{ Ελευθερία Εκατομμάτη sdi2000048, Θεοδώρα Αρχοντάκη sdi2000014 }

Random.h:

Περιέχει τους ορισμούς των συναρτήσεων “Random_Num_ND”, “Random_Num_Uniform” και “generateRandomNumber_uniform_distribution”.

Random.cpp:

Έχει την υλοποίηση των παραπάνω συναρτήσεων. Η “Random_Num_ND” δημιουργεί έναν τυχαίο αριθμό από μια κανονική κατανομή με μέση τιμή 0 και τυπική απόκλιση 1 και έπειτα στρογγυλοποιεί το αποτέλεσμα στον πλησιέστερο ακέραιο. Η “Random_Num_Uniform” δημιουργεί έναν τυχαίο αριθμό από μια ομοιόμορφη κατανομή. Η “generateRandomNumber_uniform_distribution” φτιάχνει έναν τυχαίο ακέραιο αριθμό με εύρος (0-NUMBER_IMAGES-1) με την uniform κατανομή. Η συνάρτηση αυτή χρησιμοποιείται για την επιλογή ενός τυχαίου κόμβου στο γράφο στην εκτέλεση του GNNS.

Image.h:

Έχουμε ένα struct που αναπαριστά την εικόνα MNIST και αποτελείται από έναν πίνακα διάστασης 784 και σε αυτό θα αποθηκεύουμε τα pixels ώστε να φτιάξουμε το διάνυσμα της εικόνας.

Έχουμε ορίσει μερικούς ακεραίους ως extern οι οποίοι είναι ο αριθμός των εικόνων που έχει το input αρχείο, το k (GNNS), το w το «παράθυρο» που χρησιμοποιούμε για την δημιουργία των συναρτήσεων, το N (αριθμός των πλησιέστερων γειτόνων), R (GNNS), E (GNNS), L (MRNG).

Έχουμε δύο vectors τύπου struct MNIST_Image. Στο ένα αποθηκεύουμε «Images» τα διανύσματα από τις εικόνες που πήραμε από το αρχείο εισόδου (πρακτικά θα περιέχει διανύσματα (με 784 στοιχεία) όσα ο αριθμός των εικόνων που πήραμε για είσοδο). Και ο άλλος vector «Q_Image» θα περιέχει ένα διάνυσμα με 784 το οποίο θα αντιστοιχεί στο διάνυσμα της εικόνας-query. Έχουμε και ένα string το όποιο είναι το όνομα του output αρχείου σε περίπτωση που αυτό δεν δίνεται από τον χρήστη. Και έχουμε και το output αρχείο.

Έχουμε και τον ορισμό δύο συναρτήσεων “Input”, “create_image”.

Image.cpp:

Αρχικοποιούμε με default τιμές το k,N,R,E,L οπότε αν δεν δοθούν τιμές από την γραμμή εντολών θα χρησιμοποιηθούν οι default τιμές.

Η συνάρτηση “create_image” είναι ίδια με αυτή που έχει υλοποιηθεί στα προηγούμενα αρχεία.

Η συνάρτηση “Input” παίρνει σαν όρισμα «int argc, char* argv[]» Ελέγχουμε τα ορίσματα που παίρνουμε από την γραμμή εντολών. Αν ο χρήστης τρέξει το πρόγραμμα με ένα μόνο argument τότε το πρόγραμμα θα ζητά τα αρχεία και τα k,N,R,E,L. Αν για κάποια τιμή που ζητείται από το πρόγραμμα ο χρήστης γράψει “-“ τότε θα χρησιμοποιηθούν οι default τιμές. Αν ο χρήστης δώσει arguments στη γραμμή εντολών θα πρέπει να δοθούν με τον παρακάτω τρόπο.

