

Τίτλος

Finding frequent items in data streams

Συγγραφέας

ΓΕΩΡΓΙΟΣ ΜΑΛΙΣΣΟΒΑΣ

Διπλωματική Εργασία

Επιβλέπων: Λ. ΓΕΩΡΓΙΑΔΗΣ

Ιωάννινα, Οκτώβριος 2021



**ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ**

**DEPARTMENT OF COMPUTER SCIENCE &
ENGINEERING
UNIVERSITY OF IOANNINA**

Ευχαριστίες

Καταρχήν ευχαριστώ τον επιβλέποντα καθηγητή μου Λουκά Γεωργιάδη για την βοήθεια που μου προσέφερε και την αμέριστη συμπαράστασή του.

Ακόμη η παρούσα διπλωματική είναι αφιερωμένη στην οικογένεια μου, στα άτομα που με στήριξαν και με βοήθησαν κατά την εκπόνησή της και σε εκείνους που ήταν δίπλα μου καθ' όλη τη διάρκεια των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Στις μέρες μας δημιουργείται καθημερινά όλο και περισσότερο η ανάγκη επεξεργασίας μεγάλου όγκου δεδομένων σε πραγματικό χρόνο. Πολλά υπολογιστικά προβλήματα αυτού του τύπου μπορούν να επιλυθούν με την ανάπτυξη μιας κατηγορίας αλγορίθμων οι οποίοι καλούνται *αλγόριθμοι ροής δεδομένων*.

Στη παρούσα διπλωματική εργασία μελετάμε δύο βασικά προβλήματα ανάλυσης ροών δεδομένων:

- (α) τον εντοπισμό των στοιχείων που εμφανίζονται συχνότερα σε μία ροή, και
- (β) την εύρεση των αντικειμένων που εμφανίζουν τις μεγαλύτερες μεταβολές μεταξύ δύο ροών δεδομένων.

Συγκεκριμένα, στο πλαίσιο της διπλωματικής, υλοποιήθηκε σε υπολογιστικό περιβάλλον της γλώσσας προγραμματισμού Java, η δομή δεδομένων Count Sketch που προτάθηκε από τους Charikar, Chen και Farach-Colton. Η δομή αυτή διατηρεί εκτιμήσεις του πλήθους των εμφανίσεων των αντικειμένων μιας ροής δεδομένων, χρησιμοποιώντας πολύ μικρό αποθηκευτικό χώρο, και υποστηρίζει την εύρεση των πιο συχνών αντικειμένων της ροής με αρκετά μεγάλη πιθανότητα. Με βάση αυτή τη δομή, μπορούμε επίσης να επιλύσουμε το ακόλουθο πρόβλημα. Λαμβάνοντας ως είσοδο δυο ροές δεδομένων, στις οποίες εμφανίζονται αντικείμενα από ένα κοινό σύνολο αντικειμένων, επιθυμούμε να βρούμε τα αντικείμενα με τη μέγιστη διαφορά συχνότητας εμφάνισης στις δύο ροές.

Λέξεις Κλειδιά: Αλγόριθμος ροής, μοντέλο ροής, αλγόριθμος Count-Sketch, καθολικός Κατακερματισμός, αλγόριθμος Brute-Force, μετρητής εκτίμησης

ABSTRACT

Nowadays, Streaming algorithms are becoming extremely important as people push more and more to real-time processing. Many computational problems of this type can be solved by developing a class of algorithms called data Streaming algorithms.

In this dissertation we study two main problems of data Stream analysis:

- (a) identifying the items that occur most frequently in a stream, and
- (b) finding items with the largest frequency change

Specifically, in the context of thesis, the Count Sketch data structure proposed by Charikar, Chen and Farach-Colton was implemented in a Java environment. This structure maintains estimates of the number of occurrences of objects in a stream, using very little storage space, and supports finding the most common frequent objects with a fairly high probability. Based on this structure, we can also solve the following problem. Taking as input two data streams, in which objects from a common set of objects are displayed, we wish to find the objects with the maximum occurrence frequency difference in the two streams.

Keywords: Streaming Algorithm, Streaming model, Count-Sketch Algorithm, Universal Hashing, Brute-Force Algorithm, count estimate

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ	1
1.1 Η χρησιμότητα των Streaming Αλγορίθμων και τα προβλήματα	1
1.2 Στόχοι διπλωματικής εργασίας.....	2
1.3 Οργάνωση και Δομή διπλωματικής εργασίας	2
ΚΕΦΑΛΑΙΟ 2. ΘΕΩΡΗΤΙΚΟ ΠΛΑΙΣΙΟ	3
2.1 Data Stream Model και Μοντέλο υπολογισμού.....	3
2.2 Εύρεση του πιο συχνού αντικειμένου	4
2.3 Αντικείμενα που αλλάζουν πιο συχνά.....	4
2.4 Άλλα προβλήματα των Streaming αλγορίθμων	5
ΚΕΦΑΛΑΙΟ 3- ΘΕΩΡΗΤΙΚΟ ΠΛΑΙΣΙΟ	7
3.1 Ο αλγόριθμος Count-Sketch	7
3.2 Η Δομή του Count-Sketch.....	7
3.3 Η ανάλυση του Count-Sketch	8
3.4 Παράδειγμα του Count-Sketch.....	9
3.5 Ο αλγόριθμος APPROXTOP(S; k; ϵ ;)	12
3.6.1 Καθολικός κατακερματισμός	13
3.6.2 2-Καθολική Διασπορά.....	13
3.7 Αλγόριθμος Brute-Force	14
ΚΕΦΑΛΑΙΟ 4- ΥΛΟΠΟΙΗΣΗ	15
4.1 CountSketch.java	15
4.2 primalityTest.java.....	18
4.3 Pair.java.....	19
4.4 differenttxt.java	19
4.5 Main.java.....	19
4.6 Βιβλιοθήκες της Java.....	20
ΚΕΦΑΛΑΙΟ 5- ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ.....	22

5.1 Περιβάλλον εκτέλεσης πειραμάτων	22
5.2 Πειραματικά Δεδομένα	23
5.3 Πειραματικά Αποτελέσματα	23
5.3.1 Αρχικοποίηση του Count-Sketch	23
5.3.2 Αποτελέσματα για μια ροή.....	24
5.3.3 Αποτελέσματα για δυο ροές.....	28
ΚΕΦΑΛΑΙΟ 6- ΕΠΙΛΟΓΟΣ	35
6.1 Συμπεράσματα.....	35
6.2 Μελλοντικές Προοπτικές	36
ΒΙΒΛΙΟΓΡΑΦΙΑ.....	37

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

Εικόνα 1: διαβάζει το στοιχείο, υπολογίζει κάθε ένα από τα d μετρητές αυτού του στοιχείου και ενημερώνει τις αντίστοιχες τιμές στο σκίτσο ^[14]	8
Εικόνα 2Α: Παράδειγμα Count-Sketch διάνυσμα (V1)	10
Εικόνα 2Β: Παράδειγμα Count-Sketch διάνυσμα (V1)	11
Εικόνα 2Γ: Παράδειγμα Count-Sketch διάνυσμα (V1).....	11
Εικόνα 2Δ: Παράδειγμα Count-Sketch διάνυσμα (V2)	12
Εικόνα 2Ε: Παράδειγμα Count-Sketch (Αποτέλεσμα)	12
Εικόνα 3Α: Εκτέλεση του αρχείου “mushrooms.txt”	23
Εικόνα 3Β: Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch	23
Εικόνα 3Γ: Τα αντικείμενα από 1 έως 32 μαζί με συχνότητα που υπάρχουν στο αρχείο(εκτέλεση Brute-Force)	24
Εικόνα 3Δ: Τα αντικείμενα από 33 έως 70 μαζί με συχνότητα που υπάρχουν στο αρχείο (εκτέλεση Brute-Force)	24
Εικόνα 3Ε: Τα αντικείμενα από 33 έως 107 μαζί με συχνότητα που υπάρχουν στο αρχείο (εκτέλεση Brute-Force)	25
Εικόνα 3Ζ: Τα αντικείμενα από 108 έως 128 μαζί με συχνότητα που υπάρχουν στο αρχείο, καθώς επίσης και τα 10 πιο συχνά (εκτέλεσηBrute- Force).....	25
Εικόνα 4: Εκτέλεση του αρχείου “mushrooms txt”, για $\epsilon=0,03f$ και $\delta=0.03f$	26
Εικόνα 5: Εκτέλεση του αρχείου “mushrooms txt”, για $\epsilon=0,9f$ και $\delta=0.9f$	27
Εικόνα 6Α: Εκτέλεση του αρχείου “mushrooms.txt” και του αρχείου “connect txt”.....	29
Εικόνα 6Β: Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “mushrooms.txt”	29
Εικόνα 6Γ: Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “connect.txt”	30
Εικόνα 6Δ: Τα αντικείμενα από την εναλλαγή στοιχείων από 1 έως 32 μαζί με συχνότητα που υπάρχουν στα δύο αρχείο (εκτέλεση Brute-Force)	30
Εικόνα 6Ε: Τα αντικείμενα από την εναλλαγή στοιχείων από 33 έως 65 μαζί με συχνότητα που υπάρχουν στα δύο αρχείο (εκτέλεση Brute-Force)	31
Εικόνα 6Ζ: Τα αντικείμενα από την εναλλαγή στοιχείων από 66 έως 98 μαζί με συχνότητα που υπάρχουν στα δύο αρχείο (εκτέλεση Brute-Force)	31

Εικόνα 6H: Τα αντικείμενα από την εναλλαγή στοιχείων από 99 έως 129 μαζί με συχνότητα που υπάρχουν στα δύο αρχεία (εκτέλεση Brute-Force)	32
Εικόνα 6Θ: Τα αντικείμενα από Τα 10 πιο συχνά αντικείμενα μετά από τις εναλλαγές των δύο αρχείων (εκτέλεση Brute-Force).....	32
Εικόνα 7A: Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “mushrooms.txt”(για μικρές παραμέτρους).....	34
Εικόνα 7B: Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “mushrooms.txt”(για μικρές παραμέτρους).....	34
Εικόνα 8A: Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “mushrooms.txt”(για μικρές παραμέτρους).....	35
Εικόνα 8B: Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “mushrooms.txt”(για μικρές παραμέτρους).....	35

ΚΑΤΑΛΟΓΟΣ-ΣΧΗΜΑΤΑ ΠΙΝΑΚΩΝ

Αλγόριθμος 1: ο αλγόριθμος Count-Sketch	8
Επισκόπηση Κώδικα 1: Αρχικοποίηση Αντικειμένων	15
Επισκόπηση Κώδικα 2: Σύγκριση μεταξύ δύο ροών	15
Επισκόπηση Κώδικα 3: κλάση κληρονομικότητας ταξινομώντας τον Χάρτη Διεπαφής(HashMap).....	16
Αλγόριθμος 2: Η δομή του αλγορίθμου Miller-Rabin Test	18
Επισκόπηση Κώδικα 4: Το κάλεσμα της CountSketch.java για ένα αρχείο	19
Επισκόπηση Κώδικα 5: Το κάλεσμα της CountSketch.java για δύο αρχεία.....	20
Επισκόπηση Κώδικα 6: Το κάλεσμα της κλάσης differenttxt.java	20
Πίνακας 1: Συγκεντρωτικά τα στοιχεία του συστήματος που έγιναν οι δοκιμές του κώδικα που αναπτύχθηκε	22
Πίνακας 2:Συνοπτικός πίνακας Αρχικοποιήσεων	23
Πίνακας 3: Συνοπτικός πίνακας όλων των αποτελεσμάτων για μια ροή	28
Πίνακας 4: Συνοπτικός πίνακας όλων των αποτελεσμάτων για δυο ροές	36

ΕΙΣΑΓΩΓΗ

1.1 Η χρησιμότητα των Streaming Αλγορίθμων και τα προβλήματα

Στις μέρες μας δημιουργείται καθημερινά όλο και περισσότερο η ανάγκη επεξεργασίας μεγάλου όγκου δεδομένων σε πραγματικό χρόνο. Πολλά υπολογιστικά προβλήματα αυτού του τύπου μπορούν να επιλυθούν με την ανάπτυξη μιας κατηγορίας αλγορίθμων οι οποίοι καλούνται *αλγόριθμοι ροής δεδομένων*.

Ορισμένοι από αυτούς τους αλγόριθμους είναι ευρέως γνωστοί, όπως μετρητές k-min ή hyper log log. Υπάρχουν και άλλοι νεότεροι σημαντικοί αλγόριθμοι, όπως το t-digest, streaming k-means, Count-Min-Sketch και ο Count-Sketch, όπου είναι και βασικός αλγόριθμος που μελετήθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας.^[1]

Γενικά οι αλγόριθμοι ροής, στην επιστήμη των υπολογιστών είναι αλγόριθμοι για την επεξεργασία ροών δεδομένων, στους οποίους η είσοδος παρουσιάζεται ως μια ακολουθία στοιχείων και μπορεί να εξεταστεί σε λίγα μόνο περάσματα, που τις περισσότερες φορές αρκεί και μόνο ένα.

Στα περισσότερα μοντέλα, αυτοί οι αλγόριθμοι έχουν πρόσβαση σε περιορισμένη μνήμη (γενικά λογαριθμική στο μέγεθος και τη μέγιστη τιμή στη ροή). Αυτοί οι περιορισμοί μπορεί να σημαίνουν ότι ένας αλγόριθμος παράγει μια κατά προσέγγιση απάντηση που βασίζεται σε μια περίληψη ή "σκίτσο" της ροής δεδομένων.

Βασικοί αλγόριθμοι ροών δεδομένων είχαν μελετηθεί από τους Munro και Paterson ήδη από το 1978, καθώς και μεταγενέστερα από τους Philippe Flajolet και G. Nigel Martin το 1982/83.

Όμως η επιστημονική περιοχή των αλγορίθμων ροής τυπώθηκε και δημοσιοποιήθηκε για πρώτη φορά το 1996 σε μια πρότυπη δημοσίευση από τους Noga Alon, Yossi Matias και Mario Szegedy.^[12]

Η απόδοση ενός αλγορίθμου που λειτουργεί σε ροές δεδομένων μετριέται από τρεις βασικούς παράγοντες:

A) Τον αριθμό των περασμάτων που πρέπει να κάνει ο αλγόριθμος πάνω από τη ροή.

B) Το μέγεθος της απαιτούμενης μνήμης.

Γ) Τον χρόνο λειτουργίας του αλγορίθμου.

Παρατηρούμε επίσης ότι αυτοί οι αλγόριθμοι έχουν πολλές ομοιότητες με τους άμεσους (online) αλγορίθμους, αφού και οι δύο απαιτούν να ληφθούν αποφάσεις πριν να είναι διαθέσιμα όλα τα

δεδομένα, όμως δεν είναι πανομοιότυποι. Οι αλγόριθμοι ροής δεδομένων διαθέτουν μόνο περιορισμένη μνήμη, αλλά ενδέχεται να είναι σε θέση να αναβάλουν τη δράση τους μέχρι να φτάσει μια ομάδα αντικειμένων, ενώ οι άμεσοι αλγόριθμοι τυπικά θα πρέπει να ανανεώσουν τον υπολογισμό τους αμέσως μόλις εμφανιστεί ένα νέο αντικείμενο εισόδου.

Όσον αφορά τις εφαρμογές, οι αλγόριθμοι ροής έχουν διάφορες εφαρμογές στη δικτύωση, όπως η παρακολούθηση συνδέσεων δικτύου για ροές ελέφαντα (elephants flow), η καταμέτρηση του αριθμού των διακριτών ροών, η εκτίμηση της κατανομής των μεγεθών ροής κ.ο.κ. Έχουν επίσης εφαρμογές σε βάσεις δεδομένων, όπως η εκτίμηση του μεγέθους μιας σύνδεσης.^[2]

1.2 Στόχοι διπλωματικής εργασίας

Στη παρούσα διπλωματική εργασία μελετάμε δύο βασικά προβλήματα ανάλυσης ροών δεδομένων:

- (α) τον εντοπισμό των στοιχείων που εμφανίζονται συχνότερα σε μία ροή, και
- (β) την εύρεση των αντικειμένων που εμφανίζουν τις μεγαλύτερες μεταβολές μεταξύ δύο ροών δεδομένων.

1.3 Οργάνωση και Δομή διπλωματικής εργασίας

Η Διπλωματική εργασία χωρίζεται σε 6 κεφάλαια.

