# Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα

Χειμερινό εξάμηνο 2016-17

1η Προγραμματιστική Εργασία

Υλοποίηση του Locality Sensitive Hashing (LSH) στη γλώσσα C++

ΟΝ/ΜΟ: Δουμιά Φωτεινή AM: 1115201300039

ΟΝ/ΜΟ: Κωνσταντάκης Δημήτριος AM: 1115201300079

α)Σύντομη Περιγραφή

Το πρόγραμμα δέχεται ένα αρχείο εισόδου , εντοπίζει την μετρική που θα χρησιμοποιήσει και αρχικοποιεί τις δομές ανάλογα με τους απαραίτητους τύπους δεδομένων που είναι συγκεκριμένοι για την κάθε μετρική. Ακολούθως , διαβάζει ένα προς ένα τα αντικείμενα από το αρχείο Input, δεσμεύοντας δυναμικά μνήμη για το καθένα και δημιουργώντας L ίδιους κόμβους. Χρησιμοποιώντας ένα συγκεκριμένο αλγόριθμο (σύμφωνα με τη θεωρία για τη κάθε μετρική) επιτελείται η επιλογή της λίστας στην οποία θα γίνει η εισαγωγή του κάθε κόμβου σε κάθε hashtable. Αφού διαβαστούν όλα τα αντικείμενα(items) από το αρχείο εισαγωγής , τότε γίνεται η ανάγνωση του R και των αντικειμένων από το αρχείο Query. Για κάθε αντικείμενο γίνεται η ίδια διαδικασία που επιτελείται κατά την εισαγωγή, με τη μόνη διαφορά ότι σκοπός είναι να βρούμε τη λίστα στην οποία θα γίνει η αναζήτηση των πλησιέστερων γειτόνων. Γενικά, προκειμένου να μην περιλαμβάνει η αναζήτηση τον έλεγχο πιθανών γειτόνων, για τους οποίους έχει γίνει ήδη έλεγχος , πραγματοποιήθηκε η υλοποίηση μιας τελικής λίστας που περιλαμβάνει όλους τους υποψήφιους γείτονες από τις L λίστες που προέκυψαν . Η εύρεση των γειτόνων σε απόσταση μικρότερη του R γίνεται πρακτικά με τη χρήση αυτής της λίστας, όπου θα μπορούσαμε να βάλουμε να σταματάει μετά από κάποιο αριθμό ελέγχων (3L κόμβους ). Ο χρήστης επιλέγει το πώς θα γίνει η αναζήτηση. Ομοίως, για την αναζήτηση των γειτόνων και του πλησιέστερου χρησιμοποιείται η ίδια συνάρτηση με του LSH , με τη μόνη διαφορά ότι η τελική λίστα περιλαμβάνει όλα τα στοιχεία από όλες τις λίστες του πρώτου Hashtable. Γίνεται και καταμέτρηση του χρόνου εκτέλεσης. Αφού ολοκληρωθεί η εκτέλεση του κώδικα για την πρώτη μετρική, το πρόγραμμα ζητά από τον χρήστη αν θέλει να δώσει νέα ορίσματα για τα αρχεία και τις τιμές k, L. Αν πληκτρολογήσει τους χαρακτήρες ‘y’ , ‘Y’ τότε ζητούνται τα ορίσματα με τη σειρά. Αντιθέτως , τα ‘n’ , ‘N’ τερματίζουν την εκτέλεση.

β) **main.cpp :** Παίρνει τα ορίσματα που δίνονται και ανοίγει τα αντίστοιχα αρχεία. Αφού βρει τη μετρική που χρησιμοποιείται ορίζει την δομή που θα χρειαστεί το

πρόγραμμα(hamming, Euclidean,…). Ακολούθως με τη χρήση της getline διαβάζει από το αρχείο ένα ένα τα αντικείμενα και καλεί για κάθε hashtable (πλήθος L) την αντίστοιχη reader συνάρτηση. Αφού τακτοποιηθούν σε κάθε λίστα τα αντικείμενα υπολογίζεται το radius R και ξεκινά η ανάγνωση των αντικειμένων από το αρχείο Query. Καλείται η συνάρτηση LSH για το κάθε αντικείμενο και με αυτό τον τρόπο ολοκληρώνεται η εκτέλεση. Πριν όμως τον τερματισμό εμφανίζεται μήνυμα για το αν θέλουμε να δώσουμε νέα ορίσματα .Αν δώσουμε τότε επαναλαμβάνεται η εκτέλεση της main αλλιώς τερματίζεται.

**Hamming.h / Euclidean.h / Cosine.h / DistanceMatrix.h:** Έχουν υλοποιηθεί με τον ίδιο τρόπο, με τις διαφορές ότι κάθε δείκτης σε hashtable έχει άλλο τύπο (templates), ενώ η κλάση Euclidean και DistanceMatrix περιλαμβάνουν ένα επιπλέον int **, όπου είναι ή ο πίνακας με τις τυχαίες μεταβλητες (μέγεθος k*L , Euclidean ) ή ο πίνακας με τις αποστάσεις(nItems*nItems, Matrix). Το bit_size είναι ο αριθμός των bits για τον hamming, το dims ο αριθμός των διαστάσεων για Euclidean και Cosine , και το nItems ο αριθμός των αντικειμένων για τον Matrix.

