καταλήξει σε εκτελέσιμη μορφή. Πρώτα ο μεταγλωττιστής, μετατρέπει τον πηγαίο κώδικα του προγράμματος σε μία ενδιάμεση γλώσσα που καλείται Java bytecodes. Τα Java bytecodes είναι ανεξάρτητα της πλατφόρμας και με χρήση του ερμηνευτή (interpreter) κάθε bytecode εντολή μετατρέπεται σε κατάλληλη δυαδική μορφή για να τρέξει στον εκάστοτε υπολογιστή. Η μεταγλώττιση (compilation) συμβαίνει μόνο μια φορά για κάθε Java πρόγραμμα, η ερμηνεία (interpretation) γίνεται κάθε φορά που το πρόγραμμα εκτελείται. Η τεχνική αυτή ονομάζεται "write once, run anywhere".

Τα Java bytecodes μπορούμε να τα φανταστούμε σαν την γλώσσα μηχανής για την Java Virtual Machine (JVM). Κάθε Java ερμηνευτής (π.χ. ένας Web browser που μπορεί να τρέχει applets) είναι μια λογισμική εφαρμογή της Java Virtual Machine. Η JVM αναλαμβάνει να μετατρέψει τα bytecodes σε κατάλληλη εκτελέσιμη μορφή, ανάλογα με το υποκείμενο software και hardware.

- Φορητή

Το γεγονός ότι είναι ανεξάρτητη της υποκείμενης πλατφόρμας αποτελεί μεγάλο μέρος του ότι είναι φορητή, άλλα υπάρχουν και άλλα σημεία που χαρακτηρίζουν την φορητότητα της.

Σε αντίθεση με την C/C++ δεν υπάρχουν καθόλου χαρακτηριστικά που εξαρτούνται από την CPU του υπολογιστή. Έτσι, τα μεγέθη των πρωταρχικών τύπων δεδομένων είναι καθορισμένα και η συμπεριφορά τους είναι παντού η ίδια. Για παράδειγμα, "int" σημαίνει πάντα έναν 32 bit ακέραιο και "float" πάντα αντιπροσωπεύει έναν 32 bit floating αριθμό.

- Interpreted

Τα Java bytecodes μεταφράζονται σε πραγματικό χρόνο σε εντολές μηχανής που εξαρτώνται από την εκάστοτε πλατφόρμα, και δεν αποθηκεύονται πουθενά. Η διαδικασία είναι γρήγορη και πιο αποτελεσματική. Μαζί με τα bytecodes μεταφέρονται

πληροφορίες που μπορούν να χρησιμοποιηθούν κατά την εκτέλεση και παρέχουν την βάση για τους ελέγχους που πραγματοποιεί ο συνδετής (linker). Επίσης τα προγράμματα γίνονται πιο επιδεκτικά σε debugging διαδικασίες.

- Υψηλής Απόδοσης

Η διαδικασία παραγωγής των εντολών μηχανής είναι απλή και γρήγορη. Ο κώδικας που προκύπτει είναι αποτελεσματικός. Ο μεταγλωττιστής από την μεριά του εφαρμόζει αυτόματη κατανομή των καταχωρητών (automatic register allocation) όταν παράγει τα bytecodes. Η τελική μορφή του κώδικα (εκτελέσιμη δυαδική μορφή) είναι μικρή σε μέγεθος και ταχύτατη στην εκτέλεση.

- Multithreaded

Τα προγράμματα σε Java έχουν την δυνατότητα να αντιμετωπίζουν πολλές καταστάσεις – διαδικασίες ταυτόχρονα. Σε αντίθεση, η C και C++ είναι single-threaded γλώσσες. Τα πλεονεκτήματα του multithreading είναι η καλύτερη πραγματικού χρόνου συμπεριφορά και η καλύτερη αλληλεπιδραστική ανταπόκριση.

- Δυναμική

Η Java είναι πιο δυναμική γλώσσα από την C ή C++. Έχει αναπτυχθεί για να προσαρμοστεί σε ένα εξελισσόμενο περιβάλλον. Οι βιβλιοθήκες εργαλείων αναπτύσσονται ελεύθερα με την πρόσθεση νέων μεθόδων και μεταβλητών, χωρίς να επηρεάζονται οι ήδη υπάρχουσες εφαρμογές.

2.4. Βιβλιοθήκες που χρησιμοποιήθηκαν

Ένα ακόμη πλεονέκτημα, όχι μόνο της Java, αλλά όλων των ευρέως διαδεδομένων αντικειμενοστρεφών γλωσσών προγραμματισμού, είναι το μεγάλο πλήθος έτοιμων βιβλιοθηκών που μπορεί κάποιος να βρει για να καλύψει ένα μεγάλο εύρος αναγκών υλοποίησης. Δυο τέτοιες βιβλιοθήκες χρησιμοποιήθηκαν στην ανάπτυξη της παρούσας πτυχιακής. Οι βιβλιοθήκες αυτές είναι οι παρακάτω.

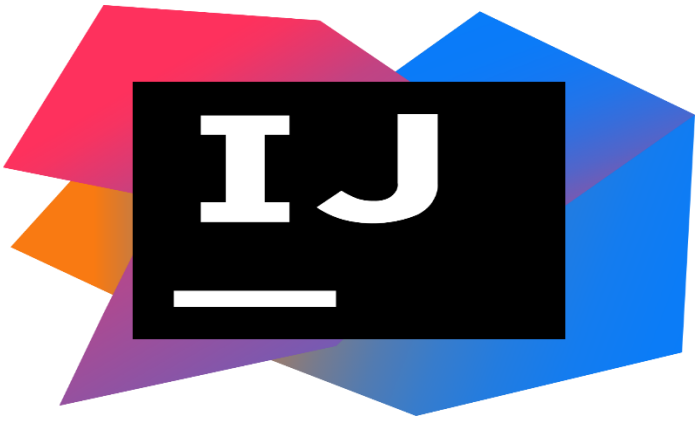
- MySQL Connector/J

Η βιβλιοθήκη αυτή είναι ο επίσημος οδηγός JDBC για MySQL. Η Java DataBase Connectivity (JDBC) είναι ένα API για τη γλώσσα προγραμματισμού Java που ορίζει τον τρόπο πρόσβασης ενός πελάτη σε μια βάση δεδομένων. Πιο συγκεκριμένα η MySQL Connector / J 8.0 είναι συμβατή με όλες τις εκδόσεις MySQL ξεκινώντας από την MySQL 5.5. Επιπλέον, η MySQL Connector / J 8.0 υποστηρίζει το νέο X DevAPI για MySQL Server 8.0.[3]

- Apache Commons CSV

Η βιβλιοθήκη αυτή καλύπτει τις ανάγκες ανάγνωσης και δημιουργίας αρχείων CSV[4]. Τα CSV χρησιμοποιούνται ευρέως ως τρόπος διασύνδεσης σε συστήματα παλαιού τύπου ή για προσωποποιημένη διαχείριση δεδομένων. Το CSV είναι ακρωνύμιο και σημαίνει "Comma Separated Values". Η μορφή δεδομένων CSV ορίζεται στο RFC 4180 αλλά υπάρχουν πολλές παραλλαγές. Κοινό σε όλες τις παραλλαγές είναι η βασική δομή τους. Η μορφή των δεδομένων στα αρχεία CSV είναι record oriented, δηλαδή κάθε νέο δεδομένο ξεκινά σε νέα γραμμή κειμένου. Κάθε δεδομένο περιέχει πολλαπλές τιμές και δεν είναι απαραίτητο όλα τα δεδομένα να έχουν τον ίδιο αριθμό τιμών.[5]

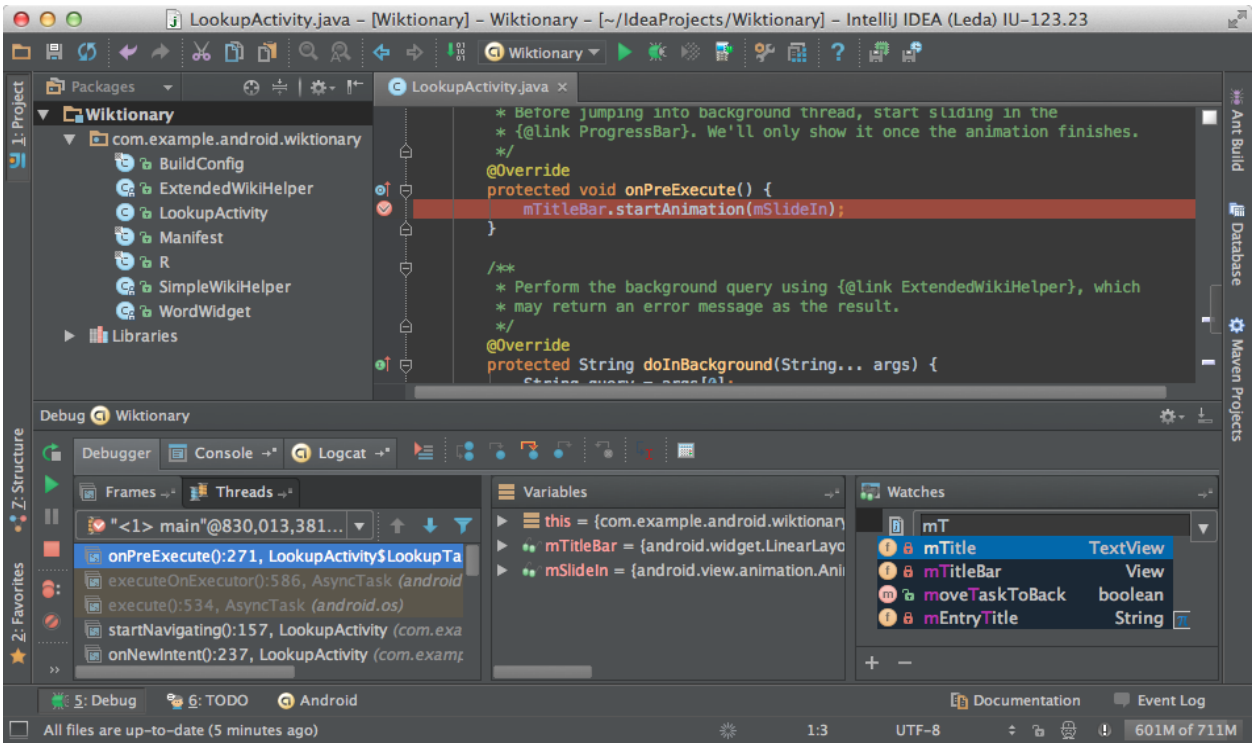
2.5. Η Πλατφόρμα υλοποίησης



[3] https://dev.mysql.com/downloads/connector/j/8.0.html
[4] https://commons.apache.org/proper/commons-csv/
[5] https://commons.apache.org/proper/commons-csv/apidocs/index.html

Η πλατφόρμα υλοποίησης που χρησιμοποιήθηκε είναι το ολοκληρωμένο περιβάλλον ανάπτυξης IntelliJ. Ένα ολοκληρωμένο περιβάλλον ανάπτυξης (integrated development environment, IDE) είναι μία σουίτα λογισμικού που βοηθάει στην ανάπτυξη προγραμμάτων υπολογιστή. Συνήθως ένα IDE περιλαμβάνει κάποιον επεξεργαστή πηγαίου κώδικα, έναν μεταγλωττιστή, εργαλεία αυτόματης παραγωγής κώδικα, αποσφαλματωτή, συνδέτη, σύστημα ελέγχου εκδόσεων και εργαλεία κατασκευής γραφικών διασυνδέσεων χρήστη για τις υπό ανάπτυξη εφαρμογές.[6] Το IntelliJ διατίθεται σε δυο εκδόσεις την Ultimate και την Community. Η Community είναι open-source και δωρεάν.