$ ./graph_search –d <input file> –q <query file> –k <int> -E <int> -R <int> -N <int> -l <int, only for MRNG> -m <1 GNNS, 2 MRNG> -o <output file>

Αν ο χρήστης θέλει να τρέξει τον αλγόριθμο GNNS τότε θα δώσει στη γραμμή εντολών:

$ ./graph_search –d <input file> –q <query file> –k <int> -E <int> -R <int> -N <int> -m 1 -o <output file>

Αν ο χρήστης θέλει να τρέξει τον αλγόριθμο search on graph τότε θα δώσει στη γραμμή εντολών:

$ ./graph_search –d <input file> –q <query file> –k <int> -E <int> -R <int> -N <int> -l <int> -m 2 -o <output file>

LSH.h

Έχουμε τις ίδιες δομές με αυτές που φτιάξαμε στον LSH

Ορίζουμε στο .h αρχείο τις παρακάτω συναρτήσεις: «Hash_initialize» αρχικοποιούμε τα hash tables, «HashFunctions_initialize» αρχικοποιούμε τις hash-functions, «Euclidian_hash» που υπολογίζουμε τις h-functions, «Euclidian_dist» υπολογίζουμε μέσω Ευκλείδειας μετρικής την απόσταση μεταξύ δύο διανυσμάτων διάστασης 784. Η «Hash_clear» που πρακτικά απελευθερώνει μνήμη από τα Hash Tables, η «Euclidian_Nearest_N» βρίσκει τον πιο κοντινό γείτονα με βάση την ευκλείδεια μετρική. Η «N_Nearestneighbors_LSH» βρίσκει τους Ν πλησιέστερους γείτονες ενός query με τον LSH αλγόριθμο. Η συνάρτηση «Euclidian_NN» που υπολογίζει με τον exhaustive αλγόριθμο τους Ν κοντινότερους γείτονες από το query. Η συνάρτηση «Query_Hash» που βρίσκει την θέση μίας εικόνας στα hash tables.

LSH.cpp

Οι συναρτήσεις: «Hash_initialize», «HashFunctions_initialize», «Euclidian_hash», «Euclidian_dist», «Hash_clear», «Euclidian_Nearest_N», «Euclidian_NN» έχουν την ίδια υλοποίηση με αυτή που έχουμε ήδη υλοποιήσει στην προηγούμενη εργασία.

«Query_Hash» και η «N_Nearestneighbors_LSH» είναι και αυτές ίδιες με την προηγούμενη υλοποίηση με την μόνη διαφορά ότι εδώ δεν θα χρησιμοποιήσουμε τον πίνακα Q_Images για το query. Πρακτικά θέλουμε να βρούμε τους Ν κοντινότερους γείτονες της κάθε εικόνας άρα το query σε αυτή την περίπτωση θα είναι ο αριθμός κάποιας εικόνας. Και θα βρεθούν οι Ν κοντινότεροι γείτονες για όλες τις εικόνες.

Graph.h

Έχουμε φτιάξει μία κλάση που αναπαριστά έναν κατευθυνόμενο γράφο. Έχει ως χαρακτηριστικά (private) τον αριθμό των ακμών και ένα διπλό vector που αναπαριστά την λίστα γειτνίασης. Έχουμε και ως public τις συναρτήσεις της κλάσης. Η «DirectedGraph» είναι ο constructor της κλάσης. Η «Edge_init» αρχικοποιεί την δομή των ακμών του γράφου. Η «addEdge» προσθέτει μία ακμή στο γράφο. Η «print_number_vertex» εκτυπώνει τον αριθμό των ακμών και η «printGraph» εκτυπώνει τον γράφο. Έχουμε και την «edges_for_a_vertex» που επιστρέφει τους γείτονες ενός συγκεκριμένου κόμβου. Destructor για την κλάση δεν δημιουργούμε γιατί δεν κάνουμε κάπου δυναμική δέσμευση μνήμης.

