Assignment 0: Sockets and Graphs

Due date: Friday September 18th at 11:00pm Waterloo time

ECE 454/751: Distributed Computing

Instructor: Dr. Wojciech Golab wgolab@uwaterloo.ca

A few house rules

Collaboration:

- individual submissions please for this assignment
- you may discuss algorithms and implementation techniques with other students, but you must write your own code

Managing source code:

- do keep a backup copy of your code outside of ecelinux, for example using GitLab (https://git.uwaterloo.ca/)
- do not post your code in a public repository (e.g., GitHub free tier)

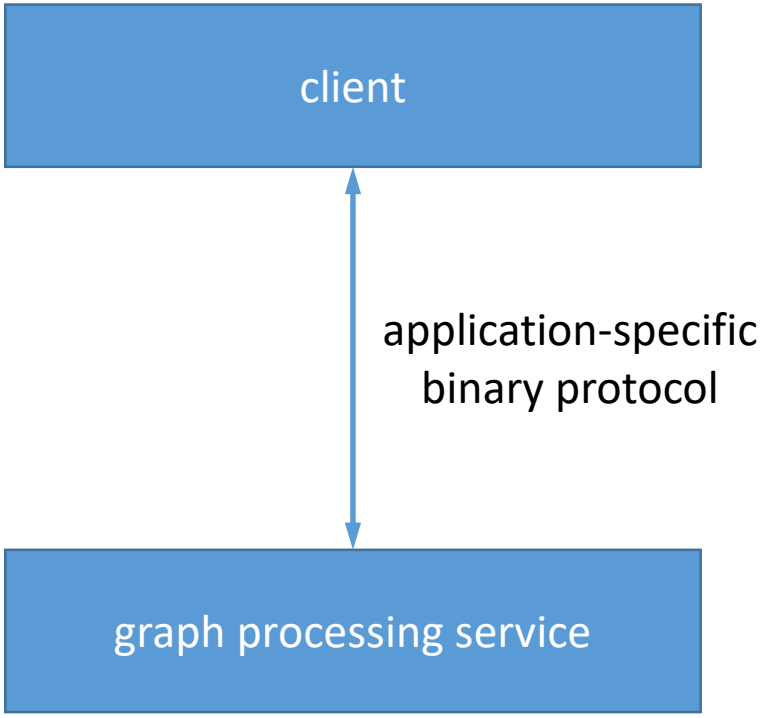
Software environment:

- test on eceubuntu and ecetesla servers
- the environment used for grading will provide Java 1.11

Overview

- In this assignment, you will develop a client-server system for processing graph data. You will be given some client and server starter code, and asked to complete the server's implementation in Java using sockets.
- The assignment has several objectives:
  1. to learn the basics of socket programming in Java
  2. to gain an appreciation of the difficult choice between efficiency and scalability
- This assignment is worth 10% of your final course grade, and must be completed individually.

Software Architecture



Protocol

- The client reads a graph from an input file and sends it in a request message to the server using TCP/IP.
- The server processes the graph and returns a response back to the client using the same TCP connection.
- The request and response messages follow the same format:
  - The first 4 bytes comprise a header that indicates the size (i.e., number of bytes) of the data payload that follows. This value is encoded as a **32-bit signed two's complement integer** using **big-endian** byte ordering.
  - The data payload is the **UTF-8** encoding of a **character string** that represents either the input graph (for a request) or the server's output (for a response). The string may contain multiple line breaks.

Input

The input is an **undirected** graph represented as a **list of edges**. Each line contains one edge, which is a pair of vertex labels listed in ascending numerical order, separated by a space.

Do not assume that vertex labels are consecutive, or that they start at 1. A vertex label may be any non-negative Java int. Vertex labels may be very large even if the input graph is small, and should not be used as array indexes.