Η πρώτη έκδοση του IntelliJ IDEA κυκλοφόρησε τον Ιανουάριο του 2001 και ήταν ένα από τα πρώτα JAVA IDE με προηγμένες δυνατότητες πλοήγησης κώδικα και επανακαθορισμού κώδικα. Το 2010 στο περιοδικό InfoWorld, το IntelliJ έλαβε το υψηλότερο σκορ ανάμεσα στα τέσσερα κορυφαία εργαλεία προγραμματισμού Java: Eclipse , IntelliJ IDEA, NetBeans και JDeveloper.[7]



[6]https://el.wikipedia.org/wiki/Ολοκληρωμένο_περιβάλλον_ανάπτυξης
[7] https://en.wikipedia.org/wiki/IntelliJ_IDEA

Κεφάλαιο 3. Στάδια ανάπτυξης

3.1. Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει μια αναλυτική αναφορά στις βασικές αρχές, ιδέες και παραδοχές που αφορούν τον αλγόριθμο ανάθεσης πτυχιακών και το πώς αυτές αλλάξαν και εξελίχτηκαν κατά την διαδικασία ανάπτυξης του. Σκοπός του κεφαλαίου αυτού είναι να γίνει αντιληπτό το πώς και το γιατί επιλέχτηκαν οι συγκεκριμένες υλοποιήσεις για να καλυφτούν οι βασικοί στόχοι και ανάγκες του προγράμματος. Τέλος, θα γίνει αναφορά σε προβλήματα που υπήρξαν και το πώς αυτά ξεπεραστήκαν.

3.2. Δείκτες Αξιολόγησης

Πρωταρχική προϋπόθεση για την ανάπτυξη του αλγορίθμου ήταν να βρεθεί ένας τρόπος αξιολόγησης κάθε ενδεχόμενης ανάθεσης πτυχιακής σε έναν φοιτητή. Ένα τέτοιο πλαίσιο αξιολόγησης θα μπορούσε να χρησιμοποιηθεί όχι μόνο κατά την διάρκεια της εκτέλεσης του αλγορίθμου για την ανάθεση πτυχιακών, αλλά θα ήταν χρήσιμο και για την τελική αξιολόγηση του αλγορίθμου ώστε να αποφασιστεί η δικαιότητα αυτού.

Όπως αναφέρθηκε και στο κεφάλαιο 4 υπάρχουν τρία βασικά κριτήρια που αφορούν την δικαιότητα του αλγορίθμου. Τα κριτήρια αυτά είναι: να δοθούν όσο το δυνατόν περισσότερα θέματα, να γίνουν σεβαστές οι προτιμήσεις του κάθε φοιτητή και να δοθεί προτεραιότητα στους καλύτερους φοιτητές. Για αυτό το λόγο χρησιμοποιήθηκαν μια σειρά δεικτών για την ποσοτικοποίηση των τριών αυτών κριτηρίων. Το εύρος τιμών των δεικτών αυτών ορίστηκε από 1-10 όχι μόνο για ευκολία, αλλά και για εξισορρόπηση της δυναμικής των τριών αυτών κριτηρίων.

Για να αξιολογηθεί η ακαδημαϊκή αξία του κάθε φοιτητή χρησιμοποιήθηκαν τα έξεις δεδομένα: ο Γενικός Μέσος Ορός(Γ.Μ.Ο) του κάθε φοιτητή, ο Συντελεστής Προόδου του(Σ.Π.), το σύνολο των Διδακτικών Μονάδων(Δ.Μ.) των μαθημάτων που έχει περάσει και ο Μέσος Ορός της βαθμολογίας που έχει ο φοιτητή στα Προαπαιτούμενα Μαθήματα της κάθε πτυχιακής(Μ.Ο.Π.Μ). Ο τρόπος υπολογισμού του δείκτη αυτού φαίνεται παρακάτω.

- Σύνολο = Γ.Μ.Ο + (Σ.Π./100) + (Δ.Μ./21) + (2*Μ.Ο.Π.Μ)

Αποφασίστηκε να δοθεί μια ιδιαίτερη βαρύτητα στον Μέσο Ορό Προαπαιτούμενων Μαθήματων, γι' αυτό και διπλασιάζεται, ώστε να επιβραβευτεί η συμβατότητα του φοιτητή με το συγκεκριμένο θέμα. Αξίζει εδώ να αναφερθεί ότι αν ένας φοιτητής δεν έχει ολοκληρώσει όλα τα προαπαιτούμενα μαθήματα της πτυχιακής που έχει επιλέξει τότε αυτομάτως αποκλείεται από την διεκδίκηση του θέματος.

Για να γίνουν σεβαστές οι προτιμήσεις του κάθε φοιτητή μοναδικός δείκτης είναι ο Αριθμός Επιλογής του κάθε Θέματος για τον κάθε φοιτητή (Α.Ε.Θ.). Για αυτό το λόγο χρησιμοποιήθηκε ο παρακάτω τύπος:

- Προτίμηση = 2*(11 - Α.Ε.Θ.)

Και εδώ υπάρχει διπλασιασμός γιατί οι προτιμήσεις του κάθε φοιτητή πρέπει να γίνουν ιδιαίτερα σεβαστές από τον αλγόριθμο.

Αποτέλεσμα της χρήσεις των παραπάνω δεικτών είναι ότι κάθε πιθανή ανάθεση θέματος σε έναν φοιτητή μπορεί ευκολά να αξιολογηθεί και να συγκριθεί με ένα και μόνο Σκορ, με μεγαλύτερα Σκορ να είναι ενδεικτικό δικαιότερης ανάθεσης.

- Σκορ = Προτίμηση + Σύνολο

Το τρίτο κριτήριο, δηλαδή το να δοθούν όσο το δυνατόν περισσότερα θέματα, γίνεται να μετρηθεί μόνο μετά το πέρας του αλγορίθμου. Τότε είναι ιδιαίτερα χρήσιμο για να συγκριθούν τα αποτελέσματα διαφορετικών αλγορίθμων ανάθεσης. Κατά την διάρκεια εκτέλεσης του αλγορίθμου δεν υπάρχει κάποιος δείκτης που να αφορά αυτό το κριτήριο, αλλά πολλαπλοί έλεγχοι κατά την ρουτίνα ανάθεσης οι οποίοι θα επεξηγηθούν στο επόμενο κεφάλαιο.

3.3. Επιλογή του καλυτέρου αλγορίθμου

Η επιλογή του καταλληλότερου αλγορίθμου για την δικαιότερη ανάθεση πτυχιακών εργασιών ήταν μια ιδιαίτερα χρονοβόρα διαδικασία. Έπειτα από αρκετή σκέψη, ανάλυση και ερευνά τρεις αλγόριθμοι, ο καθένας με διαφορετική προσέγγιση,

ξεχώρισαν. Κρίθηκε λοιπόν σκόπιμη η υλοποίηση και των τριών ώστε να συγκριθούν και να αξιολογηθούν τα αποτελέσματα τους. Οι τρεις αυτοί αλγόριθμοι και οι βασικές ιδέες τους αναφέρονται συνοπτικά παρακάτω.

- Ο πρώτος αλγόριθμος έδινε ιδιαίτερη έμφαση στο να επιβραβεύει τους ακαδημαϊκά καλυτέρους φοιτητές. Ταξινομούσε τους φοιτητές σε φθίνουσα σειρά με βάση τον δείκτη «Σύνολο» και ξεκινώντας από τον πρώτο εύρισκε την όσο το δυνατόν καλύτερη προτίμηση ήταν ακόμα διαθέσιμη. Έπειτα, για να ικανοποιηθεί και το κριτήριο του πλήθους θέματών που θα δοθούν, έλεγχε ότι κανένας από τους επόμενους φοιτητές με αρκετά κοντινό «Σύνολο» δεν θα έμενε χωρίς θέμα, οπότε και έδινε το θέμα στον φοιτητή.

- Ο δεύτερος αλγόριθμος έδινε έντονη βαρύτητα στο να μοιράσει όσο το δυνατόν περισσότερα θέματα. Ταξινομούσε τα θέματα με βάση το πλήθος τον διεκδικητών και αφού έβρισκε τον καλύτερο διεκδικητή με βάση το «Σκορ» και εφόσον κάποιος άλλος φοιτητής δεν χρειαζόταν το θέμα περισσότερο του το έδινε. Τέλος αν ο φοιτητής αυτός είχε ήδη θέμα, τότε γινόταν έλεγχος στη σειρά προτίμησης και αν το παλιό θέμα είχε χαμηλότερη προτίμηση, το παλιό θέμα ανακυκλωνόταν, και ο φοιτητής έπαιρνε το καινούριο.

- Ο τρίτος αλγόριθμος ήταν ο μονός που είχε πολλαπλά περάσματα. Για κάθε σύνολο θεμάτων - φοιτητών ο αλγόριθμος έτρεχε πολλές φορές (5000), ανακατεύοντας την λίστα των φοιτητών με τυχαίο τρόπο. Αναμενόμενο αποτέλεσμα των πολλαπλών περασμάτων ήταν ο αλγόριθμος αυτός να είναι ο πιο αργός και να καταναλώνει μεγαλύτερη μνήμη Έπειτα, από το σύνολο των αποτελεσμάτων επέλεγε το καλύτερο το οποίο και επέστρεφε.(Αναλυτική ανάλυση του παρών αλγορίθμου ακολουθεί στο κεφάλαιο 4)

Για να αξιολογηθούν τα αποτελέσματα των αλγορίθμων αυτών επιλέχθηκαν τα έξης κριτήρια: το πλήθος των θεμάτων που κάθε αλγόριθμος ανέθετε, το άθροισμα των «Σκορ» για την κάθε ανάθεση θέματος, ο μέσος ορός των προτιμήσεων που δόθηκε από κάθε αλγόριθμο και ο χρόνος εκτέλεσης. Ακόμη, φτιάχτηκε μια μέθοδος η οποία δημιουργούσε τυχαίο πλήθος θεμάτων, φοιτητών και επιλογών μέσα από

προκαθορισμένο εύρος τιμών, για να υπάρχει μεγάλο πλήθος δειγμάτων για αξιολόγηση.

Αποτέλεσμα των παραπάνω δοκιμών ήταν ότι ο τρίτος αλγόριθμος είχε συντριπτικά καλύτερα αποτελέσματα, ενώ η διαφορά στον χρόνο εκτέλεσης ήταν αμελητέα. Συνεπώς αποφασίστηκε η πτυχιακή αυτή να χρησιμοποιήσει τον τρίτο αλγόριθμο για την υλοποίηση του προγράμματος.

3.4. Η βάση δεδομένων

Για την σύνδεση με την βάση δεδομένων μονόδρομος ήταν η χρήση της βιβλιοθήκης MySQL Connector/J. Για να καλυφθούν οι απαιτήσεις επικοινωνίας προγράμματος – βάσης δεδομένων, όπως αυτές περιεγράφηκαν στο κεφάλαιο 4, κρίθηκε απαραίτητο να τηρηθούν οι παρακάτω κανόνες υλοποίησης.

- Οποιαδήποτε επικοινωνία του προγράμματος με την βάση δεδομένων για άντληση πληροφοριών θα γινόταν μόνο στην αρχή του προγράμματος. Όλα τα απαραίτητα δεδομένα από κάθε πίνακα θα μετατρεπόταν σε λίστες αντικείμενων, για μεταγενέστερη χρήση, ώστε η περεταίρω επικοινωνία με την βάση δεδομένων να μην είναι απαραίτητη.

- Αμέσως μετά την άντληση των απαραίτητων δεδομένων, η αποθήκευση τους τοπικά, υπό την μορφή αρχείων CSV, εξασφαλίζει την ικανότητα του προγράμματος να τρέξει χωρίς να χρειάζεται να συνδεθεί με την βάση δεδομένων.

- Τα αποτελέσματα του αλγορίθμου θα αποθηκεύονται μόνο τοπικά, χωρίς να γίνει καμία εισαγωγή ή αλλαγή στην βάση δεδομένων.

Σε αυτό το σημείο αξίζει να αναφερθεί ένα πρόβλημα το οποίο εντοπίστηκε και αφορά την βάση δεδομένων. Μετά τις αλλαγές των προγραμμάτων σπουδών και εν όψη της επερχόμενης, η έλλειψη ενός ολοκληρωμένου πίνακα συσχετίσεων μαθήματων δημιουργεί δυσκολίες στην εύρεση των βαθμών ενός φοιτητή για τα προαπαιτούμενα μαθήματα της κάθε πτυχιακής. Αυτό έχει σαν αποτέλεσμα φοιτητές

οι οποίοι έχουν περάσει προαπαιτούμενα μαθήματα σε προηγούμενο πρόγραμμα σπουδών να αποκλείονται από πτυχιακές εργασίες τις οποίες δικαιούνται.

