1801ICT Object-Oriented Programming
School of ICT, Griffith University
Gold Coast Campus, Trimester 2, 2019

Project 4 on Data Structures in `C++`

Release 16.09.2019 Due 03.10.2019 23:59

Total Marks Allocated: 14

1 Answering Questions

1 mark in each of week 10 and week 11. Lab instructors will ask you to discuss a number of exercise questions (which ones that will be told in the lab) among yourselves and then ask questions verbally to check your understanding. Some small programming taks may also be given. Marks will be awarded based on your satisfactory performance.

2 Programming Problems

- Programs are to be submitted online by the due date and time. You can get your programs marked anytime in week 10 or 11, but week 12 lab is dedicated for marking on first come first serve basis.
- If you need any help, please ask the instructors in the lab; they can give you hints and suggestions.
- For any confusion for any part of this project, please feel free to email the course lecturer.
- You are free to use anything from the `C/C++` library and the Standard Template Library (STL).

3 Task 1: Student Result

This task involves developing a data base system for student result processing. For this, do not write your own classes, except functor classes when needed. Use any of `string`, `pair`, `tuple`, `array`, `vector`, `deque`, `list`, `forward_list`, `set`, `map`, `unordered_set`, `unordered_map`, `binary_search`, `lower_bound`,`upper_bound`, `sort`, and any other template classes or algorithms or functors as appropriate. Put comments explaining the choice you made over the alternatives.

1. You are supplied a program named `gendata.cpp`. Compile and run the program. Every time you run this program, it generates a set of mark entries for a number of students for four subjects namely physics, mathematics, chemistry, and biology. The data are output in lines where each line has a student's name, a subject name, and the mark obtained by the student in that subject. Not necessarily marks of all subjects of a student appear in consecutive lines. Not necessarily marks of all students for a given subject appear in consecutive lines. If a student's mark for a subject is missing, assume it to be zero.
2. You can read the data from the console since input, output, and error redirection from the terminal are allowed. For more hint, you can use `cin` to read each word. As each line has three words/numbers separated by spaces, there is no complicacy in reading the data.
3. Use appropriate classes or template classes from STL to store student data: the name, the marks, and the total marks of each student. Do not define your own classes.
4. Store student data for all students in a container class. Note that once stored, we do not want any movement of any data later e.g. for sorting based on marks or names. Movement of large-sized data is coslier. Do not define your own classes.
5. You need to create an efficient container data structure to help quickly check whether a student name already exists or not. If a student name already exists, the index or the iterator for the student data is returned or the student name is inserted into the data structure. Do not define your own classes.

6. We might ask to display student results in their name order, or in the order of the marks of any subjects, or even the total marks. Note sorting the student data in the containers is not possible as mentioned earlier movement of large-sized data is costlier and we want to avoid it. You might have to write appropriate comparator functor classes and use indirections.

7. We might ask how many students have obtained more than a given marks in a particular subject or in total marks. You have to write appropriate comparator functor classes for this.

Marks: 7 marks in total: 3 marks for workable solutions and 3 marks for efficiency, 1 mark for justification of choices of various data structures.

4 Task 2: Shortest Path

This task involves comparing path finding algorithms on given graphs. You can use anything from the `C++` STL.

1. You are supplied a program named `gengraph.cpp`. Compile and run the program. Every time you run this program along with a command-line argument denoting the number of vertexes, the program generates a graph and outputs it in the adjacency matrix format. In the output, the first line contains a number n denoting the number of vertexes. Then the next n lines each has n numbers, which are in the range of 0 to 5; 0 to denote no link and non-zero to denote link cost.

2. You have to read a given graph from the output of the `gengraph.cpp` program. You will have to implement path finding algorithms given a source and a target vertex.

3. Implement the iterative version of the depth first search algorithm using a stack. You can replace the stack with a queue and you will get a breadth first search algorithm. If you replace the queue with a priority queue, you will get a uniform cost search algorithm. Check lecture slides for these algorithms.

4. In case of each algorithm, show the number of hops (one node to another is one hop) on the shortest path from the source to the target, the total length of the shortest path, the number of nodes that are added to the stack or queue or priority queue, the number of nodes that are popped from the stack, queue, or priority queue, and the time taken to find the shortest path.

Marks: 5 marks in total: 1 mark for implementing required data structures for search nodes, 2 marks for implementing one search algorithm, 1 mark for statistics collection, 1 mark for the other two algorithms.