CS 210 Data Structures (Shen) - Fall 2022

Programming Assignment 3 (**120** points)

Due Date – **Beginning of the class**, **11/02/2022, NO LATE** submission

You **must** do this assignment on your own.

A company is organizing a team building event. Each employee in the engineering group is encouraged to partner with another employee in the same organization to form a team of two to participate in the event. The two employees wanting to form a team need to go through the managers between them to get approvals. You would write a class specified in the **Part A** below to help facilitate the approval process, then answer a follow-up question related to time complexity in **Part B**.

Part A – Coding (90 pts)

Implement the **Orgtree** class in **orgtree.cpp** (code skeleton is provided on Canvas) to retrieve information from a given organization chart/tree.

Code Skeleton given to you to start with:

- An **Employee** class representing an organization chart (tree) node is provided to you. You must **NOT** change this class.
- An **Orgtree** class declaring the tree operations is also provided to you, you **must NOT** change any **signature** (or declaration) of the given tree functions; otherwise, your code would automatically fail some or all auto-grading tests on Gradescope. You will write the implementations for those operations following what's specified in the comment section above each function. You may add helpers in Orgtree class for helping your implementation.
- **Makefile:** A makefile to allow you to automate the compilation of your code. Check out the "Frequently Asked Questions" link in the C/C++ programming in Linux / Unix reference page on Canvas for simple Makefile reference.
    - Use the "make" command to compile. Use "make clean" to delete compiled object code and the executable.
    - Use **./orgtree** to execute.
    - You MUST **NOT** change the executable file name **orgtree in the Makefile**, otherwise, autograding will automatically fail**.**

Assumptions:

1) No two employees can have a same employee ID.
2) An employee ID can be inserted to the tree only ONCE. Autograder testing code will insert each employee ID only once to the tree.

Algorithm Requirements:

1) You **must** follow the **specifications** in the **comment section** **ABOVE** each function **in the orgtree.h or orgtree.cpp** for each function implementation.
2) You must use **recursive** algorithm to implement all functions **except** the findNumOfManagersBetween method; see comments in each function. Using an iterative approach for any required recursive function would result in a **30% penalty** to your assignment 3 grade.

Hint:

1) Follow the comments inside each function for hints. You do not have to implement what the hints suggest, as long as your implementation functions and satisfies the above requirements.

Testing:

1) **You MUST write your own testing code (see grading below). A driver.cpp** (available on Canvas) is provided with sample testing code to get you started on writing your own testing cases. Follow the comments there to write test code to test your implementation according to the specifications.
2) Please note: Autograder would only reveal the feedback of a limited amount of test cases (including the valgrind memory leaking detection, see below). A few test case results will be **withheld and only published** after the due.
   This is done to encourage you to write and use your own testing code for testing. Remember, that is part of your responsibilities as a software engineer or computer scientist. Establishing good habits of writing testing code would go a long way to help you become a true computer professional.

Memory Leak Detection:

1) Valgrind will be used to test your code for memory leaks caused by improper memory deallocation. Anytime the 'new' keyword (or malloc) is used, 'delete' (or free) must **eventually** be used to free the memory when that object is no longer required.
   i. You **ONLY need to worry** about deallocating memory properly with the **deleteOrgtree** function.
   ii. At the end of the driver.cpp code, you must call **deleteOrgtree** to delete the allocated memory for the tree. If your deleteOrgtree successfully deletes all the allocated Employee objects on the heap, valgrind test will pass.
2) You can run valgrind locally (on your Linux or Unix system) or on Edoras, Autograder will also show the output from valgrind as part of the feedback from testing your code.
3) Much like using GDB, using valgrind as a habit improves your C/C++ programming and debugging skills!
4) Helpful resources:
   https://valgrind.org/docs/manual/quick-start.html
   https://stackoverflow.com/questions/5134891/how-do-i-use-valgrind-to-find-memory-leaks

Programming and testing:

- Please refer to C/C++ programming in Linux / Unix page.
- You may use **C++ 11** standard for this assignment, see the given Makefile.
- We strongly recommend you set up your local development environment under a Linux environment (e.g., Ubuntu 18.04 or 20.04, or CentOS 8), develop and test your code there first, then port your code to Edoras (e.g., filezilla or winscp) to compile and test to verify. The gradescope autograder will use a similar environment as Edoras to compile and autograde your code.

