CPSC 2120/2121

Lab 06

Nearest Neighbor Search
Due date/time are on the CPSC 2121 Canvas website

Learning Objectives

- To improve our ability to implement algorithms and data structures in C++
- To implement a recursive algorithm in C++
- To use simple data structures to implement algorithms
- To become proficient in fundamental data structures used throughout computer science
- To improve our analytical skills while gaining familiarity with mathematical tools used in the analysis of algorithms
- To implement an algorithm and understand its intricacies

Problem/Exercise

For this lab, you will read points in 3-space (x,y,z) from a file and implement an algorithm to find the two nearest points and the distance between them. You are allowed to use any data structures in the C++ STL for this assignment. Your final code will be in a single file, named nearest_neighbor.h, which will compile without errors using the provided Makefile. In order to thoroughly test your program, you will also submit a single input file named lab06.txt which has data points in the same format as the provided lab06.txt.

Additional Requirements

- Your nearest_neighbor.h file must compile and run with unmodified versions of the provided lab06.txt and Makefile to produce the exact same output as our examples. If your class does not, then you'll need to fix it. After you get your class working, you should create several examples on your own to further test your program to make sure it works with any valid set of test cases.
- Your algorithm must use a recursive implementation of a divide-and-conquer algorithm to solve this problem.
- Your program must use Euclidean distance as the metric. See Wikipedia for the definition and formula if you are not familiar with Euclidean distance. Round your distance to 3 decimal places.

Examples

Your C++ source code should be in a file called nearest_neighbor.h, and it should be compiled into an executable called lab06.out using our provided Makefile. The output of nearest_neighbor.h must look exactly like the examples below when run on the command line on our Unix machines.

./lab06.out
Nearest neighbors: (1,2,3) and (1,2,5)
Distance: 2

Nearest neighbors: (-4,6,1.5) and (-3.5,6,0)
Distance: 1.581

Nearest neighbors: (50,34,62) and (52,23,60)
Distance: 11.358

Nearest neighbors: (3,4.8,6.3) and (7,5,5)
Distance: 4.211

Source Code Requirements

- Put a comment at the top of your source code file with your name (first and last), the date of your submission, your lab section, and the assignment's name.
- All functions should be commented. Use inline documentation as needed to explain ambiguous, tricky, or important parts of your code.
- All source code must follow good programming style standards such as properly indenting source code; and all variables, functions, and classes should be well-named.
- Your class must use dynamic memory allocation and deallocation properly, which means your class cannot contain a memory leak. You may use valgrind to check for memory leaks. Refer to the Unix manual and valgrind's online documentation for more information on valgrind.
- Your program must read input from a file. This file will be in the format *x,y,z*, with one such entry per line until a line is only "######"

- For each input file, print the x,y,z coordinates of the two nearest points to stdout, as well as the distance between them, following the format in the examples above.

Submission

Before the date/time stated on the CPSC 2121 Canvas webpage, you need to submit your code to handin under the correct lab assignment. Make sure to submit all of the following.

1. All source files required for this lab (nearest_neighbor.h)
2. Your testing file (lab06.txt)

After you submit, always double check that the file(s) you submitted were the correct version. To double check, download the submitted file(s), put them on one of our Unix machines, and make sure they compile and run correctly.

Grading

If your class does not compile on our Unix machines or your assignment was not submitted on time, you will receive a grade of 0 on this assignment. Otherwise, your class will be graded using the criteria below.

Your class works correctly with our testing program(s)	6 points
Proper variable names, documentation, and code organization	2 points
Your class uses dynamic memory allocation/deallocation properly (no memory leaks)	1 point
Alternate testing file containing map and query data you created for testing which is distinct from that provided.	1 point

You must test, test, and retest your code to make sure it compiles and runs correctly and efficiently on our Unix machines with any given set of valid inputs. This means that you need to create many examples on your own (that are different from the provided lab06.txt) to ensure you have a correctly working class. We will only test your program with valid sets of inputs.