**1η εργασία - Αναζήτηση και συσταδοποίηση διανυσμάτων στη C/C++**

**Χριστίνα Ιακωβίδου sdi1900064**

**Ροδάνθη Αλεξίου sdi1900007**

**Ερώτημα Α.1**

**Εισαγωγή**

Ο κώδικας σχετίζεται με τον Αναδρομικό Συναρτησιακό Κατακερματισμό (LSH) για αναζήτηση κοντινότερων γειτόνων, με την χρήση Ευκλείδιας απόστασης.

**Συναρτήσεις και Κλάσεις**

**Κλάση LSHImplementation**

Μέλη Δεδομένων

* **Euclidian\_Amplified\_Functions**: Ένας πίνακας που αποθηκεύει ανακλημένες συναρτήσεις για τον Ευκλείδιο LSH.
* **Euclidian\_Hash\_Tables**: Πίνακες κατακερματισμού για την αποθήκευση σημείων δεδομένων.
* **v**: Ένας διάνυσμα διανυσμάτων που χρησιμοποιείται για υπολογισμούς συναρτήσεων κατακερματισμού.
* **t**: Ένα διάνυσμα σταθερών που χρησιμοποιείται στους υπολογισμούς συναρτήσεων κατακερματισμού.
* **w**, **k**, **L**: Παράμετροι για το LSH.

Constructors

* **LSHImplementation(int L, int k, int w)**: Κατασκευαστής για την αρχικοποίηση των παραμέτρων LSH.

Μέθοδοι Μελών

* **getEuclidianHashTables()**: Επιστρέφει μια σταθερή αναφορά στους πίνακες κατακερματισμού του Ευκλείδιου.
* **getEuclidianAmplifiedFunctions()**: Επιστρέφει μια αναφορά στις ανακλημένες συναρτήσεις.
* **getAmplifiedFunction(int j, int k)**: Επιστρέφει μια ανακλημένη συνάρτηση για δεδομένους δείκτες.
* **getL()**, **getK()**, **getW()**: Συναρτήσεις προσπέλασης για τις παραμέτρους L, k και w.
* **normal\_distribution\_generator()**: Δημιουργεί τυχαίους αριθμούς από κανονική κατανομή.
* **modulo(long long int a, long long int b)**: Υπολογίζει τη λειτουργία modulo.
* **euclidian\_hash\_query(...)**: Υπολογίζει τη θέση στον πίνακα κατακερματισμού για ένα διάνυσμα ερωτήματος.
* **euclidian\_hash\_centroid(...)**: Υπολογίζει τη θέση στον πίνακα κατακερματισμού για ένα κεντροειδές διάνυσμα.
* **calcute\_euclidian\_distance(...)**: Υπολογίζει την Ευκλείδια απόσταση μεταξύ δύο διανυσμάτων.
* **Euclidian\_put\_hash(...)**: Εισάγει ένα διάνυσμα σε όλους τους πίνακες κατακερματισμού.
* **insertCentroidIntoLSH(...)**: Εισάγει ένα κεντροειδές διάνυσμα στους πίνακες κατακερματισμού.
* **read\_mnist\_images(...)**: Διαβάζει δεδομένα εικόνας MNIST.
* **getImageLine(...)**: Επιστρέφει μια γραμμή εικόνας δεδομένου ενός δείκτη.
* **initializeVectors(...)**: Αρχικοποιεί διανύσματα και συναρτήσεις κατακερματισμού.
* **initializeAmplifiedFunctions(...)**: Αρχικοποιεί ανακλημένες συναρτήσεις κατακερματισμού.
* **initializeT(...)**: Αρχικοποιεί τις σταθερές **t** για τις συναρτήσεις κατακερματισμού.
* **cleanupHashTables(...)**: Καθαρίζει τους πίνακες κατακερματισμού αφαιρώντας κενές θέσεις.
* **initializeLSH(...)**: Αρχικοποιεί τις παραμέτρους LSH, τα διανύσματα και τους πίνακες κατακερματισμού.
* **FindNearestNeighbor(...)**: Βρίσκει τον κοντινότερο γείτονα για ένα σημείο ερώτησης.
* **FindNearestNeighborCentroid(...)**: Βρίσκει τον κοντινότερο γείτονα για ένα κεντροειδές.
* **Euclidian\_Full\_Search\_NN(...)**: Εκτελεί πλήρη αναζήτηση για τους κοντινότερους γείτονες.
* **Euclidian\_Full\_Search\_Range(...)**: Εκτελεί πλήρη αναζήτηση για τα σημεία δεδομένων εντός δοθείσας ακτίνας.

**Χρήση**

Στην mnist.cpp υπάρχει η υλοποίηση για την σωστή ανάγνωση των arguments και μια ενδεικτική εκτέλεση είναι:

**./mnist -d train-images.idx3-ubyte -q t10k-images.idx3-ubyte -ο out.txt**

**Ερώτημα Α.2**

**Εισαγωγή**

Σε αυτό το ερώτημα είχαμε την υλοποίηση του αλγόριθμου της τυχαίας προβολής σε έναν δυαδικό υπερκύβο καθώς την έννοια της αναζήτησης του πλησιέστερου γείτονα σε δυαδικά δεδομένα μέσω αυτού. Επίσης πρέπει να υπολογιστούν και οι Ν πλησιέστεροι γείτονες καθώς και τα διανύσματα εντός μιας συγκεκριμένης ακτίνας (R).

