NOTE: ***The document is in draft mode till 28/12/2020***

TAU Advanced Topics in Programming - 2021A - Exercise 3

Amir Kirsh, Adam Segoli Schubert

Link to Ex1
Link to Ex2

Requirements and Guidelines

In this exercise you shall add a **simulator** that can run **several Navigation algorithms** with **several GIS libraries**, using multithreading.

Your exercise shall be separated into three parts, each one of them with its own makefile, each part may run independently with another team's implementation of the other parts.

The parts are:

1. The GIS (GIS folder)
2. The Navigation algorithm (Navigation folder)
3. The Simulator (Simulator folder)

Parts [1] and [2] would be compiled as shared libraries (.so) that would be dynamically loaded by the Simulation project.

You are required to organize your submission into **5 folders**: 3 folders for 3 separated projects with a makefile inside each folder (1-3) and two common folders (4a and 4b):

1. **"Simulator"** folder: includes your Simulator class as well as other helper classes required by the simulator project. The executable name shall be: `simulator_<submitter_id>` (for example: `simulator_098765432`)
2. **"Navigation"** folder: in order to allow us to load different Navigation ".so" files together, the name of the .so files and the class inside must be unique, please use the id of one of the submitters, as follows: `Navigation_098765432.so` (for the Navigation algorithm class, id of submitter is 098765432) and similarly for the class name: `class Navigation_098765432`.
3. **"GIS"** folder: in order to allow load of different GIS ".so" files together, the name of the .so files and the class inside shall be unique, please use the id of one of the submitters, as follows: `GIS_098765432.so` (for the GIS class, id of submitter is 098765432) and similarly for the class name: `class GIS_098765432`.
4. "Common" and "UserCommon" (there are no makefiles in these folders!):
   a. **"Common"** folder: shall include the common files required by more than one project, published by the Advanced Topics in Programing staff as the files that shall be used "as is" without any change.
   b. **"UserCommon"** folder: shall include your own common files, files that are required by more than one project. Your makefiles and includes may use those files where needed to avoid duplication.

The Simulator

The Simulator is able to run several Navigation algorithms (not only 2) on several GIS. Both the Navigation algorithms and the GIS are loaded dynamically as .so files, as explained below. The Simulator loads the same map file into all GIS instances and runs several given navigation requests on all combinations of <GIS>-<Navigation algorithm>.
Simulator reports:

1. The expectations are that the *resulting routes across all GIS instances*, per a given navigation request and Navigation algorithm, would all be the same. All cases where this is not true (not all results are the same), lines which are not in consensus* (see below) should be logged into “strange_GIS_results.log” with the following line:

```
Navigation name, shortests_<time/distance>, GIS name, {from}-{to}, calc distance (meters), calc time (minutes), valid
```

example:

```
Navigation_038754123, shortests_time, GIS_098765432, {40.7298, -73.9973}-{40.7295, -73.9967}, 1230.12, 2.01, 1
Navigation_038754123, shortests_time, GIS_038754123, {40.7298, -73.9973}-{40.7295, -73.9967}, 1230.15, 1.98, 0
 ...
```

2. The score calculation would take into account only routes which got consensus* (>50%, i.e. more than 50% of GIS instances agree on best distance route and more than 50% agree on best time route, not necessarily the same 50%). Routes without consensus for any of the Navigation algorithms (i.e. any cell) would be excluded for the entire column in the table, i.e. this navigation request wouldn’t appear at all. See also: GIS Consensus.
   The scoring table shall be calculated according to the following rules, accumulatively: Algorithms which got an invalid route get -1 to their score (getting invalid route for both distance and time results with -2). Algorithms which didn’t get routes when there is a route also gets -2. All algorithms which got the best minimal time route get +1. All algorithms which got the best minimal distance route get +1. Those which were qualified for best route (minimal time / distance or both), if also using minimal GIS requests (out of those which got the best route either for best time or best distance or both), get additional +1 to their score.
   Based on the calculated scores, create a .csv file with the name “simulation.results” with the following structure, including headline (replace with actual data):

```
Navigation Algorithm, <req1>, <req2>, <req3>, Total Score
Navigation_098765432, 3, 3, 1, 7
Navigation_032765555, 1, 3, 2, 6
Navigation_067765999, -1, -2, 1, -2
```

   The csv file shall be sorted according to the Total Score, from higher to lower.