Μια εύκολη λύση για αυτό το πρόβλημα, πέρα από τα πλαίσια αυτής της πτυχιακής, είναι η δημιουργία ενός πίνακα συσχετίσεων στη βάση δεδομένων ο οποίος θα περιέχει έναν νέο δευτερεύον κωδικό συσχετίσεις, κοινός στα αντίστοιχα μαθήματα διάφορων προγραμμάτων σπουδών. Ο παρών αλγόριθμος ανάθεσης πτυχιακών μπορεί πολύ ευκολά να προσαρμοστεί στην ύπαρξη ενός τέτοιου πίνακα.

3.5. Ομαδικές πτυχιακές

Η ύπαρξη ομαδικών πτυχιακών προσθέτει μια περαιτέρω επιπλοκή στον αλγόριθμο ανάθεσης πτυχιακών. Για να αντιμετωπιστεί καταλληλά η επιπλοκή αυτή οριστήκαν τα παρακάτω πλαίσια.

- Για κάθε ζευγάρι φοιτητών που επιθυμούν να δηλώσουν μια ομαδική πτυχιακή εργασία, μόνο ο ένας από αυτούς την δηλώνει, μαζί με τον αριθμό μητρώου του δευτέρου φοιτητή.
- Κάθε ζεύγος φοιτητών αντιμετωπίζεται ως μια οντότητα. Γίνεται συμψηφισμός των δεικτών αξιολόγησης και των δυο φοιτητών και υπολογίζεται ο μέσος ορός. Ακόμη, αν έστω ένας από τους δυο φοιτητές δεν έχει ολοκληρώσει κάποιο από τα προαπαιτούμενα μαθήματα της εκάστοτε πτυχιακής εργασίας, αποκλείονται και οι δυο.

Ο ήδη υπάρχον αλγόριθμος ανάθεσης πτυχιακών εργασιών μοίραζε τις ομαδικές πτυχιακές πρώτα και τις ατομικές σε δεύτερο χρόνο. Αυτό έδειχνε μια βαρύτητα στις ομαδικές πτυχιακές που δεν ήταν επιθυμητή. Το σκεπτικό που οδήγησε σε αυτήν την απόφαση απορρέει από την παρατήρηση ότι οι φοιτητές που δηλώναν ομαδική εργασία την δήλωναν σε υψηλή θέση προτίμησης. Ακόμη, η ανάθεση τον ομαδικών εργασιών σε πρώτο χρόνο βοηθούσε να δεσμευτούν περισσότεροι φοιτητές με λιγότερα θέματα και να απλοποιηθεί ο αλγόριθμος ανάθεσης πτυχιακών εργασιών.

Στον παρών αλγόριθμο ανάθεσης πτυχιακών κρίθηκε σκόπιμο να μην υπάρχει αυτός ο διαχωρισμός. Ακόμη και αν η πιθανότητα να δηλωθεί μια ομαδική εργασία σε χαμηλή θέση προτίμησης είναι μικρή, για να ικανοποιηθεί μια πρωταρχική απαίτηση που είναι η δικαιότητα του αλγορίθμου, θα πρέπει να υπάρχει ισότητα αναμεσά στις ομαδικές και στις ατομικές πτυχιακές εργασίες. Αρνητικό αποτέλεσμα αυτής της απόφασης ήταν η έντονη αύξηση της πολυπλοκότητας του αλγορίθμου, κάτι το οποίο κρίθηκε αποδεκτό.

Κεφάλαιο 4. Ανάλυση Αλγορίθμου

4.1. Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει μια διεξοδική ανάλυση του αλγορίθμου ανάθεσης πτυχιακών εργασιών, όπως αυτός είναι στην τελική του μορφή. Θα γίνει επεξήγηση της βασικής λογικής του αλγορίθμου και του τρόπου λειτουργίας του. Οπού κρίνεται απαραίτητο θα υπάρχει παράθεση κώδικα, η οποία θα γίνεται στα παραρτήματα, στο τέλος της εργασίας.

4.2. Αρχικές διαδικασίες

Πριν αρχίσει ο αλγόριθμος ανάθεσης πτυχιακών είναι απαραίτητο να γίνουν κάποιες διαδικασίες (Παράρτημα 1). Παρακάτω ακολουθεί μια αναλυτική επεξήγηση των διαδικασιών αυτών με την σειρά με την οποία συμβαίνουν και στο πρόγραμμα.

Πρώτο μέλημα του προγράμματος είναι η εύρεση ή η δημιουργία του κεντρικού φακέλου όπου θα αποθηκεύονται τα δεδομένα και τα αποτελέσματα του αλγορίθμου (Παράρτημα 2).

Αμέσως μετά ακολουθεί η συλλογή των δεδομένων που είναι απαραίτητα για την εκτέλεση του αλγορίθμου. Επειδή το πρόγραμμα έχει φτιαχτεί με στόχο να μπορεί να τρέξει και χωρίς σύνδεση στην βάση δεδομένων, σε αυτό το σημείο γίνεται έλεγχος για το αν μπορεί να επιτευχθεί σύνδεση και αν όχι τότε η συλλογή δεδομένων γίνεται από τα τοπικά αρχεία. Εμφανές σε αυτό το σημείο είναι η προϋπόθεση να έχει συνδεθεί το πρόγραμμα με την βάση δεδομένων τουλάχιστον μια φορά στο παρελθόν ώστε να έχουν δημιουργηθεί τα τοπικά αντίγραφα (Παράρτημα 3). Εφόσον η σύνδεση με την βάση δεδομένων είναι εφικτή τότε γίνεται συλλογή των απαραίτητων δεδομένων από την βάση (Παράρτημα 4) και αμέσως μετά γίνεται η ενημέρωση ή η δημιουργία των τοπικών αντιγράφων (Παράρτημα 5). Αν η σύνδεση με την βάση δεν μπορεί να επιτευχθεί τότε τα απαραίτητα δεδομένα συλλέγονται από τα τοπικά αρχεία CSV (Παράρτημα 6). Αξίζει εδώ να σημειωθεί ότι ο πίνακας grades περιέχει μόνο βαθμολογίες περασμένων μαθήματων. Επίσης, ο λόγος για τον οποίο συλλέγονται όλες οι πληροφορίες του συνόλου των ενεργών φοιτητών και θεμάτων είναι γιατί το πρόγραμμα πρέπει να μπορεί να λειτουργήσει και χωρίς σύνδεση στη βάση

δεδομένων. Σε αυτό το σημείο η συλλογή δεδομένων έχει ολοκληρωθεί, τα δεδομένα έχουν μετατραπεί σε λίστες αντικείμενων και η επεξεργασία τους μπορεί να ξεκινήσει.

Η επεξεργασία των παραπάνω δεδομένων μπορεί να χωριστεί σε 4 βασικά βήματα (Παράρτημα 7). Αρχικά θα πρέπει να βρεθούν οι βαθμοί κάθε φοιτητή σε κάθε μάθημα. Εδώ πρέπει να δοθεί ιδιαίτερη προσοχή γιατί υπάρχουν και μαθήματα που χωρίζονται σε δυο μέρη (εργαστήριο και θεωρία). Σε αυτήν την περίπτωση για να θεωρηθεί ένα μάθημα περασμένο θα πρέπει ο φοιτητής να έχει περάσει και τα δυο μέρη. Τέλος, προσοχή θα πρέπει να δοθεί και στην βαρύτητα του βαθμού κάθε μέρους, μιας και το εργαστηριακό μέρος ενός μαθήματος είναι το 40% του τελικού βαθμού και η θεωρία το 60%. Ο διαχωρισμός αναμεσά στα ήδη μαθήματων, όπως αυτά είναι αποθηκευμένα στη βάση δεδομένων, γίνεται με την παρουσία ή απουσία των καταλήξεων «-Ε» ή « - Ε» για εργαστήρια και «-Θ» ή « - Θ» για θεωρίες (Παράρτημα 8).

Επόμενο βήμα είναι η δημιουργία μια νέας λίστας που θα περιέχει μόνο τους φοιτητές/ομάδες φοιτητών που έχουν επιλέξει θέμα πτυχιακής, ποια θέματα έχουν επιλέξει, σε τι προτεραιότητα και το συνολικό «Σκορ» για κάθε επιλογή τους. Σε αυτό το σημείο αποκλείονται οι φοιτητές/ομάδες φοιτητών που δεν πληρούν τις προϋποθέσεις των Προαπαιτούμενων μαθήματων (Παράρτημα 9).

Μετά την ολοκλήρωση των παραπάνω διεργασιών, έχουμε όλους τους φοιτητές που συμμετέχουν στον αλγόριθμο και όλα τα στοιχεία που χρειαζόμαστε, όμως αυτό δεν είναι αρκετό, χρειαζόμαστε μια ακόμη μια λίστα που θα περιέχει όλα τα θέματα που έχουν έστω και έναν διεκδικητή και τους διεκδικητές τους (Παράρτημα 10).

Οι δυο αυτές λίστες περιέχουν όλες τις πληροφορίες και όλους τους δείκτες που χρειαζόμαστε για τις διαδικασίες αποφάσεων του αλγορίθμου ανάθεσης πτυχιακών, και το μόνο που μένει είναι να τις ταξινομήσουμε, την πρώτη με βάση τη σειρά προτίμησης κάθε θέματος και την δεύτερη με βάση το «Σκορ» του κάθε διεκδικητή (Παράρτημα 11).

4.3. Ο αλγόριθμος ανάθεσης πτυχιακών εργασιών

Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο ο αλγόριθμος έχει πολλαπλά περάσματα και σε κάθε ένα από αυτά ανακατεύει την λίστα των μαθήματων με τυχαίο τρόπο (Παράρτημα 12). Η λογική πίσω από τις πολλαπλές επαναλήψεις είναι να γίνει μια προσομοίωση διαφορετικής σειράς αποφάσεων, μιας και ο αλγόριθμος δεν μπορεί να «δει παρακάτω βήματα». Παρακάτω ακολουθεί ένα παράδειγμα για να γίνει εύκολα κατανοητό.

Έστω ότι έχουμε 3 θέματα (Α,Β,Γ) και 3 φοιτητές (χ,ψ,ζ). Το θέμα Α το διεκδικούν οι φοιτητές χ και ζ, το θέμα Β οι φοιτητές χ και ψ και το θέμα Γ μόνο ο ψ (με σειρά καλύτερου «Σκορ»). Αν ο αλγόριθμος ξεκινήσει να μοιράζει τα θέματα από το Α, τότε θα δώσει το θέμα στον χ, το Β στον ψ και το Γ σε κανέναν. Αν όμως ξεκινήσει από το Γ τότε θα το δώσει στον ψ, μετρά το Β στον χ και το Α στον ζ.

Φυσικά ο αλγόριθμος δεν είναι τόσο απλός και έχει πολλαπλούς ελέγχους πριν την ανάθεση κάθε θέματος, το παραπάνω παράδειγμα είναι επίτηδες απλοποιημένο για ευκολότερη κατανόηση.

Ένα ακόμη βασικό στοιχείο του αλγορίθμου που είναι χρήσιμο να αναφερθεί εδώ είναι η ανακύκλωση θεμάτων. Όταν πληρούνται κάποιες προϋποθέσεις, ένας φοιτητής που έχει πάρει ήδη θέμα, μπορεί να αφήσει το προηγούμενο του θέμα για να πάρει το θέμα που βρίσκεται υπό εξέταση τώρα. Σε αυτήν την περίπτωση το ελεύθερο πλέον, προηγούμενο θέμα, γίνεται ξανά διαθέσιμο προς διεκδίκηση (Παράρτημα 13).

Ο Αλγόριθμος ξεκινάει την προσπέλαση στην τυχαία ανακατεμένη λίστα θεμάτων. Για κάθε θέμα, βρίσκει την υπολίστα με τους διεκδικητές, η οποία υπολίστα είναι ταξινομημένη με βάση το «Σκορ» του κάθε διεκδικητή για αυτό το θέμα σε φθίνουσα σειρά, δηλαδή ο προτιμότερος διεκδικητής πρώτος. Έπειτα, ο αλγόριθμος βρίσκει το προηγούμενο θέμα του εκάστοτε διεκδικητή, αν αυτό υπάρχει. Επόμενο βήμα είναι να γίνει οι παρακάτω έλεγχοι.

- ΑΝ το θέμα αυτό είναι το τελευταίο για αυτόν τον διεκδικητή.

- ΑΝ ο διεκδικητής αυτός είναι ο τελευταίος διαθέσιμος για αυτό το θέμα.

