Ανάπτυξη Λογισμικού

για

Αλγοριθμικά Συστήματα

Εργασία 1

2019-2020

Η Ομάδα:
Κατσαρλίνου Χριστίνα ΑΜ 1115 2015 00068
Κλήτσας Βαγγέλης ΑΜ 1115 2015 00070

Δομή Κώδικα:

Ο φάκελος που παραδώσαμε χωρίζεται σε υποκαταλόγους, ένας για κάθε ερώτημα της εργασίας. Η δομή που ακολουθούν τα αρχεία είναι η εξής:

1) classes.hpp : Περιλαμβάνει όλες τις κλάσεις που δημιουργήσαμε για τους ζητούμενους αλγορίθμους.
2) class_implementation.cpp: Περιλαμβάνει την υλοποίηση των συναρτήσεων των κλάσεων.
3) funs_definition.hpp: Δηλώσεις όλων των συναρτήσεων που χρησιμοποιεί το πρόγραμμα.
4) funs_implementation.cpp: Περιλαμβάνει την υλοποίηση των συναρτήσεων του προγράμματος.

Κλάσεις:

Για τα διανύσματα:
vector_info: Αναπαράσταση διανύσματος. Περιέχει id που έχουμε δημιουργήσει εμείς, το id που περιέχεται στο αρχείο και τις συντεταγμένες του διανύσματος.

hash_node : Κόμβος των buckets. Περιέχει το id και το g που υπολογίζουμε για το διάνυσμα.

bucket : Περιέχει έναν vector από buckets.

trueNN_node & LSH_neig: Γείτονας του διανύσματος. Περιέχει id, την απόσταση από τον κοντινότερο γείτονα και τον χρόνο που χρειάστηκε για να τον βρει.

Για τις καμπύλες:
Point: Αναπαράσταση σημείου, (x, y).

curve: Αναπαράσταση καμπύλης. Περιλαμβάνει τα σημεία της, το μήκοε και το id της.

Για την μέθοδο των projection δημιουργήσαμε κλάσεις για τους αλγορίθμους lsh και hypercube για καλύτερη δομή και ευκολότερη χρήση κώδικα.

Υλοποίηση Αλγορίθμων:

Μέθοδος lsh: (Φάκελος lsh)

Οδηγίες Μεταγλώττισης και Εκτέλεσης:

$ make
$ ./lsh –d <input file> –q <query file> –k <int> -L
<int> -o <output file>
$ ./lsh –d <input file> –q <query file> -o <output file>
$ ./lsh

(Συνάρτηση check_args) Αρχικά το πρόγραμμα ελέγχει τα ορίσματα που δίνονται από τη γραμμή εντολών. Αν παραλείπονται τα ορίσματα k και L τότε παίρνει (μέσω define) τις default τιμές που δίνονται στην εκφώνηση. Αν παραλείπονται και τα ονόματα αρχείων, τότε το πρόγραμμα ζητά από τον χρήστη το path για τα αρχεία αυτά.

Διαβάζουμε τα διανύσματα απο το input file και το query file και τα αποθηκεύουμε σε vector. Στη συνέχεια, υπολογίζουμε όλα τα διανύσματα s που θα χρειαστούμε για όλα τα hashtables του lsh που θα υλοποιήσουμε. Υπολογίζουμε όλες τις συναρτήσεις h (input*l*k το πλήθος) και g που χρειαζόμαστε. Τέλος γεμίζουμε όλους τους l hashtables μέσω όλων των g που έχουμε ήδη αποθηκεύσει.

Για κάθε query που διαβάσαμε, υπολογίζουμε τα l διαφορετικά g με τον ίδιο τρόπο. Πάμε σε καθένα από τα l hashtables στην θέση που υποδεικνύει το αντίστοιχο g και γαι κάθε διάνυσμα που θα βρούμε σε αυτές τις θέσεις, υπολογίζουμε την manhattan απόσταση από το τρέχον query. Από τις αποστάσεις αυτές, φυσικά κρατάμε την μικρότερη.

