



ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών

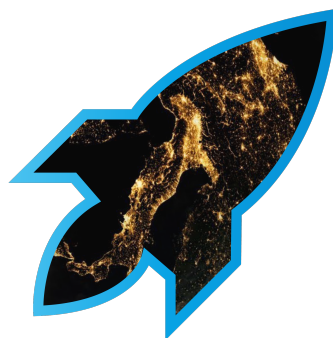
— ΙΔΡΥΘΕΝ ΤΟ 1837 —

ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΑΛΓΟΡΙΘΜΙΚΑ ΠΡΟΒΛΗΜΑΤΑ

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2020

1η ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΖΗΤΗΣΗ ΚΑΙ ΣΥΣΤΑΔΟΠΟΙΗΣΗ ΔΙΑΝΥΣΜΑΤΩΝ ΣΤΗ C/C++



Αριθμός Μητρώου(ΑΜ):

1115201700217

1115201700203

Ονοματεπώνυμο:

Ορέστης ΣΤΕΦΑΝΟΥ

Λεωνίδας ΕΦΡΑΙΜ

ΑΚΑΔΗΜΑΪΚΗ ΧΡΟΝΙΑ 2020-2021

ΠΕΡΙΕΧΟΜΕΝΑ

1	ΕΙΣΑΓΩΓΗ	3
2	ΜΕΤΑΓΛΩΤΤΙΣΗ-ΕΚΤΕΛΕΣΗ	4
3	ΥΛΟΠΟΙΗΣΗ	5
3.1	ΕΙΣΟΔΟΣ ΔΕΔΟΜΕΝΩΝ	5
3.2	ΜΕΤΡΙΚΕΣ	6
3.3	HASH TABLE	6
3.4	LSH	7
3.5	HYPER CUBE	8
3.6	CLUSTERING	10
3.6.1	Lloyd's (A)	11
3.6.2	LSH Range Search (B)	11
3.6.3	ΤΥΧΑΙΑ ΠΡΟΒΟΛΗ (Γ)	12
3.6.4	Silhouette	12

ΕΙΣΑΓΩΓΗ

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

ΜΕΤΑΓΛΩΤΤΙΣΗ-ΕΚΤΕΛΕΣΗ

Για τις ανάγκες της εργασίας δημιουργήσαμε 3ις main συναρτήσεις όπου οι δύο είναι υπεύθυνες για του αλγόριθμους LSH και Hypercube, ενώ η τρίτη είναι υπεύθυνη για το Clustering

