ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΩΝ ΣΥΣΤΑΣΕΩΝ ΜΕ JAVA STREAMS

Διπλωματική Εργασία
της

Σοφίας Νίνου

Θεσσαλονίκη, 2022

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΩΝ ΣΥΣΤΑΣΕΩΝ
ΜΕ JAVA STREAMS

Σοφία Νίνου
Πτυχίο Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας, 2016

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Κωνσταντίνος Μαργαρίτης

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την

....

Περίληψη

Το συναρτησιακό πρότυπο προγραμματισμού κερδίζει όλο και περισσότερο έδαφος μεταξύ των προγραμματιστών λόγω της απλούστευσης στον τρόπο διατύπωσης που προσφέρει. Επιπλέον, η αύξηση του όγκου των δεδομένων καθιστά αναγκαία την αποτελεσματική επεξεργασία και προσπέλασή τους. Ολοένα και περισσότερες γλώσσες προγραμματισμού που ακολουθούν το προστακτικό ή το αντικειμενοστραφές πρότυπο ενσωματώνουν χαρακτηριστικά συναρτησιακού προγραμματισμού, ανάμεσα σε αυτές και η Java. Με την έκδοση 8 της γλώσσας, προστέθηκαν σημαντικά χαρακτηριστικά, που δίνουν την δυνατότητα σχηματισμού συναρτησιακών προτάσεων, επεξεργασία των δεδομένων με την μορφή ερωτημάτων, καθώς και ευκολότερη και πιο αδιαφανή παραλληλοποίηση των λειτουργιών.

Λέξεις Κλειδιά: Java 8, Stream API, συστήματα συστάσεων, συναρτησιακός προγραμματισμός

Abstract

Functional style programming is becoming more and more popular among programmers because it simplifies the way of coding. Moreover, the continuous rise in the available data makes effective data processing a necessity. More and more programming languages that follow the declarative or the object-oriented paradigm tend to integrate functional features, one of them being Java. In Java's version 8, important features have been added that make it possible to write code in functional style, to process data in the form of database queries, as well as to parallelize processes in a non-transparent way.

Keywords: Java 8, Stream API, recommender systems, functional programming

Περιεχόμενα

Ευρετήριο Διαγραμμάτων

Αποσπάσματα Κώδικα

Ευρετήριο Πινάκων

1 Εισαγωγή

Η συνεχόμενη και γρήγορη ανάπτυξη τόσο του λογισμικού όσο και του υλικού των ηλεκτρονικών υπολογιστών οδηγεί σε συνεχόμενη αλλαγή των απαιτήσεων. Οι υπολογιστές αποκτούν περισσότερους επεξεργαστές ενώ η υπολογιστή δύναμη ενισχύεται πια και από τις κάρτες γραφικών. Αυτό καθιστά την εκμετάλλευση όλων των υλικών δυνατοτήτων μέσω του λογισμικού απαραίτητη, καθιστώντας τον παράλληλο υπολογισμό απαραίτητο. Επιπλέον, η επεξεργασία των δεδομένων γίνεται όλο και πιο εντατική, με τομείς όπως η ανάλυση δεδομένων (data analytics) να κερδίζουν όλο και περισσότερο έδαφος. Ο προγραμματισμός βασίζεται σε μεγάλο βαθμό στην ανάκτηση και την επεξεργασία δεδομένων, και πρότυπα όπως αυτά των ερωτημάτων των βάσεων δεδομένων, γίνονται πιο δημοφιλή, με αποτέλεσμα ο συναρτησιακός προγραμματισμός στις ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμούς να γίνεται όλο και πιο απαραίτητος.

Βασική αρχή του συναρτησιακού προγραμματισμού είναι πως δεν ορίζουμε το πώς θα γίνει μία λειτουργία, αλλά το ποια λειτουργία ακριβώς θέλουμε να πραγματοποιηθεί. Αυτό, αν και εκ πρώτοις μπορεί να μοιάζει περίπλοκο σε έναν προγραμματιστή που έχει μάθει το προστακτικό μοντέλο, στην πραγματικότητα απλοποιεί κατά πολύ την διαδικασία του προγραμματισμού. Γλώσσες προγραμματισμού όπως η Java, έχουν δημιουργηθεί με βάση το αντικειμενοστραφές μοντέλο, ωστόσο οι αλλαγές στις απαιτήσεις οδήγησαν στην ενσωμάτωση στην γλώσσα στοιχείων του συναρτησιακού προγραμματισμού. Η πρώτη μεγάλη αλλαγή προς αυτή την κατεύθυνση έγινε στην έκδοση 8 της Java, η οποία περιλαμβάνει πολλά συναρτησιακά χαρακτηριστικά, μεταξύ άλλων και το Stream API.

Το δεύτερο σκέλος της εργασίας, αποτελείται από τα συστήματα συστάσεων. Ο όγκος των πληροφοριών που διακινούνται, ιδιαιτέρως στο διαδίκτυο, καθιστά τα συστήματα συστάσεων απαραίτητα, προκείμενου οι χρήστες να μπορούν να επιλέγουν προϊόντα και αντικείμενα, μεταξύ της πληθώρας που υπάρχουν. Στόχος των συστημάτων συστάσεων είναι να προτείνουν στους χρήστες αντικείμενα σχετικά με τα ενδιαφέροντα και τις προτιμήσεις τους, ελαττώνοντας έτσι την πιθανή σύγχυση που μπορεί να προκληθεί από την υπερπληθώρα πληροφοριών.

Η παρούσα εργασία χωρίζεται σε δύο μέρη. Το πρώτο είναι η μελέτη του Stream API της έκδοσης 8 της γλώσσας Java, και το δεύτερο είναι ο προγραμματισμός ενός συστήματος συστάσεων χρησιμοποιώντας τα streams. Ο γενικότερος σκοπός είναι η

μελέτη και η κατανόηση του χειρισμού και του τρόπου λειτουργίας των streams, μέσα από τον προγραμματισμό ορισμένων αλγορίθμων συστημάτων συστάσεων.

1.1 Διάρθρωση της μελέτης

Η παρούσα μελέτη ξεκινάει με μία ανασκόπηση του συναρτησιακού προγραμματισμού. Στο 2ο κεφάλαιο, αναλύονται οι βασικές αρχές του καθώς και ορισμένες διαφορές του με τον αντικειμενοστραφή προγραμματισμό, ενώ στην συνέχεια παρουσιάζονται τα χαρακτηριστικά που προστέθηκαν στην έκδοση 8 της Java, έτσι ώστε να μπορεί να ακολουθεί ως έναν βαθμό και το συναρτησιακό μοντέλο.

Στο 3ο κεφάλαιο, εμβαθύνεται η μελέτη του Stream API, και περιγράφεται αναλυτικά και με συγκεκριμένα παραδείγματα ο τρόπος χρήσης των βασικότερων συναρτήσεων που προσφέρει.

Στο 4ο κεφάλαιο, εισάγονται οι έννοιες των συστημάτων συστάσεων, παρουσιάζοντας ορισμένους βασικούς αλγορίθμους. Το κυρίως μέρος αυτούς του κεφαλαίου είναι η υλοποίηση ενός συστήματος συστάσεων χρησιμοποιώντας Java streams, στο οποίο περιγράφονται αναλυτικά οι αλγόριθμοι καθώς και ο τρόπος υλοποίησής τους.

Στο 5ο κεφάλαιο, συνοψίζονται τα συμπεράσματα της μελέτης και προτείνονται μελλοντικές επεκτάσεις.

2 Συναρτησιακός Προγραμματισμός

2.1 Γενικά για τον συναρτησιακό προγραμματισμό

Ο συναρτησιακός προγραμματισμός είναι ένα πρότυπο δηλωτικού προγραμματισμού, κατά το οποίο τα προγράμματα δημιουργούνται από την σύνθεση και την εφαρμογή συναρτήσεων [1]. Οι δηλώσεις (statements), στον συναρτησιακό προγραμματισμό αντικαθίστανται από διαδοχικές συναρτήσεις, κάθε μία από τις οποίες μπορεί να παίρνει ως είσοδο ένα όρισμα και να επιστρέφει κάποια έξοδο χωρίς ωστόσο να επηρεάζει την κατάσταση του προγράμματος. Ιστορικά, το πρότυπο βασίζεται στον λάμδα λογισμό των μαθηματικών και η πρώτη γλώσσα υψηλού επιπέδου για συναρτησιακό προγραμματισμό ήταν η LISP, η οποία δημιουργήθηκε στο τέλος της δεκαετίας 1950.

Τα πρότυπα προγραμματισμού μπορούν να χωριστούν σε προστακτικά (imperative) και δηλωτικά (declarative). Στην περίπτωση του προστακτικού προτύπου, το πρόγραμμα αποτελείται από ένα σύνολο διαδοχικών δηλώσεων (statements), οι οποίες αλλάζουν την κατάσταση του προγράμματος. Ένας τύπος προστακτικού προγραμματισμού είναι ο διαδικαστικός προγραμματισμός, στον οποίο χρησιμοποιούνται διαδικασίες και υποδιαδικασίες, ενώ ο αντικειμενοστρεφής προγραμματισμός είναι μία επέκταση του διαδικαστικού [2]. Αντίθετα, στον δηλωτικό προγραμματισμό η λογική των υπολογισμών εκφράζεται χωρίς να περιγράφεται η ροή του προγράμματος με την μορφή διαδοχικών εντολών και ο στόχος είναι να περιγραφεί τί πρέπει να επιτύχει το πρόγραμμα παρά το πώς θα το επιτύχει.

Υπάρχουν αρκετές αμιγώς συναρτησιακές γλώσσες προγραμματισμού, μερικές από τις πιο γνωστές είναι οι Haskell, SML, Clojure, Erlang, κ.α. Ωστόσο πολλές μη συναρτησιακές γλώσσες προγραμματισμού, όπως οι Java, JavaScript και Python, παρέχουν πλέον εργαλεία για να υποστηρίζουν το συναρτησιακό προγραμματιστικό πρότυπο. Ένα από αυτά είναι τα Java Streams, τα οποία μελετώνται στην παρούσα εργασία.

2.1.1 Βασικές Αρχές & Τεχνικές

Παρακάτω περιγράφονται οι βασικές αρχές καθώς και ορισμένες τεχνικές που εφαρμόζονται στον συναρτησιακό προγραμματισμό. Πολλά από τα παρακάτω αποτελούν τεχνικές που χρησιμοποιούνται και σε μη αμιγώς συναρτησιακές γλώσσες.

1. Αμετάβλητα δεδομένα (*Data Immutability*):

Τα δεδομένα που δίνονται ως είσοδο στις συναρτήσεις πρέπει να είναι αμετάβλητα. Σε ένα πρόγραμμα που ακολουθεί το συναρτησιακό πρότυπο, τα δεδομένα δεν πρέπει να μπορούν να μεταβληθούν μετά την αρχικοποίησή τους. Στις γλώσσες προγραμματισμού που είναι αμιγώς συναρτησιακές, αυτή η ιδιότητα υποστηρίζεται σε επίπεδο γλώσσας.

2. «Καθαρές» συναρτήσεις (*Pure functions*):
   Για να θεωρείται μία συνάρτηση «καθαρή» (pure) πρέπει να έχει τις εξής δύο ιδιότητες: 1) να μην προκαλεί καμία παρενέργεια (no side-effects) και 2) να παράγει τα ίδια ακριβώς αποτελέσματα για συγκεκριμένη είσοδο. Ως παρενέργεια, θεωρείται οποιαδήποτε αλλαγή στα δεδομένα εισόδου ή στην κατάσταση του προγράμματος.
3. Συναρτήσεις ανώτερης τάξης (*Higher order functions*):
   Οι συναρτήσεις ανώτερης τάξης είναι αυτές που μπορούν να λάβουν ως όρισμα άλλες συναρτήσεις αλλά και να έχουν ως τύπο επιστροφής συνάρτηση. Αυτή η ιδιότητα επιτρέπει την σύνθεση συναρτήσεων στον συναρτησιακό προγραμματισμό.
4. Αναφορική διαφάνεια (*Referential Transparency*):
   Μία συνάρτηση έχει την ιδιότητα της αναφορικής διαφάνειας, εάν για μία συγκεκριμένη είσοδο παράγει πάντα την ίδια έξοδο. Επιπλέον, λέμε ότι ο κώδικας έχει αναφορική διαφάνεια εάν δεν μεταβάλλει τα δεδομένα και δεν εξαρτάται από τον εξωτερικό κόσμο για να λειτουργήσει. [3]
5. Αναδρομή (*Recursion*):
   Ως αναδρομικές ορίζονται οι συναρτήσεις οι οποίες καλούν τον εαυτό τους. Στις υλοποιήσεις αναδρομικών συναρτήσεων σε μία γλώσσα προγραμματισμού, κάποιοι όροι υπολογίζονται καλώντας την ίδια συνάρτηση με διαφορετικά ορίσματα. Η αναδρομική συνάρτηση καλεί τον εαυτό της επαναλαμβανόμενα, έως ότου φτάσει στην περίπτωση βάσης, απ' όπου και επιστρέφεται μία τιμή. Η τιμή που επιστρέφεται «προχωράει» προς τα πίσω, στις προηγούμενες αναδρομικές κλήσεις της συνάρτησης μέχρι να υπολογιστεί το τελικό αποτέλεσμα. Στον προστακτικό προγραμματισμό, η επανάληψη για την προσπέλαση συλλογών γίνεται μέσω βρόχων (π.χ. for ή while), οι οποίοι εξαρτώνται από μία συγκεκριμένη κατάσταση (state) του προγράμματος, καθώς χρησιμοποιούν συνήθως μία μεταβλητή μετρητή. Επιπλέον, μπορεί να

αλλάζουν τις τιμές των δεδομένων της συλλογής. Για τους παραπάνω λόγους, στον συναρτησιακό προγραμματισμό, αποφεύγονται οι βρόχοι και αντ' αυτών χρησιμοποιείται η αναδρομή ως τεχνική επανάληψης. [4]

2.1.2 Πλεονεκτήματα

Εξαιτίας της ντετερμινιστικής φύσης του συναρτησιακού προγραμματισμού (πάντα ίδια αποτελέσματα για συγκεκριμένη είσοδο), τα προγράμματα που είναι γραμμένα με βάση αυτό το πρότυπο παρουσιάζουν μία σειρά από πλεονεκτήματα. Το σημαντικότερο πλεονέκτημα, προκύπτει ακριβώς από τις καθαρές συναρτήσεις και τα αμετάβλητα δεδομένα, ιδιότητες που κάνουν τα συναρτησιακά προγράμματα εύκολα στην συντήρηση, την κατανόηση, τις δοκιμές (testing) και την αποσφαλμάτωση.

Τα συναρτησιακά προγράμματα είναι πιο αρθρωτά και αυτόνομα. Οι συναρτήσεις είναι γραμμένες με τέτοιον τρόπο, ώστε να μην εξαρτώνται από το εξωτερικό περιβάλλον αλλά ούτε και να το μεταβάλλουν, και να μπορούν να λειτουργήσουν με καμία, μία ή περισσότερες εισόδους, ανεξάρτητα από το υπόλοιπο πρόγραμμα. Δεν υπάρχουν παρενέργειες που πρέπει να ληφθούν υπόψιν, ούτε εξαιρέσεις να διαχειριστούν. Έτσι, γίνεται ευκολότερη και η διαδοχική σύνθεση συναρτήσεων. Ένα πρόγραμμα μπορεί να ξεκινήσει από μία ή περισσότερες συναρτήσεις, οι οποίες στην συνέχεια συνδυάζονται για να προκύψουν άλλες, υψηλότερου επιπέδου, μέχρι να καταλήξουν σε μία συνδυαστική συνάρτηση που αποτελεί το πρόγραμμα.

Τέλος, τα συναρτησιακά προγράμματα είναι ασφαλή στην παράλληλη επεξεργασία. Εφόσον τα μοιραζόμενα δεδομένα είναι αμετάβλητα, καμία συναρτησιακή λειτουργία δεν μπορεί να τα τροποποιήσει. Συνεπώς, στον παραλληλισμό ενός προγράμματος, δεν χρειάζεται ο προγραμματιστής να προσέξει για ταυτόχρονες προσπελάσεις και τροποποιήσεις των δεδομένων. [3] [5]

2.2 Συναρτησιακός και Αντικειμενοστρεφής Προγραμματισμός

Ο συναρτησιακός προγραμματισμός, που είναι τύπος του δηλωτικού, παρουσιάζει ορισμένες σημαντικές διαφορές με τον αντικειμενοστραφή, που αποτελεί τύπο προστακτικού προγραμματισμού. Η δομή των αμιγώς συναρτησιακών προγραμμάτων είναι εντελώς διαφορετική από αυτή των αμιγώς αντικειμενοστραφών. Ωστόσο, σε πολλές γλώσσες, όπως οι Java, python και JavaScript, τα δύο μοντέλα προγραμματισμού μπορούν να συνδυαστούν, έτσι ώστε να υπάρχει μεγαλύτερη ευελιξία και ο προγραμματιστής να επωφεληθεί από τις ιδιότητες τόσο του ενός όσο και του άλλου.

Η βασικότερη δομική διαφορά των δύο προτύπων, είναι η μοντελοποίηση του προβλήματος και κατ' επέκταση του προγράμματος. Στον αντικειμενοστραφή προγραμματισμό, όπως υποδεικνύει και το όνομά του, ο κώδικας περιστρέφεται γύρω από τα αντικείμενα. Τα αντικείμενα, τις περισσότερες φορές αντιπροσωπεύουν έννοιες του πραγματικού κόσμου και αποτυπώνονται στα προγράμματα με τις ιδιότητες και τα χαρακτηριστικά τους. Δομικά στοιχεία είναι τα αντικείμενα και οι μέθοδοι. Αντίθετα, ο συναρτησιακός προγραμματισμός βασίζεται στην έννοια των μαθηματικών συναρτήσεων. Τα προγράμματα δομούνται συνδυάζοντας διαδοχικές συναρτήσεις, μέχρις ότου να προκύψει το επιθυμητό αποτέλεσμα. Τα αντικείμενα αντικαθίστανται από τις μεταβλητές και οι μέθοδοι από τις συναρτήσεις. Στον συναρτησιακό προγραμματισμό αποθαρρύνεται ιδιαίτερα η χρήση καθολικών μεταβλητών (και σε ορισμένες γλώσσες δεν υπάρχουν καν ως επιλογή) γιατί, εκτός του ότι δυσκολεύουν την κατανόηση του προγράμματος, διακινδυνεύουν να παραβιαστεί η ιδιότητα των καθαρών συναρτήσεων.

Παρ' όλο που οι συναρτήσεις και οι μέθοδοι ως ορολογία περιγράφουν και οι δύο μία διαδικασία που θέλουμε να μπορεί να επαναληφθεί ξανά και ξανά μέσα σε ένα πρόγραμμα, υπάρχει μία σημαντική διαφορά στους δύο τύπους προγραμματισμού. Οι μέθοδοι του αντικειμενοστραφή προγραμματισμού, πολύ συχνά λαμβάνουν δεδομένα ως ορίσματα των οποίων τις τιμές μεταβάλλουν. Εξ' άλλου, σε πολλές αντικειμενοστραφείς γλώσσες προγραμματισμού, συμπεριλαμβανομένης της Java, τα αντικείμενα περνούν στις μεθόδους ορίσματα με αναφορά, και συνεπώς το περιεχόμενό τους μπορεί να αλλάξει. Αυτό αποτελεί παραβίαση της βασικής αρχής των καθαρών συναρτήσεων, της αναφορικής διαφάνειας και των αμετάβλητων δεδομένων του συναρτησιακού προγραμματισμού. Στον συναρτησιακό προγραμματισμό, όπως αναφέρθηκε προηγουμένως, αποτελεί απαράβατη προϋπόθεση η μη τροποποίηση των δεδομένων εισόδου στις συναρτήσεις.

Όσον αφορά στην ροή του προγράμματος, στον αντικειμενοστραφή προγραμματισμό, αυτή διενεργείται με την χρήση επαναληπτικών δομών (for και while) και δομών απόφασης (if, switch). Η σειρά εκτέλεσης των εντολών παίζει σημαντικό ρόλο στην τελική έκβαση του προγράμματος. Στον συναρτησιακό, από την άλλη πλευρά, το πρόγραμμα ρέει μέσω διαδοχικών κλήσεων συναρτήσεων και την σύνθεσή τους, αποφεύγονται οι βρόχοι επανάληψης, και προτιμάται η αναδρομή. Η συνέπεια στην σειρά εκτέλεσης των εντολών είναι πιο χαλαρή απ' ότι στον αντικειμενοστραφή ή σε κάποιον άλλο τύπο προστακτικού προγραμματισμού.

Σημαντικό ρόλο παίζει ο τρόπος σκέψης δόμησης της λύσης του προβλήματος στον αντικειμενοστραφή σε σχέση με τον συναρτησιακό προγραμματισμό. Στον αντικειμενοστραφή, καθώς και στον διαδικαστικό προγραμματισμό, για την προσέγγιση της λύσης του προβλήματος, ο προγραμματιστής οφείλει να σκεφτεί το πώς θα λύσει το πρόβλημα, και έπειτα να διαμορφώσει κατάλληλα τον κώδικα προς αυτή την κατεύθυνση. Στον συναρτησιακό προγραμματισμό εστιάζουμε περισσότερο στο ποιο είναι το πρόβλημα που πρέπει να λυθεί και όχι στον τρόπο με τον οποίο θα λυθεί και εφαρμόζονται οι κατάλληλες συναρτήσεις.

2.3 Συναρτησιακός προγραμματισμός στην Java

Η Java είναι μία αντικειμενοστραφής γλώσσα προγραμματισμού υψηλού επιπέδου. Δημιουργήθηκε το 1995 από την Sun (τώρα Oracle), με στόχο να είναι μία γλώσσα γενικής χρήσης με όσο το δυνατόν λιγότερες εξαρτήσεις, έτσι ώστε να λειτουργεί όσο το δυνατόν περισσότερες πλατφόρμες. Η σύνταξή της μοιάζει πολύ με αυτή της C και της C++, ωστόσο προσφέρει λιγότερες ευκολίες στις εντολές χαμηλού επιπέδου. Η τελευταία έκδοση (ως τον Δεκέμβριο του 2021) είναι η Java 17, ενώ υποστηρίζονται μακροπρόθεσμα (Long Term Support, LTS) και οι εκδόσεις 8 (2014) και 11 (2018). [6]

Μέχρι και την Java 7, η γλώσσα ακολουθούσε κατά κύριο λόγο το αντικειμενοστραφές πρότυπο, και δεν υποστήριζε συναρτησιακό προγραμματισμό. Η παραλληλοποίηση των προγραμμάτων έπρεπε να γίνει με την χρήση νημάτων ή διεργασιών, σχετικά χειροκίνητα από τον προγραμματιστή. Ο προγραμματιστής έπρεπε να φροντίσει για την σωστή παραλληλοποίηση, αφενός στην λογική του προγράμματος, έτσι ώστε το παράλληλο πρόγραμμα να παράγει τα ίδια αποτελέσματα με το ακολουθιακό, αφετέρου στην χρήση των μοιραζόμενων μεταβλητών, τον τρόπο προσπέλασής τους και άλλα ζητήματα που αφορούν την παραλληλοποίηση, όπως τα αδιέξοδα (deadlocks) ή οι λιμοκτονίες (starvations). Με την ανάπτυξη ωστόσο του υλικού των υπολογιστών και την προσθήκη περισσότερων επεξεργαστικών πυρήνων, τα προγράμματα που δεν χρησιμοποιούσαν αυτούς τους επιπλέον πόρους, δεν ήταν αποδοτικά. Η Java ως τότε, έκανε την παραλληλοποίηση προγραμμάτων μεγάλου μεγέθους, μία αρκετά επίπονη διαδικασία και τα συστήματα που γράφονταν σε Java έμεναν πίσω στο κομμάτι της αποδοτικής εκμετάλλευσης του διαθεσίμου υλικού.

Η Java 8 ήρθε να προσθέσει χαρακτηριστικά στην γλώσσα που βοηθούν την επίλυση του παραπάνω προβλήματος με μεγαλύτερη ευκολία. Εισήγαγε χαρακτηριστικά

που επιτρέπουν την συγγραφή κώδικα που υπακούει στις βασικές αρχές του συναρτησιακού προγραμματισμού, όπως οι εκφράσεις λάμδα (lambda expressions). Επιπλέον, με το Stream API και τις κλάσεις που αυτό παρέχει, απλούστευσε την παραλληλοποίηση των προγραμμάτων, και αποδέσμευσε τους προγραμματιστές από τον μονόδρομο ρητής χρήσης των νημάτων. Παρακάτω γίνεται μία ανασκόπηση των τρόπων με τους οποίους ο συναρτησιακός προγραμματισμός και οι βασικές αρχές του, υποστηρίζονται στην Java 8.

2.3.1 Αμετάβλητα Δεδομένα

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, στον συναρτησιακό προγραμματισμό απαραίτητη προϋπόθεση είναι να μην μεταβάλλονται τα δεδομένα εισόδου αλλά ούτε και η κατάσταση του προγράμματος. Στις αμιγώς συναρτησιακές γλώσσες, αυτή η απαίτηση καλύπτεται σε επίπεδο γλώσσας, όπου οι μεταβλητές δεν μπορούν να αλλάξουν τιμή μετά από την αρχικοποίησή τους. Στην Java, υπάρχουν ορισμένοι τύποι δεδομένων που είναι αμετάβλητοι, όπως ο τύπος String, όμως οι μεταβλητές με τους περισσότερους τύπους δεδομένων επιτρέπεται να αλλάξουν τιμή μετά την αρχικοποίησή τους.

Στην Java μπορούμε να πετύχουμε την αμεταβλητότητα των δεδομένων χρησιμοποιώντας την λέξη-κλειδί `final`. Δηλώνοντας μία μεταβλητή ως `final`, καθίσταται αδύνατη η αλλαγή της τιμής της. Αυτό όμως δεν αρκεί για να εξασφαλίσουμε την αμεταβλητότητα γενικότερα. Ένα πεδίο μπορεί να είναι μέρος μίας κλάσης, και κατά συνέπεια ενός αντικειμένου, που χρησιμοποιείται ως είσοδος σε μία συνάρτηση. Το ίδιο το αντικείμενο θα πρέπει και αυτό με τη σειρά του να είναι αμετάβλητο, άρα όλα του τα πεδία θα πρέπει να είναι δηλωμένα ως `final`. Στο παρακάτω παράδειγμα ορίζεται η κλάση Student, η οποία περιέχει τα πεδία name, am και credits. Τα στιγμιότυπα αυτής της κλάσης είναι αμετάβλητα, καθώς όλα τα πεδία της είναι δηλωμένα `final`:

```java
public class Student {

    private final String name;
    private final int am;
    private final double credits;

    public Student(String name, int am, double credits) {
        this.name = name;
        this.am = am;
        this.credits = credits;
    }

    public String getName() {
        return name;
    }

    public int getAm() {
        return am;
    }

    public double getCredits() {
        return credits;
    }
}
```

Απόσπασμα Κώδικα 2.1: Αμετάβλητα δεδομένα

Τα πεδία που είναι final πρέπει να εξασφαλιστεί ότι θα αρχικοποιηθούν με κάποιον τρόπο, στο παράδειγμα αυτό γίνεται μέσω του κατασκευαστή. Επίσης, παρέχονται μέθοδοι πρόσβασης (getters) αλλά όχι οι αντίστοιχες μέθοδοι τροποποίησης (setters) για ευνόητους λόγους.

2.3.2 Καθαρές Συναρτήσεις

Η αρχή της ενθυλάκωσης του αντικειμενοστραφούς προγραμματισμού είναι μία από αυτές που είναι εκ διαμέτρου αντίθετες με την έννοια των καθαρών συναρτήσεων. Από την μία πλευρά η αρχή της ενθυλάκωσης προτείνει ως καλή προγραμματιστική πρακτική την απόκρυψη της κατάστασης των αντικειμένων και την παροχή μεθόδων για την τροποποίησης τους. Από την άλλη πλευρά, ο συναρτησιακός προγραμματισμός έχει ως βασική αρχή την μη μεταβολή των δεδομένων εισόδου στις συναρτήσεις και την μη τροποποίηση της κατάστασης του προγράμματος από αυτές. Η αρχή της ενθυλάκωσης ωστόσο, είναι μία προτροπή και όχι απαραίτητος κανόνας για την συγγραφή προγραμμάτων. Συνεπώς, για να γράψει κανείς καθαρές συναρτήσεις στην Java, αρκεί να φροντίσει να μην μεταβάλει τα δεδομένα που πιθανώς περνιούνται ως ορίσματα και να μην έχει άλλες παρενέργειες, όπως η αλλαγή των τιμών καθολικών μεταβλητών. Ως παρενέργεια μίας συνάρτησης ακόμα, μπορεί να θεωρηθεί ακόμα και η εγγραφή σε κάποιο αρχείο ή σε μία βάση δεδομένων.

Η παρακάτω μέθοδος δεν θεωρείται καθαρή, διότι αλλάζει τις τιμές του πίνακα που λαμβάνει ως όρισμα:

```
public void increment(int[] numbers) {
    for(int i=0;i<numbers.length;i++) {
        numbers[i] ++;
    }
}
```

Απόσπασμα Κώδικα 2.2: Μη καθαρή συνάρτηση

Αντιθέτως, η επόμενη μέθοδος, επιστρέφει το άθροισμα των στοιχείων του πίνακα χωρίς να μεταβάλλει καθόλου τα δεδομένα του, και γι' αυτό θεωρείται καθαρή:

```
public int sum(ArrayList<Integer> numbers) {
    int sum = 0;
    for(Integer i : numbers)
        sum += i;
    return sum;
}
```

Απόσπασμα Κώδικα 2.3: Καθαρή συνάρτηση

Τέλος, μη καθαρές θεωρούνται και οι μέθοδοι οι οποίες ενδέχεται να δημιουργήσουν μία εξαίρεση, καθώς και αυτές προσμετρώνται ως παρενέργειες.

2.3.3 Συναρτήσεις Ανώτερης Τάξης

Μία γλώσσα προγραμματισμού αντιμετωπίζει τις συναρτήσεις ως «πολίτες πρώτης κατηγορίας» (first-class functions) εάν επιτρέπει σε αυτές να υποστηρίζουν όλες τις λειτουργίες που υποστηρίζει οποιαδήποτε άλλη οντότητα. Με άλλα λόγια, οι συναρτήσεις μπορούν να ανατίθενται ως τιμές σε μεταβλητές, να περνιούνται ως παράμετροι σε άλλες συναρτήσεις, ακόμα και να επιστρέφονται από άλλες συναρτήσεις.