Το Κεφάλαιο 2, αναφέρεται στο θεωρητικό πλαίσιο που μελετάμε, ορίζοντας τους streaming αλγόριθμους, κάποια προβλήματα που επιλύουν, όπως επίσης και τα βασικά προβλήματα που θα ασχοληθούμε στην συγκεκριμένη εργασία.

Το Κεφάλαιο 3, αναφέρεται στον αλγόριθμο που αναπτύχθηκε η παρούσα Διπλωματική εργασία (Αλγόριθμος του Count-Sketch), με την ανάλυση που έχει καθώς επίσης και ένα χρήσιμο παράδειγμα. Ακόμη αναφέρεται στον αλγόριθμο του APPROXTOP και τον αλγόριθμο της Brute-Force και ορίζει τους πίνακες κατακερματισμού.

Το Κεφάλαιο 4 παρουσιάζει την υλοποίηση της δομής Count Sketch σε γλώσσα προγραμματισμού Java. Σε αυτό το κεφάλαιο περιλαμβάνονται επίσης, κομμάτια κώδικα που αφορούν τις συναρτήσεις κατακερματισμού, όπως και σημαντικές συναρτήσεις που χρησιμοποιήθηκαν για κάθε κλάση.

Το Κεφάλαιο 5 παρουσιάζει την πειραματική αξιολόγηση του προγράμματος μας, χρησιμοποιώντας δεδομένα από γνωστές συλλογές. Επίσης παρουσιάζονται τα συμπεράσματα στα οποία καταλήξαμε με βάση τα πειραματικά αποτελέσματα.

Το Κεφάλαιο 6 παρουσιάζει τα συμπεράσματα από την εργασία, αλλά και τις προοπτικές για μελλοντικές εργασίες και επεκτάσεις πάνω στη Διπλωματική.

ΘΕΩΡΗΤΙΚΟ ΠΛΑΙΣΙΟ

Σε αυτό το κεφάλαιο παρουσιάζουμε το θεωρητικό πλαίσιο της εργασίας. Πιο συγκεκριμένα θα ορίσουμε το μοντέλο υπολογισμού ροών δεδομένων, και θα αναφερθούμε στα δύο προβλήματα στα οποία εστιάζουμε στην εργασία, δηλαδή την εύρεση των πιο συχνών αντικειμένων και των αντικειμένων που αλλάζουν πιο συχνά.

Καθώς επίσης και σε άλλα προβλήματα που απασχολούν οι ροές δεδομένων.

2.1 Μοντέλο υπολογισμού ροής Δεδομένου

Στο μοντέλο ροής δεδομένων λαμβάνουμε ως είσοδο μια ακολουθία ακέραιων αριθμών, η οποία γενικά δεν είναι διαθέσιμη για τυχαία πρόσβαση, αλλά αντ' αυτού πρέπει να εξετάσουμε ένα προς ένα τα στοιχεία της ροής, όπως αυτά φτάνουν. Οι αλγόριθμοι είναι γενικά περιορισμένοι να χρησιμοποιούν χώρο που είναι λογαριθμικός σε m και n , όπου m είναι το μέγεθος της ροής και n το πλήθος των διαφορετικών αντικειμένων που βρίσκονται στην ροή. Γενικά μπορούν να κάνουν μόνο έναν μικρό αριθμό περασμάτων πάνω από μια ροή, αλλά μερικές φορές αρκεί και μόνο ένα.^[2]

Το μοντέλο υπολογισμού που ζητάμε στο πλαίσιο της εργασίας, είναι ότι δεν αποθηκεύουμε όλο το Stream, αλλά αποθηκεύουμε μόνο ορισμένες πληροφορίες που μας βοηθούν να κάνουμε κάποιους υπολογισμούς.

Πιο συγκεκριμένα έχουμε τις παραμέτρους m, n που ορίσαμε παραπάνω και μία είσοδο ροής σ , όπου $\sigma = \langle a_1, a_2, \dots, a_m \rangle$ και τα στοιχεία της ακολουθίας, αντλούνται από το πλήθος των αντικειμένων, $[n] := \{1, 2, \dots, n\}$.

Ο κεντρικός μας στόχος θα είναι να επεξεργαστούμε τη ροή εισόδου χρησιμοποιώντας μια μικρή ποσότητα χώρου s , δηλαδή να χρησιμοποιήσουμε s λέξεις της λειτουργικής μνήμης. Δεδομένου ότι το m και το n μπορεί να θεωρηθούν ως πολύ μεγάλα, θέλουμε να τα κάνουμε πολύ μικρότερα από αυτά. Τυπικά θα θέλαμε το s να είναι υπογραμμικό τόσο στο m όσο και στο n δηλαδή

$$s = o(\min \{m, n\}).$$

Τυπικά, ένας αλγόριθμος ροής χρησιμοποιεί

$s = O(\log m + \log n)$, επειδή αυτό το μέγεθος του χώρου είναι αυτό που χρειαζόμαστε για να αποθηκεύσουμε έναν σταθερό πλήθος πληροφοριών από το ρεύμα, καθώς και έναν σταθερό πλήθος μετρητών που μπορούν να μετρήσουν μέχρι το μήκος της ροής.

Ο λόγος για τον οποίο ονομάζεται είσοδος ροής είναι ότι επιτρέπεται η πρόσβαση στην είσοδο μόνο με "τρόπο ροής". Με λίγα λόγια δεν έχουμε τυχαία πρόσβαση στα μεγέθη και μπορούμε μόνο να σαρώσουμε την ακολουθία με μια δεδομένη σειρά. ^[4]

2.2 Εύρεση του πιο συχνού αντικειμένου

Ένα από τα προβλήματα που θα μας απασχολήσει στο πλαίσιο της Διπλωματικής εργασίας που μελετάμε, είναι το πρόβλημα εύρεσης του πιο συχνού στοιχείου.

Το συγκεκριμένο πρόβλημα βρίσκει εφαρμογή σε διάφορους τομείς όπως: χρηματοπιστωτικά συστήματα, παρακολούθηση για την επισκεψιμότητα σε διάφορες ιστοσελίδες, διαφημίσεις στο διαδίκτυο κ.α. Ακόμη χρησιμεύει ως βάση για την επίλυση άλλων σχετικών προβλημάτων, όπως ο προσδιορισμός συχνών συνόλων και συχνών προσφάτων στοιχείων.

Για την αντιμετώπιση του συγκεκριμένου προβλήματος έχουμε δύο μεθόδους: την counter-based (μετρητές) και την Sketch-based (σκίτσα).

Οι αλγόριθμοι που βασίζονται με την μέθοδο της counter-based (μετρητές), διατηρούν μετρητές με ένα σταθερό αριθμό στοιχείων της ροής και μόνο αυτόν μπορούν να ελέγξουν. Αν αυτό το στοιχείο φτάσει στη ροή που θέλουμε, ο συγκεκριμένος μετρητής αυξάνεται, αλλιώς ο αλγόριθμος αποφασίζει εάν θα απορρίψει το στοιχείο ή θα αναθέσει έναν υπάρχοντα μετρητή σε αυτό το στοιχείο.

Κάποιοι αλγόριθμοι που έχουν εφαρμογή σε αυτή την μέθοδο είναι: ο Sticky Sampling and Lossy Counting (LC), Frequent (Freq) και άλλοι.

Η άλλη προσέγγιση είναι να διατηρείται ένα σκίτσο της ροής δεδομένων, χρησιμοποιώντας τεχνικές όπως ο κατακερματισμός, ώστε να μπορεί να χαρτογραφήσει αντικείμενα σε μειωμένο σύνολο μετρητών. Οι τεχνικές που βασίζονται σε σκίτσα διατηρούν κατά προσέγγιση τον αριθμό συχνοτήτων όλων των στοιχείων στη ροή και μπορούν επίσης να υποστηρίξουν διαγραφές. Ως εκ τούτου, αυτοί οι αλγόριθμοι είναι πολύ πιο ευέλικτοι από την μέθοδο με τους μετρητές.

Κάποιοι τέτοιοι αλγόριθμοι είναι: το Count-Sketch, Group-Test, Count Min-Sketch (CM) και άλλοι. ^[3]

2.3 Αντικείμενα που αλλάζουν πιο συχνά

Ένα άλλο πρόβλημα που θα μας απασχολήσει στην συγκεκριμένη εργασία είναι η εύρεση των αντικειμένων που εμφανίζουν τις μεγαλύτερες μεταβολές μεταξύ δύο ροών δεδομένων. Η αλλαγή αυτή έχει εκτεταμένο αντίκτυπο σε οποιονδήποτε αλγόριθμο επεξεργασίας δεδομένων. Για παράδειγμα, κατά την κατασκευή μοντέλων εξόρυξης ροής δεδομένων, τα δεδομένα που έφτασαν πριν από μια αλλαγή μπορούν να χρησιμοποιήσουν τα μοντέλα προς χαρακτηριστικά που δεν ισχύουν πλέον. Αν επεξεργαζόμαστε ξανά τα queries (ερωτήματα) των ροών δεδομένων, μπορεί να

θέλουμε να δώσουμε ξεχωριστές απαντήσεις για κάθε χρονικό διάστημα όπου η υποκείμενη κατανομή δεδομένων είναι σταθερή.

2.4 Άλλα προβλήματα των Streaming αλγορίθμων

Άλλα προβλήματα που προκύπτουν από τους αλγόριθμους ροής είναι:

- Υπολογισμός ροπών συχνότητας

Μια άμεση προσέγγιση για την εύρεση των ροπών συχνότητας απαιτεί τη διατήρηση ενός καταχωρητή m_i για όλα τα ξεχωριστά σύνολα συχνοτήτων a_i , όπου $a_i \in (1, 2, 3, 4, \dots, N)$ και μνήμη της τάξης $\Omega(N)$. Αλλά έχουμε περιορισμούς χώρου και απαιτούμε έναν αλγόριθμο που υπολογίζει σε πολύ χαμηλότερη μνήμη. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας προσεγγίσεις αντί για ακριβείς τιμές. Ένας αλγόριθμος που υπολογίζει μια (ϵ, δ) προσέγγιση, όπου ϵ είναι η ποιότητα της λύσης και όπου δ η πιθανότητα σφάλματος είναι η ροπή συχνότητας $F_k(a) = \sum_{i=1}^n a_k^i$, όπου F'_k είναι η (ϵ, δ) - κατά προσέγγιση τιμή του F_k . [2]

- Ανίχνευση συμβάντων

Η ανίχνευση συμβάντων σε ροές δεδομένων γίνεται συχνά χρησιμοποιώντας αλγορίθμους Heavy hitters, δηλαδή τα πιο συχνά αντικείμενα και η συχνότητά τους προσδιορίζονται χρησιμοποιώντας έναν από τους αλγόριθμους του Sketch και Count, τότε η μεγαλύτερη αύξηση σε σχέση με το προηγούμενο χρονικό σημείο αναφέρεται ως τάση. Αυτή η προσέγγιση μπορεί να βελτιωθεί χρησιμοποιώντας εκθετικά σταθμισμένους κινητούς μέσους όρους και διακύμανση για την εξομάλυνση. [2]

- Καταμέτρηση Διακριτών στοιχείων

Η καταμέτρηση του αριθμού των διακριτών στοιχείων σε ένα ρεύμα (μερικές φορές ονομάζεται στιγμή F_0) είναι ένα άλλο πρόβλημα που έχει μελετηθεί καλά.

Χρησιμοποιεί χώρο $O(\epsilon^2 + \log d)$, όπου ϵ είναι η παράμετρος για την ποιότητα της λύσης, με $O(1)$ χειρίστο χρόνο, καθώς και καθολικές λειτουργίες κατακερματισμού και μια ανεξάρτητη οικογένεια κατακερματισμού r -wise, όπου

$$r = \Omega\left(\frac{\log(\frac{1}{\epsilon})}{\log \log(\frac{1}{\epsilon})}\right). [2]$$

- Προβλήματα Μέγιστης και ελάχιστης Κάλυψης

Δυστυχώς, και για τα δύο προβλήματα, ούτε οι άπληστοι αλγόριθμοι αλλά και ούτε οι αλγόριθμοι γραμμικού προγραμματισμού δεν κλιμακώνονται καλά σε μαζικά σύνολα δεδομένων. Αυτό έχει παρακινήσει σημαντική ερευνητική προσπάθεια στο σχεδιασμό αλγορίθμων που θα μπορούσαν να

χειριστούν σύγχρονα μεγάλα δεδομένα μοντέλα υπολογισμού όπως το μοντέλο ροής δεδομένων και το μοντέλο MapReduce . Υποστηρίζεται ότι οποιοσδήποτε αλγόριθμος για το μοντέλο δυναμικής ροής γραφήματος μπορεί επίσης να χρησιμοποιηθεί σε μοντέλο σετ ροής, όπου το μοντέλο σετ ροής είναι απλώς μια ειδική περίπτωση στην οποία δεν μπορούν να υπάρξουν διαγραφές και οι άκρες δεν ομαδοποιούνται κατά το τελικό σημείο.

Υπάρχουν και πολλά άλλα προβλήματα που προκύπτουν από τους αλγορίθμους ροής, όμως εμείς θα ασχοληθούμε με τα προβλήματα της ενότητας 2.2 και 2.3

COUNT-SKETCH

Στο Κεφάλαιο 3, παρουσιάζεται ο Count-Sketch που είναι και ο βασικός αλγόριθμος που αναπτύχθηκε για την Διπλωματική εργασία. Επίσης θα δείξουμε πως ο Count-Sketch που δίνει αλγόριθμο ενός περάσματος, μπορεί με αξιόπιστα να εκτιμήσει τη συχνότητα από τα πιο συνηθισμένα στοιχεία, τα οποία αποδίδουν άμεσα έναν αλγόριθμο 1 περάσματος με σκοπό την επίλυση της APPROXTOP(S; k; ε;), όπως επίσης και ένα παράδειγμα εκτέλεσης του αλγορίθμου. Ακόμη παρουσιάζεται αναλυτικά η συνάρτηση της APPROXTOP(S; k; ε;), της Brute-Force και οι πίνακες κατακερματισμού.

3.1 Ο αλγόριθμος Count-Sketch

Ο Count-Sketch ή αλλιώς το σκίτσο καταμέτρησης, είναι ένας τύπος μείωσης δεδομένων που είναι ιδιαίτερα αποτελεσματικός στη στατιστική, τη μηχανική μάθηση και τους αλγορίθμους. Εφευρέθηκε από τους Moses Charikar, Kevin Chen και Martin Farach-Colton σε μια προσπάθεια να επιταχύνουν το AMS Sketch των Alon, Matias και Szegedy για την προσέγγιση των στιγμών συχνότητας των ροών.

Το σκίτσο είναι σχεδόν πανομοιότυπο με τον αλγόριθμο Feature hashing του John Moody, αλλά διαφέρει στη χρήση συναρτήσεων κατακερματισμού με μικρό βαθμό ανεξαρτησίας, γεγονός που το καθιστά πιο πρακτικό. Για να υπάρχει ακόμα μεγάλη πιθανότητα επιτυχίας, η διάμεσος χρησιμοποιείται για να συγκεντρώσει σκίτσα πολλαπλών μετρήσεων, αντί να χρησιμοποιούμε τον μέσο όρο.

Αυτές οι ιδιότητες επιτρέπουν τη χρήση για σαφείς μεθόδους πυρήνα, διγραμμική συγκέντρωση σε νευρωτικά δίκτυα και αποτελεί ακρογωνιαίο λίθο σε πολλούς αριθμητικούς αλγορίθμους της γραμμικής άλγεβρας.^[5]

3.2 Η Δομή του Count-Sketch

Σύμφωνα με την Διπλωματική εργασία που μελετάμε, ο αλγόριθμος Count-Sketch περιγράφεται στον Αλγόριθμος 1.

