Ανάπτυξη Λογισμικού Για Αλγοριθμικά Προβλήματα

Χειμερινό Εξάμηνο, Ακαδημαϊκό Έτος 2021-2022

Assignment 2

Ιωάννης Ροβιθάκης 1115201800164

Χρήστος Τσούχλαρης 1115201800204

Github link: https://github.com/rvthak/AlgoProject2

Important note:

We also include our README file from our first assignment to cover the structs that remain the same.

Project Structure

- **main_search.cpp:** Main functions for grid based curve analysis
- **main_cluster.cpp:** Main functions for clustering analysis

- **GridHash.h/cpp:** A combination of normal LSH structures and grids that allows better hashing when it comes to Curves.
- **VCurve.h/cpp:** A representation of our Vector struct as a Curve. Used to simplify a lot of operations, like filtering, padding, etc.
- **DistanceUnitTest.h/cpp:** Unit Tests for Discrete Frechet Distance & Tree
- **CompTree.h/.cpp:** A Complete tree struct used to calculate the Curve Cluster mean

- **hash_cube.h/cpp:** The Hypercube struct & it's helper methods
- **hash_fun.h/cpp:** This file has the H, G & F hash function structs
- **hash_lsh.h/cpp:** The LSH struct & it's helper methods
- **bucket.h/cpp:** The bucket struct that holds vector points and is used for the searching algorithms

- **Vector.h/cpp:** This file holds all the structures for Vectors, Centroids & structures that contain them both and their relationships

- **Args.h/cpp:** Structures that hold the arguments for each task provided by the user
- **shortedList.h/cpp:** The ShortedList structure that is sorts itself when you insert vectors and is used for K-Nearest-Neighbors & Range Search
- **List.h/cpp:** A simplified version of the ShortedList used for Range Search
- **distributions.h/cpp:** Functions for uniform & normal distributions
- **timer.h/cpp:** Timer functions for statistical analysis
- **utils.h/cpp:** Utility functions for all the other files like numerical methods, file manipulation, printing functions

- We have also included the necessary header and source files from the Fred library which was provided which are : config.hpp/cpp, curve.hpp/cpp, frechet.hpp/cpp, interval.hpp/cpp, point.hpp/cpp, simplification.hpp/cpp, types.hpp

Fred Library

We have copied to the project the necessary C++ files and we have removed the unnecessary references to pybind. All the necessary methods are called from our own files and there is no further setup needed from the examiner's part.

GridHash:

A struct that uses the idea of LSH, but implements it in a way more suitable to Curves. Instead of having multiple Hash tables, we have multiple 2D or 1D Grids. Hashing now works by snapping the Curve on the grid (+a bit of pre and post processing -minima_maxima, filtering etc) This struct is designed to be able to handle both Aii and Aiii given the correct arguments.

The struct works like this: A given Vector gets converted to a Curve for easier handling, then it passes through its corresponding hash pipeline (1D or 2D). The resulting vector gets used as a key for lsh hashing in the Multihash Struct. The multihash struct gets used just for Vector storage, the curve similarity is ensured by the grid struct not Multihash. When it comes to calculating Curve distances, we calculate them between the original Curves. The grid-hashed Curves are used only for creating a key and are not stored.

You can see the exact grid based hashing pipelines in the file GridHash.cpp in functions hash_2D and hash_1D.

Discrete Frechet Distance

We first allocate two 2-Dimensional arrays for the discrete distance values and the euclidean distances between points in the curve to use on the mathematical algorithm. After we have initialized the arrays we apply the mathematical formula to fill the 1st array (c table). We then get the actual discrete frechet distance from the last point in the c table.

Mean Cluster Curve - Complete Tree:

We created a Complete tree structure to get approximates of the cluster mean curve in a reduce-like manner. We create a binary complete tree in a way that ensures that we have enough leaves to store all the cluster's currently assigned Curves. After we store the Curves in the tree leaves, we start averaging them up in pairs. When the process completes we end with a good approximation of the mean Curve in the tree root.

We get the average Curve between two curves by getting their discrete frechet distance between them, and then backtracking through the table to find an optimal traversal of the Curves. Finally we average the curve points at every point of the optimal traversal to get the mean curve.

Getting the average Curve as mentioned above, does not guarantee that the resulting vector will have the same dimension with the input Curves themselves. So in order to make lsh hashing later possible (it requires dot product that requires specific vector dimensions), after the root returns the final mean-curve of the whole cluster, we pass it through some post-processing to reduce its size to the needed dimensions, while retaining as much information as possible.

Unit Tests

We are using the CppUnit Unit Testing Library. To install the library run the following command :
sudo apt install libcppunit-dev
We have one created one test suite to ensure that our implementation of the discrete frechet distance is correct. We run tests to cover both edge and normal input cases.

For given Curves p, q, we get the discrete frechet distance between them and compare the result with the discrete frechet distance function that exists in Fred library.

We check for Curve lengths:
p : 1 - q : 1
p : 1 - q : m
p : n - q : 1
p : n - q : m

Notes:

1) The makefile contains all the console arguments (We have our own fine tuned arguments). You can edit them from there
2) Valgrind is integrated in the makefile if you need it
3) When compiling with openmp (by adding flag “-fopenmp” in the makefile CFLAGS, some memory leaks appear for some reason. If we compile **without openmp there are no memory leaks**.

For any further questions/suggestions, feel free to contact us at any time at:
sdi1800164@di.uoa.g

Compilation & Execution

After extracting the project, start by running: **make init**
(Make sure you add the input/query file paths in the makefile variables)
Then you can use any of the options below:

Ai) LSH:
make lsh

Ai) Cube:
make cube

Aii) Discrete Frechet:
make disc

Aiii) Continuous Frechet:
make cont

B) Cluster Classic - Vectors:
make clavec

B) Cluster Classic - Frechet:
make clafre

B) Cluster LSH - Vector:
make lshvec

B) Cluster LSH - Frechet:
make lshfre

B) Cluster Hypercube - Vector:
make cubvec

Run Unit Tests:

make test