Weather Station Analyzer

Version 1.0
April 14th 2016
Group 25

Pedro Oliveira
Pavithran Pathmarajah
Pareek Ravi
Kunal Shah

SFWRENG / COMPSCI 2XB3
Software Engineering Practice and Experience: Binding Theory to Practice
Computing and Software Department at McMaster University

Executive Summary

Weather Station Analyzer is an application used to analyze weather station information and determine which stations can predict the data of others.

The software will take in 5 input parameters: start year, sample period, temperature and precipitation tolerance, and accuracy range. Using theses inputs the program will search, sort and graph the requested stations, then identify which weather stations are redundant and may be removed while keeping weather readings accurate.

Internal Review

Results of the project where less than ideal. More stations were expected to be predictable. If there are future attempts to implement this project there are a few things we would do differently.

We would implement the database using MySQL instead of SQLite3 for the extra speed gained for large data sets. This would also allow for more scalability as it could be used on a external server rather than a local server

We would implement a Streaming API instead of a REST API, so that the large data set would not be transmitted all at once.

We would further decouple the frontend from the backend and decouple the frontend from the electron wrapper itself.

Table of Contents

Revision

For Revision History see Project Log

Pedro Oliveira	001430273	Team Lead
Pavithran Pathmarajah	001410729	Scrum Master
Pareek Ravi	001407109	Senior Developer
Kunal Shah	001419350	Software Developer

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Contributions

Name	Role	Contributions	Comment
Pedro Oliveira	Team Lead	Weather ADT	
Pedro Oliveira	Team Lead	Station ADT	Includes Location ADT
Pedro Oliveira	Team Lead	Graphing	Algorithm and Implementation
Pedro Oliveira	Team Lead	Edge Crossing	Used in graphing
Pedro Oliveira	Team Lead	MST	Used in graphing
Pavithran Pathmarajah	Scrum Master	Edge ADT	
Pavithran Pathmarajah	Scrum Master	MinPQEdges ADT	Used in graphing
Pavithran Pathmarajah	Scrum Master	App UI	HTML/CSS
Pavithran Pathmarajah	Scrum Master	Future Testing	
Pavithran Pathmarajah	Scrum Master	FrontEnd Logic	
Pavithran Pathmarajah	Scrum Master	Interface with server	App POST requests
Pavithran Pathmarajah	Scrum Master	Output results to JSON	
Pareek Ravi	Senior Developer	Parsed CSV files	
Pareek Ravi	Senior Developer	Created SQLite3 DB	
Pareek Ravi	Senior Developer	DB Interface	
Pareek Ravi	Senior Developer	HeapSort	
Pareek Ravi	Senior Developer	DB Filtering	
Kunal Shah	Developer	Rest API	POST commands
Kunal Shah	Developer	Rest Server	
Kunal Shah	Developer	Shoes Prototype UI	
Kunal Shah	Developer	UI Loading bar	
Kunal Shah	Developer	Prototype Filtering	Later converted to SQL

Decomposition

View

Index.html

Description

Main view of the program. This is what gets wrapped with electron. There are three parts to this view. Input, message, and return map.
Input:

Input will take in the user input for the 5 user parameters and when the user hits the submit button. The frontend will pass the inputted values to the server using a JSON string.

Message:

Message will show update messages and show constantly updating progress bar to the user. This will be updated through the javascripts

Return Map:

Once run.rb is finished executing the javascript will generate a map with all the points and it will be shown to the user.

Main.js

<u>Description</u>

Javascript logic for the frontend view. Updating progress messages and bar as well as generating the d3 map to show all the valid and invalid points to the user.

<u>Implementation</u>

initialize(x):
Sets up initial values for the d3 map

increaseProgressBar(float num):
Takes percentage of progress as input. Sets the coloured rectangle length in stylesheet main.css to desired length.

loadStuff:
When the user submits the input values, this method will store them in a JSON string
check(x):
if the user has not filled in any input field supplemented with 0 instead
load:
Loads electron dependencies and watches the loading sequence, depending on what stage the program is at, it will update the load bar position

drawMap:
Generates map of california and draws it to the screen.

checkAndload:
if query has already been submitted then force reload the electron wrapper, to allow the user to submit another query. Also disables all buttons/input text boxes while program is computing weather station validity

drawPoints:
Generates point locations based on Longitude and Latitude point conversion and draws it to the screen.

updatePeriod:
Depending on the selected start year limit the number of inputs available for sample period

Main.css

Description

The style sheet for index.html.

