CS575 Design and Analysis of Algorithms

Spring 2025

***Programming Assignment* 3**

Assigned: March 26, 2025
Due: Midnight Thursday, April 10, 2025

1. [45%] Implement the longest common subsequence (LCS) algorithm using the dynamic programming method that was discussed in class. (No credit will be given if you implement a brute force algorithm, which does exhaustive comparisons between two input strings, or any other algorithm unless you prove your algorithm is correct and more efficient than the LCS algorithm described in Chapter 7.) Save your source code in a file and name the file as *lcs.cpp or lcs.java*.

   Make sure that your program can take **any** two input strings in the Linux command line and print the LCS found between the two input strings. (Assume that a string consists of at most 100 alphabetic characters.) For example, if we type "lcs abc afgbhcd" in the command line to find the LCS between string "abc" and string "afgbhcd". Again, your program should work for arbitrary two input strings. No credit will be given, if your program only works for some specific strings, but fails to find the LCS for other strings.

   Program Usage

   Your program should be invoked as follows.

   ```
   $> ./lcs <input-string1> <input-string2>
   ```

   A sample run of your program appears below.
   $> ./lcs ABCDEfghi AcbDedghaq

   A sample output is as follows (standard output in the terminal)
   Length of LCS: 4
   LCS: ADgh

2. [45%] Write a program *floyd.cpp or floyd.java* to find all pairs shortest paths using Floyd's algorithm for several undirected complete graphs, which are saved in a file called **output.txt**. Print all pairs shortest paths and their lengths.

   Program Usage

   Your program should be invoked as follows

   ```
   $> floyd <graph-file>
   ```

Graph File: `<graph-file>` is the name of a file that includes more than one problem. The lines that correspond to problem j will contains an integer n (between 5 and 10) that indicates how many cities and n x n adjacency matrix A (that is, the distance between n cities, between 1 to 10), in the next n rows. Note that no infinity will appear in the matrix A.

A sample graph file appears below.
Problem 1: n = 7
0 6 5 4 6 3 6
6 0 6 4 5 5 3
5 6 0 3 1 4 6
4 4 3 0 4 1 4
6 5 1 4 0 5 5
3 5 4 1 5 0 3
6 3 6 4 5 3 0

Problem 2: n = 6
0 1 2 1 3 4
1 0 3 2 2 3
2 3 0 3 3 6
1 2 3 0 3 5
3 2 3 3 0 5
4 3 6 5 5 0

Output File

Output the solution of problem 1 first, then problem 2, and etc. The solution of problem j should start with an integer n (the number cities) and the n x n Pointer Array P (in the next n rows). The shortest paths should then follow, one per line. Output the shortest paths from C1 to all other cities, then C2 to all other cities, and Cn to all other cities. Please note that the P matrix and shortest paths may be not unique. As long as you have the same shortest distances between cities, your answers should be OK.

A sample output file:
Problem 1: n = 7
P matrix:
0 0 0 6 3 0 6
0 0 5 0 0 4 0
0 5 0 0 0 0 5
6 0 0 0 3 0 6
3 0 0 3 0 3 0
0 4 0 0 3 0 0
6 0 5 6 0 0 0

V1-Vj: shortest path and length
V1 V1: 0
V1 V2: 6
V1 V3: 5

V1 V6 V4: 4
V1 V3 V5: 6
V1 V6: 3
V1 V6 V7: 6

V2-Vj: shortest path and length
V2 V1: 6
V2 V2: 0
V2 V5 V3: 6
V2 V4: 4
V2 V5: 5
V2 V4 V6: 5
V2 V7: 3

V3-Vj: shortest path and length
V3 V1: 5
V3 V5 V2: 6
V3 V3: 0
V3 V4: 3
V3 V5: 1
V3 V6: 4
V3 V5 V7: 6