**Hamming.cpp :**

- Hamming_Reader : Αρχικά δημιουργείται ένας νέος (με τύπο char) και στη συνέχεια καλείται η συνάρτηση Insert Node για ένα συγκεκριμένο hashtable, με σκοπό να προστεθεί στην λίστα.
- Hamming_LSH : Αρχικά δημιουργείται ένας νέος κόμβος και στη συνέχεια καλείται η συνάρτηση Hashtable_LSH για κάθε hashtable. Ορίζονται δυναμικά και δύο λίστες (final list) όπου σε αυτές μπαίνουν οι κόμβοι που θεωρούνται υποψήφιοι γείτονες . Η qlist, με τη χρήση της Hashtable_LSH, πρακτικά δείχνει κάθε φορά στην αντίστοιχη λίστα όπου θα έκανε εισαγωγή το αντικείμενο από το Query file και με τη χρήση της Insert_List «εισάγωγουμε» ένα αντίγραφο της λίστας αυτής, με τους κόμβους όμως, που δεν έχουν διαβαστεί, με αποτέλεσμα ένα αντικείμενο από το Input File να είναι μόνο μία φορά μέσα στην τελική λίστα. Παρομοίως για τον υπολογισμό του True κοντινότερου γείτονα καλείται η Hashtable_ALL για την final list ALL η οποία περιλαμβάνει όλες τις λίστες του πρώτου Hashtable. Και για το true και για το LSH καλείται η συνάρτηση Hamming_Distance που επιστρέφει την απόσταση του κοντινότερου γείτονα.Υπολογίζονται και οι χρόνοι εκτέλεσης. Οι λίστες αυτές διαγράφονται με όλους τους κόμβους – αντίγραφα που έχουν.

**Euclidean.cpp / Cosine.cpp / DistanceMatrix.cpp :**

Με την ίδια λογική έχουν υλοποιηθεί και οι Euclidean,Cosine και DistanceMatrix με τις διαφορές ότι οι πρώτες δύο φτιάχνουν κόμβους double οι πρώτες δύο ενώ η τελευταία int. Ακόμη αλλάζουν οι αντίστοιχες συναρτήσεις υπολογισμού απόστασης (Euclidean_Distance , Cosine_Distance , DistanceMatrix_Distance) καθώς και ο τύπος

αυτών των συναρτήσεων (δηλαδή της απόστασης από τον πλησιέστερο γείτονα που υπολογίζουν). Ακόμη αλλάζει ο τύπος των λιστών που δημιουργούν (double, double , int αντίστοιχα).

**Hashtable.h :** Αποτελεί μία γενική κλάση που χρησιμοποιείται από όλες τις μετρικές. Αυτό επιτυγχάνεται με τη χρήση templates (class T1 , class T2). Το T1 αναφέρεται στον τύπο των λιστών και κατεπέκταση στον τύπο των κόμβων και το T2 στο τύπο της κλάσης Heuristic.

**Hashtable.cpp :**

- **Hashtable_Init :** Επιτελεί την αρχικοποίηση για κάθε μετρική. Συγκεκριμένα για τον hamming δημιουργει μία κλάση heuristic με τύπο int *(με μέγεθος = k) για την hfunction όπου σε αυτόν τον πίνακα αποθηκεύονται τυχαίοι k αριθμοί που επιστρέφουν ενα bit από ένα bitstring. Για τις υπόλοιπες μτρικές δημιουργούνται k κλάσεις Heuristic. Επίσης , ακολούθως καλούνται οι αντίστοιχες random συναρτήσεις για την τυχαιοποίηση των αριθμών που υπάρχουν στις κλάσεις Heuristic. Για τον euclidean και τον cosine αποθηκεύεται ένα τυχαίο vector (δηλαδή το hfunction είναι τύπου double * με μέγεθος dims (=100) όπου αποθηκεύονται οι συντεταγμένες ). Για τον Distance Matrix αποθηκεύονται σε ένα πίνακα ο αριθμός των αντικειμένων x1 , x2 (μέγεθος = 2*k , τα πρώτα k στοιχεία είναι τα x1 και τα υπόλοιπα k τα x2, με x1 και x2 διαφορετικά ) . Αυτά αφορούν τα αρχεία **Heuristic.cpp** και **Heuristic.h**, τα οποία είναι κοινά για κάθε μετρική. Το στοιχείο t χρησιμοποιείται μόνο από τον ευκλείδιο.
- **Choose_List :** Κοινή για όλες τις μετρικές με έλεγχο για το ποια χρησιμοποιείται. Σκοπός της είναι η επιλογή της λίστας στην οποία θα εισέλθει ένα αντικείμενο. Γενικά τηρούνται οι κανόνες εισαγωγής σε όλους.
- **Insert_Node :** Καλεί την Choose List , βρίσκει την λίστα και στη συνέχεια κάνει εισαγωγή του αντικειμένου στη λίστα.
- **Hashtable_LSH:** Καλεί την Choose List , βρίσκει την λίστα και στη συνέχεια η λίστα qlist δείχνει σε αυτή τη λίστα όπου θα έμπαινε το αντικείμενο από το Query File.
- **Hashtable_ALL:** Χρησιμοποιείται για την δημιουργία μιας λίστας που θα περιλαμβάνει όλες τις λίστες από το πρώτο hashtable.