Για τον αλγόριθμο lsh, έχουμε ορίσει w=10*r, όπου r είναι η ακτίνα του range search. Όταν αυτή δεν δίνεται στο αρχείο των queries, θέτουμε r = μέσος
όρος των ελάχιστων αποστάσεων κάθε query με το datset.
Ως m βάζουμε max{ai}+1. Παρατηρήσαμε ότι ενώ το Average AF κυμαίνεται περίπου στο 1.10, έχοντας γενικά επιλογές γειτόνων κοντά στις βέλτιστες, το Max AF είναι αρκετά μεγάλο, περίπου 3-4. Άρα σε λίγα queries υπάρχει μεγάλη αστοχία, πράγμα που αυξάνει και το Average AF. Αν κάνουμε m++ τότε ο αλγόριθμος (σχεδόν σε κάθε μέτρηση) επιστρέφει τον πραγματικό κοντινότερο γείτονα για κάθε query, αλλά η χρονική πολυπλοκότητα αυξάνεται αρκετά.

Μέθοδος hypercube: (Φάκελος cube)

Οδηγίες Μεταγλώττισης και Εκτέλεσης:

$ make

$./cube –d <input file> –q <query file> –k <int> -M

<int> - probes <int> -o <output file>

$./cube –d <input file> –q <query file> -o <output file>

$./cube

Περιγραφή Προγράμματος:

(Συνάρτηση check_args) Αρχικά το πρόγραμμα ελέγχει τα ορίσματα που δίνονται από τη γραμμή εντολών. Αν παραλείπονται τα ορίσματα k, M και probes, τότε παίρνει (μέσω define) τις default τιμές που δίνονται στην εκφώνηση. Αν παραλείπονται και τα ονόματα αρχείων, τότε το πρόγραμμα ζητά από τον χρήστη το path για τα αρχεία αυτά.

(main γραμμές 28 - 100) Στη συνέχεια ακολουθεί την διαδικασία που περιγράφεται στις διαφάνειες των διαλέξεων. Συγκεκριμένα, αφού δημιουργήσει τα vectors g, με την διαδικασία του lsh, μέσω ενός μηχανισμού f (συνάρτηση calc_f) αντιστοιχεί τις διάφορες τιμές του g σε 0 ή 1. Η απόφαση για το αν κάποια τιμή του g αντιστοιχίζεται σε 0 ή 1, φυσικά αποθηκεύεται ώστε αν έρθει στη συνέχεια η ίδια τιμή να “πάρουμε” την ίδια απόφαση. Για την αποθήκευση αυτή έχει χρησιμοποιηθεί ένας vector μήκους d' που σε κάθε θέση περιέχει ένα map με key το g και value την bool απόφαση 0 ή 1.

Για την αναπαράσταση του p έχουμε χρησιμοποιήσει ενα string που περιλαμβάνει τις τιμές 0 ή 1 που προέκυψαν από την f. Η επιλογή του string ως τύπος για το p έγινε για να διευκολύνουμε την αποθήκευση των διανυσμάτων του dataset σε ένα map με key το p και value το διάνυσμα. Έτσι έχουμε μικρότερη χρονική πολυπλοκότητα στην αναζήτηση.

Αφού διαβάσουμε από το input_file αρχείο όλα τα διανύσματα και τα αποθηκεύσουμε στο map που αναφέρθηκε προηγουμένως, διαβάζουμε από το query_file τα διανύσματα για τα οποία θα αναζητήσουμε γείτονες. Τα διανύσματα αυτά τα αποθηκεύουμε με την σειρά που τα διαβάζουμε από το αρχείο σε ένα vector.

