OpenStreetMap Less Intersect Program

Team C: Bradley Cutshall, David Van Loon, Jordan Ross, Vamsidhar Kasireddy

1. Overview

This project operates on OpenStreetMap data. OpenStreetMap is an open source mapping project that avoids restrictions on use and legal implications. Our program takes a selected set of OpenStreetMap data (local or global) and outputs a user-defined number of the top lengths of roadways without an intersection. This may be useful in helping route planning for a trip, saving the user both time, and money on fuel costs. For example, if the user wants to know the Top 5 longest stretches of road in Duluth, MN, they can simply take the downloaded data for Duluth, MN, from the OpenStreetMap site, load it into the program, and specify that they want to know they top 5 lengths of roadway.

2. Solution Summary

OpenStreetMap overview:
The results are gathered through the execution of 3 MapReduce jobs on preprocessed input data extracted from OpenStreetMap. The entire process is controlled by the `Distance` class, which configures each job with all required parameters. This includes the command-line arguments, which are parsed in the `Distance` class. Each job is run in order, taking into account the interdependencies between them.

The following describes the task of each job in the process:

Pre-processing:

The source data acquired from OpenStreetMaps is formatted in XML, which ultimately means that the data cannot be processed in a line-by-line manner. This is because a single line could depend on one or more previous lines, depending on the tag structure of the document. Pre-processing is required to remove these dependencies from the data.

Job 1:

This job parses in the input data and identifies intersections.

Map:

The map step in Job 1 processes each line of input, extracting key data elements and passing them to the reduce step. Multiple types of input lines can be processed, so the `Mapper` detects the type of input line and extracts the relevant data. If the input line contains details about a node, the unique node identifier and the node location are passed to the reducer. If the input line contains information about a node's relationship to a way, or road, the unique

node identifier, road identifier, position of the node (relative to others in the road), and the name of the road are passed to the reducer.

Reduce:

The reducer step in Job 1 takes the output from the map and formats it so it can processed in a distance calculation in later jobs. In the raw data, the way information and the coordinates of the nodes within the way are separate. The `Reducer` matches the correct set of coordinates with the other information about the way. The output key consist of the Way ID and the index of the Node within that way. The output value will consist of the Node ID, Latitude, Longitude, Node index, Intersection Flag, and the Way name if it exists (not all of the ways have names). The intersection flag will simply be a '1' if the node is an intersection or a '0' if the node is not an intersection. This flag will be pivotal for the later distance calculation.

Job 2:

This job computes the total distance of each way segment in miles. This job can be divided into three major sub-parts: map, sort and reduce.

Map:

In job 2, the map function is designed to sort values incoming from the output of job 1. Mapping in this job is straightforward because nearly all of the work is done by the comparators and partitioner. The map of job 2 requires a pre-defined format which consists of a line of text that contains a two part composite value segment that is used as a two part composite key for the reducer, and a value. The map's primary objective is to partition the line of text into a key value pair.

Sort:

Most of the work required for a successful operation of the mapper in job 2 is completed during a sorting technique. After the key and value pairs are defined by the map, comparators are required to break up a two part composite key for successful partitioning. Two comparators are used to separate and compare the two part composite key which contains the way segment ID, and the order in which the associated values exist within the way segment. A comparator is used to partition values into reducers that have the same way segment IDs, another is used to ensure that the values arrive at the reducer sorted.

Reduce:

Now that the values have been sorted and partitioned into their respective reducers we can create a meaningful calculation from the values. For a specific way segment ID this reducer assumes that the values will be read into

the reducer in an ordered fashion. Ordering is required because this reducer takes the ordered values and creates a sum of the distance between the start and end of the way segment using these intermediate values. Within each value there exists a Latitude and Longitude point, iterating through these points we can compute an overall distance by calculating the distance between each intermediate point. The reducer takes an input's value coordinates, saves it, and obtains the next sorted coordinate point. At this point the reducer uses the Calculator class to calculate the distance between each point and adds this distance to the way segment sum. Intersection flags are used to control the start and stop of each value sum. Once a distance between two intersection points on a way ID are added together, the data is written to file with the way segment's name, starting and ending coordinates, and overall distance. At this point every way segment, regardless of distance, is written to file as input for the next job.

Job 3: Job sorts the values based on distance and displays the Top values among them

Map:

Mapper reads in the output from second reducer as values and splits them on tab space and space. It retrieves the distance and assigns it as key and value of map as the value to a tree map, once tree map size increases more than topN value it removes the key with minimum value. Each mapper will have its local topN distance values which are then send to reducer using clean up function.

