Η Δομή Count-Min Sketch και οι Εφαρμογές της

The Count-Min Sketch Data Structure and its Applications

Ανέστης Κυρκενίδης

Διπλωματική Εργασία

Επιβλέπων: Λουκάς Γεωργιάδης

Ιωάννινα, Οκτώβριος, 2021

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τη μητέρα μου που στάθηκε δίπλα μου σε όλη τη διάρκεια των σπουδών μου και με στήριξε στη προσπάθεια μου. Στους φίλους που ήταν μαζί μου και με βοήθησαν σε ότι χρειάστηκα και στους καθηγητές μου για τις γνώσεις που με πρόσφεραν. Τέλος, θα ήθελα να ευχαριστήσω και τον κ.Γεωργιάδη που με καθοδήγησε στην παρούσα εργασία.

Ανέστης Κυρκενίδης

Τμήμα Μηχ. Η/Υ και Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

Περίληψη

Σε αυτή τη διατριβή, ασχολούμαστε με το πρόβλημα διατήρησης τεράστιων συνόλων δεδομένων στατιστικών στοιχείων σε μια δομή δεδομένων, χωρίς την χρήση μεγάλου αποθηκευτικού χώρου. Πιο συγκεκριμένα θα αναλύσουμε τη δομή Count-Min Sketch, η οποία αποθηκεύει μια περίληψη των συνόλων δεδομένων σε υπογραμμικό χώρο. Η Count-Min Sketch υποστηρίζει την αποδοτική απάντηση σε διάφορους τύπους ερωτημάτων, όπως η εκτίμηση ενός αντικειμένου και η τιμή ενός εύρους διαστημάτων, παρέχοντας καλές εκτιμήσεις των δεδομένων. Στα ακόλουθα κεφάλαια θα εξηγήσουμε τη λειτουργία της δομής και θα παρέχουμε πειραματική ανάλυση για την αποτελεσματικότητά της.

Λέξεις Κλειδιά: Count-Min Sketch, σκίτσο, δομή δεδομένων, αλγόριθμος ροής, ροές δεδομένων, συχνότητα, υπογραμμική πολυπλοκότητα, sketches

Abstract

In this thesis, we are concerned with the maintenance of huge datasets of numerical data, in a data structure without the usage of large amount of storage space. Specifically, we will analyze the data structure Count-Min Sketch which stores sublinear approximations of our datasets. Count-Min Sketch supports the efficient response to several types of queries, such as Point Query and Range Query, guaranteeing good estimations of our data. In this thesis, we will explain the data structure's function and provide experimental analysis of its' efficacy.

Keywords: Count-Min Sketch, sketch, data structure, streaming algorithms, data streams, frequency, sub-linear complexity, sketches

Πίνακας Περιεχομένων

Κεφάλαιο 1. Εισαγωγή

Στη σημερινή εποχή η επέκταση του διαδικτύου επιτρέπει τους χρήστες να αλληλοεπιδρούν με τεράστια ποικιλία εφαρμογών και υπηρεσιών. Οι αλληλεπιδράσεις αυτές έχουν ως αποτέλεσμα την ύπαρξη ροών δεδομένων που εναλλάσσονται μεταξύ χρηστών και υπηρεσιών. Μια συνεχής ροή δεδομένων μετακινείται μεταξύ της πηγής και του παραλήπτη σε αληθινό χρόνο. Οι ροές δεδομένων περιέχουν ακατέργαστα δεδομένα που συλλέγονται από την φύση της αλληλεπίδρασης των χρηστών με τις υπηρεσίες παρέχοντας χρήσιμες πληροφορίες στις υπηρεσίες ή εφαρμογές με χρήση επιστημόνων δεδομένων και αλγορίθμων τεχνητής νοημοσύνης που έχουν δημιουργηθεί για την ανάλυση των ροών δεδομένων. Παρόλη την σημασία τους, οι ροές δεδομένων, εξαιτίας της μαζικής χρήσης και αλληλεπίδρασης με εφαρμογές και υπηρεσίες, καταλήγουν να απαιτούν τεράστιες ποσότητες μνήμης προκειμένου να αποθηκευτούν πλήρως, με την ποσότητα τους να αυξάνεται κάθε δευτερόλεπτο.

1.1 Υπολογιστικό Μοντέλο Ροής Δεδομένων

Για να αντιμετωπιστεί το πρόβλημα αυτό, έχουν δημιουργηθεί μοντέλα ροής δεδομένων που μπορούν να αποθηκεύσουν τις ροές δεδομένων χωρίς τη χρήση μεγάλης ποσότητας μνήμης. Ένα σκίτσο δομής δεδομένων μας επιτρέπει να αποθηκεύουμε μια εκτίμηση του αρχικού συνόλου δεδομένων σε υπογραμμικό χώρο αποθήκευσης. Εφόσον η χρησιμότητα των ροών δεδομένων είναι στην ανάλυση τους για την εξαγωγή δεδομένων και στατιστικών, μπορούμε να ανεχτούμε μικρές ανακρίβειες. Θυσιάζοντας την ακρίβεια των δεδομένων μπορούμε να αποθηκεύσουμε προσεγγίσεις των ροών αυτών λαμβάνοντας μια καλή εκτίμηση των δεδομένων τους. Τα μοντέλα αυτά έχουν εφαρμογές στη δικτύωση, όπως στην παρακολούθηση δικτύων και την καταμέτρηση διακριτών ροών. Στη δομή Count-Min Sketch η ακρίβεια της εκτίμησης κρίνεται από τις παραμέτρους (ε, δ). Συνεπώς, η απόδοση της δομής εξαρτάται άμεσα από τις

παραπάνω παραμέτρους, οι οποίες δηλώνουν ότι το σφάλμα εκτίμησης είναι στα πλαίσια της *ε* με πιθανότητα δ. Ως αποτέλεσμα, η απόδοση μνήμης και ο χρόνος ενημέρωσης του σκίτσου και απάντησης ερωτημάτων, όπως η Point Query, καθορίζονται από τις τιμές *ε* και *δ*.

1.2 Παραδείγματα Χρήσης Υπολογιστικών Μοντέλων Ροής Δεδομένων

Τα υπολογιστικά μοντέλα ροής δεδομένων χρησιμοποιούνται για την επίλυση διαφόρων προβλημάτων. Τέτοια προβλήματα είναι η εύρεση ξεχωριστών αντικειμένων σε ροές δεδομένων με επαναλαμβανόμενα αντικείμενα και η ανίχνευση συμβάντων σε ροές δεδομένων. Η ανίχνευση επιτυγχάνεται με χρήση αλγορίθμων εύρεσης των heavy hitters, όπως η δομή Count-Min Sketch και τα φίλτρα Bloom, υπολογίζοντας τα πιο συχνά αντικείμενα και έπειτα συγκρίνοντας την αύξηση της συχνότητας αυτών των αντικειμένων στο συγκεκριμένο χρονικό σημείο σε σχέση με τη προηγούμενη φορά που έγινε ο υπολογισμός της συχνότητας. Το αποτέλεσμα του προηγούμενου υπολογισμού είναι το ζητούμενο του προβλήματος ανίχνευσης συμβάντος. Ένα ακόμα πρόβλημα που μπορεί να λυθεί με χρήση υπολογιστικών μοντέλων ροής είναι η διαδικτυακή μηχανική μάθηση. Η διαδικτυακή μηχανική μάθηση είναι μια μέθοδος μηχανικής μάθησης στην οποία τα δεδομένα μιας διαδοχική σειράς δεδομένων χρησιμοποιούνται για την πρόβλεψη των μελλοντικών δεδομένων στο επόμενο βήμα της σειράς. Τέλος, ένα από τα πιο γνωστά προβλήματα είναι το πρόβλημα συχνών στοιχείων στο οποίο το αποτέλεσμα είναι τα στοιχεία που απαρτίζουν μεγαλύτερο από ένα δοθέν ποσοστό των δεδομένων ροής. Μια παραλλαγή αυτού του προβλήματος είναι το πρόβλημα της πλειοψηφίας που υπολογίζει αν κάποια τιμή αποτελεί ή όχι τη πλειοψηφία της ροής. Τα δύο τελευταία προβλήματα λύνονται κυρίως με αλγορίθμους όπως τα φίλτρα Bloom και η δομή Count-Min Sketch.

Ένα απλό παράδειγμα του προβλήματος συχνών στοιχείων που αποδεικνύει τη χρησιμότητα δομών δεδομένων όπως η Count-Min Sketch είναι η παρακολούθηση των τάσεων αναζήτησης σε ιστοσελίδες υπηρεσιών, όπως η αναζήτηση προϊόντων στην Amazon. Η πλήρης και ακριβής αποθήκευση των δεδομένων αναζήτησης μπορεί να γίνει δυνατή δίνοντας ακριβή στατιστικά στοιχεία αναζητήσεων των χρηστών. Παρόλα αυτά, μια τέτοια υλοποίηση θα ήταν ακριβή και χρονοβόρα. Η χρήση γρήγορων αλγορίθμων αναζήτησης όπως οι δεντρικές δομές είναι αργές λόγω του μεγέθους των δεδομένων. Σε αυτό το παράδειγμα, ενδιαφερόμαστε μόνο για την εξαγωγή

στατιστικών δεδομένων για την εύρεση των τάσεων αναζήτησης των χρηστών και δεν μας ενδιαφέρει η ακριβής καταμέτρηση των δεδομένων. Για το λόγο αυτό μικρά σφάλματα εκτιμήσεων γίνονται δεκτά καθώς ο σκοπός είναι η ανάλυση των συχνοτήτων αναζήτησης διαφόρων προϊόντων. Ως αποτέλεσμα, μπορούμε να ανταλλάξουμε λίγη ακρίβεια των αποτελεσμάτων με έναν πιο αποδοτικό και ελαφρύ μηχανισμό. Με το τρόπο αυτό η εταιρεία μπορεί με λιγοστό κόστος αποθήκευσης των δεδομένων, να αναλύσει τις τάσεις αναζήτησης των χρηστών και να στοχεύει τις αγορές τους προτείνοντας προϊόντα που ταιριάζουν σε κάθε χρήστη ατομικά ανάλογα με τις προτιμήσεις τους.

Μια ακόμα σημαντική εφαρμογή της Count-Min Sketch είναι η χρήση της στην Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing, NLP) όπου παρατηρούνται πολύ μεγάλα σετ δεδομένων. Η NLP αποτελεί κλάδο της τεχνητής νοημοσύνης με σκοπό την κατανόηση της ανθρώπινης γλώσσας από τους υπολογιστές, στον ίδιο βαθμό που την κατανοούν οι άνθρωποι, μέσω της παροχής κειμένου και φυσικής ομιλίας. Χρησιμοποιεί στατιστικά μοντέλα, μηχανικής και βαθιάς μάθησης, τα οποία επιτρέπουν στους υπολογιστές να επεξεργάζονται την ανθρώπινη γλώσσα, είτε σε γραπτή μορφή είτε σε ομιλία, με σκοπό την πλήρη κατανόηση του νοήματος της, συμπεριλαμβανομένου του συναισθήματος και σκοπό του συγγραφέα ή ομιλητή . Η NLP έχει λειτουργίες όπως η αναγνώριση ομιλίας, που μετατρέπει την ομιλία σε γραπτό λόγο και αναγνώριση λέξεων και της σημασίας τους. Για παράδειγμα η λέξη Γιώργος αναφέρεται σε όνομα ανθρώπου ενώ η λέξη Λονδίνο αναφέρεται σε πόλη. Επιπλέον, η NLP μπορεί να αναγνωρίσει τη σημασίας μιας λέξης σε σχέση με τα συμφραζόμενα της, να αναλύσει το γραπτό λόγο για την εύρεση συναισθημάτων και γνωρίσματα όπως ο σαρκασμός καθώς και να δημιουργήσει γραπτό δομημένο λόγο από ένα σύνολο πληροφοριών (Natural Language Generation, NLG). Μερικές σημαντικές λειτουργίες της NLP βασίζονται στην παρακολούθηση καταμέτρησης (count tracking).

Το count tracking έχει πολλές εφαρμογές, μια εκ των οποίων είναι και το πρώτο παράδειγμα χρήσης των υπολογιστικών μοντέλων ροής δεδομένων. Στην NLP είναι σημαντικό να κρατήσουμε στατιστικά στοιχεία για τη συχνότητα συνδυασμών λέξεων, συνδυασμοί όπως ζευγάρια ή τρίστιχα λέξεων. Σε ένα πείραμα, ερευνητές μπόρεσαν με την χρήση της Count-Min Sketch να μειώσουν το μέγεθος των δεδομένων από 90GB σε 8GB, που συγκριτικά είναι πολύ πιο ελαφρύ στη μνήμη.

Η χρήση της NLP σε εφαρμογές αληθινού χρόνου (real-time applications) έχει επιφέρει πολλές βελτιώσεις στην αλληλεπίδραση των ανθρώπων με το διαδίκτυο. Η ανίχνευση ανεπιθύμητων μηνυμάτων (spam detection) χρησιμοποιεί την NLP για να αναγνωρίσει

γλώσσα που χρησιμοποιείται σε spam ή επιθέσεις phishing. Άλλες τέτοιες εφαρμογές της NLP είναι η μετάφραση κειμένου με το Google Translate να είναι το πλέον διαδεδομένο εργαλείο μετάφρασης το οποίο χρησιμοποιεί την NLP, η περίληψη κειμένου που χρησιμοποιεί NLP για να συμπτύξει ένα κείμενο και η ανάλυση συναισθήματος στα μέσα κοινωνικής δικτύωσης (social media), η οποία χρησιμοποιείται από εταιρείες για καλυτέρευση και εκσυγχρονισμό των προϊόντων τους αναλόγως με τις τάσεις στα social media.

1.3 Σύνοψη

Όπως αναφέραμε παραπάνω, το μοντέλο ροής δεδομένων αποτελεί αναπόσπαστο κομμάτι του σημερινού διαδικτύου και είναι βασικός πυλώνας για πολλές εφαρμογές, αρκετές από τις οποίες αποτελούν κομμάτι της καθημερινότητας μας. Εξ αυτού, η σημασία και η χρησιμότητα της Count-Min Sketch, που είναι μια από τις πιο διαδεδομένες δομές δεδομένων, είναι αδιαμφισβήτητη.

Κεφάλαιο 2. Τεχνικές της Count-Min Sketch

Σε αυτό το κεφάλαιο θα αναλύσουμε την Count-Min Sketch που προτάθηκε από τους Graham Cormode και S. Muthukrishnan και θα περιγράψουμε τη λειτουργία και τη θεωρητική ανάλυση της απόδοσης της δομής. Το κεφάλαιο αυτό περιλαμβάνει 7 ενότητες. Το στάδιο προεπεξεργασίας, που αφορά την δημιουργία του πίνακα, την ενημέρωση του πίνακα με στοιχεία, την εκτίμηση των αντικειμένων, το εσωτερικό γινόμενο δύο πινάκων CM, την εκτίμηση εμβέλειας ενός πίνακα, την εκτίμηση φ-Ποσοτικού, στο οποίο θα αναφερόμαστε ως φ-Quantile, μιας CM και την εύρεση βαρειών χτυπημάτων, στα οποία θα αναφερόμαστε ως heavy hitters, που είναι τα πιο κοινά στοιχεία σε ένα πίνακα. [1]

Για καλύτερη κατανόηση του κειμένου αλλά και των αποδείξεων, θέτουμε κάποιους όρους στην εισαγωγή. Θεωρούμε πως κάθε στοιχείο μιας ροής δεδομένων έχει θετική τιμή. Για τα παρακάτω κεφάλαια θα αναφερόμαστε στην τιμή μιας συνάρτησης κατακερματισμού ως $h_j(i)$, όπου j αριθμός της συνάρτησης κατακερματισμού με $j \leq depth$ και i να είναι το αντικείμενο ενώ α_i να είναι η αξία του και $\forall\, \alpha_i$ ισχύει $\alpha_i \geq 0$. Συνεπώς, η τιμή του αντικειμένου i στη γραμμή j του πίνακα είναι count[j, $h_j(i)$] με count ο πίνακας της CM, ενώ η ήδη υπάρχουσα τιμή της θέσης i, j είναι $X_{i,j}$. Τέλος, θέτουμε $||\alpha||_1 = \sum_{i=1}^{n} \alpha_i$, το άθροισμα των απόλυτων τιμών των δεδομένων (L1-norm).

2.1 Στάδιο προεπεξεργασίας

Η Count-Min Sketch (CM Sketch) είναι μια πιθανοτική δομή δεδομένων που απλοποιεί πολύ μεγάλα δεδομένα σε δεδομένα συχνοτήτων με χρήση συναρτήσεων κατακερματισμού αποθηκεύοντας τα δεδομένα σε ένα διδιάστατο πίνακα

χρησιμοποιώντας συναρτήσεις κατακερματισμού. Σε αντίθεση με ένα πίνακα κατακερματισμού, η Count-Min Sketch χρησιμοποιεί υπογραμμικό χώρο αποθήκευσης έχοντας ως αποτέλεσμα την υπεραρίθμηση των δεδομένων. Αυτό σημαίνει πως η δομή CM δεν κρατάει ακριβή δεδομένα, αλλά εκτιμήσεις αυτών. Η CM αποτελείται από πολλές συναρτήσεις κατακερματισμού, που είναι ανεξάρτητες ανά ζεύγη, στις οποίες περνάει τις συχνότητες των δεδομένων που διαβάζει. Τα δεδομένα αυτά αποθηκεύονται σε ένα διδιάστατο πίνακα, που τον ονομάζουμε count, με διαστάσεις width και depth. Κάθε οριζόντια γραμμή του πίνακα αντιστοιχεί σε μια συνάρτηση κατακερματισμού, συνεπώς μια CM αποτελείται από depth συναρτήσεις κατακερματισμού, ενώ η τιμή width αντιστοιχεί στις θυρίδες κατακερματισμού για κάθε συνάρτηση κατακερματισμού. Οι τιμές width και depth εξαρτώνται από τις τιμές ε και δ. Οι (ε, δ) όπως αναφέρθηκε στην Ενότητα 1.1, είναι υπεύθυνες για την ακρίβεια της CM. Η μεταβλητή ε καθορίζει το βαθμό σφάλματος που επιτυγχάνει η δομή, ενώ η μεταβλητή δ καθορίζει την πιθανότητα εκτίμησης εντός του βαθμού σφάλματος. Οι τιμές width και depth υπολογίζονται συναρτήσει των ε και δ αντίστοιχα. Σύμφωνα με τις δημοσιεύσεις των δημιουργών της CM, Graham Cormode και S. Muthukrishnan, υπάρχουν δύο διαφορετικοί τρόποι υπολογισμού των width και depth. Εμείς θα ασχοληθούμε με τον δεύτερο τρόπο, εφόσον ο δεύτερος τρόπος έχει πιο απλούς υπολογισμούς.

