ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα

Χειμερινο εξαμηνο 2020

1η Προγραμματιστικη Εργασια

Αναζητηση και συσταδοποιηση διανυσματων στη C/C++

Αριθμός Μητρώου(ΑΜ):

1115201700217

1115201700203

Ονοματεπωνυμο:

Ορέστης Στεφανου

Λεωνίδας Εφραιμ

Ακαδημαικη Χρονια 2020-2021

ΠΕΡΙΕΧΟΜΕΝΑ

1

ΕΙΣΑΓΩΓΗ

Στα πλέσια της εργασία είχαμε να υλοποίήσουμε τον αλγόριθμο LSH για διανύσματα στον D-διάστατο χώρο, καθώς και τον αλογόριθμο τυχαίας προβολής στον υπερκύβο βάσης της μετρικής Μανχάταν L1. Στην συνέχεια έπρεπε να εκτελέσουμε κάποια querys στο dataset που μας δώθηκε έτσι ώστε να επαληθεύσουμε την σωστή λειτουργία των αλγορίθμων. Τέλος κληθήκαμε να υλοποιήσουμε τους αλγόριρθμους για την συσταδοποίση διανυσμάτων βάση της μετρικής Μανχάταν όπου η ανάθεση θα έπρεπε να γίνει με τον αλγόριθμο του Lloyd's ή με αντίστροφή ανάθεση μέσω Range Search με LSH. Η υλοιποίση της εργασίας έχει γίνει σε C++

2

ΜΕΤΑΓΛΩΤΤΙΣΗ-ΕΚΤΕΛΕΣΗ

Για τις ανάγκες τις εργασία δημιουργήσαμε 3ις main συναρτήσεις όπου οι δύο είναι υπεύθυνες για του αλγόρθιμους LSH και Hypercube, ενώ η τρίτη είναι υπεύθυνη για το Clustering
Η μεταλγώττιση γίνετε με τις παρακάτω εντολές

- **make lsh**
- **make cube**
- **make cluster**

Ενώ η εκτέλεση των προγραμμάτων γίνετε με τις εντόλες που μας δώθηκαν στην εκφώνηση της εργασίας, δηλαδή:

- **LSH**
  ./lsh –d <input file> –q <query file> –k <int> -L <int> -o <output file> -N<number of nearest> -R <radius>
- **HYPER CUBE**
  ./cube –d <input file> –q <query file> –k <int> -M <int> -probes <int> -o<output file> -N <number of nearest> -R <radius>
- **CLUSTERING**
  ./cluster –i <input file> –c <configuration file> -o <output file> -complete <optional> -m <method: Classic OR LSH or Hypercube>

3

ΥΛΟΠΟΙΗΣΗ

3.1 ΕΙΣΟΔΟΣ ΔΕΔΟΜΕΝΩΝ

Για την είσαγωγή των δεδομένων έχουμε δημιουργήσει μια συνάρτηση με το όνομα **ReadData** η οποία δέχετε σαν όρισμα το path με το αρχείο εικόνων και ένα vector όπου στην συνέχεια το γεμίζει με τις είκονες.
Η συνάρτηση αφού ανοίξει το αρχείο διαβάζει διαδοχικά 4 integers όπου αντιπροσοπεύουν αντοίστοιχα

- Το magic number
- Το ύψος της εικόνας της εικόνας
- Το πλάτος της εικόνας της εικόνας
- Τον αριθμό των εικόνων που υπάρχουν στο αρχείο

Αφού ξέρουμε τις διαστάσεις τον εικόνων τώρα μπορούμε να διαβάζουμε N*N chars και να τους αποθηκεύουμε σε μια γραμμη του vector διαδοχικά.
Για την υλοποίση της συνάρτησης ReadData χρειαστήκαμε να υλοποίσουμε ακόμα μια συνάρτηση με όνομα **NumReverse** η οποία πέρνει ένα interger και του αλλάζει το endian του με μερικά shifts γιατί ο αριθμός που υπάρχει στο αρχείο είναι ανάποδα οποτέ πρέπει να αντιστραφεί.

