Operating Systems

Final Project

Running The Project

instructions

To compile the project, open the terminal and execute the command make. The resulting executable files are Graph, server, and client. Open an additional terminal window to run the clients (one window for the server and one for each client).
To start the server, use the command: ./server
To initiate a client, enter the command: ./client localhost

Commands

Adding graph

To begin using the program, the client must add a graph. Pictures of four example graphs are available in the "Graph_Examples" directory.
(the graph examples were sourced from: https://visualgo.net/en/mst?slide=1).
The command structure for ADD_GRAPH is as follows:

<ADD_GRAPH> <graph ID> <number of vertices> <edge from v> <edge to u> <weight>

Note: Although the command specifies <edge from> and <edge to>, the edge is added in the reverse direction, resulting in an undirected graph.

The graphs located in the "Graph_Examples" directory are as follows:

Enter ADD_GRAPH 1 5 0 1 2 0 3 6 1 3 8 1 4 5 1 2 3 2 4 7 3 4 9
it will add the graph with the name id=1.

Enter ADD_GRAPH 2 9 0 1 8 0 2 12 1 2 13 1 3 25 1 4 9 2 3 14 2 6 21 3 4 20 3 5 8 3 6 12 3 7 12 3 8 16 4 5 19 5 7 11 6 8 11 7 8 9
it will add the graph with the name id=2.

Enter ADD_GRAPH 3 6 0 1 5 0 2 3 0 3 2 0 4 4 0 5 6
it will add the graph with the name id=3.

Enter ADD_GRAPH 4 5 0 1 28 0 2 13 0 3 13 0 4 22 1 2 27 1 3 13 1 4 13 2 3 19 2 4 14 3 4 19
it will add the graph with the name id=4.

Editing graph

To add a new edge to the graph, use the command structure:
<ADD_EDGE> <graph ID> <edge from v> <edge to u> <weight>
A new edge is added to the graph.
example: ADD_EDGE 3 1 2 6
Note: This command adds an undirected edge to the graph.
To remove an edge, input:
<REMOVE_EDGE> <graph ID> <edge from v> <edge to u> <weight>
The edge is removed.
example: REMOVE_EDGE 3 1 2 6
Important: This command only removes the edge; consequently, a vertex may become disconnected, which may result in a weight of -inf if an attempt is made to calculate a spanning tree.

Algorithms

To compute the minimal spanning tree of the graph, use the following command structure:
<SOLVE_MST><_design pattern> <graph id> <Algorithm type>
To execute with leader-follower pattern: SOLVE_MST_LF
To execute with pipeline pattern: SOLVE_MST_PIPELINE
example: SOLVE_MST_PIPELINE 3 PRIM , SOLVE_MST_LF 3 TARJAN , SOLVE_MST_LF 1 BORUVKA , SOLVE_MST_PIPELINE 2 KRUSKAL

Classes

Directories

Graph_DIR

Classes related to graph objects and algorithm logic include:
Graph_Interface.cpp: This program facilitates communication with the server via a pipe.
Graph_Builder.cpp: Responsible for creating a graph object.
MST.cpp: Handles additional calculations for the minimal spanning tree.
MSTAlgorithm.cpp: Implements various algorithms on the graph.
MSTFactory.cpp: Responsible for creating instances of MST algorithms.

Server_DIR

Patterns.cpp: Contains the Active Object and Leader Follower design patterns.
MSTServer.cpp: Manages client requests to perform operations on a graph.
Server.cpp: Implements the server's API

Outside any directory

pollclient.cpp: Beej poll client
Makefile

Valgring_Reports

Contains the screenshots of the required reports (Memcheck, Callgrind, Helgrind) with the appropriate file name.

gcov_Reports

Contains the gcov reports of each class. The test run is in the screenshot.

The project also available in Github:

https://github.com/LizaChepurko/Operating_Systems_FinalProject.git