Final Project

DS730

In the final project you will be working with all of the technologies you have learned in this course. The only constraint[1] for each of the problems is that you must solve them with one of the high performance tools learned in this class: Java threads, Hadoop MapReduce, Pig, Hive or Spark. **Note that a solution written without using a tool learned in this course will not be any points[2].** If you wish to solve them using some other tool, you must ask first. If you are unsure if the way you solved them meets the requirements of this project, ask first before submitting your solution.

You can download the files needed for the final project from here:
http://www.uwosh.edu/faculty_staff/krohne/ds730/final.tar.gz

If there is a tie for any of the questions, you should print out all of the correct answers.

1. (21 pts) For the first problem, you will be reading in from several weather almanac files. You are encouraged to look at the files to see what the data looks like. Once you have a good grasp on what is contained in the files, you are to answer the following questions. You should note that all questions are not necessarily best solved using only 1 tool. It's possible that part (a) is best solved with Pig whereas part (b) is best solved with Java.

   Some of these problems may have a subjective meaning. If I ask which day was the hottest, that could be interpreted in different ways. Do I want to know which day had the highest temperature or do I want to know which day was the hottest with respect to the average? For example, assume one day was 94 the entire day and another day was in the 70's for most of the day and then jumped to 95 for a short amount of time before falling back to the 70's. Which day would be considered *hotter*? I will try and explain exactly what I am looking for. If a question feels ambiguous, ask me before attempting it.

---

[1] The only exception is problem 2 must be solved with Java threads.

[2] A common mistake has been solving some of these in Java with no threading (or trivial threading by creating 1 thread to do all of the work). We did not learn basic Java in this course. We learned threading in Java, not just Java. A similar problem was writing a simple Python program to solve a problem with no MapReduce being used. We learned MapReduce and Hadoop with Python in this course, but not just Python. Ask if unsure.

You should be aware of bogus values in the input. This is almost always the case in real datasets and you will almost always have to clean your data before you analyze it. For instance, some of the values are set to -9999 if no value was recorded (or the cell is just empty). Only the bogus/missing cell should be removed from your calculations. However, the rest of the row should be included if a cell has a bogus/missing value. For example, in the Oshkosh weather file, on January 26th, 2008, the temperature was recorded as -9999 but there is a valid wind speed. The temperature should be ignored but the wind speed should be included in your answer. Make sure to look at the data before beginning so you can figure out what values are invalid. Ask me if you are not sure if a value is valid or not.

If you are using Hadoop, Pig, Hive or Spark, you must assume the data is stored in the HDFS folder of: **/user/maria_dev/final/Oshkosh/** for the Oshkosh csv file and /**user/maria_dev/final/IowaCity/** for the Iowa City csv file. If you are using the local filesystem to read in the files, you should not change the name of the files nor should you store them in a separate folder. For example, if you are writing a multithreaded Java program to solve one of the questions, the input will be in the same folder as the Java file.

You are to solve each of these problems and provide the answers to all of the questions in a file called **prob1output.txt**. For each of these problems, make sure that you create a separate script or program for each problem. Also be sure to create an appropriately named file containing your solution. Your solution should be stored in *prob1W.XYZ* where W is equal to the actual problem letter and XYZ is the appropriate extension for the tool you are using. For example, if you solve 1a with java, your program will be stored in prob1a.java. If you solve 1b with pig, your program will be stored in prob1b.pig.

Your code does not need to output the exact answer for each of the questions (a-f) in this problem. It is fine to do a *little bit* of extra manual work to find the answer. For example, for part (a), you do not need to output “cold is more common” or “hot is more common.” Rather, you could output the number of “cold” days along with the number of “hot” days and manually look at the output to determine the answer. However, the amount of manual work you do ought to be limited. An incorrect solution for number 1 would be to simply print out the entire weather file and then claim that you manually checked every row and counted the number of cold days and counted the number of hot days. If you are

unsure if your solution does too much manual work, just ask.

a. In Oshkosh, which is more common: days where the temperature was really cold (-10 or lower) at any point during the day or days where the temperature was hot (95 or higher) at any point during the day?