ADD(C, q): For $i \in [1, t]$; $h_i[q] += s_i[q]$.

$$\text{ESTIMATE}(C, q): \text{return median}_i \{h_i[q] \cdot s_i[q]\}.$$

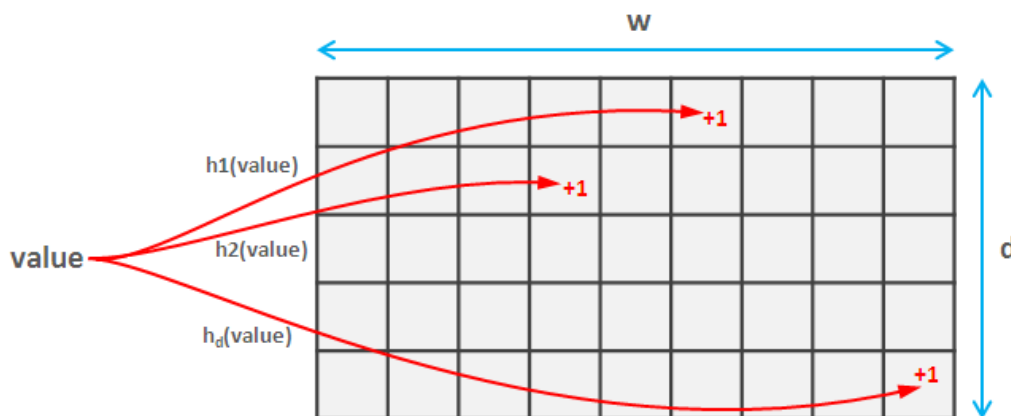
Αλγόριθμος 1- ο αλγόριθμος Count-Sketch

Έστω t και b παράμετροι με τιμές $t = \Theta(\log(\frac{n}{\delta}))$ και $b=8k$, όπου k τα πιο συχνά αντικείμενα που επιθυμούμε να υπολογίσουμε. Ορίζουμε συναρτήσεις κατακερματισμού h_1, \dots, h_t οι οποίες αντιστοιχούν τα αντικείμενα της ροής στο σύνολο τιμών $\{1, \dots, b\}$ και έστω s_1, \dots, s_t συναρτήσεις κατακερματισμού που αντιστοιχούν τα αντικείμενα στο σύνολο τιμών $\{+1, -1\}$. Ακόμη τα s_i πρέπει να είναι ανεξάρτητα κατά ζεύγη και όλες οι συναρτήσεις κατακερματισμού πρέπει να είναι ανεξάρτητες μεταξύ τους.

Η δομή δεδομένων COUNT-SKETCH αποτελείται από τις παραπάνω συναρτήσεις κατακερματισμού, μαζί με έναν πίνακα C μετρητών $t \times b$, ο οποίος ερμηνεύεται ως πίνακας από t πίνακες κατακερματισμού, όπου ο καθένας περιέχει b κάδους (μετρητές). Η λειτουργία κατακερματισμού h_i χαρτογραφεί τα αντικείμενα q σε έναν από τους μετρητές b στον πίνακα κατακερματισμού i -οστής θέσης και για αυτό το λόγο προκύπτει ο συμβολισμός $h_i[q]$.

Σύμφωνα με τον Αλγόριθμο 1, η λειτουργία ADD παίρνει ένα νέο στοιχείο q και αυξάνει ή μειώνει με κατάλληλο μετρητή τον κάθε πίνακα κατακερματισμού, ανάλογα με την τιμή $s_i[q]$ και η λειτουργία ESTIMATE(εκτίμηση) επιστρέφει την εκτιμώμενη καταμέτρηση ενός στοιχείου λαμβάνοντας το μέσο όρο από όλους τους μετρητές συσχετιζόμενη με το στοιχείο της τιμής μετρητή πολλαπλασιασμένο με $s_i[q]$.

Παρατηρούμε ότι η τελική μας εκτίμηση λαμβάνεται από τη διάμεσο των εκτιμήσεων από τους πίνακες κατακερματισμού t , αντί για το μέσο όρο. Αυτό συμβαίνει γιατί δεν εξαλείφεται εντελώς το πρόβλημα των συγκρούσεων με στοιχεία υψηλής συχνότητας, και αυτά θα εξακολουθούν να εισάγουν μεγάλα σφάλματα σε κάποιο υποσύνολο των εκτιμήσεων. Ο μέσος όρος είναι πολύ ευαίσθητος σε ακραίες τιμές, ενώ ο διάμεσος επηρεάζεται λιγότερο από αυτές και για αυτό τον λόγο την χρησιμοποιούμε.



Εικόνα 1- διαβάζει το στοιχείο, υπολογίζει κάθε ένα από τα d μετρητές αυτού του στοιχείου και ενημερώνει τις αντίστοιχες τιμές στο σκίτσο^[14].

3.3 Η ανάλυση του Count-Sketch

Σύμφωνα με την Διπλωματική εργασία που μελετάμε, υποθέτουμε ότι οι συναρτήσεις κατακερματισμού h_i και s_i (όπου αναλύονται στο Κεφάλαιο 3.6.1 και 3.6.2) είναι ανεξάρτητες κατά ζεύγη, όπως επίσης και όλες οι συναρτήσεις κατακερματισμού είναι ανεξάρτητες μεταξύ τους. Ο χρόνος που χρειάζεται για να εφαρμοστούν οι συναρτήσεις κατακερματισμού για μια τυχαία τιμή είναι $O(t \log m)$. Ακόμη θα χρησιμοποιήσουμε $t = \Theta(\log(\frac{n}{\delta}))$, όπου ο αλγόριθμος αποτυγχάνει με πιθανότητα στην παράμετρο δ (το οποίο αναφέρεται στην ενότητα 3.2).

Οπότε προκύπτει ότι το πλήθος των δυαδικών ψηφίων που απαιτείται είναι συνολικά $O(\log m \log(\frac{n}{\delta}))$.

Στην εργασία των Charikar, Chen και Farach-Colton, αποδεικνύεται το ακόλουθο αποτέλεσμα (θεώρημα 1). Ο Count-Sketch αλγόριθμος λύνει το πρόβλημα APPROXTOP($S; k; \epsilon$) (όπου S είναι μια ροή, k ένας ακέραιος και ϵ ένας οποιοδήποτε πραγματικός αριθμός) με $t = \Theta(\log(\frac{n}{\delta}))$, σε χώρο

$$O(k \log(\frac{n}{\delta}) + \frac{\sum_{q'=k+1}^m n_{q'}^2}{(\epsilon n_k)^2} \log(\frac{n}{\delta})).$$

Υποθέτουμε την εκτίμηση της συχνότητας ενός στοιχείου στη θέση l σαν είσοδο.

Έστω $n_q(l)$ ο αριθμός εμφανίσεων του στοιχείου q μέχρι τη θέση l .

Έστω $A_i[q]$ το σύνολο των στοιχείων που έχουν κατακερματιστεί στον ίδιο κάδο του i -οστού πίνακα κατακερματισμού με το στοιχείο q , δηλαδή

$$A_i[q] = \{q' : q' \neq q; h_i[q'] = h_i[q]\}.$$

Έστω $A_i^{>k}[q]$ να είναι αντικείμενα του $A_i[q]$, εκτός από τα k πιο συχνά αντικείμενα, δηλαδή $A_i^{>k}[q] = \{q' : q' \neq q, q' > k; h_i[q'] = h_i[q]\}$.

$$\text{Έστω } v_i[q] = \sum_{q' \in A_i[q]} n_{q'}^2$$

Αναλόγως ότι ισχύει για το $A_i^{>k}[q]$, ισχύει και για το $v_i^{>k}[q]$.

Αρχικά αποδεικνύεται ότι η διακύμανση του $h_i[q]s_i[q]$ οριοθετείται από το $v_i[q]$.

Στην συνέχεια από λήμμα 2 αποδεικνύεται ότι $E[v_i^{>k}[q]] = (\sum_{q'=k+1}^m n_{q'}^2)$

Ακόμη αποδεικνύεται ότι

για $t = \Theta(\log(\frac{n}{\delta}))$, με πιθανότητα $(1 - \delta)$ και για όλα τα $l \in [1, n]$, ισχύει

$$|\text{median}\{h_i[q]s_i[q]\} - n_q(l)| \leq 8\gamma.$$

Τέλος, οι Charikar, Chan και Farach-Colton απέδειξαν ότι για $b \geq 8 \max(k, 32 \frac{\sum_{q'=k+1}^m n_{q'}^2}{(\epsilon n_k)^2})$, τα εκτιμώμενα κορυφαία στοιχεία k προκύπτουν τουλάχιστον $(1 - \epsilon)n_k$ φορές στην ακολουθία και επιπλέον όλα τα στοιχεία με συχνότητες τουλάχιστον $(1 + \epsilon)n_k$ εμφανίζονται μεταξύ των εκτιμώμενων κορυφαίων στοιχείων k .

3.4 Παράδειγμα του Count-Sketch

Ένα χρήσιμο παράδειγμα λειτουργίας του αλγορίθμου σύμφωνα με μια δημοσίευση από το πανεπιστήμιό του Ελσίνκι^[7] φαίνεται στις Εικόνες 2Α έως 2Ε. Ο αλγόριθμος προσπαθεί να βρει την διάμεσο από δύο διανύσματα, μέσω ενός πίνακα που το γεμίζουμε σταδιακά τον Initial, έχοντας ως γνωστά τα διανύσματα V1 και V2, όπως επίσης ένα πίνακα κατακερματισμού g(ίδιος με s) και 3 εξισώσεις κατακερματισμού h για κάθε γραμμή.



Example of count-sketch (V1)

1) V1: (1,0,1,2,0), Index (0,1,2,3,4)

Initial

0	0	0
0	0	0
0	0	0

$$\begin{aligned} h1(j) &= j \bmod 3 \\ h2(j) &= (j \bmod 4) \bmod 3 \\ h3(j) &= (2*j) \bmod 3 \end{aligned}$$

Map index domain with h and g functions,

g hash functions

	0	1	2	3	4
g1	+1	-1	-1	+1	-1
g2	-1	+1	-1	+1	+1
g3	+1	-1	+1	-1	+1

2) V1: (1,0,1,2,0), Index (0,1,2,3,4)

Initial

1	0	0
0	0	0
0	0	0

$$\begin{aligned} h1(j) &= j \bmod 3 \\ h2(j) &= (j \bmod 4) \bmod 3 \\ h3(j) &= (2*j) \bmod 3 \end{aligned}$$

g hash functions

	0	1	2	3	4
g1	+1	-1	-1	+1	-1
g2	-1	+1	-1	+1	+1
g3	+1	-1	+1	-1	+1

HELSINGIN YLIOPISTO

Εικόνα 2Α- Στο 1) κομμάτι ξεκινάει με την παρουσίαση του προβλήματος και στο 2) παίρνει το πρώτο στοιχείο από το διάνυσμα V1 και το πρώτο στοιχείο του Index (το οποίο δείχνει στη συνάρτηση κατακερματισμού g) και υπολογίζει την g1 με βάση την εξίσωση της συνάρτησης κατακερματισμού h1 και το τοποθετεί στον πίνακα Initial.



Example of count sketch (V1)

1) V1: (1,0,1,2,0), Index (0,1,2,3,4)

Initial

1	0	0
-1	0	0
0	0	0

$$\begin{aligned} h1(j) &= j \bmod 3 \\ h2(j) &= (j \bmod 4) \bmod 3 \\ h3(j) &= (2*j) \bmod 3 \end{aligned}$$

g hash functions

	0	1	2	3	4
g1	+1	-1	-1	+1	-1
g2	-1	+1	-1	+1	+1
g3	+1	-1	+1	-1	+1

2) V1: (1,0,1,2,0), Index (0,1,2,3,4)

Initial

1	0	0
-1	0	0
1	0	0

$$\begin{aligned} h1(j) &= j \bmod 3 \\ h2(j) &= (j \bmod 4) \bmod 3 \\ h3(j) &= (2*j) \bmod 3 \end{aligned}$$

g hash functions

	0	1	2	3	4
g1	+1	-1	-1	+1	-1
g2	-1	+1	-1	+1	+1
g3	+1	-1	+1	-1	+1

Εικόνα 2Β- Στο 1) και 2) κομμάτια συνεχίζει την διαδικασία γέμισεις του πίνακα Initial πέρνωντας για κάθε γραμμή την εκάστοτε εξίσωση της h.



Example of count sketch (V1)

V1: (1,0,1,2,0), Index (0,1,2,3,4)

Initial

1	0	-1
-1	0	-1
1	1	0

$$\begin{aligned}h1(j) &= j \bmod 3 \\h2(j) &= (j \bmod 4) \bmod 3 \\h3(j) &= (2*j) \bmod 3\end{aligned}$$

g hash functions

	0	1	2	3	4
g1	+1	-1	-1	+1	-1
g2	-1	+1	-1	+1	+1
g3	+1	-1	+1	-1	+1

V1: (1,0,1,2,0), Index (0,1,2,3,4)

Initial

3	0	-1
1	0	-1
-1	1	0

$$\begin{aligned}h1(j) &= j \bmod 3 \\h2(j) &= (j \bmod 4) \bmod 3 \\h3(j) &= (2*j) \bmod 3\end{aligned}$$

g hash functions

	0	1	2	3	4
g1	+1	-1	-1	+1	-1
g2	-1	+1	-1	+1	+1
g3	+1	-1	+1	-1	+1

Εικόνα 2Γ- Στο 1) και 2) κομμάτια συνεχίζεται όπως εικόνα 2Α και 2Β η διαδικασία γεμίσματος του Initial, παρατηρώντας ότι κάθε φορά που συναντάμε στο διάνυσμα τον αριθμό 0 προχωράμε στο επόμενο αριθμό του διανύσματος, καθώς ο αλγόριθμος του Count-Sketch δεν υπολογίζει τα μηδενικά στοιχεία



Example of count sketch (V2)

1) V2: (0,0,2,1,0), Index (0,1,2,3,4)

Initial

0	0	-2
0	0	-2
0	2	0

$$\begin{aligned}h1(j) &= j \bmod 3 \\h2(j) &= (j \bmod 4) \bmod 3 \\h3(j) &= (2*j) \bmod 3\end{aligned}$$

g hash functions

	0	1	2	3	4
g1	+1	-1	-1	+1	-1
g2	-1	+1	-1	+1	+1
g3	+1	-1	+1	-1	+1

2) V2: (0,0,2,1,0), Index (0,1,2,3,4)

Initial

1	0	-2
1	0	-2
-1	2	0

$$\begin{aligned}h1(j) &= j \bmod 3 \\h2(j) &= (j \bmod 4) \bmod 3 \\h3(j) &= (2*j) \bmod 3\end{aligned}$$

g hash functions

	0	1	2	3	4
g1	+1	-1	-1	+1	-1
g2	-1	+1	-1	+1	+1
g3	+1	-1	+1	-1	+1

Εικόνα 2Δ- Στο κομμάτι 1) και 2) Ακολουθούμε την ίδια διαδικασία που είδαμε και για το διάνυσμα V1.



Example of count sketch (result)

V1: (1,0,1,2,0),

V2: (0,0,2,1,0),

3	0	-1
1	0	-1
-1	1	0

1	0	-2
1	0	-2
-1	2	0

$$\begin{aligned} 3 \times 1 + (-1) \times (-2) &= 5 \\ 1 \times 1 + (-1) \times (-2) &= 3 \\ (-1) \times (-1) + 1 \times 2 &= 3 \end{aligned}$$

Median (5,3,3) = 3

The accurate value is
2+2=4.
Underestimation!

Εικόνα 2Ε- Βλέπουμε τους συνολικούς πίνακες Initial που προέκυψαν από τα διανύσματα V1 και V2. Έπειτα πολλαπλασιάζουμε αυτούς τους δύο πίνακες έχοντας ένα αριθμό για κάθε γραμμή που προκύπτει και τέλος παίρνουμε την διάμεσο από αυτούς τους αριθμούς, βρίσκοντας το τελικό αποτέλεσμα.

3.5 Ο αλγόριθμος APPROXTOP(S; k; ε;)

Ο αλγόριθμος APPROXTOP(S;k;ε;) παίρνει σαν είσοδο μια ροή δεδομένων S, έναν σταθερό ακέραιο k και έναν σταθερό πραγματικό αριθμό ε και επιστρέφει, μια λίστα k στοιχείων από την ροή S έτσι ώστε κάθε στοιχείο o_i να ανήκει στη λίστα συχνοτήτων $n_i > (1 - \epsilon) n_k$.

Μια κάπως ισχυρότερη εγγύηση για την έξοδο είναι ότι κάθε στοιχείο με $n_i > (1 + \epsilon) n_k$ θα βρίσκεται στη λίστα εξόδου, όπου ο αλγόριθμός μας, στην πραγματικότητα, θα επιτύχει ισχυρότερη εγγύηση αποτελεσμάτων. Με άλλα λόγια, θα κάνει λάθος μόνο στις οριακές περιπτώσεις. Είναι αλγόριθμος ενός περάσματος, δηλαδή μπορεί μόνο με ένα πέρασμα να βρει με αξιοπιστία τα πιο συχνά αντικείμενα, όπως και ο Count-Sketch.