(Συνάρτηση find_NN) Για κάθε ένα από τα διανύσματα, υπολογίζουμε το p όπως ακριβώς κάναμε με τα διανύσματα του dataset. Για το p που βρήκαμε, υπολογίζουμε

όλα τα string p' για τα οποία ισχύει ότι Hamming distance(p , p') = 1 (δηλαδή διαφέρουν κατα 1 bit) και τα τοποθετούμε στο vector near_ver μαζί με το αρχικό p. Για κάθε ένα από τα p που αποθηκεύσαμε στο vector, ψάχνουμε στο map που περιέχει το dataset αν υπάρχει κάποιο διάνυσμα αποθηκευμένο στην θέση με key το p. Αν ναι, υπολογίζουμε την manhattan απόσταση του τρέχοντος query με το διάνυσμα του dataset που βρήκαμε. Από όλες τις αποστάσεις που προκύπτουν, κρατάμε την μικρότερη. Μαζί με την απόσταση αποθηκεύουμε και το χρόνο που χρειάστηκε ώστε να βρεθεί ο εκάστοτε γείτονας.

(Συνάρτηση find_R_NN) Για το range search (bonus), με τον τρόπο που περιγράφηκε παραπάνω, υπολογίζουμε τις αποστάσεις από τα διανύσματα που έχουν τοποθετηθεί στο map σε σημεία με key ένα από τα p που προέκυψαν και κρατάμε αυτές που απέχουν το πολύ R.

(Συνάρτηση write_output) Τέλος, αφού υπολογίσουμε όλα τα στοιχεία που χρειαζόμαστε, τα γράφουμε στο output_file αρχείο με την μορφή που περιγράφεται στην εκφώνηση.

Μέθοδος grid: (Φάκελος grid_lsh & grid_cube)

Οδηγίες Μεταγλώττισης και Εκτέλεσης grid_lsh:

$ make

$./curve_grid_lsh –d <input file> –q <query file> –k_vec <int> -M
<int> -probes <int> -L_grid -o <output file>

$./curve_grid_lsh –d <input file> –q <query file> -o <output file>

$./curve_grid_lsh

Οδηγίες Μεταγλώττισης και Εκτέλεσης grid_cube:

$ make

$./curve_grid_hypercube –d <input file> –q <query file> –k_hypercube <int> -M
<int> -probes <int> -L_grid -o <output file>

$./curve_grid_hypercube –d <input file> –q <query file> -o <output file>

$./curve_grid_hypercube

Περιγραφή Προγράμματος:

(Συνάρτηση check_args) Αρχικά το πρόγραμμα ελέγχει τα ορίσματα που δίνονται από τη γραμμή εντολών. Αν παραλείπονται τα ορίσματα k_hypercube, M και probes, τότε παίρνει (μέσω define) τις default τιμές που δίνονται στην εκφώνηση. Αν παραλείπονται και τα ονόματα αρχείων, τότε το πρόγραμμα ζητά από τον χρήστη το path για τα αρχεία αυτά.

Αρχικά δημιουργείται ένας vector G μεγέθους 2 που περιέχει μόνο την τιμή δ που έχει υπολογιστεί ως delta = 4 * d * m - 7, όπου d = 2 (διάσταση καμπυλων) και m = μικρότερο μήκος καμπύλης του dataset. Στη συνέχεια "μετατρέπουμε" τις καμπύλες σε διανύσματα όπως περιγράφεται στις διαφάνειες των διαλέξεων. Για κάθε καμπύλη δημιουργούνται L_grid διανύσματα.

(Συνάρτηση create_grid_curve) Για κάθε καμπύλη, για κάθε σημείο της (x,y) , υπολογίζουμε τα a1,2 ώστε να ισχύει:
|a1*δ + t - x| + |a2 * δ + t - y| = min. Αποθηκεύουμε τα σημεία που προκύπτουν σε ένα vector. Διαγράφουμε τα σημεία του vector που είναι ίδια σε διαδοχικές θέσεις. Σε ένα νέο vector τοποθετούμε με την σειρά τις συντεταγμένες των σημείων που προέκυψαν. Αν το μέγεθος του vector που προκύπτει είναι μικρότερο από το size που έχουμε ορίσει προσθέτουμε στο τέλος τον αριθμό 10000.

grid_lsh:
Για τα διανύσματα που προέκυψαν ακολουθούμε την διαδικασία του lsh όπως και στο πρόγραμμα lsh. Αυτή τη φορά όμως κάνουμε το lsh για κάθε ένα από τα L_grids και κρατάμε από όλα την μικρότερη απόσταση και τελικά βρίσκουμε τον κοντινότερο γείτονα.

grid_cube:
Για τα διανύσματα που προέκυψαν ακολουθούμε την διαδικασία του hypercube όπως και στο πρόγραμμα cube. Αυτή τη φορά όμως κάνουμε το hypercube για κάθε ένα από τα L_grids και κρατάμε από όλα την μικρότερη απόσταση και τελικά βρίσκουμε τον κοντινότερο γείτονα.

