Design Document

Overview

For this project, I designed 3 classes, a struct, and used a previously designed error class:

MSTInterface

MSTInterface is a wrapper class that contains an MSTCaculator class. It parses all inputs, prints outputs, and throws errors for the MSTCalculator class.

MSTCalculator

MSTCalculator is a class that allows the user to construct a graph, and calculate the minimum spanning tree using the graph.

MinPriorityQueue

MinPriorityQueue is a class that uses a min heap to store an std::size_t key and a double weight. It allows the user to store keys and extract the minimum weighted key in lgN time.

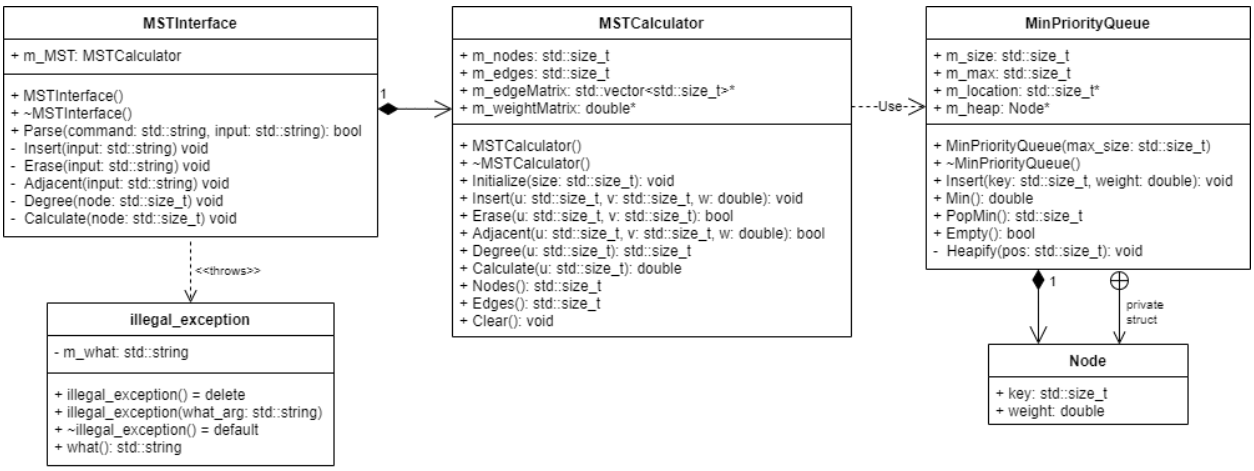
Node

Node is a struct containing an std::size_t key and a double weight.

Illegal_exception

illegal_exception is an exception class thrown by MSTInterface when there are illegal arguments in the input.

UML Class Diagram



Design Decisions

MSTInterface

This class uses the default constructor and destructor since no dynamic allocation is needed. No operator overloads were necessary.

bool Parse(std::string command std::string input)

void Insert(std::string input) void Erase(std::string input)

No const keywords are used for these functions because using a value parameter as input allows std::move(), and they all modify members.

void Adjacent(std::string input) const void Degree(std::size_t node) const

Adjacent() uses a value parameter to allow std::move(), Degree uses a value parameter because std::size_t is a small constant sized class, and they do not modify members, so a const is added.

void Calculate(std::size_t node)

This function uses a value parameter because std::size_t is a small constant sized class, and this class may modify members so no const is added.

MSTCalculator

This class initializes member variables in the constructor and deallocates members in the destructor. No operator overloads were necessary.

void Initialize(std::size_t size) void Insert(std::size_t u, std::size_t v, double w)

bool Erase(std::size_t u, std::size_t v) double Calculate(std::size_t u)

No const keywords are used for these functions because std::size_t and double are both small constant sized classes, and they all modify members.

bool adjacent(std::size_t u, std::size_t v, double w) const std::size_t Nodes() const

std::size_t Degree(std::size_t u) const std::size_t Edges() const

No const keywords are used for these functions because std::size_t and double are both small constant sized classes. None of them modify members, so a const is added.

void Clear()

This function has no parameters and modifies members, const is not used.

MinPriorityQueue

This class initializes member variables in the constructor and deallocates members in the destructor. No operator overloads were necessary.

void Insert(std::size_t key, double weight) std::size_t PopMin() void Heapify(std::size_t pos)

No const keywords are used for these functions because std::size_t and double are both small constant sized classes, and they all modify members.

double Min() const bool Empty() const

These functions do not have parameters, and do not modify members, so a const is added.

Node

This is a simple struct with a default constructor/destructor and no other methods.

illegal_exception

For this class a constructor is created with an std::string parameter containing the message. The default destructor is used since no dynamic allocation was used. No operator overloads were necessary.

const std::string& what() const

This function returns the error message, the error message should never be modified, so a const reference is returned. The function will not modify the error message, so const is added.

Test Cases

- There is not a requirement to handle invalid commands, so all invalid commands should be ignored.
- The parameters used for the ***i***, ***e***, ***adjacent***, ***degree***, and ***calculate*** functions should be checked and throw an illegal_exception if the input is not valid. The graph should not be modified if the exception is thrown.
- The number of edges should increase by 2 with each insert because the edges are undirected.
- When ***i*** and ***e*** are called, the edge should be inserted/erased in both directions.
- Decimal numbers should be printed to 2 digits.
- ***clear*** should remove all edges, but should not change the number of vertices.
- ***mst*** should print not connected if the graph is not connected, and should print the same weight no matter which node is the root node.

Performance Considerations

Let E = the number of edges in the graph, and V = the number of vertices in the graph.

degree, edge_count

These two commands have a runtime of O(1). In my implementation, these two commands simply return a stored value, which is constant time.

mst

This command has a runtime of O(E lgV). In my implementation, this command does a outer loop that runs V times, once for each vertex in the graph, and a inner loop only runs for each edge that V has. Thus, this command effectively runs the inner loop E times. In each inner loop, the weight of the edge could be added in the MinPriorityQueue, which has a runtime of O(lgV) because the height of the min heap would be at most lgV. In conclusion, the total runtime of the ***mst*** command is O(E lgV).

other commands

There are no runtime requirement for other commands.