Η μεταγλώττιση γίνεται με τις παρακάτω εντολές

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

Ενώ η εκτέλεση των προγραμμάτων γίνεται με τις εντολές 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>
```

ΥΛΟΠΟΙΗΣΗ

3.1 ΕΙΣΟΔΟΣ ΔΕΔΟΜΕΝΩΝ

Για την εισαγωγή των δεδομένων έχουμε δημιουργήσει μια συνάρτηση με το όνομα **ReadData** η οποία δέχεται σαν όρισμα το path με το αρχείο εικόνων και ένα vector όπου στην συνέχεια το γεμίζει με τις εικόνες.

Η συνάρτηση αφού ανοίξει το αρχείο διαβάζει διαδοχικά 4 integers όπου αντιπροσωπεύουν αντιστοιχία

- 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 των εικόνων τώρα μπορούμε να διαβάζουμε $N*N$ chars και να τους αποθηκεύουμε σε μια γραμμή του vector διαδοχικά.

Για την υλοποίηση της συνάρτησης ReadData χρειαστήκαμε να υλοποιήσουμε ακόμα μια συνάρτηση με όνομα **NumReverse** η οποία πέρνει ένα integer και του αλλάζει το endian του με μερικά shifts γιατί ο αριθμός που υπάρχει στο αρχείο είναι ανάποδα οπότε 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 δέχεται σαν όρισμα τις 2 εικόνες που θέλουμε να βρούμε της απόσταση τους καθώς και ακόμα ένα όρισμα το οποίο είναι το όνομα της μετρική π.χ. L1 για την μετρική Μαχνάταν. Έδω μπορούν να υλοποιηθούν και άλλες μετρικές αλλά στην εργασία μας ζητήθηκαν μόνο η μετρική Μανχάταν.

Για την υλοποίηση της Μαχάταν μετρικής πήραμε το άθροισμα της απόλυτη τιμή των σημείων των δύο εικόνων

3.3 HASH TABLE

Για την υλοποίηση του Lsh χρειαστικάμε ένα hashtable οπότε δημιουργήσαμε μια κλάση με το όνομα hashtable. Η κλάση αυτή αποτελείται από τα buckets που είναι ένας πίνακας με vectors, το μέγεθος του πίνακα, τις σταθερές K και W, μια μεταβλητή sRandInit η οποία αρχικοποιά την rand για να έχουμε τυχαία s κάθε φορά καθώς και ένα vector με vectors το οποίο περιέχει τα s. Στον **constructor** της hashtable αρχικοποιούμε όλες τις μεταβλητές καθώς και δημιουργούμε τα τυχαία s όπου τα βάζουμε στο vector. Εκτός από τον constructor το hashtable έχει και τις παράτακτω συναρτήσεις

- **hash_function**

Η συνάρτηση αυτή δημιουργεί την συνάρτηση $g(p)$ σύμφωνα με τον αλγόριθμο lsh. Αυτό το κάνει φτιάχνοντας μια διαφορετική $h(p)$ κάθε φορά βάση του παρακάτω τύπου

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

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

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

- **insert**

Η συνάρτηση αυτή δέχεται σαν όριμα μια εικόνα κάθος και τον αριθμό του bucket που πρέπει να μπει με σκόπο να εισάγει την εικόνα αυτή στο κατάλληλο bucket του hashtable

- **get_bucket_imgs**

Η συνάρτηση αυτή 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 εικόνων. Στον constructor αρχικοποιούνται όλες οι μεταβλητές. Επίσης δημιουργούνται όλα τα hashfunction και μπένει η κάθε εικόνα στο bucket που τις αντιστοιχεί. Οι συναρτήσεις που υλοποιούνται στην κλάση LSH είναι οι εξής.

- **nearest_neighbor**

Αυτή η συνάρτηση δέχεται 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 γίνεται υπολογίζοντας αρχικά το bucket που αντιστοιχεί στο query σε κάθε hashtable και στην συνέχεια πέρνουμε όλα τα στοιχεία που βρίσκονται 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 αποτέλεσμα να επιστρέφει ένα vector με K pairs που περιέχουν τον την απόσταση 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 σταθερά c, όπου αν δεν δόσουμε όρισμα πέρνει default 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 βρίσκονται σε αυτό το bucket στο vector image_indexes. Τώρα για κάθε εικόνα ελέγχει βάση τις μετρικής Μανχάταν αν βρίσκετε 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 διαδικασία αυτή γίνεται με την μέθοδο του brute force, δηλαδή ελέγχουμε όλες τις εικόνες του dataset και επιστρέφουμε τις K εικόνες με την μικρότερη Μανχάταν απόσταση από το query. Εδώ η συνάρτηση αυτή πέρνει σαν όρισμα το query, το K μα επιστέφει ένα vector με pairs όπου το κάθε ζευγάρι αποτελείται από την απόσταση και στον εικόνα που είναι πιο κόντα στο query

3.5 HYPER CUBE

