Programming assignment 2

Due: Apr 13, 2:59 pm EST

Submit through canvas

Everything submitted to this programming assignment has to be entirely your own work. Open-source code/library is strictly not allowed. Copy-paste or nearly copy-paste from others (students within the class, students from other/previous class, anyone else) is strictly not allowed. Please refer to the syllabus and the "Student Responsibilities and Statements of Academic Integrity & Honor Code" as you read in your reading assignment for details. Identical or near identical answers/report/code/documentation will be strictly considered as plagiarism.

In this assignment, you are asked to implement various data structures and algorithms including binary search tree (BST) and quick sort. You are asked to submit everything (*.h and *.cpp files) that is required to fully compile and run your code.

You are given an input file which has the following format: in the first line there is one number, which is the number of following lines in this file. In each of the following lines, there are two field separated by a single empty space. The first field is a number and the second field has a string. An example input file is as follows:

5

10 absc

-1 bcd

100 yxla

100 x

4 ifajo

That is, the number 5 in the first line indicates there will be 5 lines (number-string pairs) in the rest of the file. Then in each of the 5 following lines, the first field is always a number and second field is a string of various length (minimum length is 1).

You are asked to:

1. Read from the given input file all the number-string pairs into a proper data structure/class. You need to design the data structure/class such that it is easy to use for the following tasks. You need to implement the data structure/class in a class named **NSPair** and submit NSPair.h and NSPair.cpp, and the class NSPair needs to have a function named **ReadInNSPair** which reads the input file and construct the class properly (10 points). You need to describe the data structure/class in the README file (what member variables/functions it has; why you think it is good for the tasks, etc) (5 points).
2. For all the number-string pairs, do a sorting as follows: first sort the number-string pairs based on their numbers in increasing order numerically. If multiple pairs have a same number, then sort them based on the strings in increasing order alphabetically. The sorting result for the above example input file is as follows:
   -1 bcd
   4 ifajo

10 absc
100 x
100 yxla
That is, the pair “100 yxla” will be ordered after “100 x” since alphabetically string “x” is smaller than “yxla”.
You are asked to

a. Implement a function named QsortNSN that sorts the number-string pairs numerically. QsortNSN should implement a quick sort (15 points) on the number-string pairs and its input should include the data structure/class you designed in NSPair.h/NSPair.cpp (that is, QsortNSN has to take number-string pairs as input, not the numbers). QsortNSN needs to be implemented in an in-place fashion (5 point), that is, no extra space is allowed to be allocated during the course of the execution and every sorting has to be done within the input space.
b. Implement a function named QsortNSS that sorts the number-string pairs alphabetically. QsortNSS should again implement a quick sort (15 points) on the number-string pairs and its input should include the data structure/class you designed in NSPair.h/NSPair.cpp (that is, QsortNSS has to take number-string pairs as input, not the strings). QsortNSS needs to be implemented in an in-place fashion (5 point), that is, no extra space is allowed to be allocated during the course of the execution and every sorting has to be done within the input space.
c. Implement a function named QsortNS (10 points) which calls QsortNSN first and then QsortNSS so as to properly sort all the number-string pairs. QsortNS needs to output the sorting results into an output file named QsortNS_results.txt (5 points) which should have the same format as the input file.
d. (Bonus credits) Implement a single function QsortNSNS (30 points) that takes a function pointer (a pointer to a comparison function) as its input in addition to the number-string pairs so that it can be used as QsortNSN and QsortNSS, depending on what function pointer is supplied. Implement a function named QsortNS2 (10 points) that calls QsortNSNS and output the sorting results into an output file named QsortNS_results_bonus.txt (5 points).
e. Implement a function named IsortNS (20 point) that sorts the number-string pairs. IsortNS needs to implement an in-place insertion sort. When you insert a number-string pair, you need to decide its proper position based on both the number and the string. IsortNS needs to output the sorting results into an output file named IsortNS_results.txt (5 points).
   All the above functions QsortNS, QsortNSN, QsortNSS and IsortNS (and QsortNSNS, QsortNS2) need to be implemented as a member function of NSpair class.

3. Construct a binary search tree (BST) from the input file. Submit a BST.h and BST.cpp that contain all the required functions.
   a. First construction a BST from the input file and implement the BST construction in a function called ConstBSTNS (20 points). In the BST, the nodes will be the number-string pairs. When you compare two pairs, first use their numbers. If the numbers are equal, then compare their strings.
   b. Do an inorder traversal on the BST and output the results into an output file BST_inorder.txt (5 points). You need to implement the inorder traversal in a recursive function named InorderBSTNS (20 points).

c. Find the “maximum” string from the BST and output the results (the number-string pair which contains the maximum string) into an output file BST_maxStr.txt (5 points). You need to implement this in a recursive function named FindBSTmaxStr (20 points).
d. (Bonus credits) Do a level-by-level traversal on the BST and output the results into an output file BST_level.txt (5 points). You need to implement the level-by-level traversal in a function named LblBSTNS(20 points) using a queue. Implement the queue (function enqueue (10 points) and dequeue (10 points)) and submit a queueNS.h and queueNS.cpp file.

4. What to submit
   a. Source code in a zipped fold called **NS.zip**. You need to submit all the necessary codes, including the required files/classes and anything else that is required to correctly compile or run your code.
   b. Submit a main.cpp file which should implement the following pseudo code:

```
Main(){

Read in number-string pairs


tic

QsortNS /* QsortNS should call QsortNSN first and then QsortNSS, and then output results*/

toc


tic

/* if you do bonus credits */

QsortNS2 /* QsortNS2 should call QsortNSNS and output the results*/

toc


tic

IsortNS /* insertion sort and output the results*/

toc


ConstBSTNS /* construct a BST*/

InorderBSTNS /* in-order traversal and output the results */

FindBSTmaxStr /* find the max string from the BST and output the results*/

/* if you do bonus credits*/

LblBSTNS /* level-by-level traversal and output the results */
```

}

Where the tic-toc pair means you need to time the corresponding functions (QsortNS, QsortNS2 and IsortNS) and output the timing results into standard I/O (10 points).

c. A README file describe
    i. NSPair data structure/class
    ii. The algorithm you used in QsortNS
    iii. The algorithm you used in IsortNS
    iv. The algorithm you used in FindBSTmaxStr
    v. The algorithm you used in QsortNSNS (if you do bonus credits)
d. Your code should be developed on Linux and compile using g++. Please give a comment line you used to compile the code. For example, if you compile your code using
   g++ you_code
   then in the README file, please explicitly state this command. Please provide all necessary information in the README file. It is expected that the TA/graders can successfully compile and run your code just based on the README file.

5. Grading:
    a. Please strictly follow the naming scheme and use the names specified as in this description for the functions and files. If you are not using these names, 20 points will be taken off (non-negotiable!)
    b. Your code has to be able to compile on machine pegasus.cs.iupui.edu. If the compilation is not successful by copy-paste the compilation comment provided in your README file, one third of the total points will be taken off (non-negotiable!)