Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα (DI352)

Εργασία 1

ΟΜΑΔΑ 3:
Κωνσταντίνος Μικρούδης
1115201700089
Λουκάς Μποζίκης
1115201700098

---make---
Η μεταγλώττιση γίνεται με τις εντολές:
make lsh
make cube
make cluster
Η καθεμία παράγει ομώνυμο εκτελέσιμο με την ακριβή λειτουργικότητα της εκφώνησης.
Καλώντας την make χωρίς όρισμα παράγει και τα τρια εκτελέσιμα.

-----folder dstructs----------
Βασικές Δομές δεδομένων που χρησιμοποιήθηκαν:
Τα αρχεία δηλώσεων και υλοποίησης βρίσκονται στο φάκελο dstructs.
Όλες οι δομές έχουν ως τύπο δεδομένων το (void*) έτσι ώστε να
γενικεύονται για κάθε τύπο αντικειμένου.
Η διαγραφή των στοιχείων απο τις δομές σε περίπτωση instance δεν είναι ευθύνη της δομής,
θα πρέπει να γινει απο τον προγραμματιστή.

-Υλοποιήθηκε μια λίστα (class List) εσωτερικά με πίνακα.
Υποστηρίζει τις μεθόδους:
add(void *); // Προσθέτει στοιχεια στο τέλος της λίστας (amortized O(1))
get(int i); // Επιστρέφει το i-οστό στοιχείο της λίστας O(1)
reset(); // Αδειάζει τη λίστα [χωρίς να διαγράψει τα στοιχεια] O(1)

-Υλοποιήθηκε ένας πίνακας κατακερματισμού (class HashTable) εσωτερικά με πίνακα απο λίστες.
Η ενθυλάκωση των στοιχείων μαζί με το κλειδί (long long) γίνεται μέσω της struct HashElement.
Υποστηρίζει τις μεθόδους:
add(void *,int); // Εισάγει ενα στοιχείο μαζι με το κλειδί του στον πινακα
exists(void *, int) // Ελέγχει αν το εν λόγω στοιχείο υπάρχει εντός του πίνακα με το κλειδι που δόθηκε. Επιστρέφει τιμή αληθείας.
getChain(int); // Επιστρέφει τη λίστα στην οποία αντιστοιχεί το κλειδί που δόθηκε.
getAllChains(); // Επιστρέφει όλο τον πίνακα με τις λίστες.

-Υλοποιήθηκε μια ουρά προτεραιότητας (class PriorityQueue) εσωτερικά με heap.
Η ουρά αυτή δεν επιτρέπει διπλή εισαγωγή στοιχείων και αυτό επιτυγχάνεται εσωτερικά
με ενα hash table. Αυτό επιτρέπει τη χρήση της για την ευρέση nn χωρίς προβλήματα.
Η ενθυλάκωση των στοιχείων μαζι με την προταιρεότητα γίνεται μέσω της struct QueueElement.
Υποστηρίζει τις μεθόδους:
add(void*, float); // Προσθέτει το στοιχείο μαζι με την προτεραιότητά του στην ουρα
peek(); // Επιστρέφει χωρις να αφαιρέσει το πρώτο στοιχείο
peekPriority(); // Επιστρέφει την προταιρεότητα του πρώτου στοιχείου
remove(); // Αφαιρεί και επιστρέφει το πρώτο στοιχείο της ουράς

-----PROBLEM SPECIFIC ΛΕΠΤΟΜΈΡΕΙΕΣ ΥΛΟΠΟΊΗΣΗΣ------

Η αναπαράσταση των διανυσμάτων έγινε με την class Vector, η οποία εκτός απο τις συντεταγμένες υποστηρίζει αποθήκευση του ονοματος (label) του διανύσματος
Διαθέτει επίσης μεταβλητές για την αποθήκευση του cluster στο οποίο έχει ενταχθεί (για την υλοποίηση του ερωτήματος B)

Η αναπαράσταση και διαφοροποίηση της μετρικής γίνεται μεσω της κλάσης Metric, μιας abstract κλάσης που διαθέτει τη μέθοδο dist, η οποία υλοποιείται στο παιδί της (L2Metric) και μπορεί να επεκταθεί αν χρειαστεί με άλλη κλάση.

