Ανάπτυξη λογισμικού για αλγοριθμικά Προβλήματα
Εργασία 1

Ομάδα:
Τουρνάκη Αθανασία: 115201600172
Μυλωνόπουλος Δημήτριος: 1115201600112

Στην εργασία αυτή μας ζητήθηκε να αναπτύξουμε τους αλγορίθμους εύρεσης γειτόνων **LSH και Hypercube**, τόσο για δεδομένα σημείων όσο και για δεδομένα καμπυλών.

Σχετικά με την υλοποίηση της εργασίας:

Για την υλοποίηση της χρησιμοποιήθηκε η γλώσσα C++ (std = 14) ώστε να εκμεταλλευτούμε την ευκολία μοντελοποίησης προγραμμάτων του αντικειμενοστραφούς προγραμματισμού. Επίσης, χρησιμοποιήθηκαν βιβλιοθήκες της STL, τόσο για την αποθήκευση δεδομένων, όσο και για την επεξεργασία τους. Για την ελαχιστοποίηση της επανάληψης κώδικα χρησιμοποίηθηκαν templates.

Περιγραφή αρχείων:

dataStructs.hpp: Το συγκεκριμένο αρχείο περιέχει τον ορισμό των κλάσεων για την αποθήκευση των σημείων και των καμπυλών. Ως σύμβαση έχουμε στις κλάσεις σημείων (class Point) έναν δείκτη σε κλάσεις καμπυλών αν αυτά τα σημεία προκύψουν από τη δημιουργία διανυσμάτων του β' σκέλους της εργασίας.

fileReading.cpp: Περιέχει την υλοποίηση της κλάσης **Reading** που χρησιμοποιείται για την ανάγνωση των αρχείων εισόδου και τη μετατροπή τους σε vector σημείων ή καμπυλών, με σκοπό την εισαγωγή τους στα αλγοριθμικά μοντέλα του LSH και του Hypercube.

bruteForce.cpp: Περιέχει την υλοποίηση αλγορίθμων που χρησιμοποιούν bruteForce για την εύρεση του κοντινότερου γείτονα σημείου ή καμπύλης. Επίσης, περιέχει τη συνάρτηση **calculateW()** που βρίσκει το W για τις lsh και hypercube μέσω του υπολογισμού της μέσης απόστασης των σημείων ενός δείγματος του dataset με τους πλησιέστερους γείτονές τους.

bruteCurves.cpp & brutePoints.cpp: Αποτελούν τις main εκτελέσιμων που παράγουν τα αποτελέσματα της brute force αναζήτησης σημείων ή καμπυλών και τα αποθηκεύουν κωδικοποιημένα σε αρχείο ώστε να γίνεται γρήγορα η ανάγνωση και σύγκριση των αποτελεσμάτων τους με αυτά των υλοποιημένων αλγορίθμων.

dynamicTimeWarping.cpp: Υλοποίηση του αλγορίθμου σύγκρισης αποστάσεων καμπυλών DTW με χρήση δυναμικού προγραμματισμού.

metrics.cpp: Περιλαμβάνει τις μετρικές $l_1 \kappa\alpha\iota\, l_2$ (Manhattan και Ευκλείδεια) που ζητήθηκαν για την εύρεση των αποστάσεων σημείων.

hashTable.cpp: Το hashtable υλοποιήθηκε ως εξής:

- Για το lsh δημιουργείται hashtable που είναι ένας πίνακας (=buckets) από std::vectors, στον οποίο αποθηκεύονται δείκτες σε class Points ώστε να ελαχιστοποιηθεί η κατανάλωση μνήμης. Το hashtable περιλαμβάνει την hashFunction που έχει παρουσιαστεί στη τάξη, η οποία περιέχει ειδική συνάρτηση για την εξασφάλιση αποφυγής overflow στην ύψωση μεγάλων δυνάμεων.

- Για το hypercube χρησιμοποιείται unordered map αντί για buckets δεδομένου ότι για μεγάλες d' παραμέτρους υπάρχει πιθανότητα να δεσμεύεται περιττός χωρος.