Έχουμε ορίσει ως extern έναν γράφο και έχουμε και την συνάρτηση «k_NN_graph» που θα φτιάχνει τον k-nearest-neighbors graph. Έχουμε την συνάρτηση «compareSecondColumn» η οποία χρησιμοποιείται για σύγκριση στη συνάρτηση sort. Η «sort_R» χρησιμοποιείται για την κατασκευή του γράφου MRNG και ταξινομεί τα στοιχεία που έχει ο πίνακας R[p] με βάση την απόστασή τους από το p. Η «distance» υπολογίζει την ευκλείδεια απόσταση μεταξύ δύο εικόνων. Η συνάρτηση «isLongestEdge» επιστρέφει true αν η pr είναι η μεγαλύτερη ακμή στο τρίγωνο prt αλλιώς επιστρέφει false. Η συνάρτηση «MRNG» φτιάχνει έναν MRNG γράφο.

Graph.cpp

Ο constructor της κλάσης «DirectedGraph» αρχικοποιεί τον αριθμό των ακμών και την λίστα γειτνίασης.

Η «Edge_init» αρχικοποιεί τον διπλό vector για την λίστα γειτνίασης κάνοντας resize και clear.

Η «addEdge» προσθέτει μία ακμή στο γράφο.

Η «print_number_vertex», «printGraph» εκτυπώνουν τον αριθμό των κόμβων και τον ίδιο τον γράφο αντίστοιχα.

Η «edges_for_a_vertex» επιστρέφει τους γείτονες ενός συγκεκριμένου κόμβου.

Η «k_NN_graph» αρχικοποιεί την λίστα γειτνίασης καλώντας την κατάλληλη συνάρτηση. Για κάθε εικόνα καλεί την συνάρτηση που εκτελεί τον αλγόριθμο LSH για την εύρεση των N πλησιέστερων γειτόνων. Η συνάρτηση αποθηκεύει τους N γείτονες σε έναν vector «N_points». Πάμε και φτιάχνουμε ακμές ανάμεσα στην εικόνα και σε όλα τα στοιχεία που είναι στο συγκεκριμένο vector. Μετά καθαρίζουμε τον πίνακα N_points και κάνουμε το ίδιο για όλες τις εικόνες. Στο τέλος της συνάρτησης θα έχουμε ακμές από κάθε εικόνα στους N κοντινότερους της γείτονες.

Η «compareSecondColumn» είναι συνάρτηση σύγκρισης που χρησιμοποιείται στην συνάρτηση sort. Και πρακτικά ταξινομεί έναν πίνακα με 2 στήλες ως προς τη δεύτερη στήλη (που είναι αποστάσεις) χωρίς να χαθεί η συσχέτιση με την πρώτη στήλη.

Η «sort_R» δέχεται σαν όρισμα ένα vector int που είναι το R_p και ταξινομεί τις εικόνες από αυτή με την μικρότερη απόσταση από το p προς τη μεγαλύτερη. Υπολογίζει σε έναν πίνακα dist όλες τις αποστάσεις των σημείων του R_p από το p. Φτιάχνει έναν πίνακα με 2 στήλες. Η πρώτη στήλη είναι ο αριθμός της εικόνας και η δεύτερη η απόσταση της από το p. Ταξινομεί αυτόν τον πίνακα με βάση την δεύτερη στήλη (δηλαδή τις αποστάσεις) με την συνάρτηση sort. Και επιστρέφει την πρώτη στήλη που είναι οι εικόνες ταξινομημένες ως προς την απόσταση.

Η «distance» δέχεται σαν όρισμα τους αριθμούς δύο εικόνων και επιστρέφει την ευκλείδεια απόσταση μεταξύ τους.

Η «isLongestEdge» υπολογίζει τις ακμές pr, pt, rt στο τρίγωνο prt και επιστρέφει true αν η pr είναι η μεγαλύτερη ακμή του τριγώνου και false αλλιώς.