Οι δυο αυτοί έλεγχοι οδηγούν σε 4 διαφορετικές περιπτώσεις, οπού σε καθεμιά γίνονται κάποιες κατάλληλες τροποποιήσεις σε διάφορους δείκτες πριν ο αλγόριθμος προχωρήσει στο επόμενο βήμα (Παράρτημα 14). Αξίζει εδώ να αναφερθεί ότι σε κάποιες από αυτές τις 4 περιπτώσεις ένας έλεγχος που γίνεται είναι για το αν κάποιος άλλος φοιτητής που διεκδικεί αυτό το θέμα (από τους επόμενους στην υπολίστα) χρειάζεται το θέμα «περισσότερο». Αυτό μεταφράζεται στο ότι αν το θέμα δοθεί στον τωρινό διεκδικητή κάποιος από τους επόμενους διεκδικητές θα μείνει χωρίς θέμα. Αν αυτός ο έλεγχος είναι αληθής, τότε ο παρών διεκδικητής προσπερνάτε.

Το επόμενο βήμα του αλγορίθμου είναι και το πιο περίπλοκο, μιας και πρέπει να αντιμετωπίσει και τις ομαδικές εργασίες. Για να καταστεί αυτό δυνατό χρειάζεται μια ακόμη σειρά ελέγχων.

- Αν το θέμα που εξετάζεται είναι ατομικό ή ομαδικό.
- Αν το προηγούμενο θέμα του διεκδικητή που εξετάζεται είναι ατομικό ή ομαδικό.

Οι έλεγχοι αυτοί οδηγούν σε μια σειρά από διαφορετικές περιπτώσεις οι οποίες χρήζουν περαιτέρω ανάλυσης. Για την ανάλυση αυτή θα χρησιμοποιηθούν οι εξής συντομογραφίες.

- ΤΘ = Τωρινό Θέμα.
- ΠΘ = Προηγούμενο Θέμα.
- Δ = Ο παρών φοιτητής και Διεκδικητής του τωρινού θέματος.
- ΠΘ!Δ = Ο άλλος φοιτητής του Προηγούμενου Θέματος, όταν αυτό είναι ομαδικό, και όχι ο Διεκδικητής του τωρινού θέματος
- ΤΘΦ1 = Ο Πρώτος Φοιτητής του Τωρινού Θέματος, όταν αυτό είναι ομαδικό.

- ΠΘΜ1 = Ο Πρώτος Φοιτητής του Προηγούμενου Θέματος, όταν αυτό είναι ομαδικό.
- ΤΘΦ2 = Ο Δεύτερος Φοιτητής του Τωρινού Θέματος, όταν αυτό είναι ομαδικό.
- ΠΘΦ2 = Ο Δεύτερος Φοιτητής του Προηγούμενου Θέματος, όταν αυτό είναι ομαδικό.
- ΠΘ(ΤΘΦ2) = Το Προηγούμενο Θέμα του Δεύτερου Φοιτητή του Τωρινού Θέματος, όταν αυτό είναι ομαδικό.

Οι περιπτώσεις που παρατηρούνται είναι οι εξής.

1. ΤΘ και ΠΘ = ατομικά

   Σε αυτήν την περίπτωση ο Δ παίρνει όποιο από τα δυο θέματα είναι καλύτερο για αυτόν.

2. ΤΘ = ομαδικό και ΠΘ = ατομικό.

   Σε αυτήν την περίπτωση ακολουθεί μια σειρά από ελέγχους που έχουν σαν στόχο να αξιολογήσουν την κατάσταση του ΤΘΦ2 και του ΠΘ(ΤΘΦ2). Ενδεικτικά παρουσιάζονται κάποιες πιθανές εκβάσεις αυτόν των ελέγχων.

   - ΠΘ(ΤΘΦ2) = ομαδικό. Σε αυτήν την περίπτωση ο Δ προσπερνάτε.
   - ΠΘ(ΤΘΦ2) = ατομικό ΚΑΙ κάποιος άλλος φοιτητής το χρειάζεται. Αυτόματη ανάθεση ΤΘ και ανακύκλωση των ΠΘ και ΠΘ(ΤΘΦ2).
   - ΠΘ(ΤΘΦ2) = null ΚΑΙ ΤΘΦ2 δεν έχει άλλο ελεύθερο θέμα. Αυτόματη ανάθεση ΤΘ και ανακύκλωση του ΠΘ.
   - ΠΘ(ΤΘΦ2) = ατομικό. Σε αυτήν την περίπτωση ο Δ παίρνει όποιο από τα δυο θέματα είναι καλύτερο για αυτόν.

3. ΤΘ = ατομικό και ΠΘ = ομαδικό.

   Αυτή η περίπτωση χωρίζεται σε δυο, με μόνη διαφορά το αν Δ=ΠΘΜ1 ή Δ=ΠΘΦ2. Η λογική όμως και στις δυο περιπτώσεις είναι η ίδια, ΑΝ υπάρχει άλλο διαθέσιμο θέμα για τον ΠΘ!Δ τότε ο Δ παίρνει όποιο από τα δυο θέματα είναι καλύτερο για αυτόν.

4. ΤΘ = ομαδικό και ΠΘ = ομαδικό.

   Και εδώ ισχύει ο διαχωρισμός σε δυο περιπτώσεις με ίδια λογική. Ακολουθεί μια σειρά από ελέγχους που έχουν σαν στόχο να αξιολογήσουν την κατάσταση του ΤΘΦ2, ΠΘ!Δ, ΠΘ(ΤΘΦ2) και ΠΘ. Ενδεικτικά παρουσιάζονται κάποιες πιθανές εκβάσεις αυτών των ελέγχων.

   - Αν ΠΘ(ΤΘΦ2) = ομαδικό. Σε αυτήν την περίπτωση ο Δ προσπερνάτε.
   - ΑΝ ΠΘ(ΤΘΦ2) != null ΚΑΙ ΠΘ(ΤΘΦ2) δεν έχει άλλους διεκδικητές ΚΑΙ ΠΘ έχει άλλους διεκδικητές. Αυτόματη ανάθεση ΤΘ και ανακύκλωση των ΠΘ και ΠΘ(ΤΘΦ2).
   - ΑΝ ΠΘ(ΤΘΦ2) = null ΚΑΙ ΠΘ!Δ δεν έχει άλλα θέματα να πάρει. Σε αυτήν την περίπτωση ο Δ παίρνει όποιο από τα δυο θέματα είναι καλύτερο για αυτόν.

5. ΠΘ = null.

   Σε αυτή την περίπτωση ΑΝ ΤΘ = ομαδικό, γίνεται έλεγχος αν ο ΤΘΦ2 είναι διαθέσιμος και αναλόγως το θέμα ανατίθεται η προσπερνάτε. ΑΝ ΤΘ = ατομικό το θέμα ανατίθεται (Παράρτημα 15).

Σε περίπτωση που το θέμα ανατεθεί σε κάποιον διεκδικητή, τότε ο αλγόριθμος προσπερνά τους επόμενους διεκδικητές και προχωράει στο επόμενο θέμα. Η διαδικασία συνεχίζεται μέχρι το τελευταίο θέμα στη λίστα θεμάτων. Σε αυτό το σημείο η λίστα θεμάτων αντικαθίσταται με την λίστα ανακυκλωμένων θεμάτων. Όταν και αυτή

η λίστα αδειάσει τότε ο αλγόριθμος προχωράει στην διαχείριση αποτελεσμάτων (Παράρτημα 16).

Η διαχείριση των αποτελεσμάτων είναι μια απλή διαδικασία καταμέτρησης διάφορων στατιστικών που θα χρησιμοποιηθούν για την αξιολόγηση του συγκεκριμένου αποτελέσματος. Πιο συγκεκριμένα οι δείκτες που υπολογίζονται είναι οι εξής.

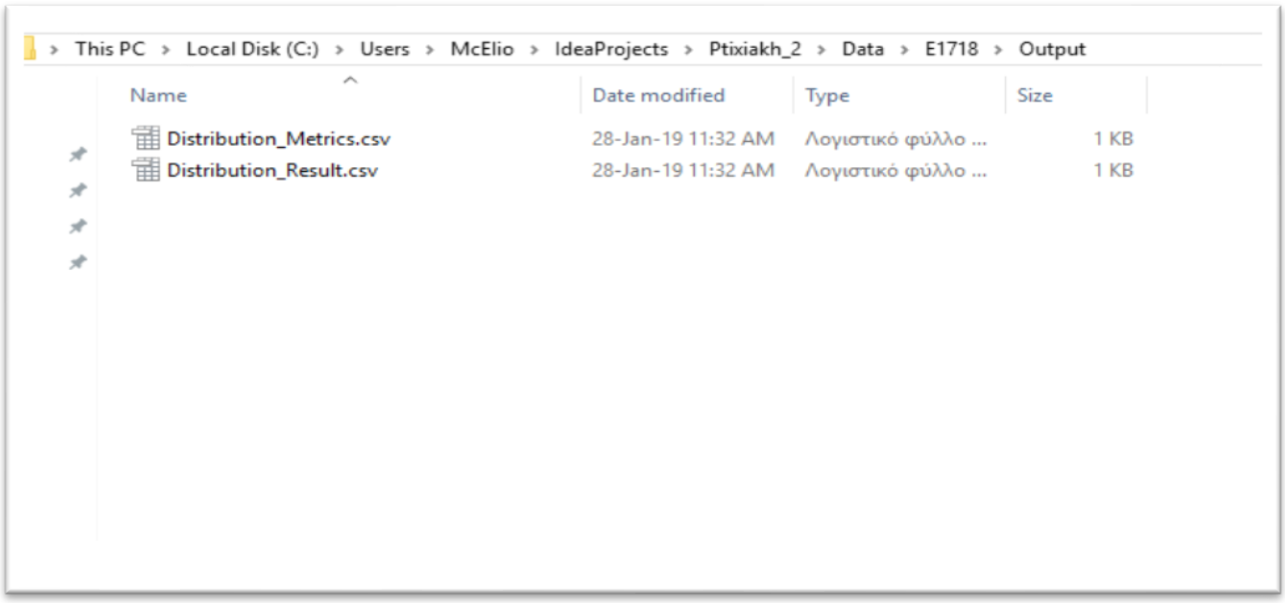
- Ο αριθμός των θεμάτων που μοιράστηκαν.
- Ο αριθμός των φοιτητών που πήραν θέμα.
- Το συνολικό σκορ του αλγορίθμου, δηλαδή το άθροισμα των «Σκορ» από όλα τα ζεύγη αποτελεσμάτων (φοιτητών και θέματος)
- Τον μέσο όρο προτεραιοτήτων που μοιράστηκαν.
- Το πλήθος θεμάτων που δόθηκαν για κάθε πιθανή τιμή προτεραιότητας(1-5).

Εφόσον συγκεντρωθούν τα παραπάνω στοιχεία, τα στοιχεία αυτά μαζί με τα αποτελέσματα αποθηκεύονται και ο αλγόριθμος ξεκινά το επόμενο πέρασμα (Παράρτημα 17).

Αφού ολοκληρωθούν όλες οι επαναλήψεις, απαραίτητη είναι η εύρεση του καλύτερου αποτελέσματος από το πλήθος αποτελεσμάτων κάθε επανάληψης. Πρωταρχικό κριτήριο για την αξιολόγηση ενός αποτελέσματος είναι το συνολικό του σκορ. Οπότε απομονώνονται μόνο τα αποτελέσματα που έχουν το μέγιστο σκορ. Από αυτά τα αποτελέσματα επιλέγουμε αυτό που έχει ικανοποιήσει το μεγαλύτερο πλήθος φοιτητών, δηλαδή αυτόν που έχει τον μεγαλύτερο αριθμό φοιτητών που πήραν θέμα. Έχοντας βρει το βέλτιστο αποτέλεσμα, υπολογίζουμε τον χρόνο εκτέλεσης του αλγορίθμου (για στατιστικούς λόγους) και επιστρέφουμε το αποτέλεσμα για εκτύπωση (Παράρτημα 18).

4.4. Αποθήκευση αποτελέσματος.

Τα αποτελέσματα αποθηκεύονται τοπικά υπό την μορφή αρχείων CSV. Στον φάκελο «Output» δημιουργούνται δυο αρχεία το ένα με τα ζεύγη φοιτητών - θεμάτων ενώ το δεύτερο περιέχει όλα τα στατιστικά στοιχεία που αναφέρθηκαν παραπάνω (Παράρτημα 19).