Στην Java, οι «πολίτες πρώτης κατηγορίας» είναι τα αντικείμενα, μιας και πρόκειται για μία κατά βάση αντικειμενοστραφή γλώσσα προγραμματισμού. Μέχρι την Java 8, υπήρχαν περιορισμένες επιλογές που υποστήριζαν τις παραπάνω λειτουργίες, όπως είναι οι ανώνυμες εσωτερικές κλάσεις. Με αυτές είναι εφικτό να περαστούν ως ορίσματα σε συναρτήσεις, απευθείας αντικείμενα από κλάσεις που δημιουργούνται εκείνη την στιγμή, χωρίς να είναι απαραίτητο να έχουν οριστεί προηγουμένως. Χαρακτηριστικό παράδειγμα είναι η ανάθεση αντικειμένων Listener σε διάφορα γεγονότα. Στο παρακάτω κομμάτι κώδικα, προστίθεται σε ένα κουμπί ένας Listener, που προκύπτει από μία ανώνυμη εσωτερική κλάση:

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        ...
    }
});
```

Απόσπασμα Κώδικα 2.4: Ανώνυμη κλάση για ActionListener

Στην Java 8 ωστόσο, εισάγονται ορισμένα χαρακτηριστικά που αλλάζουν εντελώς τον τρόπο που αντιμετωπίζονται οι συναρτήσεις και κυρίως διευκολύνουν τον προγραμματιστή κάνοντας την παραμετροποίηση με συναρτήσεις πολύ πιο απλή και κατανοητή. Αυτά είναι οι εκφράσεις λάμδα (lambda expressions), οι αναφορές μεθόδων (method references) και οι συναρτησιακές διεπαφές (functional interfaces).

2.3.3.1 Lambda Expressions

Μία έκφραση λάμδα είναι μία ανώνυμη συνάρτηση που μπορεί να περαστεί ως παράμετρος σε άλλες συναρτήσεις (ή μεθόδους) αλλά και να ανατεθεί ως τιμή σε μεταβλητή. Συγκεκριμένα, λέμε ότι είναι ανώνυμη, γιατί δεν χρειάζεται να δηλωθεί με κάποιο ορισμένο όνομα όπως οι κανονικές συναρτήσεις, συνάρτηση (και όχι μέθοδος) γιατί δεν σχετίζεται με μία συγκεκριμένη κλάση (ούτε χρειάζεται ένα στιγμιότυπο κλάσης για να κληθεί) αλλά μπορεί να έχει, όπως και οι μέθοδοι, τύπο επιστροφής, λίστα ορισμάτων και σώμα [7].

Συντακτικά, μία lambda expression αποτελείται από τρία μέρη: 1) τις παραμέτρους, 2) ένα βέλος, 3) το σώμα της έκφρασης (που αντιστοιχεί στο σώμα της συνάρτησης) και συντάσσεται ως εξής:

```
(parameters) -> {lambda body};
```

Οι παράμετροι μπορεί να είναι καμία, μία ή περισσότερες ενώ το σώμα της έκφρασης πρέπει να βρίσκεται μέσα σε άγκιστρα μόνο εάν αποτελείται από παραπάνω από μία εντολές.

Το προηγούμενο παράδειγμα με τον ActionListener για το κουμπί, μπορεί να ξαναγραφεί χρησιμοποιώντας μία lambda expression αντί για την ανώνυμη κλάση ως εξής:

```
button.addActionListener(e -> {
    ...
});
```

Απόσπασμα Κώδικα 2.5: Έκφραση λάμδα αντί ανώνυμης κλάσης ActionListener

Ένα άλλο παράδειγμα στο οποίο φαίνεται πιο καθαρά η δύναμη των εκφράσεων λάμδα είναι η περίπτωση στην οποία θέλουμε να χρησιμοποιήσουμε έναν custom Comparator. Για παράδειγμα, για να ταξινομήσουμε μία λίστα που περιέχει αντικείμενα

τύπου Student με βάση τον αριθμό μητρώου τους (πεδίο am), μπορούμε να το κάνουμε, χρησιμοποιώντας μία ανώνυμη κλάση που υπερβαίνει (overrides) την μέθοδο compare, όπως φαίνεται παρακάτω:

```
students.sort(new Comparator<Student>() {
    @Override
    public int compare(Student s1, Student s2) {
        return s1.getAm().compareTo(s2.getAm());
    }
});
```

Απόσπασμα Κώδικα 2.6: Ανώνυμη κλάση για Comparator

Οι παραπάνω γραμμές κώδικα μπορούν πολύ εύκολα να μειωθούν και να απλουστευτούν χρησιμοποιώντας στην θέση της ανώνυμης κλάση μία έκφραση λάμδα ως εξής:

```
students.sort((s1, s2) -> s1.getAm().compareTo(s2.getAm()));
```

Απόσπασμα Κώδικα 2.7: Έκφραση λάμδα αντί ανώνυμης κλάσης Comparator

2.3.3.2 Functional Interfaces

Μία συναρτησιακή διεπαφή είναι μία διεπαφή η οποία ορίζει μόνο μία αφηρημένη μέθοδο. Σε συνδυασμό με τις εκφράσεις λάμδα, παρέχουν στην Java δυνατότητες συναρτησιακού προγραμματισμού, αφού είναι ένας τρόπος οι συναρτήσεις να αντιμετωπιστούν ως «πολίτες πρώτης κατηγορίας». Η χρησιμότητα των συναρτησιακών διεπαφών βρίσκεται στην αφηρημένη μέθοδο, η υπογραφή της οποίας μπορεί να περιγράψει την υπογραφή μιας έκφρασης λάμδα. Υπάρχουν ήδη ορισμένες γνωστές συναρτησιακές διεπαφές, όπως οι κλάσεις Comparable και Runnable. Οι συναρτησιακές διεπαφές σηματοδοτούνται με το annotation @FunctionalInterface, χωρίς όμως αυτό να είναι απαραίτητο.

Παρ’ όλο που είναι εύκολο να ορίσουμε τις δικές μας συναρτησιακές διεπαφές, η Java 8 παρέχει κάποιες που μπορούμε να χρησιμοποιήσουμε εύκολα και για πολλές εφαρμογές. Μία από αυτές είναι η java.util.function.Function<T, R> η οποία ορίζει την αφηρημένη μέθοδο apply, η οποία δέχεται ως όρισμα έναν generic τύπο T και επιστρέφει έναν επίσης generic τύπο R.

```
@FunctionalInterface
public interface Function<T, R>{
    R apply(T t);
}
```

Απόσπασμα Κώδικα 2.8: Διεπαφή Function

Με την χρήση της Function και της μεθόδου apply μπορούμε να προσομοιώσουμε την ανάθεση συνάρτησης ως τιμή σε μεταβλητή και το πέρασμα συναρτήσεων ως

ορίσματα σε άλλες μεθόδους. Στο παρακάτω τμήμα κώδικα αναθέτουμε στην μεταβλητή func ως τιμή, μία έκφραση λάμδα που υπολογίζει και επιστρέφει το μήκος ενός String. Έπειτα, καλώντας την apply πάνω στο αντικείμενο func, επιστρέφεται το μήκος της συμβολοσειράς “Hello World”, και αποθηκεύεται στην μεταβλητή result:

```
Function<String, Integer> func = x -> x.length();
Integer strLength = func.apply("Hello World");
```

Απόσπασμα Κώδικα 2.9: Ανάθεση συνάρτησης ως τιμή σε μεταβλητή

Μία άλλη χρήσιμη συναρτησιακή διεπαφή που παρέχει η Java και λειτουργεί με παρόμοιο τρόπο με την Function, είναι η Consume, η οποία ορίζει την αφηρημένη μέθοδο accept, και σε αντίθεση με την apply, δεν επιστρέφει κάτι ως έξοδο:

```
@FunctionalInterface
public interface Consumer<T>{
    void accept(T t);
}
```

Απόσπασμα Κώδικα 2.10: Διεπαφή Consumer

Στο παρακάτω παράδειγμα δημιουργούμε την μεταβλητή show, στην οποία αναθέτουμε ως τιμή μία έκφραση λάμδα η οποία εκτυπώνει στην οθόνη την παράμετρο που δέχεται ως είσοδο. Στην συνέχεια με μία απλή επανάληψη καλούμε την μέθοδο accept της μεταβλητής add για να εμφανίσουμε στην οθόνη τους αριθμούς από το 0 ως το 9:

```
Consumer<Integer> show = p -> System.out.println(p);
for(int i=0;i<10;i++) {
    show.accept(i);
}
```

Απόσπασμα Κώδικα 2.11: Χρήση της accept()

Οι συναρτησιακές διεπιφάνειες είναι πολύ χρήσιμες για τον συνδυασμό διαδοχικών συναρτήσεων, μία ακόμη βασική αρχή του συναρτησιακού προγραμματισμού, όπως θα δούμε παρακάτω.

2.3.3.3 Method References

Οι αναφορές μεθόδων είναι μία περαιτέρω απλούστευση των εκφράσεων λάμδα που κάνουν τον κώδικα ακόμα πιο κατανοητό και περιεκτικό. Πρόκειται για έναν τύπο συντομευμένων εκφράσεων λάμδα, όταν αυτές καλούν μόνο μία συγκεκριμένη ήδη υλοποιημένη μέθοδο. Υπάρχουν τρεις τύποι αναφορών μεθόδων:

1. Αναφορά σε στατική μέθοδο
2. Αναφορά σε μέθοδο στιγμιότυπου (instance method)
3. Αναφορά σε κατασκευαστή

Η γενική σύνταξη για την χρήση μίας αναφοράς μεθόδου είναι η εξής:

`ClassName::methodName` ή `object::methodName`

Στο παρακάτω παράδειγμα, θέλουμε να εμφανίσουμε στην κονσόλα τις λέξεις από μία λίστα (εδώ, την words). Χρησιμοποιώντας την forEach, μπορούμε να το κάνουμε είτε με μία έκφραση λάμδα, είτε με μία αναφορά μεθόδου στην στατική μέθοδο println της κλάσης System:

```
List<String> words = Arrays.asList("hello", "world");

// using lambda
words.forEach(w -> System.out.println(w));

// using method reference
words.forEach(System.out::println);
```

Απόσπασμα Κώδικα 2.12: Αναφορά μεθόδου σε στατική μέθοδο

2.3.4 Σύνθεση Συναρτήσεων

Η σύνθεση συναρτήσεων αποτελεί μία τεχνική που ενθαρρύνεται στον συναρτησιακό προγραμματισμό και αναφέρεται στην σύνθεση πολύπλοκων συναρτήσεων από απλούστερες. Στην Java επιτυγχάνεται με τις συναρτησιακές διεπαφές (Functional Interfaces), τις εκφράσεις λάμδα και τις αναφορές μεθόδων, που αναφέρθηκαν στο προηγούμενο κεφάλαιο. Η συναρτησιακή διεπαφή Function, παρέχει τις μεθόδους compose() και andThen(), οι οποίες χρησιμοποιούνται για τον συνδυασμό συναρτήσεων.

Στο παρακάτω παράδειγμα αναθέτουμε στο αντικείμενο-μεταβλητή times2 μία λάμδα έκφραση που αντιστοιχεί σε μία συνάρτηση, η οποία δοθείσης μίας τιμής v, επιστρέφει την τιμή πολλαπλασιασμένη με το 2. Αντίστοιχα στην μεταβλητή sqrt αναθέτουμε ως τιμή μία συνάρτηση η οποία επιστρέφει το τετράγωνο μίας τιμής. Στην συνέχεια, χρησιμοποιώντας την συνάρτηση compose(), δημιουργούμε μία σύνθετη συνάρτηση, η οποία αποτελεί συνδυασμό των προηγούμενων δύο, και την αναθέτουμε ως τιμή στην μεταβλητή times2ThenSqrt. Τέλος εμφανίζουμε το αποτέλεσμα που παράγει η σύνθετη συνάρτηση καλώντας την apply() με την τιμή 3:

```
Function<Double, Double> times2 = v -> v * 2;
Function<Double, Double> sqrt = v -> v*v;

Function<Double, Double> times2ThenSqrt = sqrt.compose(times2);
System.out.println(times2ThenSqrt.apply(3.0));
```

Απόσπασμα Κώδικα 2.13: Σύνθεση συναρτήσεων με την συνάρτηση compose()

Αντίστοιχα, μπορούμε να συνθέσουμε τις δύο συναρτήσεις χρησιμοποιώντας αντί για την compose() την συνάρτηση andThen():

```
Function<Double, Double> sqrtThentimes2 = sqrt.andThen(times2);
System.out.println(sqrtThentimes2.apply(3.0));
```

Απόσπασμα Κώδικα 2.14: Σύνθεση συναρτήσεων με την συνάρτηση andThen()

Η διαφορά μεταξύ των compose() και andThen(), έγκειται στην σειρά με την οποία εφαρμόζουν την συναρτήσεις. Η μεν compose() εφαρμόζει πρώτα την συνάρτηση η οποία δίνεται ως όρισμα και έπειτα την συνάρτηση που αντιστοιχεί στο αντικείμενο που την κάλεσε. Η andThen() λειτουργεί αντίστροφα, εφαρμόζοντας πρώτα την συνάρτηση που αντιστοιχεί στο αντικείμενο που την κάλεσε και ύστερα αυτή που δέχθηκε ως όρισμα.

2.3.5 Currying

Η τεχνική currying πήρε το όνομά της από τον μαθηματικό Haskell Curry [3] (αν και δεν ήταν ο εφευρέτης της), και πρόκειται για μία μαθηματική τεχνική με την οποία μία συνάρτηση η οποία δέχεται πολλαπλά ορίσματα μετατρέπεται σε μία σειρά συναρτήσεων που δέχονται ένα μόνο όρισμα. Στον συναρτησιακό προγραμματισμό αποτελεί ένα πολύτιμο εργαλείο σύνθεσης συναρτήσεων, καθώς επιτρέπει να καλούνται συναρτήσεις χωρίς να είναι απαραίτητο να περαστούν σε αυτές όλα τους τα ορίσματα. Κατά συνέπεια, μία συνάρτηση η οποία έχει δημιουργηθεί με αυτή την τεχνική, δεν παράγει το αποτέλεσμά της έως ότου δοθούν σε αυτήν όλα τα απαραίτητα ορίσματα.

Στις αμιγώς συναρτησιακές γλώσσες προγραμματισμού η τεχνική currying υποστηρίζεται πλήρως. Στην Java ωστόσο, επιτυγχάνεται και πάλι χρησιμοποιώντας της συναρτησιακές διεπαφές. Στο παρακάτω παράδειγμα [2], χρησιμοποιείται η τεχνική currying για τον υπολογισμό του βάρους ενός ατόμου σε διαφορετικούς πλανήτες. Η μάζα του ατόμου παραμένει η ίδια αλλά το βάρος αλλάζει καθώς εξαρτάται από την τιμή της βαρύτητας στον κάθε πλανήτη:

```
Function<Double, Function<Double, Double>> weight = mass -> gravity ->
mass * gravity;

Function<Double, Double> weightOnEarth = weight.apply(9.81);
System.out.println("Weight on Earth: " + weightOnEarth.apply(60.0));

Function<Double, Double> weightOnMars = weight.apply(3.75);
System.out.println("Weight on Mars: " + weightOnMars.apply(60.0));
```

Απόσπασμα Κώδικα 2.15: Τεχνική currying χρησιμοποιώντας την συναρτησιακή διεπαφή Function

3 Java 8 & Stream API

Το Stream API εμφανίστηκε στην έκδοση 8 της Java και προσθέτει στην γλώσσα κάποια σημαντικά χαρακτηριστικά συναρτησιακού προγραμματισμού στην επεξεργασία δεδομένων. Ακόμα, διευκολύνει και απλοποιεί διαδικασίες επεξεργασίας δεδομένων με παράλληλο τρόπο, αποκρύπτοντας από τον προγραμματιστή πολλές λεπτομέρειες της υλοποίησης της παράλληλης επεξεργασίας. Ο κώδικας γράφεται με δηλωτικό τρόπο, διευκρινίζοντας το τί θέλει ο προγραμματιστής να πετύχει και όχι τον τρόπο με τον οποίο αυτό θα πραγματοποιηθεί, και συνδέοντας αλλεπάλληλες λειτουργίες πάνω στα δεδομένα. Με αυτόν τον τρόπο μειώνονται οι γραμμές κώδικα και ο κώδικας γίνεται πιο ευανάγνωστος και κατανοητός.

Στον μη-συναρτησιακό προγραμματισμό ο πιο συνήθης τρόπος επεξεργασίας συλλογών δεδομένων είναι μέσω βρόχων επαναλήψεων. Αν και οι προγραμματιστές είναι εξοικειωμένοι με αυτόν τον τρόπο, όταν ο όγκος των δεδομένων είναι μεγάλος, μπορεί να γίνει αρκετά δαπανηρός και δεν πρόκειται για την αποδοτικότερη λύση. Τα streams προσφέρουν εναλλακτικούς τρόπους προσέγγισης της επεξεργασίας των δεδομένων, βασισμένους στο πρότυπο του συναρτησιακού προγραμματισμού.

Βασικός άξονας του προγραμματισμού με streams, είναι πως το πρόβλημα προσεγγίζεται όχι με βάση το πώς θα λυθεί αλλά με βάση το τί είναι αυτό που θέλουμε να γίνει. Στην Java, ένα stream είναι μία συλλογή αντικειμένων που δημιουργείται στην μνήμη και παύει να υπάρχει μόλις η επεξεργασία ολοκληρωθεί [8]. Η επεξεργασία ενός stream γίνεται σε τρία στάδια:

1. Δημιουργία του stream, είτε από την αρχή είτε από μία υπάρχουσα συλλογή
2. Επεξεργασία του stream με ενδιάμεσες λειτουργίες. Αυτές οι λειτουργίες μεταμορφώνουν το stream σε άλλα streams και μπορούν να πραγματοποιηθούν σε ένα ή περισσότερα βήματα.
3. Εφαρμογή μίας τερματικής διαδικασίας στο stream. Η κλήση της τερματικής συνάρτησης είναι αυτή που πυροδοτεί και την εκτέλεση των προηγούμενων. Μόλις ολοκληρωθεί και η τερματική συνάρτηση, το stream παύει να υπάρχει στην μνήμη και δεν μπορεί να ξαναχρησιμοποιηθεί.

Τα streams μπορεί επιφανειακά να μοιάζουν με τις συλλογές της Java, ωστόσο υπάρχουν ορισμένες σημαντικές διαφορές. Τα δεδομένα δεν αποθηκεύονται σε ένα stream, με τον τρόπο που αποθηκεύονται σε μια μεταβλητή. Αντιθέτως, μόλις

ολοκληρωθούν οι επεξεργασίες, παύει να υπάρχει. Εφαρμόζοντας την αρχή των αμετάβλητων δεδομένων του συναρτησιακού προγραμματισμού, τα αρχικά δεδομένα του stream δεν μεταβάλλονται. Αντ' αυτού, η επεξεργασία με streams μπορεί να επιστρέψει μία νέα συλλογή δεδομένων, που προέκυψαν εφαρμόζοντας λειτουργίες πάνω στα αρχικά δεδομένα. Τέλος, οι λειτουργίες των streams είναι «οκνηρές» (lazy) όταν αυτό είναι εφικτό. Με τον όρο αυτόν, εννοείται ότι δεν εκτελούνται μέχρι την στιγμή που χρειάζεται να χρησιμοποιηθεί το αποτέλεσμά τους [9].

Στο παρακάτω κεφάλαιο περιγράφεται αναλυτικά ο τρόπος δημιουργίας και χρήσης των Streams μέσα από παραδείγματα, καθώς και ορισμένες σημαντικές συναρτήσεις. Τέλος, παρουσιάζονται τα parallel streams, με την χρήση των οποίων η επεξεργασία των δεδομένων γίνεται με παράλληλο τρόπο.

3.1 Δημιουργία streams

Η βασική κλάση, είναι η διεπαφή Stream<T>, η οποία παρέχει ένα μεγάλο σύνολο έτοιμων συναρτήσεων. Για την δημιουργία ενός stream, υπάρχουν αρκετοί διαφορετικοί τρόποι, οι σημαντικότεροι των οποίων, παρουσιάζονται παρακάτω.

1. Άδειο stream: μπορούμε να δημιουργήσουμε ένα άδειο stream, ένα stream δηλαδή χωρίς κανένα στοιχείο καλώντας την μέθοδο empty(). Συνήθως χρησιμοποιείται έτσι ώστε να μην επιστραφεί null, στην περίπτωση που μία συλλογή δεν περιέχει καθόλου στοιχεία.
2. Stream από συλλογή: πρόκειται για τον πιο συνηθισμένο τρόπο δημιουργίας ενός stream. Αυτό γίνεται με την κλήση της μεθόδου stream(), επάνω στο αντικείμενο, του οποίου τα δεδομένα θα τροφοδοτήσουν το stream. Στο παρακάτω παράδειγμα δημιουργείται ένα αντικείμενο τύπου Stream<String> από το αντικείμενο stringList που είναι τύπου ArrayList<String>:

```
ArrayList<String> stringList = new ArrayList<>();
Stream<String> stream = stringList.stream();
```

Απόσπασμα Κώδικα 3.1: Δημιουργία stream από συλλογή

3. Stream από πίνακα με τιμές: ένας πίνακας μπορεί να είναι πηγή για την δημιουργία ενός stream. Υπάρχουν δύο τρόποι για την δημιουργία ενός stream από πίνακα. Ο πρώτος είναι με την κλήση της στατικής συνάρτησης of() της κλάσης Stream, δίνοντας ως ορίσματα τα στοιχεία που θα αποτελέσουν τα αντικείμενα του stream:

```
Stream<String> stream = Stream.of("an", "array", "of",
"strings");
```
Απόσπασμα Κώδικα 3.2: Δημιουργία stream από πίνακα με κλήση της of()

Ο δεύτερος τρόπος είναι με την κλήση της στατικής συνάρτησης stream() της κλάσης Arrays, δίνοντας ως όρισμα τον πίνακα του οποίου τα στοιχεία θα αποτελέσουν τα αντικείμενα του stream:

```
String[] arr = new String[]{"this", "is", "a", "string",
"array"};
Stream<String> stream = Arrays.stream(arr);
```
Απόσπασμα Κώδικα 3.3: Δημιουργία stream από πίνακα με κλήση της stream()

4. Stream.builder(): με την μέθοδο builder() είναι εφικτό να δημιουργηθεί ένα stream προσθέτοντας τα στοιχεία το ένα πίσω από το άλλο. Τα στοιχεία προστίθενται καλώντας την μέθοδο add(), ενώ στο τέλος πρέπει να κληθεί και η συνάρτηση build(), για να συνθέσει όλα τα αντικείμενα σε ένα stream. Είναι σημαντικό ο τύπος δεδομένων του stream να οριστεί και στο δεξί μέρος της ανάθεσης, ειδάλλως το stream που προκύπτει είναι τύπου Object:

```
Stream<String> stream =
Stream.<String>builder().add("add").add("some").add("words")
.build();
```
Απόσπασμα Κώδικα 3.4: Δημιουργία stream με κλήση της Stream.builder()

5. Stream.generate(): η στατική μέθοδος generate() της κλάσης Stream, δημιουργεί ένα μη πεπερασμένο stream. Δέχεται ως όρισμα ένα αντικείμενο τύπου Supplier<T> , και είναι απαραίτητο να οριστεί ένα όριο, καλώντας την limit(), ειδάλλως η generate() θα προσθέτει αντικείμενα στο stream έως ότου να εξαντληθεί η μνήμη. Στο παρακάτω παράδειγμα δημιουργείται ένα stream με 20 μηδενικά:

```
Stream<Integer> stream =
        Stream.generate(new Supplier<Integer>() {
            @Override
            public Integer get() {
                return 0;
            }
        }).limit(20)
```
Απόσπασμα Κώδικα 3.5: Δημιουργία stream με κλήση της generate()

6. Stream.iterate(): ένας άλλος τρόπος δημιουργίας μη πεπερασμένου stream είναι με την συνάρτηση iterate(). Αυτή δέχεται δύο ορίσματα, το πρώτο είναι η τιμή του πρώτου στοιχείου του stream (ή αλλιώς «σπόρος» ή seed), και το δεύτερο είναι μία συνάρτηση (η οποία μπορεί να είναι μία λάμδα έκφραση) με την οποία ορίζεται ποια θα είναι η σχέση κάθε επόμενου στοιχείου με το

προηγούμενο. Και στην περίπτωση της iterate(), πρέπει να οριστεί ένα όριο καλώντας την limit(), αλλιώς τα στοιχεία θα προστίθενται μέχρις ότου να εξαντληθεί η μνήμη. Στο παρακάτω παράδειγμα δημιουργείται ένα stream με 100 στοιχεία (το όριο ορίζεται στην κλήση της limit()), όπου το πρώτο έχει την τιμή 0 και κάθε επόμενο έχει την τιμή του προηγούμενου συν ένα:

```
Stream<Integer> stream = Stream.iterate(0, n ->
n++).limit(100);
```

Απόσπασμα Κώδικα 3.6: Δημιουργία stream με κλήση της iterate()

7. Stream από τύπους primitive: στην περίπτωση των primitive τύπων int, double και long, υπάρχουν τρεις ειδικές διεπαφές, οι IntStream, DoubleStream και LongStream, οι οποίες παρέχουν μεθόδους για δημιουργία streams. Εκτός από αυτές στις οποίες ήδη αναφερθήκαμε, δηλαδή τις generate(), builder(), empty() και of(), ενδιαφέρον παρουσιάζουν οι range() και rangeClosed(). Η πρώτη δέχεται ως ορίσματα δύο αριθμούς, ο πρώτος εκ των οποίων θα είναι η αρχή του stream και ο δεύτερος το τέλος, χωρίς την συμπερίληψη του ίδιου του αριθμού, ενώ η δεύτερη λειτουργεί με παρόμοιο τρόπο, συμπεριλαμβάνοντας όμως και τον δεύτερο αριθμό στο stream. Τα στοιχεία αυξάνονται κατά ένα. Οι παραπάνω συναρτήσεις προσφέρονται μόνο στην περίπτωση των IntStream και LongStream. Στο παρακάτω παράδειγμα δημιουργούνται δύο streams, το πρώτο με τους ακέραιους αριθμούς 10 έως 99 και το δεύτερο με ακέραιους αριθμούς 10 έως και 100:

```
IntStream intStream = IntStream.range(10, 100);
LongStream longStream = LongStream.rangeClosed(10, 100);
```

Απόσπασμα Κώδικα 3.7: Δημιουργία stream primitive τύπων

8. Stream τυχαίων αριθμών: η κλάση Random, από την έκδοση 8 της Java και έπειτα προσφέρει μία μέθοδο για δημιουργία stream τυχαίων αριθμών τύπου int, double και long, χρησιμοποιώντας τις αντίστοιχες διεπαφές IntStream, DoubleStream, LongStream. Στο παρακάτω απόσπασμα κώδικα φαίνεται η δημιουργία των τριών αυτών streams, καλώντας αντίστοιχα τις μεθόδους ints(), doubles() και longs():

```
Random random = new Random();
IntStream intStream = random.ints();
DoubleStream doubleStream = random.doubles();
LongStream longStream = random.longs();
```

Απόσπασμα Κώδικα 3.8: Δημιουργία streams τυχαίων αριθμών

9. Stream από αρχείο: στην κλάση Files υπάρχουν πολλές μέθοδοι για παραγωγή streams από αρχεία. Ενδιαφέρον παρουσιάζει η μέθοδος lines(), η οποία δέχεται ως όρισμα το μονοπάτι προς ένα αρχείο και δημιουργεί ένα stream αλφαριθμητικών, όπου το κάθε του στοιχείο είναι μία γραμμή του αρχείου:

```
Path path = Paths.get("C:\\file.txt");
Stream<String> streamOfStrings = Files.lines(path);
```

Απόσπασμα Κώδικα 3.9: Δημιουργία stream αλφαριθμητικών από αρχείο

3.2 Σημαντικές συναρτήσεις: περιγραφή & παραδείγματα

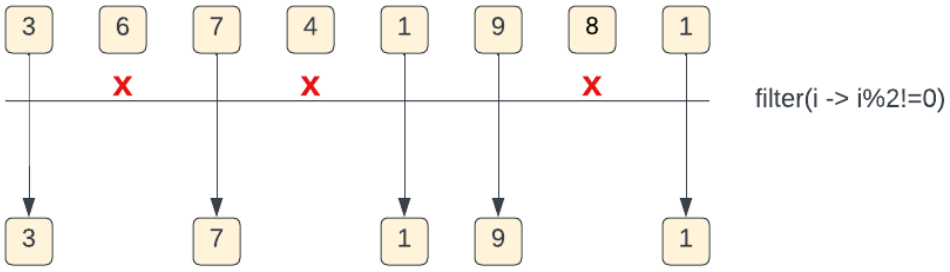
Όπως αναφέρθηκε, οι συναρτήσεις που μπορούν να εφαρμοστούν στα streams διακρίνονται σε ενδιάμεσες (intermediate) και τερματικές (terminal). Οι πρώτες μπορούν να εφαρμοστούν η μία μετά την άλλη επάνω στο stream, ενώ οι δεύτερες μόνο μία φορά. Μόλις μία τερματική συνάρτηση κληθεί για ένα stream, τότε αυτό διαγράφεται από την μνήμη και δεν μπορεί να χρησιμοποιηθεί ξανά. Παρακάτω περιγράφονται οι σημαντικότερες εξ αυτών των συναρτήσεων και παρουσιάζονται παραδείγματα χρήσης τους.

3.2.1 Ενδιάμεσες συναρτήσεις

Οι ενδιάμεσες συναρτήσεις έχουν ως τύπο επιστροφής ένα stream, επιτρέποντας με αυτόν τον τρόπο να εφαρμοστούν η μία μετά την άλλη, σχηματίζοντας μία αλυσίδα. Η εκτέλεση μίας ενδιάμεσης συνάρτησης δεν πραγματοποιείται, έως ότου να κληθεί στο stream και μία τερματική συνάρτηση. Για τον λόγο αυτό αναφέρεται ότι είναι «οκνηρές» (lazy). Ορισμένες σημαντικές ενδιάμεσες συναρτήσεις, καθώς και η λειτουργίες τους παρουσιάζονται παρακάτω.

3.2.1.1 Filter

Η συνάρτηση filter, δέχεται ως όρισμα ένα κατηγόρημα, δηλαδή μία συνάρτηση η οποία επιστρέφει τύπο Boolean, και επιστρέφει ένα stream το οποίο αποτελείται μόνο από τα στοιχεία του αρχικού stream, τα οποία ικανοποιούν την συνθήκη. Σε μία πραγματική εφαρμογή, μπορεί να χρησιμοποιηθεί για να σχηματιστούν ερωτήματα όπως: α) να επιστραφούν μόνο τα προϊόντα που έχουν τιμή πάνω από 10 ευρώ, β) να επιστραφούν τα βιβλία που έχουν ως συγγραφέα την Isabelle Allente, γ) να επιστραφούν τα ζώα που ανήκουν στην κατηγορία θηλαστικά, κ.α.



Διάγραμμα 1: Παράδειγμα φιλτραρίσματος

Στο παρακάτω παράδειγμα δημιουργείται ένα stream με τυχαίους ακεραίους, καλώντας την συνάρτηση ints(), και στην συνέχεια το stream αυτό φιλτράρεται χρησιμοποιώντας την συνάρτηση filter, με κατηγόρημα μία έκφραση λάμδα. Η έκφραση λάμδα ορίζει ότι θα επιστρέφονται μόνο οι αριθμοί που δεν διαιρούνται ακριβώς με το 0. Μετά την εκτέλεση της filter (αφότου δηλαδή κληθεί και μία τερματική συνάρτηση), η μεταβλητή intStream, θα περιέχει μόνο τους μονούς αριθμούς από το σύνολο αυτών που περιείχε:

```
// create a stream of random integers
IntStream intStream = random.ints();
// filter the stream, keeping only the odd ones
intStream.filter(i -> i%2!=0);
```

Απόσπασμα Κώδικα 3.10: Παράδειγμα συνάρτησης filter

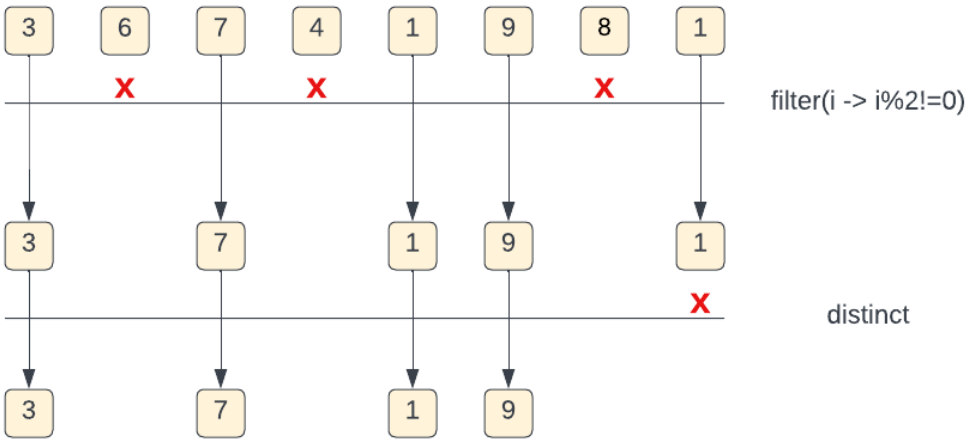
Το παραπάνω παράδειγμα μπορεί να γραφεί ισοδύναμα, δίνοντας ως όρισμα στην filter αντί για την έκφραση λάμδα, μία ανώνυμη κλάση που υλοποιεί την διεπαφή intPredicate:

```
// filter the stream, keeping only the odd ones
intStream.filter(new IntPredicate() {
    @Override
    public boolean test(int i) {
        return i % 2 != 0;
    }
});
```

Απόσπασμα Κώδικα 3.11: Παράδειγμα filter με ανώνυμη κλάση

3.2.1.2 Distinct

Με την συνάρτηση distinct τα στοιχεία ενός stream φιλτράρονται έτσι ώστε να εξαλειφθούν τα διπλότυπα, έχοντας λειτουργία παρόμοια με την αντίστοιχη distinct των γλωσσών ερωτημάτων για βάσεις δεδομένων, όπως η SQL.



Διάγραμμα 2: Παράδειγμα distinct

Παρακάτω επεκτείνεται το προηγούμενο παράδειγμα, χρησιμοποιώντας την distinct, έτσι ώστε κάθε μοναδικός αριθμός να υπάρχει στο stream μία μόνο φορά:

```
// filter the stream, keeping only the unique odd ones
intStream.filter(i -> i%2!=0).distinct();
```

Απόσπασμα Κώδικα 3.12: Παράδειγμα συνάρτησης distinct

3.2.1.3 Skip

Η συνάρτηση skip χρησιμοποιείται για να παραλειφθούν τα πρώτα n στοιχεία από το stream, λαμβάνοντας το n ως όρισμα. Για παράδειγμα διαβάζοντας ένα αρχείο, το οποίο περιέχει έναν αριθμό γραμμών επικεφαλίδας, για να εισαχθούν τα δεδομένα του γραμμή προς γραμμή σε ένα stream, πρέπει να αγνοηθούν οι πρώτες αυτές γραμμές και στην συνέχεια να διαβαστούν οι επόμενες. Στο παρακάτω παράδειγμα, χρησιμοποιώντας την συνάρτηση lines, διαβάζονται σε ένα stream οι γραμμές του αρχείου, παραλείποντας την πρώτη, καλώντας την συνάρτηση skip(1):

```
// create a stream of the lines of a file
// skipping the first
Stream<String> linesStream = Files
        .lines(Path.of("text.txt"))
        .skip(1);
```

Απόσπασμα Κώδικα 3.13: Παράδειγμα συνάρτησης skip

3.2.1.4 Sorted

Η sorted χρησιμοποιείται για να ταξινομηθούν τα στοιχεία ενός stream. Στην περίπτωση αριθμών ταξινομούνται σε φυσική αύξουσα σειρά, ενώ τα αλφαριθμητικά ταξινομούνται σε αύξουσα αλφαβητική σειρά. Αν τα προς ταξινόμηση στοιχεία είναι

αντικείμενα κάποιας άλλης κλάσης, τότε πρέπει να υλοποιούν την διεπαφή Comparable, έτσι ώστε να ορίζεται κάποιος τρόπος ταξινόμησης μέσω της μεθόδου compareTo.

Παρακάτω, δημιουργείται ένα stream τυχαίων αριθμών τύπου double, το οποίο ταξινομείται κατά αύξουσα αριθμητική σειρά:

```
// create a stream of random doubles
DoubleStream doubleStream = random.doubles();
// sort the stream in ascending order
doubleStream.sorted();
```

Απόσπασμα Κώδικα 3.14: Παράδειγμα συνάρτησης sorted

3.2.1.5 Limit

Η συνάρτηση limit, όπως και η skip, δέχεται έναν ακέραιο ως όρισμα, το οποίο ορίζει τον αριθμό των στοιχείων στο stream. Χρησιμοποιείται ουσιαστικά για να διατηρηθούν στο stream ένας συγκεκριμένος αριθμός στοιχείων. Στην περίπτωση που το stream είναι ταξινομημένο, η limit μπορεί να χρησιμοποιηθεί για να διατηρηθούν τα πρώτα n (είτε τα τελευταία n) στοιχεία. Αν όμως το stream δεν είναι ταξινομημένο, τότε τα στοιχεία που μένουν στο stream δεν ακολουθούν κάποια συγκεκριμένη σειρά.

Παρακάτω, χρησιμοποιείται η sorted έτσι ώστε να ταξινομηθούν τα στοιχεία ενός stream αριθμών double σε αύξουσα σειρά και στην συνέχεια διατηρούνται τα 3 μόνο πρώτα, καλώντας την συνάρτηση limit με όρισμα τον αριθμό 3. Κατ' αυτόν τον τρόπο, το stream περιέχει ουσιαστικά τους τρεις μικρότερους αριθμούς:

```
// sort the stream in ascending order
// and keep only the first 3 doubles
doubleStream
        .sorted()
        .limit(3);
```

Απόσπασμα Κώδικα 3.15: Παράδειγμα συνάρτησης limit

3.2.1.6 Map

Μία από τις χρησιμότερες συναρτήσεις είναι η map, με την οποία πραγματοποιούνται λειτουργίες απεικόνισης. Κατά την απεικόνιση, χρησιμοποιείται μία συνάρτηση, έστω f, η οποία εφαρμόζεται σε κάθε στοιχείο, έστω m, για να το μετατρέψει:

$$m = f(m)$$

Γι' αυτόν τον λόγο, δέχεται ως όρισμα ένα αντικείμενο – συνάρτηση, το οποίο ορίζει τον τρόπο με τον οποίο κάθε στοιχείο θα επεξεργαστεί. Η συνάρτηση που δίνεται ως όρισμα, μπορεί να είναι μία ανώνυμη κλάση που υλοποιεί την διεπαφή FunctionalInterface, μία αναφορά μεθόδου (method reference) ή μία έκφραση λάμδα.

Στο παράδειγμα, δημιουργείται ένα stream αλφαριθμητικών, στο οποίο για κάθε στοιχείο εφαρμόζεται η συνάρτηση toUpperCase, έτσι ώστε το stream που προκύπτει να περιέχει τα αλφαριθμητικά του αρχικού stream με κεφαλαία γράμματα. Η λειτουργία πραγματοποιείται με 3 ισοδύναμους τρόπους:

```
Stream<String> stringStream = Stream.of("an", "array", "of",
"strings");

// convert to upper case using anonymous class
stringStream.map(new Function<String, String>() {
    @Override
    public String apply(String s) {
        return s.toUpperCase();
    }
});

// convert to upper case using lambda
stringStream.map(s -> s.toUpperCase());

// convert to upper case using method reference
stringStream.map(String::toUpperCase);
```

Απόσπασμα Κώδικα 3.16: Παράδειγμα συνάρτησης map

Ορισμένες φορές είναι χρήσιμο ένα stream αντικειμένων να μετατρέπεται σε stream primitive τύπων δεδομένων. Για τον λόγο αυτόν, υπάρχουν στην Java 8 ορισμένες έτοιμες συναρτήσεις, οι mapToInt, mapToDouble και mapToLong.

3.2.2 Τερματικές συναρτήσεις

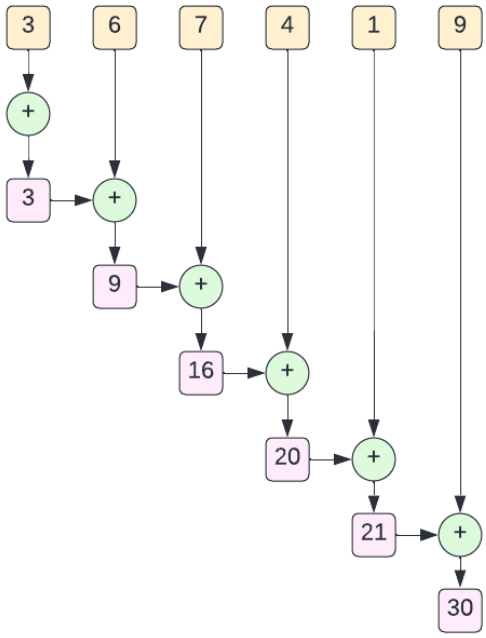
Οι τερματικές συναρτήσεις παράγουν ένα τελικό αποτέλεσμα από το stream. Ο τύπος επιστροφής τους είναι είτε ο κενός τύπος (void) είτε κάποιος τύπος που δεν είναι stream, όπως ένας ακέραιος, μία λίστα, ένα αλφαριθμητικό, κλπ. Η κλήση τερματικών συναρτήσεων είναι αυτή που αναγκάζει και τις ενδιάμεσες να εκτελεστούν. Εάν δεν κληθεί κάποια τερματική συνάρτηση, τότε οι ενδιάμεσες δεν εκτελούνται και το stream παραμένει διαθέσιμο στην μνήμη. Μόλις όμως μία τερματική συνάρτηση κληθεί, παράγεται το τελικό αποτέλεσμα, το stream παύει να υφίσταται στην μνήμη και δεν μπορεί να ξαναχρησιμοποιηθεί. Ακριβώς γι' αυτόν τον λόγο, η κλήση μίας τερματικής συνάρτησης πρέπει να είναι και η τελευταία από μία αλυσίδα κλήσεων συναρτήσεων σε streams.

Ορισμένες σημαντικές τερματικές συναρτήσεις, καθώς και η λειτουργίες τους παρουσιάζονται παρακάτω.

3.2.2.1 Reduce

Η λειτουργία της αναγωγής (reduce) είναι μία γνωστή λειτουργία στους αλγορίθμους, κατά την οποία ένα σύνολο ή μία συλλογή δεδομένων ανάγονται σε ένα

τελικό αποτέλεσμα, το οποίο μπορεί να είναι ένα άθροισμα, ένας μέσος όρος, ένα γινόμενο, κ.α.



Διάγραμμα 3: Αναγωγή για υπολογισμό αθροίσματος

Στην Java, πριν τα streams, ο συνήθης τρόπος αναγωγής ήταν ένα βρόχος, συνήθως for, μέσα στον οποίο προστίθεται (στην περίπτωση του αθροίσματος) σε μία μεταβλητή σε κάθε επανάληψη το επόμενο στοιχείο της συλλογής. Στην Java 8 και με τα streams, δίνεται η δυνατότητα πραγματοποίησης αναγωγής καλώντας την μέθοδο reduce επάνω σε ένα stream. Η reduce δέχεται ως όρισμα ένα αντικείμενο-συνάρτηση, το οποίο υλοποιεί την διεπαφή FunctionalInterface, και ορίζει το πώς δύο στοιχεία του stream θα συνδυαστούν για να παραχθεί η νέα τιμή. Μπορεί να χρησιμοποιηθεί για πολλούς υπολογισμούς, όπως αθροίσματα, υπολογισμός μεγίστου/ελαχίστου, γινόμενα, κ.α.

Παρακάτω, φαίνεται ο υπολογισμός του αθροίσματος των αριθμών από το 1 έως και το 100, χρησιμοποιώντας την reduce. Για τους αριθμούς από το 1 έως και το 100, δημιουργείται ένα stream ακεραίων καλώντας την rangeClosed. Στην συνέχεια το stream ανάγεται στο άθροισμα, δίνοντας ως όρισμα στην reduce μία έκφραση λάμδα, που περιγράφει την πράξη, a+b:

```
Integer sum = IntStream.rangeClosed(1,100).reduce((a,b) ->
a+b).getAsInt();
```

Απόσπασμα Κώδικα 3.17: Υπολογισμός αθροίσματος με reduce

Στην περίπτωση που αναζητείται το άθροισμα ιδιοτήτων αντικειμένων, τότε πριν την αναγωγή πρέπει να προηγηθεί απεικόνιση, έτσι ώστε το stream των αντικειμένων να

μετατραπεί σε ένα stream αριθμών οι οποίοι να μπορούν να προστεθούν. Έτσι, αν για παράδειγμα από μία λίστα προϊόντων σε μία αποθήκη πρέπει να υπολογιστεί το συνολικό τους βάρος, τότε από την λίστα των προϊόντων (products) δημιουργείται ένα stream, από το οποίο με την χρήση της συνάρτησης map εξάγονται για κάθε ένα το βάρος τους. Από αυτήν την λειτουργία της απεικόνισης δημιουργείται ένα stream αριθμών πάνω στο οποίο καλείται η reduce, έτσι ώστε τα βάρη να αθροιστούν και να υπολογιστεί το συνολικό βάρος:

```
Double totalWeight = products.stream()
        .map(p -> p.getWeight())
        .reduce((a,b) -> a+b).get();
```

Απόσπασμα Κώδικα 3.18: Πράξη αναγωγής σε αντικείμενα

Η αναγωγή μπορεί να χρησιμοποιηθεί για ένα μεγάλο εύρος υπολογισμών, όπως αθροίσματα, μέτρηση πλήθους στοιχείων, μέσος όρος, γινόμενα, εύρεση μεγίστου/ελαχίστου, κ.α. Ωστόσο επειδή οι υπολογισμοί αυτοί πραγματοποιούνται πολύ συχνά, στο Stream API της Java 8, έχουν ενσωματωθεί και έτοιμες συναρτήσεις για αρκετές από αυτές τις λειτουργίες, οι σημαντικότερες εκ των οποίων παρουσιάζονται παρακάτω.

3.2.2.2 Count

Για την μέτρηση του πλήθους των στοιχείων ενός stream, υπάρχει η μέθοδος count. Αυτή επιστρέφει έναν ακέραιο που αντιστοιχεί στον αριθμό των στοιχείων του stream. Στο παρακάτω απόσπασμα κώδικα, δημιουργείται ένα stream από τις γραμμές ενός αρχείου με όνομα movies.csv. Έπειτα, αφού παραλειφθεί η πρώτη γραμμή, καλώντας την συνάρτηση skip με όρισμα την τιμή 1, καθώς υποθέτουμε ότι η πρώτη γραμμή του αρχείου περιέχει πληροφορίες κεφαλίδας, καλείται η συνάρτηση map, για να αντιστοιχίσει κάθε γραμμή του αρχείου σε έναν τίτλο ταινίας και να δημιουργηθεί το αντίστοιχο αντικείμενο. Με την κλήση της distinct αφαιρούνται πιθανά διπλότυπες εγγραφές. Οι τρεις τελευταίες συναρτήσεις, καλούνται δημιουργώντας μια αλυσίδα, καθώς είναι όλες ενδιάμεσες. Τέλος, καλείται η count, η οποία επιστρέφει το πλήθος των ταινιών του stream:

```
long moviesCount = Files.lines(Path.of("movies.csv"))
        .skip(1) // skip the header line
        .map(title -> new Movie(title))
        .distinct()
        .count();
```

Απόσπασμα Κώδικα 3.19: Παράδειγμα συνάρτησης count

3.2.2.3 Min – Max

Η εύρεση ελαχίστου και μεγίστου από ένα σύνολο στοιχείων είναι μία διαδικασία πολύ συχνά χρησιμοποιούμενη. Με τα streams απλοποιείται σε πολύ μεγάλο βαθμό, δεν είναι απαραίτητη η χρήση κάποιου βρόχου, καθώς προσφέρονται οι συναρτήσεις min και max, οι οποίες εντοπίζουν και επιστρέφουν το μικρότερο και το μεγαλύτερο αντίστοιχα στοιχείο από ένα stream. Μπορούν να χρησιμοποιηθούν τόσο για primitive τύπους δεδομένων, όσο και για άλλα αντικείμενα τα οποία όμως υλοποιούν την διεπαφή Comparable, έτσι ώστε να μπορούν να πραγματοποιηθούν συγκρίσεις μεταξύ τους. Επιστρέφουν ένα αντικείμενο τύπου Optional<T>, όπου T ο τύπος δεδομένων του stream, το οποίο περιέχει το αντίστοιχο μικρότερο ή μεγαλύτερο στοιχείο, εάν υπάρχει τέτοιο, ή null σε διαφορετική περίπτωση.

Στο παρακάτω παράδειγμα, δημιουργείται από μία λίστα προϊόντων δημιουργείται ένα stream, πάνω στο οποίο στην συνέχεια καλείται η συνάρτηση max. Σε αυτήν δίνεται ως παράμετρος μία ανώνυμη κλάση για την σύγκριση των προϊόντων, η οποία μπορεί να γίνει καλώντας την μέθοδο compareTo, εφόσον η κλάση Product υλοποιεί την διεπαφή Comparable. Στην υλοποίηση της μεθόδου compareTo, τα προϊόντα συγκρίνονται με βάση την τιμή τους. Έτσι, αυτό που επιστρέφει η συνάρτηση max, είναι το προϊόν με την μεγαλύτερη τιμή. Από το αντικείμενο τύπου Optional<Product> με όνομα μεταβλητής highestPrice, μπορούμε να εξάγουμε το αντικείμενο τύπου Product καλώντας την μέθοδο get:

```
Optional<Product> highestPrice = products.stream().max(new
Comparator<Product>() {
    @Override
    public int compare(Product p1, Product p2) {
        return p1.compareTo(p2);
    }
});
Product p = highestPrice.get();
```

Απόσπασμα Κώδικα 3.20: Παράδειγμα συνάρτησης max

3.2.2.4 Sum & Average

Οι υπολογισμοί αθροισμάτων και μέσου όρου είναι ιδιαιτέρως συχνοί στον προγραμματισμό. Σε προηγούμενο κεφάλαιο δείξαμε πώς αυτό μπορεί να γίνει με χρήση της συνάρτησης reduce. Ωστόσο, υπάρχουν και δύο έτοιμες συναρτήσεις γι' αυτόν ακριβώς τον σκοπό, οι sum και average. Αυτές είναι διαθέσιμες μόνο για streams primitive τύπων, δηλαδή για τα IntStream, DoubleStream και LongStream. Έτσι η διαδικασία απλουστεύεται ακόμα περισσότερο, και ένα βρόχος τριών γραμμών μπορεί πλέον να γραφεί σε μόνο μία:

```
// calculate sum using the sum method
Integer sum = IntStream.rangeClosed(1,100).sum();

// calculate average using the average method
Double avg = IntStream.rangeClosed(1,100).average().getAsDouble();
```

Απόσπασμα Κώδικα 3.21: Παράδειγμα μεθόδων sum και average

3.2.2.5 ForEach

Η foreach είναι μία μέθοδος με την οποία εφαρμόζεται πάνω σε κάθε στοιχείο ενός stream μία λειτουργία. Σε αντίθεση όμως με την απεικόνιση (map), δεν επιστρέφει κάποιο stream το οποίο μπορεί να επεξεργαστεί περαιτέρω, καθώς πρόκειται για τερματική λειτουργία. Συνεπώς, δεν χρησιμοποιείται για να μετατραπεί ένα σύνολο δεδομένων σε κάτι άλλο βάσει μίας συνάρτησης, αλλά για λειτουργίες όπως η εμφάνιση ή η εγγραφή των δεδομένων. Δέχεται ως όρισμα μία συνάρτηση η οποία υλοποιεί την διεπαφή Consumer. Στα παρακάτω παραδείγματα, χρησιμοποιείται η forEach για να εμφανιστούν α) οι αριθμοί ενός stream ακεραίων και β) τα ονόματα και τα βάρη μίας λίστας προϊόντων:

```
// using forEach and method reference
IntStream.rangeClosed(1,100).forEach(System.out::println);

// using forEach and lambda expression
products.stream().forEach(p -> System.out.println("Name: " +
p.getTitle() + ", Weight" + p.getWeight()));
```

Απόσπασμα Κώδικα 3.22: Παράδειγμα χρήσης forEach

3.2.2.6 AnyMatch

Ένα κοινό πρότυπο στην επεξεργασία δεδομένων είναι η αναζήτηση στοιχείων τα οποία πληρούν ορισμένα κριτήρια. Η κατηγορία μεθόδων ταιριάσματος (match) των streams, επιστρέφουν Boolean τιμές. Συγκεκριμένα, η anyMatch, επιτρέπει την αναζήτηση τουλάχιστον ενός στοιχείου που να ταιριάζει σε κάποιο κατηγόρημα, και επιστρέφει true εάν βρεθεί, και false σε αντίθετη περίπτωση. Δέχεται ως όρισμα ένα αντικείμενο/συνάρτηση που υλοποιεί την διεπαφή Predicate, και μπορεί να είναι και μία έκφραση λάμδα.

Στο παρακάτω παράδειγμα, αναζητείται στην λίστα προϊόντων μίας αποθήκης, τουλάχιστον ένα προϊόν που να είναι παιχνίδι. Εάν βρεθεί εμφανίζεται κατάλληλο μήνυμα:

```
if(products.stream().anyMatch(p -> p.getTitle().equals("Toy"))) {
    System.out.println("There is at least one toy in the
warehouse.");
}
```

Απόσπασμα Κώδικα 3.23: Παράδειγμα anyMatch

3.2.2.7 AllMatch

Με την allMatch πραγματοποιείται ένας έλεγχος για το αν όλα τα στοιχεία ενός stream πληρούν ορισμένα κριτήρια. Επιστρέφεται true εάν τα πληρούν και false διαφορετικά. Και αυτή δέχεται ως όρισμα ένα αντικείμενο/συνάρτηση που υλοποιεί την διεπαφή Predicate.

Εάν για παράδειγμα, πρέπει σε μία λίστα γευμάτων ενός εστιατορίου να γίνει έλεγχος για το αν όλα τα πιάτα είναι αυστηρώς χορτοφαγικά, μπορεί να χρησιμοποιηθεί η allMatch με τον τρόπο που φαίνεται παρακάτω:

```
if(dishes.stream().allMatch(d -> d.getDescription().equals("vegan")))
{
    System.out.println("The menu is vegan");
}
```

Απόσπασμα Κώδικα 3.24: Παράδειγμα allMatch

3.2.2.8 NoneMatch

Η noneMatch είναι συμπληρωματική της allMatch, και ελέγχει εάν κανένα από τα στοιχεία ενός stream δεν πληροί ορισμένα κριτήρια. Επιστρέφει true αν δεν βρεθεί στοιχείο που τα πληροί και false σε αντίθετη περίπτωση.

Στο παρακάτω παράδειγμα, αναζητείται αν τα προϊόντα ενός καταστήματος είναι οικολογικά και δεν περιέχουν καθόλου πλαστικό.

```
if(products.stream().noneMatch(p ->
p.getMaterial().equals("plastic"))) {
    System.out.println("All products are eco friendly.");
}
```

Απόσπασμα Κώδικα 3.25: Παράδειγμα noneMatch

3.2.2.9 Collect

Η λειτουργία της συλλογής είναι και αυτή μία λειτουργία αναγωγής, η οποία συλλέγει τα δεδομένα ενός stream με έναν ορισμένο τρόπο. Η συνάρτηση που δίνει αυτή την δυνατότητα στα streams είναι η collect, η οποία δέχεται ως παράμετρο μία συνάρτηση που υλοποιεί την διεπαφή Collector και ορίζει με ποιον τρόπο θα συλλεχθούν τα δεδομένα. Ακόμη, μπορεί να χρησιμοποιηθεί για υπολογισμό αθροισμάτων, ελαχίστου/μεγίστου και για άλλους υπολογισμούς παρόμοιους με αυτούς της reduce. Η κλάση Collectors παρέχει ένα σύνολο έτοιμων τέτοιων συναρτήσεων, για τους πιο κοινούς τρόπους συλλογής, ωστόσο είναι εφικτό να δημιουργηθούν και συναρτήσεις προσαρμοσμένες στις ανάγκες του εκάστοτε προβλήματος.

Παρακάτω περιγράφονται ορισμένες σημαντικές μέθοδοι για συλλογή των δεδομένων με την collect, οι οποίες περιλαμβάνονται στην κλάση Collectors.

3.2.9.2.1 Collectors.toList()

Για την συλλογή των δεδομένων ενός stream σε ένα στιγμιότυπο της κλάσης List, μπορεί να χρησιμοποιηθεί η στατική μέθοδος toList() της κλάσης Collectors. Ωστόσο, η συγκεκριμένη μέθοδος δεν συλλέγει τα δεδομένα πάντα σε μία συγκεκριμένη υλοποίηση της κλάσης List, όπως π.χ. η ArrayList.

Στο παρακάτω παράδειγμα, ο στόχος είναι να συλλεχθούν σε μία λίστα όλα τα προϊόντα που είναι οικολογικά, θεωρώντας ως οικολογικά τα προϊόντα τα οποία δεν είναι φτιαγμένα από πλαστικό. Έτσι, αφού δημιουργηθεί ένα stream από την αρχική λίστα όλων των προϊόντων, φιλτράρονται αυτά των οποίων το υλικό δεν ισούται με «πλαστικό» και τέλος συλλέγονται σε μία λίστα, χρησιμοποιώντας την συνάρτηση collect με όρισμα την συνάρτηση Collectors.toList():

```
List<Product> ecoFriendly = products
        .stream()
        .filter(p -> !p.getMaterial().equals("plastic"))
        .collect(Collectors.toList());
```

Απόσπασμα Κώδικα 3.26: Παράδειγμα συλλογής δεδομένων σε λίστα

3.2.9.2.2 Collectors.toMap()

Ορισμένες φορές είναι απαραίτητο τα δεδομένα να συλλεχθούν στην μορφή κλειδιού-τιμής. Στην Java αυτός ο τύπος δεδομένων είναι η κλάση Map, με τις διάφορες υλοποιήσεις της (TreeMap, HashMap κλπ.). Για την συλλογή των δεδομένων σε map, μπορεί να χρησιμοποιηθεί μεταξύ άλλων, η στατική συνάρτηση toMap() της κλάσης Collectors. Αυτή δέχεται δύο παραμέτρους, η πρώτη είναι μία συνάρτηση για την επιστροφή του κλειδιού του map και η δεύτερη για την επιστροφή της τιμής που σχετίζεται με το κάθε κλειδί.

Στο παρακάτω παράδειγμα, θέλουμε από μία λίστα λέξεων να δημιουργηθεί ένα map το οποίο να περιέχει ως κλειδιά τις λέξεις και ως τιμή το μήκος των χαρακτήρων της κάθε λέξης. Από την αρχική λίστα λέξεων, δημιουργείται ένα stream, το οποίο στην συνέχεια συλλέγεται, καλώντας την collect, με όρισμα την συνάρτηση toMap. Στην toMap δίνονται δύο ορίσματα: 1) η συνάρτηση Function.identity(), η οποία απλά επιστρέφει αυτό που της δίνεται ως όρισμα, εν προκειμένω την κάθε λέξη, και 2) μία αναφορά μεθόδου, την String::length, η οποία υπολογίζει και επιστρέφει το μήκος της λέξης που δόθηκε ως όρισμα.