Η «MRNG» δέχεται ένα κατευθυνόμενο γράφο, το N (που είναι οι N κοντινότεροι γείτονες που επιστρέφει ο LSH), το k για το LSH και το L για το LSH. Αρχικά για κάθε εικόνα πάμε και υπολογίζουμε το R[p] που είναι όλες οι εικόνες του dataset εκτός από την p.

Με την συνάρτηση sort_R ταξινομούμε τον R[p] με βάση τις αποστάσεις από το p. Στο L[p] αποθηκεύουμε τους N κοντινότερους γείτονες του p που έχουμε βρει κάνοντας LSH. Για κάθε εικόνα r στο R[p] και για κάθε γείτονα t του p αν η pr είναι η μεγαλύτερη ακμή στο τρίγωνο prt τότε κάνουμε το flag false και βγαίνουμε από το loop. Αν το flag μας όμως είναι αληθές τότε προσθέτουμε το r στο L[p] (αν δεν υπάρχει ήδη). Στο τέλος προσθέτουμε τις ακμές από το p προς όλες τις εικόνες-κόμβους που είναι στο L[p].

algorithms.h

Έχουμε τις εξωτερικές μεταβλητές set, Y, U. Και τις παρακάτω συναρτήσεις. Η συνάρτηση «E_neighbors» επιστρέφει τους πρώτους E κοντινότερους γείτονες μίας εικόνας. Η «Min_number_Image» επιστρέφει από τους E κοντινότερους γείτονες μίας εικόνας αυτόν που βρίσκεται πιο κοντά στην εικόνα. Η «GNNS_algorithm» εφαρμόζει τον αλγόριθμο GNNS. Η «OutfileGNNS» καλεί τις συναρτήσεις των αλγορίθμων και υπολογίζει τον χρόνο. Η «Search_On_Graph_algorithm» που εκτελεί τον αλγόριθμο Search on graph. Και την συνάρτηση «OutfileSearch_On_Graph_algorithm» καλεί τις συναρτήσεις των αλγορίθμων και υπολογίζει τον χρόνο.

algorithms.cpp

Η «E_neighbors» δέχεται σαν όρισμα ένα γράφο και τον αριθμό μίας εικόνας (κόμβος στο γράφο). Για την συγκεκριμένη εικόνα βρίσκει από τον γράφο τους k κοντινότερους γείτονες και από αυτούς επιστρέφει τους πρώτους E γείτονες.

Η «Min_number_Image» δέχεται σαν όρισμα ένα vector με E στοιχεία (που είναι οι E καλύτεροι γείτονες μίας εικόνας) και το Q_number. Επιστρέφει τον αριθμό της εικόνας από αυτές τις E εικόνες που έχει την μικρότερη απόσταση από το query.

Η «GNNS_algorithm» εκτελούμε τον αλγόριθμο για R επαναλήψεις. Βρίσκουμε ένα τυχαίο Y[0] (με την uniform distribution) για να ξεκινήσει ο αλγόριθμος. Για T βήματα: βρίσκουμε τους E κοντινότερους γείτονες του Y[t-1] και από αυτούς αποθηκεύουμε σε μία μεταβλητή Y_image τον καλύτερο από αυτούς τους γείτονες. Αν η ευκλείδεια απόσταση του καλύτερου γείτονα είναι μεγαλύτερη από την απόσταση του ίδιου του κόμβου τότε κάνουμε break βγαίνουμε από το for και προσθέτουμε στο set τον ίδιο τον κόμβο. Αν δεν έχει γίνει το break προσθέτουμε τους E γείτονες στο set και βρίσκουμε το Y[t] που θα χρησιμοποιηθεί στην επόμενη επανάληψη και είναι ο γείτονας με την καλύτερη απόσταση από το q. Μετά τις R επαναλήψεις στο set θα έχουμε αποθηκεύσει αρκετές εικόνες από αυτές πρέπει να βρούμε τον καλύτερο γείτονα και τον N-οστό καλύτερο γείτονα. Οπότε πάμε και υπολογίζουμε τις αποστάσεις από το query όλων των σημείων και τις αποθηκεύουμε σε έναν vector U. Φτιάχνουμε έναν πίνακα με δύο στήλες η μία θα έχει τους αριθμούς των εικόνων και η άλλη τις αποστάσεις των εικόνων από το query. Καλούμε την sort και ταξινομούμε την δεύτερη στήλη χωρίς όμως να χαθεί η συσχέτιση εικόνας και απόστασης. Άρα το πρώτο στοιχείο της πρώτης στήλης θα είναι ο καλύτερος γείτονας και το πρώτο στοιχείο της δεύτερης στήλης θα είναι η απόσταση του από το query. Και το (N-1)-οστό στοιχείο της πρώτης στήλης θα είναι ο N-οστός καλύτερος γείτονας και το (N-1)-οστό στοιχείο της δεύτερης στήλης θα είναι η απόσταση του από το query.