**List.h :** Περιλαμβάνει την κλάση List η οποία έχει τα πεδία list_size και start (δείχνει στον πρώτο κόμβο της λίστας). Ορίζεται με template.

**List.cpp :** Η δομή της είναι γενική με τη χρήση των templates. Έχουν υλοποιηθεί οι βασικές συναρτήσεις μίας λίστας (όπως η εισαγωγή ενός νέου στοιχείο στην αρχή), καθώς και οι συναρτήσεις που αναζητούν τους γείτονες με ακτίνα μικρότερη του R*c (το c ορίζεται στις δηλώσεις στο αρχείο main.cpp και είναι ίσο με 1) αλλά και ο κοντινότερος γείτονας , επιστρέφοντας την απόσταση του αντικειμένου από αυτό. Οι υλοποιήσεις έγιναν σύμφωνα με τον υπολογισμό της απόστασης που αναφέρεται για κάθε μετρική. Αυτές οι συναρτήσεις *_Distance χρησιμοποιούνται και από τον LSH αλγόριθμο αλλα και από τον αντίστοιχο που κάνει αναζήτηση σε όλες τις λίστες (True). Η ετικέτα tag (με τιμές 0

ή 1) δείχνει ποιος καλεί αυτές τις συναρτήσεις να καταγράφονται στο αρχείο Output μόνο αυτά που χρειάζονται.

**Node.h :** Χρησιμοποιείται από όλες τις μετρικές. Το T* data είναι αυτό που διαφέρει και ορίζεται με templates (char για hamming (bitstring), double (vectors) Euclidean και Cosine, int (distances) για DistanceMatrix. Ακόμη υπάρχει πεδίο number όπου ορίζεται ένας μοναδικός αριθμός που προσδιορίζει το αντικείμενο, πεδίο size για το μέγεθος του T* data και πεδίο T* next που δείχνει στον επόμενο κόμβο.

**Node.cpp :** Έχουν οριστεί οι βασικές συναρτήσεις για τους κόμβους.

**Random.h/Random.cpp :** Περιλαμβάνονται οι συναρτήσεις Marsaglia, uniform_distribution που επιστρέφει ένα ακέραιο στο (M,N) και uniform_distribution_real που επιστρέφει έναν πραγματικό στο (M,N).

γ) Οδηγίες Μεταγλώττισης. Έχει δημιουργηθεί αρχείο Makefile , οπότε η εντολή make δημιουργεί ένα εκτελέσιμο αρχείο project .

Με την εντολή: ./lsh –d <input file> –q <query file> –k <int> -L <int> -o <output file>

input file = (Files/DataHamming.csv , Files/DataEuclidean.csv , Files/DataCosine.csv , Files/DistanceMatrix.csv)

query file = (Files/QueryHamming.csv , Files/QueryEuclidean.csv (και για Cosine με μικρή αλλαγή στο radius) , Files/QueryDistanceMatrix.csv)

output file = Files/Output.csv , που είναι για όλα.

δ) Το πρόγραμμα εμφανίζει στο Output όλα τα ζητούμενα της άσκησης (αποστάσεις, γείτονες, χρόνους, …) .Θα ζητηθεί επίσης από το χρήστη να επιλέξει αν θέλει να τρέξει τον LSH για τους 3L πρώτους κόμβους (πληκτρολόγηση ‘y’ , αλλιώς ‘n’). Αφού ολοκληρωθεί η εκτέλεση του κώδικα για την πρώτη μετρική, το πρόγραμμα ζητά από τον χρήστη αν θέλει να δώσει νέα ορίσματα για τα αρχεία και τις τιμές k, L. Αν πληκτρολογήσει τους χαρακτήρες ‘y’ , ‘Y’ τότε ζητούνται τα ορίσματα με τη σειρά. Αντιθέτως , τα ‘n’ , ‘N’ τερματίζουν την εκτέλεση. Στην συνέχεια ζητούνται τα input file , query file , k, L που δίνονται με τον ίδιο τρόπο όπως τα arguments της main μόνο με τη διαφορά ότι δίνονται ένα ένα χωρίς το προθέματα ( -d, -q ,…). Ο χρήστης γράφει δηλαδή μετά το μήνυμα "Give the Input File with its path.” πχ Files/DataHamming.csv και ούτω καθεξής.