```
List<String> words = Arrays.asList("hello", "world", "this", "is",
"a", "beautiful", "day");

Map<String, Integer> wordLength = words.stream()
        .collect(Collectors.toMap(Function.identity(),
String::length));
```

Απόσπασμα Κώδικα 3.27: Παράδειγμα συλλογής δεδομένων σε map

3.2.9.2.3 Collectors.groupingBy()

Στην περίπτωση που τα δεδομένα ενός stream πρέπει να συλλεχθούν με ομαδοποίηση βάσει μίας ή περισσότερων ιδιοτήτων τους, μπορεί να χρησιμοποιηθεί η στατική συνάρτηση groupingBy() της κλάσης Collectors, η οποία επιστρέφει ένα Map. Δέχεται δύο παραμέτρους: 1) μία συνάρτηση η οποία επιστρέφει το κλειδί και 2) μία συνάρτηση η οποία ορίζει τον τρόπο με τον οποίο θα συλλεχθούν οι τιμές. Η δεύτερη παράμετρος είναι προαιρετική, και χρησιμοποιείται στην περίπτωση που οι τιμές πρέπει να συλλεχθούν από ένα σύνολο δεδομένων και δεν είναι μοναδικές.

Για παράδειγμα, έστω ότι σε μία αποθήκη πρέπει να συλλεχθούν τα προϊόντα με βάση το υλικό από το οποίο είναι φτιαγμένα. Αυτό σημαίνει ότι χρειάζεται ένα αντικείμενο τύπου Map, με κλειδί το υλικό (String) και τιμή μία λίστα προϊόντων που είναι φτιαγμένα από το συγκεκριμένο υλικό. Παρακάτω, παρουσιάζεται μία λύση, κατά την οποία από το stream των προϊόντων συλλέγονται με ομαδοποίηση τα προϊόντα βάσει του υλικού τους. Στην groupingBy, δίνεται ως όρισμα μία αναφορά μεθόδου, συγκεκριμένα η getMaterial της κλάσης Product, η οποία επιστρέφει το υλικό κάθε προϊόντος:

```
Map<String, List<Product>> productsByMaterial = products
        .stream()
        .collect(Collectors.groupingBy(Product::getMaterial));
```

Απόσπασμα Κώδικα 3.28: Παράδειγμα συλλογής με ομαδοποίηση

3.2.9.2.4 Collectors.partitioningBy()

Η συλλογή των δεδομένων με διαχωρισμό έχει ως αποτέλεσμα τα δεδομένα να χωρίζονται σε δύο μέρη βάσει ενός κριτηρίου. Το αποτέλεσμα είναι ένα αντικείμενο τύπου Map, με δύο μόνο καταχωρίσεις, μία με κλειδί true και τιμή τα στοιχεία που πληρούν το κριτήριο, και μία με κλειδί false και τιμή τα στοιχεία που δεν πληρούν το κριτήριο. Αυτό γίνεται με την χρήση της στατικής μεθόδου partitioningBy της κλάσης Collectors, δίνοντας ως όρισμα μία συνάρτηση ή ένα κατηγόρημα το οποίο επιστρέφει την τιμή true ή false.

Για παράδειγμα, έστω ότι τα προϊόντα πρέπει να χωριστούν σε «πολυτελείας» και μη, με τα «πολυτελείας» να είναι αυτά που η τιμή τους ξεπερνάει τα 200 ευρώ. Μία πιθανή

λύση φαίνεται στο παρακάτω απόσπασμα κώδικα, στο οποίο από την αρχική λίστα των προϊόντων, δημιουργείται ένα stream το οποίο συλλέγεται με χρήση της partitioningBy. Ως όρισμα στην partitioningBy δίνεται μία έκφραση λάμδα, η οποία επιστρέφει true αν η τιμή του προϊόντος είναι μεγαλύτερη από 200 και false σε αντίθετη περίπτωση. Το αποτέλεσμα είναι ένα αντικείμενο τύπου Map<Boolean, List<Product>>:

```
Map<Boolean, List<Product>> productsByPrice = products
        .stream()
        .collect(Collectors.partitioningBy(p -> p.getPrice() > 200));
```

Απόσπασμα Κώδικα 3.29: Παράδειγμα συλλογής δεδομένων με διαχωρισμό

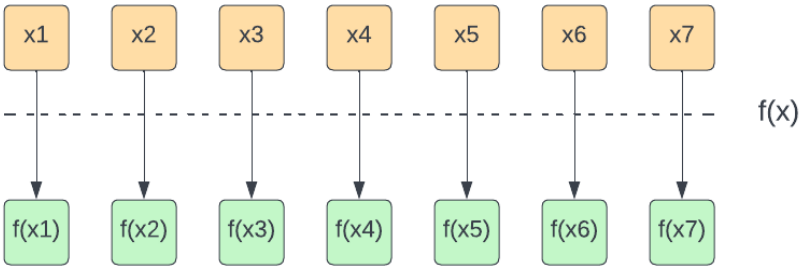
3.2.3 Stateful vs Stateless Operations

Στα παραδείγματα που παρουσιάστηκαν, οι περισσότερες ενδιάμεσες συναρτήσεις, όπως οι filter και map, δεν χρειάζονται να γνωρίζουν πληροφορίες για τα στοιχεία από προηγούμενα streams. Αυτές οι μέθοδοι ονομάζονται stateless, με την έννοια ότι δεν συγκρατούν την κατάσταση του προγράμματος για να πραγματοποιήσουν την λειτουργία τους. Στον αντίποδα, υπάρχουν οι stateful συναρτήσεις, για τις οποίες πληροφορίες για τα στοιχεία του stream είναι απαραίτητες για την δική τους λειτουργία. Για παράδειγμα, η ενδιάμεση συνάρτηση sorted, λαμβάνει τα στοιχεία ενός stream και τα ταξινομεί. Όμως για να πραγματοποιηθεί η ταξινόμηση, είναι απαραίτητο να είναι γνωστά όλα τα στοιχεία, εφόσον η θέση του κάθε στοιχείου εξαρτάται από τα υπόλοιπα.

Το αν μία συνάρτηση διατηρεί την κατάσταση (state) των δεδομένων, επηρεάζει την σειρά εκτέλεσης των συναρτήσεων σε μία αλυσίδα από streams.

3.3 Απεικόνιση και Αναγωγή (Map Reduce)

Το MapReduce είναι ένα προγραμματιστικό μοντέλο και η σχετική υλοποίηση για την επεξεργασία και την δημιουργία μεγάλων συνόλων δεδομένων [10]. Ο υπολογισμός χωρίζεται σε δύο λειτουργίες: την απεικόνιση (map) και την αναγωγή (reduce). Κατά την απεικόνιση (Διάγραμμα 4) εφαρμόζεται μία συνάρτηση σε κάθε στοιχείο των δεδομένων, η οποία τα τροποποιεί. Αυτή μπορεί να είναι κάτι απλό, όπως ο πολλαπλασιασμός κάθε στοιχείου με τον εαυτό του έτσι ώστε να προκύψει το τετράγωνο, έως κάτι αρκετά πιο πολύπλοκο, όπως η δημιουργία αντικειμένων σε κάποια γλώσσα αντικειμενοστραφούς προγραμματισμού ή η εφαρμογή περίπλοκων μαθηματικών πράξεων. Η αναγωγή (Διάγραμμα 3) είναι μία αθροιστική λειτουργία, στην οποία τα δεδομένα συλλέγονται πάλι βάσει κάποιας συνάρτησης. Πράξη αναγωγής μπορεί να είναι η άθροιση των στοιχείων ενός συνόλου, η εύρεση του γινομένου τους κ.α.



Διάγραμμα 4: Απεικόνιση

Οι έννοιες της απεικόνισης και της αναγωγής προέρχονται από τον συναρτησιακό προγραμματισμό, ωστόσο λόγω της χρησιμότητάς τους έχουν αναπτυχθεί ολόκληρα frameworks που τις αξιοποιούν. Δύο από τα γνωστότερα είναι τα Apache Hadoop και Apache Spark.

3.4 Streams και παραλληλοποίηση

Τα streams διευκολύνουν κατά πολύ την παράλληλη επεξεργασία, μιας και προσφέρουν έτοιμες συναρτήσεις, χωρίς να είναι αναγκαίο από τον προγραμματιστή να δημιουργήσει νήματα ή διεργασίες και να μοιράσει σε αυτά τις λειτουργίες. Για να πραγματοποιηθεί η επεξεργασία δεδομένων με streams με παράλληλο τρόπο, υπάρχουν δύο εναλλακτικές:

1. Δημιουργία ενός παράλληλου stream, με την parallelStream
2. Μετατροπή ενός stream σε παράλληλο, με την parallel

Όπως και στον δηλωτικό προγραμματισμό, έτσι και στον συναρτησιακό, και συγκεκριμένα στην περίπτωση των streams, πρέπει να δίνεται προσοχή στις περιπτώσεις στις οποίες πραγματοποιούνται διαδικασίες με παράλληλο τρόπο. Για την αποφυγή των λαθών και την παραγωγή έγκυρων αποτελεσμάτων, τα παράλληλα streams πρέπει να χρησιμοποιούνται σε stateless λειτουργίες.

Λεπτομέρειες της υλοποίησης, όπως ο αριθμός των νημάτων, αποκρύπτονται από τον χρήστη. Συγκεκριμένα, δημιουργούνται τόσα νήματα όσα και οι διαθέσιμοι επεξεργαστές του συστήματος [7]. Υπάρχει η δυνατότητα να αλλάξει ο προκαθορισμένος αριθμός των νημάτων, ωστόσο αυτό συστήνεται να αποφεύγεται.

3.5 Λύσεις προβλημάτων με parallel streams

Στο κεφάλαιο αυτό παρουσιάζονται λύσεις για ορισμένα κλασικά και μη προβλήματα χρησιμοποιώντας Java streams.

3.5.1 Άθροισμα διανύσματος με παραλληλισμό

Το πρόβλημα της άθροισης των στοιχείων ενός διανύσματος είναι ένα πολύ συχνό πρόβλημα. Στον προστακτικό προγραμματισμό, η απλούστερη και συνηθέστερη λύση είναι ένας βρόχος for ή while, με τον οποίο κάθε στοιχείο του διανύσματος προστίθεται σε κάθε επανάληψη σε μία μεταβλητή:

```
for(i=0;i<n;i++) {
    sum += array[i]
}
```

Προκειμένου ένα απλός υπολογισμός όπως ο παραπάνω να παραλληλιστεί, ένας προγραμματιστής πρέπει να κάνει αρκετά βήματα στον κώδικα. Πρέπει να δημιουργηθούν τα νήματα, τα οποία θα αναλάβουν να υπολογίσουν μέρος του τελικού αθροίσματος, ενώ μία διεργασία-συντονιστής, μπορεί να είναι το κυρίως πρόγραμμα, θα αναλάβει να αθροίσει τα επιμέρους αθροίσματα μόλις τα νήματα έχουν ολοκληρώσει τους υπολογισμούς. Επιπλέον, εάν κάθε νήμα προσθέτει το άθροισμα το οποίο υπολογίζει σε μία μοιραζόμενη μεταβλητή, τότε θα πρέπει να εφαρμοστούν και μηχανισμοί ταυτοχρονισμού, όπως ο αμοιβαίος αποκλεισμός, έτσι ώστε να εξασφαλιστεί η ορθή εγγραφή και ανάγνωση στην μοιραζόμενη μεταβλητή.

Όπως δείξαμε παραπάνω, με τα Java streams η διαδικασία απλουστεύεται κατά πολύ, ενώ υπάρχουν διάφοροι τρόποι να υπολογιστεί το άθροισμα. Μπορεί να χρησιμοποιηθεί η συνάρτηση sum, αν πρόκειται για stream primitive τύπων, ή η reduce χρησιμοποιώντας ως όρισμα μία έκφραση λάμδα ή μία αναφορά μεθόδου.

```
// Vector sum using sum()
int sum1 = Arrays.stream(ints).sum();

// Vector sum using reduce and lambda
int sum2 = Arrays.stream(ints).reduce((i,j) -> i+j).getAsInt();

// Vector sum using reduce and method reference
int sum3 = Arrays.stream(ints).reduce(Integer::sum).getAsInt();
```

Απόσπασμα Κώδικα 3.30: Άθροισμα διανύσματος με streams

Η παραπάνω διαδικασία μπορεί με τα streams να παραλληλιστεί εξαιρετικά εύκολα, καλώντας απλά την parallel κατά την δημιουργία των streams ακεραίων. Έτσι η διαδικασία παραλληλίζεται, οι επιμέρους υπολογισμοί ανατίθενται σε νήματα, χωρίς να είναι απαραίτητο αυτά να δημιουργηθούν ρητά από τον προγραμματιστή, και χωρίς να χρειάζεται αυτός να φροντίσει για την προστασία πιθανά μοιραζόμενων μεταβλητών.

```
// Vector parallel sum using sum()
int sum1 = Arrays.stream(ints).parallel().sum();

// Vector parallel sum using reduce and lambda
int sum2 = intStream.parallel().reduce((i,j) -> i+j).getAsInt();

// Vector parallel sum using reduce and method reference
int sum3 =
Arrays.stream(ints).parallel().reduce(Integer::sum).getAsInt();
```

Απόσπασμα Κώδικα 3.31: Άθροισμα διανύσματος με παραλληλισμό και streams

3.5.2 Υπολογισμός του π με ολοκλήρωση

Ο υπολογισμός του π με ολοκλήρωση βασίζεται στον χωρισμό του πεδίου τιμών σε μικρά τμήματα με ορισμένο μήκος και τον υπολογισμό του εμβαδού του κάθε τμήματος. Το σειριακό πρόγραμμα είναι το παρακάτω:

```
double step = 1.0 / (double)sections;

for (long i=0; i < sections; ++i) {
    double x = ((double)i+0.5)*step;
    sum += 4.0/(1.0+x*x);
}
double pi = sum * step;
```

Απόσπασμα Κώδικα 3.32: Σειριακός υπολογισμός π με ολοκλήρωση

Ο παραπάνω βρόχος μπορεί να υλοποιηθεί με streams, δημιουργώντας ένα stream καλώντας την iterate με βήμα 1. Αυτή παράγει άπειρους ακεραίους και γι' αυτό πρέπει να χρησιμοποιηθεί σε συνδυασμό με την limit, με την οποία το σύνολο των ακεραίων περιορίζεται στον αριθμό sections:

```
double step = 1.0 / (double)sections;

// calculate sum using streams
sum = Stream
        .iterate(0, i -> i+1)
        .limit(sections)
        .parallel()
        .mapToDouble(i -> {
            double x = ((double)i + 0.5) * step;
            return 4.0 / (1.0 + x*x);
        })
        .reduce(0.0, Double::sum);

double pi = sum * step;
```

Απόσπασμα Κώδικα 3.33: Παράλληλος υπολογισμός του π με streams

Στην παραπάνω λύση, δημιουργείται ένα stream ακεραίων, με στοιχεία τους αριθμούς από το 0 έως την τιμή της μεταβλητής sections. Το stream παραλληλοποιείται καλώντας την μέθοδο parallel. Στην συνέχεια, κάθε αριθμός του stream, μετατρέπεται σε έναν άλλο με την κλήση της συνάρτησης mapToDouble, βάσει του τύπου:

$$x = (i + 0.5) * step$$

και επιστρέφεται μία νέα τιμή που προκύπτει χρησιμοποιώντας την τιμή του x του προηγούμενου βήματος. Τέλος, το άθροισμα υπολογίζεται με την κλήση της συνάρτησης reduce, στην οποία δίνεται ως αρχική τιμή του αθροίσματος το 0, και ως συνάρτηση άθροισης, μία αναφορά μεθόδου στην στατική συνάρτηση sum της κλάσης Double.

3.5.3 Ιστόγραμμα χαρακτήρων

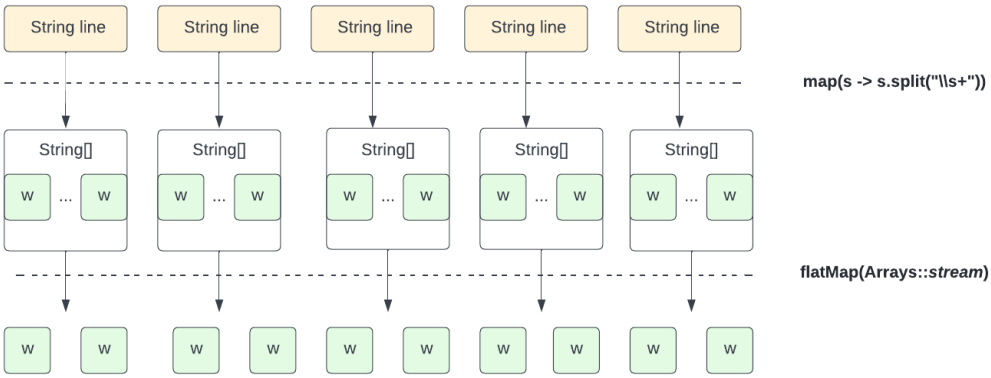
Το πρόβλημα του ιστογράμματος χαρακτήρων, αφορά την δημιουργία ενός πίνακα, κάθε θέση του οποίου έχει ως τιμή, το πλήθος εμφάνισης ενός χαρακτήρα σε ένα κείμενο. Η θέση που αντιστοιχίζεται στον τελικό πίνακα για κάθε χαρακτήρα, σχετίζεται με το σύνολο των χαρακτήρων των οποίων το πλήθος εμφάνισης αναζητείται. Έτσι για παράδειγμα, εάν οι χαρακτήρες που μας ενδιαφέρουν είναι οι αγγλικοί χαρακτήρες από το A έως το Z, τότε ο πίνακας του ιστογράμματος είναι ένας πίνακας με 26 θέσεις, όπου στην πρώτη αποθηκεύεται το πλήθος εμφάνισης του χαρακτήρα A, στην δεύτερη το πλήθος εμφάνισης του χαρακτήρα B, κ.ο.κ.

Πρώτο βήμα είναι η δημιουργία της λίστας των λέξεων από το κείμενο. Δεδομένου του αρχείου κειμένου, έστω text.txt, αυτό γίνεται όπως φαίνεται παρακάτω:

```
List<String> words = Files.lines(Path.of("resources/text.txt"))//
read file line by line to a stream
        .map(s -> s.split("\\s+")) // split each line to its words,
space separated
        .flatMap(Arrays::stream) // flatmap previous stream<String[]>
to stream<String>
        .map(w -> w.replaceAll("[,.]*", "")) // remove commas and
dots from each word
        .collect(Collectors.toList());
```

Απόσπασμα Κώδικα 3.34: Δημιουργία λίστα χαρακτήρων από λίστα λέξεων

Με την κλήση της στατικής μεθόδου lines της κλάσης Files, και δίνοντας ως όρισμα το μονοπάτι για ένα αρχείο, δημιουργείται ένα stream αλφαριθμητικών όπου κάθε στοιχείο του είναι μία γραμμή του αρχείου. Στην συνέχεια για κάθε ένα αλφαριθμητικό πραγματοποιείται μία λειτουργία απεικόνισης, κατά την οποία κάθε γραμμή χωρίζεται σε έναν πίνακα αλφαριθμητικών, όπου κάθε στοιχείο αυτού του πίνακα είναι μία λέξη. Αυτό γίνεται με την κλήση της μεθόδου map, με όρισμα μία έκφραση λάμδα, στην οποία για κάθε αλφαριθμητικό καλείται η μέθοδος split με την κατάλληλη κανονική έκφραση. Σε αυτό το σημείο το stream των αλφαριθμητικών έχει μετατραπεί σε ένα stream πινάκων αλφαριθμητικών και για να μπορέσει να γίνει περαιτέρω επεξεργασία κάθε λέξης, πρέπει να εξομαλυνθεί καλώντας την μέθοδο flatMap, η λειτουργιά της οποίας φαίνεται στο παρακάτω διάγραμμα:



Απόσπασμα Κώδικα 3.35: Εξομάλυνση stream πινάκων με την flatMap

Τέλος, για κάθε λέξη πραγματοποιείται μία ακόμα λειτουργία απεικόνισης, όπου με την κλήση της μεθόδου replaceAll και την κατάλληλη κανονικη έκφραση, αφαιρούνται από τις λέξεις τα κόμματα και οι τελείες. Οι λέξεις συλλέγονται σε μία λίστα αλφαριθμητικών καλώντας την collect.

Το επόμενο βήμα είναι η διάσπαση των λέξεων σε χαρακτήρες. Από την λίστα λέξεων του προηγούμενου βήματος, δημιουργείται ένα παράλληλο stream, στο οποίο για κάθε λέξη καλείται η μέθοδος flatMap με όρισμα μία έκφραση λάμδα η οποία διασπά την κάθε λέξη στους χαρακτήρες της. Οι χαρακτήρες συλλέγονται σε μία λίστα χαρακτήρων με την collect:

```
List<Character> chars = words
        .parallelStream()
        .flatMap(x -> x.chars().mapToObj(i -> (char) i))
        .collect(Collectors.toList());
```

Απόσπασμα Κώδικα 3.36: Δημιουργία stream χαρακτήρων από λέξεις

Το επόμενο βήμα είναι ο υπολογισμός των εμφανίσεων του κάθε χαρακτήρα. Για να πραγματοποιηθούν 26 επαναλήψεις, μία για κάθε χαρακτήρα του αγγλικού αλφαβήτου, δημιουργείται ο πίνακας freqs, ο οποίος σε κάθε θέση έχει ως τιμή τον αριθμό της θέσης. Έπειτα για κάθε στοιχείο αυτού του πίνακα πραγματοποιείται μία λειτουργία απεικόνισης, πάνω σε ένα παράλληλο stream, κατά την οποία μετρούνται οι χαρακτήρες οι οποίοι έχουν ίδια αριθμητική τιμή με αυτή του εκάστοτε στοιχείου. Αυτό γίνεται δημιουργώντας ένα επιπλέον stream από την λίστα των χαρακτήρων, φιλτράροντας μόνο αυτούς που έχουν ίδια αριθμητική τιμή με το i, και μετρώντας τους με την count. Έτσι, στην λίστα ακεραίων result, εμφανίζεται σε κάθε στοιχείο της το πλήθος εμφάνισης κάθε χαρακτήρα στο κείμενο:

```
Integer[] freqs = new Integer[26];
IntStream.rangeClosed(0,25).forEach(i -> freqs[i] = i);
```

```
List<Integer> result = Arrays.stream(freqs)
        .parallel()
        .map(i -> (int)
                chars.stream()
                        .filter(c -> Character.getNumericValue(c) ==
i+10)
                        .count())
        .collect(Collectors.toList());
```

Απόσπασμα Κώδικα 3.37: Υπολογισμός πλήθους εμφανίσεων χαρακτήρα με streams

3.5.4 Ταίριασμα αλφαριθμητικών

Το πρόβλημα του ταιριάσματος αλφαριθμητικών αναφέρεται στο μέτρημα του πλήθους των εμφανίσεων ενός συγκεκριμένου αλφαριθμητικού μέσα σε ένα κείμενο. Δοθέντος ενός κειμένου ο στόχος είναι να μετρηθεί πόσες φορές εμφανίζεται ένα μοτίβο ή μία λέξη μέσα σε αυτό. Έχουν προταθεί διάφορες λύσεις, με την χρήση ποικίλων δομών δεδομένων. Η πιο απλή, είναι μία εξαντλητική αναζήτηση (brute force) με την χρήση βρόχων for ή while.

```
for(int i=0; i<n-m; i++) {
    j = 0;
    while(j<m && pattern[j] == text[i+j]) {
        j++;
    }
    if(j == m) matches++;
}
```

Απόσπασμα Κώδικα 3.38: Εξαντλητική αναζήτηση string

Η παραλληλοποίηση του παραπάνω προβλήματος απαιτεί την δημιουργία νημάτων και τον κατάλληλο συντονισμό τους, έτσι ώστε να αναλαμβάνουν την ανάγνωση μέρους του κειμένου (πίνακας text) και να ενημερώνουν την μοιραζόμενη μεταβλητή (matches) για επιπλέον εμφάνιση του μοτίβου. Η εγγραφή σε μία μοιραζόμενη μεταβλητή απαιτεί επιπλέον εφαρμογή μηχανισμών ταυτοχρονισμού.

Το πρόβλημα λύνεται και παραλληλοποιείται εύκολα με την χρήση των streams. Αρχικά, διαβάζονται οι γραμμές ενός αρχείου σε ένα stream αλφαριθμητικών, και το stream παραλληλίζεται καλώντας την μέθοδο parallel. Έπειτα ακολουθούν οι διαδικασίες διαχωρισμού των λέξεων κάθε γραμμής, εξομάλυνσής του stream πινάκων που προκύπτει σε stream αλφαριθμητικών, και αφαίρεσης των κομμάτων και τελειών από κάθε λέξη, με τον ίδιο τρόπο που περιγράφηκε και στο πρόβλημα του ιστογράμματος. Τέλος, από το stream των λέξεων που έχει προκύψει φιλτράρονται και διατηρούνται μόνο αυτές που ταιριάζουν στο μοτίβο, με την κλήση της μεθόδου filter και όρισμα μία έκφραση λάμδα

που κάνει τον απαραίτητο έλεγχο. Οι λέξεις μετρούνται με την κλήση της τερματικής μεθόδου count:

```
long matches = Files.lines(Path.of("resources/text.txt")) // read
file line by line to a stream
        .parallel() // parallelize the stream
        .map(s -> s.split("\\s+")) // split each line to its words,
space separated
        .flatMap(Arrays::stream) // flatmap previous stream<String[]>
to stream<String>
        .map(w -> w.replaceAll("[,.]*", "")) // remove commas and
dots from each word
        .filter(w -> w.equals(pattern)) // filter out words that
don't match the pattern
        .count(); // count the words that match the pattern
```

Απόσπασμα Κώδικα 3.39: Ταίριασμα αλφαριθμητικών με streams

3.5.5 Μέσος όρος ακολουθίας DNA

Ένα θεμελιώδες πρόβλημα της βιο-πληροφορικής είναι η αντιστοίχιση ακολουθιών νουκλεοτιδίων σε ένα γονιδίωμα, έτσι ώστε να ταυτοποιηθούν παρόμοιες περιοχές, οι οποίες φανερώνουν δομικές και εξελικτικές σχέσεις [11]. Οι ακολουθίες είναι αποθηκευμένες σε αρχεία τύπου FASTQ [12], τα οποία περιέχουν τέσσερις γραμμές ανά εγγραφή:

1. Η πρώτη γραμμή ξεκινάει με το σύμβολο @ και ακολουθείται από ένα αναγνωριστικό της ακολουθίας
2. Η δεύτερη γραμμή περιέχει την ακολουθία των νουκλεοτιδίων
3. Η τρίτη γραμμή έχει μόνο το σύμβολο +
4. Η τέταρτη γραμμή αποτελείται από την ποιότητα της ακολουθίας με σύμβολα σύμφωνα με την κωδικοποίηση Phred [13], η οποία έχει υποχρεωτικά το ίδιο μήκος με αυτό της ακολουθίας νουκλεοτιδίων.

Μία τυπική εγγραφή είναι της μορφής:

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

Η ποιότητα λαμβάνει τιμές από το 0 έως το 40, και αναπαρίσταται από ένα σύμβολο του πίνακα ASCII ξεκινώντας από την θέση 33 έως 73. Μια συνηθισμένη μέθοδος αξιολόγησης των ακολουθιών είναι ο υπολογισμός του μέσου όρου της κάθε ακολουθίας. Καθώς ένα αρχείο FASTQ μπορεί να περιέχει εκατομμύρια εγγραφές, είναι σημαντικό οι υπολογισμοί να γίνονται αποδοτικά και ει δυνατόν με τρόπο παράλληλο.

Παρακάτω παρουσιάζεται μία λύση στο πρόβλημα χρησιμοποιώντας Java Streams. Αρχικά δημιουργήθηκε μία κλάση με όνομα FastQ, για την αναπαράσταση κάθε εγγραφής. Η κλάση έχει ως ιδιότητες το αναγνωριστικό της ακολουθίας, την ακολουθία των νουκλεοτιδίων, την ακολουθία των συμβόλων ποιοτήτων και τον μέσο όρο. Επιπλέον, δημιουργήθηκε μία βοηθητική κλάση για το διάβασμα και την συλλογή των δεδομένων από το αρχείο, η FastQCollector. Αυτή περιλαμβάνει την μέθοδο accept, στην οποία περιγράφεται το πώς θα διαβάζονται τα δεδομένα ανά τέσσερις γραμμές και θα δημιουργείται ένα αντικείμενο της κλάσης FastQ για να προστεθεί στην λίστα.

Στο κυρίως πρόγραμμα, το αρχείο διαβάζεται γραμμή προς γραμμή με streams, χρησιμοποιώντας την στατική μέθοδο lines της κλάσης Files. Καλώντας σε αυτό το stream την μέθοδο collect και δίνοντας ως όρισμα την μέθοδο collector της προσαρμοσμένη κλάσης FastQCollector, οι γραμμές συλλέγονται ανά τέσσερις και δημιουργείται ένα stream από αντικείμενα της κλάσης FastQ, που όμως δεν έχουν καταχωρημένο τον μέσο όρο της ακολουθίας τους. Στο επόμενο βήμα, αφού το stream παραλληλοποιηθεί καλώντας την μέθοδο parallel, κάθε στοιχείο FastQ περνάει από μία διαδικασία απεικόνισης, κατά την οποία δημιουργείτα ένα νέο αντικείμενο, αντίγραφο του προηγούμενου, για το οποίο υπολογίζεται και το μέσος όρος. Ο μέσος όρος υπολογίζεται λαμβάνοντας ένα stream των χαρακτήρων ποιοτήτων και μέσω αυτού το άθροισμά τους και στην συνέχεια διαιρώντας το άθροισμα με το πλήθος τους. Τέλος, τα νέα αντικείμενα FastQ, κάθε ένα από τα οποία περιέχει και την τιμή του μέσου όρου της ποιότητάς του, συλλέγονται σε μία λίστα χρησιμοποιώντας την μέθοδο collect με όρισμα την μέθοδο toList της κλάσης Collectors.

