# Μάριος Παπαμιχαλόπουλος 1115201400149

# Tools used

- Ubuntu Gnome 17.10
- git
- Valgrind 3.13.0
- C++11

# Compilation

```
make
```

# Ενδεικτικές εκτελέσεις

```
./lsh -d <input_file> -q <query_file> -o <output_file> -k
<int> -L <int>
```

```
./cube -d <input_file> -q <query_file> -o <output_file> -k
 <int> -M <int> -probes <int>
```

Αν ο χρήστης δεν δώσει ορίσματα τότε το πρόγραμμα ζητά από το χρήστη το μονοπάτι των αρχείων καθώς τρέχει και χρησιμοποιούνται οι default τιμές:

**Για το lsh: default k=4, L=5**

**Για το cube: default k=3, M=10, probes=2**

Ο χρήστης μπορεί να δώσει **όσες παραμέτρους θέλει** κατα τη κλήση του προγράμματος. Όποια λείπει ή δίνεται default τιμή ή του ζητείται καθώς το πρόγραμμα τρέχει.

# Περιγραφή προγράμματος

Η φιλοσοφία του προγράμματος είναι να βρίσκεται ο τύπος που θέλουμε να αναλύσουμε (euclidan ή cosine) και ύστερα να δημιουργούμε την δομή που ζητείται με τη χρήση πολυμορφισμού. Με τη βοήθεια της κληρονομικότητας κλάσεων καλείται η σωστή συνάρτηση που θέλουμε.

Για παράδειγμα, έστω ότι θέλουμε cosine LSH. Στην αρχή του προγράμματος εφαρμόζουμε τις παρακάτω εντολές:

```
hash_tableptr = new HashTable<vector<double>>*[L];
for(int i=0; i<L; i++)
    hash_tableptr[i] = new HashTable_COS<vector<double>>(t
ableSize, k, dimensions);
```

Μετά για την εισαγωγή ενός στοιχείου αρκεί απλά να πούμε απλά

```
hash_tableptr[x]->put(...)
```

και το στοιχείο θα εισαχθεί στο κατάλληλο είδος πίνακα.

Επίσης, η φιλοσοφία του προγράμματος είναι να μπορεί να αλλάξει ο τύπος των σημείων που υπάρχουν στο dataset, π.χ. να γίνει int απο double, αρκετά εύκολα λόγω της χρήσης **templates**. Τέλος, το πρόγραμμα μεριμνέι για **overflows**.

# Περιγραφή αρχείων & διεπαφών

- **hash_table.h: Template** που περιέχει 3 κλάσεις. Μια κλάση **γονιός HashTable** που περιέχει pure virtual μεθόδους σχετικα με την τοποθέτηση των στοιχείων και την αναζήτηση sto hash table και οι δύο κλάσεις **παιδιά HashTable_EUC και HashTable_COS**.

- **hash_node.h: Template** που περιγράφει τα nodes του hash table.

- **hyper_cube.h:** Ομοίως ότι και στο hash_table.h μόνο που τώρα έχουμε **HyperCube γονιό** κλάση και **HyperCube_EUC, HyperCube_COS παιδιά**.

- **hyper_node.h: Template** που περιγράφει τα nodes του hypercube.

- **hyperplane.cpp:** Αρχείο που περιέχει 3 κλάσεις που χρησιμοποιούνται για τον υπολογισμό των h. Υπάρχει η κλάση γονιός Hyperplane που περιέχει pure virtual μεθόδους και οι δύο κλάσεις παιδία Hyperplane_EUC και Hyperplane_COS. Η κλάση Hyperplane περιέχει πάντα το τυχαίο διάνυσμα που προκύπτει από κανονική κατανομή (0,1). Η Hyperplane_EUC επεκτείνει την Hyperplane αποθηκέυοντας το t[0,w] και το w.

- **fi.cpp:** Κλάση που χρησιμοποιείται ως **hash_function** για το euclidean LSH και το euclidean binary cube. Για το LSH, αυτή η κλάση χρησιμοποιείται για τον υπολογισμό του hash_value, που βασίζεται στον τυπο **φ** των διαφανειών, ο οποίος είναι το άθροισμα των γινομένων r με τα h mod tableSize. Για να γίνει αυτό καλέι τη συνάρτηση computeH().
  Για το cube, αυτή η συνάρτηση απλά μετατρέπει το h σε 0 ή 1.

- **gi.cpp:** Κλάση που χρησιμοποιείται ως **hash_function** για το cosine LSH και το cosine binary cube. Αυτή η κλάση, χρησιμοποιείται για τον υπολογισμό του hash_value, που βασίζεται σε random projections στο χώρο. Το hash_value είναι concat των 0 ή 1 που προκύπτουν από το εσωτερικό γινόμενο των random projections με το διάνυσμα.