Example input:

1000000 2000000

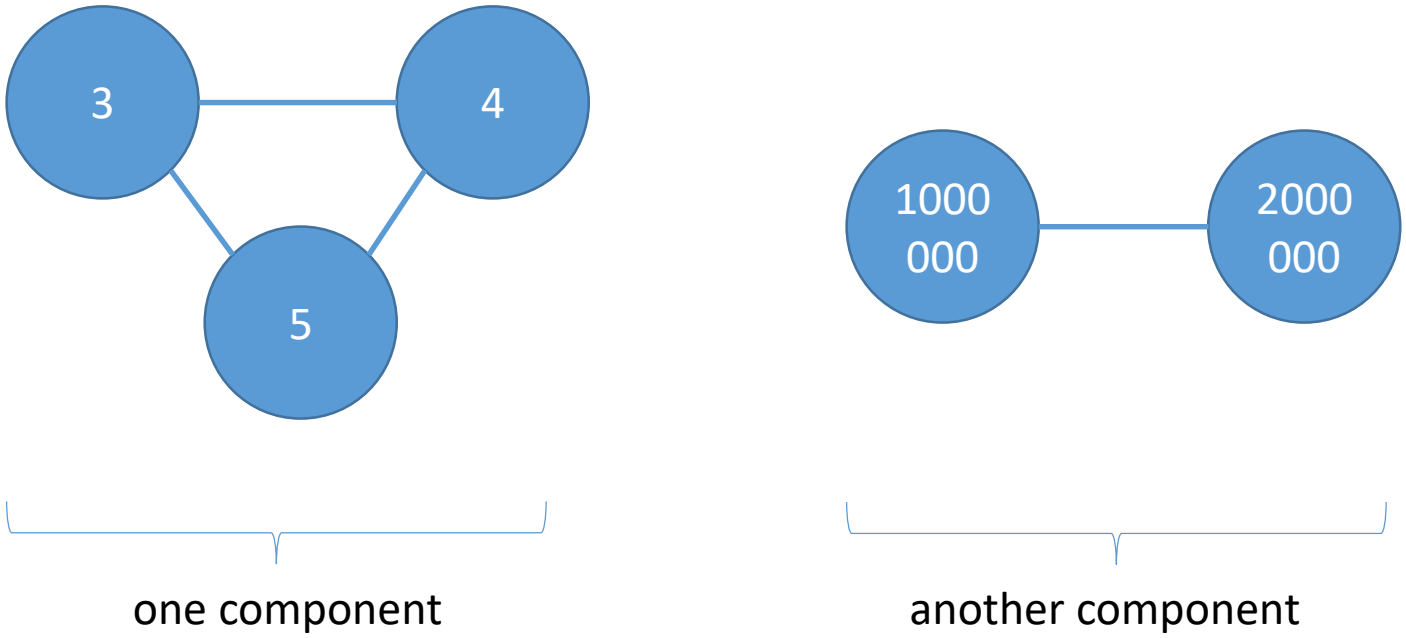
3 4

3 5

4 5

Input

The graph shown in the previous slide can be visualized as follows:



Output

Your goal is to solve the **triangle enumeration problem**. The output is a collection of lines, each representing a triangle (i.e., cycle of length 3). A triangle is encoded as a list of vertex labels in ascending numerical order, separated by spaces. The **order of the lines does not matter**. Each triangle must be output exactly once.