b. When I moved from Wisconsin to Iowa for school, the summers and winters seemed similar but the spring and autumn seemed much more tolerable. For this problem, we will be using meteorological seasons:
Winter - Dec, Jan, Feb
Spring - Mar, Apr, May
Summer - Jun, Jul, Aug
Fall - Sep, Oct, Nov
Compute the average temperature (sum all temperatures in the time period and divide by the number of readings) for each season for Oshkosh and Iowa City. What is the difference in average temperatures for each season for Oshkosh vs Iowa City?

c. For Oshkosh, what 7 day period was the hottest? By hottest I mean, the average temperature of all readings from 12:00am on day 1 to 11:59pm on day 7.

d. Solve this problem for Oshkosh only. For each day in the input file (e.g. February 1, 2004, May 11, 2010, January 29, 2007), determine the coldest time for that day. The coldest time for any given day is defined as the hour that has the coldest average. For example, a day may have had two readings during the 4am hour, one at 4:15am and one at 4:45am. The temperatures may have been 10.5 and 15.3. The average for 4am is 12.9. The 5am hour for that day may have had two readings at 5:14am and 5:35am and those readings were 11.3 and 11.5. The average for 5am is 11.4. 5am is thus considered colder. Once you have determined the coldest hour for each day, return the hour that has the most occurrences of the coldest average.

e. Which city had a time period of 24 hours or less that saw the largest temperature difference? Report the city, the temperature difference and the minimum amount of time it took to obtain that difference. Do not only consider whole days for this problem. The largest temperature difference may have been from 3pm on a Tuesday to 3pm on a Wednesday. The

largest temperature difference could have been from 11:07am on a Tuesday to 4:03am on a Wednesday. Or the largest difference could have been from 3:06pm on a Wednesday to 7:56pm on that same Wednesday. For a concrete example, consider Iowa City on January 1, 2000 at 2:53pm through January 2, 2000 at 2:53pm. The maximum temperature in that 24 hour span was 54 and the minimum temperature in that 24 hour span was 36. Therefore, in that 24 hour span, the largest temperature difference was 18 degrees. If this were the final answer, you would output “Iowa City”, “18 degrees” and January 2, 2000 3:53am to January 2, 2000 10:53am.

f. As a runner, I want to know when is the best time and place to run. For each month, provide the hour (e.g. 7am, 5pm, etc) and city that is the best time to run. The best time and place to run will be defined as the time where the temperature is as close to 50 as possible. For this problem, you are averaging all temperatures with the same city and same hour and checking how far that average is from 50 degrees. If there is a tie, a tiebreaker will be the least windy hour on average. If there is still a tie, both hours and cities are reported.

2. (14 pts) The solution to this problem must be done using Java threads. Due to budget cutbacks, the postal services at UW-Oshkosh can only afford 1 mail deliverer. Even worse, that deliverer is a student who works part-time. Postal services wants to minimize the amount of time that student has to work in order to save money. Because of this, they are interested in the fastest way to visit all buildings and return back to the Campus Services Building, BldgOne in the example below. There are obvious routes that are terrible (e.g. going from one side of campus to the other and then back) but the optimal route is not obvious. Your goal is to read in a file that gives the time in seconds to get from a building to every other building and determine the best possible route such that you start at the building listed on the first line, visit all other buildings and end at the building listed on the first line. The building names in the example below are arbitrary and can be called anything. The input file you will read in is called **input2.txt** and will be formatted in the following manner:

BldgOne : t(BldgOne) t(BldgTwo) t(BldgThree) t(BldgFour) t(BldgFive)
BldgTwo : t(BldgOne) t(BldgTwo) t(BldgThree) t(BldgFour) t(BldgFive)
BldgThree : t(BldgOne) t(BldgTwo) t(BldgThree) t(BldgFour) t(BldgFive)
BldgFour : t(BldgOne) t(BldgTwo) t(BldgThree) t(BldgFour) t(BldgFive)

BldgFive : t(BldgOne) t(BldgTwo) t(BldgThree) t(BldgFour) t(BldgFive)

Take the first line for example. t(BldgTwo) will be an integer value denoting the number of seconds it takes to get from BldgOne to BldgTwo. On the first line, t(BldgOne) will be 0. In other words, it takes 0 time to get from BldgOne to BldgOne. The input will always be formatted in this manner. If another building is constructed, it will be added to the end and the file will be updated accordingly. For example, if BldgSix were constructed, the time to BldgSix will be added at the end of every list and BldgSix will be added to the end of the file. The time from BldgOne to BldgThree may not be the same as the time from BldgThree to BldgOne. There may be one way streets; it may be uphill, etc. A sample input file is shown below:

BldgA : 0 5 7
BldgB : 4 0 3
BldgC : 6 4 0

Each row in the the sample input will start with the building ID, followed by a space, followed by a colon, followed by another space, then followed by a list of integer values to get to all of the other buildings where each value is separated by a space.