- **lsh.cpp:**
  – Αρχικά, γίνεται η επεξεργασία των παραμέτρων που χρησιμοποιούνται κατα την κληση τον προγράμματος.
  – Υπολογίζονται οι διάφορες χρήσιμες μεταβλητές **type,**

**tableSize και dimensions**, με τη βοήθεια κατάλληλων συναρτήσεων του namespace που ορίζεται στο help_function.h. Το type ισούται με **“EUC”** ‘η **“COS”** ανάλογα την πρώτη γραμμή του **<input_file>**. Με βάση το type υπολογίζεται και το tableSize.
– Δημιουργείται το hash_table μέσω της χρήσης πολυμορφισμού. Ορίζουμε μια μεταβλητη hash_tableptr που είναι δείκτης σε δείκτη σε HashTable<K>. Ανάλογα το type δημιουργείται και ο κατάλληλος τύπος hash_table.
– Ύστερα, ξεκινάμε το διάβασμα του **<input_file>** γραμμή γραμμή. Με τη χρήση ενός vector αποθηκεύουμε το σημείο που διαβάζουμε και το τοποθετούμε στη δομή μας.
– Ξεκινάμε το διάβασμα του **<query_file>** γραμμή γραμμη, όπως και το διάβασμα του **<input_file>**. Κάνουμε τα search που απαιτούνται(NN, ANN, RS) σε κάθε πίνακα L και τα εκτυπώνουμε στο <output_file>. Στο τέλος, εκτυπώνουμε το κλάσμα προσέγγισης και το μέσο χρόνο ANN.

- **cube.cpp:**
  – Αρχικά, γίνεται η επεξεργασία των παραμέτρων που χρησιμοποιούνται κατα την κληση τον προγράμματος.
  – Υπολογίζονται οι διάφορες χρήσιμες μεταβλητές **type, tableSize και dimensions**, με τη βοήθεια κατάλληλων συναρτήσεων του namespace που ορίζεται στο help_function.h. Το type ισούται με **“EUC”** ‘η **“COS”** ανάλογα την πρώτη γραμμή του <input_file>. Με βάση το type υπολογίζεται και το tableSize.

– Δημιουργείται το hyper_cube μέσω της χρήσης πολυμορφισμού. Ορίζουμε μια μεταβλητη hyper_cubeptr που είναι δείκτης σε δείκτη σε HyperCube<K>. Ανάλογα το type δημιουργείται και ο κατάλληλος τύπος hyper_cube.
– Ύστερα, ξεκινάμε το διάβασμα του **<input_file>** γραμμή γραμμή. Με τη χρήση ενός vector αποθηκεύουμε το σημείο που διαβάζουμε και το τοποθετούμε στη δομή μας.
– Ξεκινάμε το διάβασμα του **<query_file>** γραμμή γραμμη, όπως και το διάβασμα του **<input_file>**. Κάνουμε τα search που απαιτούνται(NN, ANN, RS) και τα εκτυπώνουμε στο **<output_file>**. Στο τέλος, εκτυπώνουμε το κλάσμα προσέγγισης και το μέσο χρόνο ANN.

- **help_functions.h:** Διάφορες χρήσιμες συναρτήσεις για να ναι το πρόγραμμα πιο ευκολοδιάβαστο.

# Σύγκριση LSH με προβολή σε υπερκύβο

Για τη σύγκριση χρησιμοποίησα τη μετρική euclidean. Επίσης, χρησιμοποίησα για dataset το αρχείο input_small και αρχείο αναζήτησης το query_small. Εκτέλεσα μερικές φορές το lsh με παραμέτρους **k=3** και **L=5** και το cube με παραμέτρους **k=5**, **M=5000** και **probes=16**. Το κλάσμα προσέγγισης και στις 2 περιπτώσεις κυμαίνεται κοντά στο **1.7**. Κατέληξα στα εξείς συμπεράσματα:

| | **LSH** | **Hypercube** |
|---|---|---|

| | | |
|---|---|---|
| **μνήμη** | 54815689 bytes | 10645216 bytes |
| **χρόνος** | 0.00081861 secs | 0.0113149 secs |

Παρατηρούμε ότι η **προβολή σε υπερκύβο** ενώ χρησιμοποιεί λιγότερη μνήμη είναι πολύ πιο αργή συγκριτικά με το lsh. Αυτο συμβαίνει γιατί το **LSH** βασίζεται στην υλοποίηση πολλών πινάκων για να έχει καλύτερο αποτέλεσμα, αλλα εξετάζει λιγότερα στοιχεία. Ενώ η **προβολή σε υπερκύβο**, για να έχει καλά αποτελέσματα πρέπει να εξετάσει αρκετά στοιχεία, με αποτέλεσμα να είναι πιο αργή.