3.2 ΜΕΤΡΙΚΕΣ

Για τις μετρικές δημιουργίσαμε μια κλάση με το όνομα **Metrics** η οποία έχει μια συνάρτηση με το όνομα **get_distance** η οποία δέχετε σαν όρισμα τις 2 είκονες που θέλουμε να βρούμε της απόσταση τους κάθως και ακόμα ενα όριμσα το οποίο είναι το όνομα της μετρική π.χ. L1 για την μετρική Μαχνάταν. Έδω μπορούν να υλοιποιηθούν και άλλές μετρικές αλλα στην εργασία μας ζητήθηκες μόνο η μετρική Μανχάταν.
Για την υλοιπόιση της Μαχάταν μετρικής πήραμε το αθροσμα της απόλυτη τιμή των σημέιων των δύο εικόνων

3.3 HASH TABLE

Για την υλοποίση του Lsh χρειαστικάμε ενα hashtable οπότε δημιουργήσαμε μια κλάση με το όνομα hashtable.Η κλάση αυτή αποτελείτε από τα buckets που είναι ένας πίνακας με vectors, το μέγεθος του πίνακα, τις σταθερές K και W, μια μεταβλητή sRandInit η οποία αρχικοποία την rand για να έχουμε τυχαία s κάθε φορά καθός και ένα vector με vectors το οποίο περιέχει τα s. Στον **constructor** της hashtable αρχικοποιούμε όλες τισ μεταβλητές κάθος και δημιουργόυμε τα τυχαία s όπου τα βάζουμε στο vector. Εχτός από τον constructor το hashtable έχει και τις πάρατάκτω συναρτήσεις

- **hash_function**
  Η συνάρτηση αυτή διμιουργεί την συνάρτηση g(p) συμφωνα με τον αλγόριθμο lsh. Αυτό το κάνει φτίαχονατας μια διαφορική h(p) κάθε φόρα βάση του πάρακατω τύπου

$$h(p) = a_{d-1} + m \cdot a_{d-2} + \cdots + m^{d-1} \cdot a_0 \bmod M \in \mathbb{N},$$

  Στην συνέχεια ενώνει όλες τις h(p) για να δημιουργήσεις την g(p)

$$g(p) = [h_1(p)|h_2(p)|\cdots|h_k(p)] \in \mathbb{N}.$$

- **insert**
  Η συνάρτηση αυτή δέχετε σαν όριμα μια είκονα κάθος και τον αριθμό του bucket που πρέπει να μπει με σκόπο να είσάγει την είκονα αυτή στο κατάλληλο bucket του hashtable

- **get_bucket_imgs**
  Η συνάρτηση αυτη πέρνει σαν όρισμα τον αριθμό κάποιου bucket και επιστρέφει ένα vector με τα στοιχεία αυτού του bucket

3.4 LSH

Ο αλγόριθμος LSH υλοποιήτε μέσο μιας κλάσης με το αντίστοιχο όνομα. Η κλάση αυτή περιέχει τις σταθερές K,L,r , ένα hash table και ένα vector με όλα τα δεδομένα των είκόνων. Στον constructor αρχικοποιούντε όλες οι μεταβλητές. Επίσης δημιουργούντε όλα τα hashfunction και μπένει η κάθε εικόνα στο bucket που τις αντιστοιχεί. Οι συναρτήσεις που υλοιποιούντε στην κλάση LSH είναι οι εξής.

- **nearest_neighbor**
  Αυτή η συνάρτηση δέχετε σαν όρισμα ένα vector με το query και μας επιστρέφει τον ένα pair που περιέχει τον κοντινότερο γείτονα μαζί με την απόσταση που έχει από αυτό τον γείτονα. Η διαδικασία αυτή γίνετε υπολογίζοντας αρχικά το bucket που αντιστοιχεί στο query σε κάθε hashtable και στην συνέχεια πέρνουμε όλα τα στοιχεία που βρισκοντε σε αυτό το bucket στο vector image_indexes. Αφού αποθηκεύσουμε στο img_indexes προσορινά του κοντινούς γείτονες βρίσκουμε την Μανχάταν απόσταση μεταξύ αυτών και του query. Τέλος πέρνουμε την πιο κοντινή απόσταση από όλα και την επιστέφουμε

- **knn**
  Η συνάρτηση αυτή λειτουργεί με παρόμοιο τρόπο με την nearest_neighbor με την μόνη διαφορά αντι να επιστρέψει ένα κοντινό γείτονα επιστρέφει του

K κοντινούς γείτονες. Οπότε εδώ δέχετε σαν όρισμα το query και το K που μας προσδιορίζει τον αριθμό των γειτόνων που θέλουμε να επιστρέψουμε με αποτέλεμσα να επιστρέφει ένα vector με K pairs που περιέχουν τον την απόσταστη και τον N κοντινότερο γείτονα του query