Θα μπορούσαμε να χρησιμοποιήσουμε και τον αλγόριθμο CANDIDATETOP(S;k;l;), όπου S μια ροή δεδομένων και k,l σταθεροί ακέραιοι αριθμοί, αλλά είναι πολύ δύσκολο να λυθεί, για αυτό καταφεύγουμε στον αλγόριθμο APPROXTOP(S;k;ε;).

3.6.1 Καθολικός κατακερματισμός

Έστω μια πεπερασμένη συλλογή H συναρτήσεων κατακερματισμού που αντιστοιχεί το σύνολο U των κλειδιών στο σύνολο $\{0, 1, \dots, m-1\}$, όπου m το μέγεθος του πίνακα κατακερματισμού.

Η H είναι καθολική εάν για κάθε ζεύγος κλειδιών k, l ∈ U, όπου $k \neq l$, ο αριθμός των συναρτήσεων κατακερματισμού $h \in H$ για τις οποίες $h(k) = h(l)$ είναι μικρότερος ή ίσος με $|H|/m$ (αυτό είναι το μέγεθος H διαιρούμενο με m). Με άλλα λόγια, με μια συνάρτηση κατακερματισμού h που επιλέγεται

τυχαία από το H , η πιθανότητα σύγκρουσης μεταξύ δύο διαφορετικών κλειδιών δεν υπερβαίνει το $1/m$, έτσι η πιθανότητα σύγκρουσης κατά την επιλογή δύο αντικειμένων είναι τυχαία και ανεξάρτητη.

Ο τύπος για να βρούμε τον καθολικό πίνακα H συναρτήσεων κατακερματισμού σύμφωνα με την Διπλωματική εργασία και χρησιμοποιήθηκε για την υλοποίηση του αλγορίθμου Count-Sketch, για τις συναρτήσεις κατακερματισμού h είναι:

$H = \{h_{ab}(k): h_{ab}(k) = ((ak + b) \bmod p) \bmod m\}$, όπου $a \in \{1, 2, \dots, p-1\}$ και $b \in \{0, 1, \dots, p-1\}$, όπου a, b επιλέγονται ομοιόμορφα τυχαία και το p είναι πρώτος αριθμός μεγαλύτερος από το μέγεθος m του πίνακα κατακερματισμού.^[9]

3.6.2 2-Καθολική Διασπορά

Έστω H μια πεπερασμένη συλλογή συναρτήσεων διασποράς $U \rightarrow K$, $U > K = m$, όπου U είναι ένα σύνολο $\{0, 1, \dots, m-1\}$, K ένα αυθαίρετο υποσύνολο του U με $|K|=n \leq m$, με m και n όπως ορίστηκαν στην αρχή.

Η συλλογή H είναι 2-καθολική (ή 2-ανεξάρτητη) αν για 2 διαφορετικά κλειδιά x_1, x_2 και 2 τιμές a_1, a_2 (όχι απαραίτητα διαφορετικές) ισχύει:

$\Pr_{h \in H} [h(x_1) = a_1 \wedge h(x_2) = a_2] \leq \frac{1}{m^2}$, όπου h μια τυχαία επιλεγμένη συνάρτηση της H

Για να είναι όμως η συνάρτηση γενικά k - ανεξάρτητη θα πρέπει:

- Αν η H είναι k -καθολική τότε είναι και $(k - 1)$ -καθολική, και
- για κάθε $x \in U$ και $a \in K$ ισχύει $\Pr_{h \in H} [h(x)=a] = \frac{1}{m}$.

Αποδεικνύεται ότι η οικογένεια συναρτήσεων κατακερματισμού $H = \{h_{ab}(k): h_{ab}(k) = ((ak + b) \bmod p) \bmod m\}$, που αναφέραμε παραπάνω, είναι και 2-καθολική.

3.7 Αλγόριθμος Brute-Force

Ο Αλγόριθμος Brute-Force είναι ένας απλοϊκός αλγόριθμος, που για την συγκεκριμένη εργασία το χρησιμοποιούμε για να διαβάσει ένα-ένα όλα τα αντικείμενα από το αρχείο και να βρίσκει την συχνότητά τους, όπως επίσης να εμφανίζει και τα 10 κορυφαία αντικείμενα εξ-αυτών με την συχνότητά τους. Όταν έχουμε δύο αρχεία διαβάσει όλα τα αντικείμενα, δηλαδή κάθε φορά που βλέπει κάποιο ίδιο αντικείμενο που υπάρχει και στα δύο αρχεία το μειώνει κατά ένα μέχρι να βρει όλα τα αντικείμενα μαζί με την τελική συχνότητα που έχουν(μετά την αλλαγή που έγινε), εμφανίζοντας στο τέλος τα 10 κορυφαία αντικείμενα εξ-αυτών με την συχνότητά τους, με αποτέλεσμα να συγκρίνουμε τα αποτελέσματα με τον αλγόριθμο της Count-Sketch.

Παρατηρούμε ότι ο αλγόριθμος Brute-Force απαιτεί χώρο $O(n)$.

ΥΛΟΠΟΙΗΣΗ

Σε αυτό το Κεφάλαιο παρουσιάζεται η υλοποίηση της δομής Count Sketch, όπως ακόμα και του αλγορίθμου Brute Force. Συγκεκριμένα θα αναλύσουμε τις κλάσεις που χρησιμοποιήθηκαν καθώς επίσης και τις συναρτήσεις που αποτελούνται. Ακόμη περιγράφονται και οι βιβλιοθήκες που χρησιμοποιήθηκαν, για την απαίτηση του κώδικα. Η γλώσσα προγραμματισμού που χρησιμοποιείτε είναι η Java.

4.1 CountSketch.java

Η Κλάση CountSketch είναι μια από τις βασικότερες κλάσεις του προγράμματός που δημιουργήθηκε για την Διπλωματική εργασία που μελετάμε, καθώς σε αυτή την κλάση έχουμε την δημιουργία του αλγορίθμου του Count-Sketch.

```
private static HashMap<Integer, Integer> hsbrut = new HashMap<Integer, Integer>();
private static int t; // row
private static int b; // coll
private int [] A; // A random hash
private int [] B; // B random hash
private static int [] h; // h table hash
private static int [] s; // s table hash
private float delta = 0.3f;
private float epsilon = 0.2f;
private int k = 5; // ta top k antikeimena
private PriorityQueue<Pair > pq= new
PriorityQueue<>(k,Comparator.comparing(Pair::getValue)); // taksinomw to heap me
bash to estimate san Value
```

Επισκόπηση κώδικα 1- Αρχικοποίηση Αντικειμένων

```
For each q over S1, where we perform the following step:
For each q for i ∈ [1; t]; hi[q] -= si[q].
Next, we make a pass over S2, doing the following:
For each q for i ∈ [1; t]; hi[q] += si[q].
```

Επισκόπηση κώδικα 2- Σύγκριση μεταξύ δύο ροών (στη πρότυπη δημοσίευση της
Διπλωματικής μας εργασίας)

```

class ValueComparator implements Comparator<Integer> {
    Map<Integer, Integer> base;
    public ValueComparator(Map<Integer, Integer> base) {
        this.base = base;
    }

    public int compare(Integer a, Integer b) {
        if (base.get(a) >= base.get(b)){
            return -1;
        } else {
            return 1;
        }
    }
}

```

Επισκόπηση κώδικα 3- κλάση κληρονομικότητας ταξινομώντας τον Χάρτη Διεπαφής(HashMap).

Για την δημιουργία της συγκεκριμένης κλάσης, αρχικοποιούμε τα αντικείμενα (βλ. Επισκόπηση κώδικα 1) και στην συνέχεια τα κάνουμε constructor στην συνάρτηση public CountSketch (για τις στήλες b) και public void computeT(String fileToRead) (για τις γραμμές t) .

Οι συναρτήσεις που χρησιμοποιήσαμε φαίνονται παρακάτω:

- **public int findNFromFile(String fileToRead):** η συνάρτηση αυτή παίρνει σαν όρισμα ένα αρχείο, διαβάζει την πρώτη γραμμή του αρχείου χωρισμένο με κενά και επιστρέφει τον ακέραιο που βρίσκεται στην δεύτερη θέση, που είναι το n, δηλαδή το σύνολο των αντικείμενων.
- **public int findPrime():** η οποία βρίσκει τον πρώτο αριθμό, μέσα από το κάλεσμα της κλάσης primalityTest (η οποία αναλύεται στο Κεφάλαιο 4.2) και τον επιστρέφει.
- **long hashfuction_h(int k,long p,int A, int B):** η συνάρτηση έχει ως παραμέτρους σαν k μία σταθερά με τα πιο σημαντικά στοιχεία, σαν p, τον πρώτο αριθμό που βρίσκεται από την συνάρτηση findPrime() και σαν A και B δύο τυχαίοι πίνακες, επιστρέφοντας τον πίνακα καθολικού κατακερματισμού για το h[] (αναφέρθηκε στο Κεφάλαιο 3 ενότητα 3.6.1).
- **long hashfuction_s(int k,long p,int A, int B):** η συνάρτηση έχει σαν παραμέτρους ίδιους με την hashfuction_h και επιστρέφει πίνακα καθολικού κατακερματισμού 2- ανεξαρτησίας, s[] με τυχαίες τιμές {-1,+1} (αναφέρθηκε στο Κεφάλαιο 3 ενότητα 3.6.2).
- **int medianCalnew():** η συνάρτηση αυτή μας επιστρέφει έναν ταξινομημένο πίνακα, με την διάμεσο πολλαπλασιάζοντας τους πίνακες κατακερματισμού h[] με s[] (βλ. Επισκόπηση κώδικα 2).

- boolean updateHeap(Pair pair):** η συνάρτηση αυτή μας αφαιρεί όλα τα στοιχεία από την ουρά(heap) μέχρι να βρεθεί το στοιχείο που αναζητούμε. Μόλις βρεθεί προσθέτει την τιμή του κατά 1 και ανατοποθετεί τα υπόλοιπα στοιχεία στις αρχικές τους θέσεις. Η ουρά είναι τύπου πίνακας λίστας (ArrayList) priority queue, η οποία καλεί την κλάση Pair.java (η οποία αναλύεται στο Κεφάλαιο 4.3).
- void checkheap(int q):** η συγκεκριμένη συνάρτηση παίρνει ως παράμετρο ένα αντικείμενο q. Αρχικά καλούμε την συνάρτηση medianCalnew() με σκοπό να βρεθεί ο μετρητής εκτίμησης (count estimate) και έπειτα ελέγχει αν η ουρά προτεραιότητας (priority queue) είναι κενή.
 Αν είναι τότε προσθέτει στην ουρά το ζευγάρι (τιμή και μέγεθος), αλλιώς αν δεν είναι κενή προσθέτει το στοιχείο κατά 1, συγκρίνοντας την εκτίμηση του μεγέθους(value) με το στοιχείο που πήρε από την ουρά και το διαγράφει ταυτόχρονα από την λίστα.
 Ακόμη αν η απόλυτη τιμή της εκτίμησης είναι μεγαλύτερη από το στοιχείο της κεφαλής τότε απλώς το προσθέτει στη λίστα(priority queue) αλλιώς το επιστρέφει στην αρχική θέση που ήταν από την αρχή το στοιχείο στην ούρα προτεραιότητας.
- public void fillH(int q, int A,int B, long p,String action):** η συνάρτηση αυτή παίρνει σαν παράμετρο ως q ένα αντικείμενο (από το Stream), δύο τυχαίους πίνακες A και B, ως p έναν πρώτο αριθμό και μια εντολή action που είναι για πρόσθεση και αφαίρεση, διότι μπορεί να χρειαστεί να έχουμε σύγκριση 2 ροών. Εδώ κάνουμε εφαρμογή για ένα πρόβλημα της Διπλωματικής εργασίας που μελετάμε, των αντικειμένων που αλλάζουν πιο συχνά (όπως αναφέρθηκε στο κεφάλαιο 2 ενότητα 2.3).
 Έτσι σύμφωνα με τον Επισκόπηση κώδικα 3 προσθέτω-αφαιρούμαι το αντικείμενο q στις δύο συναρτήσεις κατακερματισμού hashfuction_h και hashfuction_s για κάθε ροή ξεχωριστά. Για την ροή S1 αφαιρώ το q ορίζοντας ως action=sub και αντίστοιχα για την ροή S2 προσθέτω το q ορίζοντας ως action=add (λειτουργεί και αντίστροφα).
- public void fillCFromFile(String Stream,String action):** η συνάρτηση αυτή παίρνει σαν όρισμα ένα αρχείο ροής και την ενέργεια που κάνει το εκάστοτε Stream όπως στη συνάρτηση fillH.
 Αρχικά διαβάζουμε το αρχείο το οποίο στην πρώτη γραμμή έχουμε το m και το n, όπου σαν m ορίζουμε το μέγεθος των αντικειμένων και σαν n όλο το πλήθος των αντικειμένων, δηλαδή τις γραμμές που έχει το αρχείο και οι επόμενες γραμμές περιέχουν τα ακέραια αντικείμενα γραμμή-γραμμή.
 Είναι η βασική μας συνάρτηση καθώς εδώ καλούμε όλες τις υπόλοιπες συναρτήσεις(fillH και check-heap) ώστε να εκτελεστεί ο αλγόριθμος του Count-Sketch και να μας εμφανίσει τα 3 αντικείμενα που πήρε από την ουρά μαζί με την εκτίμησή τους.
- public static void brute force(String Stream, String action):** η συνάρτηση αυτή εκτελεί τον απλοϊκό αλγόριθμό της Brute-Force (αναφέρθηκε Κεφάλαιο 3 ενότητα 3.7), παίρνοντας ένα -ένα αντικείμενα από το αρχείο και βρίσκει την συχνότητά τους. Την brute force την χρησιμοποιούμε για να ελέγξουμε προσεγγίστηκα την ορθότητα των αποτελεσμάτων που προκύπτουν από τον αλγόριθμο Count-Sketch(συνάρτηση void fillCFromFile).
 Η συνάρτηση αυτή μας εμφανίζει, όταν έχουμε σαν είσοδο μόνο μια ροή, όλα τα αντικείμενα που υπάρχουν μέσα μαζί με την συχνότητά τους. Αν έχουμε όμως, δύο ροές τότε ελέγχει αν υπάρχει κάποιο αντικείμενο που να βρίσκεται στο S1 και στο S2 αν υπάρχει το μειώνει κατά 1 και το

τοποθετεί στη λίστα(η οποία είναι HashMap),, ενώ αν δει αντικείμενο μόνο στο ένα αρχείο το αφήνει όπως είναι στην ουρά.

Ακόμη εμφανίζει και τις εναλλαγές μεταξύ των δύο αρχείων.

- **void getfinalist():** η συνάρτηση αυτή μας επιστρέφει τα 10 πιο σημαντικά αντικείμενα που βρίσκονται μέσα στη ροή τα οποία είναι σε μια λίστα (HashMap). Η ίδια διαδικασία ισχύει και όταν έχουμε 2 ροές.

Αρχικά χρησιμοποιούμε ένα χάρτη treemap, όπου αποθηκεύεται, όπως κάθε άλλος ταξινομημένος χάρτης (βλ. Επισκόπηση κώδικα 3) ανεξάρτητα από τους ρητούς συγκριτές. Η εφαρμογή treemap δεν συγχρονίζεται, με την έννοια ότι εάν ένας χάρτης έχει πρόσβαση σε πολλά νήματα, ταυτόχρονα και τουλάχιστον ένα από τα νήματα τροποποιεί δομικά τον χάρτη, πρέπει να συγχρονιστεί εξωτερικά.

Ακόμη εάν οι τιμές των 2 κλειδιών είναι ίδιες, ο συγκριτής λαμβάνει υπόψη μόνο την 1η τιμή και αγνοεί τη δεύτερη (στην υλοποίηση κρατάει το μεγαλύτερο). Ακόμη καλεί και την συνάρτηση int GoodQuality.