```
// read file and create the list of FastQs using FastQCollector
List<FastQ> fastQList = Files.lines(Path.of("resources/100K.fastq"))
        .collect(FastQCollector.collector())
        .stream()
        .parallel()
        .map(f -> {
                    // create a new FastQ object
                    FastQ newF = new
FastQ(f.getName(),f.getNucSequence(), f.getQualitySequence());

                    // set the avgQuality for the new FastQ

newF.setAvgQuality((f.getQualitySequence().chars()
                                .map(c -> c-33).sum() /
(double)f.getQualitySequence().chars().count()));
                    return newF;
                })
        .collect(Collectors.toList());
```

Απόσπασμα Κώδικα 3.40: Υπολογισμός μέσου όρου ακολουθίας νουκλεοτιδίων

4 Δημιουργία ενός συστήματος συστάσεων

4.1 Τί είναι τα συστήματα συστάσεων

Ένα πολύ κοινό πρόβλημα στις μεγάλες συλλογές δεδομένων, είναι η υπερφόρτωση πληροφορίας. Ένας χρήστης βρίσκεται ανάμεσα σε μια πληθώρα επιλογών, τις οποίες είναι σχεδόν αδύνατο να μπορέσει να τις αξιολογήσει όλες έτσι ώστε να καταλήξει σε αυτή που του ταιριάζει. Η έλευση του διαδικτύου έκανε αυτό το πρόβλημα ιδιαιτέρως απτό, με χαρακτηριστικό παράδειγμα (αλλά όχι μόνο) τις πλατφόρμες παροχής περιεχόμενου, όπως το Netflix ή το Spotify. Στις αρχές της δεκαετίες του 1990, άρχισαν να μπαίνουν τα θεμέλια για τα συστήματα συστάσεων, τα οποία φιλοδοξούν να βοηθήσουν τους χρήστες να πραγματοποιούν πληροφορημένες αποφάσεις, βασισμένες στις προτιμήσεις τους.

Ένα σύστημα συστάσεων είναι μία εφαρμογή η οποία παράγει συστάσεις ή προτάσεις για ορισμένα αντικείμενα ή οντότητες σε συγκεκριμένους χρήστες. Τα πρώτα συστήματα συστάσεων αναπτύχθηκαν την δεκαετία του 1990 και βασίζονταν στο συνεργατικό φιλτράρισμα. Επινοήθηκαν για να βοηθήσουν τους χρήστες να διαχειριστούν τον όγκο των διαθέσιμων πληροφοριών, δημιουργώντας μοντέλα προβλέψεων τα οποία παράγουν εκτιμήσεις σχετικά με το πόσο αρεστό θα είναι ένα αντικείμενο ή ένα σύνολο αντικειμένων σε ορισμένους χρήστες [14]. Έκτοτε, ο όρος έχει διευρυνθεί για να περιλαμβάνει συστήματα συστάσεων που λειτουργούν όχι μόνο με συνεργατικό φιλτράρισμα αλλά και με μεθόδους βασισμένες στο περιεχόμενο, σε δημογραφικά δεδομένα κ.α.

Οι βασικές οντότητες σε ένα σύστημα συστάσεων είναι οι χρήστες (users) και τα αντικείμενα (items). Ως χρήστης θεωρείται ένας οποιοσδήποτε μεμονωμένος λογαριασμός στο σύστημα και αντιπροσωπεύει τις προτιμήσεις ενός ατόμου. Ως αντικείμενο θεωρείται μία οντότητα που μπορεί να προταθεί ανεξάρτητα. Αντικείμενο μπορεί να είναι κάθε προϊόν σε ένα ηλεκτρονικό κατάστημα, κάθε ταινία σε μία πλατφόρμα streaming, κάθε τραγούδι σε μία εφαρμογή αναπαραγωγής μουσικής, κ.ο.κ. Οι χρήστες εκφράζουν προτιμήσεις για διάφορα αντικείμενα και λαμβάνουν συστάσεις για άλλα. Στα συστήματα που βασίζονται στις αξιολογήσεις (rating based), ο συνδετικός κρίκος μεταξύ των δύο βασικών οντοτήτων είναι η βαθμολογία (rating) των χρηστών για τα αντικείμενα, ενώ σε συστήματα που βασίζονται στο περιεχόμενο είναι απαραίτητες και διάφορες πληροφορίες

που αφορούν το περιεχόμενο των αντικειμένων, όπως για παράδειγμα το είδος της ταινίας ή του τραγουδιού, κ.α. [15]

Στην πιο απλή μορφή του ένα σύστημα συστάσεων μπορεί απλά να προωθεί στους χρήστες το πιο δημοφιλές περιεχόμενο, αυτό που έχει παραδείγματος χάριν αναπαραχθεί τις περισσότερες φορές ή αυτό που έχει τις καλύτερες αξιολογήσεις. Αυτή η στρατηγική ωστόσο αγνοεί τις εξειδικευμένες προτιμήσεις του κάθε χρήστη και καθιστά το σύστημα μη-εξατομικευμένο. Μία απλή επέκταση αυτού του μοντέλου προς μία πιο εξατομικευμένη κατεύθυνση είναι ευκόλως εφικτή, εφόσον το σύστημα έχει βασικές γνώσεις για τα είδη που προτιμά ο χρήστης και μπορεί να πραγματοποιήσει μία απλή κατηγοριοποίηση των διαθέσιμων αντικειμένων ως προς το περιεχόμενό τους. Αυτές οι στρατηγικές οδήγησαν στην ανάπτυξη των αλγορίθμων συνεργατικού φιλτραρίσματος [16]. Οι συγκεκριμένοι αλγόριθμοι βασίζονται στην (εύλογη) υπόθεση ότι οι άνθρωποι έχουν σχετικά σταθερές προτιμήσεις. Έτσι, εάν δύο άτομα έχουν συμφωνήσει στο παρελθόν ως προς ένα αντικείμενο είναι πολύ πιθανό να συνεχίσουν να συμφωνούν και στο μέλλον. Περισσότερες λεπτομέρειες για τα συστήματα που χρησιμοποιούν συνεργατικό φιλτράρισμα, θα ακολουθήσουν σε επόμενο κεφάλαιο.

Η χρήση των συστημάτων συστάσεων μπορεί να επιφέρει μία σειρά από πλεονεκτήματα, τόσο στους παρόχους όσο και στους χρήστες. Το πιο προφανές και άμεσο είναι η προώθηση, και κατ' επέκταση πώληση, περισσότερων αντικειμένων. Προτείνοντας στους χρήστες αντικείμενα που είναι πιο κοντά στα ενδιαφέροντά τους, είναι πιθανότερο αυτοί να καταλήξουν να τα αγοράζουν, αυξάνοντας έτσι την κερδοφορία του παρόχου. Ακόμα, αντικείμενα που διαφέρουν και δεν ανήκουν στα πιο δημοφιλή μπορούν να φτάσουν στον κατάλληλο χρήστη μέσω ενός συστήματος συστάσεων που αναγνωρίζει συγκεκριμένες προτιμήσεις. Ένας πάροχος στην στρατηγική που ακολουθεί για την διαφήμιση των προϊόντων του, προτιμά συνήθως να προωθεί τα πιο ευπώλητα και δημοφιλή καθώς η διαφήμιση αυτών ενέχει μικρότερο ρίσκο. Αυτή η τακτική ωστόσο, μπορεί να αφήσει στην αφάνεια μία πληθώρα διαθέσιμων επιλογών.

Από την πλευρά του χρήστη, τα συστήματα συστάσεων αυξάνουν την ικανοποίηση που λαμβάνει κατά την χρήση μίας πλατφόρμας. Οι εξατομικευμένες προτάσεις βοηθούν στην επιλογή μεταξύ ενός μεγάλου όγκου δεδομένων, οδηγώντας τον χρήστη να απολαμβάνει περισσότερο τις υπηρεσίες. Επιπλέον, όσο περισσότερο ένας χρήστης χρησιμοποιεί μία συγκεκριμένη πλατφόρμα, τόσο καλύτερα μπορεί το συγκεκριμένο σύστημα συστάσεων να τον «γνωρίσει». Οι περισσότερες λεπτομέρειες για τα

ενδιαφέροντα του χρήστη, οδηγούν με την σειρά τους σε ακόμη πιο στοχευμένες συστάσεις και σε μεγαλύτερη αφοσίωση του χρήστη στην συγκεκριμένη υπηρεσία.

4.1.1 Προβλήματα και προκλήσεις

Η ανάπτυξη και η λειτουργία των συστημάτων συστάσεων, συνοδεύεται μέχρι και σήμερα από μία σειρά προκλήσεων που αφορούν σε μεγάλο βαθμό την απόκτηση και διαχείριση των δεδομένων των χρηστών. Παρακάτω αναλύονται τα σημαντικότερα προβλήματα των σύγχρονων συστημάτων συστάσεων.

1. Πρόβλημα του πρώτου βαθμολογητή ή Cold Start

   Το συγκεκριμένο πρόβλημα προκύπτει όταν προστίθενται στο σύστημα νέοι χρήστες ή νέα αντικείμενα. Στην πρώτη περίπτωση, όταν εγγράφεται ένας νέος χρήστης, είναι αδύνατο να υπάρχει γνώση των προτιμήσεών του και συνεπώς να παραχθούν αποτελεσματικές συστάσεις. Μία συνηθισμένη λύση είναι να ζητείται από τον χρήστη να εισάγει κάποια δεδομένα σχετικά με τα ενδιαφέροντά του κατά την εγγραφή ή να βαθμολογήσει αντικείμενα για τα οποία έχει κάποια παρελθοντική εμπειρία. Στην δεύτερη, το νέο αντικείμενο δεν μπορεί να έχει κάποια αρχική βαθμολογία σύμφωνα με την οποία θα προταθεί σε χρήστες, αλλά ούτε και προηγούμενους χρήστες για να εντοπιστούν όμοιοί τους στους οποίους θα προταθεί το αντικείμενο. Λύση στο συγκεκριμένο πρόβλημα προσφέρουν οι αλγόριθμοι βάσει περιεχομένου, οι οποίοι κατατάσσουν το αντικείμενο σε μια κατηγορία με βάση το περιεχόμενό ή το είδος του.

2. Επεκτασιμότητα (Scalability)

   Καθώς ο όγκος των δεδομένων, δηλαδή οι βαθμολογίες των χρηστών αυξάνεται, οι υπολογισμοί που πραγματοποιούν οι αλγόριθμοι συνεργατικού φιλτραρίσματος απαιτούν περισσότερο χρόνο για την επεξεργασία τους. Ένας αλγόριθμος πρέπει να είναι επεκτάσιμος, με την έννοια ότι εξακολουθεί να λειτουργεί αποτελεσματικά και χωρίς σημαντικές καθυστερήσεις όσο τα δεδομένα αυξάνονται. Αλγόριθμοι οι οποίοι παράγουν χρήσιμα αποτελέσματα με μικρό όγκο δεδομένων μπορεί να αποδειχθούν αναποτελεσματικοί με τον πολλαπλασιασμό τους [15]. Έχουν προταθεί διάφορες λύσεις για το πρόβλημα της επεκτασιμότητας, οι οποίες είναι εξειδικευμένες στον τύπο του

αλγορίθμου. Κάποιες από αυτές είναι η μείωση των διαστάσεων με τεχνικές SVD και η ομαδοποίηση (clustering) των δεδομένων [17].

3. Αραιά δεδομένα (Sparsity)

   Στους αλγορίθμους της οικογένειας του συνεργατικού φιλτραρίσματος, το συγκεκριμένο πρόβλημα είναι πολύ συχνό. Τα αρχικά δεδομένα που είναι απαραίτητα ως είσοδο στους συγκεκριμένους αλγορίθμους, είναι οι βαθμολογίες των χρηστών επί των αντικείμενων, κάτι που συνήθως μοντελοποιείται ως ένας πίνακας βαθμολογιών. Εάν αυτές οι βαθμολογίες δεν είναι αρκετές, δηλαδή οι χρήστες έχουν βαθμολογήσει λίγα μόνο αντικείμενα, τότε ο πίνακας είναι «αραιός» [18]. Το πρόβλημα των αραιών δεδομένων έχει ως αποτέλεσμα την δυσκολία στον υπολογισμών ακριβών συστάσεων. Για την επίλυση του συγκεκριμένου προβλήματος έχουν προταθεί διάφορες λύσεις όπως τεχνικές SVD (Singular Value Decomposition), χρήση δημογραφικών δεδομένων και χρήση τεχνικών αλγορίθμων συστάσεων με βάση το περιεχόμενο [19].

4. Συνωνυμία (Synonymy)

   Ως συνωνυμία εννοείται η τάση ορισμένων όμοιων αντικειμένων να ονοματίζονται με διαφορετικές ονομασίες [17]. Τα περισσότερα συστήματα συστάσεων δεν έχουν την ικανότητα να εντοπίζουν αυτή την λανθάνουσα συσχέτιση μεταξύ των αντικειμένων και τα αντιμετωπίζουν με διαφορετικό τρόπο. Παραδείγματος χάρις, ένας αλγόριθμος συνεργατικού φιλτραρίσματος που βασίζεται στην μνήμη, δεν έχει τρόπο να αντιληφθεί την ομοιότητα μεταξύ αντικειμένων με ονόματα “horror film” και “horror movie”, οπότε και δεν λαμβάνει υπόψιν την ομοιότητά τους κατά τους υπολογισμούς. Μία τεχνική για την αντιμετώπιση του προβλήματος είναι οι μέθοδοι Latent Semantic Indexing (LSI) [19] και SVD (Singular Value Decomposition) [20].

5. Αφάνεια (Latency)

   Οι αλγόριθμοι συνεργατικού φιλτραρίσματος αντιμετωπίζουν το πρόβλημα της αφάνειας όταν νέα αντικείμενα προστίθενται στην συλλογή και το σύστημα προωθεί μόνο τα βαθμολογημένα αντικείμενα και όχι τα καινούρια που δεν έχουν λάβει ακόμα καμία βαθμολογία [19]. Οι αλγόριθμοι που βασίζονται σε κατηγορίες σε συνδυασμό με ένα μοντέλο «στερεοτυπικού χρήστη» μπορούν να αντιμετωπίσουν το συγκεκριμένο πρόβλημα [21].

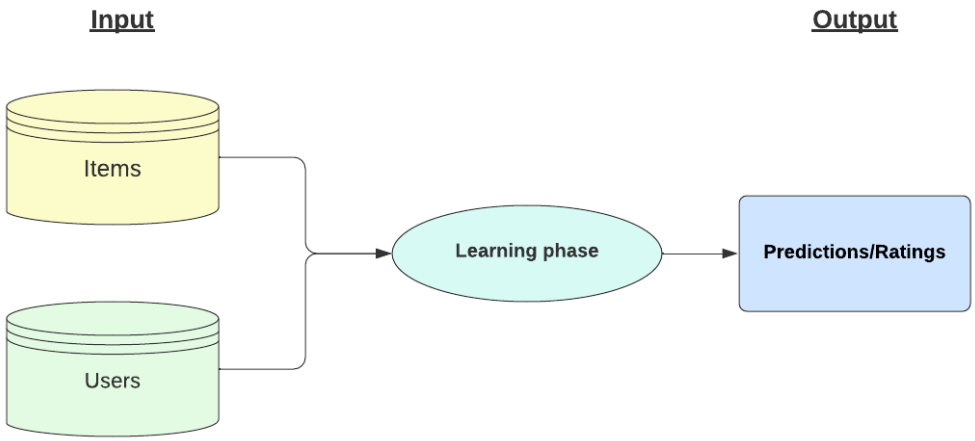
6. Πρόβλημα του «Ασυνήθιστου χρήστη»

   Το πρόβλημα του «Ασυνήθιστου χρήστη» είναι γνωστό και ως “Grey Sheep Problem”. Ανακύπτει και αυτό σε συστήματα που χρησιμοποιούν αλγορίθμους συνεργατικού φιλτραρίσματος, και αναφέρεται σε χρήστες των οποίων η άποψη δεν συμφωνεί ή διαφωνεί με συνεπή τρόπο [22]. Ο συνδυασμός των αλγορίθμων συνεργατικού φιλτραρίσματος με αλγορίθμους βασισμένους στο περιεχόμενο μπορεί να βοηθήσει στην παραγωγή καλύτερων συστάσεων [23].

4.2 Αλγόριθμοι

Ο σκοπός των συστημάτων συστάσεων είναι να παράγουν συστάσεις ή προβλέψεις των βαθμολογιών αντικειμένων για τους χρήστες [24]. Λόγω της ευρείας χρήσης του διαδικτύου τις τελευταίες δεκαετίες, η ανάγκη για συστήματα συστάσεων γίνεται όλο και πιο επιτακτική και για αυτόν τον λόγο, έχουν προταθεί και υλοποιηθεί ένας μεγάλος αριθμός αλγορίθμων και βελτιώσεών τους.

Η διαδικασία που ακολουθούν οι περισσότεροι αλγόριθμοι, φαίνεται στο παρακάτω διάγραμμα.

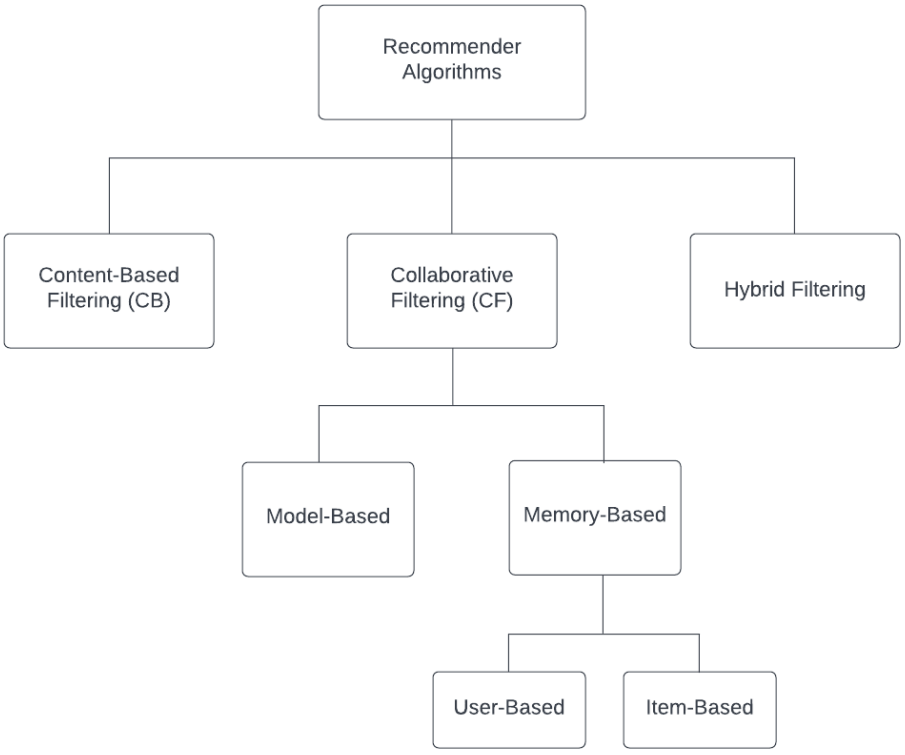


Διάγραμμα 5: Γενική διαδικασία αλγορίθμου συστάσεων

Ως είσοδος είναι απαραίτητα τα δεδομένα για τα αντικείμενα και τους χρήστες. Αυτά, ανάλογα με τον αλγόριθμο μπορεί να είναι βαθμολογίες και αξιολογήσεις, δεδομένα σε σχέση με το περιεχόμενο των αντικειμένων, δημογραφικά δεδομένα για τους χρήστες, γενικά ενδιαφέροντα των χρηστών, κ.α. Τα δεδομένα της φάσης εισόδου υπόκεινται μία επεξεργασία κατά την φάση της εκμάθησης, η οποία μπορεί να υπολογίζει παρόμοια αντικείμενα ή παρόμοιους χρήστες. Στο τελικό στάδιο, ο αλγόριθμος μπορεί να παράγει

ως έξοδο μία λίστα με προβλέψεις βαθμολογιών ή ένα σύνολο προτεινόμενων αντικειμένων για κάποιον χρήστη.

Οι αλγόριθμοι συστάσεων μπορούν να ταξινομηθούν ανάλογα με τον τρόπο με τον οποίο επεξεργάζονται τα δεδομένα, όπως φαίνεται στο παρακάτω διάγραμμα.



Διάγραμμα 6: Ταξινόμηση αλγορίθμων συστάσεων

Στα παρακάτω κεφάλαια παρουσιάζονται επιγραμματικά τα βασικότερα χαρακτηριστικά αυτών των αλγορίθμων, ενώ ιδιαίτερο βάρος δίνεται μόνο σε ορισμένους αλγορίθμους συνεργατικού φιλτραρίσματος, καθώς τέτοιοι υλοποιήθηκαν στο σύστημα συστάσεων με Java Streams.

Εκτός από τους αλγορίθμους του διαγράμματος υπάρχουν και άλλοι όπως αυτοί που χρησιμοποιούν δημογραφικά δεδομένα, βάσεις γνώσεων, συστήματα ομαδοποίησης, βασισμένοι σε γράφους κ.α. [25]. Ωστόσο, δεν θα αναφερθούν εκτενώς, καθώς υπερβαίνουν τα όρια και τους σκοπούς αυτής της μελέτης.

4.2.1 Σημειογραφία

Όπως αναφέρθηκε στο εισαγωγικό κεφάλαιο, οι δύο κυριότερες οντότητες ενός συστήματος συστάσεων είναι οι χρήστες και τα αντικείμενα. Αυτοί οι όροι είναι σκόπιμα γενικευμένοι και ουδέτεροι έτσι ώστε να περικλείουν όλους τους τύπους των

προβλημάτων. Παρακάτω περιγράφεται η μαθηματική σημειογραφία που θα χρησιμοποιηθεί στην συνέχεια αυτής της εργασίας και αναφέρεται κυρίως στους αλγορίθμους που χρησιμοποιούν το συνεργατικό φιλτράρισμα, μιας και αυτοί αποτελούν το κυρίως θέμα της εργασίας.

Ένας χρήστης (user) αντιπροσωπεύει έναν μοναδικό λογαριασμό στο σύστημα και κάνουμε την υπόθεση ότι εκφράζει τα ενδιαφέροντα ή τις προτιμήσεις ενός ατόμου. Θα συμβολίζουμε το σύνολο των χρηστών ως U, και ως $u \in U$, έναν οποιοσδήποτε χρήστη που ανήκει στο σύνολο.

΄Ενα αντικείμενο (item) αντιπροσωπεύει ένα μοναδικό *πράγμα* στο σύστημα το οποίο μπορεί να προταθεί σε κάποιον χρήστη. Ένα αντικείμενο μπορεί να είναι ένα υλικό αντικείμενο σε μια πλατφόρμα όπως ένα ηλεκτρονικό κατάστημα, ή να είναι περιεχόμενο όπως ένα τραγούδι ή μια ταινία σε μία streaming πλατφόρμα. Θα συμβολίζουμε το σύνολο των αντικειμένων ως I, και ως $i \in I$, ένα οποιοδήποτε αντικείμενο που ανήκει στο σύνολο.

Μία βαθμολογία (rating) αντιπροσωπεύει μία τιμή (συνήθως αριθμητική), την οποία ανέθεσε ένας χρήστης σε ένα αντικείμενο ως αποτέλεσμα της ικανοποίησής του από αυτό. Θα συμβολίζουμε το σύνολο των βαθμολογιών ως R, και ως $r_{ui} \in R$, την βαθμολογία r που ανέθεσε ένας χρήστης u σε ένα αντικείμενο i.

Χρήστες (Users)	Το σύνολο των χρηστών στο σύστημα	U
	Ένας συγκεκριμένος χρήστης	$u \in U$
Αντικείμενα (Items)	Το σύνολο των αντικειμένων στο σύστημα	I
	Ένα συγκεκριμένο αντικείμενο	$i \in I$
Βαθμολογίες (Ratings)	Το σύνολο των βαθμολογιών στο σύστημα	R
	Μία συγκεκριμένη βαθμολογία του χρήστη u για το αντικείμενο i	$r_{ui} \in R$
Βαθμολογίες χρήστη	Το σύνολο των αντικειμένων που βαθμολόγησε ο χρήστης u	I_u
	Το διάνυσμα των βαθμολογιών του χρήστη u	r_u
Βαθμολογίες αντικειμένου	Το σύνολο των χρηστών που βαθμολόγησαν το αντικείμενο i	U_i

	Το διάνυσμα των βαθμολογιών για το αντικείμενο i	r_i

Πίνακας 1: Σημειογραφία όρων ενός συστήματος συστάσεων

4.2.2 Συνεργατικό φιλτράρισμα (Collaborative Filtering)

Το συνεργατικό φιλτράρισμα είναι ο πιο δημοφιλής και ο πιο ευρέως χρησιμοποιούμενος αλγόριθμος για τα συστήματα συστάσεων. Βασίζεται στη εύρεση όμοιων χρηστών (ή αντικειμένων), υποθέτοντας πώς χρήστες με παρόμοια ενδιαφέροντα, δηλαδή αυτοί που έχουν παρόμοιες βαθμολογίες για ένα σύνολο αντικειμένων, θα έχουν γενικότερα παρόμοιες προτιμήσεις [24] [26]. Εάν για παράδειγμα ο στόχος είναι να προταθούν αντικείμενα σε έναν χρήστη u, τότε πρέπει να εντοπιστούν χρήστες που του μοιάζουν ως προς τις προηγούμενες βαθμολογίες και οι τελικές προτάσεις για τον u να είναι αντικείμενα που έχουν βαθμολογήσει θετικά οι χρήστες που είναι παρόμοιοι με αυτόν.

Το περιεχόμενο και το είδος των αντικειμένων καθώς και το ιδιαίτερο προφίλ του κάθε χρήστη δεν παίζουν κανέναν ρόλο στο συνεργατικό φιλτράρισμα, όπως συμβαίνει στο φιλτράρισμα με βάση το περιεχόμενο. Το αν ένας χρήστης προτιμά ταινίες του είδους «horror» είναι άσχετο με τον τρόπο λειτουργίας και παραγωγής των συστάσεων. Η μόνη απαραίτητη είσοδος είναι οι βαθμολογίες των χρηστών ως προς τα αντικείμενα. Αυτές συνήθως παρέχονται στην μορφή ενός πίνακα, όπου οι γραμμές αντιστοιχούν στους χρήστες και οι στήλες στα αντικείμενα. Στην παρακάτω εικόνα απεικονίζεται ένας τέτοιος πίνακας:

	$Item_1$	$Item_2$	$Item_3$	…	$Item_n$
$User_1$	$r_{11} = 5$				$r_{1n} = 4$
$User_2$					
…					
$User_{n-1}$			$r_{(n-1)3} = 2$		
$User_n$	$r_{n1} = 1$		$r_{n3} = 3$		

Διάγραμμα 7: Πίνακας βαθμολογιών χρηστών-αντικειμένων

Οι μέθοδοι συνεργατικού φιλτραρίσματος κατηγοριοποιούνται ανάλογα με τον τρόπο λειτουργίας τους σε αυτές που βασίζονται στο μοντέλο (model based) και σε αυτές

που βασίζονται στην μνήμη (memory based). Οι δύο αυτές κατηγορίες περιγράφονται παρακάτω.

4.2.2.1 Memory Based

Σημαντικό ρόλο στους αλγορίθμους αυτούς παίζουν τα αντικείμενα που ο κάθε χρήστης έχει ήδη βαθμολογήσει, καθώς χρησιμοποιούνται για την εύρεση άλλων χρηστών «γειτονικών» ως προς αυτόν, γι' αυτό και αναφέρονται και ως αλγόριθμοι βάσει γειτνίασης [27]. Μόλις εντοπιστούν παρόμοιοι χρήστες, μπορούν να χρησιμοποιηθούν διάφοροι αλγόριθμοι για τον συνδυασμό των προτιμήσεών τους και την δημιουργία συστάσεων [28]. Ολόκληρος ο πίνακας των βαθμολογιών χρησιμοποιείται για τον υπολογισμό. Αυτή η κατηγορία αλγορίθμων έχει σημαντικά ποσοστά επιτυχίας και γι' αυτό χρησιμοποιείται ευρέως.

Υπάρχουν δύο βασικές κατηγορίες αλγορίθμων σε αυτήν την οικογένεια:

α) Βασισμένοι στον χρήστη (user-based), όπου υπολογίζεται η ομοιότητα μεταξύ των χρηστών συγκρίνοντας τις βαθμολογίες τους για τα ίδια αντικείμενα. Η πρόβλεψη της βαθμολογίας για ένα αντικείμενο για τον τρέχοντα χρήστη προκύπτει από τον σταθμισμένο μέσο όρο των βαθμολογιών των όμοιων χρηστών για το συγκεκριμένο αντικείμενο.

β) Βασισμένοι στα αντικείμενα (item-based), όπου υπολογίζεται η ομοιότητα μεταξύ των αντικειμένων, και όχι των χρηστών. Η πρόβλεψη γίνεται με βάση τα αντικείμενα που έχει βαθμολογήσει ο τρέχων χρήστης, για τα οποία αναζητούνται παρόμοια αντικείμενα με αυτά που έχουν ήδη αρέσει στον χρήστη. Υπάρχουν διάφορες μέθοδοι για τον υπολογισμό της ομοιότητας μεταξύ των αντικείμενων, με τις πιο δημοφιλείς να είναι αυτές που βασίζονται στην συσχέτιση (correlation-based) και αυτές που βασίζονται στο συνημίτονο (cosine-based) [28]. Η συγκεκριμένη κατηγορία αλγορίθμων, θα αναλυθεί αναλυτικότερα παρακάτω, εφόσον μέρος της εργασίας αποτελεί η υλοποίηση ενός από αυτούς με java streams.

4.2.2.2 Model Based

Οι memory based αλγόριθμοι χρησιμοποιούν κάθε φορά ολόκληρο το σύνολο των δεδομένων των βαθμολογιών για τον υπολογισμό των προβλέψεων. Αυτό μπορεί να έχει σημαντικό κόστος στην ταχύτητα όταν τα δεδομένα είναι πολλά. Ως βελτίωση, προτάθηκαν αλγόριθμοι που δημιουργούν ένα μοντέλο χρησιμοποιώντας ένα μέρος των προηγούμενων βαθμολογιών. Οι προτάσεις δημιουργούνται βασισμένες σε αυτό το

μοντέλο, χωρίς να είναι απαραίτητη κάθε φορά η προσπέλαση και επεξεργασία του συνόλου των δεδομένων. Η δημιουργία του μοντέλου μπορεί να γίνει χρησιμοποιώντας τεχνικές μηχανικής μάθησης ή εξόρυξης δεδομένων. Μερικές από αυτές τις τεχνικές είναι οι SVD (Singular Value Decomposition), Τεχνική Συμπλήρωσης Μητρώων (Matrix Completion Technique), τεχνικές οπισθοδρόμησης (Regression) και ομαδοποίησης (Clustering) [28].

4.2.2.3 Πλεονεκτήματα & μειονεκτήματα

Τα μειονεκτήματα του συνεργατικού φιλτραρίσματος είναι κατά βάση αυτά που αναφέρθηκαν στο εισαγωγικό κεφάλαιο, όπως τα αραιά δεδομένα, το πρόβλημα του πρώτου βαθμολογητή, η συνωνυμία και η επεκτασιμότητα. Για κάποια από αυτά έχουν προταθεί διάφορες μέθοδοι αντιμετώπισής τους, όπως περιγράφεται στο προηγούμενο κεφάλαιο.

Στον αντίποδα, το συνεργατικό φιλτράρισμα, σε αντίθεση με το φιλτράρισμα με βάση το περιεχόμενο, δεν απαιτεί καμία γνώση επί του περιεχομένου των αντικειμένων ή του προφίλ των χρηστών. Η μόνη απαραίτητη είσοδος για τον υπολογισμό των προβλέψεων είναι οι βαθμολογίες των χρηστών για τα αντικείμενα. Η αποτελεσματικότητά του, το έχει καταστήσει μία από τις πιο δημοφιλής και ευρέως χρησιμοποιούμενες τεχνικές σε εφαρμογές του πραγματικού κόσμου.

4.2.3 Φιλτράρισμα με βάση το περιεχόμενο (Content-Based filtering)

Τα συστήματα συστάσεων που χρησιμοποιούν αλγορίθμους με βάση το περιεχόμενο, προτείνουν στον χρήστη αντικείμενα που είναι παρόμοια με αυτά που του έχουν αρέσει στο παρελθόν. Η ομοιότητα των αντικειμένων υπολογίζεται βάσει των χαρακτηριστικών των αντικειμένων που συγκρίνονται [29]. Για παράδειγμα, σε μία πλατφόρμα με ταινίες, εάν ο χρήστης έχει βαθμολογήσει θετικά μία ταινία του είδους «ρομαντική κομεντί», το σύστημα θα συνεχίσει να του προτείνει ταινίες που κατατάσσονται σε αυτό το είδος.

Στα συγκεκριμένα συστήματα, δημιουργείται αρχικά ένα προφίλ για κάθε χρήστη, το οποίο περιέχει πληροφορίες σχετικά με τις προτιμήσεις και τα ενδιαφέροντά του. Αυτό μπορεί να δημιουργηθεί κατά την εγγραφή του χρήστη στην πλατφόρμα, ζητώντας του να βαθμολογήσει αντικείμενα ή να υποδείξει ρητά τα ενδιαφέροντά του. Η σύσταση του προφίλ ωστόσο, συνεχίζεται όσο ο χρήστης χρησιμοποιεί την πλατφόρμα και βαθμολογεί νέα αντικείμενα με τα οποία αλληλοεπιδρά, έτσι ώστε να παραμένει ενημερωμένο. Στην

συνέχεια, το προφίλ του χρήστη συγκρίνεται με τα χαρακτηριστικά των αντικειμένων και προτείνονται αυτά που κρίνονται ως «ταιριαστά» [19].

Τα χαρακτηριστικά των αντικειμένων εξάγονται συνήθως από λέξεις-κλειδιά που υπάρχουν στην περιγραφή τους και είναι κατά βάση κειμενικού χαρακτήρα. Οι περιγραφές μπορούν να εξαχθούν από ιστοσελίδες, emails, άρθρα ή τις ρητές περιγραφές του ίδιου του προϊόντος. Οι ιδιότητες που έχουν σημασία για το σύστημα εξαρτώνται από τον τομέα του προβλήματος. Για παράδειγμα, σε μία πλατφόρμα streaming για ταινίες, είναι σημαντικά χαρακτηριστικά όπως οι ηθοποιοί, ο σκηνοθέτης, το είδος, κ.α. Αυτά τα χαρακτηριστικά δεν είναι δομημένα ούτε αριθμητικά, έτσι ανακύπτει το ζήτημα του πώς θα χρησιμοποιηθούν με τρόπο που να έχει νόημα στο σύστημα. Η πιο απλή προσέγγιση είναι η σύγκριση αλφαριθμητικών, έτσι ώστε να εντοπιστούν τα προφίλ χρηστών που ταιριάζουν με κάθε αντικείμενο. Αυτή η απλοϊκή όμως μέθοδος δημιουργεί προβλήματα όσον αφορά την πολλαπλή σημασιολογία των λέξεων αλλά και την συνωνυμία, όπου διάφορες λέξεις μπορεί να έχουν το ίδιο νόημα. Μία από τις πιο αποτελεσματικές λύσεις είναι η σημασιολογική ανάλυση (semantic analysis), στην οποία χρησιμοποιούνται λεξικά και βάσεις γνώσεων (knowledge bases) για να ερμηνευτούν σημασιολογικά οι προτιμήσεις των χρηστών και τα χαρακτηριστικά των αντικειμένων [30].

4.2.3.1 Πλεονεκτήματα & μειονεκτήματα

Οι μέθοδοι φιλτραρίσματος με βάση το περιεχόμενο εξαλείφουν ορισμένα προβλήματα του συνεργατικού φιλτραρίσματος. Τα νέα αντικείμενα που προστίθενται στο σύστημα δεν είναι απαραίτητο να έχουν λάβει βαθμολογίες για να προταθούν, καθώς οι προτάσεις βασίζονται στο αντικείμενο αυτό καθ’ εαυτό και στο περιεχόμενό του και όχι σε βαθμολογίες παρόμοιων αντικειμένων. Ακόμα και αν στην βάση δεν υπάρχουν καταχωρημένες καθόλου προτιμήσεις χρηστών, δεν επηρεάζεται η ακρίβεια των συστάσεων [28]. Έτσι, εξαλείφεται το πρόβλημα του πρώτου βαθμολογητή που ανακύπτει στις μεθόδους συνεργατικού φιλτραρίσματος.

Τα προφίλ των χρηστών είναι ανεξάρτητα το ένα από το άλλο και δεν υπάρχει ανάγκη συσχέτισής τους για την παραγωγή συστάσεων. Σε αντίθεση με το συνεργατικό φιλτράρισμα, κατά το οποίο πρέπει να εντοπιστούν χρήστες με παρόμοια ενδιαφέροντα, στο φιλτράρισμα με βάση το περιεχόμενο το προφίλ του χρήστη δημιουργείται από πληροφορίες που αφορούν αποκλειστικά αυτόν και τις προτιμήσεις του που εκφράζει με έμμεσο ή άμεσο τρόπο. Τέλος, ο τρόπος που παράγονται οι προτάσεις είναι διαφανής, με

την έννοια ότι το σύστημα μπορεί να παρέχει επακριβείς εξηγήσεις σε σχέση με το πώς πρότεινε ένα αντικείμενο σε έναν συγκεκριμένο χρήστη. Ένας χρήστης για παράδειγμα έχει εκφράσει το ενδιαφέρον του για ταινίες που ανήκουν στο είδος «κωμωδία» και γι' αυτό η πλατφόρμα του πρότεινε την ταινία «Life of Brian».

Παρά τα πολύ σημαντικά πλεονεκτήματα, η μέθοδος CB δεν είναι φυσικά απολύτως αψεγάδιαστη. Ένα σημαντικό μειονέκτημα είναι η λεγόμενη υπέρ-εξειδίκευση περιεχομένου (*content over-specialization*) [28]. Οι συγκεκριμένοι αλγόριθμοι δεν έχουν τρόπο να εντοπίζουν και να προτείνουν αντικείμενα που παρεκκλίνουν με κάποιον τρόπο από τα ενδιαφέροντα του χρήστη, έτσι το προτεινόμενο περιεχόμενο είναι πάντα πολύ κοντά σε αυτό στο οποίο ο χρήστης έχει δείξει εμφανή προτίμηση. Αν για παράδειγμα ο χρήστης αντέδρασε θετικά σε μία ταινία με σκηνοθέτη την *Jane Campion,* θα λαμβάνει συνεχώς προτάσεις ταινιών από την ίδια σκηνοθέτη.

Για να λειτουργήσει το φιλτράρισμα με βάση το περιεχόμενο πρέπει να δεχθεί ως είσοδο έναν αριθμό από πληροφορίες σχετικές με το περιεχόμενο των αντικειμένων. Εάν αυτές δεν υπάρχουν, τότε δεν μπορούν να παραχθούν κατάλληλες προτάσεις. Έτσι στις περιπτώσεις στις οποίες το αντικείμενο δεν συνοδεύεται από μία λεπτομερή περιγραφή ή από ένα σύνολο μετά-δεδομένων, είναι αδύνατο να ενταχθεί στο σύνολο των συστάσεων. Το συγκεκριμένο πρόβλημα είναι γνωστό και ως περιορισμένη ανάλυση περιεχομένου (*limited content analysis*) [30]. Τέλος, αν και αυτή η μέθοδος αντιμετωπίζει το πρόβλημα του cold-start από την πλευρά των αντικειμένων, δεν μπορεί να το κάνει και από την πλευρά του χρήστη. Ένας χρήστης μόλις εγγραφεί στην πλατφόρμα, είναι αδύνατο να λάβει προτάσεις πριν αξιολογήσει κάποια αντικείμενα ή πριν δώσει κάποιες πληροφορίες για τα ενδιαφέροντά του, έτσι ώστε να συσταθεί το προφίλ του.

4.2.4 Υβριδικό φιλτράρισμα

Οι τεχνικές υβριδικού φιλτραρίσματος αποτελούν συνδυασμό του συνεργατικού φιλτραρίσματος και του φιλτραρίσματος με βάση το περιεχόμενο και έχουν αναπτυχθεί προκειμένου να αντιμετωπίσουν ορισμένα προβλήματα των δύο προηγούμενων αλλά και να βελτιώσουν την αποδοτικότητά και την ακρίβεια των συστάσεων. Ο συνδυασμός των δύο τεχνικών μπορεί να γίνει με έναν από τους παρακάτω τρόπους [25]:

1. Μεμονωμένη υλοποίηση συνεργατικού φιλτραρίσματος και φιλτραρίσματος με βάση το περιεχόμενο και συνένωση των αποτελεσμάτων τους

2. Ενσωμάτωση ορισμένων χαρακτηριστικών φιλτραρίσματος με βάση το περιεχόμενο σε ένα σύστημα με συνεργατικό φιλτράρισμα
3. Ενσωμάτωση ορισμένων χαρακτηριστικών συνεργατικού φιλτραρίσματος σε σύστημα με φιλτράρισμα με βάση το περιεχόμενο
4. Σύστημα που ενοποιεί χαρακτηριστικά και από τις δύο τεχνικές

Οι τεχνικές φιλτραρίσματος με βάση το περιεχόμενο επιτρέπουν την παραγωγή συστάσεων ακόμα και όταν δεν υπάρχουν διαθέσιμες βαθμολογίες, αντιμετωπίζοντας έτσι το πρόβλημα του πρώτου βαθμολογητή (cold start). Από την άλλη οι τεχνικές συνεργατικού φιλτραρίσματος παράγουν πιο ακριβή αποτελέσματα. Τα συστήματα που χρησιμοποιούν υβριδικό φιλτράρισμα, έχουν την δυνατότητα συνδυάζοντας τις δύο τεχνικές να αντιμετωπίσουν αυτά τα προβλήματα.

Οι τεχνικές υβριδικού φιλτραρίσματος μπορούν να ταξινομηθούν περαιτέρω ως εξής [31]:

- Σταθμισμένο (weighted): οι συστάσεις προκύπτουν από τον συνδυασμό των αποτελεσμάτων διαφορετικών συστημάτων. Αρχικά, τα αποτελέσματα κάθε συστήματος μπορεί να είναι ισοβαρή, στην συνέχεια όμως, καθώς αξιολογούνται από τους χρήστες, λαμβάνουν μία διαφορετική τιμή ως βάρος για τον υπολογισμό της συνεισφοράς τους στο τελικό αποτέλεσμα.
- Με εναλλαγή (switching): το σύστημα έχει την δυνατότητα να εναλλάσσει δυναμικά την τεχνική του φιλτραρίσματος ανάλογα με τα τρέχοντα δεδομένα. Για παράδειγμα, εάν δεν υπάρχουν διαθέσιμες βαθμολογίες το σύστημα μπορεί να επιλέξει να χρησιμοποιήσει φιλτράρισμα με βάση το περιεχόμενο, έτσι ώστε να αντιμετωπιστεί το πρόβλημα του πρώτου βαθμολογητή.
- Μικτό (mixed): η τελική λίστα των συστάσεων αποτελείται από αποτελέσματα που προέκυψαν χρησιμοποιώντας τόσο συνεργατικό φιλτράρισμα όσο φιλτράρισμα με βάση το περιεχόμενο ταυτοχρόνως. Αυτή η τεχνική μπορεί να αντιμετωπίσει το πρόβλημα του νέου αντικειμένου, καθώς μπορεί να χρησιμοποιήσει φιλτράρισμα με βάση το περιεχόμενο όταν ένα νέο αντικείμενο εισάγεται στα δεδομένα, και να το προτείνει παρά το γεγονός ότι δεν υπάρχουν ακόμα διαθέσιμες βαθμολογίες για αυτό.

- Με αλληλουχία (cascade): σε αυτήν την μέθοδο, εφαρμόζεται πρώτα μία τεχνική φιλτραρίσματος και τα αποτελέσματά της βελτιώνονται εφαρμόζοντας μία δεύτερη τεχνική φιλτραρίσματος.
- Συνδυασμός χαρακτηριστικών (feature combination): σε αυτό το μοντέλο ορισμένα χαρακτηριστικά από μία μέθοδο φιλτραρίσματος χρησιμοποιούνται για την βελτίωση των αποτελεσμάτων μία άλλης μεθόδου. Για παράδειγμα, ο υπολογισμός της ομοιότητας των χρηστών μπορεί να χρησιμοποιηθεί ως επιπλέον δεδομένο σε ένα σύστημα που χρησιμοποιεί φιλτράρισμα με βάση το περιεχόμενο.
- Επαύξηση χαρακτηριστικών (feature augmentation): μία τεχνική φιλτραρίσματος χρησιμοποιείται για την παραγωγή συστάσεων και αυτή η πληροφορία ενσωματώνεται σε ένα δεύτερο στάδιο στην επεξεργασία των δεδομένων από το επόμενο φιλτράρισμα. Σε αντίθεση με την μέθοδο της αλληλουχίας, όπου τα αποτελέσματα συνδυάζονται με προτεραιότητα, στην επαύξηση χαρακτηριστικών, τα χαρακτηριστικά που χρησιμοποιούνται στο δεύτερο στάδιο περιλαμβάνουν τα αποτελέσματα του πρώτου.
- Μετα-επιπέδου (meta-level): ολόκληρο το μοντέλο που έχει δημιουργηθεί σε ένα πρώτο στάδιο φιλτραρίσματος, δίνεται ως είσοδος στο επόμενο στάδιο. Αυτού του είδους η υβριδοποίηση μπορεί να λύσει το πρόβλημα των αραιών δεδομένων (sparsity) [28].

4.3 Use Case: Σύστημα Συστάσεων με Java streams

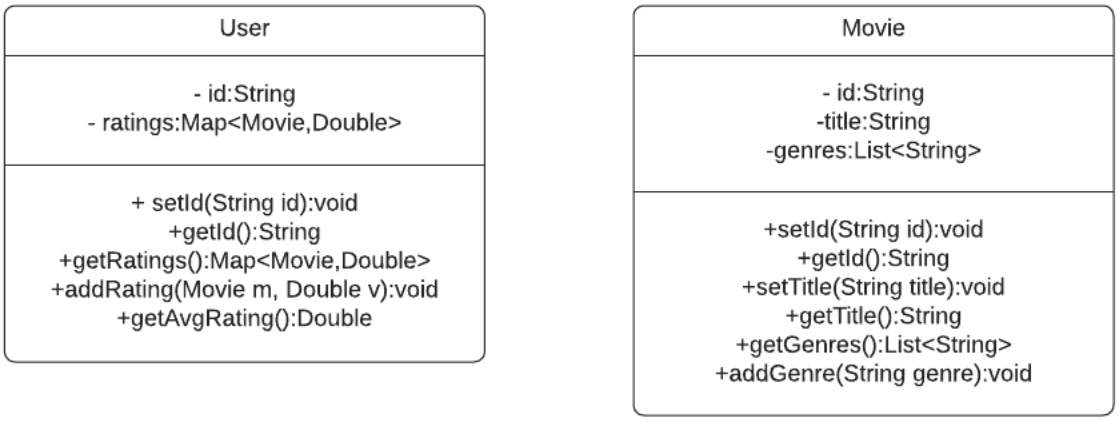
Το κυρίως μέρος της παρούσας εργασίας αποτελεί η ανάπτυξη ενός συστήματος συστάσεων, εφαρμόζοντας ορισμένες τεχνικές, χρησιμοποιώντας java streams για την υλοποίηση των αλγορίθμων. Στα παρακάτω κεφάλαια, παρουσιάζονται οι αλγόριθμοι που υλοποιήθηκαν, παραθέτοντας σε σημεία και μέρη του κώδικα για την καλύτερη κατανόηση του τρόπου λειτουργίας τους.

Οι αλγόριθμοι που υλοποιούνται χρησιμοποιούν συνεργατικό φιλτράρισμα για τις προβλέψεις. Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, στο συνεργατικό φιλτράρισμα βασικό εργαλείο αποτελεί ο πίνακας βαθμολογιών χρηστών-αντικειμένων (βλ. Διάγραμμα 7), ο οποίος και θα χρησιμοποιηθεί παρακάτω σε όλες τις υλοποιήσεις. Οι αλγόριθμοι υλοποιούνται με παράλληλο τρόπο, που όμως είναι αδιαφανής, καθώς καλούνται μέθοδοι

του Stream API οι οποίες παραλληλοποιούν τις διαδικασίες μιας αλυσίδας streams, χωρίς να είναι αναγκαία η ρητή δημιουργία και διαχείριση νημάτων και μηχανισμών ταυτοχρονισμού.

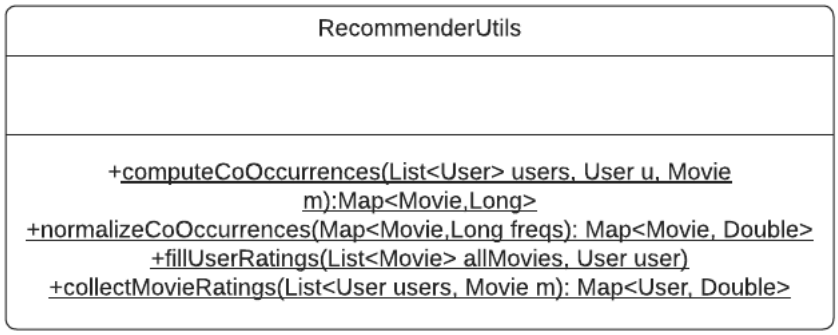
4.3.1 Περιγραφή μοντέλου στην Java

Για την υλοποίηση χρησιμοποιήθηκαν δεδομένα από το Movie Lens (https://movielens.org/), τα οποία αφορούν χρήστες οι οποίοι βαθμολογούν ένα σύνολο ταινιών. Τα δεδομένα είναι σε μορφή csv, και περιλαμβάνουν πολλές πληροφορίες για τις ταινίες, συμπεριλαμβανομένων και ορισμένων για το περιεχόμενό τους. Τα δεδομένα που αξιοποιήθηκαν στο πλαίσιο της εργασίας είναι το αναγνωριστικό του χρήστη (id), το αναγνωριστικό της ταινίας (id), ο τίτλος της ταινίας, τα είδη της ταινίας (genres) (για απλή ανάγνωση και όχι για χρήση τους στην διαδικασία παραγωγής των συστάσεων), και οι βαθμολογίες του κάθε χρήστη για τις ταινίες. Με βάση τα παραπάνω δημιουργήθηκαν οι παρακάτω κλάσεις:



Διάγραμμα 8: Κλάσεις μοντέλου στην Java

Επιπλέον, δημιουργήθηκε η κλάση RecommenderUtils, στην οποία υλοποιούνται ορισμένες λειτουργίες που είναι κοινές στους αλγορίθμους (όπως ο υπολογισμός των συν-εμφανίσεων), καθώς και κάποιες ακόμα βοηθητικές μέθοδοι, των οποίων η λειτουργία θα αναλυθεί παρακάτω:



Διάγραμμα 9: Κλάση RecommenderUtils

4.3.2 Ένας απλός αλγόριθμος απεικόνισης και αναγωγής

Ο πρώτος αλγόριθμος που μελετάμε είναι ένας απλός αλγόριθμος που χρησιμοποιεί την τεχνική απεικόνισης και αναγωγής (map-reduce) και τον πολλαπλασιασμό πινάκων [32]. Το φιλτράρισμα είναι βασισμένο στα αντικείμενα και όχι στους χρήστες, δεδομένου ότι το σύνολο των δεδομένων αφορά βαθμολογίες χρηστών επί των ταινιών και οι βαθμολογίες των χρηστών είναι πιο πιθανό να αλλάζουν συχνότερα σε σχέση με το σύνολο των ταινιών. Οι σχέσεις μεταξύ των αντικειμένων δημιουργούνται αποκλειστικά με βάση τις δοθείσες βαθμολογίες τους, καθώς χρησιμοποιούμε συνεργατικό φιλτράρισμα, και όχι άλλα χαρακτηριστικά, όπως π.χ. το είδος τους ή ο/η σκηνοθέτης, που βασίζονται στο περιεχόμενο.

Βασική έννοια, η οποία θα χρησιμοποιηθεί και στις υλοποιήσεις των επόμενων αλγορίθμων είναι αυτή του πίνακα συν-εμφανίσεων (co-occurrence matrix) [33]. Σε αυτόν τον πίνακα αποθηκεύονται αφού υπολογιστούν οι συχνότητες με τις οποίες οποιεσδήποτε δύο ταινίες εμφανίζονται μαζί, έχουν δηλαδή βαθμολογηθεί από τον ίδιο χρήστη.

Υποθέτουμε ότι το σύνολο των βαθμολογιών των χρηστών επί των ταινιών είναι το παρακάτω (εύρος βαθμολογιών 1-5):

	Inception	**Pretty Woman**	**Pulp Fiction**	**Django**	**The power of the dog**	**12 Years a slave**
Sara	4	2			3	
Thomas			5	4	3	
Joanna	3	1				4
Mary		2		2		5
Bill	4		4			1

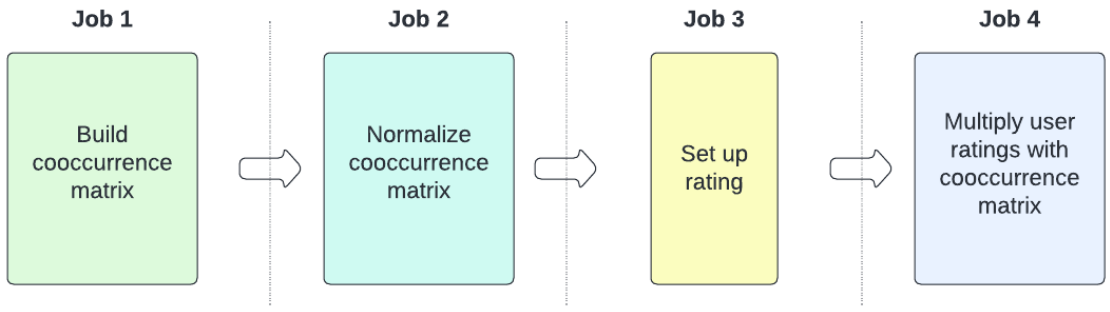
Πίνακας 2: Πίνακας Βαθμολογιών χρηστών-ταινιών

Για τον υπολογισμό των συν-εμφανίσεων πρέπει για κάθε ζεύγος ταινιών να βρεθεί το πλήθος των φορών που και οι δύο ταινίες έχουν βαθμολογηθεί από διαφορετικούς χρήστες. Για παράδειγμα, παρατηρούμε στον παραπάνω πίνακα ότι οι ταινίες Inception και Pretty Woman έχουν βαθμολογηθεί και από την Sara αλλά και από την Joanna. Συνεπώς, το πλήθος των συν-εμφανίσεών τους ισούται με 2. Για τον παραπάνω πίνακα βαθμολογίων, ο πίνακας συν-εμφανίσεων είναι ο παρακάτω:

	Inception	**Pretty Woman**	**Pulp Fiction**	**Django**	**The Power of the Dog**	**12 Years a Slave**
Inception	3	2	1	0	1	2
Pretty Woman	2	2	0	1	1	2
Pulp Fiction	1	1	2	1	1	1
Django	0	1	1	2	1	1
The Power of the Dog	1	1	1	1	2	0
12 Years a Slave	2	2	1	1	0	3

Πίνακας 3: Παράδειγμα πίνακα συν-εμφανίσεων

4.3.2.1 Περιγραφή του αλγορίθμου



Διάγραμμα 10: Βήματα αλγορίθμου map reduce

Ο πυρήνας της λειτουργίας αυτού του αλγορίθμου είναι ο πολλαπλασιασμός πινάκων και η τεχνική απεικόνισης και αναγωγής. Απεικόνιση πραγματοποιείται στα πρώτα βήματα, όπου με είσοδο τις βαθμολογίες των χρηστών, παράγονται οι συν-εμφανίσεις, ενώ η αναγωγή εφαρμόζεται στο τελευταίο βήμα, με τον πολλαπλασιασμό των συν-εμφανίσεων με τις αρχικές βαθμολογίες για την εύρεση της προβλεπόμενης

βαθμολογίας. Δεδομένου του πίνακα βαθμολογιών, ο αλγόριθμος υλοποιείται σε τέσσερα βασικά βήματα:

1. Υπολογισμός συν-εμφανίσεων: ο υπολογισμός των συν-εμφανίσεων για κάθε ζεύγος ταινιών και η διαμόρφωση του αντίστοιχου πίνακα γίνεται όπως παρουσιάστηκε στο προηγούμενο κεφάλαιο
2. Κανονικοποίηση συν-εμφανίσεων: σε αυτό το βήμα, κάθε κελί του πίνακα συν-εμφανίσεων διαιρείται με το άθροισμα των συν-εμφανίσεων κάθε γραμμής προκειμένου να κανονικοποιηθούν. Δεδομένου του πίνακα συν-εμφανίσεων του προηγούμενου παραδείγματος (Πίνακας 3), οι κανονικοποιημένες συν-εμφανίσεις είναι οι παρακάτω:

	Inception	**Pretty Woman**	**Pulp Fiction**	**Django**	**The Power of the Dog**	**12 Years a Slave**
Inception	3/9	2/9	1/9	0	1/9	2/9
Pretty Woman	2/8	2/8	0	1/8	1/8	2/8
Pulp Fiction	1/7	1/7	2/7	1/7	1/7	1/7
Django	0	1/6	1/6	2/6	1/6	1/6
The Power of the Dog	1/6	1/6	1/6	1/6	2/6	0
12 Years a Slave	2/9	2/9	1/9	1/9	0	3/9

Πίνακας 4: Κανονικοποιημένος πίνακας συν-εμφανίσεων

3. Διαμόρφωση βαθμολογιών: στον πίνακα βαθμολογιών οι κενές βαθμολογίες του κάθε χρήστη πρέπει να συμπληρωθούν με τον μέσο όρο των βαθμολογιών του. Ο πίνακας του προηγούμενου κεφαλαίου διαμορφώνεται όπως φαίνεται παρακάτω:

	Inception	**Pretty Woman**	**Pulp Fiction**	**Django**	**The power of the dog**	**12 Years a slave**
Sara	4	2	3	3	3	3
Thomas	4	4	5	4	3	4
Joanna	3	1	2.66	2.66	2.66	4
Mary	3	2	3	2	3	5
Bill	4	3	4	3	3	1

Πίνακας 5: Πίνακας βαθμολογιών με συμπληρωμένους μέσους όρους κάθε χρήστη

4. Πολλαπλασιασμός διανυσμάτων: στο τελευταίο βήμα, αυτό από το οποίο προκύπτουν οι προτάσεις, ο κανονικοποιημένος πίνακας συν-εμφανίσεων πολλαπλασιάζεται με το διάνυσμα βαθμολογιών του κάθε χρήστη. Έτσι, για το παραπάνω παράδειγμα για τον χρήστη Sara, προκύπτουν οι εξής βαθμολογίες:

	Inception	PW	PF	Django	TPotD	12YaS
Inception	3/9	2/9	1/9	0	1/9	2/9
PW	2/8	2/8	0	1/8	1/8	2/8
PF	1/7	1/7	2/7	1/7	1/7	1/7
Django	0	1/6	1/6	2/6	1/6	1/6
TPotD	1/6	1/6	1/6	1/6	2/6	0
12YaS	2/9	2/9	1/9	1/9	0	3

x

Sara
4
2
? (3)
? (3)
3
? (3)

=

Sara
3.11
3
3
2.83
3
3

Recommendations

Διάγραμμα 11: Προτεινόμενες βαθμολογίες αλγορίθμου map reduce

4.3.2.2 Υλοποίηση με Java streams

Παρακάτω, περιγράφεται η υλοποίηση με Java streams των βημάτων του αλγορίθμου που παρουσιάστηκε. Τα streams παραλληλοποιούνται σε όλα τα σημεία στα οποία η παράλληλη επεξεργασία δεν δημιουργεί πρόβλημα και αλλαγές στις τιμές των αποτελεσμάτων, καλώντας είτε την parallel είτε την parallelStream.

Στο πρώτο βήμα υπολογίζονται οι συν-εμφανίσεις. Γι' αυτήν την λειτουργία υλοποιήθηκε η μέθοδος computeCoOccurrences() στην κλάση RecommenderUtils, η οποία δέχεται ως ορίσματα α) μία λίστα με όλους τους υπάρχοντες χρήστες, β) τον χρήστη για τον οποίο πρόκειται να υπολογιστεί/προβλεφθεί βαθμολογία και γ) την ταινία για την οποία θα υπολογιστεί/προβλεφθεί βαθμολογία, και επιστρέφει ένα Map με κλειδιά τις ταινίες με τις οποίες συνδυάζεται η δοθείσα ταινία προς βαθμολόγηση, και τιμές την αντίστοιχη συν-εμφάνιση.