Ο δεύτερος, και πιο πρόσφατος, τρόπος δημιουργίας του δυσδιάστατου πίνακα για τη CM είναι να θεωρήσουμε το δ ως την πιθανότητα η δομή να πετύχει σφάλμα ε, δηλαδή η εκτίμηση της δομής να είναι στα πλαίσια του ε. Ο υπολογισμός του width γίνεται $width = \left\lceil \frac{2}{\varepsilon} \right\rceil$ και του depth γίνεται $depth = \lceil -\log_2(1-\delta) \rceil$. Έστω οι τιμές (ε, δ) = (0.01, 0.99), δηλαδή πετυχαίνει σφάλμα 0.01 με πιθανότητα 0.99. Ως αποτέλεσμα λαμβάνουμε τις τιμές width = 200 και depth = 7. Η CM χρησιμοποιεί 7 συναρτήσεις κατακερματισμού με τη καθεμία να έχει 200 θυρίδες κατακερματισμού. Υλοποιήσαμε τη δική μας CM χρησιμοποιώντας τον παραπάνω τρόπο καθώς έχουμε απλούστερους υπολογισμούς και επιτυγχάνουμε ταχύτερες ενημερώσεις το οποίο θα παραθέσουμε στην Ενότητα 2.2.

Συνολικά, ο πίνακας της CM περιέχει τις συχνότητες των αντικειμένων του συνόλου δεδομένων, όπου η συχνότητα ενός αντικειμένου λαμβάνεται από μια διαδικασία εκτίμησης (ESTIMATE) της ολικής συχνότητας του αντικειμένου στο πίνακα. Δεδομένου ότι το πλάτος του πίνακα (width) είναι μικρότερο από το πλήθος των δεδομένων που διαπερνιούνται στον πίνακα, θα υπάρχουν συγκρούσεις, δηλαδή η συχνότητα ενός αντικειμένου i σε ένα σημείο θα είναι ίση με το άθροισμα των

συχνοτήτων όλων των αντικειμένων που έχουν τοποθετηθεί σε αυτή τη θέση. Επειδή μια συνάρτηση κατακερματισμού κατανέμουν τα αντικείμενα ομοιόμορφα σε κάθε σειρά του πίνακα, περιμένουμε να έχουμε ένα ομοιόμορφο ποσοστό αντικειμένου που συγκρούονται με το αντικείμενο i. Κάθε συνάρτηση h∈H θεωρείται ανεξάρτητη ανά ζεύγη, αν για οποιαδήποτε τιμή x και y με $x \neq y$, και x,y ∈ U και n το πλήθος των θυρίδων κατακερματισμού, ισχύει [2]:

$$P[h(x) = u_1 και\ h(y) = u_2] \leq \frac{1}{n^2}$$

Ή αλλιώς:

$$P[h(x) = h(y)] \leq \frac{1}{n}$$

Για να εκτιμήσουμε τη πιθανότητα σφάλματος χρησιμοποιούμε την Markov's inequality. Σύμφωνα με την ανισότητα αυτή, η πιθανότητα η τιμή να είναι α φορές παραπάνω από την αναμενόμενη είναι πολύ $\frac{1}{\alpha}$. Έστω N το ακριβές άθροισμα τιμών των αντικειμένων σε ένα σύνολο δεδομένων και X μια τυχαία μεταβλητή. Επειδή οι συναρτήσεις κατακερματισμού κάνουν ομοιόμορφη κατανομή, θεωρούμε πως η αναμενόμενη τιμή της διακριτής τυχαίας μεταβλητής X θα είναι $E(X) = \frac{N}{width}$. Όπως αναφέραμε στη παραπάνω εξίσωση, επειδή οι συναρτήσεις κατακερματισμού είναι ανεξάρτητες, ο μόνος τρόπος με τον οποίον θα έχουμε α φορές μεγαλύτερη αναμενόμενη τιμή, θα είναι αν έχουμε σύγκρουση. Οπότε:

$$P(\ X\ \geq \alpha\ ) \leq \frac{E(X)}{\alpha} \Leftrightarrow \mathrm{P}(\ X \geq \alpha \times E(X)\ ) \leq \frac{1}{\alpha}$$

Δηλαδή, αν α = 2, τότε η πιθανότητα η αναμενόμενη τιμή να είναι μεγαλύτερη της $\frac{\alpha N}{width}$ είναι 0.5. Η διαδικασία αυτή επαναλαμβάνεται για κάθε γραμμή, με διαφορετικές συναρτήσεις κατακερματισμού. Επειδή οι συναρτήσεις κατακερματισμού είναι διαφορετικές θα μας δίνουν διαφορετικά αποτελέσματα για το ίδιο αντικείμενο σε κάθε γραμμή. Κάθε φορά θα έχουμε το πολύ 50% πιθανότητα η αναμενόμενη τιμή να ξεπεράσει την $\frac{\alpha N}{width}$ και τουλάχιστον 50% πιθανότητα να μην την ξεπεράσει. Επειδή στην διαδικασία εκτίμησης παίρνουμε τις μικρότερες τιμές για το αντικείμενο x σε κάθε γραμμή, η πιθανότητα να προκύψει σφάλμα μεγαλύτερο του $\frac{\alpha N}{width}$ είναι το πολύ 0.5^{depth}. Συνεπώς αν θέλουμε να πετύχουμε σφάλμα 0.1% με 99% πιθανότητα θα έχουμε $\varepsilon = \frac{2}{width} = 0.01 \Leftrightarrow width = 200$ και $0.5^{depth} = \delta \Leftrightarrow depth = -\log_2(1 - \delta) = 7$. [3]

Αφότου ορίσουμε τα width και depth μπορούμε να δημιουργήσουμε τον πίνακα, του οποίου τα στοιχεία τα αρχικοποιούμε όλα σε 0 και θέτουμε το μετρητή για το συνολικό πλήθος των στοιχείων στο 0.

2.2 Ενημέρωση πίνακα

Για να προσθέσουμε τα δεδομένα στο πίνακα πρέπει να τον ενημερώσουμε τις τιμές των δεδομένων αυτών. Η ενημέρωση του πίνακα με ένα αντικείμενο i με αξία α_i, γίνεται μέσω της διαδικασίας update(i, α_i), ως εξής:

- Παίρνουμε τη τιμή hash του αντικειμένου i. Αν το αντικείμενο είναι ήδη ακέραιος, τότε δε χρειάζεται να βρούμε τη hash τιμή του καθώς αυτή είναι ο ίδιος ο ακέραιος.
- Χρησιμοποιούμε τη συνάρτηση κατακερματισμού για να βρούμε τη θέση στην οποία αποθηκεύεται η αξία της i σε μια γραμμή. Κάθε θέση του πίνακα περιέχει μια τιμή, έστω w. Αφού βρούμε τη θέση, προσαυξάνουμε την τιμή της w με την αξία του αντικειμένου i, α_i. Αυτό επαναλαμβάνεται για κάθε γραμμή του πίνακα.

Κάθε συνάρτηση κατακερματισμού στον πίνακα είναι διαφορετική και κατανέμει τα αντικείμενα ομοιόμορφα σε κάθε γραμμή του πίνακα, καθώς οι συναρτήσεις είναι ανεξάρτητες ανά ζεύγη όπως παραπάνω εξηγήσαμε παραπάνω. Ας θεωρήσουμε πως ενημερώνουμε ένα πίνακα μεγέθους 3x4 και N = 10 η συνολική αξία των δεδομένων. Ας υποθέσουμε ότι έχουμε 3 συναρτήσεις κατακερματισμού h1(k), h2(k), h3(k), με $h1(k) = (2 \times k + 3) mod\ 4$, $h2(k) = (3 \times k + 2) mod\ 4$ και $h3(k) = (4 \times k + 1) mod\ 4$. Αν εκτελέσουμε update (3, 2) θα λάβουμε τα εξής αποτελέσματα:

- $h1(3) = (2 \times 3 + 3) mod\ 4 = (6 + 3) mod\ 4 = 9\ mod\ 4 = 1$
- $h2(3) = (3 \times 3 + 2) mod\ 4 = (9 + 2) mod\ 4 = 11\ mod\ 4 = 3$
- $h3(3) = (4 \times 3 + 1) mod\ 4 = (12 + 1) mod\ 4 = 13\ mod\ 4 = 1$

Ενημερώνουμε τον πίνακα με ένα επιπλέον στοιχείο. Έστω εκτελούμε Update (8, 4). Λαμβάνουμε τις εξής θέσεις από τις συναρτήσεις κατακερματισμού:

- $h1(8) = (2 \times 8 + 3) mod\ 4 = (16 + 3) mod\ 4 = 19\ mod\ 4 = 3$
- $h2(8) = (3 \times 8 + 2) mod\ 4 = (24 + 2) mod\ 4 = 26\ mod\ 4 = 2$
- $h3(8) = (4 \times 8 + 1) mod\ 4 = (32 + 1) mod\ 4 = 33\ mod\ 4 = 1$

Αρχικός Πίνακας

0	0	0	0
0	0	0	0
0	0	0	0

Update (3, 2)

0	2	0	0
0	0	0	2
0	2	0	0

Update (8, 4)

0	2	0	4
0	0	4	2
0	6	0	0

Εικόνα 2.1: Λειτουργία της update στην Count-Min Sketch

Παρατηρούμε ότι στο συγκεκριμένο παράδειγμα, τα δύο αντικείμενα που προστίθενται έχουν συγκρουστεί. Στη τελευταία συνάρτηση κατακερματισμού, στη τελευταία γραμμή του πίνακα και στη θέση 1, θεωρώντας πως ένας πίνακα ξεκινάει από το 0, υπήρξε σύγκρουση. Αυτό συμβαίνει γιατί $h3(3) = h3(8)$. Παρότι τα δύο αντικείμενα είναι διαφορετικά υπήρξε σύγκρουση. Η συγκεκριμένη σύγκρουση όμως δε μας επηρεάζει για λόγους που θα αναλύσουμε παρακάτω.

2.3 Εκτίμηση αντικειμένων

Στα περισσότερα προβλήματα μας ζητείται να βρούμε την συχνότητα ενός ή πολλών αντικειμένων σε μια ροή ή μια βάση δεδομένων. Αυτό επιτυγχάνει με την εκτίμηση σημείου (point query), μέσω της διαδικασίας estimate. Το point query αναφέρεται στο αίτημα εύρεσης της συχνότητας ενός αντικειμένου της ροής. Θεωρώντας πως κάθε αντικείμενο i έχει θετική αξία, για να βρούμε την καλύτερη εκτίμηση της α_i θα πρέπει να βρούμε την θέση του i στον πίνακα με τη μικρότερη τιμή, δηλαδή την θέση του i που έχει τη μικρότερη (ή και καθόλου) υπερεκτίμηση δεδομένου, καθώς είναι πολύ πιθανό κάθε θέση του πίνακα να έχει υποστεί τουλάχιστον μία σύγκρουση. Επομένως η καλύτερη εκτίμηση ενός αντικειμένου i με αξία α_i είναι:

$$\widehat{a_i} = min_j count\big[\, j, h_j(i) \,\big]$$

Δηλαδή η point query βρίσκει τη μικρότερη τιμή σε κάθε συνάρτηση κατακερματισμού του πίνακα και στη συνέχεια τις συγκρίνει και επιστρέφει τη μικρότερη εξ αυτών. Σύμφωνα με τους Graham Cormode και S. Muthukrishnan θα παραθέσουμε την ίδια απόδειξη για την εκτίμηση αντικειμένων.

Ας υποθέσουμε ότι $\alpha_i \leq \widehat{a_i}$ με πιθανότητα τουλάχιστον δ ώστε να ισχύει $\widehat{a_i} \leq \alpha_i + \varepsilon||\alpha||_1$.

Δεδομένου ότι όλες οι τιμές των αντικειμένων είναι θετικές και σε συνδυασμό με τις συγκρούσεις που προκύπτουν στο πίνακα της CM, η σχέση $\alpha_i \leq \widehat{a_i}$ ισχύει πάντα.

Θέτουμε $\widehat{a_{i,j}} = count\left[j, h_j(i) \right]$ που είναι η εκτίμηση της α_i στη γραμμή j του πίνακα. Από τα παραπάνω ξέρουμε πως ισχύει $count\left[j, h_j(i) \right] = a_i + X_{i,j}$, όπου $X_{i,j}$ υπενθυμίζουμε πως είναι η ήδη υπάρχουσα τιμής στη θέση i,j. Συνεπώς η εκτίμηση είναι άθροισμα της κανονικής τιμής α_i και μιας τιμής σφάλματος που προκύπτει από τις διάφορες συγκρούσεις στο συγκεκριμένο σημείο του πίνακα. Προκειμένου να μπορέσουμε να αναλύσουμε καλύτερα το σφάλμα αυτό, χρειαζόμαστε έναν δείκτη που θα δείχνει αν έχει συμβεί σύγκρουση σε αυτό το σημείο. Θέτουμε το δείκτη $I_{i,j,k}$ με το k να είναι και αυτό αντικείμενο με $i \neq k$ με $I_{i,j,k} = 1 \leftrightarrow h_j(i) = h_j(k)$, αλλιώς $I_{i,j,k} = 0$.

Η εκτιμώμενη τιμή σφάλματος που προκύπτει, δηλαδή Ε($I_{i,j,k}$), θα είναι ίση με τη πιθανότητα το $I_{i,j,k}$ να είναι μηδέν. Επομένως έχουμε:

$$E\left(I_{i,j,k} \right) = P\left(h_j(i) = h_j(k) \right) = \frac{1}{range\left(h_j\right)} = \frac{1}{width} = \frac{1}{\frac{2}{\varepsilon}} = \frac{\varepsilon}{2}$$

Εφόσον ο δείκτης $I_{i,j,k}$ είναι είτε 1 είτε 0, μπορούμε να εκφράσουμε το σφάλμα $X_{i,j,k}$ως εξής:

$$X_{i,j,k} = \sum_{k=1}^{n} I_{i,j,k} \times a_k$$

Συνεπώς, η εκτιμώμενη τιμή σφάλματος σε μια θέση (i,j) είναι:

$$E\left(X_{i,j,k} \right) = \sum_{k=1}^{n} a_k \times \mathrm{E}\left(I_{i,j,k} \right) \leq ||\alpha||_1 \frac{\varepsilon}{2}$$

Η παραπάνω ανισότητα προκύπτει από την Markov's inequality και από την L1-norm.

Επαναλαμβάνουμε τη παραπάνω διαδικασία για ολόκληρη την εκτίμηση. Η μοναδική περίπτωση που θα πετύχουμε σφάλμα μεγαλύτερο του $\varepsilon||\alpha||_1$ θα είναι αν κάθε γραμμή έχει σφάλμα μεγαλύτερο του $\varepsilon||\alpha||_1$ εφόσον το αποτέλεσμα είναι η μικρότερη εκτίμηση από όλες τις γραμμές. Επομένως η πιθανότητα η εκτίμηση της a_i να είναι μεγαλύτερη από $a_i + \varepsilon||\alpha||_1$ είναι ίση με τη πιθανότητα να είναι το $X_{i,j,k} > \varepsilon||\alpha||_1$ για κάθε j. Άρα:

$$P\left(\widehat{a_i} > a_i + \varepsilon\left|\left|\alpha\right|\right|_1 \right) = P\left(\forall j: X_{i,j,k} > \varepsilon\left|\left|\alpha\right|\right|_1 \right)$$

Αναφέραμε στις Ενότητες 2.1 και 2.2 πως έχουμε συναρτήσεις κατακερματισμού που είναι ανεξάρτητες ανά ζεύγη και κάνουν ομοιόμορφη κατανομή των αντικειμένων σε κάθε γραμμή του πίνακα της Count-Min Sketch. Επειδή οι συναρτήσεις είναι ανεξάρτητες κατά ζεύγη μπορούμε να αντιστοιχήσουμε τη παραπάνω εξίσωση με:

$$P\left(\forall j: X_{i,j,k} > \varepsilon\left|\left|\alpha\right|\right|_1 \right) = P\left(\forall j: X_{i,j,k} > 2E\left(X_{i,j,k}\right) \right) < 2^{-depth} = 2^{-(-\log_2(1-\delta))} = 1 - \delta$$

Συνεπώς η πιθανότητα να έχει σφάλμα μεγαλύτερο του $\varepsilon\left|\left|\alpha\right|\right|_1$ είναι 1-δ.

Άρα η estimate καταφέρνει σφάλμα το πολύ $\varepsilon||\alpha||_1$ με πιθανότητα δ. Ο απαιτούμενος χρόνος για την εκτίμηση είναι $O(-\log_2(1-\delta))$ εφόσον η εύρεση του μικρότερου κόστους καθώς και η ενημέρωση του πίνακα γίνεται σε γραμμικό χρόνο. Ο συνολικός χώρος που χρησιμοποιείται είναι ίσος με $O(-\log_2(1-\delta))1/\varepsilon^2)$.

2.4 Εκτίμηση εσωτερικού γινομένου Count-Min Sketch

Η εκτίμηση εσωτερικού γινομένου (Inner Product query) εκτιμά το εσωτερικό γινόμενο του πίνακα α της CM με ένα δοθέν πίνακα b. Το εσωτερικό γινόμενο των δύο αυτών πινάκων είναι : $a \odot b = \sum_{i=1}^{n} a_i b_i$. Για την CM μας θεωρούμε πως οι τιμές στους πίνακες α, b είναι ακέραιες τιμές στο διάστημα [1, n]. Το a_i δείχνει πόσες τιμές υπάρχουν στον πίνακα α για το αντικείμενο i και αντίστοιχα το b_i πόσες τιμές υπάρχουν για το αντικείμενο i στο πίνακα b. Με τη χρήση σκίτσων μπορούμε να υπολογίσουμε την ένωση των α, b ενόσω ενημερώνεται συνεχώς ο πίνακας.

Για τις εκτιμήσεις μας θεωρούμε δύο CM οι οποίες έχουν τις ίδιες τιμές (ε, δ), συνεπώς έχουν ίδιο μέγεθος πινάκων καθώς $width_a = width_b$ και$depth_a = depth_b$ με σκίτσα $count_a$ και $count_b$. Το $count_a$ αντιστοιχεί στον πίνακα της πρώτης CM, ενώ το $count_b$ στο πίνακα της δεύτερης. Η εκτίμηση του εσωτερικού γινομένου θα γίνει με την ίδια λογική που χρησιμοποιήσαμε στην point query. Δηλαδή υπολογίζουμε το εσωτερικό γινόμενο για το αντικείμενο i σε κάθε γραμμή των δύο πινάκων και στη συνέχεια επιστρέφουμε το μικρότερο εσωτερικό γινόμενο. Δηλαδή:

$$\widehat{a \odot b} = min_j \sum_{k=1}^{n} count_a(j,k) \times count_b(j,k)$$

Θα παραθέσουμε την εκτίμηση $\widehat{a \odot b}$ από τους Graham Cormode και S. Muthukrishnan όπως παραθέσαμε την εκτίμηση στην point query.