Reduce:

Reducer reads distance of the stretch as value sorted in ascending order then finds the topN from the finalists sent by the mappers. Once this job completes we will be expecting data to be read in the following format:

```
primary 27.7845005 -8.0874668 30.9398849 -2.826074 410.6301074343289
primary 27.7499764 -8.1331811 30.9629344 -2.7785076 408.5866465064338
primary 27.8487387 -8.0326954 30.9641922 -2.7749952 393.1262132067393
```

3. Team Member Contributions

Vamsidhar Kasireddy

- Developed Job3 Mapper and Reducer
- Developed Job2 MapReduce in conjunction with Bradley

- Contributed in Main class

David Van Loon
- Designed base project architecture
- Designed & implemented data pre-processing
- Designed, implemented, and tested Job 1 mapper
- Designed & implemented `getdata.sh` script

Bradley Cutshall
- Contributed to the development of Distance main class
- Developed and tested Job 2 MapReduce, comparators and partitioner in conjunction with Vamsidhar
- Developed and tested the Calculator class

Jordan Ross
- Wrote the Job 1 Reducer
- Worked on the debugging of Job 2, and the distance calculator
- Contributed to the overall abstraction of the problem and the approach to solving it.
- Contributed to the documentation of code, and this README

4. Instructions

Pre-processing:

The `getdata.sh` script automates the data aquisition and pre-processing. Due to the size of the input data and the requirement that pre-processing be done in a linear manner, the script can take up to 24 hours to run. Since a normal interactive session on Itasca will quickly time out, the script needs to be run in an interactive session with a longer wall time.

To request a new interactive session, use the `isub` command:

```
isub -n nodes=1:ppn=4 -m 8GB -w 24:00:00
```

Once the new node is ready, run `getdata.sh`. Use the flag `-s` to download a small dataset for testing, or omit the flag to download the entire dataset. The full dataset could take between 12-18 hours before it is ready, while the small dataset should be ready within a few minutes.

Once complete, the file `input_data.osm` will be prepared for use in MapReduce. This file is an XML representation of OpenStreetMap data.

Example input data:

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="Osmosis 0.43.1">
<bounds minlon="-92.60300" minlat="46.57500" maxlon="-91.23400" maxlat="47.09500"
origin="http://www.openstreetmap.org/api/0.6"/>
<node id="19188464" version="12" timestamp="2014-04-06T12:42:38Z" uid="1811853"
user="olaf1000" changeset="21531160" lat="46.7729322" lon="-92.1251218">
<tag k="name" v="Duluth"/>
<tag k="is_in" v="Minnesota USA"/>
<tag k="place" v="city"/>
</node>
<node id="19193953" version="2" timestamp="2012-03-30T16:43:55Z" uid="10786" user="stucki1"
changeset="11157125" lat="46.858986" lon="-91.234218"/>
<way id="18245686" version="2" timestamp="2012-12-30T11:45:46Z" uid="451693" user="bot-mode"
changeset="14462509"> index=0 <nd ref="188154974"/> <tag k="name" v="Sunshine Lake Road"/>
<tag k="highway" v="residential"/>
<relation id="4709557" version="1" timestamp="2015-03-20T15:39:41Z" uid="22925"
user="ELadner" changeset="29617804">
<member type="way" ref="333867357" role="outer"/>
<member type="way" ref="333867355" role="inner"/>
<tag k="type" v="multipolygon"/>
<tag k="leisure" v="track"/>
</relation>
</osm>
```

Running the project:

To run the project, change the parameter `NUMBEROFSEGMENTS` in `generic.pbs` to the desired value. `NUMBEROFSEGMENTS` defines the number of the top road segments to output. Point `LOCALINPUT` at the desired input file, change `LOCALOUTPUT` to the desired output location, and execute `runit.sh` to submit the job.

Expected output:

Output will be presented in the following format:

Road Name	Starting Lat.	Starting Lon.	Ending Lat	Ending Lon	Distance
primary	27.784500	-8.0874668	30.939884	-2.826074	410.6301

4. Commentary

Since all of the data is provided from an open source repository, there is no guarantee that a downloaded data set will be clear of any erroneous information. Our program does not make any assumptions about the data being real and verified, so any information about roads provided by OpenStreetMap data is assumed to be valid.

As a result, some roads that appear high in the results are roads consisting of few points that span long distances. These roads do not actually exist and are artifacts of the OpenStreetMap open-source process.

Overall, the accuracy of the results depends on the accuracy of the input data.