Επίλογος

Κατά την διαδικασία υλοποίησης του αλγορίθμου έγινε γρηγορά κατανοητό ότι η αυτοματοποίηση της ανάθεσης πτυχιακών εργασιών δεν είναι μια εύκολη διαδικασία. Υπάρχουν πολλοί παράμετροι, διαφορετικοί και αντικρουόμενοι στόχοι και η ύπαρξη των ομαδικών εργασιών προσθέτει περεταίρω πολυπλοκότητα.

Η πρόκληση της δημιουργίας ενός αλγορίθμου λήψης αποφάσεων σε συνδυασμό με τη προοπτική ότι το τελικό πρόγραμμα θα αξιοποιηθεί από το Τμήμα Μηχανικών Πληροφορικής προσφέραν ένα σημαντικό κίνητρο για την υλοποίηση της πτυχιακής αυτής.

Η πτυχιακή εργασία είναι μια μοναδική ευκαιρία για κάθε φοιτητή και ένας αξιόλογος τρόπος της ολοκλήρωσης των σπουδών του. Ειδικότερα όταν η πτυχιακή αυτή συνδυάζεται με την υλοποίηση ενός ολοκληρωμένου προγράμματος δίνει την ευκαιρία στον φοιτητή όχι μόνο να εφαρμόσει, εξελίξει και συνδυάσει τις γνώσεις και ικανότητες που αποκόμισε κατά την διάρκεια της φοίτησης του, αλλά του δίνει ένα κίνητρο να δημιουργήσει κάτι για το οποίο μπορεί να είναι περήφανος, ένα δημιούργημα το οποίο θα αναδεικνύει τις δυνατότητες του φοιτητή, το οποίο μπορεί ακόμη να χρησιμοποιηθεί για να ενισχύσει το πορτφόλιο του και να τον βοηθήσει στην εύρεση εργασίας.

Προβλήματα που αντιμετωπίστηκαν και αφορούσαν την βάση δεδομένων, όπως ήταν η ελλιπής συσχέτισης μαθήματων από διαφορετικά προγράμματα σπουδών, δεν μπορούσαν να διορθωθούν οπότε έπρεπε να παρακαμφθούν. Σαν αποτέλεσμα, η πιθανότητα ότι η τελική μορφή του προγράμματος θα χρειαστεί μικρές τροποποιήσεις και αλλαγές είναι αναμενόμενη. Σαν περήφανος απόφοιτος του τμήματος, θα βρίσκομαι στην διάθεση του τμήματος και της Επιτροπής Πτυχιακών Εργασιών ώστε το πρόγραμμα αυτόματης ανάθεσης πτυχιακών εργασιών να παραμείνει χρήσιμο για την σχολή μου και αντάξιο της φήμης αυτής.

Βιβλιογραφία

https://el.wikipedia.org/wiki/Java (Πρόσβαση: 09-02-2019)

http://www.islab.demokritos.gr/gr/html/ptixiakes/kostas-aris_ptyxiakh/Phtml/java.htm (Πρόσβαση: 09-02-2019)

https://dev.mysql.com/downloads/connector/j/8.0.html (Πρόσβαση: 09-02-2019)

https://commons.apache.org/proper/commons-csv/ (Πρόσβαση: 09-02-2019)

https://commons.apache.org/proper/commons-csv/apidocs/index.html (Πρόσβαση: 09-02-2019)

https://el.wikipedia.org/wiki/Ολοκληρωμένο_περιβάλλον_ανάπτυξης (Πρόσβαση: 09-02-2019)

https://en.wikipedia.org/wiki/IntelliJ_IDEA (Πρόσβαση: 09-02-2019)

Παραρτήματα

Παράρτημα 1.

```
public static void main(String rags[]) {
    Ptixiakh_2 obj = new Ptixiakh_2();
    obj.begin();
}

private void begin() {
    csv_handler.find_Or_Create_Parent_Directory();
    get_Data();
    modify_Data();
    calculate_Result_Begin_Metrics();
    final_result = distribution_algorithm.start(choices_of_students, students_for_each_subject);
    handle_Final_Result();
}
```

Παράρτημα 2.

```
public void find_Or_Create_Parent_Directory() {
    File directory = new File(parent_directory);
    if (!directory.exists())
        directory.mkdir();
}
```

Παράρτημα 3.

```
private void get_Data() {
    if (db_connection.can_Connect_To_DB()) {
        get_Data_From_DB();
        csv_handler.write_Data_To_CSV(current_semester, dissertations, students, candidacies, grades);
    } else
        get_Data_From_CSV();
}

public boolean can_Connect_To_DB() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        System.err.println("No MySQL JDBC Driver?");
        e.printStackTrace();
        return false;
    }

    try {
        theconnection = DriverManager
                .getConnection(db_url, db_username, db_password);

    } catch (SQLException e) {
        System.err.println("Connection Failed! Check output console");
        e.printStackTrace();
        return false;
    }

    if (theconnection != null) {
        return true;
    } else {
        System.err.println("Failed to make connection!");
        return false;
    }
}
```

Παράρτημα 4.

```
private void get_Data_From_DB() {
    current_semester = db_connection.get_Semester_From_DB();
    dissertations = db_connection.get_Dissertations_From_DB(current_semester);
    students = db_connection.get_Students_From_DB();
    candidacies = db_connection.get_Candidacy_From_DB();
    grades = db_connection.get_Grades_From_DB();
}

public String get_Semester_From_DB() {
    final String querycurrentsemester = "SELECT semester FROM current_semester";
    String cursemester = "";

    try {
        PreparedStatement pssemester = theconnection.prepareStatement(querycurrentsemester);
        ResultSet rssemester = pssemester.executeQuery();

        while (rssemester.next())
            cursemester = rssemester.getString("semester");
    } catch (Exception e) {
        e.printStackTrace();
    }
    return cursemester;
}

public List<DB_Dissertation> get_Dissertations_From_DB(String currentsemester) {
    final String querydissertations = "SELECT id, titlegr, supervisor, course1, course2, course3, course4 FROM dissertations
WHERE semester = ?";
    List<DB_Dissertation> dissertations = new ArrayList<>();
    try {
        PreparedStatement psdissertations = theconnection.prepareStatement(querydissertations);
        psdissertations.setString(1, currentsemester);
        ResultSet rsdissertations = psdissertations.executeQuery();

        while (rsdissertations.next()) {
            int id = rsdissertations.getInt("id");
            String titlegr = rsdissertations.getString("titlegr");
            String supervisor = rsdissertations.getString("supervisor");
            List<String> tmp = new ArrayList<>();
            tmp.add(rsdissertations.getString("course1"));
            tmp.add(rsdissertations.getString("course2"));
            tmp.add(rsdissertations.getString("course3"));
            tmp.add(rsdissertations.getString("course4"));

            List<String> courses = tmp.stream().filter(temp -> temp != null && !temp.isEmpty()).collect(Collectors.toList());

            DB_Dissertation dissertation = new DB_Dissertation(id, titlegr, supervisor, courses);
            dissertations.add(dissertation);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return dissertations;
}

public List<DB_Student> get_Students_From_DB() {
    final String querystudents = "SELECT am, first_name, last_name, didaktikes, gmo, syntel  FROM students WHERE cond_ID
= 1";
    List<DB_Student> students = new ArrayList<>();
    try {
        PreparedStatement psstudents = theconnection.prepareStatement(querystudents);
        ResultSet rsstudents = psstudents.executeQuery();

        while (rsstudents.next()) {
            String am = rsstudents.getString("am");
            String first_name = rsstudents.getString("first_name");
            String last_name = rsstudents.getString("last_name");
            int didaktikes = rsstudents.getInt("didaktikes");
```

```
            double gmo = rsstudents.getDouble("gmo");
            int syntel = rsstudents.getInt("syntel");

            DB_Student student = new DB_Student(am, first_name, last_name, didaktikes, gmo, syntel);
            students.add(student);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return students;
}

public List<DB_Candidacy> get_Candidacy_From_DB() {
    final String querycandidacy = "SELECT dissertation, student, preference, student2 FROM candidacy";
    List<DB_Candidacy> candidacies = new ArrayList<>();
    try {
        PreparedStatement pscandidacy = theconnection.prepareStatement(querycandidacy);
        ResultSet rscandidacy = pscandidacy.executeQuery();

        while (rscandidacy.next()) {
            int dissertationid = rscandidacy.getInt("dissertation");
            String studentam = rscandidacy.getString("student");
            int preference = rscandidacy.getInt("preference");
            String student2am = rscandidacy.getString("student2");

            DB_Candidacy tmp = new DB_Candidacy(dissertationid, studentam, preference, Optional.ofNullable(student2am));
            candidacies.add(tmp);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return candidacies;
}

public List<DB_Grade> get_Grades_From_DB() {
    final String querygrades = "SELECT grades.am, grades.title, grades.cgrade, students.cond_ID FROM grades INNER JOIN
students ON grades.am = students.am WHERE students.cond_ID = 1";
    List<DB_Grade> grades = new ArrayList<>();
    try {
        PreparedStatement psgrades = theconnection.prepareStatement(querygrades);
        ResultSet rsgrades = psgrades.executeQuery();

        while (rsgrades.next()) {
            String studentam = rsgrades.getString("am");
            String title = rsgrades.getString("title");
            double cgrade = rsgrades.getDouble("cgrade");

            DB_Grade tmp = new DB_Grade(studentam, title, cgrade);
            grades.add(tmp);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return grades;
}
```

Παράρτημα 5.

```
public void write_Data_To_CSV(String current_semester, List<DB_Dissertation> dissertations, List<DB_Student> students,
List<DB_Candidacy> candidacies, List<DB_Grade> grades) {
    update_Or_Create_Current_Semester(current_semester);
    find_Or_Create_Directory(parent_directory + "/" + this.current_semester);
    find_Or_Create_Directory(parent_directory + "/" + this.current_semester + "/" + input_directory_name);
    update_Or_Create_Dissertations(dissertations);
    update_Or_Create_Students(students);
    update_Or_Create_Candidacy(candidacies);
    update_Or_Create_Grades(grades);
}



private void update_Or_Create_Current_Semester(String current_semester) {
    Path current_semester_file = Paths.get(parent_directory + "/" + latest_semester_csv_file_name);
    if (current_semester_file.toFile().exists())
        current_semester_file.toFile().delete();
    try {
        BufferedWriter writer = Files.newBufferedWriter(current_semester_file);

        CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT
                .withHeader("semester"));
        csvPrinter.printRecord(current_semester);
        csvPrinter.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
    this.current_semester = current_semester;
}

private void update_Or_Create_Dissertations(List<DB_Dissertation> dissertations) {
    Path dissertations_file = Paths.get(parent_directory + "/" + current_semester + "/" + input_directory_name + "/" +
dissertations_file_name);
    if (dissertations_file.toFile().exists())
        dissertations_file.toFile().delete();
    try {
        BufferedWriter writer = Files.newBufferedWriter(dissertations_file);

        CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT
                .withHeader("id", "titlegr", "supervisor", "course1", "course2", "course3", "course4"));
        for (DB_Dissertation dissertation : dissertations) {
            List<String> tmp = new ArrayList<>();
            List<String> required_courses = dissertation.getCourses();
            for (int i = 0; i < 4; i++) {
                try {
                    tmp.add(required_courses.get(i));
                } catch (IndexOutOfBoundsException e) {
                    tmp.add("");
                }
            }
            csvPrinter.printRecord(Integer.toString(dissertation.getId()), dissertation.getName(), dissertation.getSupervisor(),
tmp.get(0), tmp.get(1), tmp.get(2), tmp.get(3));
        }
        csvPrinter.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void update_Or_Create_Students(List<DB_Student> students) {
    Path students_file = Paths.get(parent_directory + "/" + current_semester + "/" + input_directory_name + "/" +
students_file_name);
    if (students_file.toFile().exists())
        students_file.toFile().delete();
    try {
        BufferedWriter writer = Files.newBufferedWriter(students_file);
```

```
        CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT
                .withHeader("am", "first_name", "last_name", "didaktikes", "gmo", "syntel"));
        for (DB_Student student : students) {
            csvPrinter.printRecord(student.getAm(), student.getFirst_name(), student.getLast_name(),
                    Integer.toString(student.getDidaktikes()), Double.toString(student.getGmo()), Integer.toString(student.getSyntel()));
        }
        csvPrinter.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void update_Or_Create_Candidacy(List<DB_Candidacy> candidacies) {
    Path candidacies_file = Paths.get(parent_directory + "/" + current_semester + "/" + input_directory_name + "/" +
candidacies_file_name);
    if (candidacies_file.toFile().exists())
        candidacies_file.toFile().delete();
    try {
        BufferedWriter writer = Files.newBufferedWriter(candidacies_file);

        CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT
                .withHeader("dissertation", "student", "preference", "student2"));
        for (DB_Candidacy candidacy : candidacies) {
            csvPrinter.printRecord(Integer.toString(candidacy.getDissertation_id()), candidacy.getStudent_am(),
Integer.toString(candidacy.getPreference()), candidacy.getStudent2_am());
        }
        csvPrinter.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void update_Or_Create_Grades(List<DB_Grade> grades) {
    Path grades_file = Paths.get(parent_directory + "/" + current_semester + "/" + input_directory_name + "/" +
grades_file_name);
    if (grades_file.toFile().exists())
        grades_file.toFile().delete();
    try {
        BufferedWriter writer = Files.newBufferedWriter(grades_file);

        CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT
                .withHeader("am", "title", "cgrade"));
        for (DB_Grade grade : grades) {
            csvPrinter.printRecord(grade.getStudent_am(), grade.getTitle(), Double.toString(grade.getCourse_grade()));
        }
        csvPrinter.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Παράρτημα 6.