- **int GoodQuality(int ni, int nk):** η συνάρτηση αυτή εκτελεί τον αλγόριθμο APPROX-TOP(αναφέρθηκε παραπάνω Κεφάλαιο 3 ενότητα 35), δηλαδή για το πόσο καλή είναι η λύση για τις διάφορες τιμές των παραμέτρων ϵ και δ . Παίρνει σαν είσοδο το μέγεθος και την συχνότητα του αντικειμένου και επιστρέφει αν η ποιότητα της λύσης είναι καλή True, ειδάλλως αν όχι False.

4.2 primalityTest.java

Η συγκεκριμένη κλάση είναι από μία δημοσίευση^[8] και είναι μία δημιουργία του Gary L. Miller και του Michael O. Rabin, με σκοπό την εισαγωγή ενός ακέрайου αριθμού και την επιστροφή αν αυτός ο αριθμός είναι πρώτος ή όχι. Η δομή του αλγορίθμου που χρησιμοποιήσαμε φαίνεται στον Αλγόριθμο 2 παρακάτω.

```
Input #1:  $n > 3$ , an odd integer to be tested for primality
Input #2:  $k$ , the number of rounds of testing to perform
Output: "composite" if  $n$  is found to be composite, "probably prime" otherwise
write  $n$  as  $2^r \cdot d + 1$  with  $d$  odd (by factoring out powers of 2 from  $n - 1$ )
WitnessLoop: repeat  $k$  times:
    pick a random integer  $a$  in the range  $[2, n - 2]$ 
     $x \leftarrow a^d \bmod n$ 
    if  $x = 1$  or  $x = n - 1$  then
        continue WitnessLoop
    repeat  $r - 1$  times:
         $x \leftarrow x^2 \bmod n$ 
        if  $x = n - 1$  then
            continue WitnessLoop
return "composite" return "probably prime"
```

4.3 Pair.java

Η λειτουργία της συγκεκριμένης κλάσης είναι να κρατάει τα ζευγάρια, δηλαδή τον αριθμό που διαβάζουμε σαν Key(κλειδί) και το estimate count σαν μέγεθος (Value).

Αποτελείται από τρεις συναρτήσεις την

- public int getKey(): η οποία μας επιστρέφει το κλειδί
- public int getValue(): οποία μας επιστρέφει το μέγεθος και την
- Public void addTovalue(): η οποία αυξάνει το μέγεθος κατά 1 μονάδα

4.4 differenttxt.java

Η differenttxt.java, δημιουργήθηκε με σκοπό την μετατροπή αρχείου, για αρχεία που δεν είναι συμβατά με βάση το προκαθορισμένο τύπο αρχείου που ζητάμε, για την σωστή εκτέλεση του κώδικα. Πιο συγκεκριμένα θέλουμε να δημιουργήσουμε ένα αρχείο, όπου στην πρώτη γραμμή να γράφει το μέγιστο κατά μία μονάδα αριθμό από τα αντικείμενα που υπάρχουν και τον συνολικό αριθμό γραμμών που υπάρχουν στο αρχείο και οι επόμενες γραμμές να περιέχουν μόνο ένα αντικείμενο. Οπότε η βασική λειτουργία της είναι ότι εισάγουμε οποιοδήποτε αρχείο (πάντα με ακέραιες τιμές) το διαχωρίζουμε από κόμματα(αν περιέχουν) και μας δημιουργεί ένα νέο χωρίζοντας τα αντικείμενα ανά γραμμή, εμφανίζοντάς μας τον μέγιστο αριθμό κατά μία μονάδα από αυτούς, όπως επίσης και τον μέγιστο αριθμό γραμμών από το νέο αρχείο που δημιουργήθηκε, γράφοντάς στα στην πρώτη γραμμή του νέου εκτελέσιμου αρχείου.

4.5 Main.java

Είναι η βασική κλάση για την εκτέλεση του κώδικα. Στην συγκεκριμένη κλάση καλούμε τις κλάσεις CountSketch.java και differenttxt.java

```
CountSketch ck= new CountSketch();  
ck.fillCFromFile("kosarak.txt ","add");  
ck.bruteforce("kosarak.txt","add");
```

Επισκόπηση κώδικα 4- Το κάλεσμα της CountSketch.java για ένα αρχείο

Στην Επισκόπηση κώδικα 4 βλέπουμε πώς εκτελείτε ο κώδικας, ζητώντας μόνο τις δύο βασικές συναρτήσεις από την CountSketch.java, την fillCFromFile και την brute force, με παραμέτρους ένα

αρχείο και μία ενέργεια, όπου η fillCFromFile μας εμφανίζει το αντικείμενο στην ουρά μαζί με το count-estimate που έχει και αντίστοιχα η brute force μας εμφανίζει τα δέκα πιο κορυφαία αντικείμενα μαζί με την συχνότητα που υπάρχουν στο αρχείο.

```
CountSketch ck= new CountSketch();
ck.fillCFromFile("kosarak.txt ","sub");
ck.fillCFromFile("SUSY.txt ","add");
ck.bruteforce("kosarak.txt","sub");
ck.bruteforce("SUSY.txt","add");
```

Επισκόπηση κώδικα 5- Το κάλεσμα της CountSketch.java για δύο αρχεία

Σε περίπτωση που τρέχουμε δύο αρχεία (βλ. Επισκόπηση κώδικα 5) η fillCFromFile μας εμφανίζει τρία αντικείμενα που βρίσκονται στην ουρά μαζί με τα count-estimate που έχουν, δύο φορές μια για το πρώτο αρχείο και μια για το δεύτερο και η brute force μας εμφανίζει τα 10 κορυφαία αντικείμενα μαζί με τις συχνότητές τους μετά την εναλλαγή των στοιχείων που έγινε στα δύο αρχεία.

```
differenttxt dif=new differenttxt();
dif.createFile();
```

Επισκόπηση κώδικα 6- Το κάλεσμα της κλάσης differenttxt.java

Η Επισκόπηση κώδικα 6 μας δείχνει το κάλεσμα της κλάσης differenttxt.java, δημιουργώντας μας ένα νέο αρχείο, μόνο όταν έχουμε αρχείο που δεν ταιριάζει με το βασικό αρχείο που επιθυμούμε.

4.6 Βιβλιοθήκες της Java

Η γλώσσα της Java είναι μια γλώσσα που βασίζεται στις βιβλιοθήκες της μιας και ακόμα και η πιο απλή συνάρτηση θα περιέχεται σε μία από αυτές.

Παρακάτω θα δούμε τις βιβλιοθήκες που χρησιμοποιήσαμε για τον κώδικα:

- **java.util.***- Αυτή η βιβλιοθήκη περιλαμβάνει όλες τις κλάσεις καθώς και όλες τις συναρτήσεις εισόδου ή εξόδου που περιέχονται. Είναι μία από τις βασικές βιβλιοθήκες της Java.
- **java.io.***- Η συγκεκριμένη βιβλιοθήκη εισάγει τη λειτουργία εισόδου και εξόδου, και περισσότερο χρησιμοποιείται σε λειτουργία αρχείων, όπως άνοιγμα, κλείσιμο, αποθήκευση κλπ. Όπως η java.util.* έτσι και αυτή είναι μία από τις βασικές βιβλιοθήκες της Java.
- **java.util.HashMap**- Η συγκεκριμένη βιβλιοθήκη περιλαμβάνει τη διεπαφή Χάρτη(HashMap), η οποία μας επιτρέπει να αποθηκεύσουμε ζεύγος κλειδιών και τιμών, όπου κάθε κλειδί πρέπει να είναι μοναδικό. Έχουμε την δυνατότητα επίσης την αποθήκευση των μηδενικών στοιχείων, όμως για να γίνει αυτό, θα πρέπει να υπάρχει απαραίτητα μόνο ένα μηδενικό κλειδί.

- **java.util.Random-** Η βιβλιοθήκη χρησιμοποιείται για τη δημιουργία/παραγωγή τυχαίων αριθμών. Επίσης παρέχει διάφορες κλήσεις μεθόδων για τη δημιουργία διαφορετικών τύπων τυχαίων δεδομένων, όπως float, double, int.
- **java.util.Scanner-** Η Scanner χρησιμοποιείται όταν ο χρήστης επιθυμεί να δώσει κάτι σαν είσοδο. Είναι και αυτή μία από τις βασικές βιβλιοθήκες της προγραμματιστικής γλώσσας Java.
- **java.io.File-** Η βιβλιοθήκη αυτή περιέχει διάφορες μεθόδους για το όνομα διαδρομής του αρχείου, τη διαγραφή και τη μετονομασία αρχείων. Επίσης τη δημιουργία νέων καταλόγων, την καταχώριση των περιεχομένων ενός καταλόγου και τον προσδιορισμό πολλών κοινών χαρακτηριστικών αρχείων και καταλόγων.
- **java.io.FileInputStream-** Χρησιμοποιείται για την ανάγνωση δεδομένων προσανατολισμένων σε byte (όπως επίσης και ροές ακατέργαστων byte), όπως δεδομένα εικόνας, ήχος, βίντεο κ.λπ. Μπορούμε επίσης να διαβάσουμε δεδομένα ροής χαρακτήρων.
- **java.io.FileNotFoundException-** Είναι μία χρήσιμη βιβλιοθήκη για να μας ειδοποιήσει αν κάποιο αρχείο δεν υπάρχει ή έχει κάποιο σφάλμα, όπως μορφή κ.α.
- **java.io.FileWriter-** Η FileWriter είναι μια βιβλιοθήκη διευκόλυνσης της Java για την εγγραφή δεδομένων κειμένου σε αρχεία.
- **java.io.BufferedWriter-** Χρησιμοποιείται για την προσωρινή αποθήκευση ενός εγγράψιμου αρχείου μέσα από χαρακτήρες προσωρινής αποθήκευσης, έχοντας έτσι αποτελεσματικότερη γραφή μεμονωμένων συστοιχιών , χαρακτήρων και συμβολοσειρών.
- **java.util.Map.Entry-** Αύτη η βιβλιοθήκη χρησιμοποιείται για την επιστροφή δεδομένων (ζευγαριών) μέσα από ενός χάρτη διεπαφής(HashMap), όπου είναι έγκυρα μόνο όταν βρίσκονται μέσα σε μία οποιαδήποτε μορφής επανάληψης.
- **org.apache.commons.lang3.ArrayUtils-** Η βιβλιοθήκη αυτή χρησιμεύει για να δηλώσουμε ακέραιους πίνακες (int και Integer), προσπαθώντας αυτοί οι πίνακες να έχουν μηδενική είσοδο. Δίνοντας έτσι την ευκαιρία να μην γίνει καμία εξαίρεση για μηδενική είσοδο πίνακα. Ωστόσο, ένας πίνακας αντικειμένου που περιέχει μόνο ένα μηδενικό στοιχείο μπορεί να εξαιρεθεί.
- **java.math.BigInteger-** Η βιβλιοθήκη BigInteger χρησιμοποιείται για μαθηματική λειτουργία, η οποία περιλαμβάνει πολύ μεγάλους ακέραιους υπολογισμούς που βρίσκονται εκτός του ορίου όλων των διαθέσιμων ακεραίων τύπων δεδομένων. Για παράδειγμα, το factorial των 100 περιέχει 158 ψηφία, οπότε δεν μπορούμε να το αποθηκεύσουμε σε οποιονδήποτε ακέραιο διαθέσιμο τύπο δεδομένων(int ή Integer), αλλά με την βοήθεια αυτής της βιβλιοθήκης, μπορούμε να αποθηκεύσουμε όσο μεγάλο ακέραιο αριθμό θέλουμε.

ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ

Σε αυτό το Κεφάλαιο παρουσιάζεται η πειραματική αξιολόγηση του προγράμματος μας, χρησιμοποιώντας δεδομένα από γνωστές συλλογές. Στις επιμέρους ενότητες του Κεφαλαίου 5 αναλύεται το περιβάλλον εκτέλεσης των πειραμάτων μαζί με όλα τα τεχνητά του στοιχεία, τα δεδομένα που χρησιμοποιήθηκαν για τον έλεγχο της σωστής λειτουργίας του συστήματος και τέλος η παρουσίαση των αποτελεσμάτων που προέκυψαν. Καθώς επίσης και τα συμπεράσματα που προέκυψαν.

5.1 Περιβάλλον εκτέλεσης πειραμάτων

Ο κώδικας που περιεγράφηκε στο Κεφάλαιο 4, εκτελέστηκε σε υπολογιστικό περιβάλλον με λειτουργικό σύστημα Windows και την έκδοση των Windows 10 Home. Ο επεξεργαστής του συστήματος αυτού είναι Intel Core i5 με συνολικό χώρο αποθήκευσης 500 GB ενώ η ταχύτητα του ρολογιού(CPU) του είναι 2,40GHz. Επίσης το υλικό του περιλαμβάνει 6GB μνήμης RAM και ο τύπος συστήματός του είναι x64bit.

Ο κώδικας μεταγλωττίστηκε με την βοήθεια ενός προγράμματος, μεταγλώττισης Java το λεγόμενο Eclipse IDE. Πιο συγκεκριμένα χρησιμοποιήθηκε η έκδοση Eclipse IDE 2020-09, το οποίο μας βοήθησε στην υλοποίηση, όπως επίσης και στην απεικόνιση των αποτελεσμάτων.

Επίσης το κατέβασμα των αρχείων (πειραμάτων) έγινε μέσω διαδικτύου, με ασύρματη σύνδεση, με το διαδίκτυο να έχει μετρημένη ταχύτητα για download περίπου 10,66 Mbps και για upload περίπου 2,45 Mbps (οι μετρήσεις για την ταχύτητα της σύνδεσης με το διαδίκτυο έγιναν με χρήση της πλατφόρμας που βρίσκεται στην ιστοσελίδα speedtest.net).

Συνοπτικά όλα τα στοιχεία παρουσιάζονται στον παρακάτω Πίνακα 1:

Σύστημα	Μέγεθος-Έκδοση
Windows	10 Home
Επεξεργαστής	Intel Core i5
Συνολικός χώρος αποθήκευσης	500GB
CPU	2,40GHz
RAM	6GB
Τύπος συστήματος	x64bit
Eclipse	Eclipse IDE 2020-09
Ταχύτητα Internet	10,66 Mbps Down/2,45 Mbps Up

5.2 Πειραματικά Δεδομένα

Τα πειραματικά Δεδομένα προήλθαν από την ιστοσελίδα (<https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php>), η οποία παρέχει αρχεία κειμένου ως εισόδους, ώστε να μπορέσουμε να εξετάσουμε το πρόβλημα για τα σύνολα δεδομένων για τα πιο συχνά στοιχεία (αναφέρθηκε στο Κεφάλαιο 2 ενότητα 2.2), όπως επίσης και για να συγκρίνουμε τα αντικείμενα που αλλάζουν πιο συχνά. Επίσης η ιστοσελίδα παρέχει μεγάλα σύνολα δεδομένων που μπορούν να χρησιμοποιηθούν για την αξιολόγηση και τη σύγκριση της απόδοσης του Count-Sketch αλγορίθμου.

5.3 Πειραματικά Αποτελέσματα

Ο Κώδικας που υλοποιήθηκε στο Κεφάλαιο 4 για την επίλυση της Διπλωματικής εργασίας, δοκιμάστηκε για διαφορετικούς τύπους αρχείων από την ιστοσελίδα (αναφέρθηκε ενότητα 5.2) και ενδεικτικά θα αναλύσουμε αν τα πιο συχνά αντικείμενα της Count-Sketch (αναφέρθηκε στο Κεφάλαιο 3) μετά την υλοποίησή της (αναφέρθηκε ενότητα 4.1) ταιριάζουν με τα πιο συχνά αντικείμενα της συνάρτησης Brute-Force (υλοποιήθηκε στο Κεφάλαιο 4 ενότητα 4.1).

5.3.1 Αρχικοποίηση του Count-Sketch

Πριν ξεκινήσουμε τα πειράματα θα πρέπει να αρχικοποιήσουμε όλους τους παραμέτρους που χρησιμοποιούμε στην κλάση CountSketch.java.

Αρχικά ορίζουμε σαν παράμετρο την πιθανότητα σφάλματος $\delta=0.3$, σαν παράμετρο την ποιότητα της λύσης $\epsilon=0.2$ και σαν παράμετρο τα κορυφαία αντικείμενα $k=10$.

Ακόμη για τις συναρτήσεις κατακερματισμού h και s , ορίζουμε σαν πρώτο αριθμό $p=83$, τους πίνακες A και B σαν τυχαίους που θα πρέπει να περιέχουν αριθμούς από $\{0,1,\dots,p-1\}$.

