Please don't look around or talk with others. Cheating results in 0 credits!

2016-6-11

1. Examination time: 8:30-10:30.
2. Submission time: Please start to submit your source code by Tsinghua Web Learning (网络学堂) at least before 10:30. The submission site will be closed at 10:40. You may also submit your source code by USB drive to our TA if you cannot access internet. Please NOTE that late submissions are NOT accepted.
3. Submission requirement: Please follow the rules for our homework submission.

1. Bug fix (Credits 40%)

(1) Fix the bugs (including memory leaks) in the given source code. Mark the bugs in the code with comment "//BUGFIX", and then modify the code to fix them (note: there are four major bugs to be found); (20%)

(2) Implement another sorting algorithm (e.g. bubble sort, 冒泡排序), and reorganize the code using strategy design pattern (策略模式). Please report the runtime of the two methods (if needed, please increase the data size, i.e., the number of students) to verify their efficiency. (20%)

2. K-D Tree (Credits 60%)

From Wikipedia (https://en.wikipedia.org/wiki/K-d_tree), the definition of a k-d tree is as follows: "a k-d tree (short for k-dimensional tree) is a space-partitioning data structure for organizing points in a k-dimensional space. k-d trees are a useful data structure for several applications, such as searches involving a multidimensional search key (e.g. range searches and nearest neighbor searches). k-d trees are a special case of binary space partitioning trees."

Please read the given source code for a sample implementation of the k-d tree, and other related classes (BaseNearestNeighbor and MyNN) for using the k-d tree. Our goal is to reuse the k-d tree class for our nearest neighbor query tasks (复用 k-d 树完成最近邻查找应用).

Requirements:

(1) Add the constructors for class BaseNearestNeighbor in BaseNN.cpp and for class MyNN in MyNN.h; (10%)

(2) Fill in the code for member function BaseNearestNeighbor::search() in BaseNN.cpp to search for the nearest neighbor and store visited points; (10%)

(3) Define the member functions getNN() and getVisitedPoint() such that the codes could work correctly as required in main.cpp. Please test the program on 3D (in real applications, there are more dimensions for the data points) integer points as follows: generate the integer points (x, y, z) (x, y, z ∈ [0, 5, 10, 15, …, 95, 100]), and then query the point (6,7,8); (10%)

(4) Write the overloaded stream operator "<<" for class MyNN such that it could work correctly in main.cpp. The stream operator should report at least the query point (所查找的点), the computed nearest neighbor (最近邻点), and the number of visited points (访问过的点); (10%)

(5) Assume the distance between two k-dimensional points (x_1,x_2,…,x_k) and (y_1,y_2,…,y_k) is defined as $\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_k-y_k)^2}$. Please write another class named SimpleNN in SimpleNN.h and SimpleNN.cpp, which directly computes the nearest neighbor from all the given points. By the results of Simple NN, we may check whether the results of k-d tree are correct. Please report the runtime of the two different methods to verify the efficiency of the k-d tree; (10%)

(6) Make another copy of the source code in a new directory "source2" for template (模板) implementation of the k-d tree structure. By using the template KDTree, the original data type "int" can be changed to "short", "float", or "double". Modify all the related source codes (it is not required to write templates for MyNN and BaseNearestNeighbor) including main.cpp to test the template KDTree

on "float" data. It should be tested on 3D floating points as follows: generate the floating points (x, y, z) (x, y, z ∈ [0.0, 0.5, 1.0, 1.5, …, 9.5, 10.0]), and query the point (5.6,5.7,5.8). (10%)

NOTE for Problem No. 2:

(1) It is NOT allowed to modify existing code unless asked to. And it is NOT allowed to add new member functions (e.g., for classes MyNN and BaseNearestNeighbor), unless it is required to do so.

(2) In Question (6), it is not suggested to compare between floating numbers. Therefore, it is suggested to modify KDNode::equal() for the template implementation. For example, if |x1-x2| < 1e-6, then we can regard x1 and x2 as equal to each other. Moreover, please **be careful about all the distance computation parts for template implementation**, including KNode::distance2(). Otherwise, there may be errors in the query results.