Μέθοδος projections:

Οδηγίες Μεταγλώττισης και Εκτέλεσης proj_lsh:

$ make

$./curve_projection_lsh –d <input file> –q <query file> –k_vec <int> -M

<int> -probes <int> -L_grid -o <output file>
$./curve_projection_lsh –d <input file> –q <query file> -o <output file>

$./curve_projection_lsh

Οδηγίες Μεταγλώττισης και Εκτέλεσης proj_cube:

$ make

$./curve_projection_hypercube –d <input file> –q <query file> –k_vec <int> -M

<int> -probes <int> -L_grid -o <output file>

$./curve_projection_hypercube –d <input file> –q <query file> -o <output file>

$./curve_projection_hypercube

Περιγραφή προγράμματος:

(Συνάρτηση check_args) Αρχικά το πρόγραμμα ελέγχει τα ορίσματα που δίνονται από τη γραμμή εντολών. Αν παραλείπονται τα ορίσματα k_vec, -L_vec και e, τότε παίρνει (μέσω define) τις default τιμές που δίνονται στην εκφώνηση. Αν παραλείπονται και τα ονόματα αρχείων, τότε το πρόγραμμα ζητά από τον χρήστη το path για τα αρχεία αυτά.

Στη συνέχεια, δημιουργεί ένα πίνακα (vector) που σε κάθε θέση i,j περιέχει όλα τα δυνατά traversals δύο καμπυλών μήκους i και j. Για να υπολογίσουμε όλα τα traversals πρέπει να βρούμε τα σημεία που "επιτρέπεται" να "πατήσουν" οι δύο καμπύλες.

Για την εύρεση των σημείων του πίνακα που ακουμπάνε την διαγωνιο ακολουθήθηκε η εξής λογική. Υπολογίζεται η (μαθηματική) εξίσωση ευθείας που θα ένωνε τα σημεία 0,0 και i,j. Στη συνέχεια για κάθε ακέραιο x στο (0, i) υπολογίζεται το y που αντιστοιχεί σύμφωνα με την εξίσωση που υπολογίστηκε προηγουμένως. Από το y που προκύπτει, παίρνουμε το floor και ceil (αν δεν είναι ακέραιος) και δημιουργούμε

τα σημεία (x, floor(y)) και (x, ceil(y)) τα οποία είναι τα σημεία που ακουμπάνε την διαγώνιο. Στα σημεία που προέκυψαν παραπάνω προσθέτουμε τα γειτονικά, δηλαδή αυτά των οποίων οι συντεταγμένες διαφέρουν κατά 1.

Για κάθε καμπύλη που διαβάζουμε από το input_file αρχείο, βρίσκουμε το μήκος της, πάμε στην αντίστοιχη σειρά του πίνακα των traversals και για κάθε traversal που συναντάμε στην συγκεκριμένη σειρά, υπολογίζουμε το G * U όπως περιγράφεται στις διαφάνειες. Έτσι προκύπτουν τα διανύσματα που δίνουμε στη συνέχεια στο lsh ή hypercube.

Δημιουργούμε 1 lsh/hypercube για κάθε traversal που υπάρχει στο M * M πίνακα.

Για κάθε καμπύλη που διαβάζουμε από το query_file αρχείο, δημιουργούμε ένα G * V διάνυσμα για κάθε ένα από τα δυνατά traversals που έχουμε ακολουθώντας αντίστοιχο τρόπο με το dataset. Για κάθε ένα από αυτά τα διανύσματα που προκύπτουν, πάμε στο lsh/hypercube που αντιστοιχεί στο traversal που διαχειριζόμαστε τη δεδομένη στιγμή και αναζητούμε τον γείτονα.