Command line parameters

Running the Simulator shall support the following command line options:

```
simulator [-num_threads <num>] [-navigation <path>] [-gis <path>]
[-map_file <file_path>] [-navigation_requests <file_path>] [-output <path>]
```

Details:

1. The order of arguments may change
2. The `-num_threads` parameter dictates the number of parallel threads for running the simulation. **This parameter is optional**, in case it is not provided the simulator would not spawn any new threads (as if `-num_threads` is set to 1). See more under: Threading model.
3. The `-navigation` sets the location to look for the Navigation algorithm .so files.
   This parameter is mandatory and shall be an absolute path
4. The `-gis` sets the location to look for the GIS .so files.
   This parameter is mandatory and shall be an absolute path
5. The `-map_file` sets the map file to load.
   This parameter is mandatory and shall include the absolute path and filename
6. The `-navigation_requests` sets the file name to use for navigation requests.
   This parameter is mandatory and shall include the absolute path and filename
7. The `-output` sets the location for the simulator output files.
   This parameter is optional, in case it is not provided the simulator would create the output files under the run directory / current working directory.

The navigation_requests file

This will be a text file with a line per navigation_request in the following format:

```
{40.729824, -73.997302}-{40.729541, -73.996723}
{41.729824, -76.997302}-{41.729541, -76.996723}
```

New Abstract Base Classes

To allow common implementation you need to use common abstract classes. The actual header files would be published for use *as-is* and may/shall be in use by any of the three projects: *GIS*, *Navigation* and *Simulator*. Actual implementation of the abstract classes would be created only in a single proper project (which may be different for each abstract class).

AbstractNavigation - abstract base class

To allow common implementation you need to use the following abstract class:

```
class AbstractNavigation {
public:
  virtual ~AbstractNavigation() {}
  virtual std::unique_ptr<AbstractRoutes>
        getRoutes(const Coordinates& start, const Coordinates& end) const = 0;
};
```

Your Navigation class must inherit from this exact base class, enabling any simulator to run your navigation algorithm, even if the simulator wasn't created by you.

```
class AbstractRoute {
public:
  virtual ~AbstractRoute() {}
  virtual const std::vector<std::pair<EntityId, Direction>>& getWays() const = 0;
  virtual const Coordinates& getWayStartPoint() const = 0;
  virtual const Coordinates& getWayEndPoint() const = 0;
  virtual Meters totalLength() const = 0;
  virtual Minutes estimatedDuration() const = 0;
};

class AbstractRoutes {
public:
  virtual ~AbstractRoutes() {}
  virtual bool isValid() const = 0;
  // following functions can be called only if isValid is true - undefined otherwise
  virtual const AbstractRoute& shortestDistance() const = 0;
  virtual const AbstractRoute& shortestTime() const = 0;
  // following function can be called only if isValid is false - undefined otherwise
  virtual const std::string& getErrorMessage() const = 0;
};

class AbstractWay {
public:
  virtual ~AbstractWay() {}
  virtual std::pair<EntityId, EntityId> getJunctions() const = 0; // from, to
  virtual Meters getLength() const = 0;
  virtual bool isBidirectional() const = 0;
  virtual int getSpeedLimit() const = 0; // as Km/h
  virtual bool isHighway() const = 0;
  virtual bool isToll() const = 0;
  virtual const Coordinates& getFromJunctionCoordinates() const = 0;
  virtual const Coordinates& getToJunctionCoordinates() const = 0;
  virtual std::pair<Meters, Meters> // = {from-junction->point, point->to-junction}
        getSegmentPartsOnWay(std::size_t segment, const Coordinates& c) const = 0;
};
```

AbstractGIS - an abstract base class for your GIS

To allow common implementation you need to define the following abstract classes:

```
class AbstractGIS {
public:
  virtual ~AbstractGIS() {}
  virtual std::vector<EntityId> loadMapFile(const std::string& filename) = 0;
  virtual std::vector<EntityId> getWaysByJunction(const EntityId& junctionId) const = 0;
  virtual std::tuple<Coordinates, EntityId, std::size_t>
        getWayClosestPoint(const Coordinates& coords) const = 0;
  virtual std::tuple<Coordinates, EntityId, std::size_t>
        getWayClosestPoint(const Coordinates& coords, const Restrictions& res) const = 0;
  virtual const AbstractWay& getWay(const EntityId& wayId) const = 0;
};
```

Automatic Registration

To allow simple discovery of GIS instances and Navigation algorithms we need to define a common registration process so all loaded classes would register themselves automatically.