Υποθέτουμε ότι για την $\widehat{a \odot b}$ ισχύουν:

- $a \odot b \leq \widehat{a \odot b}$
- $\widehat{a \odot b} \leq a \odot b + \varepsilon||\alpha||_1||b||_1$ με πιθανότητα δ.

Όμοια με την point query, το $a \odot b \leq \widehat{a \odot b}$ ισχύει πάντα καθώς οι τιμές είναι μη αρνητικές και υπάρχουν συγκρούσεις. Συνεπώς μπορεί να γίνει ακριβής εκτίμηση ή υπερεκτίμηση, αλλά ποτέ υποτίμηση. Έστω δύο αντικείμενα x, y με $x \neq y$. Το εκτιμώμενο εσωτερικό γινόμενο για μία μόνο γραμμή είναι $(\widehat{a \odot b})_j = a \odot b + X_j$,

όπου $X_j = \sum_{h(x)=h(y)} a_x\, b_y$. Χρησιμοποιούμε εκ νέου το δείκτη $I_{i,j,k}$ που χρησιμοποιήσαμε στο κεφάλαιο 2.4 και προκύπτει με αντικατάσταση ότι $X_j = \sum_{(x,y)} I_{i,j,k} a_x b_y$. Επομένως η αναμενόμενη τιμή του X_j είναι :

$$E(X_j) = E\left(\sum_{(x,y)} I_{i,j,k} a_x b_y\right) = \sum_{(x,y)} a_x b_y E(I_{i,j,k}) \leq \sum_{(x,y)} a_x b_y \frac{\varepsilon}{2}$$

Με χρήση της Markov's Inequality και της L1-norm μπορούμε να αντικαταστήσουμε το $\sum_{(x,y)} a_x b_y \frac{2}{\varepsilon}$ με:

$$\sum_{(x,y)} a_x b_y \frac{\varepsilon}{2} = \frac{\varepsilon}{2} \sum_{x=1}^{n} \sum_{y=1}^{n} a_x b_y = \frac{\varepsilon}{2} \sum_{x=1}^{n} ax \sum_{y=1}^{n} b_y = \frac{\varepsilon}{2} ||a||_1 ||b||_1$$

Έχοντας υπολογίσει το εκτιμώμενο σφάλμα για μια γραμμή μπορούμε να υπολογίσουμε τη πιθανότητα η εκτίμηση του εσωτερικού γινομένου για όλες τις γραμμές να είναι μεγαλύτερη από το $\frac{\varepsilon}{2} ||a||_1 ||b||_1$.

$$P\left(\forall j: X_{i,j,k} > \varepsilon \left|\left|\alpha\right|\right|_1 \left|\left|b\right|\right|_1\right) = P\left(\forall j: X_{i,j,k} > 2E(X_{i,j,k})\right) < 2^{-depth} = 2^{-(-\log_2(1-\delta))}$$

$$P\left(\forall j: X_{i,j,k} > \varepsilon \left|\left|\alpha\right|\right|_1 ||b||_1\right) = 1 - \delta$$

Συνεπώς η εκτίμηση εσωτερικού γινομένου έχει πιθανότητα το πολύ 1-δ για να είναι μεγαλύτερη της $a \odot b + \varepsilon ||\alpha||_1 ||b||_1$, οπότε πετυχαίνει σφάλμα το πολύ ε με πιθανότητα δ.

Ο χρόνος εκτίμησης καθώς και το κόστος αποθηκευτικού χώρου είναι $O\left(-\log_2(1-\delta)\frac{1}{\varepsilon}\right)$ που ισχύει εφόσον τα δύο σκίτσα έχουν ίδιο μέγεθος με $width = \left\lceil \frac{2}{\varepsilon} \right\rceil$, $depth = \lceil -\log_2(1-\delta) \rceil$. Η πολυπλοκότητα χώρου και χρόνου ταυτίζονται καθώς πρέπει να ελέγξουμε κάθε είσοδο του σκίτσου κατά τη διάρκεια του υπολογισμού.

2.5 Εκτίμηση εύρους διαστημάτων

Η εκτίμηση εύρους διαστημάτων, ή αλλιώς Range Query, υπολογίζει το άθροισμα της αξίας των αντικείμενων σε ένα διάστημα. Με τιμές r και l υπολογίζεται το άθροισμα $\sum_{i=l}^{r} a_i$. Υποθέτουμε πως ένα σκίτσο αποθηκεύει 10000 καθημερινούς χρήστες μιας εφαρμογής,. Αν η σειρά είναι τυχαία τότε το διάστημα [500,3000] θα μας επιστρέψει τυχαίους χρήστες που βρίσκονται στο διάστημα αυτό. Αν όμως, οι χρήστες έχουν τοποθετηθεί σε σειρά με βάση τη γεωγραφική τους τοποθεσία και το διάστημα [500,3000] περιέχει όλους τους χρήστες από την Ευρώπη, τότε η range query για το

διάστημα αυτό θα επιστρέψει όλους τους χρήστες της εφαρμογής που βρίσκονται στην Ευρώπη.

Η εκτίμηση τιμής ενός διαστήματα [l,r] μπορεί να υλοποιηθεί με τρεις μεθόδους. Η πρώτη, και πιο απλή, μέθοδος είναι να πάρουμε το άθροισμα των εκτιμήσεων σημείου από point query για όλα τα στοιχεία που ανήκουν στο διάστημα [l,r].

$$a_{[l,r]} = \sum_{i=l}^{r} a_i$$

Παρά την απλότητα της παραπάνω μεθόδου, παρουσιάζονται προβλήματα, ειδικότερα σε μεγάλα διαστήματα. Η διαδικασία αυτή είναι χρονοβόρα, ειδικά για μεγάλα δεδομένα χωρίς να είναι ιδιαίτερα αποδοτική. Πιο συγκεκριμένα, ο παράγοντας σφάλματος αυξάνεται γραμμικά με την έκταση του διαστήματος. Οπότε έχουμε αποτελεσματικό παράγοντα ίσο με $r - l + 1$. Το κόστος χρόνου εξαρτάται άμεσα από το μέγεθος του διαστήματος r-l.

Η δεύτερη μέθοδος για την επίλυση αυτού του προβλήματος είναι η εκτίμηση εσωτερικού γινομένου. Για την εκτίμηση αυτή, σε συνδυασμό με τον αρχικό μας πίνακα α θα χρειαστεί να δημιουργήσουμε έναν πίνακα x με στοιχεία x_i σε κάθε γραμμή, όπου $x_i = 1$ για $l \leq i \leq r$ και $x_i = 0$ διαφορετικά. Αν υπολογίσουμε το εσωτερικό γινόμενο των σκίτσων α και x θα μας επιστρέψει ακριβώς την εκτίμηση αξίας στο διάστημα [l,r].

Η τρίτη και αποτελεσματικότερη μέθοδος είναι η χρήση δυαδικών διαστημάτων. Τα δυαδικά διαστήματα μας βοηθούν να αποκλείσουμε την γραμμική αύξηση του σφάλματος ανάλογα με το μέγεθος του πίνακα. Για να αυξηθεί λογαριθμικά και όχι γραμμικά το σφάλμα χρειαζόμαστε τα δυαδικά διαστήματα.

Για $x \epsilon N, y \epsilon N_0$ ένα δυαδικό διάστημα είναι ένα μεσοδιάστημα της μορφής:

$$D_{x,y} \coloneqq [x2^y + 1, (x+1)2^y]$$

Κάθε μεσοδιάστημα μεγέθους n μπορεί να καλυφθεί με χρήση το πολύ $\lceil 2 \log n \rceil$ μη επικαλυπτόμενα δυαδικά διαστήματα. Αυτό συμβαίνει καθώς για ένα διάστημα w=[l,r] ένα δέντρο δυαδικών διαστημάτων έχει ύψος $\log(n) + 1$ και σε κάθε ύψος του δέντρου υπάρχουν το πολύ 2 δυαδικά διαστήματα σε ένα ελάχιστο κάλυμμα δυαδικής πληθικότητας C που να καλύπτει όλο το w.

Θεωρούμε πως n είναι δύναμη του 2. Στα δυαδικά διαστήματα χρησιμοποιούμε $\log(n) + 1$ σκίτσα. Κάθε σκίτσο αναπαριστά ένα ύψος του δέντρου δυαδικών διαστημάτων του διαστήματος [1,n]. Όλα τα σκίτσα μοιράζονται τις ίδιες παραμέτρους ε, δ και τις ίδιες συναρτήσεις κατακερματισμού. Για να κάνουμε ενημέρωση update(i, c) βρίσκουμε στο δέντρο το $D_{x,y}$ με $i \epsilon D_{x,y}$ και έπειτα χρησιμοποιούμε το x στις συναρτήσεις κατακερματισμού προκειμένου να κάνουμε την ενημέρωση update(x,c) στο $count_y$. Με

τη διαδικασία αυτή μπορούμε να κάνουμε εκτίμηση σημείου στα δυαδικά διαστήματα. Θέτουμε α[l,r] το πραγματικό αποτέλεσμα του διαστήματος [l,r] και $\widehat{a[l,r]}$ το εκτιμώμενο αποτέλεσμα. Σύμφωνα με τη παραπάνω διαδικασία θα λάβουμε:

$$\widehat{a[l,r]} \coloneqq \sum_{D_{x,y}\epsilon C} min_j count_y\big(j, h_j(x)\big)$$

Παρόμοια με τη point query και inner product query θα ακολουθήσουμε την ίδια απόδειξη. Θεωρούμε πως $\alpha[l,r] \leq \widehat{a[l,r]}$ και $\widehat{a[l,r]} \leq \alpha[l,r] + 2\varepsilon \log(n)\,||a||_1$ με πιθανότητα δ.

Το $\alpha[l,r] \leq \widehat{a[l,r]}$ ισχύει πάντα εφόσον τα δεδομένα μας έχουν μη αρνητικές τιμές. Λόγω συγκρούσεων η εκτίμηση μπορεί να είναι είτε ίση είτε μεγαλύτερη της πραγματικής τιμής, αλλά όχι μικρότερη. Για να βρούμε το σφάλμα της εκτιμώμενης τιμής θέτουμε $X_{y,j,x}$ το σφάλμα που ήδη υπάρχει στο σκίτσο y στη συνάρτηση κατακερματισμού j για το αντικείμενο x.

$$count_y\big(j, h_j(x)\big) = \sum_{i\epsilon D_{x,y}} a_i + X_{y,j,x}$$

Θα χρειαστούμε το δείκτη $I_{i,j,k}$ που έχουμε για στις Ενότητες 2.3, 2.4. Ολόκληρο το δυαδικό μεσοδιάστημα κάθε στοιχείου k για $h_j(x) = h_j(k)$ αποτελεί κομμάτι του σφάλματος αν παρατηρηθεί σύγκρουση. Οπότε:

$$X_{y,j,x} = \sum_{k=1}^{n} I_{i,j,k} \sum_{i\epsilon D_{k,y}} a_i$$

Από τη παραπάνω σχέση και με τη βοήθεια της Markov's inequality αλλά και των αποδείξεων στα προηγούμενα κεφάλαια βγαίνει η αναμενόμενη τιμή του $X_{y,j,x}$.

$$E\big(X_{y,j,x}\big) = \sum_{k=1}^{n} I_{i,j,k} \sum_{i\epsilon D_{k,y}} a_i \leq \frac{\varepsilon}{2}||\alpha||_1$$

Για λόγους απλοποίησης υποθέτουμε πως όλες οι εκτιμήσεις μπορούν να απαντηθούν με τη χρήση μίας μόνον γραμμής j.

$$\widehat{a[l,r]} \coloneqq \sum_{D_{x,y}\epsilon C} min_j count_y\big(j, h_j(x)\big) \leq \widehat{a[l,r]} \coloneqq min_j \sum_{D_{x,y}\epsilon C} count_y\big(j, h_j(x)\big)$$

Έτσι έχουμε: $X_j = \sum_{D_{x,y}\epsilon C} X_{y,k,x}$. Οπότε $E\big(X_j\big) = E\left(\sum_{D_{x,y}\epsilon C} X_{y,k,x}\right) \leq \log(n)\varepsilon||\alpha||_1$ αφού έχουμε το πολύ $2\log(n)$ μη επικαλυπτόμενα δυαδικά διαστήματα. Επειδή οι συναρτήσεις κατακερματισμού είναι ανεξάρτητες μεταξύ τους μπορούμε να θεωρήσουμε πως και τα αποτελέσματα τους είναι ανεξάρτητα. Οπότε η πιθανότητα σφάλματος για όλες τις γραμμές σε ένα σκίτσο είναι ίσο με το γινόμενο της πιθανότητας η μία γραμμή να είναι μεγαλύτερη του $2\log(n)\,\epsilon\big|\big|a\big|\big|_1$:

$$P\left(min_j X_j > 2\log(n)\,\epsilon||a||_1\right) \leq \prod_{i=1}^{depth} \frac{\log(n)\varepsilon||\alpha||_1}{2\log(n)\varepsilon||\alpha||_1} \leq \prod_{i=1}^{depth} 0.5 \leq 1-\delta$$

Λόγω Markov's inequality και ανεξαρτησίας δεδομένων προκύπτει ότι η πιθανότητα να έχουμε σφάλμα μεγαλύτερο από $2\log(n)\varepsilon||\alpha||_1$ είναι 1-δ. Αντίστοιχα η πιθανότητα να πετύχουμε σφάλμα το πολύ $2\log(n)\varepsilon||\alpha||_1$ είναι δ.

Συμπεραίνουμε πως η χρονική και χωρική πολυπλοκότητα είναι υπογραμμικές. Η πολυπλοκότητα χρόνου για εκτίμηση στοιχείου ή ενημέρωση είναι $O(-\log(1-\delta)\log(n))$ εφόσον χρειαζόμαστε $\log(n)+1$ σκίτσα, ενώ η χωρική πολυπλοκότητα είναι $O\left(-\log(1-\delta)\log(n)\frac{1}{\varepsilon}\right)$. Η απαιτούμενη πολυπλοκότητα χώρου μόνο για τις συναρτήσεις κατακερματισμού είναι $O(\log(n))$. [4]

2.5.1 Εφαρμογές Range Query: φ-Quantiles

Μία από τις λειτουργίες της Range query είναι η εύρεση φ-Quantile σε ένα σύνολο δεδομένων. Το φ-Quantile, με $0 \leq \varphi \leq 1$ ενός συνόλου δεδομένων που περιέχει N αντικείμενα, είναι ένα αντικείμενο x τέτοιο ώστε να υπάρχουν φN μικρότερα του x και $(1-\varphi)N$ μεγαλύτερα του x. Υπάρχουν δύο μέθοδοι για την εύρεση του. Η πρώτη μέθοδος, χρησιμοποιεί τυχαία υποσύνολα αθροισμάτων ενώ η δεύτερη μέθοδος χρησιμοποιεί δυαδική αναζήτηση. Εμείς θα ασχοληθούμε με τη δεύτερη μέθοδο. Παρόλα αυτά αξίζει να σημειωθεί πως και οι δύο μέθοδοι αποφέρουν παρόμοια χρονική απόδοση. Θεωρούμε μη αρνητικές τιμές. Χρησιμοποιώντας το φ-Quantile μπορούμε να μετατρέψουμε τις εισόδους μιας CM σε range queries. Εκτελούμε δυαδικές αναζητήσεις στα διαστήματα [1,r] όπου $\alpha[1,r] > k\varphi||a||_1$, με $1 \leq k \leq \frac{1}{\varphi} - 1$. Έτσι μπορούμε να υπολογίσουμε τα ε-προσεγγιστικά φ-Quantiles με πιθανότητα τουλάχιστον δ. Η εύρεση φ-Quantile απλοποιείται σε log(n) points queries.

2.5.2 Εφαρμογές Range Query: Heavy Hitters

Οι heavy hitters ή αλλιώς φ-heavy hitters ενός συνόλου δεδομένων είναι τα αντικείμενα ή γεγονότα, δηλαδή μια συγκεκριμένη ακολουθία αντικειμένων όπως η φράση «για να» στο γραπτό ή προφορικό λόγο, με τις μεγαλύτερες συχνότητες. Οι φ-heavy hitters είναι ακέραια αντικείμενα στο διάστημα [1,n] τα οποία αποτελούν φ ποσοστό της συνολικής αξίας. Δηλαδή ένα αντικείμενο x είναι φ-heavy hitter αν ισχύει ότι: $a_x \geq \varphi||a||_1$. Σε ένα

σύνολο δεδομένων υπάρχουν από 0 έως και $\frac{1}{\varphi}$ heavy hitters. Η εκτίμηση ενός τέτοιου αντικειμένου θα ισχύει αν $\hat{a}_i \geq (\varphi + \varepsilon)\left|\left|a\right|\right|_1$ με $\varepsilon < \varphi$.

Διατηρούμε log(n) σκίτσα για τα log(n) δυαδικά διαστήματα. Εκτελούμε δυαδική αναζήτηση στο δέντρο που περιέχει τα δυαδικά διαστήματα και βρίσκουμε την εκτίμηση του κάθε αντικειμένου με χρήση της διαδικασίας που περιγράψαμε στη Range Query. Σε κάθε ένα από τα log(n) επίπεδα του δέντρου υπάρχουν το πολύ $\frac{2}{\varphi}$ αντικείμενα που είναι heavy hitters. Αν η εκτιμώμενη συχνότητα ενός αντικειμένου είναι τουλάχιστον $(\varphi + \varepsilon)||\alpha||_1$ τότε το αντικείμενο αυτό είναι αποτέλεσμα της heavy hitters. Αντίστοιχα αν η εκτιμώμενη συχνότητα ενός αντικειμένου είναι μικρότερη του $(\varphi + \varepsilon)\left|\left|\alpha\right|\right|_1 - \varepsilon\left|\left|\alpha\right|\right|_1 = \varphi||\alpha||_1$ το αντικείμενο δεν αποτελεί αποτέλεσμα της heavy hitters. Η παραπάνω διαδικασία αποδίδει με πιθανότητα δ, αντικείμενα με εκτιμώμενη συχνότητα μεγαλύτερη ή ίση με $\varphi||\alpha||_1$. Επιπλέον κάθε αντικείμενο με συχνότητα $(\varphi + \varepsilon)\left|\left|\alpha\right|\right|_1$ είναι αποτέλεσμα της heavy hitters.