Για την υλοποίηση του Hyper Cube δημιουργήσαμε μια κλάση με το όνομα BinaryHyperCube η οποία έχει σαν σταθερές το d, M, probes, R καθώς και τρία vectors οποία είναι τα δεδομένα των εικόνων, 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 δημιουργούνται με τυχαίο τρόπο τα s και μπένουν στο αντιστοιχο vector. Τέλος τα δεδομένα hasharοντε και μπένουν στο ανάλογο bucket της δομής του υπερκύβου. Η κλάση BinaryHyperCube υλοποιεί και τις παρακάτω συναρτήσεις.

- **f**

Η συνάρτηση F σύμφωνα με την θεωρία για την υλοποίηση του αλγοριθμου του υπερκύβου πρέπει να επιστρέφει 0 ή 1 με ομοιόμορφη κατανομη. Ετσι λοιπόν αποφασίσαμε η συνάρτηση f να δέχεται ένα integer, 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 αλλιώς επιστρέψει 0.

- **h**

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

- **get_number_from_bits**

Η συνάρτηση αυτή πέρνει σαν όρισμα ένα vector και επιστρέφει των αριθμών των bits με σκόπο να γνωρίζουμε τον αριθμό του bucket που ζητούμε

- **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 δέχεται 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 χρειάζεστε σαν όρισμα το query και τον αριθμό K για να επιστέψει ένα vector με τα K κοντινότερα ζευγάρια απόστασης-εικόνας. Η υλοποίηση είναι παρόμοια με τον knn του Lsh με την μόνη διαφορά είναι ο υπολογισμός της απόστασης των σημείων που βρίσκονται στο ίδιο bucket, όπου πρώτα υπολογίζονται 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 σταθερά c, όπου αν δεν δόσουμε όρισμα πέρνει default τιμή 1. Όπως η συνάρτηση knn έτσι και αυτή η συνάρτηση έχει παρόμοια υλοποίηση με το range_search του Lsh με την μόνη διαφορά ότι προσεγγίζουμε τους γείτονες με την μικρότερη αποσταση hamming

- **exact_nearest_neighbor**

Η συνάρτηση exact_nearest_neighbor έχει ακριβώς την ίδια υλοποίηση με την συνάρτηση exact_nearest_neighbor της κλάσης Lsh όπου εξηγήτε πιο πάνω

- **get_bucket_imgs**

Η συνάρτηση αυτή πέρνει σαν όρισμα τον αριθμό κάποιου bucket και επιστρέφει ένα vector με τα στοιχεία αυτού του bucket

3.6 CLUSTERING

Για την υλοποίηση του clustering δημιουργήσαμε μια ξεχωριστή κλάση με το όνομα Clustering και μια καινούρια main. Σε αυτή την κλάση υλοποιούνται οι συνάρτησεις lloyd, lsh, hypercube οι οποίοι είναι υπεύθυνοι στο βήμα της ανάθεσης για το clustering. Επίσης υλοποιήτε η συνάρτηση silhouette_score για την αξιολόγηση των αποτελεσμάτων.

3.6.1 LLOYD'S (A)

To clustering με τον ακριβή αλγόριθμο του Lloyd's υλοποιήτε στην συνάρτηση **loyds**. Η συνάρτηση δέχεται ως όρισμα την παράμετρο K που αντιστοιχεί στο πλήθος των clusters. Στην συνέχεια δημιουργεί ένα vector με τα κεντρικά σημεία των clusters (centroids), ένα vector για να κρατά προσωρινά σημεία νέα κεντρικά σημεία και ένα vector που περιέχει τα K clusters. Τα κεντρικά σημεία αρχικοποιούνται και στην συνέχεια γίνεται ο υπολογισμός των κενούριων κεντρικών σημείων βάση της μέσης απόστασης όλων των σημείων του cluster. Η διαδικασία αυτή συνεχίζεται μέχρι να η απόσταση των νέων κεντρικών σημείων από τα παλιά να είναι πολύ μικρή (καθορίζετε από την μεταβλητή centroids_difference)

3.6.2 LSH RANGE SEARCH (B)

To clustering με αντίστροφη ανάθεση (reverse) μέσω Range search με Lsh γίνεται στην συνάρτηση **lsh** με την μόνη διαφορά (σε σχέση με τον συνάρτηση **loyds**) στο βήμα της ανάθεσης των σημείων όπου αναθέτουντε με διαφορετικό τρόπο. Η διαφορά είναι ότι αφού έχουν επιλεγεί 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 του σημείου. Στην συνέχεια διπλασιάζουμε την ακτίνα μέχρι να καληψουμε αρκετή επιφάνια των κοντινών σημείων από το κεντρικό σημείο και συνεχίζουμε την αναέωση των κεντρικών κανικά βάση της απόστασης. Σε περίπτωση που ένα σημείο διδικικίτε απο δύο cluster, πηγένει στο cluster με την κοντινότερη Μανχαταν απόσταση. Για την υλοποίηση αυτού του σημείου χρησιμοποιήσαμε 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 διεκδικήσει κάποιο άλλο cluster ότι είναι ήδη καταχωρισμένο

3.6.3 ΤΥΧΑΙΑ ΠΡΟΒΟΛΗ (Γ)

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

3.6.4 SILHOUETTE