ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΑΛΓΟΡΙΘΜΙΚΑ ΠΡΟΒΛΗΜΑΤΑ
ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΡΓΑΣΙΑ 2

ΠΑΡΑΛΛΑΓΕΣ ΣΤΙΣ ΜΕΘΟΔΟΥΣ CLUSTERING KMEANS ΚΑΙ KMEDOIDS

ΚΑΤΣΑΝΗΣ ΙΩΑΝΝΗΣ
1115201500066

ΓΕΝΙΚΗ ΠΕΡΙΓΡΑΦΗ

Έχουν υλοποιηθεί όλοι οι ζητούμενοι αλγόριθμοι ομαδοποίησης (clustering) με όσο τον δυνατό πιο αποδοτικό τρόπο. Κάθε μέθοδος των τριών βημάτων (initialization-assignment-update), είναι ανεξάρτητη και διακριτή σε αρχεία.

ΜΕΤΑΓΛΩΤΤΙΣΗ ΚΑΙ ΕΚΤΕΛΕΣΗ ΚΩΔΙΚΑ

Έχει γίνει χρήση Makefile.

Εντολή **make**: Μεταγλώττιση κώδικα.

Εντολή **make clean**: Διαγραφή αντικείμενων αρχείων.

Για την εκτέλεση του κώδικα χρησιμοποιείται η εξής εντολή με τα εξής ορίσματα:

```
./cluster -i <input_file> -c <cluster.conf> -o <output_file>
          -a IAU* -d metric -C 1**
```

*: Το -a είναι ο συνδυασμός που θέλουμε να γίνει το initialization, assignment και update αντίστοιχα με σύνολο τιμών για το I={1,2}={rand,kmeans++},
το A={1,2,3}={lsh,hypercube,Lloyd’s}
και το U={1,2}={kmeans,kmedoids}.

!: Είναι υποχρεωτικό αυτό το όρισμα.

**: Το -C είναι η εντολή για complete output. Αν το θέλουμε γράφουμε -C 1, αλλιώς δεν χρειάζεται να γράψουμε κάτι (ούτε καν -C).

***: Το μέγεθος του διανύσματος των δεδομένων ορίζεται με #define στο αρχείο /configuration/configure.h (αρχικά ορισμένο σε 204 για το μικρό twitter dataset).

ΟΡΓΑΝΩΣΗ ΚΩΔΙΚΑ

```
Project
     assign    // Φάκελος των μεθόδων απόδοσης σημείων σε κέντρα.
          Cube  // Υλοποίηση Υπερκύβου.
               Lsh  // Υλοποίηση LSH
                    lsh.cpp
                    lsh.h
                    lsh_bucket.cpp
                    lsh_bucket.h
                    lsh_hashtable.cpp
                    lsh_hashtable.h
                    lsh_item.cpp
                    lsh_item.h
               nearest_neighbour   // Υλοποίηση range search με
                                   // nearesh neighbour αλγόριθμο.
                    nearest_neighbour.cpp
                    nearest_neighbour.h
               cube.cpp
               cube.h
          lloyd_assign   // Υλοποίηση απόδοσης σημείων σε κέντρα με
                         // τον αλγόριθμο του Lloyd.
               lloyd_assign.cpp
               lloyd_assign.h
     clustering     // Περιέχει την main και την silhouette.
          cluster.cpp
          cluster.h
          silhouette.cpp
          silhouette.h
     configuration  // Υλοποίηση για configuration του συστήματος.
          configure.cpp
          configure.h
          data.cpp
          data.h
     init      // Υλοποίηση μεθόδων αρχικοποίησης.
          Kmeanspp  // Υλοποίηση Kmeans++
               kmeanspp_init.cpp
               kmeanspp_init.h
          Random_selection_of_k_points  // Υλοποίση τυχαίας
                                        // επιλογής εκκίνησης.
               random_init.cpp
               random_init.h
     obj  // Εδώ αποθηκεύονται τα αντικείμενα αρχεία όταν
          // δημιουργούνται. (Αρχικα κενός.)
     Update    // Υλοποίηση μεθόδων ανανέωσης κέντρων.
          kmeans_update  // Υλοποίηση ανανέωσης με kmeans.
               kmeans_update.cpp
               kmeans_update.h
          PAM_a_la_Lloyd      // Υλοποίηση ανανέωσης με τον
                              // βελτιωμένο αλγόριθμο Medoids.
```

pam_lloyd_update.cpp
pam_lloyd_update.h

ΣΧΟΛΙΑΣΜΟΣ ΚΩΔΙΚΑ

Έχουν υλοποιηθεί όλα τα ζητούμενα. Το παραπάνω πλάνο είναι αρκετά βοηθητικό στη πλοήγηση στα κομμάτια του κώδικα.

Όλες οι μονάδες είναι λογικά χωρισμένες σε φακέλους και έπειτα σε αρχεία κώδικα και διεπαφής (.cpp και .h).

Έχει γίνει χρήση βοηθητικών δομών δεδομένων και αλγορίθμων, όταν αυτό είναι δυνατόν, για την καλύτερη απόδοση του προγράμματος.
πχ δομή αποθήκευσης αποστάσεων hashtable,
έλεγχος τερματισμού βασισμένος στην μετακίνηση των κέντρων και ένα ανώφλι επαναλήψεων εμπειρικά επιλεγμένο.

ΣΥΜΠΕΡΑΣΜΑΤΑ ΜΕΤΡΗΣΕΩΝ

ΟΛΟΙ ΟΙ ΣΥΝΔΥΑΣΜΟΙ ΓΙΑ ΕΝΑ FIXED K

	K = 200
1x1x1	0.17589930493286443
1x1x2	0.26514277758453852
1x2x1	0.15246827893581993
1x2x2	0.26958620465858312
1x3x1	0.19420054913278177
1x3x2	0.25784420176806832
2x1x1	0.16819852304538024
2x1x2	0.22637367493759913
2x2x1	0.18952451508012507
2x2x2	0.25627415265219477
2x3x1	0.17169136412023331
2x3x2	0.26208865633788756

Παρατηρείται από το στατικό κ κιόλας η κατα λίγο καλύτερη απόδοση της χρήσης του kmedoids στο update.

ΣΥΓΡΙΣΗ KMEANS VS KMEDOIDS

	KMEANS	KMEDOIDS
K = 2	0.07543668394048759	0.07879886253010343
K = 5	0.09968894326449905	0.07936758406986849
K = 10	0.08047257822526399	0.10373501322713017
K = 50	0.16737294226977916	0.18430950449230695
K = 100	0.16057708123290354	0.20633864545136085
K = 150	0.19951207015475377	0.25638304244346714
K = 200	0.20008817224812053	0.25249806701528159
K = 250	0.22105282190545759	0.28697898237311712
K = 500	0.23200275901003311	0.26218141614037665

Σε μεγάλο εύρος των κ πλέον, παρατηρείται ξανά λίγο καλύτερη απόδοση του kmedoids. Επομένως εκ των πραγμάτων βλέπουμε πως ο kmedoids έχει καλύτερη εφαρμογή για το συγκεκριμένο dataset, από ότι ο kmeans.
Άλλωστε, η βασική διαφορά του kmeans και του kmedoids είναι ότι ο kmeans προσπαθεί να βρεί την μέση λύση χωρίς να λαμβάνει υπόψη του τη διασπορά των τιμών (αρκετά ταχύς), ενώ ο kmedoids προσπαθεί να λάβει υπόψη του την διασπορά κα επιλέγει με βασικό κριτήριο τη διάμεσο και όχι τη μέση τιμή.