Ο αλγόριθμος για την εύρεση των heavy hitters χρησιμοποιεί χώρο $O\left(-\log\left(\frac{2\log(n)(1-\delta)}{\varphi}\right)\log(n)\frac{1}{\varepsilon}\right)$ και χρόνο $O\left(-\log\left(\frac{2\log(n)(1-\delta)}{\varphi}\right)\log(n)\right)$.

Κεφάλαιο 3. Υλοποίηση

Στο κεφάλαιο αυτό θα αναλύσουμε την υλοποίηση μας για την Count-Min Sketch. Η γλώσσα προγραμματισμού που χρησιμοποιήσαμε είναι η C++ 14 και έκδοση μεταγλωττιστή g++ 9.3.0. Για να υλοποιήσουμε τη Count-Min Sketch στηριχτήκαμε στις τεχνικές που αναλύσαμε στο κεφάλαιο 2. Η υλοποίηση μας απαρτίζεται από 5 αρχεία. Στη συνέχεια θα αναφερθούμε στα αρχεία αυτά καθώς και σε κάποιες από τις συναρτήσεις που υπάρχουν οι οποίες παίζουν καθοριστικό ρόλο για την εκτέλεση των παραπάνω τεχνικών κώδικα.

3.1 Αρχεία Header

Τα αρχεία header είναι αρχεία που περιέχουν δηλώσεις συναρτήσεις αλλά και ορισμούς global μεταβλητών. Εμείς δημιουργήσαμε δύο τέτοια αρχεία που περιέχουν τις απαραίτητες βιβλιοθήκες για την Count-Min Sketch, *cmsketch.h* και *rangecms.h*. Το αρχείο *cmsketch.h* περιέχει τις βιβλιοθήκες για την υλοποίηση του σκίτσου για την επίλυση των Point Queries και Inner Product Queries, ενώ το *rangecms.h* περιέχει τις βιβλιοθήκες που χρειάζονται για την επίλυση των Range Queries και των δύο προβλημάτων που αναφέραμε στις Ενότητες 2.5.1 και 2.5.2, φ-Quantiles και heavy hitters. Τα δύο παραπάνω αρχεία περιέχουν και τους ορισμούς των global μεταβλητών αλλά και τις δηλώσεις όλων των συναρτήσεων που θα χρησιμοποιήσουμε.

3.2 Υλοποίηση σκίτσου και ανάγνωση αρχείων

Για την υλοποίηση του σκίτσου, παρέχουμε τη δυνατότητα στο χρήστη να επιλέξει τη τιμή του βάθους, του πλάτους και της συνάρτησης κατακερματισμού του σκίτσου. Το

πρόγραμμα δέχεται ως είσοδο τις μεταβλητές ε, δ και έναν ακέραιο στο διάστημα [0, 2] για την επιλογή της συνάρτησης κατακερματισμού.

Το σκίτσο μας μπορεί να αποθηκεύσει σύνολα δεδομένων τύπου BMS, SPMF, Kosarak και απλά αρχεία κειμένου, όπως βιβλία ή αναφορές. Προκειμένου να μπορέσουμε να διαβάσουμε τα αρχεία και να μπορούμε να τα αποθηκεύσουμε στη Count-Min Sketch δημιουργήσαμε βοηθητικές συναρτήσεις για την ενημέρωση και την εκτίμηση του κάθε αντικειμένου που αντικρίζουμε στο αρχείο εισόδου. Δηλαδή για κάθε είδος αρχείου έχουμε δύο συναρτήσεις, μία για ενημέρωση του σκίτσου και μία για την εκτίμηση κάθε αντικειμένου. Σε κάθε συνάρτηση αποθηκεύουμε τις ακριβείς τιμές του κάθε αντικειμένου σε μια λίστα, ή σε ένα χάρτη αν το αρχείο εισόδου είναι ένα βιβλίο, για ευκολία και για εξοικονόμηση χρόνου στην εκτίμηση αντικειμένων χωρίς να χρειαστεί να διαβάσουμε εκ νέου το αρχείο.

3.2.1 Βοηθητικές συναρτήσεις ανάγνωσης αρχείου

Όπως αναφέραμε παραπάνω υλοποιήσαμε βοηθητικές συναρτήσεις προκειμένου να ενημερώσουμε το πίνακα της Count-Min Sketch. Κάθε τύπος συνόλου αρχείου είναι μοναδικός με συγκεκριμένους κανόνες τους οποίους πρέπει να υπολογίσουμε κατά τη διάρκεια της ενημέρωσης του σκίτσου. Το σύνολο δεδομένων BMS έχει γραμμές της μορφής: *id value*, όπου id είναι το αντικείμενο και value είναι η αξία του αντικειμένου. Ένα αρχείο της μορφής BMS δε χρειάζεται καμία τροποποίηση προκειμένου να μπορέσουμε να το αποθηκεύσουμε στο σκίτσο δεδομένου ότι η Count-Min Sketch δέχεται ως είσοδο ένα αντικείμενο και μια τιμή που αντιστοιχεί στην αξία του.

Το σύνολο δεδομένων SPMF έχει γραμμές της μορφής: $value_1$ *-1* $value_2$ *-1 .... -1* $value_i$ *-1 -2,* με $i \in \mathbb{N}_0$. Η κάθε γραμμή είναι μια ακολουθία με το -1 να δηλώνει πως υπάρχει και άλλη τιμή για την ακολουθία αυτή ενώ το -1 -2 δηλώνει το τέλος της ακολουθίας. Μια γραμμή SPMF αντιστοιχεί σε i συνεχόμενες γραμμές BMS με ένα μόνο id το οποίο έχει διαφορετικές τιμές σε κάθε γραμμή. Δηλαδή η πρώτη γραμμή 1 -1 2 -1 3 -1 4 -1 5 -1 -2 αντιστοιχεί στις γραμμές:

1 1 , 1 2 , 1 3 , 1 4 , 1 5

Συνεπώς για μια γραμμή id σε σύνολο δεδομένων της μορφής SPMF εκτελούμε i ενημερώσεις πίνακα, όπου κάθε ενημέρωση έχει το ίδιο id και διαφορετική τιμή.

Παρόμοια με το σύνολο SPMF υλοποιούμε και την βοηθητική συνάρτηση για την αποθήκευση συνόλου δεδομένων μορφής Kosarak. Σε αντίθεση με τη μορφή SPMF, η

Kosarak έχει γραμμές που περιέχουν μόνο τις τιμές της κάθε ακολουθίας και έχει μορφή: $value_1\ value_2\ \dots\ value_i$ με i $\in \mathbb{N}_0$. Συνεπώς, η ενημέρωση στο σκίτσο είναι ίδια με την SPMF εφόσον εκτελούμε i ενημερώσεις για κάθε id, με κάθε ενημέρωση να έχει διαφορετική τιμή.

```
void KosarakAddItem( char* filename, CMSketch* item1, CMSketch* item2, RangeCMSketch* item3)
{

    std::ifstream afile(filename);
    std::string line; // the line extracted from the file
    int id = 0; // stores the number we hash
    int itemset; // stores the times the number appears
    unsigned int counter; // stores the sum of values in each line

    // read the lines and add to query
    while (std::getline(afile,line))
    {
        counter = 0;
        stringstream content(line);
        while( content >> itemset )
        {
            counter += itemset;
            item1->addItem( id, itemset );
            item2->addItem( id, itemset );
            item3->addRangeItem( id, itemset );
        }

        counts[id] = counter;
        id++;
    }

    afile.close();

}
```

Εικόνα 3.3.2-1. Ενημέρωση πίνακα με αρχείο μορφής Kosarak

Για την ανάγνωση αρχείων με αλφαριθμητικά, χρησιμοποιήσαμε δύο συναρτήσεις, η πρώτη για το χωρισμό λέξεων αν αυτές περιέχουν ή ακολουθούνται αμέσως από σύμβολα, δηλαδή η λέξη « result.» θα γίνει «result». Η συνάρτηση αυτή είναι ιδιαίτερα χρήσιμη καθώς το πρόγραμμα μπορεί να θεωρήσει τις δύο παραπάνω λέξεις ως διαφορετικές, με αποτέλεσμα να έχουμε λάθος εκτιμήσεις. Η δεύτερη συνάρτηση είναι μια συνάρτηση καταμέτρησης λέξεων η οποία αποθηκεύει τις ακριβείς εμφανίσεις της κάθε λέξης.

```cpp
void split(const std::string &s, const std::regex &r, std::vector<std::string> &v)
{
    auto rit = std::cregex_token_iterator(s.data(), s.data() + s.length(), r, -1);
    auto rend = std::cregex_token_iterator();
    v.clear();
    while(rit != rend)
    {
        v.emplace_back(rit->first, rit->second);
        ++rit;
    }
}
```

Εικόνα 3.2. Χωρισμός λέξεων που περιέχουν σύμβολα

```cpp
void WordCount( char* filename )
{
    std::ifstream afile(filename);
    std::string word;
    std::regex re("[\\!,;:\\-._?\\[]");
    std::vector<std::string> v;
    int id = 0; // used to given identification number to each unique word
    // read the lines and add to query

    while ( afile >> word )
    {
        split( word, re, v );
        for ( auto t : v )
        {
            ++words[t];
            if( idholder.find(t) == idholder.end() )
            {
                idholder[t] = id;
                id++;
            }
        }
    }

    afile.close();
}
```

Εικόνα 3.3. Καταμέτρηση λέξεων

3.3 Υλοποίηση Update, Point και Inner Product Query

Η Count-Min Sketch χωρίζεται σε δύο αρχεία για περισσότερη κατανόηση του κώδικα. Το πρώτο αρχείο είναι το *cmsketch.cpp* που αφορά την υλοποίηση των Point Queries

και Inner Product Queries. Η *cmsketch* περιέχει συγκεκριμένα έναν copy constructor για τη δημιουργία μιας ίδιας CM για την επίλυση του Inner Product Query. Με τη βοήθεια του copy constructor δημιουργούμε ένα ακόμα σκίτσο που έχει τις ίδιες ακριβώς παραμέτρους (ε,δ), συνεπώς και width και depth, με το αρχικό σκίτσο.

3.3.1 Υλοποίηση update και κατακερματισμού

Η ενημέρωση του πίνακα αποτελείται από 2 συναρτήσεις και βασίζεται σε μια συνάρτηση hash για τη χρήση συναρτήσεων κατακερματισμού. Έχουμε υλοποιήσει δύο συναρτήσεις addItem οι οποίες ενημερώνουν τον πίνακα. Η μία συνάρτηση είναι για τα ακέραια αντικείμενα ενώ η άλλη είναι για τα αλφαριθμητικά. Αν το αντικείμενο είναι αλφαριθμητικό χρησιμοποιούμε πρώτα την συνάρτηση της C++: std::hash<std::string>{}(αντικείμενο). Η συνάρτηση αυτή μετατρέπει το αλφαριθμητικό σε hash το οποίο μπορούμε να χρησιμοποιήσουμε ύστερα στις συναρτήσεις κατακερματισμού, καθώς η συνάρτηση hash δέχεται αντικείμενα τύπου std::size_t.

Σε κάθε ενημέρωση της CM κρατάμε την ακριβή συνολική αξία της ροής δεδομένων προκειμένου να ελέγξουμε την εγκυρότητα του προγράμματος μας. Στις συναρτήσεις κατακερματισμού μας, χρησιμοποιούμε 3 μεταβλητές, a b και p. Οι μεταβλητές a και b είναι τυχαίοι αριθμοί επιλεγμένοι από τον p, ο οποίος είναι ένας πολύ μεγάλος πρώτος αριθμός, πιο συγκεκριμένα ίσος με 4294967311l. Η μεταβλητή a ανήκει στο διάστημα [1,p-1], ενώ η μεταβλητή b ανήκει στο [0,p-1].

Το πρόγραμμα παρέχει τρεις οικογένειες συναρτήσεων κατακερματισμού τις οποίες μπορεί να διαλέξει ο χρήστης. Η συνάρτηση κατακερματισμού επιλέγεται αναλόγως με τη μεταβλητή mode.

- Αν mode = 0, τότε η συνάρτηση κατακερματισμού έχει τη μορφή: $(ax + b)\ mod\ width$
- Αν mode = 1, τότε η συνάρτηση κατακερματισμού έχει τη μορφή της γνωστής, ανεξάρτητη ανά ζεύγη συνάρτησης κατακερματισμού: $((ax + b)\ mod\ p)\ mod\ width$
- Αν mode = 2, η συνάρτηση κατακερματισμού είναι η συνάρτηση της mode = 1 με έναν επιπλέον προσαρμοσμένο παράγοντα: $\Big(\big((ax + b) + ((ax + b) \gg 32)\big)\ mod\ p\Big)\ mod\ w$

```
unsigned int CMSketch :: hash ( std::size_t item, int i )
{
    std::size_t hash_item;
    if ( mode == 0 )
        hash_item = ( ( long(hashAB[i][0])*item )
        + hashAB[i][1] ) % width;
    else if( mode == 1 )
        hash_item = ( ( ( long(hashAB[i][0]) * item )
        + hashAB[i][1] ) % PRIME ) % width;
    else if( mode == 2 )
    {
        hash_item = (long(hashAB[i][0])*item) + hashAB[i][1];
        hash_item += hash_item >> 32;
        hash_item %= PRIME;
        hash_item = (hash_item)  % width;
    }

    return (unsigned int) hash_item;
}
```

Εικόνα 3.3-1.4. Επιλογή και εκτέλεση συναρτήσεων κατακερματισμού

3.3.2 Υλοποίηση εκτίμησης σημείου

Για την εκτίμηση σημείου ακολουθήσαμε τις ίδιες τακτικές με την ενημέρωση σκίτσου προκειμένου να μπορούμε να αποθηκεύσουμε περισσότερα είδη αρχείων. Για αυτό την επίτευξη του σκοπού μας, έχουμε υλοποιήσει δύο συναρτήσεις estimatePoint για την επίλυση της Point query όπου η πρώτη εκτιμά ακέραια αντικείμενα και η δεύτερη αλφαριθμητικά. Η υλοποίηση μας εκτελεί την ανάλυση της Ενότητας 2.3 για την εκτίμηση αντικειμένων.

```
std::size_t CMSketch :: estimatePoint ( string item )
{

    std::size_t hash_item = std::hash<std::string>{}(item);
    int i;
    int estimate = RAND_MAX;

    for ( i = 0; i < depth; i++ )
    {

        estimate = MIN( estimate, table[i][hash( hash_item, i )] );

    }

    // should be estimate < actual_appearance + epsilon*N,
    // where N is the sum of appearances of the inputs
    return estimate;

}
```

Εικόνα 3.5. Εκτίμηση string αντικειμένων

3.3.3 Υλοποίηση εκτίμησης εσωτερικού γινομένου

Η υλοποίηση εσωτερικού γινομένου έγινε με βάση την ανάλυση στην Ενότητα 2.4. Η μετάφραση της ανάλυσης αυτής είναι συνάρτηση estimateInnerProduct η οποία δέχεται εισόδους τις τιμές width, depth καθώς και ένα δυσδιάστατο πίνακα. Ο πίνακας αυτός πρέπει να έχει ίδιο μέγεθος με τον πίνακα του σκίτσου μας προκειμένου να μπορέσουμε να βρούμε το εσωτερικό γινόμενο. Αν είναι ίδιου μεγέθους τότε υπολογίζουμε το εσωτερικό γινόμενο.

```
std::size_t CMSketch :: estimateInnerProduct ( int w, int d, std::size_t** new_table )
{
    int i, j;
    int temp;
    std::size_t inner = SIZE_MAX;

    // check if the two CMSketches are the same ( in terms of width and depth )
    if ( width != w || depth != d )
    {
        cout << "The two Count-Min Sketches are not compatible. " << endl;
        return 0;
    }

    for ( j = 0; j < depth; j++ )
    {
        temp = 0;
        // std::inner_product exists in <numeric>
        for ( i = 0; i < ((int)width); i++ )
            temp += ( table[j][i] *  new_table[j][i]);

        inner = MIN( inner, temp );
    }

    return inner;
}
```

Εικόνα 3.6. Υλοποίηση εσωτερικού γινομένου δύο Count-Min Sketches

3.4 Υλοποίηση Range Query, φ-Quantiles και Heavy Hitters

Το δεύτερο αρχείο για την Count-Min Sketch είναι το αρχείο *rangecms.cpp*. Το αρχείο αυτό υλοποιεί την Range Query και τα προβλήματα φ-Quantiles και heavy hitters που αυτή λύνει. Λόγω της πολυπλοκότητας των διαδικασιών για τα παραπάνω, το αρχείο αυτό περιέχει πολλές βοηθητικές συναρτήσεις για να μπορέσουμε να εκτελέσουμε τις διαδικασίες διατηρώντας έναν καθαρό και ευανάγνωστο κώδικα.

3.4.1 Range Query

Στη δημιουργία του δέντρου, επιλέγουμε ένα επίπεδο στο οποίο θα διατηρούμε ακριβείς τιμές για τα αντικείμενα όταν ενημερώνουμε το πίνακα. Δηλαδή η update από το επιλεχθέν επίπεδο και πάνω κρατά ακριβείς ενώ πριν από αυτό λειτουργεί κανονικά όπως η διαδικασία update που περιγράψαμε στην Ενότητα 2.2. Οι συναρτήσεις

κατακερματισμού που χρησιμοποιούμε στο *rangecms.cpp* είναι ίδιες με τις συναρτήσεις που χρησιμοποιούνται στο *cmsketch.cpp*.

Για να μπορέσουμε να εκτελέσουμε τη διαδικασία της Range Query πρέπει αρχικά να δημιουργήσουμε την point query για το δέντρο δυαδικών διαστημάτων. Η συνάρτηση αυτή είναι η estimateCR η οποία βρίσκει την εκτιμώμενη τιμή του αντικειμένου σε ένα συγκεκριμένο επίπεδο. Λειτουργεί όπως η κανονική point query με τη μόνη διαφορά να είναι πως εκτιμά τη τιμή ενός αντικειμένου για ένα μόνο επίπεδο του δέντρου.