Η «OutfileGNNS» καλεί τις συναρτήσεις για την εκτέλεση των αλγορίθμων GNNS, exhaustive search για εύρεση κοντινότερου γείτονα και exhaustive search για εύρεση N κοντινότερων γειτόνων. Υπολογίζει τον χρόνο και κάνει τις κατάλληλες εκτυπώσεις στο αρχείο εξόδου.

Η «Search_On_Graph_algorithm» εκτελεί τον αλγόριθμο search on graph. Έχουμε ένα διπλό vector marked_images ο οποίος έχει δύο στήλες και NUMBER_IMAGES γραμμές. Στην πρώτη στήλη αποθηκεύουμε τους αριθμούς των εικόνων και στη δεύτερη στήλη 0 αν η εικόνα είναι un-marked και 1 αν είναι marked. Στην αρχή όλη η δεύτερη στήλη είναι μηδέν αφού δεν έχουμε τσεκάρει καμία εικόνα. Έχουμε τον ακέραιο p που είναι ο κόμβος στον οποίο είμαστε σε κάθε επανάληψη. Στην αρχή του αλγορίθμου το p αρχικοποιείται με μία τυχαία εικόνα καλώντας την συνάρτηση "generateRandomNumber_uniform_distribution". Βάζουμε αρχικά αυτόν τον κόμβο στο σύνολο set. Για L επαναλήψεις μαρκάρουμε αρχικά το p, μετά με τον αλγόριθμο του LSH βρίσκουμε τους N κοντινότερους γείτονες του p. Για κάθε γείτονα του p αν δεν τον έχουμε ξανά συναντήσει τον βάζουμε στο set. Ταξινομούμε το set με βάση την απόσταση των κόμβων από το query. Αν το set έχει υπερβεί σε μέγεθος το L τότε σβήνουμε τα στοιχεία από το index >= L. Βρίσκουμε το πρώτο un-marked στοιχείο του set και αυτό γίνεται το p για την επόμενη επανάληψη.

Η «OutfileSearch_On_Graph_algorithm» καλεί τις συναρτήσεις για την εκτέλεση των αλγορίθμων search on graph, exhaustive search για εύρεση κοντινότερου γείτονα και exhaustive search για εύρεση N κοντινότερων γειτόνων. Υπολογίζει τον χρόνο και κάνει τις κατάλληλες εκτυπώσεις στο αρχείο εξόδου.

