Ονομα: Θεοδώρα
Επώνυμο: Αναστασίου
Α.Μ : 1115201400236
Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα : Εργασία 1

Παρατηρήσεις:

1)Το πρόγραμμα έχει υλοποιηθεί σε c++ .

2)Στον φάκελο περιέχεται αρχείο Makefile το οποίο μεταγλωτίζει και τα 2 προγράμματα.

3)Εκτελείται με τις εξής εντολές :
make
./lsh –d input_small –q query_small –k 1 -L 5 -o out_put
./cube -d input_small -q query_small -k 5 -M 2 -probes 1 -o cube_file

4)Το έχω τρεξει με valgrind για απαλοιφή errors/leaks

5) το αρχείο εισόδου δέχεται στην πρώτη γραμμή την μέθοδο που θα χρησιμοποιήσε
π.χ euclidean ή cosine (χρειάζεται να υπάρχει ένα space μετά την λέξη κλειδί)

6) το query αρχείο δέχεται στην πρώτη γραμμή το radius π.χ Radius 0.2
(0-1 για cosine και 300-400 euclidean) (χρειάζεται να υπάρχει ένα space μετά τον αριθμό)

Έχω υλοποιήσει τις εξής δομές :

Για το 1ο πρόγραμμα : ./lsh (DataSet.cpp/DataSet.h , Hash_Euclidean.cpp/Hash_Euclidean.h, Hash_Cosine.cpp/Hash_Cosine.h,HF_Bucket.cpp/HF_Bucket.h,Lsh.cpp/Lsh.h,knn.cpp/knn.h, Project.cpp)

- **Items** → που κρατάνε ουσιαστικά μία γραμμή του input_file (ένα id, και ένα vector απο ints για τα διανύσματα του κάθε στοιχείου).
- **DataSet** → στην οποία περιέχεται ένας vectors απο items
- **h** → συναρτήσεις κατακερματισμού της ευκλείδιας ,κρατάνε ένα vector με v float διανύσματα ,t
- **h_Cosine** → συναρτήσεις κατακερματισμού της Cosine, κρατάνε ένα vector με v float διανύσματα
- **Bucket** → κρατάει μια λίστα με τα items τα οποία περιέχονται σε αυτό
- **Hash_Euclidean** → κρατάει ένα vector απο δομές h, ενα vector απο k τιμές Ri, ενα vector απο Buckets, ένα vector που αντιπροσωπεύει την g(τιμές των h ουσιαστικά) ,ένα vector απο items Checked ο οποίος με βοηθά για να μήν έχω διπλότυπα.
- **Hash_Cosine** → κρατάει ένα vector απο δομές h_fun, ενα vector απο k τιμές Ri, ενα vector απο Buckets, ένα vector απο items Checked ο οποίος με βοηθά για να μήν έχω διπλότυπα.

- **LSH** → περιέχει ένα E vector με (L) Hash_Euclidean ή ένα C vector με (L) Hash_Cosine (ανάλογα με το τι θα διαβάσει στην είσοδο του input_file ότι θέλει να υλοποιήσει ο χρήστης.

Αρχικά παίρνει σαν είσοδο ένα αρχείο το οποίο βάζει σε μία δομή Dataset γραμμή-γραμμή. Μετά παίρνει κάθε Item απο την δομή DataSet και αποφασίζει βάση της συνάρτησης κατακερματισμού ανάλογα Ευκλείδιας/Cosine σε ποιό Bucket θα το κατανέμει και το βάζει σε αυτό. Αργότερα παίρνει το query_file και κάνει την ίδια διαδικασία για να βρεί σε ποιό bucket ανήκει κάθε γραμμή/item, βρίσκει μέσα σε αυτό ανάλογα με το radius που δίνεται στην 1η γραμμή του και την απόσταση Ευκλείδιας/Cosine αν είναι κοντινός του γείτονας η όχι, εκεί ψάχνει να βρεί και τον κοντινότερο γείτονα του συγκεκριμένου itemq και τα καταγράφει στο αρχείο εξόδου, μαζί με τους χρόνους εκτέλεσης για lsh και για **nn***.

nn = εξαντλητική αναζήτηση δηλαδή απλά ψάχνει όλο το dataset υπολογίζοντας για το καθένα την απόσταση ούτως ώστε να βρεί τον πιο κοντινό. (δεν περιέχει κατανομή σε bucket δηλαδή , οπότε είναι πιο αργό αφού δεν αποκλείει αντικείμενα και πρέπει να τα κοιτάξει ένα προς ένα.)