Έχοντας υλοποιήσει την estimateCR, μπορούμε να υλοποιήσουμε τη διαδικασία της Range Query όπως περιγράψαμε στην Ενότητα 2.5 με τη συνάρτηση estimateRange.

```
std::size_t RangeCMSketch :: estimateCR ( int current, string item )
{

    int j;
    std::size_t hash_item = std::hash<std::string>{}(item);
    int offset;
    std::size_t estimate = SIZE_MAX;
    offset = 0;

    if ( current >= levels )
        return return_actual_count();

    for ( j = 0; j < depth; j++ )
    {
        estimate = MIN( estimate, range_table[current][hash(hash_item, current, j ) + offset] );
        offset += width;
    }

    return estimate;
}
```

Εικόνα 3.7. Η υλοποιήση της estimateCR

```
std::size_t RangeCMSketch :: estimateRange ( int start, int end )
{
    unsigned int i, j, left, right, top;
    std::size_t  estimate;

    top = logbits;
    estimate = 0;

    end = MIN( top, end ); // must not exceed logbits

    for ( i = 0; i < levels; i++ )
    {

        if ( start == end )
            break;

        if ( ( end - start + 1 ) < ( 1 << 1 ) )
        {
            for ( j = start; j < end; j++ )
                estimate += estimateCR( i, j );
            break;
        }
        else
        {
            left = ( ( (start>>1) +1 ) << 1 ) - start;
            right = end - ( ( end >> 1 ) << 1 );
            if ( left > 0 && ( start < end ) )
            {
                for ( j = 0; j < left; j++ )
                {
                    estimate += estimateCR( i, start + j );
                }
            }
```

Εικόνα 3.8. Υλοποίηση της Range Query

```
            if ( right > 0 && ( start < end ) )
            {
                for ( j = 0; j < right; j++ )
                {
                    estimate += estimateCR( i, end - j -1 );
                }
            }
            start = start >> 1;
            if ( left > 0 )
                start++;
            end = end >> 1;
        }

    }

    return estimate;
}
```

Εικόνα 3.9. Συνέχεια υλοποίησης της Range Query

3.4.2 φ-Quantile

Η συνάρτηση findQuantile(float fraction) έχει ως είσοδο το φ. Προκειμένου να βρούμε το μικρότερο φ-Quantile πρέπει πρώτα να βρούμε το μεγαλύτερο αντικείμενο, έστω I, τέτοιο ώστε το εκτιμώμενο Range Query(0, i) = $\varphi||\alpha||_1$. Αντίστοιχα βρίσκουμε το μεγαλύτερο αντικείμενο τέτοιο ώστε Range Query(i, n) = $(1-\varphi)||\alpha||_1$, με n το τελευταίο αντικείμενο του συνόλου δεδομένων. Οι δύο παραπάνω εκτιμήσεις γίνονται με τη χρήση δύο συναρτήσεων findLeft και findRight οι οποίες αφότου ολοκληρωθούν οι δύο εκτιμήσεις βρίσκουμε το φ-Quantile με την εξίσωση : $quantile = \frac{left+right}{2}$. [5]

3.4.3 Heavy Hitters

Τέλος, το αρχείο *rangecms.cpp* υλοποιεί τη διαδικασία εύρεσης heavy hitters. Η αναδρομική συνάρτηση υπεύθυνη για την εύρεση των heavy hitters είναι η *findHeavyHitters(int current_level, int start, int mincount),* όπου current_level είναι το τωρινό επίπεδο της αναδρομής, start το αντικείμενο ελέγχουμε αν είναι heavy hitter και mincount, που μένει ανέπαφο σε κάθε αναδρομή, είναι το μικρότερο ποσό που θέλουμε να ξεπεράσει ένα αντικείμενο για να θεωρηθεί heavy hitter. Η συνάρτηση ξεκινά από το μεγαλύτερο επίπεδο και σε κάθε αναδρομή κατεβαίνει ένα επίπεδο. Κρατάμε έναν πίνακα, τον οποίο ονομάσαμε heavyhitters, ο οποίος περιέχει όλους τους heavy hitters που έχουμε βρει. Αρχικοποιούμε κάθε στοιχείο του σε 0 μόλις αρχίσει το πρόγραμμα. Αυτό γίνεται με τη βοήθεια ενός flag το οποίο αρχικά είναι 0, αλλά μόλις καλέσουμε για πρώτη φορά τη συνάρτηση γίνεται 1. Αυτό συμβαίνει γιατί ο αλγόριθμος της findHeavyHitters είναι αναδρομικός και πρέπει να αποφευχθεί η επαναφορά του πίνακα στην αρχική κατάσταση σε κάθε αναδρομή. Εκτελούμε point query για να βρούμε την εκτιμώμενη τιμή του αντικειμένου start και αν η τιμή αυτή ξεπερνά το mincount και είμαστε στο χαμηλότερο επίπεδο και αν δεν έχει γεμίσει ο πίνακας heavy hitters, τότε ενημερώνουμε τον πίνακα heavy hitters με το αντικείμενο start. Αλλιώς εκτελούμε δυαδική αναζήτηση καλώντας ξανά το findHeavyHitters. Όταν ολοκληρωθεί ο αναδρομικός αλγόριθμος επαναφέρουμε το flag στην αρχική του κατάσταση και επιστρέφουμε τον πίνακα ως αποτέλεσμα.

Κεφάλαιο 4. Αποτελέσματα

Στο κεφάλαιο αυτό θα αναλύσουμε τα αποτελέσματα της Count-Min Sketch που υλοποιήσαμε σε δεδομένα από πραγματικές εφαρμογές, αλλά και σε λογοτεχνικά βιβλία, που είναι διαθέσιμα στο public domain για ελεύθερη χρήση. Σε κάθε πείραμα που επιχειρήσαμε παρείχαμε 8 διαφορετικούς συνδυασμούς τιμών για τις μεταβλητές *ε* και *δ*, ενώ κάθε σετ εκτελέστηκε σε επανάληψη 6 φορές προκειμένου να παρατηρήσουμε την απόδοση της Count-Min Sketch με διαφορετικές τιμές a και b των συναρτήσεων κατακερματισμού. Στο παρακάτω πίνακα αναγράφουμε τους συνδυασμούς που χρησιμοποιήσαμε, το πίνακα που προκύπτει από κάθε συνδυασμό καθώς και τον αποθηκευτικό χώρο που χρειάστηκε για την υλοποίηση της Point και Inner Product Query, αλλά και τον χώρο που χρειάστηκε για τα δυαδικά διαστήματα στην υλοποίηση της Range Query.

(ε, δ)	Μέγεθος Πίνακα (Width Depth)	Αποθηκευτικός Χώρος Point Query	Αποθηκευτικός Χώρος Range Query
(0.1, 0.9)	20 4	<1Kb	13Kb
(0.2, 0.9)	10 4	<1Kb	6Kb
(0.1, 0.8)	20 3	<1Kb	10Kb
(0.01, 0.9)	200 4	6Kb	134Kb
(0.01, 0.99)	200 7	11Kb	235Kb
(0.001, 0.99)	2000 7	112Kb	2Mb
(0.001, 0.999)	2000 10	160Kb	3Mb
(0.0001, 0.999)	20000 10	1Mb	32Mb

Πίνακας 3.4-1. Συνδυασμοί (ε,δ), μέγεθος σκίτσου και αποθηκευτικός χώρος

Κάθε επανάληψη ήταν διαφορετική από τη προηγούμενη καθώς το σκίτσο μας είναι πιθανοτικό, επομένως οι μεταβλητές των συναρτήσεων κατακερματισμού που χρησιμοποιούνται δεν είναι ίδιες. Τα boxplots που θα παρουσιάσουμε παρακάτω απεικονίζουν τους μέσους όρους των επαναλήψεων του κάθε σετ για τα ποσοστά εσφαλμένων εκτιμήσεων της Point Query, τις τιμές εύρους διαστημάτων της Range Query καθώς και τις τιμές των φ-Quantiles, με το φ=0.6. Για την εύρεση heavy hitters θεωρώντας heavy hitter ένα αντικείμενο που απαρτίζει τουλάχιστον το 10% του συνολικού αθροίσματος των αντικειμένων. Το πλήθος των heavy hitters το περιορίσαμε στο μέγεθος του width για περισσότερη απλοποίηση και καλύτερη ανάλυση των αποτελεσμάτων. Για τις ανάγκες της Point Query έχουμε κρατήσει τις ακριβείς τιμές του κάθε αντικειμένου προκειμένου να μπορέσουμε να προσδιορίσουμε αν ένα αντικείμενο έχει υπερεκτιμηθεί σε βαθμό που ξεπερνά το ε. Ο προσδιορισμός έγινε με βάση την εξίσωση: $\widehat{a_i} \leq a_i + \varepsilon||\alpha||_1$ όπως αναφέραμε στην Ενότητα 2.3. Επιπλέον, έχουμε κρατήσει το ακριβές άθροισμα των αντικειμένων προκειμένου να το συγκρίνουμε τα αποτελέσματα των Inner Product και Range Query.

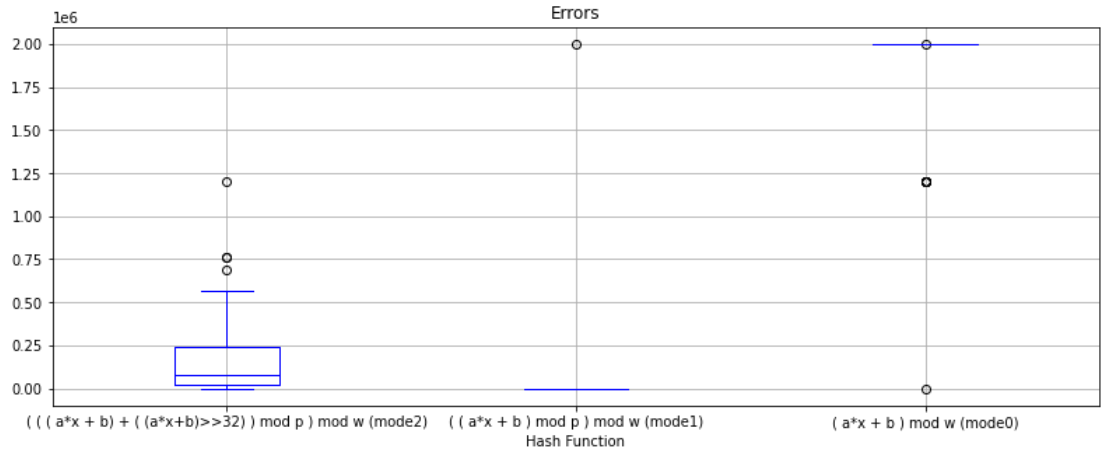
Η Range Query, σε όλα μας τα πειράματα, επιστρέφει την εκτιμώμενη τιμή του εύρους διαστημάτων (1,N), όπου 1 είναι το αρχικό αντικείμενο ενός αρχείου και N είναι το τελευταίο αντικείμενο του αρχείου. Το συγκεκριμένο εύρος διαστήματος μας επιτρέπει καλύτερα να δούμε την αποτελεσματικότητα των δυαδικών διαστημάτων εφόσον μπορούμε να συγκρίνουμε το αποτέλεσμα της με την κανονική τιμή του διαστήματος (1,N), και επιπλέον παρέχει τη δυνατότητα να αναλύσουμε τη σχέση της Range Query με την εύρεση του φ-Quantile, καθώς όπως θα παρουσιάσουμε παρακάτω, όσο μικρότερη απόκλιση έχει η Range Query από την κανονική τιμή του διαστήματος (1,N), τόσο μικραίνει το φ-Quantile έχοντας μικρότερη απόκλιση από το πραγματικό φ-Quantile του αρχείου. Σε αρκετές περιπτώσεις στα πειράματα μας η Range Query φτάνει τη κανονική τιμή του διαστήματος, δίνοντας μας έτσι ένα φ-Quantile, που είναι ίσο, ή έχει αμελητέα απόκλιση, από το πραγματικό φ-Quantile.

Ανάλογα με το τύπου αρχείου παρατηρούμε και διαφορετική αλληλεπίδραση με το σκίτσο μας, όπως εξηγήσαμε στην Ενότητα 3.2.1. Για το λόγο αυτό θα χωρίσουμε το κεφάλαιο μας ανά τύπο αρχείου. Τέλος, για απλοποίηση στην ανάλυση, θα ονομάσουμε τις συναρτήσεις κατακερματισμού mode2, mode1, mode0 αντιστοιχώντας τη κάθε συνάρτηση κατακερματισμού με το mode που της αναλογεί. Στους πίνακες των παρακάτω ενοτήτων θα ταξινομήσουμε τις συναρτήσεις κατακερματισμού με βάση την εμφάνιση τους στα boxplots.

4.1 BMS

Ξεκινάμε την ανάλυση μας με τη μορφή BMS, καθώς αυτή έχει την πιο απλή αλληλεπίδραση με τη δομή μας. Τα id έχει μόνον μια αξία και ενημερώνουμε μία φορά το σκίτσο για κάθε id. Για τους σκοπούς του πειράματος μας χρησιμοποιήσαμε το αρχείο *test2m*, μεγέθους 19.521KB, το οποίο περιέχει 2 εκατομμύρια id σε αύξουσα σειρά, όπου id ∈ {1, 2, …., 2000000} και έχουν τυχαίες ακέραιες αξίες που ανήκουν στο διάστημα {1, 2, …., 20}. Το συνολικό άθροισμα των αντικειμένων για το test2m είναι 20993758.

4.1.1 Εύρεση Εσφαλμένων Εκτιμήσεων



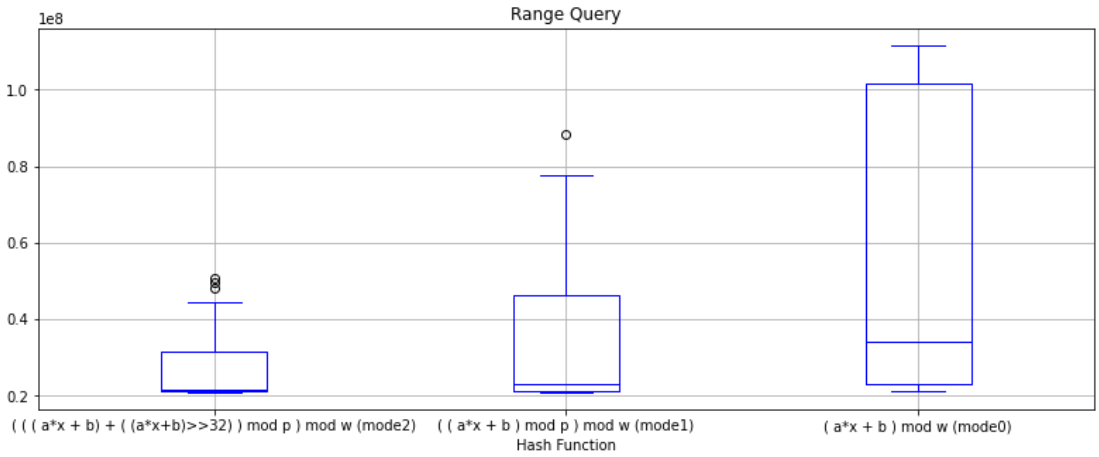
Εικόνα 4.1-1. Point Query για test2m

Το παραπάνω boxplot απεικονίζει τις εσφαλμένες εκτιμήσεις της Point Query για το αρχείο test2m. Όπως αναφέραμε προηγουμένως, ο έλεγχος γίνεται με τη χρήση της εξίσωσης $\widehat{a_i} \leq \alpha_i + \varepsilon||\alpha||_1$ η οποία είναι δυνατή με τη βοήθεια ενός συμπληρωματικού πίνακα μεγέθους 2000000 στον οποίο κρατούσαμε τις ακριβείς τιμές του κάθε id. Παρατηρούμε πως η συνάρτηση κατακερματισμού mode0 παρουσιάζει λάθη σε όλους τους συνδυασμούς και σχεδόν όλες τις επαναλήψεις. Το μηδενικό λάθος που παρατηρούμε είναι ακραία περίπτωση από μία επανάληψη του συνδυασμού (0.1 0.9). Οι υπόλοιποι συνδυασμοί πλην του (0.2 0.9) παρουσιάζουν σφάλμα 100% ή 99.999%, με τον συνδυασμό (0.2 0.9) να παρουσιάζει σφάλμα 60% σε κάθε επανάληψη.

Η συνάρτηση κατακερματισμού mode1, πέρα από μία ακραία περίπτωση στην οποία έχει σφάλμα 100%, παρουσιάζει σταθερά μηδενικό σφάλμα εκτίμησης που είναι αναμενόμενο αφού είναι ανεξάρτητη ανά ζεύγη, ενώ η συνάρτηση mode0 έχει διακύμανση σφάλματος για όλους τους συνδυασμούς. Παρατηρούμε πως τα σφάλματα

μειώνονται αγγίζοντας σχεδόν το μηδέν. Τα σφάλματα αυτά συμβαίνει για το συνδυασμό (0.01 0.99) και τους ακόλουθους του που έχουν ε < 0.01. Συνεπώς παρατηρούμε πως όσο μειώνεται το ε, τόσο αυξάνεται η επίδοση της mode2.

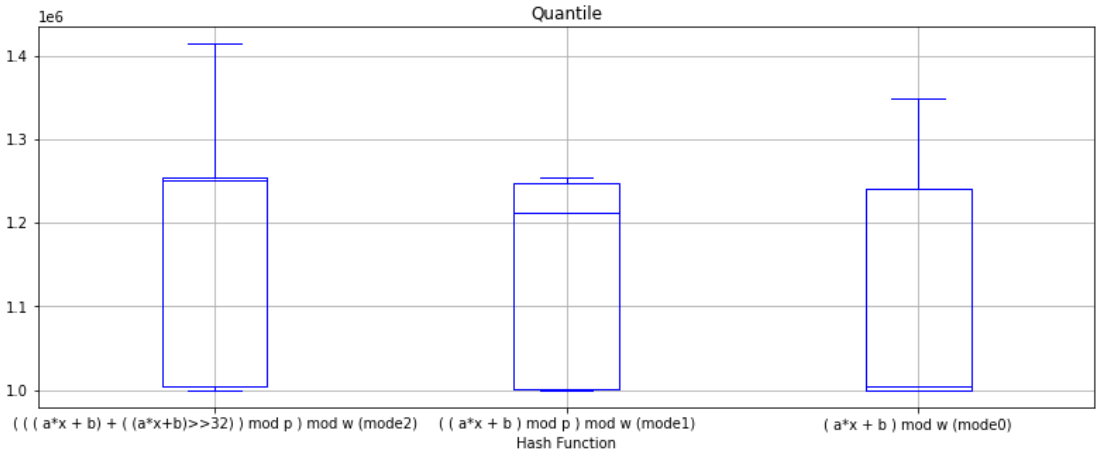
4.1.2 Εύρεση Εύρους Διαστημάτων



Εικόνα 4.1-2. Range Query για test2m

Στο εύρος διαστημάτων παρατηρούμε αρκετή διαφορά σε σχέση με τη Point Query. Η συνάρτηση mode2 παρουσιάζει τη μικρότερη απόκλιση των αποτελεσμάτων ενώ η mode1, παρά την ακρίβεια της στην Point Query παρουσιάζει μεγαλύτερη απόκλιση, η οποία μειώνεται με την μείωση της ε. Η συνάρτηση mode2 έχει ίδια συμπεριφορά με την mode1, αλλά έχει μεγαλύτερη απόκλιση καθώς τα αποτελέσματα της για μεγάλες τιμές ε είναι έως και 3 φορές μεγαλύτερες από τις αντίστοιχες της mode1. Παρόμοια με την Point Query, όσο μειώνεται η τιμή της ε, τόσο μικραίνει η απόκλιση της Range Query από τη κανονική τιμή του διαστήματος (1,2000000). Παρατηρούμε σε όλες τις συναρτήσεις κατακερματισμού μείωση, η οποία οφείλεται στη μείωση της ε και δ, ή πιο απλά, στο μεγαλύτερο μέγεθος του πίνακα της Count-Min Sketch. Οι 3 συναρτήσεις συγκλίνουν στο 0.2 που είναι αναμενόμενο εφόσον η πραγματική τιμή του διαστήματος είναι 20993758, με την Range Query της συνάρτησης mode2 να είναι ίση με 20993758 για το συνδυασμό (0.0001 0.999).