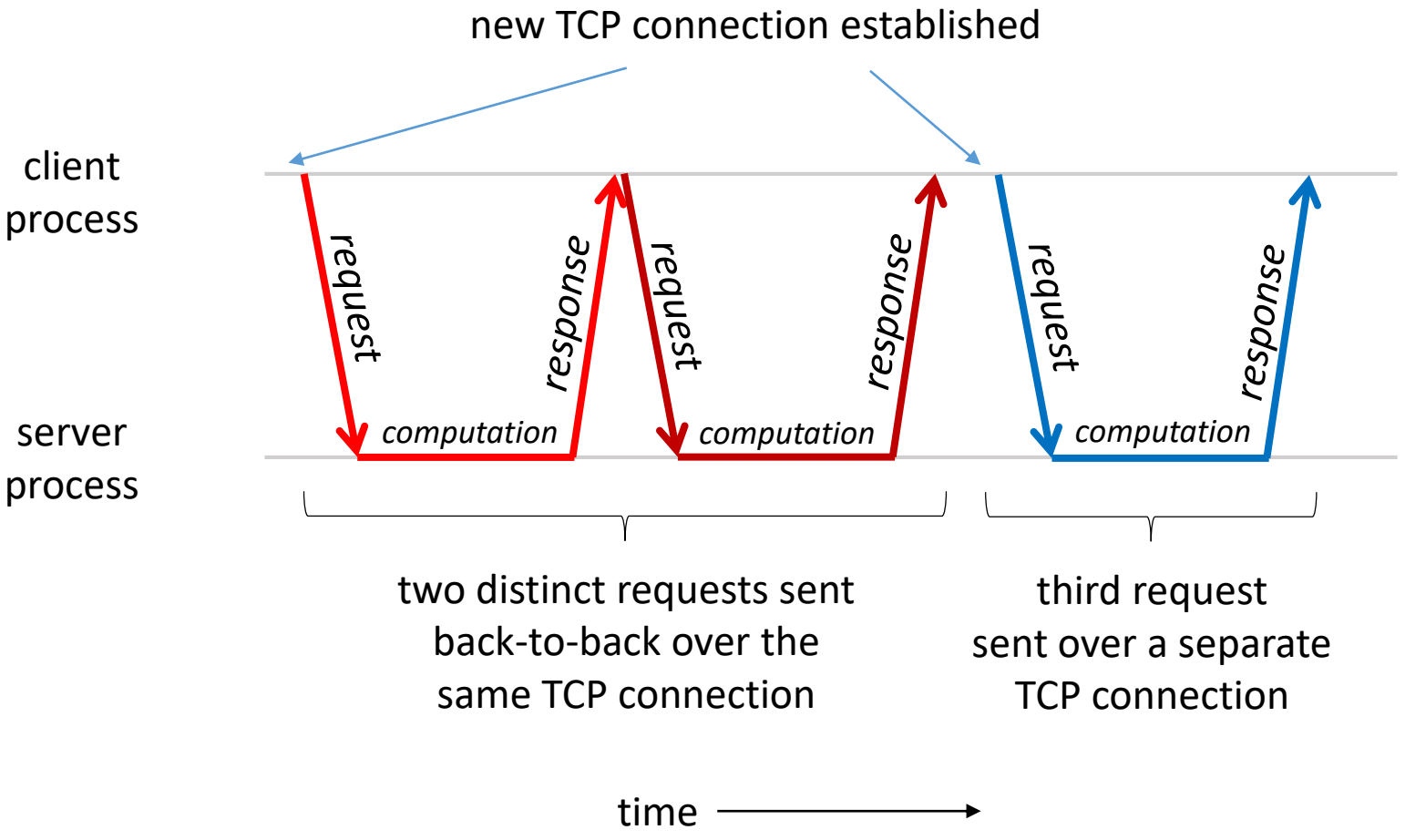
Example output for the input shown earlier:

3 4 5

Your coding task

- Your mission is to complete the Java server implementation by adding code for connection handling and graph processing.
- The Java server is implemented in the file **Server.java** in the default Java package. You may create additional source code files in the default Java package as you complete the implementation.
- The server program accepts one command line argument, namely the **TCP port number**. Do not change this part of the code.
- Inside the server program, you must implement a loop that performs the following steps:
  1. Accept a new client connection.
  2. Process requests from this connection repeatedly, **one at a time**.
  3. When the client closes the connection, go back to step 1.
- The above loop terminates when the server process receives a SIGINT (Ctrl + C), SIGTERM, or SIGKILL.

Example of client-server interaction



Packaging and submission

- All classes must be in the default Java package.
- Use the provided package.sh script to create a tarball for electronic submission, and upload it to the appropriate LEARN dropbox before the deadline.
- The tarball should contain only your Java files, all in the root directory of the archive.
- **Do not include any code that does not compile.**
- **Do not use any external libraries or jar files** except the ones provided under lib/ in the starter code tarball.

How to test your code

- Four sample inputs are provided, along with three sample outputs. Please test your code at least on these inputs.
- The largest input used for grading will be similar in size to sample_input/large.txt
- We will run your code on two cores (no hyperthreading) with 1GB maximum Java heap size. The test script will control the number of cores assigned to your process using the taskset command. Example:

  taskset -c 0,1 java -Xmx1g Server 10123

- The grading script will run on a 2.2 GHz Xeon E5-4620.
- We will wait at most 10 seconds for your code to terminate.

Evaluation

Grading scheme:

Correctness:	50%
Performance:	50%

The running times anticipated for the large.txt input on our grading platform are in the range from 10 seconds down to a fraction of one second.

Solutions whose performance is in the middle of this range (i.e., roughly 5s running time on our platform for large.txt) will receive 75% if they produce correct outputs. That means full credit for correctness, and half credit for performance.

Solutions that run correctly on small inputs and fail on larger inputs (e.g., produce incorrect outputs, run out of memory, or run for more than 10s) will receive partial credit for correctness, and no credit for performance.

Good luck!