Για το 2ο πρόγραμμα : ./cube
(DataSet.cpp/DataSet.h, HF_Bucket.cpp/HF_Bucket.h,
HyperCube.cpp/HyperCube.h,hypercybe.cpp)

- **Items** → που κρατάνε ουσιαστικά μία γραμμή του input_file (ένα id, και ένα vector απο ints για τα διανύσματα του κάθε στοιχείου).
- **DataSet** → στην οποία περιέχεται ένας vectors απο items
- **Bucket** → κρατάει μια λίστα με τα items τα οποία περιέχονται σε αυτό
- **HyperCube** → περιέχει ενα vector με h, ένα vector με Bucket, ένα vector με ένα unordered_map<int><int> ο οποίος με βοηθάει στην αντιστοιχία των h με τα bit που μου επιστρέφοντε (cointos). Ένα vector με item checked ο οποίος με βοηθά για να μήν έχω διπλότυπα.

Για το 2ο μέρος της εργασίας έχω χρησιμοποιήσει πολλά κομμάτια κώδικα απο το 1ο κομμάτι. Αρχικά γίνεται ακριβώς η ίδια διαδικασία κατανομής του dataset και για υπολογισμό του Bucked με χρήση μόνο των h συναρτήσεων και όχι με mods κλπ που είχαμε στην lsh-euclidean. Με το που πάρω ένα αριθμό απο την h τσεκάρω αν υπάρχει στο bitmap και βρίσκω την αντιστοιχεία του αριθμού με bit , σιγά σιγά χτίζω τον αριθμό του bucket συνθέτωντας τα bits και βάζω ξανά τα items στα buckets. Με τον ίδιο τρόπο υπολογίζω σε ποιό bucket ταιριάζει μια μια γραμμή απο το query_file και αργότερα ανάλογα με τα M , probes συνεχίζει. Ψάχνει τους γείτονες του συγκεκριμένου bucket και ψάχνει μέχρι M αντικείμενα και μέχρι probes κορυφές. Αν κάποιο απο τα 2 ξεπεράσει το όριο τερματίζει και συνεχίζει με το επομενο query.
Για αυτό τον υπολογισμό των γειτόνων χρησιμοποίησα κάποιες συναρτήσεις απο τους εξής συνδέσμους καθώς και δεν είχα καθόλου χρόνο να φτιάξω εξ ολοκήρου δικές μου,

//https://stackoverflow.com/questions/18858115/c-long-long-to-char-conversion-function-in-embedded-system

//https://stackoverflow.com/questions/1838368/calculating-the-amount-of-combinations

//https://www.programiz.com/cpp-programming/examples/binary-decimal-convert

Αναφέρομαι στις εξής συναρτήσεις στις οποίες έχω κάνει μικρές αλλαγές ούτως ώστε να ταιριάξουν στην εργασία και τα δικά μου δεδομένα:

- int HyperCube::binaryToDecimal(int n)
- char* lltoa(long long int val, int base);
- unsigned long long int gcd(unsigned long long int x, unsigned long long int y);
- long long int convertDecimalToBinary(int n,unsigned long long int &n_bits);
- unsigned long long int Combinations(unsigned long long int n, unsigned long long int k);

Η διαδικασία ευρεσης γειτόνων έχει ως εξής:
- αφού έχω τον αριθμό του bucket στο οποίο ανήκει το query μου
- τον μετατρέπω σε δυαδικό
- αφού έχω τον αριθμό των bits που αντιπροσωπεύουν το bucket μου
- ψάχνω να βρώ μέχρι ποία τάξη πρέπει να ψάξω
- τρέχω τόσες φορές ώστε το (probes - c(x,y)=x!/x!(y-x)!) (Απο διακριτά -συνδιασμούς του bitstring) το probes να γίνει 0.
- έτσι τελικά έχω το clas μου.
- για class φορές καλώ την συνάρτηση find_neigh η οποία μου επιστρέφει όλους τους γείτονες μέχρι τάξη/clas σε ένα vector.
- for probes
- μετατρέπω τους γείτονες σε δεκαδικό και το βάζω σε συνάρτηση που μου υπολογίζει τους γείτονες σε ακτίνα Radius και τον κοντινότερο μέχρι να ψάξω και σε M στοιχεία.

Συμπεράσματα απο μετρήσεις :

1) Η lsh ευκλείδια χρησιμοποιεί περισσότερο χώρο απο ότι ο hypercube ευκλείδια, αφού έχουμε περισσότερους πίνακες κατακερματισμού άρα περισσότερα buckets και γενιότερα δομές.