```
private void get_Data_From_CSV() {
    current_semester = csv_handler.get_Semester_From_CSV();
    dissertations = csv_handler.get_Dissertations_From_CSV();
    students = csv_handler.get_Students_From_CSV();
    candidacies = csv_handler.get_Candidacy_From_CSV();
    grades = csv_handler.get_Grades_From_CSV();
}

public String get_Semester_From_CSV() {
    String current_semester = new String();
    Path current_semester_file = Paths.get(parent_directory + "/" + latest_semester_csv_file_name);
    if (Files.exists(current_semester_file)) {
        try {
            Reader reader = Files.newBufferedReader(current_semester_file);
            CSVParser csvparser = new CSVParser(reader, CSVFormat.DEFAULT
                    .withFirstRecordAsHeader()
                    .withIgnoreHeaderCase()
                    .withTrim());
            for (CSVRecord csvrecord : csvparser)
                current_semester = csvrecord.get("semester");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    this.current_semester = current_semester;
    return current_semester;
}

public List<DB_Dissertation> get_Dissertations_From_CSV() {
    List<DB_Dissertation> dissertations = new ArrayList<>();
    Path dissertations_file = Paths.get(parent_directory + "/" + current_semester + "/" + input_directory_name + "/" +
dissertations_file_name);
    if (Files.exists(dissertations_file)) {
        try {
            Reader reader = Files.newBufferedReader(dissertations_file);
            CSVParser csvParser = new CSVParser(reader, CSVFormat.DEFAULT
                    .withFirstRecordAsHeader()
                    .withIgnoreHeaderCase()
                    .withTrim());

            for (CSVRecord csvRecord : csvParser) {
                int id = Integer.parseInt(csvRecord.get("id"));
                String titlegr = csvRecord.get("titlegr");
                String supervisor = csvRecord.get("supervisor");
                List<String> tmp = new ArrayList<>();
                tmp.add(csvRecord.get("course1"));
                tmp.add(csvRecord.get("course2"));
                tmp.add(csvRecord.get("course3"));
                tmp.add(csvRecord.get("course4"));

                List<String> courses = tmp.stream().filter(temp -> temp != null && !temp.isEmpty()).collect(Collectors.toList());

                DB_Dissertation dissertation = new DB_Dissertation(id, titlegr, supervisor, courses);
                dissertations.add(dissertation);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return dissertations;
}

public List<DB_Student> get_Students_From_CSV() {
    List<DB_Student> students = new ArrayList<>();
    Path students_file = Paths.get(parent_directory + "/" + current_semester + "/" + input_directory_name + "/" +
students_file_name);
    if (Files.exists(students_file)) {
```

```
        try {
            Reader reader = Files.newBufferedReader(students_file);
            CSVParser csvParser = new CSVParser(reader, CSVFormat.DEFAULT
                    .withFirstRecordAsHeader()
                    .withIgnoreHeaderCase()
                    .withTrim());

            for (CSVRecord csvRecord : csvParser) {
                String am = csvRecord.get("am");
                String first_name = csvRecord.get("first_name");
                String last_name = csvRecord.get("last_name");
                int didaktikes = Integer.parseInt(csvRecord.get("didaktikes"));
                double gmo = Double.parseDouble(csvRecord.get("gmo"));
                int syntel = Integer.parseInt(csvRecord.get("syntel"));

                DB_Student student = new DB_Student(am, first_name, last_name, didaktikes, gmo, syntel);
                students.add(student);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return students;
}

public List<DB_Candidacy> get_Candidacy_From_CSV() {
    List<DB_Candidacy> candidacies = new ArrayList<>();
    Path candidacies_file = Paths.get(parent_directory + "/" + current_semester + "/" + input_directory_name + "/" +
candidacies_file_name);
    // check if candidacy.csv file exists. If it does no need to connect to DB
    if (Files.exists(candidacies_file)) {
        // get data from candidacy.csv
        try {
            Reader reader = Files.newBufferedReader(candidacies_file);
            CSVParser csvParser = new CSVParser(reader, CSVFormat.DEFAULT
                    .withFirstRecordAsHeader()
                    .withIgnoreHeaderCase()
                    .withTrim());

            for (CSVRecord csvRecord : csvParser) {
                int dissertationid = Integer.parseInt(csvRecord.get("dissertation"));
                String studentam = csvRecord.get("student");
                int preference = Integer.parseInt(csvRecord.get("preference"));
                String student2am = csvRecord.get("student2");

                DB_Candidacy candidacy = new DB_Candidacy(dissertationid, studentam, preference,
Optional.ofNullable(student2am));
                candidacies.add(candidacy);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return candidacies;
}

public List<DB_Grade> get_Grades_From_CSV() {
    List<DB_Grade> grades = new ArrayList<>();
    Path grades_file = Paths.get(parent_directory + "/" + current_semester + "/" + input_directory_name + "/" +
grades_file_name);
    if (Files.exists(grades_file)) {
        try {
            Reader reader = Files.newBufferedReader(grades_file);
            CSVParser csvParser = new CSVParser(reader, CSVFormat.DEFAULT
                    .withFirstRecordAsHeader()
                    .withIgnoreHeaderCase()
                    .withTrim());

            for (CSVRecord csvRecord : csvParser) {
                String studentam = csvRecord.get("am");
```

```
                String title = csvRecord.get("title");
                double cgrade = Double.parseDouble(csvRecord.get("cgrade"));

                DB_Grade grade = new DB_Grade(studentam, title, cgrade);
                grades.add(grade);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return grades;
}
```

Παράρτημα 7.

```
private void modify_Data() {
    add_Grades_To_Student();
    create_Choices_Of_Students();
    invert_To_Subjects_Remove_Zero();
    sort_The_lists();
}
```

Παράρτημα 8.

```
private void add_Grades_To_Student() {
    for (DB_Student student : students) {
        List<DB_Grade> raw_grades_of_student = grades.stream()
                .filter(grade -> grade.getStudent_am().equals(student.getAm())).collect(Collectors.toList());
        student.setGrades(calculate_Course_Grade(raw_grades_of_student));
    }
}

private List<DB_Grade> calculate_Course_Grade(List<DB_Grade> raw_grades_of_student) {
    List<DB_Grade> final_grades = new ArrayList<>();
    for (DB_Grade the_grade : raw_grades_of_student) {
        if (the_grade.getTitle().toUpperCase().endsWith("-E".toUpperCase()) || the_grade.getTitle().toUpperCase().endsWith("-
Θ".toUpperCase()))
            find_Both_Parts_And_Calculate_Course_Average(2, the_grade, final_grades,
raw_grades_of_student).ifPresent(final_grades::add);
        else if (the_grade.getTitle().toUpperCase().endsWith(" - E".toUpperCase()) ||
the_grade.getTitle().toUpperCase().endsWith(" - Θ".toUpperCase()))
            find_Both_Parts_And_Calculate_Course_Average(4, the_grade, final_grades,
raw_grades_of_student).ifPresent(final_grades::add);
        else
            final_grades.add(new DB_Grade(the_grade.getStudent_am(), the_grade.getTitle(), the_grade.getCourse_grade()));
    }
    return final_grades;
}

private Optional<DB_Grade> find_Both_Parts_And_Calculate_Course_Average(int the_ending, DB_Grade the_grade,
List<DB_Grade> final_grades, List<DB_Grade> raw_grades_of_student) {
    DB_Grade the_course = null;
    String course_title = the_grade.getTitle().substring(0, (the_grade.getTitle().length() - the_ending));
    boolean isassigned = final_grades.stream().anyMatch(final_grade ->
course_title.toUpperCase().equals(final_grade.getTitle().toUpperCase()));
    List<DB_Grade> the_course_parts = raw_grades_of_student.stream()
            .filter(raw_grade -> raw_grade.getTitle().toUpperCase().substring(0, (raw_grade.getTitle().length() -
the_ending)).equals(course_title.toUpperCase())).collect(Collectors.toList());
    if (the_course_parts.size() == 2 && !isassigned)
        the_course = new DB_Grade(the_grade.getStudent_am(), course_title, calculate_Course_Average(the_course_parts));
    return Optional.ofNullable(the_course);
}

private double calculate_Course_Average(List<DB_Grade> the_course_parts) {
    if (the_course_parts.get(0).getTitle().toUpperCase().endsWith("E".toUpperCase()))
        return ((the_course_parts.get(0).getCourse_grade() * 0.4) + (the_course_parts.get(1).getCourse_grade() * 0.6));
    else
        return ((the_course_parts.get(1).getCourse_grade() * 0.4) + (the_course_parts.get(0).getCourse_grade() * 0.6));
}
```

Παράρτημα 9.

```
private void create_Choices_Of_Students() {
    List<DB_Candidacy> copy = candidacies;
    for (DB_Candidacy candidate : copy) {
        Optional<DB_Dissertation> the_subject = dissertations.stream()
                .filter(subject -> subject.getId() == candidate.getDissertation_id()).findFirst();
        if (!the_subject.isPresent())
            continue;
        Optional<DB_Student> the_first_student = students.stream()
                .filter(student -> student.getAm().equals(candidate.getStudent_am())).findFirst();
        if (!the_first_student.isPresent())
            continue;
        Optional<DB_Student> the_second_student = students.stream()
                .filter(student -> student.getAm().equals(candidate.getStudent2_am())).findFirst();
        boolean has_other_choices = choices_of_students.stream().anyMatch(the_choice ->
candidate.getStudent_am().equals(the_choice.getStudent().getAm()));
        double first_avarage = calculate_Average_Required_Courses_Score(the_subject.get(), the_first_student.get());
        double second_avarage;
        My_Student my_second_student;
        if (the_second_student.isPresent()) {
            second_avarage = calculate_Average_Required_Courses_Score(the_subject.get(), the_second_student.get());
            my_second_student = new My_Student(the_second_student.get(), second_avarage);
        } else {
            second_avarage = -1;
            my_second_student = null;
        }
        if ((first_avarage * second_avarage) == 0)
            continue;
        if (has_other_choices) {
            My_Dissertation tmp = new My_Dissertation(the_subject.get(), candidate.getPreference(),
Optional.ofNullable(my_second_student), first_avarage);
            choices_of_students.stream()
                    .filter(thechoice -> candidate.getStudent_am().equals(thechoice.getStudent().getAm()))
                    .findFirst()
                    .ifPresent(thechoise -> thechoise.getSubjectchoices().add(tmp));
        } else {
            List<My_Dissertation> tmplist = new ArrayList<>();
            tmplist.add(new My_Dissertation(the_subject.get(), candidate.getPreference(), Optional.ofNullable(my_second_student),
first_avarage));
            Choices_Of_Student tmpchoise = new Choices_Of_Student(the_first_student.get(), tmplist);
            choices_of_students.add(tmpchoise);
        }
    }
}

private double calculate_Average_Required_Courses_Score(DB_Dissertation the_subject, DB_Student the_student) {
    double the_avarage = 0;
    int number_of_courses = the_subject.getCourses().size();

    for (String course : the_subject.getCourses()) {
        Optional<DB_Grade> coursegrade = the_student.getGrades().stream()
                .filter(grade -> grade.getTitle().toUpperCase().equals(course.toUpperCase())).findFirst();

        if (coursegrade.isPresent())
            the_avarage += coursegrade.get().getCourse_grade();
        else
            return 0;
    }
    the_avarage = the_avarage / number_of_courses;
    return the_avarage;
}

public class Choices_Of_Student {
    private DB_Student student;
    private List<My_Dissertation> subjectchoices;
    Choices_Of_Student(DB_Student student, List<My_Dissertation> subjectchoices) {
        this.student = student;
        this.subjectchoices = subjectchoices;
```