Model

BackEnd

Description

It is the majority of the Model

Interface

self.parse(String dataFile):

Parses the CSV file into Station objects

self.setData(Array data):

Assigns data to @fullStationList

self.getData:

Returns @fullStationList

self.createGrid:

Splits Stations into a searchable 2D Array, separated by latitude and longitude degrees

self.createEdges:

Creates edges for graph

self.trimEdges:

Trims superfluous edges for the graph

self.makeEdges(Station curr, Set<Station> nodes):

Makes edges between curr and each Station in nodes

self.adjacent(Station node, int x, int y):

Returns a Set<Station> of Stations near by node

self.checkRelated:

Returns a Hash[Station.code:String, Array of Stations it predicts including itself]

self.getEdges:

Returns the graph's edges

self.createMST:

Creates an MST

self.run:

Runs the backend

Implementation

No private classes implemented

Station

<u>Description</u>

An ADT designed to hold all the data for each weather station. That is: ID, Name, Elevation, Location, and Weather data

<u>Interface</u>

initialize(string code, string name, int elev, float lat, float lon):
    Constructs a Station object

location:
    Returns the Station's location

lat:
    Returns the Station's latitude

lon:
    Returns the Station's longitude

elevation:
    Returns the Station's elevation

code:
    Returns the Station's ID

name:
    Returns the Station's name

add_weather(Weather day):
    Adds a Weather day to the Station

weather:
    Returns the Station's weather data as an Array of Weather

to_s:
    Returns the Station as a string

==(Station other):
    Equality operator for two Stations. Returns true if self and other are equal

hash:
    Overrides the Station's hash function, used for Sets and Hashs

state:

Returns the Station's state

Implementation

No private functions implemented

Location

Description

An ADT to store a location's latitude and longitude

Interface

initialize(float latitude, float longitude):

Constructs a Location object

==(Location other):

Equality operator for two Locations. Returns true if self and other are equal

to_s:

Converts itself to a string and returns it

state:

Returns the Location's state

float latitude:

Location's latitude

float longitude:

Location's longitude

Implementation

No private functions implemented

Weather

Description

An ADT that contains the date, the precipitation, the maximum and minimum temperature

Interface

Initialize (String date, int precipitation, float t_max, float t_min)

Splits the date into its month, day and year. Initializes the precipitation and the temperatures

date

Return the date

temp

Return the range of the temperature

t_max

Return the maximum temperature

t_min

Return the minimum temperature

precipitation

Return the precipitation

to_s

Returns a string of the date, precipitation and the temperature range

== (weather other)

Returns a boolean of this is the same as other

Implementation

state

Returns the weather's state

Edge

<u>Description</u>

An ADT which is used to connect two stations together and used to make the Minimum spanning tree

<u>Interface</u>

setTolerances (int rain, int temp, int days)
    Initializes the variables into instance variables
initialize (Station node1, Station node2)
    Sets the two stations that it connects and the length between them
getLength (Weather[] val1, Weather[] val2)
    Returns the lengths of the shorter list
distanceCalc(Station s1, station s2)
    Returns the distance between the two stations based on the longitude and latitude
checkTempTolerence (Weather[] val1, Weather[] val2)
    Returns a boolean if the number of days that are in temperature tolerance + accuracy is greater than or equal the the length of the edge
checkRainTolerance(Weather[] val1, Weather[] val2)
    Returns a boolean if the number of days that are in precipitation tolerance + accuracy is greater than or equal the the length of the edge
withinTolerance(int val1, int val2, int tolerance)
    Returns whether the temperature/precipitation is within the tolerance given
is_related?
    Returns a boolean if the nodes on the edge can predict each other's weather
is_related (Station s1, Station s2)
    Returns a boolean if s1 and s2 can predict each other's weather
nodes
    Return the two stations that are on the edge
distance
    Return the length of the edge
>= (Edge other)
    Returns a boolean of if this length is greater than or equal to the other edge's length
cross (Edge other)
    Creates an equation of a line for both edges, and then determines if the two intersect within their longitude and latitude. It returns a boolean true if they both cross
reverse
    Returns a new edge that is in the reverse order

eql? (Edge other)

Returns if this edge is equal to other

hash

Returns the hash of s1 and s2

state

Returns Edge’s state

to_s

Returns a string of the latitude, longitude of the two stations

Implementation

No private functions implemented

Float

<u>Description</u>

An extension of float with an additional function

<u>Interface</u>

near? (float other, float epsilon = 1e-6)

Returns if the difference of the two float values is less than epsilon. They values are not equal, but are similar to a large degree of decimal places