Επιπλέον το αρχείο που θα χρησιμοποιήσουμε για να βρούμε το πιο συχνό στοιχείο θα είναι το Stream="mushrooms.txt" με την ιδιότητα action="add" και για την εύρεση των αντικειμένων που εμφανίζουν τις μεγαλύτερες μεταβολές μεταξύ δύο ροών δεδομένων θα χρησιμοποιήσουμε τα αρχεία ως $S1="mushrooms.txt"$ με action="add" και ως $S2="connect.txt"$ με action="sub".

δ	ϵ	k	p	A	B	$S1,action$	$S2,action$
0.3	0.2	10	83	$\{0,1,\dots,82\}$	$\{0,1,\dots,82\}$	"mushrooms.txt","add"	"connect.txt","sub"

Πίνακας 2 -Συνοπτικός πίνακας Αρχικοποιήσεων

5.3.2 Αποτελέσματα για μια ροή.

Όπως αναφερθήκαμε και παραπάνω θα χρησιμοποιήσω μια ροή, δηλαδή θα δώσω σαν είσοδο στο πρόγραμμά μου ένα αρχείο και θα μου εμφανίσει τα 10 πιο συχνά αντικείμενα.

Το αρχείο που θα χρησιμοποιήσω θα είναι το “mushrooms.txt”, το οποίο περιέχει 129 αντικείμενα n (Πλήθος αντικειμένων). και 193568 γραμμές m . Ακόμη θα χρησιμοποιήσω ως παράμετρο πιθανότητα σφάλματος $\delta=0.3f$ και παράμετρο της καλής ποιότητας $\epsilon=0.2f$.

Αναλυτικότερα τα αποτελέσματα που προκύπτουν φαίνονται στις Εικόνες 3Α έως 3Ζ:

```
1 package newdip;
2
3
4 import java.util.*;
5
6
7
8
9
10 public class main {
11
12     public static void main(String[] args) {
13
14         //differenttxt dif=new differenttxt();
15         // dif.createFile();
16
17         // call class CountSketch
18
19         CountSketch ck= new CountSketch();
20         ck.computeT("mushrooms.txt");
21         ck.fillFromFile("mushrooms.txt", "add");
22         ck.bruteforce("mushrooms.txt", "add");
23         ck.getfinallist();
24     }
25 }
26
27
```

Εικόνα 3Α- Εκτέλεση του αρχείου “mushrooms.txt”

```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (20 Οκτ 2021, 5:09:57 π.μ. – 5:10:02 μ.μ.)
emfanise sthles b
80
emfanise grammes t
14

o q poy brisketai sto heap - count estimate
21-1109

o q poy brisketai sto heap - count estimate
39-1663

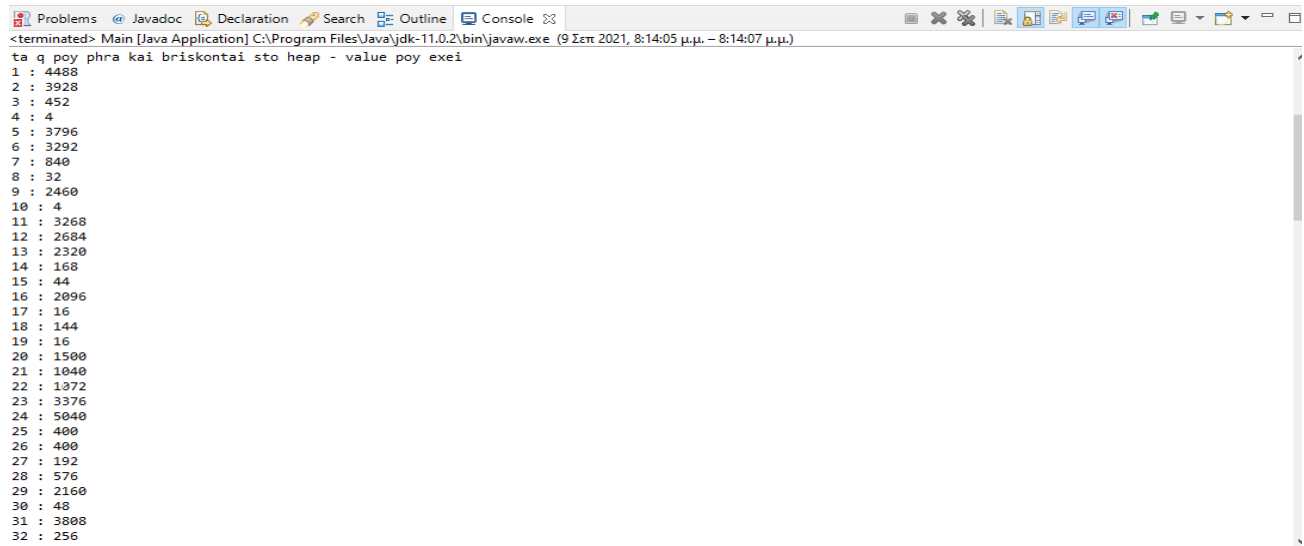
o q poy brisketai sto heap - count estimate
128-3230

o q poy brisketai sto heap - count estimate
71-5147

o q poy brisketai sto heap - count estimate
97-7841

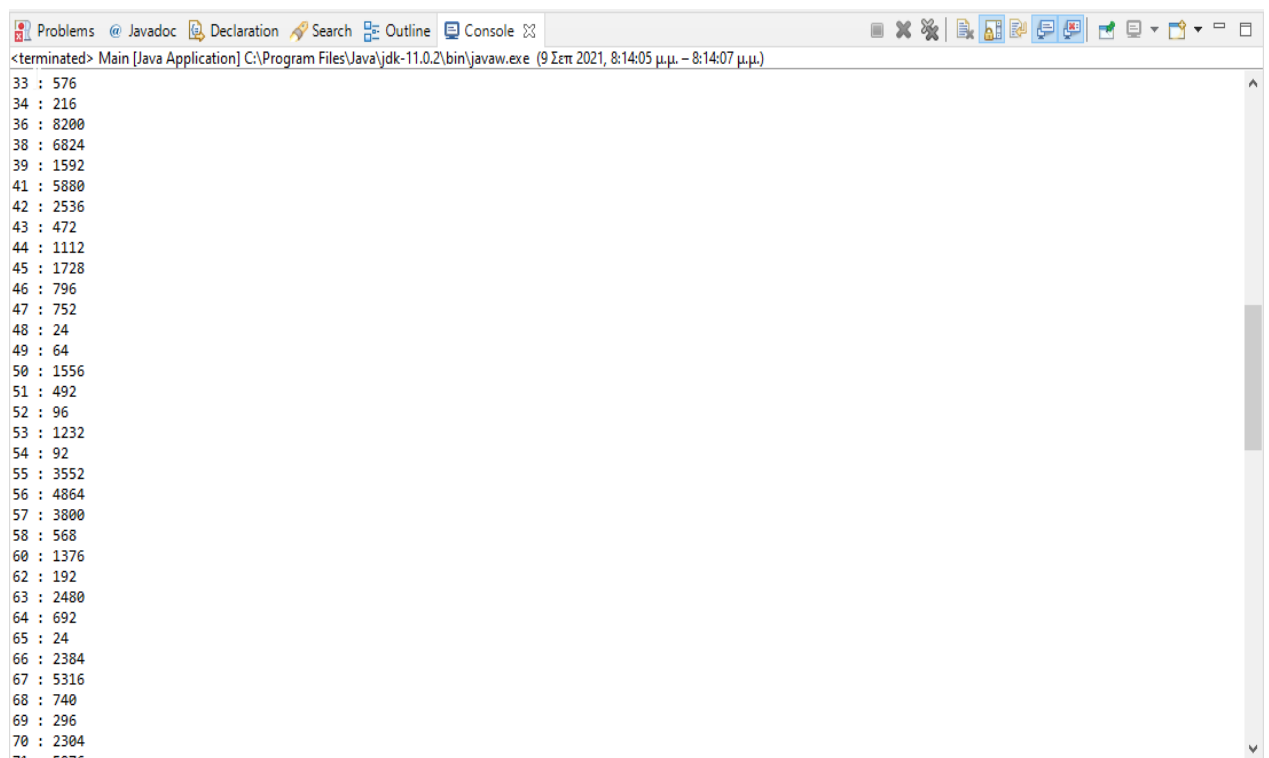
o q poy brisketai sto heap - count estimate
36-8276
```

Εικόνα 3Β- Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch



```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (9 Σεπ 2021, 8:14:05 μ.μ. - 8:14:07 μ.μ.)
ta q poy phra kai briskontai sto heap - value poy exei
1 : 4488
2 : 3928
3 : 452
4 : 4
5 : 3796
6 : 3292
7 : 840
8 : 32
9 : 2460
10 : 4
11 : 3268
12 : 2684
13 : 2320
14 : 168
15 : 44
16 : 2096
17 : 16
18 : 144
19 : 16
20 : 1500
21 : 1040
22 : 1072
23 : 3376
24 : 5040
25 : 400
26 : 400
27 : 192
28 : 576
29 : 2160
30 : 48
31 : 3808
32 : 256
```

Εικόνα 3Γ- Τα αντικείμενα από 1 έως 32 μαζί με συχνότητα που υπάρχουν στο αρχείο(εκτέλεση Brute-Force)



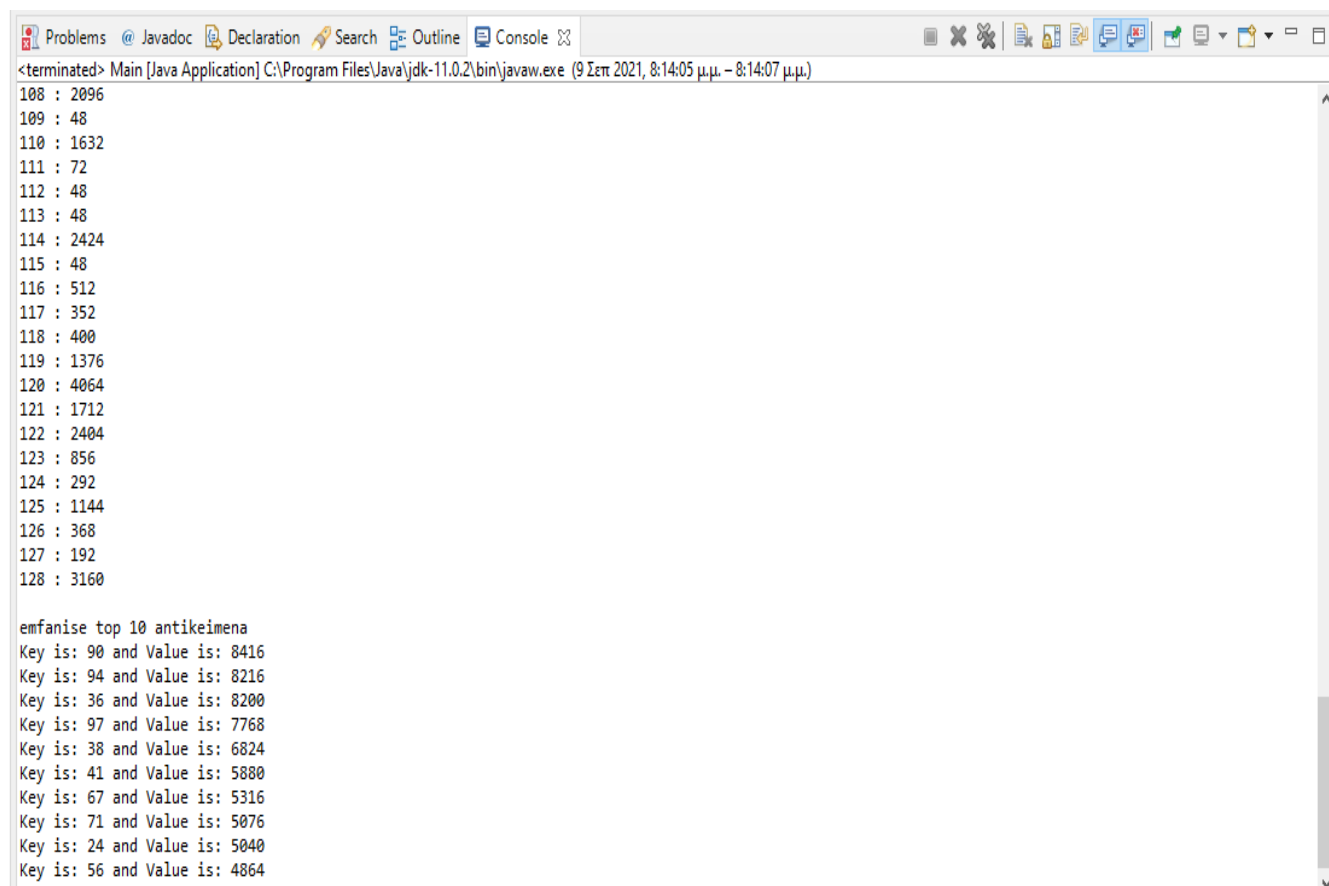
```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (9 Σεπ 2021, 8:14:05 μ.μ. - 8:14:07 μ.μ.)
33 : 576
34 : 216
36 : 8200
38 : 6824
39 : 1592
41 : 5880
42 : 2536
43 : 472
44 : 1112
45 : 1728
46 : 796
47 : 752
48 : 24
49 : 64
50 : 1556
51 : 492
52 : 96
53 : 1232
54 : 92
55 : 3552
56 : 4864
57 : 3800
58 : 568
60 : 1376
62 : 192
63 : 2480
64 : 692
65 : 24
66 : 2384
67 : 5316
68 : 740
69 : 296
70 : 2304
```

Εικόνα 3Δ- Τα αντικείμενα από 33 έως 70 μαζί με συχνότητα που υπάρχουν στο αρχείο (εκτέλεση Brute-Force)



```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (9 Σεπ 2021, 8:14:05 μ.μ. – 8:14:07 μ.μ.)
71 : 5076
72 : 448
73 : 432
74 : 48
75 : 576
76 : 192
77 : 1872
78 : 96
79 : 4744
80 : 8
81 : 536
82 : 432
83 : 48
84 : 576
85 : 192
86 : 1872
87 : 96
88 : 4640
89 : 24
90 : 8416
92 : 96
93 : 96
94 : 8216
95 : 8
96 : 48
97 : 7768
98 : 600
100 : 3056
101 : 48
102 : 1296
103 : 48
104 : 3968
107 : 2000
```

Εικόνα 3Ε- Τα αντικείμενα από 33 έως 107 μαζί με συχνότητα που υπάρχουν στο αρχείο (εκτέλεση Brute-Force)