Your goal is this, for the best route possible, print out the total time taken to start with the building on the first line, visit all buildings and then return to the building on the first line. You must also output the order in which you visited the buildings using the name of the buildings as defined in the input file. In the above example, there are only 2 possible routes: BldgA -> BldgB -> BldgC -> BldgA which is 5 + 3 + 6 == 14. The other route is: BldgA -> BldgC -> BldgB -> BldgA which is 7 + 4 + 4 == 15. Therefore, the output would be:

14 BldgA BldgB BldgC

You should save this output to a file called **output2.txt**.

A few constraints on your solution are given below:

a. In order to get from BldgA to BldgC, the direct path will always be the best path. In other words, if going from BldgA to BldgC, then going directly from BldgA to BldgC will always be faster than to go from BldgA to any other

arbitrary building (say BldgB) to BldgC[3].

b. Your code must split up the computation is a reasonable fashion. Think about the solution space and how the potential solutions can be split up and checked in parallel.

c. Your code must run faster than my brute force non-parallelized way of solving it. My brute force solution that uses no multithreading and is not optimized checks all permutations of 12 buildings. It runs in roughly 60 seconds.

d. Because this problem blows up in complexity, your code only needs to work for up to 13 buildings. As a guideline, my threaded code with 13 buildings takes about 4 minutes to run.

3. (15 pts) The goal of the last problem is to answer questions about a large dataset that you are interested in. Amazon has many large datasets available at https://aws.amazon.com/public-datasets/. Here is another link to many large datasets:

   http://www.datasciencecentral.com/profiles/blogs/big-data-sets-available-for-free.

   You do not need to find all of your data from one place. You can combine data from multiple sources if need be. Find something that interests you. Once you have found a dataset that interests you, create at least 4 interesting questions about that dataset and answer them. The questions themselves are entirely up to you but they should be somewhat involved. Since the data might not be in a perfect format, you will likely have to clean and prepare your data before you can analyze it.
   For example, determining the most common wind direction in the weather file is quite trivial to do and would not be an acceptable question. A good example from the baseball document: one could try and come up with a query that would accurately predict the most valuable player for a given season. One could combine the weather and baseball datasets to determine the likelihood a player will hit a homerun given certain weather conditions (wind, temperature, humidity, location, etc). Because I do not know if everyone has taken the statistics course, I do not know your statistics background. Therefore, your use of statistics in your

[3] For those interested, this is called the triangle inequality. The input will not break this rule.

questions and answers is completely up to you. I also can't assume you have had the visualization course so feel free to submit your answer to this question in any reasonable format.

If you can find a dataset that is many gigabytes, you'll be able to see the power of cloud computing on AWS. Assuming you found a dataset that is many gigabytes in size, transferring it from your local computer to S3 might take a long time. However, using wget on an EC2 server and then transferring it to S3 from your EC2 instance will be much faster. As a reminder from a previous activity, the following command uploads all contents of the current folder to some bucket in S3:

aws s3 sync . s3://name-of-your-bucket-here/folderName

To give you an idea of cost when running mapreduce on a large file, I've tested a file on the order of 10GB. I created 17 m3.xlarge instances (1 master and 16 core nodes). With 17 instances, I ran a rather simple Pig script and it cost roughly $6. I ran a similar program in the past with 10 instances and noticed a considerable speedup. That particular running cost about $4. Choose your instance size wisely. This is not something you want to run more than a couple of times especially if you are using large instances.

The following are what you should submit, at minimum, for this problem:

a. Where you obtained your data. A link to the data is sufficient. Please do not uploaded many GB's of data to the dropbox.

b. The questions that you asked of your data in a reasonable format (e.g. a Word document).

c. The answers to the questions in a reasonable format.

d. The code you used to obtain your answers.

What to Submit

When you are finished, zip up the following and upload a single zipped file to the Final Project dropbox:

1. prob1output.txt
2. Code from problem 1
3. Code from problem 2
4. Everything detailed above for problem 3.