```
public static Map<Movie, Long> computeCoOccurrences(List<User> users,
User u, Movie m) {

    Map<Movie, Double> userRatings = u.getRatings();

    // contains the co-occurrences
    List<Map<Movie, Long>> freqsMap = userRatings.keySet().stream()
            .parallel()


            .map(rating -> users.stream().filter(user ->
                            !user.equals(u) &&
user.getRatings().containsKey(rating) &&
user.getRatings().containsKey(m)).collect(Collectors.toList())
                    .stream().map(user -> new Combo(1L, rating))
                    .collect(Collectors.groupingBy(Combo::getMovie,
```

```
Collectors.counting())))
            .collect(Collectors.toList());

    // the frequencies list flattened
    return freqsMap.parallelStream()
            .flatMap(map -> map.entrySet().stream())
            .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue));
}
```

Απόσπασμα Κώδικα 4.1: Υπολογισμός συν-εμφανίσεων

Στην πρώτη ακολουθία, δημιουργείται μία λίστα από Maps με κλειδιά ταινίες και τιμές την τιμή των συν-εμφανίσεων. Ξεκινώντας από το σύνολο των βαθμολογιών του χρήστη, που είναι ένα Map με ζεύγη ταινίας-βαθμολογίας, ανακτάται το σύνολο των κλειδιών, το οποίο και επεξεργάζεται ως stream με παράλληλο τρόπο. Πάνω σε αυτό το stream πραγματοποιείται μία λειτουργία απεικόνισης (map) για κάθε μία ταινία-κλειδί (rating). Στο εσωτερικό stream, για κάθε μία από αυτές τις ταινίες, φιλτράρονται οι χρήστες οι οποίοι έχουν βαθμολογήσει τόσο την ταινία για την οποία θα παραχθεί βαθμολογία (m), όσο και την ταινία-όρισμα στην έκφραση λάμδα (rating) και συλλέγονται σε μία λίστα. Από αυτή την λίστα δημιουργείται εκ νέου ένα stream, στο οποίο για κάθε χρήστη στην λειτουργία της απεικόνισης δημιουργείται ένα αντικείμενο της βοηθητικής κλάσης Combo με τιμή 1 και ταινία την αρχική ταινία του πρώτου εξωτερικού stream. Αυτό το stream συλλέγεται με την χρήση της μεθόδου Collectors.groupingBy(), παράγοντας ένα Map με κλειδιά τις ταινίες και τιμές το αθροιστικό σύνολο των προηγούμενων αντικειμένων Combo, που προκύπτει δίνοντας ως δεύτερο όρισμα της Collectors.groupingBy() την μέθοδο Collectors.counting(). Σε αυτό το σημείο, το σύνολο των Map συλλέγονται σε μία λίστα. Και οι δύο υπολογισμοί πραγματοποιούνται παράλληλα, εφόσον στα streams που δημιουργούνται καλείται η μέθοδος parallel(). Η παραλληλοποίηση γίνεται με τρόπο αδιαφανή ως προς τον προγραμματιστή.

Προκειμένου να είναι πιο κατανοητό αλλά και πιο διαχειρίσιμο το αποτέλεσμα, η λίστα με τα Maps, εξομαλύνεται σε ένα επίπεδο, με το τελευταίο stream και την μέθοδο flatMap(), εξάγοντας τελικά τις συν-εμφανίσεις σε ένα Map με κλειδί την κάθε ταινία και τιμή την τιμή της συν-εμφάνισής της με την ταινία-στόχο.

Στο δεύτερο βήμα, πρέπει οι συν-εμφανίσεις να κανονικοποιηθούν. Αυτό γίνεται καλώντας την μέθοδο normalizeCoOccurrences(), που βρίσκεται επίσης στην κλάση RecommenderUtils, η οποία δέχεται ως όρισμα το Map με τις συν-εμφανίσεις και επιστρέφει ένα Map με τις κανονικοποιημένες συν-εμφανίσεις. Αρχικά, υπολογίζεται το

άθροισμα των συν-εμφανίσεων στην μεταβλητή totalFreqs, χρησιμοποιώντας ένα stream από τις τιμές του Map με τις συν-εμφανίσεις και κάνοντας αναγωγή με την συνάρτηση reduce(). Έπειτα επιστρέφεται ένα νέο Map το οποίο είναι αποτέλεσμα ενός stream του αρχικού Map, έχοντας τα ίδια κλειδιά και τιμή την αρχική τιμή διαιρεμένη με το άθροισμα των συν-εμφανίσεων:

```
public static Map<Movie, Double> normalizeCoOccurrences(Map<Movie,
Long> freqs) {

    Optional<Long> totalFreqs =
freqs.values().stream().parallel().reduce(Long::sum);

    return freqs.entrySet().stream().parallel()
            .collect(Collectors.toMap(e -> e.getKey(), e ->
e.getValue() / Double.valueOf(totalFreqs.get())));
}
```

Απόσπασμα Κώδικα 4.2: Κανονικοποίηση συν-εμφανίσεων

Στο τρίτο βήμα, οι ταινίες τις οποίες δεν έχει βαθμολογήσει ο χρήστης προστίθενται στο σύνολο των βαθμολογιών του, με τιμή τον μέσο όρο βαθμολογιών του χρήστη. Από την μεταβλητή allMovies, η οποία περιέχει όλες τις διαθέσιμες ταινίες των δεδομένων, δημιουργείται ένα stream το οποίο φιλτράρεται έτσι ώστε να περιέχει μόνο αυτές τις οποίες δεν έχει βαθμολογήσει ο χρήστης. Τέλος, αυτές οι ταινίες συλλέγονται σε ένα Map, χρησιμοποιώντας την συνάρτηση Collectors.toMap(), με κλειδί την κάθε ταινία και τιμή τον μέσο όρο βαθμολογιών του χρήστη (userAvg). Κάθε καταχώριση αυτού του Map προστίθεται στο Map με όλες τις βαθμολογίες του χρήστη:

```
// movies not rated by the user should have the user's avg rating
Map<Movie, Double> collect = allMovies.stream().parallel()
        .filter(movie -> !user.getRatings().containsKey(movie))
        .collect(Collectors.toMap(Function.identity(), value ->
userAvg));

// add missing movie ratings to the user
collect.forEach((m,v) -> user.getRatings().put(m, v));
```

Απόσπασμα Κώδικα 4.3: Συμπλήρωση βαθμολογιών με τον μέσο όρο του χρήστη

Το τελευταίο βήμα είναι ο υπολογισμός της προβλεπόμενης βαθμολογίας, πολλαπλασιάζοντας τις βαθμολογίες του χρήστη με το διάνυσμα των συν-εμφανίσεων. Και σε αυτό το σημείο πραγματοποιείται μία λειτουργία απεικόνισης και αναγωγής. Δημιουργείται ένα stream από τις καταχωρίσεις του Map των βαθμολογιών του χρήστη και σε κάθε μία από αυτές εφαρμόζεται μία συνάρτηση η οποία επιστρέφει την βαθμολογία πολλαπλασιασμένη με την αντίστοιχη συν-εμφάνιση. Τέλος, αυτό το stream

ανάγεται στο τελικό άθροισμα με την κλήση της συνάρτηση reduce και το αποτέλεσμα είναι η προβλεπόμενη βαθμολογία του χρήστη για την μία συγκεκριμένη ταινία.

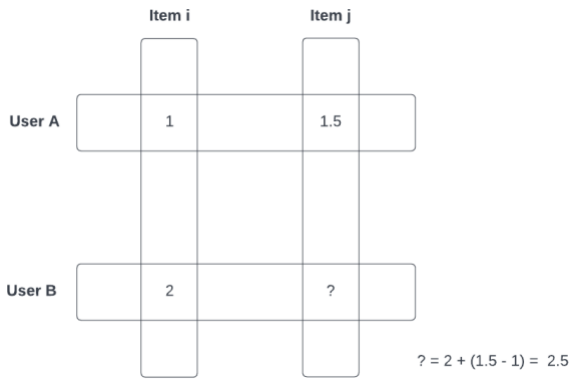
```
Optional<Double> prediction = userRatings.entrySet().stream()
        .map(entry -> {
            Double original = entry.getValue();
            Double co = coOccurrences.get(entry.getKey());
            if(co == null) co = 0.0;
            return original * co;
        }).reduce(Double::sum);
```

Απόσπασμα Κώδικα 4.4: Πολλαπλασιασμός συν-εμφανίσεων με βαθμολογίες χρήστη

4.3.3 Αλγόριθμος Slope One

Η οικογένεια αλγορίθμων Slope One, ανήκει και αυτή στο συνεργατικό φιλτράρισμα βασισμένο στα αντικείμενα. Χρησιμοποιούνται ευρύτατα, τόσο λόγω της ευκολίας στην υλοποίηση όσο και της ακριβείας των συστάσεων που παράγουν. Ακόμα, χρησιμοποιούνται σε συνδυασμό με άλλους αλγορίθμους προκειμένου να βελτιώσουν τις παραγόμενες συστάσεις.

Ο αλγόριθμος λαμβάνει υπόψιν τόσο πληροφορίες από άλλους χρήστες που βαθμολόγησαν το ίδιο αντικείμενο, όσο και πληροφορίες από τα αντικείμενα που βαθμολόγησε ο ίδιος χρήστης [34]. Βασική λειτουργία του αλγορίθμου είναι να ορισθούν ανά ζεύγη τα αντικείμενα και η μεταξύ τους σχέση όσον αφορά το πόσο αρεστό είναι το ένα σε σχέση με το άλλο. Αυτό γίνεται, όπως φαίνεται στο παρακάτω διάγραμμα, αφαιρώντας την μέση εκτίμηση δύο αντικειμένων, η οποία χρησιμοποιείται στον υπολογισμό της πρόβλεψης του ενός αντικειμένου για έναν άλλον χρήστη [34].



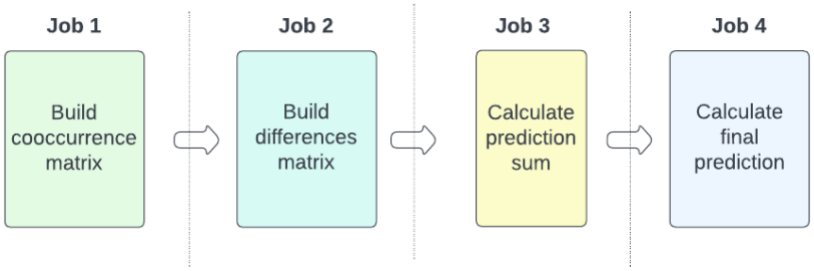
Διάγραμμα 12: Βάση αλγορίθμου Slope One

Ένα βασικό μειονέκτημα αυτής της απλής μορφής του αλγορίθμου είναι ότι δεν λαμβάνει υπόψιν το σύνολο των παρατηρήσεων για κάθε αντικείμενο. Ως απάντηση σε αυτό το πρόβλημα αναπτύχθηκε ο σταθμισμένος Slope One (weighted Slope One), ο

οποίος χρησιμοποιεί και τις συν-εμφανίσεις. Παρακάτω, περιγράφονται αναλυτικά τα βήματα αυτής της εκδοχής του αλγορίθμου η οποία υλοποιήθηκε στην παρούσα εργασία.

4.3.3.1 Περιγραφή του αλγορίθμου Weighted Slope One

Βασικό βήμα και του αλγορίθμου Slope One είναι ο υπολογισμός των συν-εμφανίσεων, ο οποίος παρουσιάστηκε αναλυτικά σε προηγούμενο κεφάλαιο. Με ανάλογο τρόπο, υπολογίζονται ανά ζεύγη και οι διαφορές μεταξύ των αντικειμένων. Τα δεδομένα του πίνακα συν-εμφανίσεων, του πίνακα διαφορών και των αρχικών βαθμολογιών χρησιμοποιούνται για τον υπολογισμό της τελικής προβλεπόμενης βαθμολογίας.



Διάγραμμα 13: Βήματα αλγορίθμου Slope One

Αναλυτικά τα βήματα είναι τα εξής [27] [35]:

1. Υπολογισμός πίνακα συν-εμφανίσεων
2. Υπολογισμός πίνακα διαφορών: κάθε τιμή στον πίνακα των διαφορών, υπολογίζεται αθροίζοντας τις διαφορές στις βαθμολογίες ανά ζεύγη αντικειμένων και διαιρώντας το άθροισμα με την αντίστοιχη τιμή συν-εμφάνισης:

$$diff(i,j) = \sum_{i \in S_{j,i}(x)} \frac{u_j - u_i}{freqS_{j,i}(x)}$$

Όπου, $S_{j,i}(x)$, το σύνολο των εκτιμήσεων ενός χρήστη και $freqS_{j,i}(x)$, η αντίστοιχη συν-εμφάνιση των αντικειμένων i και j. Λαμβάνοντας ως παράδειγμα τις βαθμολογίες του προηγούμενου κεφαλαίου (βλ. Πίνακας 2) και τις συν-εμφανίσεις που υπολογίστηκαν γι' αυτές (βλ. Πίνακας 3), προκύπτει ο παρακάτω πίνακας διαφορών:

	Inception	**Pretty Woman**	**Pulp Fiction**	**Django**	**The Power of the Dog**	**12 Years a Slave**
Inception	0	2	0	0	1	1

Pretty Woman	2	0	0	0	-1	-3
Pulp Fiction	0	0	0	1	2	3
Django	0	0	1	0	1	-3
The Power of the Dog	1	-1	2	1	0	0
12 Years a Slave	1	-3	3	-3	0	0

Πίνακας 6: Παράδειγμα πίνακα διαφορών

3. Υπολογισμός μη-σταθμισμένου αθροίσματος πρόβλεψης: σε αυτό το βήμα, το μη σταθμισμένο άθροισμα υπολογίζεται βάσει των διανύσματος των βαθμολογιών του χρήστη, του διανύσματος των διαφορών και αυτού των συν-εμφανίσεων:

$$sum = \sum_{i \in I} (u_i + diff_i) \times freq_i$$

4. Υπολογισμός σταθμισμένης πρόβλεψης: στο τελευταίο βήμα, το άθροισμα του προηγούμενου βήματος διαιρείται με το άθροισμα των συν-εμφανίσεων (λαμβάνονται υπόψιν μόνο οι συν-εμφανίσεις αντικειμένων των οποίων έχει βαθμολογήσει ο χρήστης). Οι τελικές βαθμολογίες που προκύπτουν για το παραπάνω παράδειγμα είναι οι παρακάτω:

	Inception	**Pretty Woman**	**Pulp Fiction**	**Django**	**The power of the dog**	**12 Years a slave**
Sara	4	2	9/2 = 4.5	3.0	3	4.0
Thomas	4.5	3.0	5	4	3	4.5
Joanna	3	1	5.0	1.0	2.0	4
Mary	5.0	2	5.0	2	2.0	5
Bill	4	0	4	0.5	2.5	1

Πίνακας 7: Προτεινόμενες βαθμολογίες με Slope One

4.3.3.2 Υλοποίηση με Java streams

Το πρώτο βήμα και αυτού του αλγορίθμου είναι ο υπολογισμός των συν-εμφανίσεων. Η μέθοδος που χρησιμοποιείται είναι και εδώ η computeCoOccurrences() της κλάσης RecommenderUtils. Οι συν-εμφανίσεις επιστρέφονται στην μορφή ενός

Map<Movie, Long>, ο οποίος έχει ως κλειδιά κάθε μία από τις υπόλοιπες ταινίες του συνόλου των ταινιών και ως τιμές την συν-εμφάνιση κάθε ταινίας με την ταινία-στόχο.

Στην συνέχεια υπολογίζονται οι διαφορές μεταξύ της ταινίας-στόχου και των υπόλοιπων ταινιών. Γι' αυτό το βήμα υλοποιούνται οι παρακάτω τρεις διαδικασίες:

```
// contains the differences
List<Map<Movie, Double>> diffs = userRatings.keySet().stream()
        .parallel()
        .map(rating -> users.stream().filter(u ->
                        !u.equals(user) &&
u.getRatings().containsKey(rating)
                                    &&
u.getRatings().containsKey(movie)).collect(Collectors.toList())
                .stream().map(user -> {
                    double differ = user.getRatings().get(movie) -
user.getRatings().get(rating);
                    return new Combo(differ, rating);
                })
                .collect(Collectors.groupingBy(Combo::getMovie,
Collectors.summingDouble(Combo::getDiff))))
        .collect(Collectors.toList());

// the differences list flattened
Map<Movie, Double> diffsFlat = diffs.parallelStream().flatMap(map ->
map.entrySet().stream())
        .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue));

// the list of final differences
diffsFinal = diffsFlat.entrySet().stream().parallel()
        .map(d -> {
            Long freq = freqsFlat.get(d.getKey());
            return new Combo(d.getValue() / freq, freq, d.getKey());
        }).collect(Collectors.toList());
```

Απόσπασμα Κώδικα 4.5: Υπολογισμός διανύσματος διαφορών

Παρόμοια με τον τρόπο υπολογισμού των συν-εμφανίσεων, ξεκινάμε με ένα stream, που παραλληλοποιείται, από τις ταινίες που έχει βαθμολογήσει ο χρήστης, το σύνολο των κλειδιών δηλαδή των βαθμολογιών του. Για κάθε μία από αυτές τις ταινίες πραγματοποιείται μία λειτουργία απεικόνισης, όπου:

α) δημιουργείται ένα stream από όλους τους υπάρχοντες χρήστες, το οποίο φιλτράρεται έτσι ώστε να μείνουν μόνο αυτοί που έχουν βαθμολογήσει την ταινία,

β) οι χρήστες αυτοί συλλέγονται σε μία λίστα, από την οποία δημιουργείται εκ νέου ένα stream, για το οποίο

γ) για κάθε χρήστη υπολογίζεται η διαφορά στην βαθμολογία της ταινίας αυτού του χρήστη με τον χρήστη για τον οποίο θέλουμε να προβλέψουμε την βαθμολογία.

Για την συλλογή των παραπάνω, έχει δημιουργηθεί η βοηθητική κλάση Combo, στην οποία αποθηκεύεται η ταινία και η τιμής της διαφοράς. Οι διαφορές συλλέγονται ως

μία λίστα αντικειμένων Combo, τα οποία στο τέλος ομαδοποιούνται σε ένα Map, με την χρήση της συνάρτησης groupingBy(), το οποίο έχει ως κλειδί την ταινία και ως τιμή το άθροισμα των διαφορών, χρησιμοποιώντας ως δεύτερο όρισμα στην groupingBy() την συνάρτηση Collectors.summingDouble(). Τέλος, τα παραπάνω Map, συλλέγονται σε μία λίστα, η οποία στο επόμενο βήμα εξομαλύνεται σε ένα επίπεδο, καλώντας την συνάρτηση flatMap() επάνω στο stream που προκύπτει από την λίστα του προηγούμενου βήματος.

Στην τελευταία ακολουθία εντολών, από τις καταχωρίσεις της εξομαλυμένης λίστας με τις διαφορές, δημιουργείται ένα stream, έτσι ώστε να παραχθεί μία λίστα με αντικείμενα της κλάση Combo, τα οποία περιλαμβάνουν συγκεντρωτικά την ταινία, την διαφορά αλλά και την τιμή της συν-εμφάνισης.

Στην συνέχεια πρέπει να υπολογιστεί το μη σταθμισμένο άθροισμα της πρόβλεψης. Αυτό γίνεται χρησιμοποιώντας την λίστα των Combo του προηγούμενου βήματος, για κάθε ένα από τα οποία πραγματοποιείται μία λειτουργία απεικόνισης. Στην απεικόνιση λαμβάνεται από κάθε αντικείμενο Combo η τιμή της διαφοράς στην οποία προστίθεται η τιμή της αρχικής βαθμολογίας του χρήστη. Το άθροισμα πολλαπλασιάζεται με την τιμή της συν-εμφάνισης και οι τιμές που προκύπτουν ανάγονται σε ένα τελικό άθροισμα, καλώντας την συνάρτηση reduce():

```
// calculate the prediction sum
Optional<Double> pred = diffsFinal.stream().parallel().map(combo -> {
    double orig = userRatings.get(combo.getMovie());
    Long freq = combo.getFreq();
    Double differ = combo.getDiff();
    return (differ + orig) * freq;
}).reduce(Double::sum);
```

Απόσπασμα Κώδικα 4.6: Υπολογισμός μη σταθμισμένου αθροίσματος

Τέλος, η τελική προβλεπόμενη βαθμολογία προκύπτει ως το πηλίκο του μη σταθμισμένου αθροίσματος προς το άθροισμα των συν-εμφανίσεων. Το άθροισμα των συν-εμφανίσεων υπολογίζεται απλά ξεκινώντας με ένα παράλληλο stream από τις συν-εμφανίσεις, στο οποίο πραγματοποιείται μία απλή λειτουργία αναγωγής σε άθροισμα:

```
// sum the frequencies
Optional<Long> totalFreqs =
freqsFlat.values().stream().parallel().reduce(Long::sum);

// final prediction
Double prediction = pred.get()/totalFreqs.get();
```

Απόσπασμα Κώδικα 4.7: Υπολογισμός τελικής βαθμολογίας με Slope One και streams

4.3.4 Αλγόριθμος K-Nearest-Neighbors

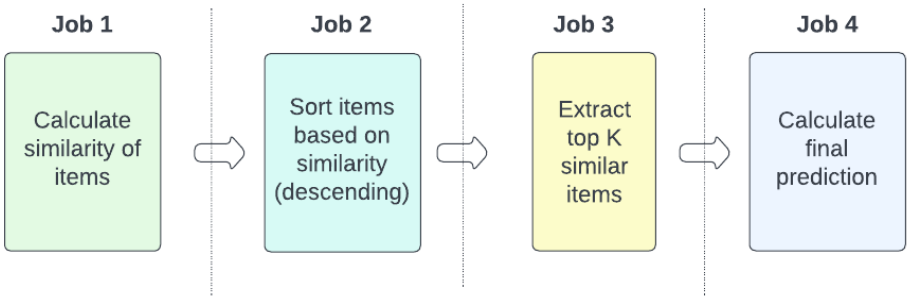
Οι αλγόριθμοι που βασίζονται στην εγγύτητα ή την γειτνίαση, αποτέλεσαν αρχικά μέθοδο της στατιστικής επιστήμης για την κατηγοριοποίηση και παλινδρόμηση [36]. Στην συνέχεια επεκτάθηκαν και σε διάφορους άλλους τομείς, ένας από αυτούς είναι και τα συστήματα συστάσεων. Η ιδέα της γειτνίασης των αντικειμένων (ή χρηστών), αναφέρεται στον υπολογισμό της εγγύτητάς τους με βάση την μεταξύ τους ομοιότητα και βασίζεται στην λογική ότι σε παρόμοιους χρήστες θα αρέσουν τα ίδια αντικείμενα, ή αντιστοίχως σε έναν χρήστη θα αρέσουν παρόμοια μεταξύ τους αντικείμενα. Και αυτή η κατηγορίας αλγορίθμων ανήκουν στο συνεργατικό φιλτράρισμα. Υπάρχουν διάφοροι μέθοδοι για τον υπολογισμό της ομοιότητας, με τις πιο συνηθισμένες να είναι η ομοιότητα Jaccard, η ομοιότητα συνημίτονου και ο συντελεστής συσχέτισης Pearson [18].

Οι αλγόριθμοι μπορεί να είναι βασισμένοι στα αντικείμενα ή βασισμένοι στους χρήστες, και η προσέγγιση και στις δύο εκδοχές είναι πολύ παρόμοια. Η γενική προσέγγιση αυτών των αλγορίθμων συνίσταται σε δύο βήματα:

1. Εύρεση του περίγυρου γειτνίασης του αντικειμένου (ή χρήστη) προς βαθμολόγηση
2. Υπολογισμός της βαθμολογίας από τις βαθμολογίες παρόμοιων αντικειμένων (ή χρηστών)

Στην περίπτωση υπολογισμού βάσει αντικειμένων ο αλγόριθμος ξεκινάει εντοπίζοντας τα παρόμοια αντικείμενα με αυτό του οποίου η βαθμολογία αναζητείται, ενώ στην περίπτωση υπολογισμού βάσει χρηστών εντοπίζονται οι παρόμοιοι χρήστες με αυτόν για τον οποίο αναζητείται η βαθμολογία για ένα συγκεκριμένο αντικείμενο.

4.3.4.1 Περιγραφή του αλγορίθμου



Διάγραμμα 14: Βήματα αλγορίθμου K-NN

Στο πλαίσιο της εργασίας επιλέχθηκε να υλοποιηθεί αλγόριθμος k-NN βάσει αντικειμένων αν και τα βήματα του αντίστοιχου αλγορίθμου βάσει χρηστών είναι πολύ παρόμοια.

Όπως αναφέρθηκε, πρόκειται για έναν αλγόριθμο συνεργατικού φιλτραρίσματος, το οποίο σημαίνει ότι το μόνο δεδομένο που είναι απαραίτητο ως είσοδος για την παραγωγή των προτάσεων είναι ο πίνακας βαθμολογιών. Ο αλγόριθμος λειτουργία πάνω στην υπόθεση, πως οι χρήστες βαθμολογούν με παρόμοιο τρόπο, και άρα προτιμούν, παρόμοια αντικείμενα. Ως «γειτονιά» εννοούνται ένα σύνολο αντικειμένων τα οποία έχουν παρόμοιο βαθμολογικό μοτίβο και συνεπώς τα διανύσματα των βαθμολογιών τους είναι «κοντά». Γενικότερα μπορούμε να θεωρήσουμε ότι όσο μεγαλύτερη η απόσταση μεταξύ δύο αντικειμένων τόσο μικρότερη η ομοιότητά τους, και αντίστροφα.

Τα βήματα του αλγορίθμου k-NN προκειμένου να υπολογιστεί η προβλεπόμενη βαθμολογία ενός αντικειμένου για έναν χρήστη είναι τα παρακάτω [18] [37]:

1. Υπολογισμός ομοιότητας: σε πρώτη φάση πρέπει να υπολογιστεί η ομοιότητα του αντικειμένου με όλα τα υπόλοιπα διαθέσιμα αντικείμενα έτσι ώστε να οριστεί η «γειτονιά». Για τον υπολογισμό της ομοιότητας υπάρχουν διάφορες συναρτήσεις όπως η ομοιότητα Jaccard, ο συντελεστής συσχέτισης Pearson και η ομοιότητα συνημιτόνου. Η επιλογή της συνάρτησης ομοιότητας γίνεται βάσει της μορφής των δεδομένων των βαθμολογιών. Για παράδειγμα, αν οι βαθμολογίες είναι σε μορφή μοναδιαίας ή δυαδικής προτίμησης (π.χ. στον χρήστη Α αρέσει η ταινία Β, στον χρήστη Α δεν αρέσει η ταινία Γ), επιλέγεται η ομοιότητα Jaccard, ενώ αν τα δεδομένα των βαθμολογιών είναι αριθμητικές τιμές (π.χ. ο χρήστης Α βαθμολόγησε την ταινία Β με 5.0), προτιμάται η ομοιότητα συνημιτόνου ή ο συντελεστής συσχέτισης Pearson [18]. Ακριβώς επειδή τα δεδομένα που χρησιμοποιήθηκαν στην εργασία είναι της δεύτερης μορφής, επιλέχθηκε ο υπολογισμός της ομοιότητας με βάση το συνημίτονο και η αντίστοιχη μέθοδος περιγράφεται παρακάτω:

Ομοιότητα συνημιτόνου:

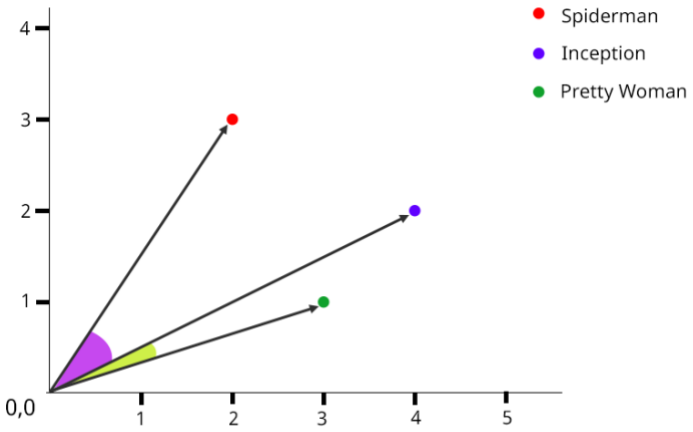
Ο υπολογισμός της ομοιότητας με τον τύπο του συνημιτόνου, χρησιμοποιεί τα διανύσματα των βαθμολογιών των αντικειμένων (ή των χρηστών) και αναπαριστώντας τα στον χώρο, υπολογίζει την μεταξύ τους γωνία για να εντοπίσει πόσο «κοντά» ή «μακριά

είναι [18]. Στο παρακάτω παράδειγμα φαίνεται ένα στιγμιότυπο του πίνακα βαθμολογιών για τρεις μόνο ταινίες:

	Sara	**Joanna**
Inception	4	2
Spiderman	2	3
Pretty Woman	3	1

Πίνακας 8: Απόσπασμα πίνακα βαθμολογιών

Τα διανύσματα στον δισδιάστατο χώρο γι' αυτές τις ταινίες φαίνονται στο επόμενο διάγραμμα:



Διάγραμμα 15: Διανύσματα βαθμολογιών στον χώρο [18]

Όπως είναι φανερό, η γωνία που σχηματίζεται μεταξύ των ταινιών "Spiderman" και "Inception" είναι μεγαλύτερη σε σχέση με την γωνία "Pretty Woman" – "Inception", και έτσι μπορούμε να πούμε ότι η ταινία "Inception" είναι πιο παρόμοια με το "Pretty woman" παρά με το "Spiderman".

Ο τύπος για τον υπολογισμό της ομοιότητας δύο αντικειμένων, έστω i,j, είναι ο παρακάτω [18]:

$$sim(i,j) = \frac{\sum_u r_{i,u} r_{j,u}}{\sqrt{\sum_u r_{i,u}^2} \sqrt{\sum_u r_{j,u}^2}}$$

Αυτός ο τύπος παρ' όλα αυτά δεν είναι σταθμισμένος με βάση την μέση βαθμολογία του χρήστη. Γι' αυτό χρησιμοποιείται ο σταθμισμένος τύπος, ο οποίος διαμορφώνεται ως εξής [38]:

$$sim(i,j) = \frac{\sum_u (r_{u,i} - \overline{r_u})(r_{u,j} - \overline{r_u})}{\sqrt{\sum_u (r_{u,i} - \overline{r_u})^2}\sqrt{\sum_u (r_{u,j} - \overline{r_u})^2}}$$

2. Ανάκτηση των k πιο όμοιων αντικειμένων: αφού υπολογιστεί η ομοιότητα μεταξύ του αντικειμένου-στόχου και όλων των υπόλοιπων αντικειμένων στο σύνολο των δεδομένων, τα ζεύγη αντικείμενο-ομοιότητα πρέπει να ταξινομηθούν σε φθίνουσα σειρά προκειμένου να ανακτηθούν τα k πιο όμοια αντικείμενα.
3. Υπολογισμός τελικής βαθμολογίας: η τελική βαθμολογία προκύπτει λαμβάνοντας υπόψιν το σύνολο των k πιο όμοιων αντικειμένων, την τιμή της ομοιότητας και την αρχική βαθμολογία του χρήστη για κάθε αντικείμενο. Για έναν χρήστη x και ένα αντικείμενο i, η προβλεπόμενη βαθμολογία δίνεται από τον τύπο:

$$r_{x,i} = \sum_{j \in J} s_{i,j} r_{x,i}$$

όπου, $s_{i,j}$ η υπολογισθείσα τιμή της ομοιότητας μεταξύ των αντικειμένων i,j και $J \subseteq I$, το σύνολο των k πιο όμοιων αντικειμένων με το αντικείμενο i. Για καλύτερη ακρίβεια, μπορεί να χρησιμοποιηθεί ο σταθμισμένος τύπος που είναι ο παρακάτω:

$$r_{x,i} = \frac{\sum_{j \in J} s_{i,j} r_{x,i}}{\sum_{j \in J} s_{i,j}}$$

4.3.4.2 Υλοποίηση με Java streams

Παρακάτω περιγράφεται η υλοποίηση του αλγορίθμου K-NN με Java Streams, δοθείσης μίας ταινίας, έστω movie, για την οποία αναζητείται μία πρόβλεψη βαθμολογίας για έναν χρήστη, έστω user. Σε όλα τα σημεία στα οποία η παραλληλοποίηση δεν δημιουργεί αλλαγές στις τιμές των αποτελεσμάτων, χρησιμοποιήθηκαν οι μέθοδοι parallel και parallelStream για την παράλληλη επεξεργασία.

Το πρώτο βήμα είναι ο υπολογισμός των τιμών ομοιότητας μεταξύ της ταινίας movie και όλων των υπόλοιπων ταινιών στο σύνολο των δεδομένων. Απαραίτητη είναι πρώτα μία προ-επεξεργασία, κατά την οποία στο διάνυσμα βαθμολογιών του κάθε χρήστη προστίθενται και οι ταινίες τις οποίες δεν έχει βαθμολογήσει, με τιμή βαθμολογίας 0. Επίσης, πρέπει να ανακτηθεί το διάνυσμα όλων των βαθμολογιών της ταινίας movie. Στην

παρακάτω υλοποίηση αυτό είναι στην μεταβλητή movieRatings, ένα αντικείμενο τύπου Map<User, Double>, με κλειδί τον χρήστη που βαθμολόγησε την ταινία movie και τιμή την αντίστοιχη βαθμολογία. Στην συνέχεια δημιουργείται η μεταβλητή similarMovies, που είναι ένα αντικείμενο τύπου Map<Movie,Double> και περιέχει ως κλειδιά τις ταινίες και ως τιμές την τιμή της ομοιότητας την ταινίας-στόχου με κάθε ταινία-κλειδί:

```
similarMovies = allMovies.stream().parallel().map(m ->
                new Pair(m,
ItemCosineSimilarity.itemsSimilarity(movieRatings,
RecommenderUtils.collectMovieRatings(allUsers, m))))
        .collect(Collectors.toMap(t -> (Movie) t.getKey(), t ->
(Double) t.getRating()));
```

Απόσπασμα Κώδικα 4.8: Υπολογισμός τιμών ομοιότητας

Στο παραπάνω απόσπασμα κώδικα, ξεκινώντας από την μεταβλητή allMovies, που είναι μία λίστα με όλες τις διαθέσιμες ταινίες του συνόλου, δημιουργείται ένα stream και για κάθε ταινία αυτού του stream πραγματοποιείται μία λειτουργία απεικόνισης. Στην απεικόνιση, από κάθε ταινία που έρχεται ως είσοδος, δημιουργείται ένα αντικείμενο τύπου Pair (βοηθητική κλάση που δημιουργήθηκε στα πλαίσια της εργασίας), το οποίο έχει δύο μεταβλητές στιγμιοτύπου α) την ταινία και β) την τιμή της ομοιότητας. Τέλος, τα αντικείμενα τύπου Pair που δημιουργήθηκαν συλλέγονται στο τελικό Map, με κλειδί την ταινία και τιμή την τιμή της ομοιότητας, χρησιμοποιώντας την συνάρτηση Collectors.toMap.

Για τον υπολογισμό της ομοιότητας με βάση τον τύπο του συνημιτόνου, έχει υλοποιηθεί η στατική μέθοδος itemsSimilarity στην κλάση ItemCosineSimilarity, η οποία δέχεται ως ορίσματα τα διανύσματα βαθμολογίων για δύο ταινίες (ratings1 & ratings2), στην μορφή Map<User,Double>, όπου κλειδί είναι οι χρήστες και τιμές οι αντίστοιχες βαθμολογίες για την ταινία:

```
Double sum = ratings1.entrySet().stream().parallel()
        .map(e -> ratings2.get(e.getKey()) != null ? e.getValue() *
ratings2.get(e.getKey()) : 0.0)
        .reduce(Double::sum)
        .get();

Double r1 = Math.sqrt(ratings1.values().stream().parallel()
        .map(v -> v*v)
        .reduce(Double::sum).get());