main.cpp

Ανεξάρτητα από την μέθοδο αλγορίθμου που ζητάει ο χρήστης το πρόγραμμα καλεί την συνάρτηση Input για να επεξεργαστεί τα δεδομένα που του δίνει ο χρήστης και κάνει αρχικοποίηση των δομών του LSH. Αν ο χρήστης ζητήσει την μέθοδο GNNS τότε το πρόγραμμα καλεί τις συναρτήσεις για την δημιουργία του k-NN γράφου και την OutfileGNNS για την εκτέλεση του GNNS και των συναρτήσεων με τους exhaustive αλγόριθμους.
Αν ο χρήστης ζητάει τον αλγόριθμο search on graph το πρόγραμμα τον ρωτάει αν θέλει να φτιάξει από την αρχή τον MRNG γράφο ή αν θέλει να χρησιμοποιήσει έναν έτοιμο γράφο για εξοικονόμηση χρόνου.
Αν ο χρήστης θέλει να φτιάξει καινούριο γράφο τότε καλούμε τις συναρτήσεις για τη δημιουργία του γράφου και την εκτέλεση του αλγορίθμου search on graph. Απλώς αποθηκεύουμε και σε ένα αρχείο graph.txt τον γράφο που φτιάξαμε με τη μορφή λίστας γειτνίασης. Δηλαδή η γραμμή i του αρχείου θα περιέχει τους γείτονες της εικόνας-κόμβου i.

Ο έτοιμος γράφος που χρησιμοποιείται αν το ζητήσει ο χρήστης είναι πάντα αποθηκευμένος σε ένα αρχείο με όνομα "graph.txt". Και σε περίπτωση που θέλουμε να φτιάξουμε τον MRNG γράφο από την αρχή τον αποθηκεύουμε στο αρχείο "graph.txt".

Αν ο χρήστης θέλει να χρησιμοποιήσει τον έτοιμο γράφο τότε καλούμε την συνάρτηση για την εκτέλεση του αλγορίθμου search on graph. Απλώς πριν την εκτέλεση του αλγορίθμου πρέπει να γεμίσουμε την κλάση graph που θα χρειαστεί στην συνάρτηση του αλγορίθμου. Ανοίγουμε το αρχείο graph.txt στο οποίο είναι αποθηκευμένος ο γράφος, διαβάζουμε κάθε γραμμή i και προσθέτουμε μία ακμή από το i (που είναι ο αριθμός της εικόνας) προς όλους τους ακεραίους που υπάρχουν στη γραμμή i.

Οδηγίες μεταγλώττισης:

- Για τον αλγόριθμο GNNS, καλούμε: make graph_search έτσι δημιουργείται το εκτελέσιμο αρχείο graph_search. Μπορούμε να το εκτελέσουμε με τους παρακάτω δύο τρόπους:
    - ./graph_search σε αυτήν την περίπτωση το πρόγραμμα θα ζητήσει από τον χρήστη τις διαδρομές για τα κατάλληλα αρχεία και τις τιμές για τα k, E, R, N όπως και την μέθοδο που θα εκτελέσει (1 για GNNS 2 για MRNG).
    - ./graph_search –d <input_file> –q <query_file> –k <int> -E <int> -R <int> -N <int> -m 1 -o <output_file>

    Στην πρώτη περίπτωση αν ο χρήστης δεν θέλει να δώσει κάποια τιμή σε μία μεταβλητή τότε απλά γράφει "-" και σε αυτήν την περίπτωση χρησιμοποιούνται οι default τιμές.

- Για τον αλγόριθμο search on graph, καλούμε: make graph_search έτσι δημιουργείται το εκτελέσιμο αρχείο graph_search. Μπορούμε να το εκτελέσουμε με τους παρακάτω δύο τρόπους:
    - ./graph_search σε αυτήν την περίπτωση το πρόγραμμα θα ζητήσει από τον χρήστη τις διαδρομές για τα κατάλληλα αρχεία και τις τιμές για τα N,L όπως και την μέθοδο που θα εκτελέσει (1 για GNNS 2 για MRNG).
    - ./graph_search –d <input_file> –q <query_file> –k <int> -E <int> -R <int> -N <int> -l <int> -m 2 -o <output_file>

    Στην πρώτη περίπτωση αν ο χρήστης δεν θέλει να δώσει κάποια τιμή σε μία μεταβλητή τότε απλά γράφει "-" και σε αυτήν την περίπτωση χρησιμοποιούνται οι default τιμές.