- **range_search**
  Η συνάρτηση range_search βρίσκει τους γείτονες του query απόσταση r. Δέχετε σαν όρισμα το query, την ακτίνα του κύκλου όπου θα γίνει το range search και μια σταθέρα c, όπου αν δεν δόσουμε όρισμα πέρνει deufult τιμή 1. Στην συνέχεια όπως και οι προηγούμενες συνάρτησεις έτσι και η range search βρίσκει το bucket που αντιστοιχεί στο query σε κάθε hashtable και στην συνέχεια πέρνουμε όλα τα στοιχεία που βρισκοντε σε αυτό το bucket στο vector image_indexes. Τώρα για κάθε είκονα ελέχει βάση τις μετρικής Μανχάταν άν βρίκετε εντός της ακτίνας r. Σε περίπτωση που βρίσκετε τότε προσθέτει την εικόνα στο results έτσι ώστε στο τέλος να τις επιστρέψει

- **exact_nearest_neighbor**
  Αυτή η συνάρτηση έχει σκοπό να μας επιστρέψει τον ακριβές πιο κοντινους γείτονες για να ελέξουμε ότι τα αποτελέσματα των παραπάνω συναρτήσεων είναι σωστά. Η διαδιακάσια αυτή γίνετε με την μέθοδο του brute force, δηλαδή ελέχουμε όλες τις είκονές του dataset και επιστρέφουμε τις K είκόνες με την μικρότερη Μαχνάταν απόσταση από το query. Εδώ η συνάρτηση αυτή πέρνει σαν όριμα το query, το K μα επιστέφει ενα vector με pairs όπου το κάθε ζευγάρι αποτελείτε από την απόσταση και στον είκονα που είναι πιο κόντα στο query

3.5 HYPER CUBE

Για την υλοποίση του Hyper Cube δημιουργήσαμε μια κλάση με το όνομα BinaryHyperCube η οποία έχει σαν σταθερές το d,M,probes,R καθός και τρία vectors οποία είναι τα δεδομένα των είκόνων, οι τιμές των s και μια δομή για τον υπερκύβο. Στον **constructor** του υπερκύβου αρχικοποιούνε όλες η μεταβλητές και μπένουν τα

δεδομένα στο data vector. Στην συνέχεια δημιουργούντε με τυχαίο τρόπο τα s και μπένουν στο αντοίστοιχο vector. Τέλος τα δεδομένα hashaροντε και μπένουν στο ανάλογο bucket της δομής του υπερκύβου. Η κλάση BinaryHyperCube υλοιποιέι και τις παρακάτω συναρτήσεις.

- **f**
  Η συνάρτηση F σύμφωνα με την θεωρία για την υλοποίηση του αλογοριθμου του υπερκύβου πρέπει να επιστρέφει 0 ή 1 με ομοιόμορφη κατανομη. Ετσι λοιπόν αποφασίσαμε η συνάρτηση f na δέχετε ένα interger, να πέρνει την διαδική του μορφή και να μετράει πόσα μηδινικά και πόσους άσσους έχει. Αν οι ασσοι είναι περισσότεροι από τα μηδινικά τότε επιστρέψει 1 αλίος επιστρέψει 0.

- **h**
  Η συνάρτηση h είναι η hashfunction που χρησιμοποιά ο υπερκύβος αλλά έιναι ίδια με την συνάρτηση h(p) του Lsh που αναφέραμε πιο πάνω. Οπότε η υλοιποίση είναι η ίδια

- **get_number_from_bits**

- **hamming_distance**
  Αυτή η συνάρτηση μετρά την απόσταση hamming μεταξύ δύο αριθμών. Δηλαδή βρίσκει τον ελάχιστο αριθμό αντικαταστάσεων που χρειάζονται ώστε να μετατραπεί ο ένας αριθμός στο άλλο. Οπότε η συνάρτηση δέχετε δυο integers του σπάζει σε bits και τους βάζει σε δύο arrays αντίστοιχα. Στην συνέχεια συγκρίνει τα bits και μετρά πόσα είναι διαφορετικά. Τέλος επιστρέφει την απόσταση hamming των δύο αριθμών

- **knn**
  Η συνάρτηση αυτή υπολογίζει του K κοντινότερους γείτονες μια εικονάς. Για

αυτό τον υπολογισμό χρειάζετε σαν όρισμα το query και τον αριθμό K για να επιστέψει ένα vector με τα K κοντινότερα ζευγάρια απόστασης-εικόνας. Η υλοποίση είναι παρόμοια με τον knn του Lsh με την μόνη δίαφορά είναι ο υπολογισμός της απόστασης των σημείων που βρίσκοντε στο ίδιο bucket, όπου πρώτα υπολογίζοντε οι κοντινοί γείτονες βάση της απόστασης hamming