Double r2 = Math.sqrt(ratings2.values().stream()
        .map(v -> v * v)
        .reduce(Double::sum).get());

return sum / (r1*r2);
```

Απόσπασμα Κώδικα 4.9: Υπολογισμός ομοιότητας συνημιτόνου

Ξεκινώντας από το ένα από τα δύο διανύσματα, έστω το ratings1, δημιουργείται επί αυτού ένα stream από τις καταχωρίσεις του (ratings1.entrySet()). Για κάθε καταχώριση πραγματοποιείται μία λειτουργία απεικόνισης, η οποία επιστρέφει την βαθμολογία της ταινίας 1 από τον χρήστη u πολλαπλασιασμένη με την βαθμολογία της ταινίας 2 από τον ίδιο χρήστη. Αυτές οι τιμές αθροίζονται με την κλήση της συνάρτησης reduce και το τελικό αποτέλεσμα αποθηκεύεται στην μεταβλητή sum.

Στην συνέχεια, για κάθε ένα από τα δύο διανύσματα βαθμολογιών, υπολογίζεται η ρίζα του αθροίσματος του τετραγώνου τους. Δημιουργείται ένα stream με τις τιμές του κάθε Map βαθμολογιών (ratings1.values().stream()), όπου για κάθε τιμή πραγματοποιείται μία λειτουργία απεικόνισης και μίας αναγωγής. Στην απεικόνιση επιστρέφεται το τετράγωνο της κάθε βαθμολογίας και στην αναγωγή αθροίζεται το σύνολο των τετραγώνων. Η τελική ομοιότητα υπολογίζεται ως το πηλίκο του πρώτου αθροίσματος προς το γινόμενο της ρίζας του αθροίσματος των τετραγώνων των δύο διανυσμάτων βαθμολογιών.

Το επόμενο βήμα είναι η ταξινόμηση του διανύσματος με τις ομοιότητες σε φθίνουσα σειρά. Χρησιμοποιώντας το αντικείμενο similarMovies που δημιουργήθηκε στο προηγούμενο βήμα, δημιουργείται ένα stream με τις καταχωρίσεις του :

```
Map<Movie, Double> similarMoviesSorted = similarMovies.entrySet()
        .stream()
        .parallel()
        .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
        .collect(Collectors.toMap(
                Map.Entry::getKey,
                Map.Entry::getValue,
                (oldValue, newValue) -> oldValue,
LinkedHashMap::new));
```

Σε αυτό το stream, καλείται η συνάρτηση sorted, η οποία χρειάζεται ως όρισμα ένα αντικείμενο Comparator, έτσι ώστε να ταξινομήσει τα στοιχεία του stream. Στην προκειμένη περίπτωση, δίνεται ως παράμετρος το αντικείμενο που επιστρέφεται καλώντας την στατική συνάρτηση comparingByValue() της κλάση Map.Entry, για να ταξινομηθούν τα στοιχεία με βάση την τιμή (και όχι το κλειδί) και επιπλέον, σε αυτήν την συνάρτηση δίνεται ως παράμετρος η στατική συνάρτηση reverseOrder() της κλάσης Comparator, έτσι ώστε η ταξινόμηση να γίνει σε φθίνουσα σειρά. Τέλος, οι καταχωρίσεις που προκύπτουν από την κλήση της sorted(), συλλέγονται καλώντας την συνάρτηση collect() σε ένα αντικείμενο τύπου LinkedHashMap, το similarMoviesSorted, για να μπορεί να διατηρηθεί η σειρά των καταχωρίσεων.

Οι k πιο παρόμοιες ταινίες ανακτώνται χρησιμοποιώντας έναν βρόχο for, στην μεταβλητή topKSimilarMovies, η οποία είναι και αυτή τύπου LinkedHashMap<Movie,Double>. Με βάση αυτή, υπολογίζονται τα αθροίσματα του αριθμητή και του παρονομαστή που είναι απαραίτητα για τον υπολογισμό της τελικής πρόβλεψης:

```
Double numerator = topKSimilarMovies.entrySet().stream()
        .parallel()
        .map(e -> e.getValue()* user.getRatings().get(e.getKey()))
        .reduce(Double::sum)
        .get();

Double denominator =
topKSimilarMovies.values().stream().parallel().reduce(Double::sum).ge
t();

Double prediction = userAvgRating + (numerator/denominator);
```

Απόσπασμα Κώδικα 4.10: Υπολογισμός τελικής πρόβλεψης με knn

Ο αριθμητής προκύπτει πολλαπλασιάζοντας την τιμή της ομοιότητας κάθε ταινίας με την ταινία-στόχο με την βαθμολογία που έχει δώσει ο χρήστης σε αυτήν την ταινία. Γι' αυτό από τις k πιο όμοιες ταινίες δημιουργείται ένα stream από τις καταχωρίσεις του αντικειμένου και έπειτα πραγματοποιείται μία λειτουργία απεικόνισης και μίας αναγωγής. Στην απεικόνιση, κάθε τιμή πολλαπλασιάζεται με την αντίστοιχη βαθμολογία και στην αναγωγή συγκεντρώνονται τα γινόμενα σε ένα άθροισμα. Ο υπολογισμός του παρονομαστή είναι απλούστερος, καθώς προκύπτει από το άθροισμα των υπολογισθέντων ομοιοτήτων. Έτσι από την μεταβλητή topKSimilarMovies, δημιουργείται ένα stream, μόνο από τις τιμές του αυτή την φορά (values().stream()), και όχι από τις καταχωρίσεις, και τέλος καλείται η συνάρτηση reduce(), έτσι ώστε να αθροιστούν οι τιμές των ομοιοτήτων. Στην πρόβλεψη προστίθεται και ο μέσος όρος βαθμολογιών του χρήστη.

4.3.5 Φόρτωση δεδομένων με streams

Η ανάγνωση και εισαγωγή των δεδομένων στην εφαρμογή, αν και δεν αποτελεί μέρος των αλγορίθμων του συστήματος συστάσεων, περιγράφεται παρακάτω, μιας και είναι σημαντικό κομμάτι της εφαρμογής, υλοποιημένο με Java streams.

Τα δεδομένα που χρησιμοποιήθηκαν είναι σε μορφή .csv και περιλαμβάνουν μεταξύ άλλων, ένα μοναδικό αναγνωριστικό για κάθε χρήστη, ένα μοναδικό αναγνωριστικό για κάθε ταινία, τον τίτλο της ταινίας και την βαθμολογία που έδωσε ο χρήστης σε ένα σύνολο ταινιών. Δύο αρχεία χρησιμοποιήθηκαν, αυτό που περιλαμβάνει

τις πληροφορίες για τις ταινίες μεμονωμένα, και αυτό που περιλαμβάνει τις βαθμολογίες, στιγμιότυπα των οποίων φαίνονται στις παρακάτω εικόνες:

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children
9	Sudden Death (1995)	Action

Διάγραμμα 16: Στιγμιότυπου αρχείου movies.csv

userId	movieId	rating	timestamp
1	1	4.0	964982703
1	3	4.0	964981247
1	6	4.0	964982224
1	47	5.0	964983815
1	50	5.0	964982931
1	70	3.0	964982400
1	101	5.0	964980868
1	110	4.0	964982176

Διάγραμμα 17: Στιγμιότυπο αρχείου ratings.csv

Όπως φαίνεται στην εικόνα, κάθε ταινία σχετίζεται και με ένα σύνολο από είδη (genres). Τα συγκεκριμένα δεδομένα εισάγονται στο σύστημα μέσω streams, δεν αξιοποιούνται ωστόσο από τους αλγορίθμους συστάσεων, καθώς αποτελούν αντικείμενο αλγορίθμων βάσει περιεχόμενου.

Για το διάβασμα και την εισαγωγή των δεδομένων δημιουργήθηκε η κλάση MovieDataLoader, η οποία περιέχει δύο μεθόδους, μία για το διάβασμα όλων των ταινιών (loadMovies()) και μία για το διάβασμα των χρηστών με τις βαθμολογίες που έχουν δώσει σε κάθε ταινία (loadUserRatings()).

Η μέθοδος loadMovies δέχεται ως όρισμα ένα αντικείμενο τύπου Path, το οποίο αντιπροσωπεύει το μονοπάτι προς το αρχείο με τα δεδομένα των ταινιών, και επιστρέφει μία λίστα με αντικείμενα τύπου Movie. Στο επόμενο απόσπασμα κώδικα φαίνεται ο τρόπος δημιουργίας της λίστας των ταινιών:

```
try {
    movies = Files.lines(path)
            .skip(1) // skip the header line
            .map(new MovieMapper()) // transform each line to an
array
            .distinct()
            .collect(Collectors.toList());
} catch (IOException e) {
    movies = new ArrayList<>();
}
```

Απόσπασμα Κώδικα 4.11: Διάβασμα δεδομένων ταινιών

Η στατική μέθοδος lines(<path>) της κλάση Files, επιστρέφει ένα stream με αλφαριθμητικά, κάθε ένα από τα οποία είναι μία γραμμή του αρχείου που δέχεται ως όρισμα. Καλώντας την συνάρτηση skip(1), παραλείπεται η πρώτη γραμμή του αρχείου, καθώς πρόκειται για γραμμή επικεφαλίδας, χωρίς δεδομένα για τις ταινίες. Έπειτα για κάθε γραμμή του αρχείου πραγματοποιείται μια λειτουργία απεικόνισης, καλώντας την συνάρτηση map με όρισμα ένα αντικείμενο της κλάσης MovieMapper(), ο τρόπος λειτουργίας της οποίας θα παρουσιαστεί λίγο παρακάτω. Τέλος, καλείται η συνάρτηση distinct(), έτσι ώστε να αφαιρεθούν από την τελική λίστα πιθανά διπλότυπες εγγραφές, και η συνάρτηση collect για την συλλογή των ταινιών στην λίστα.

Η κλάση MovieMapper() δημιουργήθηκε για τους σκοπούς της εργασίας και υλοποιεί την διεπαφή Function, έτσι ώστε στιγμιότυπά της να μπορούν να δοθούν ως ορίσματα στην συνάρτηση map:

```
public class MovieMapper implements Function<String, Movie> {

    @Override
    public Movie apply(String line) {
        Movie m = new Movie();

        String[] tokens = line.split(",");
        m.setId(tokens[0]);
        m.setTitle(tokens[1]);

        String[] genres = tokens[2].split("\\|");
        for(String g : genres) {
            m.addGenre(g);
        }

        return m;
    }
}
```

Απόσπασμα Κώδικα 4.12: Κλάση MovieMapper

Υλοποιεί μόνο μία μέθοδο, την apply, την οποία κληρονομεί από την διεπαφή Function, και είναι αυτή που καλείται όταν ένα αντικείμενο τύπου MovieMapper δίνεται ως όρισμα στην συνάρτηση map. Η apply δέχεται ως παράμετρο ένα αλφαριθμητικό, που

αντιστοιχεί σε μία γραμμή του αρχείου με τα δεδομένα ταινιών. Στόχος είναι να διαβαστούν σωστά τα στοιχεία κάθε γραμμής και να δημιουργηθεί και να επιστραφεί ένα αντικείμενο τύπου Movie.

Η μέθοδος loadUserRatings(<path>) δέχεται και αυτή ως παράμετρο ένα αντικείμενο τύπου Path το οποίο αντιπροσωπεύει το μονοπάτι στο οποίο βρίσκεται το αρχείο με τα δεδομένα των βαθμολογιών και επιστρέφει μία λίστα με αντικείμενα τύπου User, κάθε ένα από τα οποία περιέχει ένα αντικείμενο τύπου Map<Movie,Double> με τις αντίστοιχες βαθμολογίες:

```
users = Files.lines(path)
                .parallel()
                .skip(1) // skip the header line
                .map(line -> line.split(",")) // transform each line
to an array
                .map(splitted -> {
                    User u = new User(splitted[0]);
                    Movie m = new Movie(splitted[1]);
                    Double rating = Double.parseDouble(splitted[2]);
                    u.addRating(m,rating);
                    return u;
                }) // transform each array to an entity
        .collect(Collectors.toList())
        .stream()
        .map(new UserMovieMapper())
        .distinct()
        .collect(Collectors.toList());
```

Απόσπασμα Κώδικα 4.13: Διάβασμα δεδομένων βαθμολογιών χρηστών

Αντίστοιχα με την μέθοδο loadMovies(), δημιουργείται ένα stream με τις γραμμές του αρχείου καλώντας την συνάρτηση Files.lines(), και παραλείπεται η πρώτη γραμμή που είναι επικεφαλίδα. Για κάθε γραμμή εφαρμόζονται δύο λειτουργείες απεικόνισης: η πρώτη αναλαμβάνει να διαχωρίσει την γραμμή με διαχωριστικό τον χαρακτήρα «,» (κόμμα), επιστρέφοντας έναν πίνακα με τα στοιχεία της κάθε γραμμής, και η δεύτερη λαμβάνει ως είσοδο αυτόν τον πίνακα και δημιουργεί τα αντικείμενα τύπου User. Τα αντικείμενα τύπου User που δημιουργήθηκαν συλλέγονται σε μία λίστα, που όμως περιέχει πολλές φορές τον κάθε χρήστη, με μία μόνο βαθμολογία σε κάθε καταχώρισή του. Για τον λόγο αυτόν, δημιουργείται εκ νέου ένα stream από την τελευταία λίστα, έτσι ώστε να συγκεντρωθούν οι βαθμολογίες των χρηστών σε μία λίστα όπου ο κάθε χρήστης περιέχεται μία μόνο φορά.

Για τον σκοπό αυτόν, δημιουργήθηκε η κλάση UserMovieMapper(), η οποία υλοποιεί και αυτή την διεπαφή Function:

```
public class UserMovieMapper implements Function<User, User> {

    List<User> users = new ArrayList<>();

    @Override
    public User apply(User user) {
        if(users.contains(user)) {
            User old = users.get(users.indexOf(user));
            old.getRatings().putAll(user.getRatings());
            return old;
        } else {
            users.add(user);
            return user;
        }
    }
}
```

Απόσπασμα Κώδικα 4.14: Κλάση UserMovieMapper

Η κλάση έχει μία μεταβλητή στιγμιοτύπου, την λίστα users, έτσι ώστε να γνωρίζει ανά πάσα στιγμή ποιος χρήστης υπάρχει ήδη στην λίστα και ποιος όχι. Υλοποιεί μόνο την μέθοδο apply η οποία δέχεται ως παράμετρο ένα αντικείμενο τύπου User, και ελέγχει αν ο συγκεκριμένος χρήστης υπάρχει ήδη στην λίστα. Εάν υπάρχει, προσθέτει την βαθμολογία του χρήστη-παραμέτρου, στις βαθμολογίες του χρήστη που υπάρχει ήδη στην λίστα. Σε αντίθετη περίπτωση, προσθέτει τον νέο χρήστη στην λίστα.

Πίσω στην συνάρτηση loadUserRatings, αφού κληθεί και η συνάρτηση map με όρισμα ένα αντικείμενο της κλάσης που περιγράφηκε, η τελική λίστα των χρηστών με όλες τις βαθμολογίες τους, συγκεντρώνεται καλώντας την συνάρτηση collect σε μία λίστα.

4.4 Χρόνοι εκτέλεσης παράλληλων και ακολουθιακών αλγορίθμων

Οι αλγόριθμοι που υλοποιήθηκαν για τα συστήματα συστάσεων παρουσιάζονται στην παράλληλη εκδοχή τους, όπου για τα περισσότερα streams καλείται η μέθοδος parallel ή parallelStream, έτσι ώστε να δημιουργηθούν αυτόματα τα νήματα που αναλαμβάνουν τον υπολογισμό. Εκτός από την απλότητα με την οποία παραλληλίζονται οι αλγόριθμοι με το stream API, καλώντας ουσιαστικά μία μόνο μέθοδο πάνω σε κάθε stream, το κέρδος είναι φυσικά εμφανές και στους χρόνους εκτέλεσης.

Προκειμένου να μετρηθούν και να συγκριθούν οι χρόνοι εκτέλεσης των δύο εκδοχών των αλγορίθμων, δημιουργήθηκαν δοκιμαστικές κλάσεις, στις οποίες τρέχουν τόσο η ακολουθιακή όσο και η παράλληλη εκδοχή σε ένα σύνολο προβλέψεων, και μετριέται ο μέσος όρος του χρόνου που απαιτείται για τον υπολογισμό κάθε προβλεπόμενης τιμής. Λόγω του κόστους δημιουργίας και διαχείρισης των νημάτων, το κέρδος του παραλληλισμού δεν είναι εμφανές για μία μόνο πρόβλεψη. Γι’ αυτό, οι δοκιμές

αφορούν 30, 50 και 100 προβλέψεις. Ο χρόνος μετριέται σε milliseconds και αφορά τον μέσο όρο του χρόνο εκτέλεσης κάθε αλγορίθμου για μία πρόβλεψη. Τα αποτελέσματα φαίνονται στους παρακάτω πίνακες:

	30	**50**	**100**
Ακολουθιακός	19.233	17.58	18.21
Παράλληλος	7.633	5.9	5.9

Πίνακας 9: Αλγόριθμος απεικόνισης-αναγωγής

	30	**50**	**100**
Ακολουθιακός	27.566	25.26	25.67
Παράλληλος	7.8	6.56	6.41

Πίνακας 10: Αλγόριθμος Slope One

	30	**50**	**100**
Ακολουθιακός	2680.133	2740.76	2650.27
Παράλληλος	1403.466	1549.86	1510.9

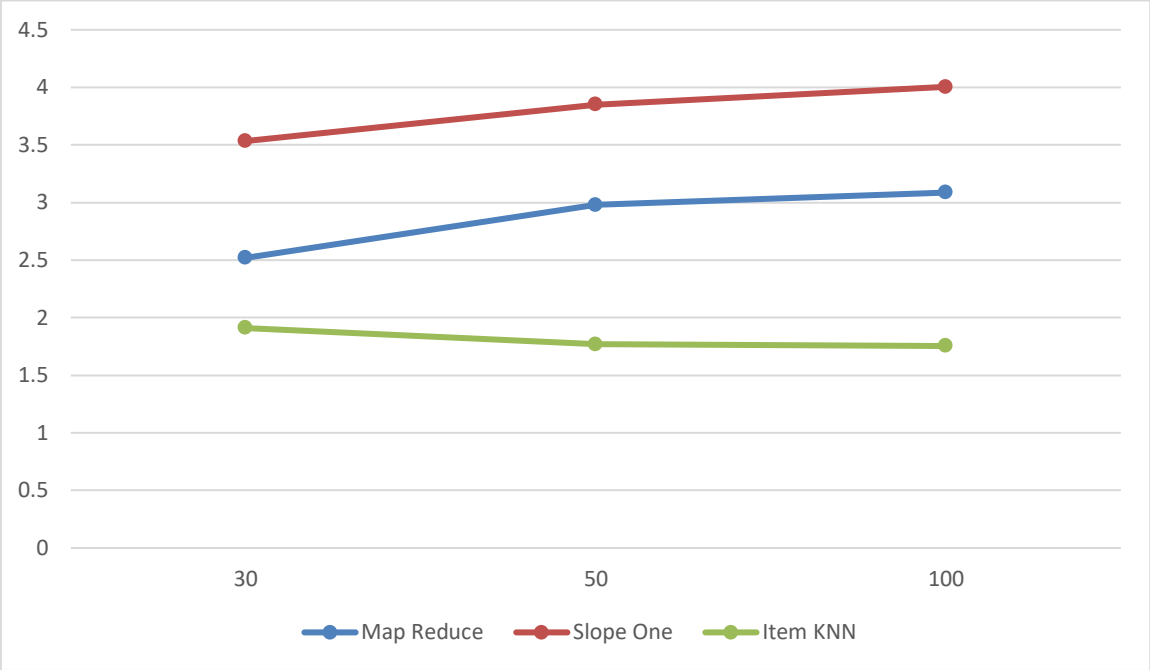
Πίνακας 11: Αλγόριθμος KNN

Από τα αποτελέσματα, μπορούν να συγκριθούν τόσο οι εκδοχές του κάθε αλγορίθμου, όσο και οι αλγόριθμοι μεταξύ τους. Είναι φανερό πως ο αλγόριθμος απεικόνισης και αναγωγής υπερτερεί των άλλων δύο, με σημαντική χρονική διαφορά για τον υπολογισμό κάθε πρόβλεψης. Σε κάθε αλγόριθμο όμως, η διαφορά στην ακολουθιακή και την παράλληλη έκδοσή τους είναι αρκετά μεγάλη, και φτάνει σε ορισμένες περιπτώσεις έως και τα 2/3. Σε ακόμη μεγαλύτερα σύνολα δεδομένων τα χρονικά πλεονεκτήματα εφαρμογής της παράλληλης εκδοχής έναντι της ακολουθιακής θα είναι φυσικά όλο και εμφανέστερα.

Αν θεωρήσουμε ότι η επιτάχυνση δίνεται από τον λόγο του ακολουθιακού χρόνου εκτέλεσης προς τον παράλληλο,

$$επιτάχυνση = \frac{ακολουθιακός\ χρόνος}{παράλληλος\ χρόνος}$$

Τα παραπάνω αποτελέσματα φαίνονται στο παρακάτω διάγραμμα σε όρους επιτάχυνσης:



Διάγραμμα 18: Επιτάχυνση από την παραλληλοποίηση των αλγορίθμων

Για τους αλγορίθμους Map Reduce και Slope One, η επιτάχυνση αυξάνεται όσο μεγαλώνει το πρόβλημα αλλά με μικρότερο ρυθμό. Η μείωση στην επιτάχυνση του αλγορίθμου Knn, μπορεί να εξηγηθεί λόγω των περιορισμένων υπολογιστικών δυνατοτήτων του υπολογιστή που χρησιμοποιήθηκαν στις δοκιμές και των ήδη χρονοβόρων υπολογισμών που πραγματοποιεί. Ο χρόνος ωστόσο των παράλληλων έναντι των ακολουθιακών δοκιμών είναι πάντα καλύτερος.

5 Επίλογος

5.1 Σύνοψη και συμπεράσματα

Ο συναρτησιακός προγραμματισμός προσφέρει μία σειρά πλεονεκτημάτων, τα οποία μπορεί να μην είναι εκ πρώτοις εμφανή σε έναν προγραμματιστή που έχει συνηθίσει στο προστακτικό μοντέλο. Οι βρόχοι απλουστεύονται και σημασία δεν έχει το πώς θα πραγματοποιηθεί μία λειτουργία αλλά το ποια ακριβώς είναι αυτή η λειτουργία.

Η Java 8 άλλαξε τα δεδομένα της γλώσσας, προσφέροντας πολύ σημαντικά συναρτησιακά χαρακτηριστικά, όπως οι εκφράσεις λάμδα, το πέρασμα συναρτήσεων ως παραμέτρων αλλά και την ανάθεση συναρτήσεων ως τιμές, και φυσικά το stream API. Με το stream API, δίνεται η δυνατότητα για συγγραφή κώδικα που μοιάζει με τα ερωτήματα των βάσεων δεδομένων. Έτσι, η επεξεργασία γίνεται πιο κατανοητή και ευανάγνωστη.

Επιπλέον, ένα βασικό πλεονέκτημα και μία σημαντική αλλαγή είναι ο τρόπος με τον οποίο γίνεται η παράλληλη επεξεργασία. Το stream API προσφέρει έτοιμες μεθόδους για την παραλληλοποίηση των streams, με τις οποίες η δημιουργία των νημάτων γίνεται με τρόπο απλό και αδιαφανή. Στην υλοποίηση των αλγορίθμων συστημάτων συστάσεων της παρούσας εργασίας, όπου ήταν δυνατό χρησιμοποιήθηκε παραλληλισμός, καλώντας απλά μία από τις δύο μεθόδους, parallel ή parallelStream, στα streams που δημιουργούνται. Η παράλληλη επεξεργασία και η εκμετάλλευση όλων των επεξεργαστών του υλικού, μπορεί να προσφέρει σημαντικές επιταχύνσεις στους υπολογισμούς. Ωστόσο, η δημιουργία νημάτων, το μοίρασμα του υπολογισμού και η συλλογή των τελικών αποτελεσμάτων μπορεί να προσθέσει σημαντικό overhead στον συνολικό χρόνο. Γι' αυτό ο παραλληλισμός με σκοπό την βελτίωση της χρονικής απόδοσης έχει νόημα στην επεξεργασία μεγάλων συνόλων δεδομένων.

5.2 Μελλοντικές Επεκτάσεις

Το stream API προσφέρει πάρα πολλές δυνατότητες στον προγραμματιστή και λύσεις για τα περισσότερα προβλήματα. Με αφορμή το σύστημα συστάσεων που αναπτύχθηκε στα πλαίσια της εργασίας, μελετήθηκαν πολλές από αυτές τις δυνατότητες και βρέθηκαν λύσεις σε αρκετά προβλήματα.

Ως επέκταση του συστήματος που αναπτύχθηκε, προτείνεται η προσαρμογή του έτσι ώστε να μην λειτουργεί μόνο για ταινίες αλλά και για άλλους τύπους δεδομένων.

Αυτό μπορεί πιθανά να γίνει επεκτείνοντας την εφαρμογή έτσι ώστε να λειτουργεί με generic τύπους.

Ακόμα, μπορεί να γίνει υλοποίηση και άλλων αλγορίθμων, τόσο συνεργατικού φιλτραρίσματος, όσο και άλλων όπως αλγορίθμων με βάση το περιεχόμενο. Ιδιαίτερα σε αλγορίθμους με βάση το περιεχόμενο, είναι πιθανό να αναδεικνύεται ακόμα περισσότερο η δύναμη των streams, μιας και θα χρησιμοποιούνται περισσότερο στην λογική των ερωτημάτων για φιλτράρισμα του περιεχομένου.

Εκτός από αλγορίθμους για συστήματα συστάσεων, το stream API μπορεί να εφαρμοστεί σε πληθώρα προβλημάτων, ειδικά σε όσα έχουν να κάνουν με επεξεργασία δεδομένων. Η ιδιαίτερη σύνταξή τους, εμπνευσμένη από τον συναρτησιακό προγραμματισμό, επιτρέπει την ανάκτηση και επεξεργασία δεδομένων με παρόμοιο τρόπο όπως και σε μία γλώσσα βάσεων δεδομένων. Έτσι θα είχε ενδιαφέρον να υλοποιηθούν αλγόριθμοι του τομέα της Ανάλυσης Δεδομένων.

6 Βιβλιογραφία

[1] Wikipedia, «Wikipedia,» November 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Functional_programming.

[2] K. Chandrakant, «Baeldung,» December 2020. [Ηλεκτρονικό]. Available: https://www.baeldung.com/java-functional-programming.

[3] P.-Y. Saumont, Functional Programming in Java, Manning Publications Co., 2017.

[4] M. Türkmenoğlu, «Medium,» 29 July 2020. [Ηλεκτρονικό]. Available: https://medium.com/swlh/understand-the-key-functional-programming-concepts-bca440f1bcd6.

[5] M. Martin, «guru99,» October 2021. [Ηλεκτρονικό]. Available: https://www.guru99.com/functional-programming-tutorial.html#2.

[6] Wikipedia, «Wikipedia,» 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Java_(programming_language).

[7] M. F. A. M. Raoul-Gabriel Urma, Java 8 in Action, Manning Publications Co, 2015.

[8] Q. Charatan και A. Kans, «The Stream API,» σε *Java in Two Semesters. Texts in Computer Science*, Springer, Cham, 2019.

[9] C. S. Horstmann, Java SE for the Really Impatient, Pearson Education, Inc, 2014.

[10] J. Dean και S. Ghemawat, «Jeffrey Dean;Sanjay Ghemawat,» *Communications of the ACM,* τόμ. 51, αρ. 1.

[11] Wikipedia, «Sequence Alignment,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Sequence_alignment.

[12] Wikipedia, «FASTQ Format,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/FASTQ_format.

[13] Wikipedia, «Phred quality score,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Phred_quality_score.

[14] J. A. Konstan και J. Riedl, «Recommender systems: from algorithms to user experience,» 2012.

[15] E. Vozalis και K. G. Margaritis, «Analysis of Recommender Systems' Algorithms».

[16] D. Kluver, M. D. Ekstrand και J. A. Konstan, «Rating-Based Collaborative Filtering: Algorithms and Evaluation,» σε *Social Information Access*, Springer, Cham, 2018.

[17] X. Su και T. M. Khoshgoftaar, «A Survey of Collaborative Filtering Techniques,» *Advances in Artificial Intelligence,* 2009.

[18] K. Falk, Practical Recommender Systems, New York: Manning Publications, 2019.

[19] S. Khusro, Z. Ali και I. Ullah, «Recommender Systems: Issues, Challenges, and Research Opportunities,» σε *Information Science and Applications (ICISA) 2016*, Singapore, Springer, 2016.

[20] M. Vozalis και K. Margaritis, «Using SVD and demographic data for the enhancement of generalized Collaborative Filtering,» *Information Sciences,* τόμ. 177, 2007.

[21] M. Sollenborn και P. Funk, «Category-Based Filtering and User Stereotype Cases to Reduce the Latency Problem in Recommender Systems,» *Advances in Case-Based Reasoning* , 2002.

[22] «Combining content-based and collaborative filters in an online newspaper,» *ACM SIGIR Workshop on Recommender Systems-Implementation and Evaluation,* 1999.

[23] I. Avazpour, T. Pitakrat, L. Grunske και J. Grundy, «Dimensions and Metrics for Evaluating Recommendation Systems,» σε *Recommendation Systems in Software Engineering*, Springer, 2014.

[24] E. Karydi και K. G. Margaritis, «Parallel and Distributed Collaborative Filtering: A Survey,» 2014.

[25] M.Sridevi, R. Rao και M. Rao, «A Survey on Recommender System,» *International Journal of Computer Science and Information Security,* τόμ. 14, αρ. 5, 2016.

[26] M. Tsunoda, T. Kakimoto, N. Ohsugi, A. Monden και K.-i. Matsumoto, «Javawock: A Java Class Recommender System Based on Collaborative Filtering,» σε *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering*, Taipei, 2005.

[27] E. Karydi και K. G. Margaritis, «Parallel Implementation of the Slope One Algorithm for Collaborative Filtering,» σε *16th Panhellenic Conference on Informatics,* 2012.

[28] F. Isinkaye, Y. Folajimi και B. Ojokoh, «Recommendation systems: Principles, methods and evaluation,» *Egyptian Informatics,* 2015.

[29] F. Ricci, L. Rokach και B. Shapira, «Recommender Systems: Introduction and Challenges,» σε *Recommender Systems Handbook*, Boston, Springer, 2015.

[30] P. Lops, M. d. Gemmis και G. Semeraro, «Content-based Recommender Systems: State of the Art and Trends,» σε *Recommender Systems Handbook*, Boston, Springer, 2011.

[31] R. Burke, «Hybrid recommender systems: survey and experiments,» *User Modeling and User-Adapted Interaction,* 2002.

[32] J. Zhao, «jifuzhao.github.io,» [Ηλεκτρονικό]. Available: https://jifuzhao.github.io/2017/08/15/movie.html.

[33] Wikipedia, «Co-occurrence matrix,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Co-occurrence_matrix.

[34] D. Lemire και A. Maclachlan, «Slope One Predictors for Online Rating-Based Collaborative Filtering,» σε *Proceedings of the 2005 SIAM International Conference on Data Mining (SDM)*, 2005.

[35] baeldung, «A Collaborative Filtering Recommendation System in Java,» [Ηλεκτρονικό]. Available: https://www.baeldung.com/java-collaborative-filtering-recommendations.

[36] «Wikipedia,» 15 3 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[37] V. Kotu και B. Deshpande, Data Science Concepts and Practice, Cambridge, MA 02139, United States: Elsevier, 2019.

[38] B. Sarwar, G. Karypis, J. Konstan και J. Riedl, «Item-Based Collaborative Filtering Recommendation Algorithms,» GroupLens Research Group/Army HPC Research Center Department of Computer Science and Engineering University of Minnesota, Minneapolis.