Εξήγηση κάθε προγράμματος:

- **Random.h**, **Random.cpp** περιέχει τους ορισμούς και την υλοποίηση των συναρτήσεων που θα χρησιμοποιήσουμε για να φτιάξουμε τυχαίους αριθμούς με την Normal, Uniform distribution.
- **Image.h**, I**mage.cpp** έχουμε την συνάρτηση που διαβάζει τα δεδομένα από το binary αρχείο και αποθηκεύουμε τα διανύσματα των εικόνων και του query σε μία extern μεταβλητή. Επίσης έχουμε και την συνάρτηση που διαχειρίζεται και τα δεδομένα που δίνει ο χρήστης.
- **LSH.h**, **LSH.cpp** έχουμε υλοποιήσει τις δομές και τις συναρτήσεις που χρειαζόμαστε για τον αλγόριθμο του LSH και για τους exhaustive αλγορίθμους.
- **Graph.h**, **Graph.cpp** έχουμε την κλάση ενός κατευθυνόμενου γράφου, τις συναρτήσεις της κλάσης (constructor, εκτυπώσεις και συναρτήσεις για προσπέλαση στα private κομμάτια της κλάσης) και έχουμε και την συνάρτηση για την δημιουργία του k-nearest-neighbors graph. Και τις συναρτήσεις για την δημιουργία του MNRG γράφου.
- **algorithms.h**, **algorithms.cpp** περιέχει τις συναρτήσεις και τις δομές που είναι απαραίτητες για την υλοποίηση του αλγορίθμου GNNS και search on graph.

- **main.cpp** περιέχει την main συνάρτηση η οποία αρχικά καλεί την συνάρτηση για να φτιάξουμε τα διανύσματα των εικόνων και του query και διαβάζει και τα δεδομένα που έχει δώσει ο χρήστης. Κάνει αρχικοποίηση των δομών για τον LSH. Αν ο χρήστης ζητάει την εκτέλεση του GNNS τότε η main φτιάχνει τον k-NN graph και καλεί την συνάρτηση για την εκτέλεση του αλγορίθμου. Αν ο χρήστης ζητήσει την εκτέλεση του αλγορίθμου search on graph τότε το πρόγραμμα ρωτάει τον χρήστη αν θέλει να φτιάξει από την αρχή τον γράφο MRNG(αρκετά χρονοβόρο) ή να εκτελέσει τον αλγόριθμο πάνω σε έναν ήδη δημιουργημένο MRNG γράφο.
  Μετά από την πρώτη εκτέλεση των αλγορίθμων το πρόγραμμα ρωτάει τον χρήστη αν θέλει να τερματίσει ή αν θέλει να συνεχίσει με άλλα δεδομένα αν ο χρήστης γράψει "END" τότε τερματίζει αν γράψει "continue" θα του ζητηθούν να ξανά δώσει τις παραμέτρους και θα ξανά εκτελεστούν οι αλγόριθμοι.

Έχουμε και τα παρακάτω αρχεία:

- Το **graph10.txt** περιέχει τον MRNG γράφο από το dataset με τις 10.000 εικόνες και N = 10.
- Το **graph5.txt** περιέχει τον MRNG γράφο από το dataset με τις 10.000 εικόνες και N = 4.
- Το **graph.txt** περιέχει τον MRNG γράφο από το dataset με τις 60.000 εικόνες και N = 5.

Αν ο χρήστης θέλει να τρέξει το πρόγραμμα με έτοιμο γράφο τότε πρέπει απλά στο αντίστοιχο ".txt" αρχείο να δώσει το όνομα «graph.txt» και όταν το πρόγραμμα ρωτάει αν θέλει να χρησιμοποιήσει κάποιο έτοιμο γράφο να γράψει στο terminal "yes". Τότε το πρόγραμμα θα χρησιμοποιήσει τον έτοιμο γράφο που βρίσκεται στο αρχείο "graph.txt".