LSH.cpp: Περιέχει την template κλάση LSH η οποία προσαρμόζεται ανάλογα με το αν θα εισαχθούν σε αυτό σημεία ή σημεία που προέρχονται από καμπύλες.
Η συνάρτηση του constructor αρχικοποιεί τα hashtables και εισάγει στα σε αυτα τα σημεία του Dataset.
Η συνάρτηση **findNN()** βρίσκει την προσέγγιση του κοντινότερου γείτονα ενός σημείου ή καμπύλης εκτός από το κομμάτι Bonus που επιστρέφει τους γείτονες με απόσταση μικρότερη ή ίση του Radius (**findRadiusNN()**).

cube.cpp: Παρόμοια υλοποίηση με το LSH.
Να σημειωθεί ότι για την εύρεση των κοντινότερων γειτόνων έχει υλοποιηθεί αλγόριθμος που ελέγχει πρώτα στην ακμή του που καθορίζει η hash function και μετά αναζητά διαδοχικά για κοντινούς γείτονες στις ακμές που απέχουν από αυτή hamming distance 1,2,....k

Για bitstring χρησιμοποιήθηκε unsigned int στο οποίο γίνονται bitwise operations ώστε να μειωθούν οι απαιτήσεις χώρου για την αποθήκευση του κλειδιού κάθε ακμής του υπερκύβου, αλλά και για εύκολη και γρήγορη διαχείριση και επεξεργασια του. Ως επιπρόσθετη βελτίωση αγνοούμε τις γειτονικές ακμές που δεν περιέχουν καταχωρήσεις σημείων.

gridCurve.cpp: Σε αυτό το αρχείο υλοποιήθηκε η διαδικασία προβολής σημείων σε grid με σκοπό τη δημιουργία διανυσμάτων. Χρησιμοποιήθηκε συνάρτηση που υπολογίζει την παράμετρο δέλτα με βάση τη μέση απόσταση των διαδοχικών σημείων των καμπυλών του dataset. Για τον υπολογισμό του w χρησιμοποιήθηκε η συνάρτηση **calculateW()** του **bruteforce.cpp**, εφόσον, ανάλογα με τον πίνακα-παράμετρο τ, οι μέσες αποστάσεις των διανυσμάτων που προήλθαν από καμπύλες ήταν διαφορετικές. Επιπλέον, αφαιρέθηκαν τα συνεχόμενα διπλότυπα ζεύγους συντεταγμένων από το παραχθέν διάνυσμα και χρησιμοποιήθηκε για padding η snapped τιμή της μέγιστης συντεταγμένης που βρέθηκε στο dataset. Κάθε διάνυσμα έχει διαστάσεις όσο το διπλάσιο του μέγιστου αριθμού σημείων σε καμπύλη του dataset.

Οι συναρτήσεις **main*.cpp:** χρησιμοποιούνται για την εκτέλεση των διαφορετικών αλγορίθμων οι οποίες διαβάζουν απευθείας τα αποτελέσματα του bruteForce από αρχείο για γρήγορη εκτέλεση.

Traversals.cpp: το συγκεκριμένο αρχείο παρατέθηκε επιπρόσθετα ως κομμάτι της υλοποίησης του 2β, το οποίο βρίσκει όλα τα relevant traversals για καμπύλες μήκους i,j σε πίνακα MxM με |i-j| < 4, όπως δηλαδή είχε ζητηθεί από την εργασία. (κανονικό compile g++ **Traversals.cpp -g -o traversals)**

Οδηγίες μεταγλώττισης:

Ανακατεύθυνση στον φάκελο **source** και εκτέλεση της εντολής make για να γίνει compile του πηγαίου κώδικα. Εκτέλεση της εντολής **make clean** για καθαρισμό των .o (object files) και εκτελέσιμων αρχείων.

Οδηγίες Εκτέλεσης:

Brute force προσέγγιση (για σύγκριση αποτελεσμάτων):

Σημεία: $./brute -d <input_file> -q <query_file> -o <output_file>

Καμπύλες: $./bruteCurve -d <input_file> -q <query_file> -o <output_file>

LSH σημείων:

$./lsh –d <input_file> –q <query file> –k <int> -L <int> -o <outputfile>

Hypercube σημείων:

$./cube –d <input_file> –q <query file> –k <int> -M <int> -o <outputfile> -p <int_probes>

LSH_LSH:

$./curve_grid_lsh –d <input_file> –q <query file> –k <int> -L <int> -o <outputfile>

LSH_HYPERCUBE:

$./curve_grid_lsh –d <input_file> –q <query file> –k <int> -M <int> -p <int> -L <int> -o <outputfile>