4.1.3 Εύρεση 0.6-Quantiles



Εικόνα 4.1-3. φ-Quantile με φ=0.6 για test2m

Στην εύρεση 0.6-Quantiles παρατηρούμε πως οι τιμές φ-Quantiles μειώνονται. Αυτό συμβαίνει λόγω της μείωσης της τιμής της Range Query, όπως αναλύσαμε στη προηγούμενη Ενότητα και στην Υποενότητα 4.1.2. Όπως συμβαίνει και με τη Range Query, αναμένουμε οι τιμές φ-Quantiles για όλες τις συναρτήσεις να συγκλίνουν σε βαθμό που είναι σχεδόν ταυτόσημες. Σε αντίθεση με την Range Query, το φ-Quantile αυξάνεται καθώς η υπερεκτίμηση της Range Query μικραίνει. Όπως ήδη αναφέραμε, ο συνδυασμός (0.0001 0.999) για τη συνάρτηση mode2 αγγίζει τη πραγματική τιμή και το φ-Quantile παραμένει σταθερό στις επαναλήψεις στη τιμή 1254819, που είναι το πραγματικό φ-Quantile του αρχείου test2m.

4.1.4 Εύρεση Heavy Hitters

Σχεδόν σε κάθε επανάληψη για του συνδυασμού (0.1 0.9) το σκίτσο μας βρήκε heavy hitters με συνάρτηση κατακερματισμού mode0, ενώ για κάθε επανάληψη στους συνδυασμούς (0.2 0.9) υπήρχαν heavy hitters με τις συναρτήσεις mode2, mode0 ενώ για το συνδυασμό (0.1 0.8) heavy hitters επέστρεψε μόνον η mode0 σε κάθε επανάληψη. Με τη συνάρτηση mode1 δεν υπήρχαν heavy hitters καθώς η συνάρτηση mode1 δεν έχει σφάλματα οπότε δεν υπερεκτιμά τα αντικείμενα παραπάνω από το βαθμό ε.

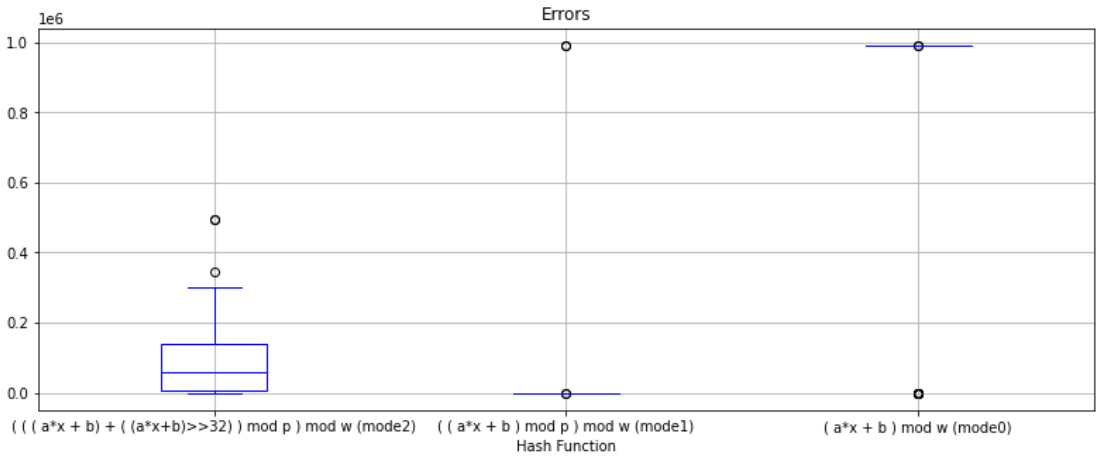
Συνάρτηση Κατακερματισμού	(0.1 0.9)	(0.2 0.9)	(0.1 0.8)
mode2	-	393216 393217 393218 393219 393220 393221 393222 393223 393224 393225	-
mode0	8192 8193 8194 8195 8196 8197 8198 8199 8200 8201 8202 8203 8204 8205 8206 8207 8208 8209 8210 8211	6144 6145 6146 6147 6148 6149 6150 6151 6152 6153	40960 40961 40962 40963 40964 40965 40966 40967 40968 40969 40970 40971 40972 40973 40974 40975 40976 40977 40978 40979

Πίνακας 4.1-1. Heavy Hitters για test2m

4.2 SPMF

Όπως αναφέραμε στο κεφάλαιο 3.2.1, ένα dataset μορφής SPMF μπορεί να έχει πολλές τιμές για ένα id. Επομένως, γίνονται πολλαπλές ενημερώσεις της Count-Min Sketch για ένα μόνο id, όπου κάθε id ∈ {1, 990002}. Το αρχείο που χρησιμοποιήσαμε στο πείραμα μας είναι το *kosarak*, μεγέθους 57.673KB και το συνολικό άθροισμα των αντικειμένων είναι 19142758381.

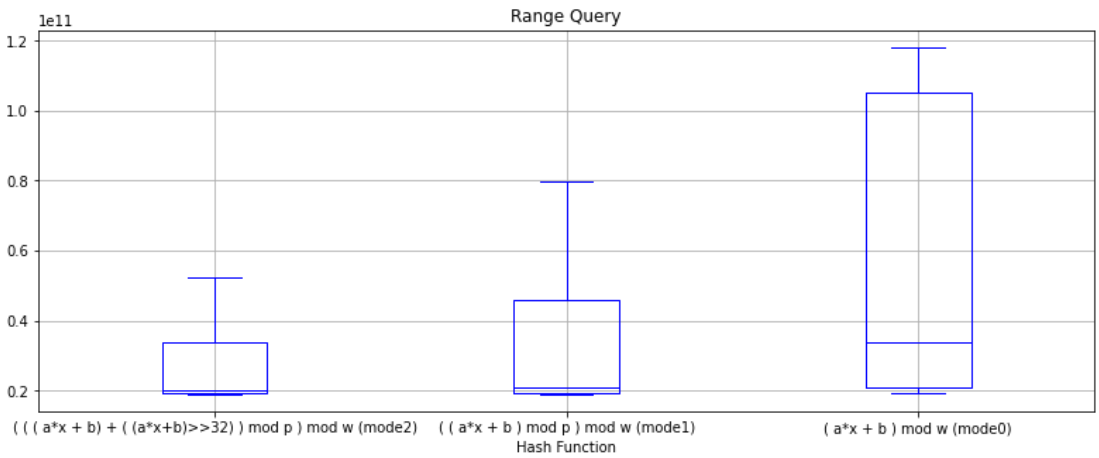
4.2.1 Εύρεση Εσφαλμένων Εκτιμήσεων



Εικόνα 4.2-1. Point Query για kosarak

Όπως στο αρχείο test2m βλέπουμε πως η συνάρτηση mode0 έχει 100% σφάλμα, με μόνη εξαίρεση το συνδυασμό (0.2 0.9) στον οποίο παρουσιάζει μηδενικό σφάλμα. Αντίστοιχα η συνάρτηση mode1, έχει μηδενικό σφάλμα με μόνο μία ακραία περίπτωση που έχει 100% σφάλμα. Τέλος, η mode2 παρουσιάζει μικρότερη απόκλιση, και τυπικά έχει μικρό σφάλμα το οποίο μειώνεται έως και το 0% όσο αυξάνεται το width αλλά και στο συνδυασμό (0.2 0.9) που δεν έχει εσφαλμένες εκτιμήσεις.

4.2.2 Εύρεση Εύρους Διαστημάτων

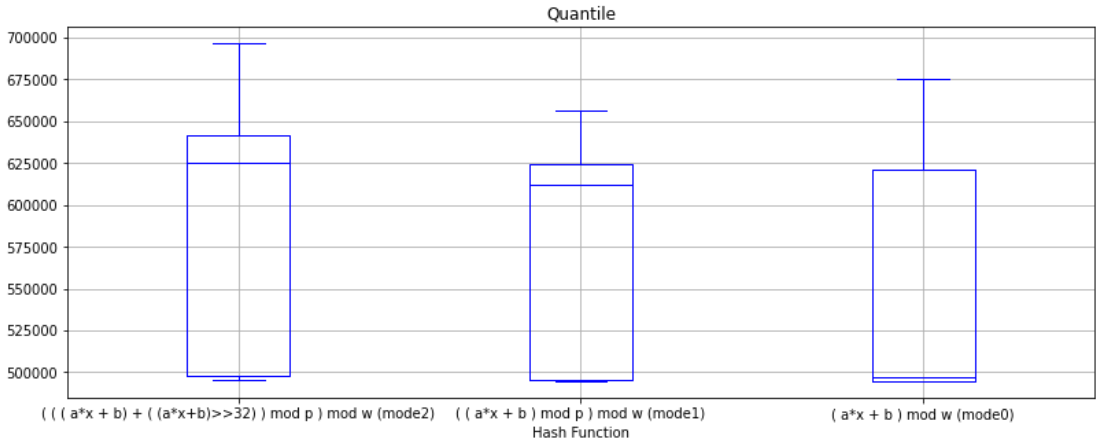


Εικόνα 4.2-2. Range Query για kosarak

Στην εύρεση εύρους διαστημάτων παρατηρούμε ίδια συμπεριφορά με τη Range Query στο αρχείο *test2m*. Συγκεκριμένα, το mode2 έχει τη καλύτερη εκτίμηση σε όλους τους

συνδυασμούς, ακολουθούμενη από την mode1 και τέλος την mode0. Όσο μειώνεται το ε τόσο μικραίνει και η απόκλιση της Range Query από το πραγματικό συνολικό άθροισμα του αρχείου. Οι συναρτήσεις κατακερματισμού συγκλίνουν στη πραγματική τιμή του διαστήματος, σχεδόν αγγίζοντας τη πραγματική τιμή του διαστήματος.

4.2.3 Εύρεση 0.6-Quantiles



Εικόνα 4.2-3. φ-Quantile με φ=0.6 για kosarak

Στο παραπάνω boxplot παρατηρούμε πως οι τιμές των 0.6-Quantiles είναι περίπου ίδιες μεταξύ τους και όσο μειώνεται το ε τόσο πιο πολύ συγκλίνουν τα ποσοτικά που επιστρέφουν οι 3 συναρτήσεις κατακερματισμού. Επειδή οι συναρτήσεις δε δίνουν τη πραγματική τιμή του διαστήματος, το φ-Quantile που επιστρέφουν δεν μπορεί να είναι το πραγματικό φ-Quantile. Παρόλα αυτά, οι συναρτήσεις mode2, mode1 για το συνδυασμό (0.0001 0.999) δίνουν φ-Quantile ίσο με 627027.

4.2.4 Εύρεση Heavy Hitters

Στην εύρεση Heavy Hitters, οι συνδυασμοί (0.1 0.9), (0.2 0.9) και (0.1 0.8) επέστρεψαν heavy hitters με συνάρτηση κατακερματισμού mode0, ενώ η συνάρτηση mode2 απέδωσε heavy hitters μόνον για το συνδυασμό (0.1 0.8). Αντίστοιχα με το *test2m*, με συνάρτηση κατακερματισμού mode1 δεν υπήρχαν heavy hitters.

Συνάρτηση Κατακερματισμού	(0.1 0.9)	(0.2 0.9)	(0.1 0.8)
mode2	-	196608 196609 196610 196611 196612 196613 196614 196615 196616 196617	-
mode0	32768 32769 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779 32780 32781 32782 32783 32784 32785 32786 32787	98304 98305 98306 98307 98308 98309 98310 98311 98312 98313	32768 32769 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779 32780 32781 32782 32783 32784 32785 32786 32787

Πίνακας 4.2-1. Heavy Hitters για kosarak

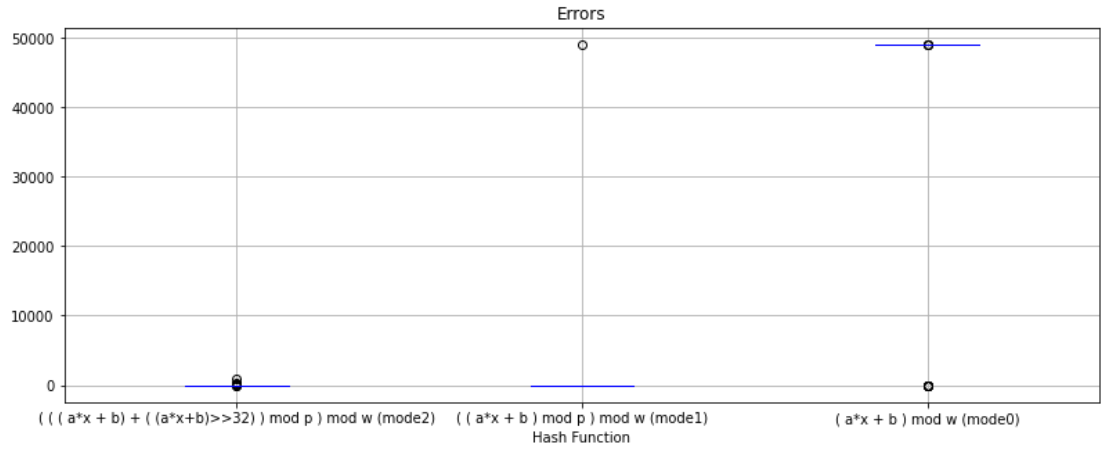
4.3 Kosarak

Η πλειονότητα των αρχείων των πειραμάτων μας είναι της μορφής Kosarak, καθώς τα αρχεία αυτής της μορφής είναι τα μεγαλύτερα σε μέγεθος αρχεία.

4.3.1 Πείραμα pumsb

Το πρώτο μας αρχείο είναι το *pumsb*, μεγέθους 16.253KB με 49046 id. Το συγκεκριμένο αρχείο αποτελεί μια ιδιαίτερη περίπτωση καθώς σχεδόν όλα τα id έχουν περίπου την ίδια τιμή χωρίς να διαφέρουν σημαντικά μεταξύ τους και έχουν συνολικό άθροισμα 12885948494.

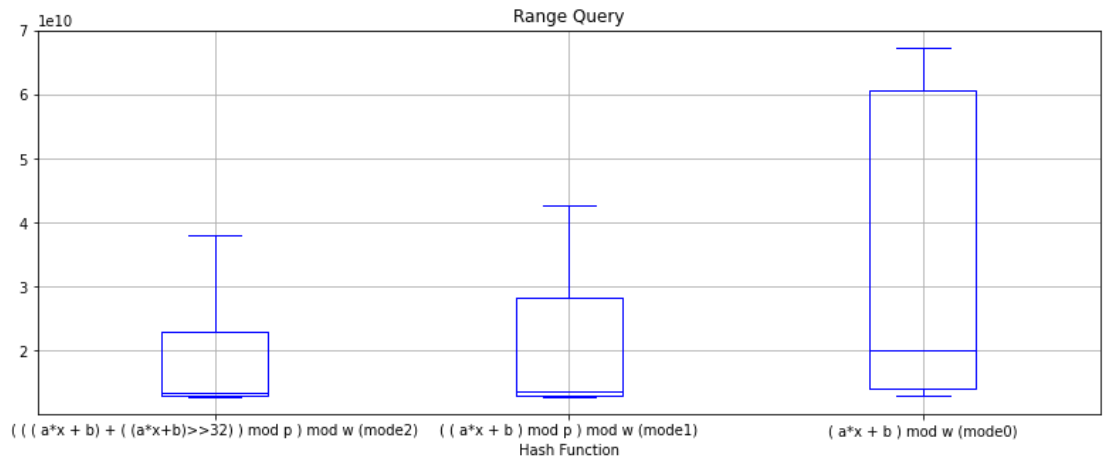
4.3.1.1 Εύρεση Εσφαλμένων Εκτιμήσεων



Εικόνα 4.3-1. Point Query για pumsb

Σε αυτή τη περίπτωση παρατηρούμε πως οι συναρτήσεις mode1, mode0 συμπεριφέρονται όπως περιμένουμε, βάσει των δύο προηγούμενων πειραμάτων, ενώ η συνάρτηση mode2 παρουσιάζει πολύ μικρή απόκλιση σφάλματος, το οποίο πολλές φορές είναι μηδενικό.

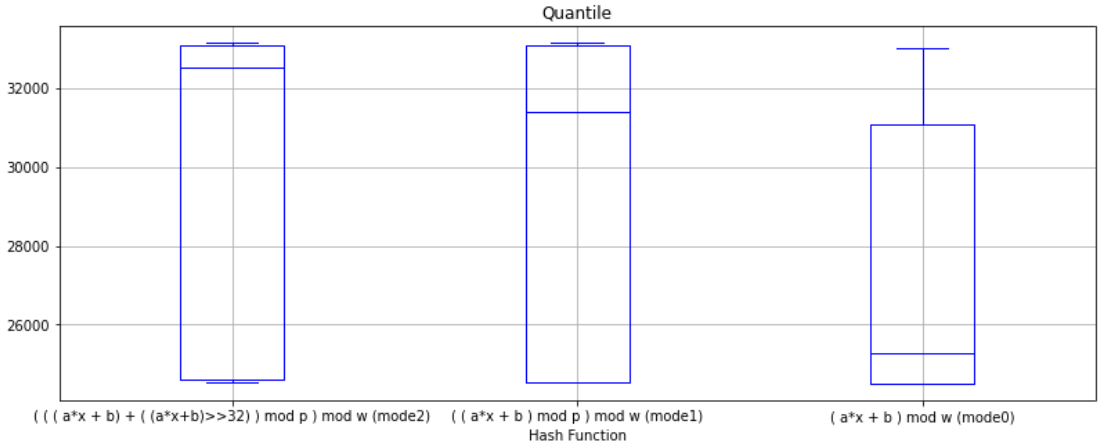
4.3.1.2 Εύρεση Εύρους Διαστημάτων



Εικόνα 4.3-2. Range Query για pumsb

Στην εύρεση διαστημάτων για το pumsb, παρατηρούμε πως οι συναρτήσεις έχουν τα ίδια αποτελέσματα με τα παραπάνω πειράματα. Η απόκλιση είναι μικρότερη για το mode2 και mode1, με τη τιμή της Range Query να μειώνεται, όσο αυξάνεται το width. Η συνάρτηση mode2 καταφέρνει να φτάσει στη πραγματική τιμή του διαστήματος, που είναι 12885948494.