- **range_search**
  Η συνάρτηση range_search βρίσκει τους γείτονες του query απόσταση r. Δέχετε σαν όρισμα το query, την ακτίνα του κύκλου όπου θα γίνει το range search και μια σταθέρα c, όπου αν δεν δόσουμε όρισμα πέρνει deufult τιμή 1. Όπως η συνάρτηση knn έτσι και αυτή η συνάρτηση έχει παρόμοια υλοποίση με το range_search του Lsh με την μόνη διαφορά οτί προσεγγίζουμε τους γείτονες με την μικρότερη αποσταση hamming

- **exact_nearest_neighbor**
  Η συνάρτηση exact_nearest_neighbor έχει ακριβός την ίδια υλοιποίση με την συνάρτηση exact_nearest_neighbor της κλάσης Lsh οπού εξηγήτε πιο πάνω

- **get_bucket_imgs**
  Η συνάρτηση αυτη πέρνει σαν όρισμα τον αριθμό κάποιου bucket και επιστρέφει ένα vector με τα στοιχεία αυτού του bucket

3.6 CLUSTERING

Για την υλοιποίση του clustering δημιουργήσαμε μια ξεχωριστή κλάση με το όνομα Clustering και μια καινουρια main. Σε αυτή την κλαση υλοιπούντε οι συνάρτησεις loyds, lsh, hypercube οι οποίοι είναι υπεύθυνοι στο βήμα της ανάθεσης για το clustering. Επίσης υλοποιήτε η συνάρτηση silhouette_score για την αξιολόγιση των αποτελεσμάτων.

3.6.1 Lloyd's (A)

Το clustering με τον ακριβή αλγόριμο του Lloyd's υλοιποιήτε στην συνάρτηση **loyds**. Η συνάρτηση δέχετε ως όρισμα την παράμετρο K που αντοιστοιχεί στο πλήθος των clusters. Στην συνέχεια δημιουργεί ένα vector με τα κεντρικά σημέια των clusters (centroids), ενα vector για να κρατά πρωσωρινά σημεία νέα κεντρικά σημεία και ένα vector που περέχει τα K clusters. Τα κεντρικά σημεία αρχικοιπούντε και στην συνέχεια γίνετε ο υπολογισμός των κενούριων κεντρικών σημείων βάση της μέσης απόστασης όλων των σημέιων του cluster. Η διαδιακασία αυτή συνεχίζετε μεχρι να η απόσταση των νέων κεντρικών σημείων από τα παλία να είναι πολυ μικρή (καθορίζετε από την μεταβλητή centroids_difference)

3.6.2 LSH Range Search (B)

Το clustering με αντίστροφη ανάθεση (reverse) μέσω Range search με Lsh γίνετε στην συνάρτηση **lsh** με την μόνη διαφορά (σε σχέση με τον συνάρτηση loyds) στο βήμα της ανάθεσης των σημιών όπου αναθέτουντε με διαφορετικό τροπό. Η διαφορά είναι ότι αφού έχουν επιλεγέι τα κεντρικά σημεία τότε κάνουμε range search γύρο από το κάθε κεντρικό σημείο και όσα σημεία είναι σε αυτή την ακτίνα τότε μπένουν στο cluster στου σημείου. Στην συνέχεια διπλασιάζουμε την ακτίνα μέχρι να καληψουμε αρκετή επιφάνια των κοντινών σημείων από το κεντρικό σημείο και συνέχιζουμε την αναέωση των κεντρικών κανικά βάση της απόστασης. Σε περίπτωση που ένα σημείο διδκικίτε απο δύο cluster, πηγένει στο cluster με την κοντινότερη Μανχαταν απόσταση. Για την υλοποίση αυτού του σημίου χρησιμοιποιήσαμε ένα vector με δύο στήλες όπου στην πρώτη στήλη βάλαμε ένα flag και στην δεύτερη το cluster στο οποίο ανήκει. Όταν ένα σημείο μπει σε ένα cluster τότε το μάρκάρουμε και σημειώνουμε το cluster που ανήκει έτσι ώστε να γνωρίζουμε όταν πάει να το διεκδικίσει κάποιο αλλο cluster ότι είναι ήδη καταχωριμένο

3.6.3 ΤΥΧΑΙΑ ΠΡΟΒΟΛΗ (Γ)

Το clustering με τυχαία προβολή υλοποιήτε ακριβός με την ίδιο τρόπο με την LSH range search με την μόνο διαφορά ότι στον αλγόριμο ανάθεσης χρησιμοποιούμε την κλαση BinaryHyperCube και στην αναζήτηση των κοντινών σημείων την συνάρτηση cube.range_search

3.6.4 Silhouette