**Συναρτήσεις :**

* **generateRandomBinaryPoint(…):** Δημιουργεί ένα τυχαίο δυαδικό σημείο με δεδομένη διάσταση.
* **imageToBinaryVector(…):** Μετατρέπει μια εικόνα MNIST σε δυαδικό διάνυσμα.
* **readMNISTImages(…):** Διαβάζει εικόνες MNIST από ένα αρχείο και τις επιστρέφει ως δυαδικά διανύσματα.
* **hammingDistance(…):** Υπολογίζει την απόσταση Hamming μεταξύ δύο δυαδικών σημείων.

**Δομή Δυαδικού Υπερκύβου:**

**Η δομή Hypercube αντιπροσωπεύει ένα δυαδικό υπερκύβο με τα ακόλουθα χαρακτηριστικά:**

* **numBits:** Ο αριθμός των bits στη δυαδική αναπαράσταση.
* **numTables:** Ο αριθμός των πινάκων κατακερματισμού.
* **numFunctions:** Ο αριθμός των συναρτήσεων κατακερματισμού ανά πίνακα.
* **buckets:** Ένα διάνυσμα από διανύσματα για την αποθήκευση των δυαδικών σημείων.

**Συναρτήσεις Δυαδικού Υπερκύβου:**

* **addPoint(const BinaryPoint& point**): Προσθέτει ένα δυαδικό σημείο στον δυαδικό υπερκύβο.
* **findNearestNeighbor(const BinaryPoint& query):** Βρίσκει τον πλησιέστερο γείτονα σε ένα δυαδικό σημείο στον δυαδικό υπερκύβο.
* **hashFunction(const BinaryPoint& point, int tableIndex):** Υπολογίζει την τιμή κατακερματισμού για ένα δυαδικό σημείο.
* **printHypercube(const Hypercube& hypercube):** Εκτυπώνει το περιεχόμενο του δυαδικού υπερκύβου για οπτικοποίηση.

**Συναρτήσεις Αναζήτησης του Πλησιέστερου Γείτονα:**

* **findNearestNeighbor(const BinaryPoint& query, const vector<BinaryPoint>& data):** Βρίσκει τον πλησιέστερο γείτονα σε ένα σύνολο δυαδικών σημείων.
* **findKNearestNeighbors(const BinaryPoint& query, const vector<BinaryPoint>& data, int k):** Βρίσκει τους k πλησιέστερους γείτονες σε ένα σύνολο δυαδικών σημείων.
* **rangeSearch(const BinaryPoint& query, const vector<BinaryPoint>& data, int radius**): Βρίσκει όλα τα σημεία εντός ενός καθορισμένου ακτίνας από ένα δυαδικό σημείο σε ένα σύνολο δυαδικών σημείων.

**Υλοποιήση :**

Αρχικά διαβάζονται εικόνες MNIST, γεμίζει ο hypercube με αυτά τα δεδομένα, δημιουργούνται τυχαία queries και βάση αυτών εκτελούνται αναζητήσεις του πλησιέστερου γείτονα. Δυστυχώς αυτός ο κώδικας δεν υλοποιεί ακριβώς αυτό που είχε ζητηθεί στην εκφώνηση , αλλά λόγω έλλειψης χρόνου είναι ότι πιο κοντινό μπορούσε να υλοποιηθεί. Για αυτόν τον λόγο , δεν τον προσθέσαμε καθόλου και σαν επιλογή στο δεύτερο ερώτημα.

**Ερώτημα Β**

**Εισαγωγή**

Υλοποίηση αλγορίθμων για τη συσταδοποίηση διανυσμάτων στον χώρο R.

**Συναρτήσεις**

* **std::vector<unsigned char\*> kmeans\_plusplus\_init(const fileData& data, int K)** - Συνάρτηση για την αρχικοποίηση του K-Means++.
* **double euclideanDistance(const unsigned char\* point1, const unsigned char\* point2, int size)** - Υπολογισμός της Ευκλείδιας απόστασης μεταξύ δύο σημείων δεδομένων.
* **std::vector<int> assignToNearestCentroids(const fileData& data, const std::vector<unsigned char\*>& centroids)** - Ανάθεση σημείων δεδομένων στο πλησιέστερο κέντρο με τον αλγόριθμο Lloyd
* **int assignToNearestCentroidLSH(LSHImplementation& lsh, fileData& data, int L, int k, std::vector<double>& t, std::vector<unsigned char\*>& centroids, int queryImage)** - Ανάθεση ενός σημείου ερωτήματος στο πλησιέστερο κέντρο χρησιμοποιώντας το LSH
* **void updateCentroidsMacQueen(const fileData& data, const std::vector<int>& assignments, int K, std::vector<unsigned char\*>& new\_centroids)** - Ενημέρωση των κέντρων χρησιμοποιώντας τη μέθοδο του MacQueen.

**Υλοποίηση**

Στο στάδιο που ενημερώνουμε τα κέντρα μετά το βήμα ανάθεσης που έχουμε επιλέξει, τα ενημερωμένα κέντρα εμφανίζονται όλα 0 και δεν καταφέραμε να τυπώσουμε τα σωστά σημεία (από το debugging εντοπίσαμε ένα θέμα στο πως θα ορίσουμε τον τύπο των κέντρων και τις πράξεις που θα κάνουμε σε αυτά μέσω του McQueen αλγορίθμου).