Nexustec Gmbh

Test for the role of 'Software Engineer (C++)'

1. Solve at least one question from the following list of questions.
2. Complete the test on your own without the help from another person. However, you are free to use Internet resources / reference books as you may like.
3. Use CMake as the build system for your project, and provide a list of dependencies used.
4. You can use any additional libraries that you may want to use however please make sure to provide necessary references and installation scripts.
5. Your code should compile and run on a widely used desktop Linux distribution, such as Ubuntu 20.04. Try to use standard library functions, or cross-platform libraries to ensure compatibility.
6. You have 72 hours (3 days) to submit your solutions to this test after you get it by email. Please make sure you plan your time accordingly, ideally the test should not take more than 4 hours to solve, up to 6 hours including bonus steps.
7. Submit your solution as a zip file containing source files only (CMakeLists.txt .h, .cxx, *.md) along with necessary documentation wherever applicable.
8. All the best!

Question 1:

Traveling salesman problem is one of the most common problems that can be solved using Graph type data structures. This problem is handled at much higher complexity in modern world with lots of maps and mapping applications.

You can read more about the traveling salesman problem on Wikipedia here (https://en.wikipedia.org/wiki/Travelling_salesman_problem).

There are many widely known solutions for Traveling salesman problem, and you can also solve it using 2D arrays, or any other data structure that allows traversal between nodes.

Create an interactive command line application which allows users to perform following:

1. On program start, create and display a simple graph with 4 nodes where the nodes are as given in this (https://www.geeksforgeeks.org/wp-content/uploads/Euler12-300x225.png) image. Calculate and display the fastest route to visit all the nodes.

2. Display a help menu which prints the syntax for the following two steps, your program should wait until user explicitly exits the program.

3. Write a function to add another node in the graph, asking user to input distances to other nodes. Verify that the node Inserted by the user is valid, i.e. input contains distances to all the nodes already existing in the graph are presented. Following this recalculate the fastest route to visit all the nodes.

4. Allow user to delete a node present in the graph. When user select the delete option, print a list of all node names and allow user to choose which node to delete. After removing the node from the graph, recalculate the fastest route to visit all the nodes,

allow this option to be used such that shortest distance required to be traveled by the salesman is never zero.

5. **Bonus**: Print the graph in console and / or export to a .txt file.

Question 2:

Create an interactive version control system called `bit` which has following features.

1. User starts the program on a specified working directory. `bit` can store any data needed for its program state in a subdirectory named `.bit` in the working directory.

`bit` can be run for a project like this:

```
/path/to/executable/bit /home/user/my_project
```

Running this command should print a help menu, and the program should wait for user input.

2. User can commit changes to the system with a message.

```
commit "Final_version_noMorechanges_v2"
```

This will create a new version with the current files in the working directory. Each version will get a unique version number. If there are no new changes, `bit` will print a message saying so, and will not create a new version.

3. User can list previous versions.

```
list versions
```

This will print a list of all the versions with the commit message and timestamp in the format `YYYYMMDD_HHMMSS`

4. User can reset the state of the working directory to a given version.

```
reset 23
```

This will print a message showing the user that the files have been reset to the version 23, along with the commit message and timestamp for that version.

5. **Bonus**: User can list the uncommited (changed) files.

```
list changes
```

This will print a list of all the new and changed files since the last commit. You only need to list the changed files. (No need to show the difference between file contents)