CMPT_431_Project

This repository contains C++ implementations for the All-Pairs Shortest Path problem using the Floyd-Warshall Algorithm. Include is a graph generator, a serial implementation, parallel implementation and distributed implementation of the Flyod-Warshall Algorithm.

Dependencies

- C++14 or above
- make
- MPI
- Python 3.X (to run the tests)

Project Structure

- `/core` is a set of header files used across the project. These files were taken from previous CMPT 431 assignments.
- `/inputs` is a directory for graph files generated by `input_generator`
- `/lib` is a set of header files containing functions for the three different implementations of the Floyd-Warshall Algorithm. Additionally, the Edge class and Matrix class is also located here.
- `tests` contains a Python script called `tests.py`. This file will run unit tests and validation tests for the serial and parallel implementations.
- `all_pairs_distributed.cpp` is the entry point to the distributed version of the Floyd-Warshall algorithm utilizing MPI.
- `all_pairs_parallel.cpp` is the entry point to the parallel version of the Floyd-Warshall algorithm utilizing C++ threads
- `all_pairs_serial.cpp` is the entry point to the serial version of the Floyd-Warshall algorithm.
- `input_generator.cpp` generates input files to be used with the above programs
- `Makefile` is a helpful tool to compile the above programs

PROF

Input Generator Usage:

`input_generator.cpp` is used to generate a graph in the format used by the other programs.
It takes the following arguments:

- `--nNodes`: Takes an integer. Sets the number of nodes to be used in the graph. Defaults to 100. Cannot be 1 or less.
- `--nEdges`: Takes an integer. Sets the number of outbound edges each node should have. Defaults to 5. Cannot be 0 or less.
- `--randEdges`: Takes no parameter, just using this flag enables this mode. Randomizes the number of edges out of each node. When enabled, the number of edges on a node ranges from 0 to 2*nEdges, with a normal distribution centered on nEdges.
- `--minWeight`: Takes an integer. Sets the minimum weight an edge could have. Defaults to 0. Cannot be negative, or greater than maxWeight.

- `--maxWeight`: Takes an integer. Sets the maximum weight an edge could have. Cannot be negative, or less than minWeight. Defaults to 10. Edge weights are determined with a linear distribution.
- `--fileName`: Takes a string. Sets a custom file name for the output graph to use. Defaults to "graph.txt". **WARNING: If the file already exists, this may overwrite its contents!**

Example usage:

```
--nNodes 10 --nEdges 5 --minWeight 0 --maxWeight 20 --randEdges
```

This will generate a graph with 10 nodes, each of which has a random number of edges (with an average value of 5). Each edge will have a weight between 0 and 20. Because no file name was given, the output will be in a file named `graph.txt`.

Running The Project

Compiling Files

To compile all files:

```
>>> make all
```

To compile a specific C++ file:

```
>>> make input_generator
>>> make all_pairs_serial
>>> make all_pairs_parallel
>>> make all_pairs_distributed
```

Executing Files

PROF

Sample input_generator. See above for more details:

```
>>> ./input_generator --nNodes 10 --nEdges 5 --minWeight 0 --maxWeight
20 --randEdges
```

Sample all_pairs_distributed. see the slurm tutorial from more information:

```
>>> ./all_pairs_distributed --inputFile ./inputs/graph.txt
```

Sample all_pairs_parallel:

```
>>> ./all_pairs_parallel --inputFile ./inputs/graph.txt --nThreads 4
```

Sample all_pairs_serial:

```
>>> ./all_pairs_serial--inputFile ./inputs/graph.txt
```

Testing

To run the tests:

```
>>> make tests
```

Clean Up

To remove all executables for the project structure

```
>>> make clean
>>> make clean_windows
```