4.3.1.3 Εύρεση 0.6-Quantiles



Εικόνα 4.3-3. φ-Quantile με φ=0.6 για pumsb

Τα 0.6-Quantiles που επιστρέφει το σκίτσο για κάθε συνδυασμό, μειώνονται όσο μειώνεται η τιμή της Range Query, πλησιάζοντας περισσότερο στο πραγματικό 0.6-Quantile του αρχείου. Οι συναρτήσεις mode2, mode1 συγκλίνουν στις τιμές 33141 – 33146 για το συνδυασμό (0.0001 0.999).

4.3.1.4 Εύρεση Heavy Hitters

Παρόμοια με τα παραπάνω πειράματα, υπάρχουν Heavy Hitters για κάθε επανάληψη στο συνδυασμό (0.2 0.9) με συναρτήσεις mode2, mode0, ενώ για τους συνδυασμούς (0.1 0.9) και (0.1 0.8) με τη συνάρτηση mode0.

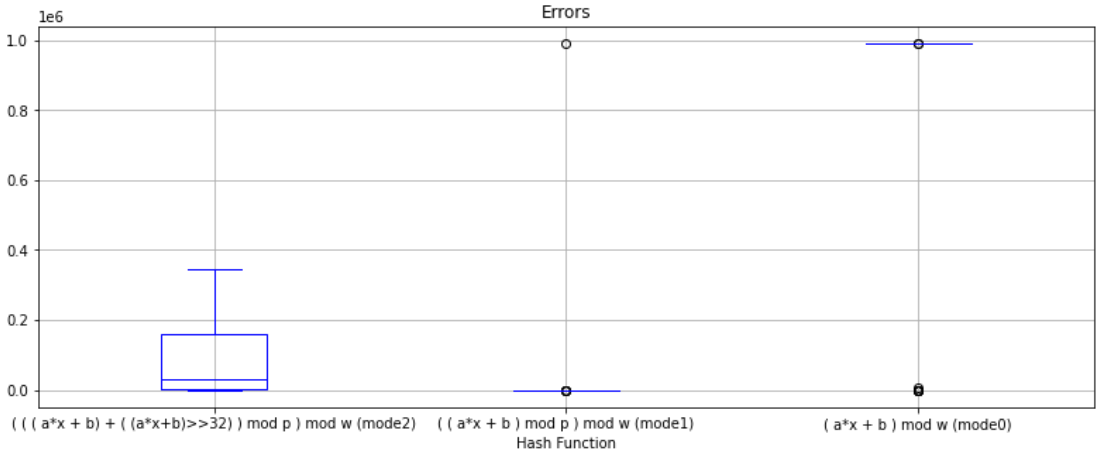
Συνάρτηση Κατακερματισμού	(0.1 0.9)	(0.2 0.9)	(0.1 0.8)
mode2	-	8192 8193 8194 8195 8196 8197 8198 8199 8200 8201	-
mode0	4096 4097 4098 4099 4100 4101 4102 4103 4104 4105 4106 4107 4108 4109 4110 4111 4112 4113 4114 4115	1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043	1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043

Πίνακας 4.3-1. Heavy Hitters για pumsb

4.3.2 Πείραμα kosarak

Για το πείραμα αυτό χρησιμοποιήσαμε το αρχείο kosarak, αυτή τη φορά σε μορφή Kosarak, μεγέθους 31.279KB με 990002 διαφορετικά id και όπως στη 4.2 έχουν συνολικό άθροισμα 19142849591.

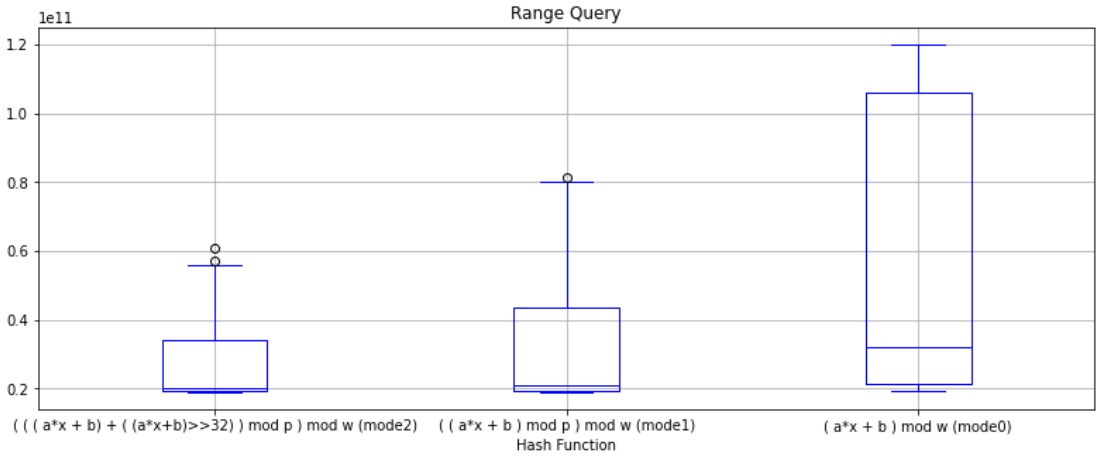
4.3.2.1 Εύρεση Εσφαλμένων Εκτιμήσεων



Εικόνα 4.3-4. Point Query για kosarak

Οι εσφαλμένες εκτιμήσεις της Point Query είναι όμοιες με αυτές των προηγούμενων πειραμάτων, με τη μοναδική διαφορά να είναι η συνάρτηση mode0 για το συνδυασμό (0.2 0.9) που έχει μηδενικό σφάλμα εκτίμησης σε όλες τις επαναλήψεις.

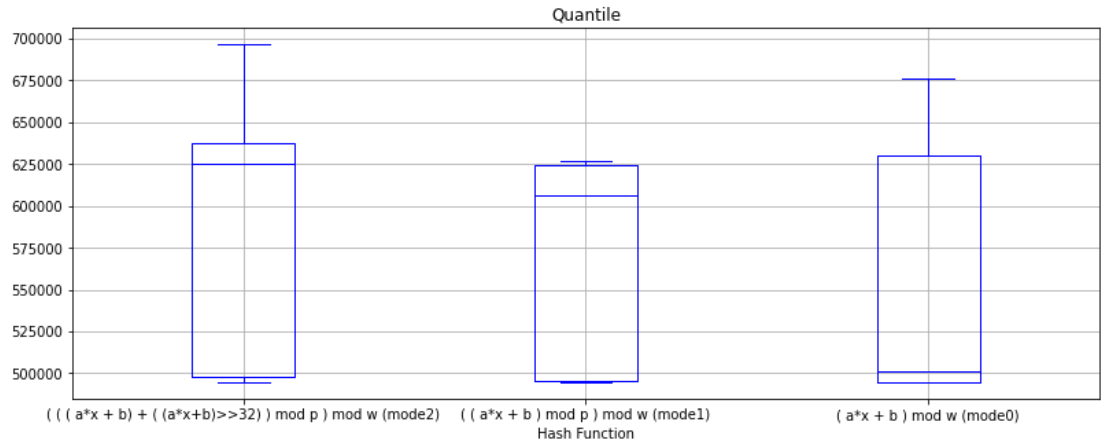
4.3.2.2 Εύρεση Εύρους Διαστημάτων



Εικόνα 4.3-5. Range Query για kosarak

Το παραπάνω boxplot, παρόμοια με τα προηγούμενα, δείχνει τη συνάρτηση mode2 να έχει τη μικρότερη απόκλιση από το πραγματικό άθροισμα των αντικειμένων, ενώ τη συνάρτηση mode0 να έχει τη μεγαλύτερη, με την απόκλιση για όλες τις συναρτήσεις να μειώνεται με την αύξηση της width.

4.3.2.3 Εύρεση 0.6-Quantiles



Εικόνα 4.3-6. φ-Quantile με φ=0.6 για kosarak

Στο παραπάνω boxplot, τα ποσοτικά αυξάνονται με τη μείωση της τιμής της Range Query, η οποία επιτυγχάνεται για μικρότερες τιμές της ε. Καμία από τις συναρτήσεις δεν αγγίζει τη πραγματική τιμή της Range Query, αλλά οι συναρτήσεις mode2, mode1 δίνουν ως αποτέλεσμα φ-Quantile τα 627025 και 627026 για το συνδυασμό (0.0001 0.999).

4.3.2.4 Εύρεση Heavy Hitters

Τα δυαδικά διαστήματα για το αρχείο kosarak, επιστρέφουν heavy hitters μόνον στην mode2 και στην mode0, όπου υπάρχουν heavy hitters στο συνδυασμό (0.2 0.9) σε κάθε επανάληψη και με τις δύο συναρτήσεις, ενώ οι συνδυασμοί (0.1 0.9) και (0.1 0.8) επιστρέφουν heavy hitters μόνον για τη συνάρτηση mode0

Συνάρτηση Κατακερματισμού	(0.1 0.9)	(0.2 0.9)	(0.1 0.8)
mode2	-	196608 196609 196610 196611 196612 196613 196614 196615	-

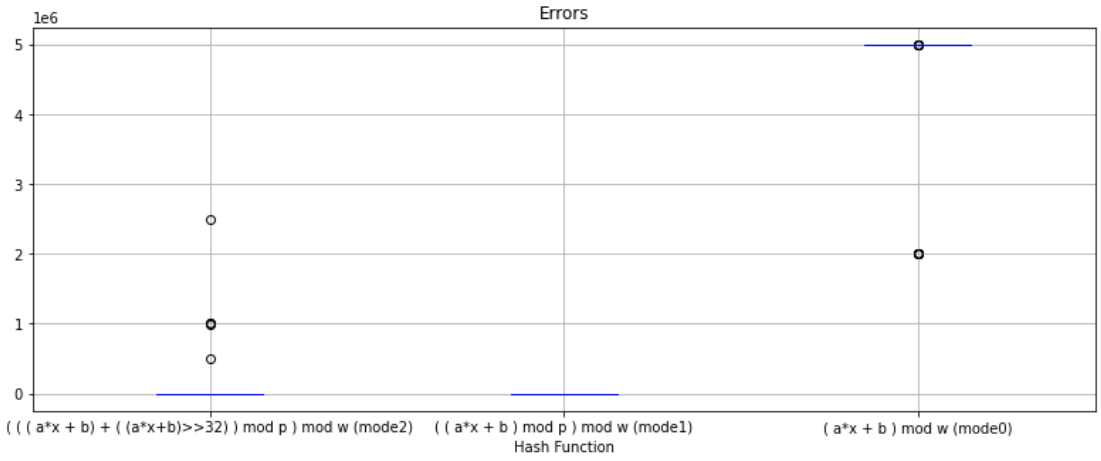
		196616 196617	
mode0	32768 32769 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779 32780 32781 32782 32783 32784 32785 32786 32787	1664 1665 1666 1667 1668 1669 1670 1671 1672 1673	8192 8193 8194 8195 8196 8197 8198 8199 8200 8201 8202 8203 8204 8205 8206 8207 8208 8209 8210 8211

Πίνακας 4.3-2. Heavy Hitters για kosarak

4.3.3 Πείραμα SUSY

Το αρχείο SUSY είναι το μεγαλύτερο σε μέγεθος αρχείο που χρησιμοποιήσαμε στα πειράματα μας, μεγέθους 333.651KB με 5000000 διαφορετικά id και συνολικό άθροισμα 8751887152.

4.3.3.1 Εύρεση Εσφαλμένων Εκτιμήσεων

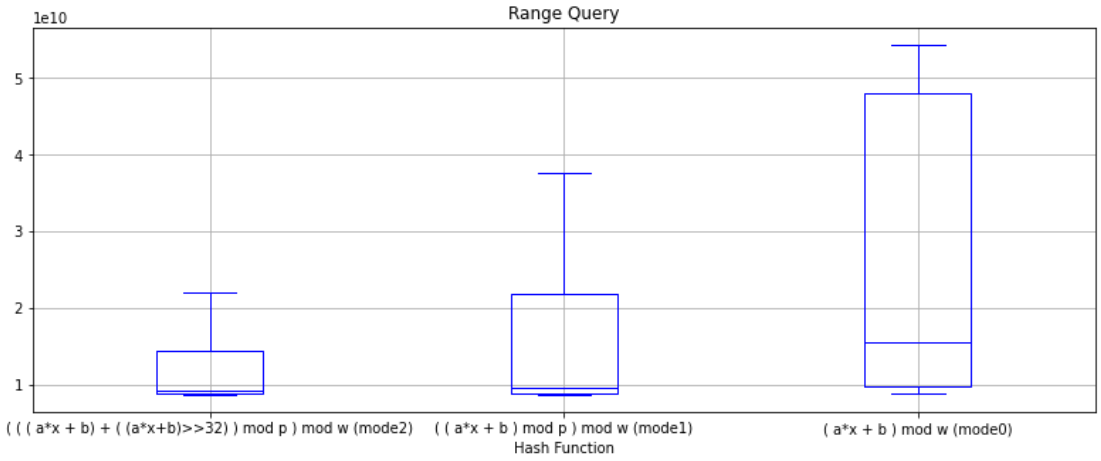


Εικόνα 4.3-7. Point Query για SUSY

Εξαιτίας του μεγέθους του αρχείου και του πλήθους των αντικειμένων, η Point Query στη συνάρτηση mode0 δε φτάνει ποτέ στο μηδενικό σφάλμα. Αντίθετα, ο συνδυασμός που στα προηγούμενα πειράματα έδινε σχεδόν μηδενικό σφάλμα, (0.2 0.9), στη συγκεκριμένη περίπτωση δίνει σφάλμα 40%, όπως και στη συνάρτηση mode2. Η

συνάρτηση mode2 επιτυγχάνει μηδενικό σφάλμα με την αύξηση του μεγέθους του σκίτσου, δηλαδή με τη μείωση της τιμές ε. Η συνάρτηση mode1 διατηρεί το μηδενικό της σφάλμα σε όλους τους συνδυασμούς.

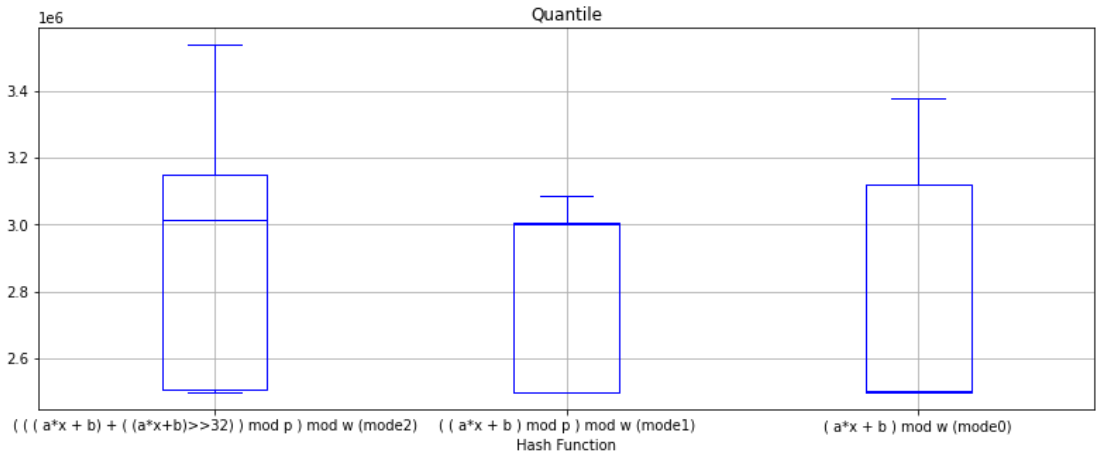
4.3.3.2 Εύρεση Εύρους Διαστημάτων



Εικόνα 4.3-8. Range Query για SUSY

Παρά τη διαφορά στην εκτίμηση της Point Query συγκριτικά με τα προηγούμενα πειράματα, η τιμή της Range Query ακολουθεί την συμπεριφορά των προηγούμενων πειραμάτων όπως έχουμε ήδη εξηγήσει.

4.3.3.3 Εύρεση 0.6-Quantiles



Εικόνα 4.3-9. φ-Quantile με φ=0.6 για SUSY

Παρόμοια με τα προηγούμενα αποτελέσματα, η εύρεση των 0.6-Quantiles για όλες τις συναρτήσεις έχει μεγάλη απόκλιση. Το αρχικό ποσοτικό για συνδυασμούς με μεγάλο ε

είναι μικρό και αυξάνεται όσο μειώνεται το ε, καθώς μειώνεται η απόκλιση του εύρους δεδομένων με δυαδικά διαστήματα από το πραγματικό άθροισμα των id.

4.3.3.4 Εύρεση Heavy Hitters

Η εύρεση των heavy hitters για το αρχείο SUSY είναι σύμφωνη με τις προβλέψεις που έχουμε σύμφωνα με τα προηγούμενα πειράματα. Η συνάρτηση mode1 δεν έχει heavy hitters, η συνάρτηση mode2 επιστρέφει heavy hitters μόνο για το συνδυασμό (0.2 0.9), ενώ η mode2 επιστρέφει heavy hitters για τους συνδυασμούς (0.1 0.9) (0.2 0.9) (0.1 0.8).

