Homework #8

This homework assignment requires submitting both written answers and code. All written answers should be submitted in a PDF file named `hw8.pdf`. Each individual question specifies the submission guidelines for the code.

Question 1 (4 pt.)

Modify the constructor of class `Graph` with the following prototype:

```
Graph(int size, float alpha = 0.0);
```

The new optional argument `alpha` (with a default value of 0.0) is a real number between 0 and 1 that determines an *approximate* initial load factor (number of edges) in the graph. A load factor of 0 means that the graph has no edges, while a load factor of 1 implies that the graph has all possible $|V|^2$ edges. For any other value in between, the edges are generated randomly, using the following approach:

- A pair of nested loops traverse every possible combination of source and target vertices (u, v) for which there could be a directed edge. Overall, the body of the inner loop executes $|V|^2$ times.

- In each iteration of the inner loop, a random number between 0 and 1 is generated. If the value is less than the target load factor, edge $u \rightarrow v$ is added to the graph. If the value is equal or greater than the load factor, no edge is added.

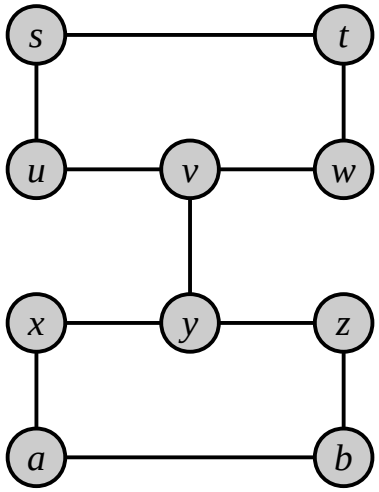
- The loops do not execute at all if the target load factor is set to 0, in order to avoid unnecessary overhead.

Write a main program that takes a load factor as a command-line argument, creates a random graph with this load factor, and prints it.

Upload your code in a ZIP file named `q1.zip`, containing all necessary files to compile and run your program correctly on the Linux CoE machines.

Question 2 (6 pt.)

Consider the following undirected, unweighted graph:



a) (2 pt.) Apply a manual trace of the Bellman-Ford algorithm to calculate the shortest paths starting at vertex *s*, and using the same format presented in the support material for Unit 5. The content of each circle should represent the value of the *distance* field of each vertex.

Draw thicker directed arrows to represent the values of the *parent* fields that form the shortest paths tree in every intermediate state. For example, if vertex *v*'s *parent* field is set to *u*, the arrow should point in the direction $u \rightarrow v$.

Draw a new graph corresponding to each intermediate state at the end of each iteration of the outer loop, and include the overall initial and final states.

b) (3 pt.) Instrument the implementation of Bellman-Ford given in the support material in order to print information about the intermediate state of the graph at the end of every iteration of the outer loop. Write a main program that builds the graph given above, and runs Bellman-Ford while showing this additional output. Then verify that your manual trace matches this output.

Upload a file named `q2.zip` containing all files needed to compile and run this program without errors on the Linux CoE machines. The output of your program should match the trace presented in part a).

c) (1 pt.) Is there an alternative way to calculate the shortest paths on the given graph with a lower computational cost? Justify your answer.