Για την ταξιθέτηση στοιχείων για γρήγορη απάντηση ερωτημάτων υλοποιήθηκαν οι δυο μέθοδοι τις εκφώνησης μέσω κατάλληλων κλάσεων. Αρχικά, ορίσαμε μια θυγατρική abstract κλάση (class PointStruct) η οποία διαθέτει τις αντίστοιχες abstract μεθόδους που υλοποιούνται στα παιδιά της:
addVectorList(List *); // Δεχεται και τοποθετεί μια λίστα απο Vectors στη δομή
approximateNN(Vector*,Metric*); // Επιστρέφει τον nearest neighbor του διανυσματος που δόθηκε.
approximatekNN(Vector*,Metric*); // Επιστρέφει τους nearest neighbors του διανυσματος που δόθηκε εντός μιας PriorityQueue.
approximateRange(float,Vector*,Metric*); // Επιστρέφει τα διανύσματα within range εντοός μιας PriorityQueue.

Η κλάση αυτή υιοθετείται απο τις κλάσεις LSH και Hypercube, καθε μια απο τις οποιες υλοποιεί τα ανωθεν με τον δικό της τρόπο. Και οι δυο χρησιμοποιουν τις συναρτήσεις κατακερματισμού διανύσματος που υλοποιούνται στην class LocalityHashFamily.
Η κλάση αυτή ορίζει διανύσματα v και τιμές t τυχαία, και διαθέτει μέθοδο hash(int, Vector*) η οποία μιμειται τις συναρτήσεις h_i() απο τις διαφάνειες.

Για την υλοποίηση των f() για το Hypercube εκμεταλευθήκαμε την ομοιομορφία που παρέχει η rand(), τοποθετώντας στην srand() την τιμή που θέλουμε να μετατρέψουμε και το id της συνάρτησης, εξασφαλίζοντας οτι καθε συνάρτηση θα εχει μοναδική τιμή για κάθε είσοδο.

Τέλος, για το clustering των σημείων κατασκευάστηκε η κλάση LloydClusterer.
Η κλάση αυτή δέχεται στον constructor τη λίστα των σημείων τα οποία πρόκειται να μπουν σε συστάδες, καθώς και μια προαιρετική δομή απάντησης ερωτημάτων (LSH, Hypercube) για την περίπτωση ανάθεσης μεσω Range queries.
Η κλάση διαθέτει τις μεθόδους:

initialization(Metric *); // Αρχικοποιεί τους κεντροειδείς με τη μέθοδο spread out
assignment(Metric *); // Επανατοποθετεί τα διανύσματα σε συστάδες. Αυτό ανάλογα με το είδος του clusterer γινεται είτε με την directAssignment() η την rangeAssignment()
update(); // Μετακινεί τους κεντροειδής στο κέντρο μάζας της τελευταίας ανάθεσης. Επιστρέφει true εφόσον δεν πραγματοποιηθεί αλλαγή, η εφόσον η αλλαγή είναι πολυ μικρή
performClustering(Metric*, int); // Εκτελεί την διαδικασία του clustering όσες φορές δοθούν ως παράμετρος, η μέχρι η update() να σηματοδοτήσει τερματισμό.
calculateSillouete(Metric *); // Υπολογίζει για καθε cluster και συνολικά το δείκτη συσταδοποίησης sillouete.
getCluster(int); // Επιστρέφει τα περιεχόμενα του cluster ως λίστα
getCentroid(int); // Επιστρέφει τον αντιστοιχο κεντροειδή

getSillouete(int); // Εφόσον εχει εκτελεστει η calculateSillouete(), επιστρέφει τον δείκτη για το cluster που δόθηκε ως ορισμα.
getGlobalSillouete() // Εφόσον έχει εκτελεστει η calculateSillouete(), επιστρέφει το δείκτη συνολικά.

Τέλος, η ανάγνωση αρχείων διανύσματος έγινε μεσω της κλάσης VectorFile, η οποία στον constructor δέχεται ενα filename και επιστρέφει την λίστα με τα διανύσματα καλώντας την μέθοδο getVectorList();