V4-Vj: shortest path and length
V4 V6 V1: 4
V4 V2: 4
V4 V3: 3
V4 V4: 0
V4 V3 V5: 4
V4 V6: 1
V4 V6 V7: 4

V5-Vj: shortest path and length
V5 V3 V1: 6
V5 V2: 5
V5 V3: 1
V5 V3 V4: 4
V5 V5: 0
V5 V3 V6: 5
V5 V7: 5

V6-Vj: shortest path and length
V6 V1: 3
V6 V4 V2: 5
V6 V3: 4
V6 V4: 1

V6 V3 V5: 5
V6 V6: 0
V6 V7: 3

V7-Vj: shortest path and length
V7 V6 V1: 6
V7 V2: 3
V7 V5 V3: 6
V7 V6 V4: 4
V7 V5: 5
V7 V6: 3
V7 V7: 0

Problem 2: n = 6
P matrix:
0 0 0 0 2 2
0 0 1 1 0 0
0 1 0 1 0 2
0 1 1 0 0 2
2 0 0 0 0 2
2 0 2 2 2 0

V1-Vj: shortest path and length
V1 V1: 0
V1 V2: 1
V1 V3: 2
V1 V4: 1
V1 V2 V5: 3
V1 V2 V6: 4

V2-Vj: shortest path and length
V2 V1: 1
V2 V2: 0
V2 V1 V3: 3
V2 V1 V4: 2
V2 V5: 2
V2 V6: 3

V3-Vj: shortest path and length
V3 V1: 2
V3 V1 V2: 3
V3 V3: 0
V3 V1 V4: 3
V3 V5: 3
V3 V1 V2 V6: 6

V4-Vj: shortest path and length
V4 V1: 1
V4 V1 V2: 2
V4 V1 V3: 3
V4 V4: 0
V4 V5: 3
V4 V1 V2 V6: 5

V5-Vj: shortest path and length
V5 V2 V1: 3
V5 V2: 2
V5 V3: 3
V5 V4: 3
V5 V5: 0
V5 V2 V6: 5

V6-Vj: shortest path and length
V6 V2 V1: 4
V6 V2: 3
V6 V2 V1 V3: 6
V6 V2 V1 V4: 5
V6 V2 V5: 5
V6 V6: 0

3. [10%] 10% of the grade will be based on *good coding style* and *meaningful* comments.

All programming must be done using **C or C++ or java in Linux** (remote.cs.binghamton.edu) where your code will be tested. Create a tar file that includes (1) source code files, (2) a readme file that clearly describes how to run your code. Submit only the tar file through the Blackboard. The name of the tar file should be yourlastname_yourfirstname _proj3.tar (Do not use special characters like #, @, or &, because they have caused Blackboard problems in the past.) Suppose that your assignment files are under the directory of /your_userid/yourlastname_yourfirstname_proj3/ and you are under that directory right now. To create a tar file under /your_userid directory, do the following in Linux command line:

```
>cd ..
>tar cvf  yourlastname_yourfirstname_proj3.tar  yourlastname_yourfirstname_proj3
```

To view the content of the created tar file, do the following in Linux command line:
```
>tar tvf  yourlastname_yourfirstname_proj3.tar
```

Finally, read the following policies carefully:

- *All work must represent each individual student's own effort. If you show your code or any other part of your work to somebody else or copy or adapt somebody else's work found*

online or offline, you will get zero and be penalized per the Watson School Academic Honesty Code (http://www.binghamton.edu/watson/about/honesty-policy.pdf).

- *To detect software plagiarism, everybody's code will be cross-checked using an automated tool.*
- *Your code will be compiled and executed. If your code does not compile or produce any runtime errors such as segmentation faults or bus errors, you will get zero.*
- *The instructor and TA will not read or debug your code. The instructor and TA will not take a look at an emailed code. If you need general directions, show your code to TA during her office hours. The TA will not do programming or debugging for you though. The TA will only help you understand algorithms to be implemented and answer basic questions related to implementation.*