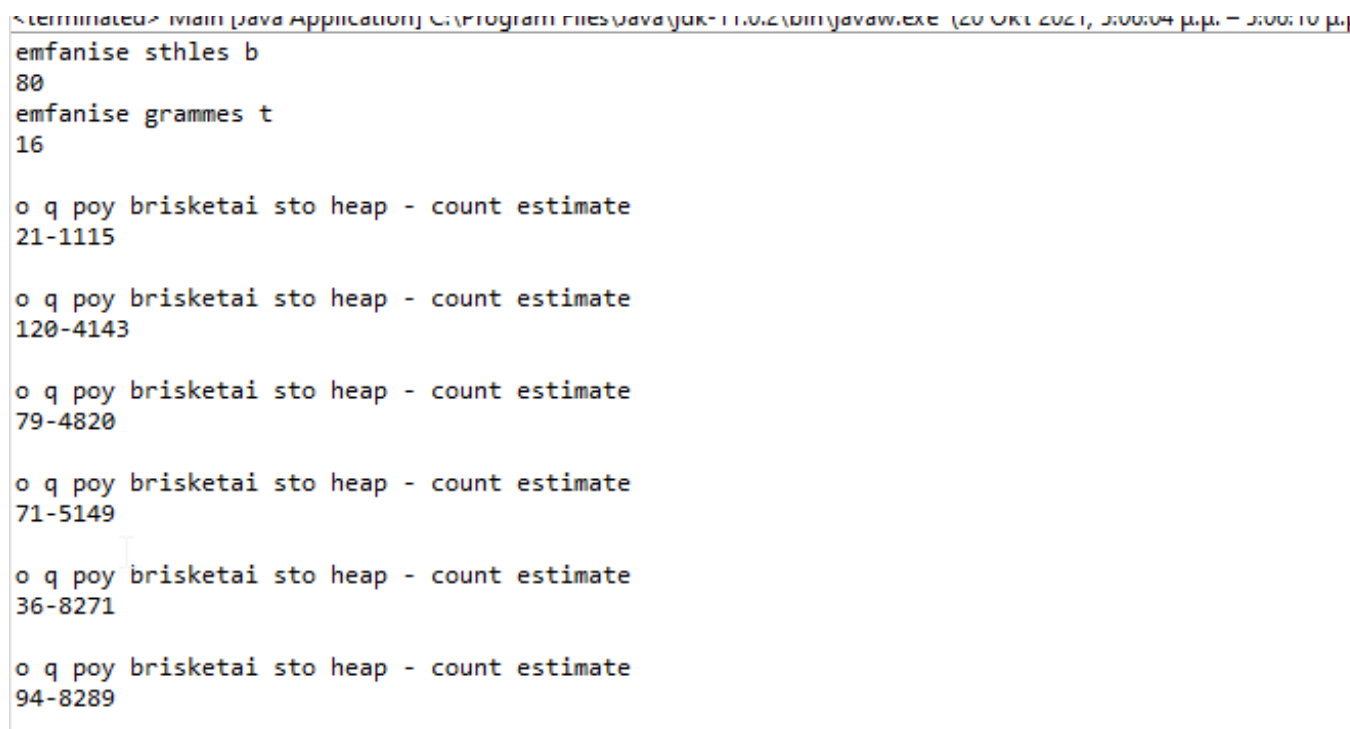
```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (9 Σεπ 2021, 8:14:05 μ.μ. – 8:14:07 μ.μ.)
108 : 2096
109 : 48
110 : 1632
111 : 72
112 : 48
113 : 48
114 : 2424
115 : 48
116 : 512
117 : 352
118 : 400
119 : 1376
120 : 4064
121 : 1712
122 : 2404
123 : 856
124 : 292
125 : 1144
126 : 368
127 : 192
128 : 3160

emfanise top 10 antikeimena
Key is: 90 and Value is: 8416
Key is: 94 and Value is: 8216
Key is: 36 and Value is: 8200
Key is: 97 and Value is: 7768
Key is: 38 and Value is: 6824
Key is: 41 and Value is: 5880
Key is: 67 and Value is: 5316
Key is: 71 and Value is: 5076
Key is: 24 and Value is: 5040
Key is: 56 and Value is: 4864
```

Εικόνα 3Ζ- Τα αντικείμενα από 108 έως 128 μαζί με συχνότητα που υπάρχουν στο αρχείο, καθώς επίσης και τα 10 πιο συχνά (εκτέλεση Brute-Force)

Με τα αποτελέσματα που προκύπτουν από τις εικόνες 3Α έως 3Ζ παρατηρούμε αρχικά στην εικόνα 3Α ότι οι γραμμές είναι $t=14$ και οι στήλες $b=80$. Ακόμη η εικόνα 3Β, εμφανίζει τα 6 πιο συχνά αντικείμενα μαζί με τις εκτιμήσεις τους τρέχοντας την συνάρτηση `void fillCFromFile("mushrooms.txt","add")`, (στην οποία γίνεται η υλοποίηση του αλγορίθμου του Count-Sketch) (αναφέρθηκε Κεφάλαιο 4 ενότητα 4.1). Επίσης χάρις τις αρχικοποιήσεις των ϵ και δ παρατηρούμε ότι τα τρία από τα 6 και πιο συγκεκριμένα ο 71 με εκτίμηση 5149, ο 97 με εκτίμηση 7841 και ο 36 με εκτίμηση 8276, υπάρχουν στα 10 κορυφαία αντικείμενα που προκύπτουν μετά από εκτέλεση της συνάρτησης, `void brute force("mushrooms.txt","add")`, (στην οποία γίνεται η υλοποίηση του αλγορίθμου Brute-Force) (αναφέρθηκε Κεφάλαιο 4 ενότητα 4.1) με συχνότητα 5076 και 7768, 8200 αντίστοιχα.

Άρα καταλήγουμε στο συμπέρασμα ότι τα αποτελέσματα είναι πάρα πολύ κοντά ανάμεσα στους δύο αλγορίθμους τόσο τα αντικείμενα όσο και οι εκτιμήσεις τους.



```

terminated> main java Application C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (20 Οκτ 2021, 3:00:04 μ.μ. - 3:00:10 μ.μ.)
emfanise sthles b
80
emfanise grammes t
16

o q poy brisketai sto heap - count estimate
21-1115

o q poy brisketai sto heap - count estimate
120-4143

o q poy brisketai sto heap - count estimate
79-4820

o q poy brisketai sto heap - count estimate
71-5149

o q poy brisketai sto heap - count estimate
36-8271

o q poy brisketai sto heap - count estimate
94-8289

```

Εικόνα 4- Εκτέλεση του αρχείου “mushrooms.txt”, για $\epsilon=0,03f$ και $\delta=0,03f$

Αν τώρα αλλάξουμε τους παραμέτρους δ και ϵ , με 0.03f και 0,03f αντίστοιχα παρατηρούμε από την εικόνα 4, ότι προκύπτουν παραπάνω αντικείμενα που ταιριάζουν στα 10 πιο συχνά, καταλαβαίνοντας ότι έχουμε ακόμη πιο καλύτερα αποτελέσματα, όπως το αντικείμενο 94 και παρατηρούμε ότι ο 36 έχει πολύ καλύτερη προσεγγιστική εκτίμηση από πριν. Ακόμη παρατηρούμε ότι και οι γραμμές αυξήθηκαν, την πρώτη φορά ήταν $t=14$ και τώρα με αυτές τις εκτελέσεις έχουμε $t=16$. Αυτό συμβαίνει γιατί παίζει ρόλο και η πολύ καλή προσέγγιση αποτελεσμάτων που λαμβάνουμε σε σχέση με πριν.

```

emfanise sthles b
80
emfanise grammes t
13

o q poy brisketai sto heap - count estimate
113-124

o q poy brisketai sto heap - count estimate
53-1308

o q poy brisketai sto heap - count estimate
39-1663

o q poy brisketai sto heap - count estimate
57-3871

o q poy brisketai sto heap - count estimate
79-4819

o q poy brisketai sto heap - count estimate
67-5394

```

Εικόνα 5- Εκτέλεση του αρχείου “mushrooms.txt”, για $\epsilon=0,9f$ και $\delta=0,9f$

Αντίθετα, αν αλλάξουμε τους παραμέτρους δ και ϵ , με $0,9f$ και $0,9f$ αντίστοιχα παρατηρούμε από την εικόνα 5, ότι ένα μόνο από τα 6 αντικείμενα με τις συχνότητες τους, που βρέθηκαν, χρησιμοποιώντας τον αλγόριθμο του Count-Sketch, ανήκει στα 10 κορυφαία αντικείμενα, όπου έχει πολύ κακή εκτίμηση.

Αυτό συμβαίνει γιατί όσο πιο μεγάλες τιμές δίνουμε στους παραμέτρους δ και ϵ τόσο πιο κακή είναι η λύση.

Ακόμη παρατηρούμε ότι και οι γραμμές μειώθηκαν, την πρώτη φορά ήταν $t=14$ και τώρα με αυτές τις εκτελέσεις έχουμε $t=13$. Αυτό συμβαίνει γιατί παίζει ρόλο και η κακή εκτίμηση που λαμβάνουμε.

Τέλος, τα αποτελέσματα από το παραπάνω πείραμα παρουσιάζεται συνοπτικά στον Πίνακα 3

Σύνολο Αποτελεσμάτων								
Σύνολο Δεδομένων	Αντικείμενα n	Πλήθος αντικειμένων m	Παράμετρος πιθανότητας σφάλματος δ	Παράμετρ οποιότητα λύσης ϵ	Γραμμές t	Στήλες b	Αλγόριθμος Count- Sketch (αντικείμεν ο – εκτίμηση)	Αλγόριθμος Brute- Force 10 κορυφαία (αντικείμεν ο – πραγματική τιμή)
Mushrooms.txt	129	193568	0.03f	0.03f	16	80	21-1115	90- 8416
							120-4143	94- 8216
							79-4820	36- 8200
							71-5149	97- 7768
							36-8271	38- 6824
							94-8289	41- 5880
			0.3f	0.2f	14	80	21-1109	67- 5316
							39-1663	71- 5076
							128-3230	24- 5040
							71-5147	56- 4864
							97-7841	
							36-8276	
			0.9f	0.9f	13	80	113-124	
							53-1308	
							39-1663	
							57-3871	
							79-4819	
							67-5394	

Πίνακας 3- Συνοπτικός πίνακας όλων των αποτελεσμάτων για μια ροή.

5.3.3 Αποτελέσματα για δυο ροές.

Σε αυτή την περίπτωση θα έχουμε σαν είσοδο 2 αρχεία και θα συγκρίνουμε τα κορυφαία αντικείμενα που θα προκύψουν μετά από την εναλλαγή των στοιχείων, δηλαδή θα παρατηρήσουμε τα αρχικά 3 αντικείμενα μαζί με τις εκτιμήσεις τους που θα προκύψουν από το πρώτο αρχείο και ακολούθως τα επόμενα 3 αντικείμενα αντίστοιχα σε σχέση με τα 10 κορυφαία αντικείμενα που θα προκύψουν μετά την εναλλαγή των στοιχείων.

Πιο συγκεκριμένα θα χρησιμοποιήσουμε το αρχείο “ mushrooms.txt” (περιγράφεται στην ενότητα 5.3.2) και το αρχείο “connect.txt”, το οποίο περιέχει 130 αντικείμενα n(Πλήθος διαφορετικών αντικειμένων) και 2904951 γραμμές m.

Ακόμη θα ορίσουμε σαν παράμετρο πιθανότητας σφάλματος $\delta=0.3f$ και παράμετρο ποιότητας λύσης $\epsilon=0.2f$

Αναλυτικότερα τα αποτελέσματα που προκύπτουν φαίνονται στις Εικόνες 6Α έως 6Θ:

```
1 package newdip;
2
3
4 import java.util.*;
5 import java.io.*;
6 import java.io.File;
7
8 public class main {
9
10     public static void main(String[] args) {
11
12         //differenttxt dif=new differenttxt();
13         // dif.createFile();
14
15         // call class CountSketch
16
17         CountSketch ck= new CountSketch();
18         ck.computeT("mushrooms.txt");
19         ck.fillFromFile("mushrooms.txt","add");
20         ck.bruteForce("mushrooms.txt","add");
21         ck.computeT("connect.txt");
22         ck.fillFromFile("connect.txt","sub");
23         ck.bruteForce("connect.txt","sub");
24         ck.getfinallist();
25     }
26 }
27
```

Εικόνα 6Α- Εκτέλεση του αρχείου “mushrooms.txt” και του αρχείου “connect.txt”

```
emfanise sthles b
80
emfanise grammes t
14

o q poy brisketai sto heap - count estimate
113-124

o q poy brisketai sto heap - count estimate
12-2754

o q poy brisketai sto heap - count estimate
23-3450

o q poy brisketai sto heap - count estimate
57-3875

o q poy brisketai sto heap - count estimate
36-8270

o q poy brisketai sto heap - count estimate
94-8292
```

Εικόνα 6Β- Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “mushrooms.txt”

```

emfanise grammes t
17

o q poy brisketai sto heap - count estimate
73-297

o q poy brisketai sto heap - count estimate
61-10361

o q poy brisketai sto heap - count estimate
115-48181

o q poy brisketai sto heap - count estimate
82-62427

o q poy brisketai sto heap - count estimate
52-65218

o q poy brisketai sto heap - count estimate
127-67541

```

Εικόνα 6Γ- Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “connect.txt”

```

<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (16 Σεπ 2021, 2:56:05 μ.μ. - 2:56:19 μ.μ.)

ta q poy phra kai briskontai sto heap - value poy exei
1 : 39985
2 : 2521
3 : 16183
4 : 24978
5 : 16528
6 : 18959
7 : 42545
8 : 11992
9 : 9688
10 : 55329
11 : 3007
12 : 3265
13 : 59296
14 : 2922
15 : 2807
16 : 63169
17 : 1139
18 : 993
19 : 67024
20 : 1136
21 : 887
22 : 20700
23 : 16520
24 : 20849
25 : 40780
26 : 12864
27 : 12921
28 : 53776
29 : 4687
30 : 6310
31 : 57398
32 : 3053

```

Εικόνα 6Δ- Τα αντικείμενα από την εναλλαγή στοιχείων από 1 έως 32 μαζί με συχνότητα που υπάρχουν στα δύο αρχεία(εκτέλεση Brute-Force)

```
Problems Javadoc Declaration Outline Console
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (16 Σεπ 2021, 2:56:05 μ.μ. – 2:56:19 μ.μ.)
33 : 2466
34 : 64988
35 : 1212
36 : 7061
37 : 67040
38 : 6462
39 : 1439
40 : 32031
41 : 13341
42 : 13767
43 : 14954
44 : 36559
45 : 12732
46 : 52102
47 : 6967
48 : 6916
49 : 60659
50 : 2016
51 : 2770
52 : 65047
53 : 43
54 : 1047
55 : 63492
56 : 4504
57 : 3647
58 : 18977
59 : 18994
60 : 27640
61 : 10285
62 : 10145
63 : 44453
64 : 4077
65 : 4908
```

Εικόνα 6E- Τα αντικείμενα από την εναλλαγή στοιχείων από 33 έως 65 μαζί με συχνότητα που υπάρχουν στα δύο αρχεία(εκτέλεση Brute-Force)

```
66 : 55472
67 : 2965
68 : 1370
69 : 62800
70 : 1599
71 : 4265
72 : 65593
73 : 210
74 : 42
75 : 66669
76 : 40655
77 : 17202
78 : 7540
79 : 50783
80 : 5533
81 : 5953
82 : 61920
83 : 2368
84 : 2213
85 : 65258
86 : 721
87 : 860
88 : 62288
89 : 335
90 : 8146
91 : 67471
92 : 30
93 : 78
94 : 25951
95 : 18873
96 : 14461
97 : 43401
98 : 7577
```

Εικόνα 6Z- Τα αντικείμενα από την εναλλαγή στοιχείων από 66 έως 98 μαζί με συχνότητα που υπάρχουν στα δύο αρχεία(εκτέλεση Brute-Force)

```
99 : 8209
100 : 57318
101 : 3533
102 : 2306
103 : 64791
104 : 2478
105 : 1226
106 : 66817
107 : 1553
108 : 1805
109 : 67421
110 : 1562
111 : 54
112 : 29681
113 : 18897
114 : 16459
115 : 48056
116 : 9384
117 : 9205
118 : 58469
119 : 3070
120 : 178
121 : 62589
122 : 619
123 : 615
124 : 66418
125 : 611
126 : 54
127 : 67273
128 : 3086
129 : 16
```

Εικόνα 6H- Τα αντικείμενα από την εναλλαγή στοιχείων από 99 έως 129 μαζί με συχνότητα που υπάρχουν στα δύο αρχεία(εκτέλεση Brute-Force)

```
emfanise top 10 antikeimena
Key is: 91 and Value is: 67471
Key is: 109 and Value is: 67421
Key is: 127 and Value is: 67273
Key is: 37 and Value is: 67040
Key is: 19 and Value is: 67024
Key is: 106 and Value is: 66817
Key is: 75 and Value is: 66669
Key is: 124 and Value is: 66418
Key is: 72 and Value is: 65593
Key is: 85 and Value is: 65258
```

Εικόνα 6Θ - Τα 10 πιο συχνά αντικείμενα μετά από τις εναλλαγές των δύο αρχείων (εκτέλεση Brute-Force)

Με τα αποτελέσματα που προκύπτουν από τις εικόνες 6B έως 6Θ παρατηρούμε ότι η εικόνα 6B, εμφανίζει τα 6 πιο συχνά αντικείμενα μαζί με τις εκτιμήσεις τους τρέχοντας την συνάρτηση `void fillCFromFile("mushrooms.txt","add")`, (στην οποία γίνεται η υλοποίηση του αλγορίθμου του Count-Sketch) (αναφέρθηκε Κεφάλαιο 4 ενότητα 4.1), παρατηρώντας ότι τα αντικείμενα που βγάζει μαζί με τις εκτιμήσεις τους δεν ανήκουν στα 10 κορυφαία αντικείμενα.

