CS 221 – Lab #10

This document describes today's lab and your next programming assignment. Enjoy!

Lab: Shortest Distance

In our previous lab, we looked at the problem of finding the centroid of several points, each with its own x/y location and weight. Today, we'll look at a slightly different problem. It's important to appreciate that a minor change to a question often leads to a significant change in how we formulate a solution. You will also practice reading input from a file, and using built-in mathematical functions.

Suppose you wanted to organize a conference, and you wanted to make the venue as convenient as possible for all potential attendees. The way to do this is to calculate the weighted average distance that people would have to travel to make it to your location. And you would need to perform this calculation for each possible location named in the input file. When you are done processing the data, you find the location with the *shortest weighted average distance*.

Note that the answer to this question is not the same thing as the centroid. For example, if your locations were all in the Hawaiian Islands, the centroid could be in the middle of the ocean. This might not be a good location for a conference. ☺

The first thing to realize about this problem is that it requires a nested loop. For each location i, we need to look at all locations j to compute the distance from i to j in order to calculate the weighted average. In the previous program you worked on, finding the centroid, you only need to make a single pass through the entire list of locations, and this could be done conceptually with a single loop. Because today's program requires a different loop structure, you will need to think about how you want to distribute your data among your processes, and which appropriate MPI functions to use for communication.

One minor difference today is the format of the input. I have added a name to each location. Please download and use any of the plain text input file(s) from the lab10-distance folder online.

Input format: tokens separated by commas. The first two tokens identify the location. The remaining tokens are like before: x (latitude), y (longitude) and weight (population).

Distance calculation: You may use the ordinary distance formula to find the distance between points, as if you were finding the hypotenuse of a right triangle. To convert to miles, assume that 1 degree of x (latitude) is 69 miles. And assume that 1 degree of y (longitude) is 69 times the cosine of the average x (latitude) between the two points. For example if the two points are at latitudes 30 and 40, you would need the cosine of 35

degrees. Use the built-in square root and cosine functions. Also, be sure to convert your degrees to radians before using cosine. Finally, you need to compile your program with the –lm option in order for your program to include the math library.

Output: When your program is done, please print out all of the locations, each with its weighted average distance. Finally, tell the user which location was the most convenient.

Compare the output of today's shortest distance program with your centroid program.

Programming assignment #2 – "Letters Game" – due 5:00pm on Friday 5/26.

If you have ever played the board game Scrabble or the online game Apterous, you know that the main skill of the game is knowing some fairly long words when given a random selection of letters. It's not practical to memorize the entire dictionary, though some people try! So, to excel at word games like this, it is helpful to know which long words are more likely to turn up and be useable in the game.

Specifically, you will write a (parallel) program that analyzes the following letters game scenario. In the game, a *selection* of letters is a random collection of 9 letters, of which either 3, 4, or 5 letters must be vowels. Given a selection, we would like to know which words can be formed using the letters in the selection. For example, the selection (S,M,T,E,I,O,R,E,U) can yield the possible 8-letter words TIRESOME, EMERITUS, MISROUTE and MOISTURE. But no 9-letter word is possible. The longer your word is that you can find, the more points you can receive. For example, in Apterous, if you find an 8-letter word and your opponent only finds a 7-letter word, you get 8 points and your opponent gets nothing.

Your program will not play the game, just analyze it in order to help players know which words are more likely to turn up. Your program will need to run millions of trials of generating random selections of letters. For each selection, scan the dictionary, as given by words2.txt (see the class Web site for the latest version), to find words that can be formed from the selection. You need to keep track of statistics for each word in the dictionary, so that at the end of your trials we can evaluate each word to see which ones are the best ones worth knowing for playing the game.

In order to generate a selection of letters, first decide whether the 9 letters will include 3, 4 or 5 vowels. There should be an equal probability for each. Next, it's time to choose the letters themselves. Use the following vowel and consonant probability distributions (given here in percent) when choosing letters randomly:

A	E	I	O	U
22	30	20	19	9

B	C	D	F	G	H	J	K	L	M	N	P	Q	R	S	T	V	W	X	Y	Z
3	4	8	3	6	3	1	2	7	5	10	5	1	11	12	11	2	2	1	2	1

Your program needs to answer the following questions concerning solutions of the letters game.

1. What are the most common words of length n, where n = 3..9?

2. What are the most common longest words? In other words, words that are guaranteed to score points. For a given selection, these are the words that have the longest length of any solution.

3. In Apterous lingo, a “darren” is a uniquely optimal solution. So, what are the most common darrens? A word is a darren if it is the longest solution possible, and the only solution of its length. Once you have computed darrens and equally interesting question is: what are the most common double darrens? To be a double darren, a word must be the uniquely longest solution, and there cannot exist any solution word 1 letter shorter. For example, this can happen if the longest possible solution has 7 letters, there is only one such 7-letter word, and there are no solutions of length 6.

Output: Your program needs to produce several sorted lists of words that answer the above questions. Each time a word is listed, you need to give the number of times that it appears in that category. For example, if you are printing the darren list, how many times each word appears as a darren.

1. The 1000 longest n letter solutions, for each n from 3 to 9 inclusive.

2. The 1000 most common longest solutions.

3. The 1000 most common darrens and 1000 most common double darrens.

This is a longer program than the others you have worked on. Please work on it in stages and enjoy the experience. You may find it helpful to write the entire program in a serial version so that you are sure that your overall C algorithm is correct, before parallelizing it.