```
    }

    public DB_Student getStudent() {
        return student;
    }

    public List<My_Dissertation> getSubjectchoices() {
        return subjectchoices;
    }
}

public class My_Dissertation extends DB_Dissertation {
    private int priority;
    private Optional<My_Student> student2;
    private double required_courses_grade_average;
    private boolean is_group;

    My_Dissertation(DB_Dissertation subject, int priority, Optional<My_Student> student2, double
required_courses_grade_average) {
        super(subject.getId(), subject.getName(), subject.getSupervisor(), subject.getCourses());
        this.priority = priority;
        this.student2 = student2;
        this.required_courses_grade_average = required_courses_grade_average;
        if (student2.isPresent())
            is_group = true;
        else
            is_group = false;
    }

    My_Dissertation(DB_Dissertation subject, int priority, double required_courses_grade_average) {
        super(subject.getId(), subject.getName(), subject.getSupervisor(), subject.getCourses());
        this.priority = priority;
        this.student2 = null;
        this.required_courses_grade_average = required_courses_grade_average;
        is_group = false;
    }

    public int getPriority() {
        return priority;
    }

    public Optional<My_Student> getStudent2() {
        return student2;
    }

    public double getRequired_courses_grade_average() {
        return required_courses_grade_average;
    }



public boolean isIs_group() {
        return is_group;
    }
}
```

Παράρτημα 10.

```
private void invert_To_Subjects_Remove_Zero() {
    for (DB_Dissertation subject : dissertations) {
        List<My_Student> contenders_per_subject = new ArrayList<>();
        for (Choices_Of_Student choice : choices_of_students) {
            Optional<My_Dissertation> found_subject = choice.getSubjectchoices().stream()
                    .filter(selected_subject -> selected_subject.getName().toUpperCase().equals(subject.getName().toUpperCase()))
                    .findFirst();
            found_subject.ifPresent(the_found_subject -> contenders_per_subject
                    .add(new My_Student(choice.getStudent(), the_found_subject.getPriority(),
the_found_subject.getRequired_courses_grade_average(), the_found_subject.getStudent2())));
        }
        if (!contenders_per_subject.isEmpty())
            students_for_each_subject.add(new Students_Per_Subject(subject, contenders_per_subject));
    }
}

public class Students_Per_Subject {
    private DB_Dissertation subject;
    private List<My_Student> contenders;

    Students_Per_Subject(DB_Dissertation subject, List<My_Student> contenders) {
        this.subject = subject;
        this.contenders = contenders;
    }

    public DB_Dissertation getSubject() {
        return subject;
    }

    public List<My_Student> getContenders() {
        return contenders;
    }
}

public class My_Student extends DB_Student {
    private int priority;
    private double required_courses_grade_average;
    private Optional<My_Student> student2;
    private double total_score;


    My_Student(DB_Student student, int priority, double required_courses_grade_average) {
        super(student.getAm(), student.getFirst_name(), student.getLast_name(), student.getDidaktikes(), student.getGmo(),
student.getSyntel(), student.getBathmos(), student.getGrades());
        this.priority = priority;
        this.required_courses_grade_average = required_courses_grade_average;
    }

    My_Student(DB_Student student, double required_courses_grade_average) {
        super(student.getAm(), student.getFirst_name(), student.getLast_name(), student.getDidaktikes(), student.getGmo(),
student.getSyntel(), student.getBathmos(), student.getGrades());
        this.priority = -1;
        this.required_courses_grade_average = required_courses_grade_average;
    }

    My_Student(DB_Student student, int priority, double required_courses_grade_average, Optional<My_Student> student2) {
        super(student.getAm(), student.getFirst_name(), student.getLast_name(), student.getDidaktikes(), student.getGmo(),
student.getSyntel(), student.getBathmos(), student.getGrades());
        this.priority = priority;
        this.required_courses_grade_average = required_courses_grade_average;
        this.student2 = student2;
    }

    public int getPriority() {
        return priority;
    }
```

```
    public double getRequired_courses_grade_average() {
        return required_courses_grade_average;
    }

    public double getTotal_score() {
        return total_score;
    }

    public void setTotal_score(double total_score) {
        this.total_score = total_score;
    }

    public Optional<My_Student> getStudent2() {
        return student2;
    }
}
```

Παράρτημα 11.

```
private void sort_The_lists() {
    for (Choices_Of_Student student : choices_of_students)
        student.getSubjectchoices().sort(Comparator.comparingInt(My_Dissertation::getPriority));
    calculate_Total_Score();
    for (Students_Per_Subject subject : students_for_each_subject)
        subject.getContenders().sort(Comparator.comparingDouble(My_Student::getTotal_score));
}

private void calculate_Total_Score() {
    for (Students_Per_Subject subject : students_for_each_subject) {
        for (My_Student student : subject.getContenders()) {
            student.getStudent2().ifPresent(student2 -> student2.setTotal_score((student2.getRequired_courses_grade_average() *
2) + (11 - student.getPriority()) + student2.getBathmos()));
            student.setTotal_score((student.getRequired_courses_grade_average() * 2) + (11 - student.getPriority()) +
student.getBathmos());
        }
    }
}
```

Παράρτημα 12.

```
private final int number_of_shuffles = 50000;

public Distribution_Final_Result start(List<Choices_Of_Student> choices_of_students, List<Students_Per_Subject>
students_for_each_subject) {
    start_time = System.nanoTime();
    this.choices_of_students = choices_of_students;
    this.students_for_each_subject = students_for_each_subject;
    for (int i = 0; i < number_of_shuffles; i++) {
        randomly_Sort_The_List();
        start_Distributing();
        handle_Results();
        results_data.clear();
    }

    return find_Best_Result();
}



private void randomly_Sort_The_List() {
    Collections.shuffle(students_for_each_subject);
}
```

Παράρτημα 13.

```
if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));

}

private Students_Per_Subject recycleTheSubject(Result_Data to_recycle) {
    results_data.remove(to_recycle);
    return result_Data_To_Students_Per_Subject(to_recycle);
}

private Students_Per_Subject result_Data_To_Students_Per_Subject(Result_Data to_recycle) {
    return students_for_each_subject.stream()
            .filter(subject -> subject.getSubject().getId() == to_recycle.getAssigned_subject().getId())
            .findFirst().get();
}
```

Παράρτημα 14.

```
private void start_Distributing() {
    List<Students_Per_Subject> copy = students_for_each_subject;
    while (!copy.isEmpty()) {
        List<Students_Per_Subject> recycled = new ArrayList<>();
        for (Students_Per_Subject contended_subject : copy) {
            for (My_Student contender : contended_subject.getContenders()) {
                Optional<Result_Data> previous_pairing = find_The_Previous_Pairing(contender);
                int tmp = results_data.size();
                switch (identify_Case_Student_Subject(contended_subject, contender)) {
                    case 0:
                        if (!other_Student_need_Subject_More(find_Remaining_Contenders(contended_subject, contender),
contended_subject.getSubject()))
                            recycled.addAll(default_Situation(previous_pairing, contended_subject, contender));
                        break;
                    case 1:
                        if (previous_pairing.isPresent() &&
!subject_Has_No_Other_Student_To_Boolean(find_Remaining_Contenders(result_Data_To_Students_Per_Subject(previous_p
airing.get()), contender)))
                            previous_pairing.get().setScore(-999);
                        recycled.addAll(default_Situation(previous_pairing, contended_subject, contender));
                        break;
                    case 2:
                        if (!previous_pairing.isPresent() ||
!other_Student_need_Subject_More(find_Remaining_Contenders(contended_subject, contender),
contended_subject.getSubject()))
                            recycled.addAll(default_Situation(previous_pairing, contended_subject, contender));
                        break;
                    case 3:
                        recycled.addAll(default_Situation(previous_pairing, contended_subject, contender));
                        break;
                }
                if(tmp < results_data.size())
                    break;
            }
        }
        copy = recycled;
        recycled.clear();
    }
    results_data.sort((a, b) -> Double.compare(b.getStudent().getBathmos(), a.getStudent().getBathmos()));
}

private int identify_Case_Student_Subject(Students_Per_Subject subject, My_Student original_contender) {
    return subject_Has_No_Other_Student(find_Remaining_Contenders(subject, original_contender)) +
            student_Has_No_Other_Subject(find_Remaining_Choices(original_contender, subject.getSubject()));
}
private List<My_Student> find_Remaining_Contenders(Students_Per_Subject subject, My_Student original_contender) {
    return subject.getContenders().stream()
            .filter(contender -> !contender.getAm().equals(original_contender.getAm()))
            .collect(Collectors.toList());
```

```
}

private List<My_Dissertation> find_Remaining_Choices(My_Student student, DB_Dissertation subject) {
    Choices_Of_Student student_choices = choices_of_students.stream()
            .filter(choice -> choice.getStudent().getAm().equals(student.getAm()))
            .findFirst().get();
    return student_choices.getSubjectchoices().stream()
            .filter(remaining_choice -> !(remaining_choice.getId() == subject.getId()))
            .collect(Collectors.toList());
}

private int subject_Has_No_Other_Student(List<My_Student> remaining_contenders) {
    if (remaining_contenders.isEmpty())
        return 1;

    for (My_Student remaining_contender : remaining_contenders) {
        if (results_data.stream().noneMatch(result -> result.getStudent().getAm().equals(remaining_contender.getAm()) ||
                result.getStudent2().getAm().equals(remaining_contender.getAm())))
            return 0;
    }
    return 1;
}

private int student_Has_No_Other_Subject(List<My_Dissertation> remaining_subjects) {
    if (remaining_subjects.isEmpty())
        return 2;

    for (My_Dissertation remaining_subject : remaining_subjects) {
        if (results_data.stream().noneMatch(result -> result.getAssigned_subject().getId() == remaining_subject.getId()))
            return 0;
    }
    return 2;
}

private Optional<Result_Data> find_The_Previous_Pairing(My_Student contender) {
    return results_data.stream()
            .filter(result -> result.getStudent().getAm().equals(contender.getAm()) ||
                    result.getStudent2().getAm().equals(contender.getAm()))
            .findFirst();
}

private boolean other_Student_need_Subject_More(List<My_Student> remaining_contenders, DB_Dissertation subject) {
    for (My_Student remaining_contender : remaining_contenders)
        if (student_Has_No_Other_Subject_To_Boolean(find_Remaining_Choices(remaining_contender, subject)))
            return true;
    return false;
}
```

Παράρτημα 15.