Note: we will explain in one of the following class meetings (with Amir or with Adam) the way the automatic registration should work. It may seem complicated but it is not so much.

You should use the following header files (the exact header file would be published and you would have to use it as-is) for auto registration:

```
// Common/GISRegistration.h
class GISRegistration {
public:
  GISRegistration(std::function<std::unique_ptr<AbstractGIS>()>);
};

#define REGISTER_GIS(class_name) \
GISRegistration register_me_##class_name \
        ([]{return std::make_unique<class_name>();} );
```

```
// Common/NavigationRegistration.h
class NavigationRegistration {
public:
  NavigationRegistration(std::function<std::unique_ptr<AbstractNavigation>()>);
};

#define REGISTER_NAVIGATION(class_name) \
NavigationRegistration register_me_##class_name \
        ([](const NavigationGIS& navigation_gis) \
        {return std::make_unique<class_name>(navigation_gis);} );
```

You are free to implement the .cpp file for the above auto registration classes as you wish but **you are not allowed to change the header files**!

Your GIS and Navigation .cpp files will have the following line in their global scope:

```
REGISTER_GIS(<class_name>)  e.g.:
REGISTER_GIS(GIS_098765432)
```

And similarly:

```
REGISTER_NAVIGATION(<class_name>)  e.g.:
REGISTER_NAVIGATION(Navigation_098765432)
```

The registered classes are unaware of the actual implementation of the registration process. Note that the easiest way to implement the registration is to make the Simulation class a Singleton.

Note: the registration headers are in use by both the Simulator project and the Navigation/GIS projects. However, the .cpp files of the registration **should be part of the Simulator project only** and their implementation is on you.

Error Handling

The Navigation project shall not create any error logs.
The GIS project may create error logs.
The Simulator project shall create the described log above for strange behaving GIS.
There is no need for the Simulator to handle a crash scenario of GIS / Navigation, however in any other case the Simulator shall not crash.

Bonus

A bonus in this exercise would be given for the best navigation algorithms.

Fixing failing tests from Ex1 and Ex2

In the check of Ex3 we would re-run failed tests of Ex1 and Ex2 to allow getting back points for fixing your code.

Parts of the exercise

Tests

We would provide a few simple google tests that present how the Simulator can be tested. You are not required in this submission to add your own tests. However you are encouraged to test your GIS and Navigation projects against Simulator of other teams and vice versa.

The exercise would be checked and tested on the same environment as ex1 and ex2.

Submission

You shall submit a zip named ex3_<student1_id>_<student2_id>.zip that contains:

- ***5 folders*** - as described above
- ***3 makefiles*** - each one inside the folder of the project that it builds
- ***students.txt*** - a text file that includes one line per submitter: <user_name> <id> (do not include the character ‘<’ nor ‘>’ - directly in the zip without any folder
- ***README.md*** - info and remarks about your implementation - directly in the zip without any folder

DO NOT SUBMIT THE FOLLOWING FILES

- binary files
- external libraries (you may only use standard C++ libraries and rapidjson)

Additional Notes and Requirements

1. **Threading model**
   - Note that the exact number of threads may be lower than requested in the command line, in case there is no way to properly utilize the required number of threads.
   - The exact threading model is your decision, try to make the most out of the provided threads.

- Note that it is totally OK that your main thread will be waiting for all other threads in a join call (or any other similar blocking wait). In that case, if the number of requested threads in the command line is 2, the proper handling would be to treat it as a request for a single thread (there is no reason to spawn a single worker thread and let the main thread to wait for it).
- Note that the support for a single thread run was declared **optional**.
- It is better to use the same GIS (after it loaded a map) for all Navigations rather than unload it and load it again or create the GIS and load the map again.
- It is better NOT to create too many GIS instances simultaneously if you are not going to use them concurrently. But if you find it better to load all GIS and use them it might be a proper solution!
- It is better not to lock if you can avoid locking. But if you need to lock, you should of course lock.
- The results table is known in advance and can be "created" ahead (there is no need for a sparse matrix for it).
- The analysis of "strange GIS" that generated different routes for the same navigation request and with the same navigation algorithm, requires collecting the data for all GIS instances. This can be done using an in-memory map of [Routes object=>a list of GIS instances that generated the exact same route]. Don't be too troubled about memory constraints for holding intermediate results during the run.
- Creating a navigation algorithm should be cheap. To ensure that navigation algorithms do not cache data between runs you should recreate the navigation algorithm per each new pair of <GIS>-<navigation request>. Note that recreating doesn't mean reloading the .so file.