Grading for Part A:

Passing 100% auto-grading may NOT give you a perfect score for this Part A. The satisfaction of the algorithm requirements (see above), you code structure, coding style, and commenting will also be part of the rubrics (**see Syllabus Course Design - assignments**). Your code shall follow industry best practices:

- Testing code: your testing code in driver.cpp will be inspected as part of the manual grading for Part A (refer to comments in driver.cpp), points will be deducted for NO, minimal or insufficient testing code, meaning your tests need to cover necessary testing for every function you implemented.
- Be sure to comment your code appropriately. Code with no or minimal comments are automatically lowered one grade category.
- NO hard code – Magic numbers, etc.

- Have proper code structure between .h and .c / .cpp files, do not #include .cpp files.
- Design and implement clean interfaces between modules.

Part A grade breakdown:

- Main Autograding Test Cases (80%)
- Quality of your testing code (refer to comments in driver.cpp) (15%)
- Memory Leak detection from valgrind (5%), see above

Part B – Complexity Analysis (30 pts)

To improve the search efficiency of finding the closest shared manager between two employees, the organization tree data structure is augmented by adding a parent node pointer to the Employee node class, **devise an algorithm** that:

Given two employee nodes, in the worst-case scenario, the algorithm would take **O(h) (h is the height of the tree)** time to locate the **closest shared manager** between the two employees.

- **Write pseudo-code of findClosestSharedManager (or C syntax code)** to **describe the algorithm.**
- **Present the worse-case scenario, and**
- **Show the algorithm has an O(h) time complexity.**

Note:

1) **For the time complexity analysis**, you would need to analyze the implementation logic for the **worst-case** scenario to separate the constant and non-constant operations with respect to input size, then show and explain the time complexity is **O(h).** Follow **zyBook 12.7.1 and 12.7.3** for **time** complexity analysis.
2) You **do not need** to consider any constraints on the auxiliary space complexity for the algorithm.

Turning In

You need to submit the following program artifacts on Gradescope. Make sure that all submitted files contain your **name** and **Red ID**.

- **Part A:**
  - You should ONLY submit **orgtree.h, orgtree.cpp, and driver.cpp** source code files. **Do not** upload the Makefile, nor any of the compiled .o files. **Orgtree.h** is only required if you made changes to the original.
  - Make sure you type the **Single Programmer Affidavit** (refer to the template on Canvas) as part of the comments at the beginning of your **orgtree.cpp file**.
- **Part B: One PDF file**. It is recommended that you type it up, however, a scanned copy is allowed so long as the handwriting is legible. You can convert Word (.docx) files to PDF using the "Save as" functionality. You must only submit a **single** PDF file, image files (.jpg, etc) **will not** be accepted.
- **Important**:
  - Upload your files directly to Gradescope.
  - Do NOT **compress / zip** files into a ZIP file and submit, submit all files as they are.
  - Do NOT submit any .o files.
- **Number of submissions:**
  - Please note the autograder submission count when submitting on Gradescope. For this assignment, you are **limited to a maximum of 10 submissions.** As stressed in the class, you are supposed to do the testing in your own dev environment instead of using the autograder for testing your code. It is also the responsibility of you as a programmer to sort out the test

cases based on the requirement specifications instead of relying on the autograder to give the test cases.

Academic honesty

Posting this assignment to any online learning platform and asking for help is considered academic dishonesty and will be reported.

An automated program structure comparison algorithm will be used to detect code plagiarism.

- The plagiarism detection generates similarity reports of your code with your peers as well as from online sources. We will also include solutions from the popular learning platforms (such as **Chegg**, etc.) as part of the online sources used for the plagiarism similarity detection. Note not only the plagiarism detection checks for matching content, but also it checks the structure of your code.
- Refer to Syllabus for penalties on plagiarisms.
- Note the provided source code in the code skeleton would be excluded in the plagiarism check.