<u>Implementation:</u>

No private functions implemented

SQL

Description

Class that contains all the functions that use the database. The creation and all queries are run through here

Interface

self.makeSQL

Creates an SQLite database based on the csv files. The csv files are read and accordingly placed into tables in the csv file. A text file keeps track of the files that have already been read such that the same file is not read again

self.getValid (int input, int period)

The function first converts the input (start year) to a value that matches the date format used as the Ids and then calculates the end year based on the period. A query is called from the database to access all station data that is not does not contain -9999 and the ids are between the two time periods. The dates for each station are then put into a hash map keyed by the station id

self.parse (int input, int period)

The function that is called by backend.rb. This function first calls getValid and then using accesses the dates that were in the hash maps and creates the station objects with their respective weather objects. The stations are added into a list and returned to the backend which then processes this data

self.daysBetween (int startYear, int period)

Calculates the days that are between the time period that accounts for leap years. This return the number of days in the years passed

Implementation

No private functions implemented

Run

<u>Description</u>

Main method for the backend. This method gets called from the rest server, and gets passed all program input variables. Once called run.rb will call all child methods from backend.rb module.

<u>Interface</u>

initialize(backend start_year, period, temp, percip, accuracy):

Initialize takes in the parsed information from the server and runs it through the algorithm while writing the output file for the front-end to read.

futureCheck
(String start_year,String period,Map<String Station_name, Set<String Station_name>> tmp):

This method is called to check our algorithms calculations against a two year future sample period, taking in the user's specified current start year and sample period. And then using the set of estimating stations and predicted stations given, creates an edge between the them, and if the predicting stations cannot predict the weather at the removes station with accuracy that station is denoted as a must keep. This is done to ensure that the stations selected are not a fluke.

writeOutputFile(String start_year,String period)

This method takes in the start year and the sample period and writes the output json file called goodbad.json. This calls the future check then writes it's output from file and then calls the sql parser to get the stations with insufficient data and parse them.

<u>Implementation</u>

No private functions implemented

MinPQEdges

Description

A priority Que specifically designed to handle edges and prioritizes the shortest edge first. Based on the minPQ algorithm from *“Algorthims 4th edition”.* Is called in the backend when forming the Minimum spanning tree and the initial graph.

Interface

Initialize():

Called on the creation of a new que, prepares the array for the priority que and sets it size to 0.

empty?:

No parameters and returns whether the priority que is empty or not. Does so by checking if the size is equal to 0.

Push (Edge e / Set<Edge> e):

Is called to add an edge to the que. Has a single parameter requiring either a single edge to add to the que or a set of edges to add to the que. This is done by pushing the edge(s) into the array adding them to the back of the que and then swiming them up to their spot.

swim(int ii):

Is called to move an edge to its position in the que. Does so by checking if the edge at position ‘i’ in the array is greater than the one above it, if so exchange them and move up one more level until it cannot move any-more.

sink(int i):

Is called to move an edge to its position in the que. Does so by checking if the below item in the que is greater than it if so sink it one more level. This is repeated until the edge at position ‘i’ in the array finds its spot.

greater(int i,int j):

Is used in the swim and sink functions mainly, determine if the edge at ‘i’ is greater than the edge at ‘j’ in the array.

min():

returns the front of the que

size():

Is used to retrieve the size of the que. Has no parameters and does so by calling the size operation on the array used.

exch(int i,int j):

Is used to swap two positions in the que. Does not return anything. Does the swap by using a temp variable to hold one of the values and then moves the array values around, once that is done places the tmep value in its new spot in the array

pop():

No parameters and returns the next edge in the que. Does so by returning the edge in the first array position of the que. To deal with the pop exchanges front and back of que, then removes the back of the que and sinks the new top of the que to restore it.

<u>Implementation</u>

No private methods implemented

Controller

Rest-server

<u>Description</u>

Sinatra rest server to receive GET requests from front end, and call backend with the passed json input.

<u>Interface:</u>

/query/:data'

Parameters : :data - JSON string

Returns: return_message - to console

Parse the passed json string (:data) and call the backend method in Run.rb. with the program input parameters from the JSON.