Αυτό συμβαίνει, γιατί το εκτελούμενο αρχείο, που κάνει την πράξη `hi[q] += si[q]` στην συνάρτηση `void fillH(int q, int A,int B, long p,String action)` (αναφέρθηκε Κεφάλαιο 4 ενότητα 4.1), ενώ αυξάνει τα στοιχεία στις πράξεις κατακερματισμού κατά 1, ώστε κάθε αντικείμενο που βρίσκει να προστίθεται στην ουρά μέχρι να βρεθεί το πιο συχνό μαζί με την εκτίμηση, δεν μπορεί να βρει το πιο συχνό, γιατί τα αντικείμενα η ενώ είναι ίδια σχεδόν (περιέχει 128 αντικείμενα ενώ το αρχείο "connect.txt" περιέχει 129), διαφέρουν στο σύνολο `m` των αντικειμένων καθώς στο "mushrooms.txt", έχουμε `m=193568`, ενώ στο άλλο αρχείο "connect.txt" που συγκρίνουμε έχει `m=2904951`.

Στην εικόνα 6Γ παρατηρούμε να εμφανίζει τα 6 πιο συχνά αντικείμενα μαζί με τις εκτιμήσεις τους τρέχοντας την συνάρτηση `void fillCFromFile("connect.txt","sub")`, (στην οποία γίνεται η υλοποίηση του αλγορίθμου του Count-Sketch) (αναφέρθηκε Κεφάλαιο 4 ενότητα 4.1), παρατηρώντας ότι ένα από τα 6 αντικείμενα που βγάζει μαζί με τις εκτιμήσεις τους, πιο συγκεκριμένα το 127 και το 109 με εκτίμηση 67541, ανήκει στα 10 κορυφαία αντικείμενα (διότι ενώ κάνει την πράξη $h_i[q] -= s_i[q]$ έχει μεγαλύτερη συχνότητα στα αντικείμενα, από το άλλο αρχείο).

Στις εικόνες 6Δ έως 6Η βλέπουμε τις εναλλαγές που γίνονται των αντικειμένων μαζί με την εκάστοτε συχνότητά τους τρέχοντας τον αλγόριθμο Brute-Force και συγκεκριμένα την συνάρτηση `void brute force(String Stream, String action)`, κάνοντας την πράξη

$$|n_q^{S^2} - n_q^{S^1}|, \text{ όπου } n_q = \text{median}\{h_i[q]s_i[q]\} \text{ (Ισχύει και } |n_q^{S^1} - n_q^{S^2}|)$$

Με λίγα λόγια, αρχικά ελέγχει τα αντικείμενα με την συχνότητά τους από το πρώτο αρχείο και στη συνέχεια αφού εκτελέσει το δεύτερο αρχείο συγκρίνει τα αντικείμενα μεταξύ αυτών των αρχείων, αν βρει κάποιο κοινό αντικείμενο που υπάρχει στο πρώτο αρχείο το αφαιρεί από το δεύτερο εμφανίζοντας τα τελικά αντικείμενα με τις εκτιμήσεις τους μετά την εναλλαγή που έγινε.

Τέλος η εικόνα 6Θ εμφανίζει τα 10 πιο συχνά αντικείμενα αντιστοιχίζοντας το 127 που βρήκαμε μέσω του αλγορίθμου Count-Sketch (εικόνα 6Γ), να ανήκει στα 10 κορυφαία αντικείμενα με εκτίμηση 67273.

Άρα καταλήγουμε στο συμπέρασμα ότι όταν έχουμε να συγκρίνουμε δύο ροές δεδομένων, συνήθως τα αντικείμενα των αρχείων με μικρότερη συχνότητα, δεν θα βρίσκονται στα κορυφαία αντικείμενα, αλλά σίγουρα κάποιο από τα αντικείμενα του αρχείου που είναι μεγαλύτερο, δηλαδή να περιέχει αντικείμενα με την μεγαλύτερη συχνότητα, σίγουρα θα περιέχονται στα κορυφαία αντικείμενα, με τις εκτιμήσεις ορισμένες φορές να αποκλίνουν, διότι τα αντικείμενα βρίσκονται μετά την εναλλαγή, με αποτέλεσμα να μειώνονται αντίστοιχα οι εκτιμήσεις.

Σε περίπτωση που ορίζαμε, σαν παράμετρο σφάλματος $\delta = 0.03f$ και σαν παράμετρο για την ποιότητα της λύσης $\epsilon = 0.03f$, τότε οι γραμμές θα αυξάνονταν και για τα δύο αρχεία και θα είχαμε καλύτερη ακρίβεια στα αντικείμενα μαζί με τις εκτιμήσεις τους.

Άρα θα είχαμε μεγαλύτερους υπολογισμούς στους πίνακες κατακερματισμού, οπότε και καλύτερη ακρίβεια στο να βρούμε τα πιο συχνά αντικείμενα μαζί με τις εκτιμήσεις τους.

Πιο αναλυτικά φαίνονται στις παρακάτω εικόνες Εικόνα 7Α και Εικόνα 7Β που δείχνουν τα αντικείμενα μαζί με τις εκτιμήσεις τους, χρησιμοποιώντας τον αλγόριθμο Count-Sketch, για τα δυο αρχεία “mushrooms.txt” και “connect.txt” αντίστοιχα.

```

emfanise sthles b
80
emfanise grammes t
16

o q poy brisketai sto heap - count estimate
39-1665

o q poy brisketai sto heap - count estimate
12-2760

o q poy brisketai sto heap - count estimate
128-3233

o q poy brisketai sto heap - count estimate
120-4135

o q poy brisketai sto heap - count estimate
88-4711

o q poy brisketai sto heap - count estimate
90-8488

```

Εικόνα 7Α- Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “mushrooms.txt”(για μικρές παραμέτρους)

```

emfanise grammes t
19

o q poy brisketai sto heap - count estimate
70-783

o q poy brisketai sto heap - count estimate
61-10364

o q poy brisketai sto heap - count estimate
40-32109

o q poy brisketai sto heap - count estimate
76-40923

o q poy brisketai sto heap - count estimate
100-60449

o q poy brisketai sto heap - count estimate
109-67545

```

Εικόνα 7Β- Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “connect.txt”(για μικρές παραμέτρους)

Αντίθετα, αν ορίζαμε, σαν παράμετρο σφάλματος $\delta = 0.9f$ και σαν παράμετρο για την ποιότητα της λύσης $\varepsilon = 0.9f$, τότε οι γραμμές θα μειώνονταν και για τα δύο αρχεία λόγω κακής ακρίβειας.

Άρα θα είχαμε μικρότερους πίνακες κατακερματισμού, οπότε και χειρότερη ακρίβεια στο να βρούμε τα πιο συχνά αντικείμενα μαζί με τις εκτιμήσεις τους.

Πιο αναλυτικά φαίνονται στις παρακάτω εικόνες Εικόνα 8Α και Εικόνα 8Β που δείχνουν τα αντικείμενα μαζί με τις εκτιμήσεις τους, χρησιμοποιώντας τον αλγόριθμο Count-Sketch, για τα δυο αρχεία “mushrooms.txt” και “connect.txt” αντίστοιχα.

```
emfanise sthles b
80
emfanise grammes t
13

o q poy brisketai sto heap - count estimate
39-1663

o q poy brisketai sto heap - count estimate
23-3446

o q poy brisketai sto heap - count estimate
79-4815

o q poy brisketai sto heap - count estimate
56-4936

o q poy brisketai sto heap - count estimate
71-5148

o q poy brisketai sto heap - count estimate
94-8292
```

Εικόνα 8Α- Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “mushrooms.txt”(για μεγάλες παραμέτρους)

```
emfanise grammes t
15

o q poy brisketai sto heap - count estimate
67-2429

o q poy brisketai sto heap - count estimate
40-32108

o q poy brisketai sto heap - count estimate
97-51245

o q poy brisketai sto heap - count estimate
10-55407

o q poy brisketai sto heap - count estimate
82-62428

o q poy brisketai sto heap - count estimate
37-67120
```

Εικόνα 8Β- Αντικείμενα και εκτίμηση που προκύπτουν από εκτέλεση του Count-Sketch για το αρχείο “connect.txt”(για μεγάλες παραμέτρους)

Τέλος, τα αποτελέσματα από το παραπάνω πείραμα παρουσιάζεται συνοπτικά στον Πίνακα 4.

Σύνολο Αποτελεσμάτων								
Σύνολο Δεδομένων	Αντικείμενα n	Πλήθος αντικειμένων m	Παράμετρος πιθανότητας σφάλματος δ	Παράμετρο ποιότητας λύσης ε	Γραμμές t	Στήλες b	Αλγόριθμος Count- Sketch (αντικείμενο ο – εκτίμηση)	Αλγόριθμος Brute-Force 10 κορυφαία (αντικείμενο – πραγματική τιμή)
mushrooms.txt	129	193568	0.03f	0.03f	16	80	39-1665	91- 67471
							12-2760	109- 67421
							128-3233	127-67471
							120-4135	37- 67040
							88-4711	19- 67024
							90-8488	106- 66817
			0.3f	0.2f	14	80	113-124	75- 66669
							12-2754	124- 66418
							23-3450	72- 65593
							57-3875	85- 65258
							36-8270	
							94-8292	
			0.9f	0.9f	13	80	39-1663	
							23-3446	
							79-4815	
							56-4936	
							71-5148	
							94-8292	
Connect.txt	130	2904951	0.03f	0.03f	19	80	70-783	
							61-10364	
							40-32109	
							76-40923	
							100-60449	
							19-67545	
			0.3f	0.2f	17	80	73-279	
							61-10361	
							115-48181	
							82-62427	
							52-65218	
							127-67541	
							67-2429	
							40-32108	
							97-51245	

			0.9f	0.9f	15	80	10-55407
							82-62428
							37-67120

Πίνακας 4- Συνοπτικός πίνακας όλων των αποτελεσμάτων για δυο ροές.

ΕΠΙΛΟΓΟΣ

Σε αυτό το Κεφάλαιο όπου είναι και ο επίλογος παρουσιάζονται συνοπτικά τα συμπεράσματα και παραθέτει σκέψεις για το πώς η Διπλωματική εργασία μπορεί να επεκταθεί μελλοντικά.

6.1 Συμπεράσματα

Ο κώδικας που αναπτύχθηκε για την δημιουργία του αλγορίθμου του Count Sketch είναι αρκετά αποδοτικός, καθώς καταφέρνει να βρίσκει τα πιο συχνά αντικείμενα από μια ροή δεδομένων μαζί με την εκτίμησή τους, όπως επίσης να βρίσκει και τις μεγαλύτερες μεταβολές μεταξύ δύο ροών δεδομένων.

Την ορθότητα των αποτελεσμάτων (από τη Count Sketch) την ελέγξαμε με τη δημιουργία του αλγορίθμου της Brute Force συγκρίνοντας τα αντικείμενα με τις εκτιμήσεις που έβγαλε ο αλγόριθμος Count Sketch και τα αντικείμενα με την συχνότητα που εμφάνισε ο αλγόριθμος της Brute Force και ανακαλύπτοντας ότι τα αντικείμενα ταιριάζουν και οι εκτιμήσεις είναι πολύ κοντά αριθμητικά μεταξύ τους.

Ακόμη συμπεραίνουμε ότι όταν τρέχουμε δύο ροές δεδομένων και στην πρώτη ροή δεδομένων εκτελούμε το αρχείο κάνοντας την πράξη $h_i[q] \leftarrow s_i[q]$, παρατηρούμε ότι όταν τρέχουμε τον αλγόριθμο του Count Sketch τα αντικείμενα μαζί με τις εκτιμήσεις που εμφανίζει, δεν είναι τα πιο συχνά αντικείμενα. Αυτό συμβαίνει συνήθως για ίδιο ή παρόμοιο αριθμό πλήθος m . (αναφέρθηκε Κεφάλαιο 5 ενότητα 5.3.3).

Ακόμη σημαντικό ρόλο παίζει και το πλήθος των αντικειμένων που περιέχει μια ροή καθώς, όταν συγκρίνουμε δυο ροές, όπου η μια έχει πολύ μεγαλύτερο πλήθος αντικειμένων από την άλλη, τα πιο συχνά αντικείμενα εμφανίζονται συνήθως στη ροή με το μεγαλύτερο σύνολο αντικειμένων.

Παρατηρούμε ακόμη, ότι σημαντικό ρόλο παίζει και για την τιμή που θα δώσουμε για τους παραμέτρους ϵ , δ , καθώς αυτοί είναι που καθορίζουν τις γραμμές t των πινάκων κατακερματισμού και για το πόσο καλή είναι η προσέγγιση της λύσης ή όχι.

Έτσι όσο πιο μεγάλες τιμές έχουμε στις παραμέτρους δ, ϵ τόσο οι γραμμές μειώνονται και όσο πιο μικρούς πίνακες κατακερματισμού έχουμε τόσο πιο ανακρίβεια στα αποτελέσματα έχουμε (βλ. Εικόνα 6Α και Εικόνα 6Β).

Οπότε για να έχουμε ακρίβεια θα πρέπει να δίνουμε όσο πιο μικρές τιμές γίνεται ακόμη και να τείνει προς το 0, καθώς οι τιμές που μπορούν να πάρουν οι παράμετροι δ, ϵ είναι από 0 έως 1.

6.2 Μελλοντικές Προοπτικές

Το πρόγραμμα που αναπτύχθηκε για την παρούσα διπλωματική εργασία θα μπορούσε μελλοντικά να επεκταθεί ώστε να γίνει ακόμα πιο ακριβές.

Μία από τις βελτιώσεις που θα μπορούσαν να γίνουν ώστε τα δυο προβλήματα που μελετάμε στην συγκεκριμένη Διπλωματική εργασία (αναφέρθηκαν Κεφάλαιο 2 ενότητα 2.2 και 2.3) να είναι ακόμη πιο ακριβή, είναι στο χρονισμό των αποτελεσμάτων, διότι τα αποτελέσματα εξαρτώνται κυρίως από την παράμετρο δ (επιλογή αντικειμένων όσο και η λειτουργία).

Με αποτέλεσμα ο Count-Sketch αλγόριθμος να δίνει ανακριβή αποτελέσματα, για πολύ μεγάλη τιμή στη παράμετρο πιθανότητα σφάλματος, δ και έτσι να χρειαστεί να κάνει πιο λίγους υπολογισμούς για να βρει τους πίνακες κατακερματισμού (από κάποιον άλλον αλγόριθμο).

Ένα άλλο ενδιαφέρον πρόβλημα που πρέπει να προσέξουμε και να βελτιώσουμε μελλοντικά είναι ο υπολογισμός των σφαλμάτων του Count-Sketch στην πράξη, όταν περιορίζουμε το χώρο του στη χρήση.

Με αποτέλεσμα να μην δίνει σωστά αποτελέσματα για το μέσο σφάλμα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] https://databricks.com/session/some-important-streaming-algorithms-you-should-know-about?fbclid=IwAR2O8Da--0zTu75loZ-uGW4ndE9jbBClnCLxpjFuncO3OL5TODm-Ep_GD7
- [2] https://en.wikipedia.org/wiki/Streaming_algorithm
- [3] <http://helios.mi.parisdescartes.fr/~themisp/publications/dke09.pdf>
- [4] <https://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf>
- [5] https://en.wikipedia.org/wiki/Count_sketch
- [6] https://ecourse.uoi.gr/pluginfile.php/91446/mod_folder/content/0/Hash_Functions.pdf?forcedownload=1
- [7] <https://www.cs.helsinki.fi/u/jilu/paper/Course5.pdf>
- [8] <https://www.sanfoundry.com/java-program-miller-rabin-primality-test-algorithm/>
- [9] <http://www2.hawaii.edu/~suthers/courses/ics311f20/Notes/Topic-06.html>
- [10] https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test
- [11] <http://www.eng.ucy.ac.cy/christos/Courses/ECE325/Lectures/BruteForce.pdf>
- [12] Alon, Noga; Matias, Yossi; Szegedy, Mario (1996), "The space complexity of approximating the frequency moments", *Proceedings of the 28th ACM Symposium on Theory of Computing (STOC 1996)*, pp. 20–29, [CiteSeerX 10.1.1.131.4984](#), [doi:10.1145/237814.237823](#), [ISBN 978-0-89791-785-8](#), [S2CID 1627911](#)
- [13] <https://www.freecodecamp.org/news/brute-force-algorithms-explained/>
- [14] <https://stackoverflow.com/questions/6811351/explaining-the-count-sketch-algorithm>