```
private List<Students_Per_Subject> default_Situation(Optional<Result_Data> previous_pairing, Students_Per_Subject
contended_subject, My_Student contender) {
    List<Students_Per_Subject> recycled = new ArrayList<>();
    Optional<Result_Data> student2_previous_pairing;
    switch (identify_Case_Current_Previous(previous_pairing, contender)) {
        case 0:
            if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                        contender.getPriority(), contender.getTotal_score(), contender.getStudent2().orElse(null)));
            }
            break;

        case 1:
            student2_previous_pairing = find_The_Previous_Pairing(contender.getStudent2().get());
            if (student2_previous_pairing.isPresent()) {
                if (!student2_previous_pairing.get().getIs_group()) {
                    if
(subject_Has_No_Other_Student_To_Boolean(find_Remaining_Contenders(result_Data_To_Students_Per_Subject(student2_p
revious_pairing.get()), contender.getStudent2().get()))) {
                        if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                            previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                            student2_previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                            results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                                    contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) /
2), contender.getStudent2().orElse(null)));
                        }
                    } else {
                        previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                        student2_previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                        results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                                contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                    }
                }
            } else {
                if (student_Has_No_Other_Subject_To_Boolean(find_Remaining_Choices(contender.getStudent2().get(),
contended_subject.getSubject()))) {
                    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                            contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                } else {
                    if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                        previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                        results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                                contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                    }
                }
            }
            break;

        case 2:
            if (!student_Has_No_Other_Subject_To_Boolean(find_Remaining_Choices(previous_pairing.get().getStudent2(),
previous_pairing.get().getAssigned_subject()))) {
                if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                            contender.getPriority(), contender.getTotal_score(), contender.getStudent2().orElse(null)));
                }
            }
            break;

        case 3:
            student2_previous_pairing = find_The_Previous_Pairing(contender.getStudent2().get());
            switch (identify_Case_Current_Previous2(student2_previous_pairing, contender.getStudent2().get(),
previous_pairing.get(), previous_pairing.get().getStudent2())) {
```

```
            case 99:
                break;
            case 10:
            case 12:
                if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    student2_previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                            contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                }
                break;
            case 11:
                if
(subject_Has_No_Other_Student_To_Boolean(find_Remaining_Contenders(result_Data_To_Students_Per_Subject(student2_p
revious_pairing.get()), contender.getStudent2().get()))) {
                    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    student2_previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                            contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                }
                break;
            case 0:
                previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                        contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));

                break;
            case 2:
                if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                            contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                }
                break;
        }
        break;

    case 4:
        if (!student_Has_No_Other_Subject_To_Boolean(find_Remaining_Choices((My_Student)
previous_pairing.get().getStudent(), previous_pairing.get().getAssigned_subject()))) {
            if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                        contender.getPriority(), contender.getTotal_score(), contender.getStudent2().orElse(null)));
            }
        }
        break;

    case 5:
        student2_previous_pairing = find_The_Previous_Pairing(contender.getStudent2().get());
        switch (identify_Case_Current2_Previous2(student2_previous_pairing, contender.getStudent2().get(),
previous_pairing.get(), (My_Student) previous_pairing.get().getStudent())) {
            case 99:
                break;
            case 10:
            case 12:
                if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    student2_previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                            contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                }
                break;
```

```
            case 11:
if(subject_Has_No_Other_Student_To_Boolean(find_Remaining_Contenders(result_Data_To_Students_Per_Subject(student2_
previous_pairing.get()), contender.getStudent2().get()))) {
                    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    student2_previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                            contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                }
                break;
            case 0:
                previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                        contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));

                break;
            case 2:
                if (this_Subject_Better_Than_Previous(previous_pairing, contender)) {
                    previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                    results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                            contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
                }
                break;
        }
        break;

      case 99:
        if (contender.getStudent2().isPresent()) {
            student2_previous_pairing = find_The_Previous_Pairing(contender.getStudent2().get());
            if (student2_previous_pairing.isPresent() && !student2_previous_pairing.get().getIs_group()) {
                student2_previous_pairing.ifPresent(pairing -> recycled.add(recycleTheSubject(pairing)));
                results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                        contender.getPriority(), ((contender.getTotal_score() + contender.getStudent2().get().getTotal_score()) / 2),
contender.getStudent2().orElse(null)));
            }
        } else
            results_data.add(new Result_Data(contender, contended_subject.getSubject(),
                    contender.getPriority(), contender.getTotal_score(), contender.getStudent2().orElse(null)));
        break;
    }
    return recycled;
}

private int identify_Case_Current_Previous(Optional<Result_Data> previous_pairing, My_Student contender) {
    if (!previous_pairing.isPresent())
        return 99;
    return current_Is_Group(contender) + previous_Is_Group(previous_pairing.get(), contender.getAm());
}

private int current_Is_Group(My_Student contender) {
    if (contender.getStudent2().isPresent())
        return 1;
    return 0;
}
private int previous_Is_Group(Result_Data previous_pairing, String contender_Am) {
    if (!previous_pairing.getIs_group())
        return 0;
    if (previous_pairing.getStudent().getAm().equals(contender_Am))
        return 2;
    return 4;
}
private boolean this_Subject_Better_Than_Previous(Optional<Result_Data> previous_pairing, My_Student contender) {
    if (previous_pairing.get().getScore() < contender.getTotal_score())
        return true;
    return (previous_pairing.get().getScore() == contender.getTotal_score() && previous_pairing.get().getPriority() >
contender.getPriority());
}
```

```
private int identify_Case_Current2_Previous2(Optional<Result_Data> student2_previous_pairing, My_Student student_2,
Result_Data previous_pairing, My_Student previous_pairing_student2) {
    if (student2_previous_pairing.isPresent()) {
        if (student2_previous_pairing.get().getIs_group())
            return 99;
        return 10 +
subject_Has_No_Other_Student(find_Remaining_Contenders(result_Data_To_Students_Per_Subject(student2_previous_pairin
g.get()), student_2)) +

subject_Has_No_Other_Student(find_Remaining_Contenders(result_Data_To_Students_Per_Subject(previous_pairing),
previous_pairing_student2));
    } else {
        return student_Has_No_Other_Subject(find_Remaining_Choices(previous_pairing_student2,
previous_pairing.getAssigned_subject()));
    }
}

private Students_Per_Subject recycleTheSubject(Result_Data to_recycle) {
    results_data.remove(to_recycle);
    return result_Data_To_Students_Per_Subject(to_recycle);
}

private Students_Per_Subject result_Data_To_Students_Per_Subject(Result_Data to_recycle) {
    return students_for_each_subject.stream()
            .filter(subject -> subject.getSubject().getId() == to_recycle.getAssigned_subject().getId())
            .findFirst().get();
}
```

Παράρτημα 16.

```
private void start_Distributing() {
    List<Students_Per_Subject> copy = students_for_each_subject;
    while (!copy.isEmpty()) {
        List<Students_Per_Subject> recycled = new ArrayList<>();
        for (Students_Per_Subject contended_subject : copy) {
            for (My_Student contender : contended_subject.getContenders()) {
                Optional<Result_Data> previous_pairing = find_The_Previous_Pairing(contender);
                int tmp = results_data.size();

                …….

                if(tmp < results_data.size())
                    break;
            }
        }
        copy = recycled;
        recycled.clear();
    }
    results_data.sort((a, b) -> Double.compare(b.getStudent().getBathmos(), a.getStudent().getBathmos()));
}
```

Παράρτημα 17.

```
private void handle_Results() {
    int number_of_assigned_subjects = results_data.size();
    long number_of_students_with_subject = results_data.stream().filter(result_data -> result_data.getIs_group()).count() +
number_of_assigned_subjects;
    double score = 0;
    List<Integer> assigned_subjects_per_priority = new ArrayList<>();
    int score_weighted_by_priority = 0;
    double average_priority_assigned;
    List<Result_Data> for_count;
    for (int count = 0; count < 5; count++) {
        int count2 = count + 1;
        for_count = results_data.stream().filter(result_data -> result_data.getPriority() == count2).collect(Collectors.toList());
        if (for_count.isEmpty())
            assigned_subjects_per_priority.add(0);
        else
            assigned_subjects_per_priority.add(for_count.size());
    }
    int index = 0;
    for (Integer assigned_subject_per_priority : assigned_subjects_per_priority) {
        score_weighted_by_priority = score_weighted_by_priority + assigned_subject_per_priority * (5 - index);
        index++;
    }
    for (Result_Data result_data : results_data){
        score += result_data.getScore();
    }
    average_priority_assigned = (double) score_weighted_by_priority / number_of_assigned_subjects;
    average_priority_assigned = 6 - average_priority_assigned;

    List<Result_Data> temp = new ArrayList<>();
    temp.addAll(results_data);

    Distribution_Final_Result thisloopresult = new Distribution_Final_Result(temp, number_of_students_with_subject,
number_of_assigned_subjects, score, score_weighted_by_priority, average_priority_assigned, assigned_subjects_per_priority);
    final_results.add(thisloopresult);
}
```

Παράρτημα 18.

```
private Distribution_Final_Result find_Best_Result() {
    double threshold = 0.01;
    final_results.sort((a, b) -> Double.compare(b.getScore(), a.getScore()));
    double max_score = final_results.get(0).getScore();
    List<Distribution_Final_Result> final_results_max_score;
    final_results_max_score = final_results.stream()
            .filter(result -> Math.abs(result.getScore() - max_score) < threshold).collect(Collectors.toList());
    final_results_max_score.sort((a, b) -> Long.compare(b.getNumber_of_students(), a.getNumber_of_students()));
    long finishtime = System.nanoTime();
    final_results_max_score.get(0).setDuration((double) (finishtime - start_time) / 1000000000);
    return final_results_max_score.get(0);
}
```

Παράρτημα 19.

```
private void handle_Final_Result() {
    csv_handler.write_Result_To_CSV(final_result, start_metrics);
}

public void write_Result_To_CSV(Distribution_Final_Result final_result, Start_Metrics start_metrics) {
    find_Or_Create_Directory(parent_directory + "/" + this.current_semester + "/" + output_directory_name);
    update_Or_Create_Result(final_result.getResult_data());
    update_Or_Create_Metrics(final_result, start_metrics);
}

private void update_Or_Create_Result(List<Result_Data> results_data) {
    Path results_file = Paths.get(parent_directory + "/" + current_semester + "/" + output_directory_name + "/" +
results_file_name);
    if (results_file.toFile().exists())
        results_file.toFile().delete();
    try {
        BufferedWriter writer = Files.newBufferedWriter(results_file);

        CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT
                .withHeader("Arithmos_Mitroou_1\t", "Eponimo_1\t", "Onoma_1\t", "ID_Ptixiakhs\t", "Titlos_Ptixiakhs\t", "Kathigitis\t",
                        "Priority\t", "Score\t", "Is_Group", "Arithmos_Mitroou_2\t", "Eponimo_2\t", "Onoma_2\t"));
        for (Result_Data result_data : results_data) {
            if (result_data.getIs_group())
                csvPrinter.printRecord(result_data.getStudent().getAm() + "\t", result_data.getStudent().getLast_name() + "\t",
result_data.getStudent().getFirst_name() + "\t",
                        Integer.toString(result_data.getAssigned_subject().getId()) + "\t", result_data.getAssigned_subject().getName() +
"\t", result_data.getAssigned_subject().getSupervisor() + "\t",
                        Integer.toString(result_data.getPriority()) + "\t", Double.toString(result_data.getScore()) + "\t",
Boolean.toString(result_data.getIs_group()) + "\t",
                        result_data.getStudent2().getAm() + "\t", result_data.getStudent2().getLast_name() + "\t",
result_data.getStudent2().getFirst_name() + "\t");
            else
                csvPrinter.printRecord(result_data.getStudent().getAm() + "\t", result_data.getStudent().getLast_name() + "\t",
result_data.getStudent().getFirst_name() + "\t",
                        Integer.toString(result_data.getAssigned_subject().getId()) + "\t", result_data.getAssigned_subject().getName() +
"\t", result_data.getAssigned_subject().getSupervisor() + "\t",
                        Integer.toString(result_data.getPriority()) + "\t", Double.toString(result_data.getScore()) + "\t",
Boolean.toString(result_data.getIs_group()) + "\t", "", "", "");
        }
        csvPrinter.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
private void update_Or_Create_Metrics(Distribution_Final_Result final_result, Start_Metrics start_metrics) {
    Path results_file = Paths.get(parent_directory + "/" + current_semester + "/" + output_directory_name + "/" +
metrics_file_name);
    if (results_file.toFile().exists())
        results_file.toFile().delete();
    try {
        BufferedWriter writer = Files.newBufferedWriter(results_file);

        CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT
                .withHeader("Arithmos_Mathiton\t", "Arithmos_Thematon\t", "Arithmos_Diekdikomenon_Thematon\t",
"Arithmos_Azititon_Thematon\t", "Arithmos_Mathiton_Pou_Piran_Thema\t", "Arithmos_Thematon_Pou_Dothikan\t","Score\t",
"Priority_Score\t", "Score_Ratio\t", "Arithmos_Protereotita_1\t", "Arithmos_Protereotita_2\t", "Arithmos_Protereotita_3\t",
"Arithmos_Protereotita_4\t", "Arithmos_Protereotita_5\t"));
        List<String> tmp = new ArrayList<>();
        tmp.addAll(start_metrics.toCSV());
        tmp.addAll(final_result.toCSV());
        csvPrinter.printRecord(tmp);
        csvPrinter.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```