/terminate

No parameters

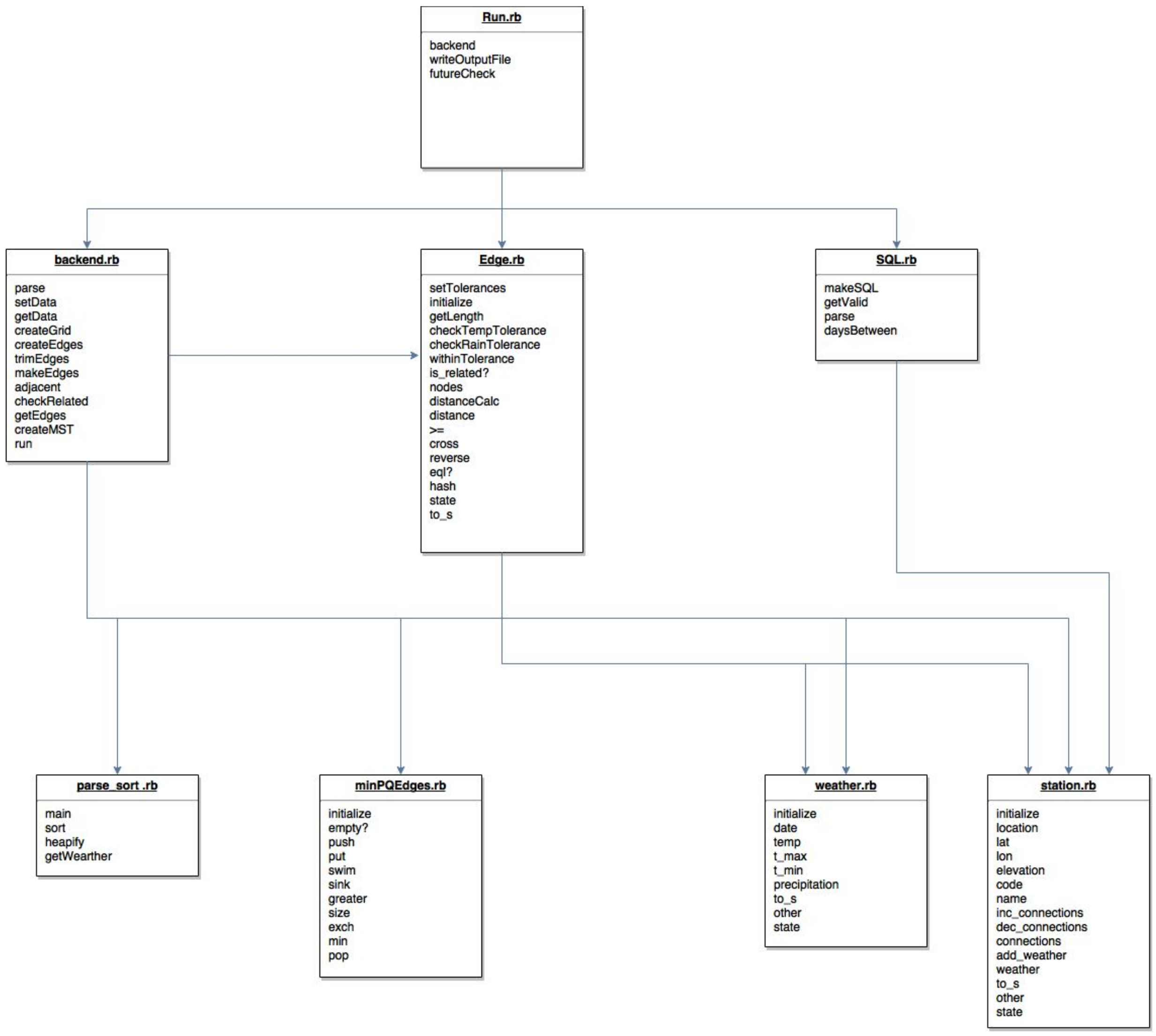
No return

When called, terminates the rest server.

<u>Implementation</u>

No private classes implemented

UML


Run.rb
backend
writeOutputFile
futureCheck
backend.rb
parse
setData
getData
createGrid
createEdges
trimEdges
makeEdges
adjacent
checkRelated
getEdges
createMST
run
Edge.rb
setTolerances
initialize
getLength
checkTempTolerance
checkRainTolerance
withinTolerance
is_related?
nodes
distanceCalc
distance
>=
cross
reverse
eql?
hash
state
to_s
SQL.rb
makeSQL
getValid
parse
daysBetween
parse_sort .rb
main
sort
heapify
getWearther
minPQEdges.rb
initialize
empty?
push
put
swim
sink
greater
size
exch
min
pop
weather.rb
initialize
date
temp
t_max
t_min
precipitation
to_s
other
state
station.rb
initialize
location
lat
lon
elevation
code
name
inc_connections
dec_connections
connections
add_weather
weather
to_s
other
state