Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα – Εργασία 2

Στέφανος Αντωνίου sdi1800008
Κωνσταντίνος-Μάριος Κοκόλιας sdi1800077

*Τα αρχεία data_handling.cpp, LSH_Vector.cpp, Hypercube.cpp, HashTable_LSH.cpp, HashTable_Hypercube.cpp όπως και τα αντίστοιχα .h είναι τα ίδια με την εργασία 1 (πιθανές αλλαγές αφορούν την μορφοποίηση του χρόνου σε seconds).

Στο αρχείο functions.cpp προσθέθηκαν οι συναρτήσεις που χρειάζονται για το LSH_Frechet (δηλαδή συναρτήσεις τύπου snapping, padding, filtering κλπ).

*Ο τύπος για το snap στην continuous είναι σύμφωνα με τις διαφάνειες, ενώ ο τύπος στην discrete είναι σύμφωνα με αυτόν που έδωσε ο κύριος Χαμόδρακας στο eclass.

Η κλάση LSH_Frechet ακολουθεί αντίστοιχο μοντέλο με την LSH_Vector, με την διαφορά ότι, πριν τοποθετήσει τις καμπύλες στο αντίστοιχο hash table, προσθέτουμε τις διαδικασίες του snapping και padding (στην περίπτωση του continuous και το filtering), έπειτα κάνουμε την συνένωση των συντεταγμένων του κάθε σημείου και εισάγουμε δηλαδή την real_vector_x στο hash table, η οποία όμως περιέχει δείκτες και για την αρχική καμπύλη (για τον υπολογισμό των αποστάσεων) αλλά και την grid_curve.
Οι nearest neighbors και range search δουλεύουν με τον ίδιο τρόπο όπως και στο LSH_Vector και Hypercube, με την διαφορά ότι το curve του query τροποποιείται πριν εισαχθεί, όπως και των αρχικών δεδομένων.

*Για την συνένωση των συντεταγμένων, παρατηρήσαμε ότι ένα απλό concat των x και y δεν φέρνει τόσο καλά αποτελέσματα όσο το άθροισμά τους. (η συνάρτηση concat είναι υλοποιημένη και σχολιασμένη στο πρόγραμμα)
*Για σύγκριση των queries και input curves, ελέγχουμε το id από την πρώτη εργασία (δηλαδή αυτό που υπολογίζει το hash function) και όχι το grid curve τους.

Για το clustering, παραμένουν οι ίδιες συναρτήσεις όπως και στην πρώτη εργασία, και προστίθονται οι assignment_rangeSearch_LSH_Frechet (η οποία είναι η ακριβως ίδια με την αντίστοιχη της LSH_Vector απλώς τώρα αναζητά με βάση την ακτίνα σε ένα LSH_Frechet) και η update_Frechet, η οποία υπολογίζει το mean curve όπως φαίνεται και στις διαφάνειες.

*Η υλοποίηση έγινε, αντί για δέντρα όπως στις διαφάνειες, με vectors, όπου κάθε στοιχείο του vector αναπαριστά ένα φύλλο του δέντρου, και αφού “ενωθούν” όλα τα φύλλα (δηλαδή υπολογιστεί η μέση καμπύλη ανα δύο καμπυλών), επαναλαμβάνουμε για τα νέα φύλλα μέχρι να φτάσουμε στη ρίζα, δηλαδή να μας μείνει ένα μοναδικό σημείο, δηλαδή η τελική μέση καμπύλη.
*Το όριο για να σταματήσει η επανάληψη στις assign functions είναι αυθαίρετα 5, ενώ στις update είναι η ελάχιστη απόσταση μεταξύ δύο κεντροειδών (δοκιμάσαμε και για τη μισή/διπλάσια με παρόμοια αποτελέσματα).

Για την μεταγλώττιση:
make search / cluster αντίστοιχα

Για την εκτέλεση:
Σύμφωνα με την εκφώνηση

*Για την εκτέλεση της αναζήτησης με τον αλγόριθμο Frechet, η καλύτερη παράμετρος για L είναι το 3 (υπάρχουν μικρές διαφορές σε χρόνο και αποτελέσματα).
*Οι βέλτιστες παράμετροι για το cluster είναι στο αρχείο cluster.conf.

*Υπάρχει ένα αρχείο unit testing που ελέγξαμε τέσσερις ενδεικτικές συναρτήσεις.
```
g++ -o test UnitTesting.cpp functions.cpp data_handling.cpp -lcunit
./test
```

*https://github.com/kostiskok/Project_Emiris_Erg2