Συνάρτηση Κατακερματισμού	(0.1 0.9)	(0.2 0.9)	(0.1 0.8)
mode2	-	786432 786433 786434 786435 786436 786437 786438 786439 786440 786441	-
mode0	327680 327681 327682 327683 327684 327685 327686 327687 327688 327689 327690 327691 327692 327693 327694 327695 327696 327697 327698 327699	311296 311297 311298 311299 311300 311301 311302 311303 311304 311305	104448 104449 104450 104451 104452 104453 104454 104455 104456 104457 104458 104459 104460 104461 104462 104463 104464 104465 104466 104467

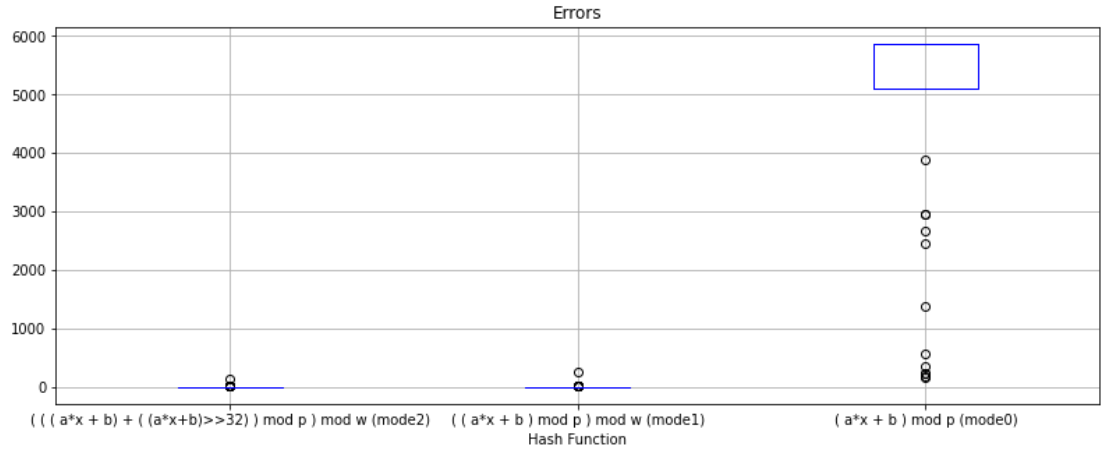
Πίνακας 4.3-3. Heavy Hitters για SUSY

4.4 Λογοτεχνικά Συγγράμματα

Το τέταρτο είδους αρχείου που χρησιμοποιήσαμε είναι λογοτεχνικά βιβλία. Τα αποτελέσματα της Count-Min Sketch διαφέρουν από τα προηγούμενα πειράματα καθώς χρησιμοποιούμε διπλή συνάρτηση κατακερματισμού για την ενημέρωση των λέξεων στο σκίτσο μας. Το βιβλίο που θα αναλύσουμε είναι το βιβλίο του Jacob Abbott

«Η ιστορία του Ιουλίου Καίσαρα» με 5862 διαφορετικές λέξεις και συνολικό αριθμό λέξεων 54215. Η ενημέρωση των δυαδικών διαστημάτων για την εύρεση εύρους διαστημάτων, φ-Quantiles και Heavy Hitters υλοποιήθηκε όπως περιγράψαμε στο κεφάλαιο 3.2.1, αναθέτοντας ένα id σε κάθε ξεχωριστή λέξη.

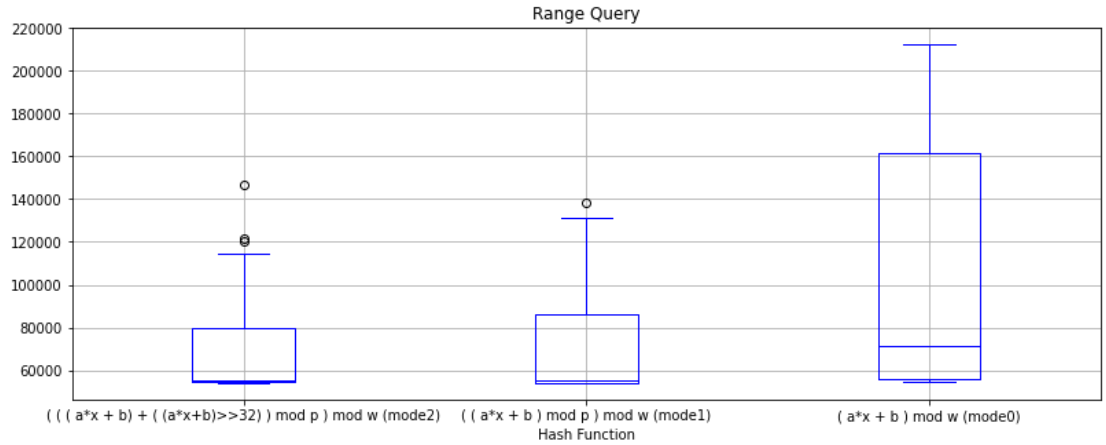
4.4.1 Εύρεση Εσφαλμένων Εκτιμήσεων



Εικόνα 4.4-1. Point Query για "Η ιστορία του Ιουλίου Καίσαρα"

Στο παραπάνω boxplot παρατηρούμε πως το σφάλμα εκτίμησης για τις mode2, mode1 είναι μηδαμινό για όλους τους συνδυασμούς. Η έλλειψη σημαντικού σφάλματος είναι αναμενόμενη δεδομένης της χρήσης δύο συναρτήσεων κατακερματισμού. Με τη διπλή χρήση συναρτήσεων κατακερματισμού σε κάθε λέξη εγγυόμαστε μικρότερο περιθώριο σφαλμάτων. Παρόλα αυτά, η συνάρτηση mode0, παρουσιάζει σφάλμα, που σε πολλούς συνδυασμούς κυμαίνεται στο 95-100% με μερικές ακραίες περιπτώσεις όπου παρουσιάζει σφάλμα κοντά στο 40 ή 60%. Η mode0 για το συνδυασμό (0.2 0.9) παρουσιάζει πολύ μικρό σφάλμα που κυμαίνεται στο 2-3%.

4.4.2 Εύρεση Εύρους Διαστημάτων



Εικόνα 4.4-2. Range Query για "Η ιστορία του Ιουλίου Καίσαρα"

Το παραπάνω boxplot είναι αναμενόμενο βάσει των προηγούμενων πειραμάτων. Η συνάρτηση mode2 έχει τη μικρότερη απόκλιση από το πραγματικό πλήθος λέξεων. Η τιμή της Range Query μειώνεται με την αύξηση του μεγέθους του width του σκίτσου.

4.4.3 Εύρεση 0.6-Quantiles

Στην εύρεση 0.6-Quantiles, σχεδόν κάθε επανάληψη σε κάθε συνδυασμό επέστρεψε διαφορετική λέξη σαν αποτέλεσμα. Για εξοικονόμηση χώρου και για απλοποίηση των αποτελεσμάτων παρέχουμε ένα πίνακα με τις λέξεις που είχαμε σταθερά ως αποτέλεσμα για τους τελευταίους συνδυασμούς μας (0.001 0.999) και (0.0001 0.999):

mode2	Civil
mode1	Quarrel
mode0	Quarrel

Πίνακας 4.4-1. 0.6-Quantiles για "Η ιστορία του Ιουλίου Καίσαρα"

4.4.3.1 Εύρεση Heavy Hitters

Σε αντίθεση με τα προηγούμενα πειράματα, όλες οι συναρτήσεις κατακερματισμού αποδίδει Heavy Hitters. Στους συνδυασμούς (0.1 0.9), (0.2 0.9) και (0.1 0.8) υπάρχουν heavy hitters, με τη συνάρτηση mode2 να επιστρέφει heavy hitters για κάθε επανάληψη σε όλους τους συνδυασμούς.

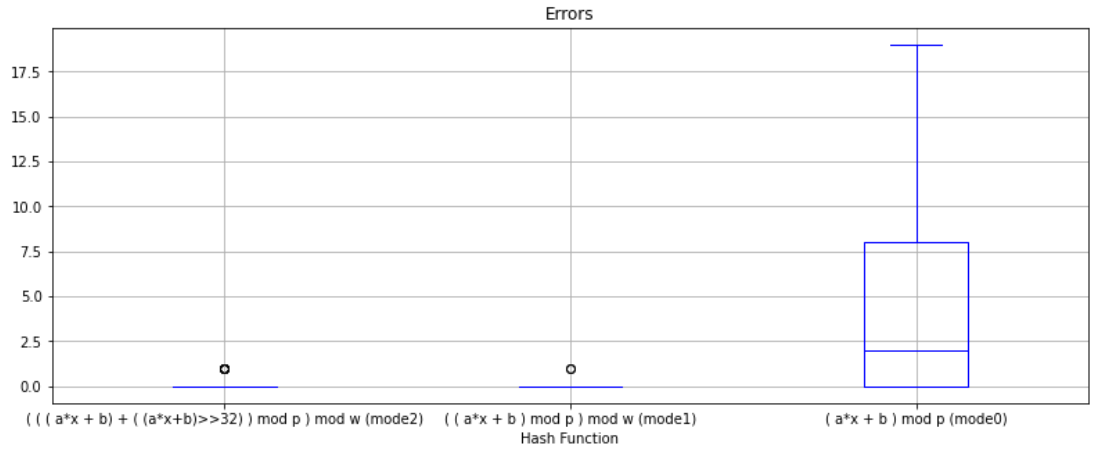
Συνάρτηση Κατακερματισμού	(0.1 0.9)	(0.2 0.9)	(0.1 0.8)
mode2	the	the	a the terrible The more be some boundaries had as
mode1	the	of the and to	the
mode0	There were three in ancient days each which furnished history with a hero the Greeks Carthaginians and Romans Alexander was	CAESAR CHAPTER I MARIUS AND SYLLA Sidenote Three great European nations of antiquity] There were three in	I MARIUS AND SYLLA Sidenote Three great European nations

Πίνακας 4.4-2. Heavy Hitters για "Η ιστορία του Ιουλίου Καίσαρα"

4.5 Κείμενο NLP

Στο τελευταίο κεφάλαιο των αποτελεσμάτων, αναλύουμε τα αποτελέσματα του βιβλίου «Η ιστορία του Ιουλίου Καίσαρα» το οποίο είναι γραμμένο σε μορφή NLP με 32 διαφορετικές λέξεις, που ονομάζονται tags, με συνολικό αριθμό λέξεων 47294. Στα παρακάτω αποτελέσματα θα εξηγήσουμε τι σημαίνουν τα αποτελέσματα μορφής NLP. [6]

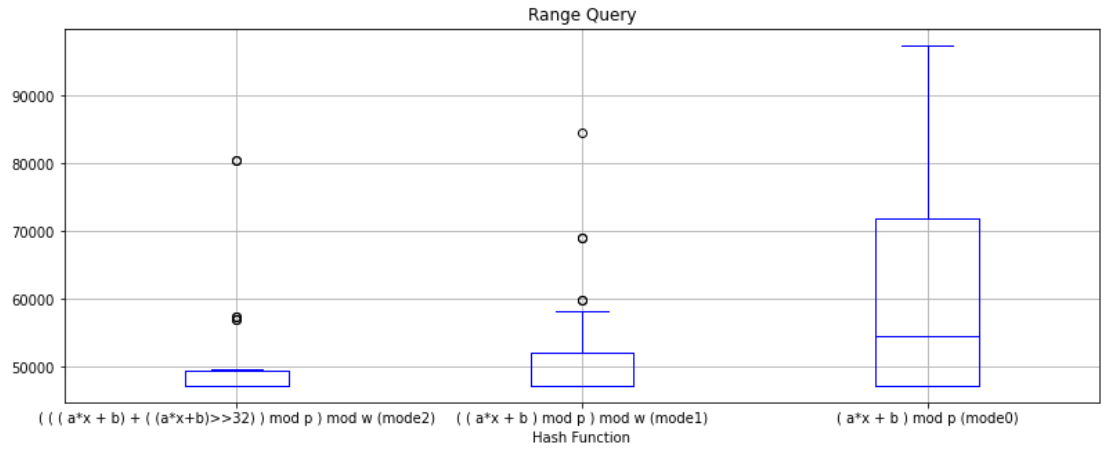
4.5.1 Εύρεση Εσφαλμένων Εκτιμήσεων



Εικόνα 4.5-1. Point Query για "Η ιστορία του Ιουλίου Καίσαρα" σε μορφή NLP

Η Point Query για το βιβλίο σε μορφή NLP παρουσιάζει την ίδια συμπεριφορά με τη κανονική μορφή του βιβλίου, για το λόγο εξηγήσαμε στην Υποενότητα 4.4.1. Οι συναρτήσεις mode2 και mode1 παρουσιάζουν μηδενικό σφάλμα σε όλους τους συνδυασμούς έχοντας ακραίες περιπτώσεις σφάλματος 3.1% σε μερικές επαναλήψεις για τους συνδυασμούς (0.1 0.8) και (0.1 0.9). Η mode0 παρουσιάζει σφάλμα 53-59% για το συνδυασμό (0.1 0.9), σταθερό σφάλμα 21% για το συνδυασμό (0.1 0.8) και 3-9% σφάλμα για το συνδυασμό (0.2 0.9). Για τους υπόλοιπους συνδυασμούς η mode0 παρουσιάζει μηδενικό σφάλμα.

4.5.2 Εύρεση Εύρους Διαστημάτων



Εικόνα 4.5-2. Range Query για "Η ιστορία του Ιουλίου Καίσαρα" σε μορφή NLP

Το παραπάνω boxplot ακολουθεί την αναμενόμενη συμπεριφορά βάσει των προηγούμενων πειραμάτων, όπου η απόκλιση με το πραγματικό άθροισμα λέξεων μειώνεται με την αύξηση μεγέθους της Count-Min Sketch.

4.5.3 Εύρεση 0.6-Quantiles

Στην εύρεση 0.6-Quantiles, σχεδόν κάθε επανάληψη σε κάθε συνδυασμό επέστρεψε διαφορετική λέξη σαν αποτέλεσμα. Για εξοικονόμηση χώρου και για απλοποίηση των αποτελεσμάτων παρέχουμε τη λέξη που είχαμε σταθερά ως αποτέλεσμα από το συνδυασμό (0.01 0.99) και έπειτα, που είναι η «ex» που σημαίνει Existential there.

4.5.4 Εύρεση Heavy Hitters

Στην εύρεση Heavy Hitters και οι τρεις συναρτήσεις επέστρεφαν τα ίδια αποτελέσματα από το συνδυασμό (0.01 0.99) και έπειτα. Τα αποτελέσματα αυτά ήταν «nn», «in» και «dt». Τα αποτελέσματα δηλαδή είναι «Noun (Singular)» ουσιαστικό, «Preposition or subordinating conjunction» πρόθεση ή υποτακτική σύνδεση και «Determiner» προσδιορισμός.

4.6 Εσωτερικό Γινόμενο

Στα παραπάνω πειράματα υπολογίσαμε και το εσωτερικό γινόμενο με την υλοποίηση της Υποενότητας 3.3.3. Για να μπορέσουμε να ελέγξουμε τη λειτουργία του, επιλέξαμε να χρησιμοποιήσουμε έναν μοναδιαίο πίνακα ίδιων διαστάσεων με αυτό του σκίτσου μας. Συνεπώς το εσωτερικό γινόμενο που θα επέστρεφε θα ήταν ίσο με το πραγματικό άθροισμα του συνόλου δεδομένων. Όλα τα πειράματα επέστρεφαν σταθερό εσωτερικό γινόμενο για όλους τους συνδυασμούς, το οποίο ήταν ίσο με το συνολικό άθροισμα του κάθε αρχείου που πειραματιζόμασταν. Αυτό συμβαίνει καθώς όπως αναφέραμε στην Ενότητα 2.4, λαμβάνουμε το μικρότερο άθροισμα των θυρίδων της κάθε συνάρτησης κατακερματισμού. Το άθροισμα αυτό θα είναι πάντα ίσο με το συνολικό άθροισμα των αντικειμένων καθώς η κάθε συνάρτηση κατακερματισμού περιέχει όλα τα αντικείμενα του συνόλου δεδομένων. Επομένως, ακόμα και οι συναρτήσεις mode0, mode2 που παρουσιάζουν σφάλματα που οφείλονται σε υπερεκτίμηση λόγω «κακού» κατακερματισμού, επιστρέφουν πάντα το ακριβές άθροισμα των αντικειμένων.

Κεφάλαιο 5. Συμπεράσματα

Στην εργασία αυτή παρουσιάσαμε την δομή Count-Min Sketch και αναλύσαμε την λειτουργία και τους λόγους για τους οποίους η Count-Min Sketch είναι ευρέως διαδεδομένη και απολύτως απαραίτητη εξαιτίας του εκτεταμένου μεγέθους και πλήθους ροών δεδομένων που υπάρχουν στο διαδίκτυο και αυξάνονται συνεχώς με ταχύ ρυθμό. Η Count-Min Sketch επιτρέπει έναν αποτελεσματικό και φτηνό τρόπο αποθήκευσης και επεξεργασίας των δεδομένων που λαμβάνονται από τις ροές χωρίς να επιβαρυνθούν τόσο οι διακομιστές από τα δεδομένα.

Παρόλη την αποτελεσματικότητα της, η δομή Count-Min Sketch εξαρτάται άμεσα από τις συναρτήσεις κατακερματισμού που θα χρησιμοποιηθούν για τη δημιουργία του σκίτσου, γεγονός που αποδείξαμε με τα αποτελέσματα που προέκυψαν από το πρόγραμμα μας. Οι καλύτερες συναρτήσεις κατακερματισμού είναι οι strongly 2-universal hashing συναρτήσεις δεδομένου ότι είναι ανεξάρτητες κατά ζεύγη, καθώς εγγυώνται ελάχιστη έως καθόλου υπερεκτίμηση λόγω της ανεξαρτησίας τους. Με χρήση όμως κατάλληλου μεγέθους για τη Count-Min Sketch μπορούμε να πετύχουμε μικρά ποσοστά σφάλματος και για άλλες, όχι και τόσο κατάλληλες, συναρτήσεις.

Η εξοικονόμηση χρόνου και επεξεργαστικής ισχύος μπορεί να βελτιωθεί περισσότερο με τη χρήση δυαδικών διαστημάτων. Τα δυαδικά διαστήματα απαιτούν περισσότερη μνήμη από το απλό σκίτσο, αλλά είναι πιο αποτελεσματική σε σχέση με το απλό σκίτσο για πολύ βασικές λειτουργίες, όπως η εύρεση των βαρειών αντικειμένων και των φ-Quantiles, που είναι αναγκαία για τη λειτουργία πολλών υπηρεσιών για καλύτερη εξυπηρέτηση των χρηστών.

Εν κατακλείδι, τα αποτελέσματα μας δείχνουν ότι η δομή Count-Min Sketch, με χρήση κατάλληλων συναρτήσεων κατακερματισμού για τη δημιουργία της, αποτελεί ζωτικής σημασίας κομμάτι των υποδομών του διαδικτύου, των υπηρεσιών που παρέχει καθώς και διαφόρων εφαρμογών όπως η NLP.

Βιβλιογραφία

[1] Graham Cormode, S. Muthukrishnan. *An Improved Data Stream Summary: The Count-Min Sketch and its Applications,* 2005.

[2] Jon Kleinberg, Éva Tardos, *Algorithm Design,* Hashing: A Randomized Implementation of Dictionaries, pp 812-817, Cornell University, 2008.

[3] Graham Cormode, S. Muthukrishnan. *Approximating Data with the Count-Min Data Structure.* The Count-Min Sketch, pp. 4, August 2011. Also available here.

[4] Jannik Sundermeier. *The Count-Min-Sketch and its Applications.* Dyadic Ranges 4.3.2, pp 15-16.

[5] Ankit R. Chadha, Rishikesh Misal, Tanaya Mokashi. *Modified Binary Search Algorithm*. In International Journal of Applied Information System, pp. 37-38, New York, USA, April 2014.

[6] Beatrice Santorini. *Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision) (3rd Revision)*. List of tags with corresponding part of speech, pp 6, University of Pennsylvania, Philadelphia, USA, July 1990. Also available here.