2. Additions / Changes to the API from Ex2

(a)

NavigtionGIS would change the signature of the function `getWayClosestPoint` to the below signature (you may of course change also your relevant GIS function or add a new one):

```
std::tuple<Coordinates, EntityId, std::size_t>
getWayClosestPoint(const Coordinates& coords, const Restrictions& res) const;
```

The change is in the return type - adding a third value to the return type: `std::size_t` - the segment index on the way where the closest point sits, zero based (zero being the segment from the from-junction to the first curve point).

Note that the new signature appears also in AbstractGIS.

(b)

Way would have a new function `getSegmentPartsOnWay` to allow getting the distance to a certain coordinate on a way, from both ends of the way:

```
virtual std::pair<Meters, Meters> // = {from-junction->point, point->to-junction}
getSegmentPartsOnWay(std::size_t segment, const Coordinates& c) const = 0;
```

Note that the new signature appears also in AbstractWay.

3. Dependencies

You should use the dependencies provided by the course staff from a git repository - GIS-ex3-skeleton. This includes the files mentioned in this document as well as other dependencies that you are already familiar with, from previous exercises.

4. Additions / Changes after Exercise Published

- The structure of "strange_GIS_results.log" has been changed:
    - adding a field for the Navigation algorithm name

    - adding a field identifying if this result was calculated for shortest_time or shortest_distance
    - adding two fields for the calculated distance and time for this strange result
    - removing the field that counted number of instances with same result

- The definition of “Consensus” is made more clear:
    - When a Route is calculated by Navigation algorithm, per Navigation request, the route is checked by the simulator, if it is valid then the data for consensus checking can be narrowed into the Route **distance** and **time** (different valid routes which result with the exact same distance and time, up to the accuracy definition of Meters and Minutes, are considered a legitimate result, and in the same consensus group).
    - If there are results for Navigation algorithm and Navigation request, for shortest time OR shortest distance which are not the same (i.e. different **distance** or **time**, or both) when using different GIS instances, then we fall into a consensus check.
    - If there are more than 50% of results across the different GIS instances used, the group of this majority is the consensus group and we have a consensus.
    - Non consensus and results which are out of consensus:
        - If there is NO consensus for either shortest time OR shortest distance OR for BOTH: The entire **column** for this Navigation request would not appear in the scores table. Note that if only one of shortest distance / shortest time has consensus and the other doesn’t have, cell is still **invalid** and the entire column of this Navigation request would not be in the table.
        - In any case, if there are results that are out of the consensus group (including cases where there is no consensus group), all such results, shall be logged into strange_GIS_results.log
        - If there is a consensus for BOTH shortest time AND shortest distance requests - the cell is valid and may have a score. BUT - in order for actually having a score, all cells in that column should have their own consensus (which is unrelated to each other).

- A new column added to strange_GIS_results.log: “valid” - valid results (those which passed the NavigationValidator successfully) would have the value 1. Invalid results would have the value 0. Note that we consider results that say there is no route as invalid as we assume all Navigation Requests do have a valid route. So there can be two reasons for a result to be invalid: (1) a result saying that there is no route, in which case the values for “calc distance (meters)” and “calc time (minutes)” columns would be 0 (zero), as well as for the “valid” column. This result would be logged twice, both for “shortest time” row and for “shortest distance” row. (2) a result that didn’t pass NavigationValidator, in which case the values for “calc distance (meters)” and “calc time (minutes)” columns would be the actual results returned, while the “valid” column would be 0 (zero).

- No need to handle a single thread run: we hereby declare that the need to support a single thread run is **optional**. You are free to handle cases where requested number of threads is *one or two* in one of the following ways, as you choose:
    - print a usage message saying that you support only 3 threads and above (i.e. at least main thread + 2 worker threads) and exit.

- ignore the requested number of threads (*one or two*) and decide internally in your code that the minimal number of threads is 3 (i.e. at least main thread + 2 worker threads). You may write something to log, or not. Note that this behavior implicitly changes the default value of the command line parameter `-num_threads` to 3 (in a way) or alternatively the command line parameter passed into the program would still be 1, then would be changed to 3 somewhere inside the threading model logic.
- You may decide to support single thread run, if you wish - but this is not mandatory and there is no bonus for this feature.

Good Luck!