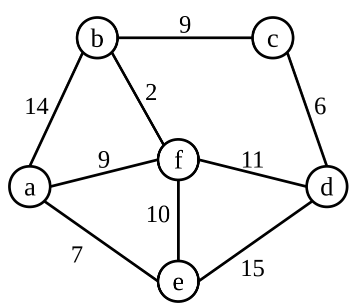
Homework #9

This homework assignment requires submitting only written answers. Upload your answers in a file named `hw9.pdf`.

Question 1 (3 pt.)

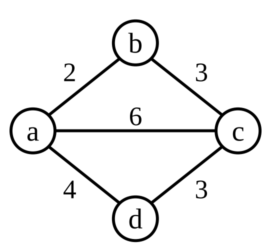
For the weighted, undirected graph shown below, run a trace of the Dijkstra algorithm to calculate the shortest paths relative to vertex *a*. Consider that the vertexes have been added to the graph in the following order: *a, b, c, d, e, f*.

Write the state of the graph right after the initialization loop of the Dijkstra algorithm implementation, and at the end of each iteration of the central *while* loop. Write the value of the *distance* fields associated with each vertex inside of the circle representing them. Use thick arrows with a defined direction to represent the value of the *parent* fields, as done in the traces shown in the support material. For each representation of the graph, write a sentence describing the updates over the previous state.



Question 2 (3 pt.)

For the following weighted, undirected graph, write a trace of the Floyd-Warshall algorithm calculating all-to-all shortest paths. Consider the following order for the vertex indexes: *a, b, c, d*. Each state represented in your trace should include the values of all *dist* and *parent* fields, represented as two individual matrices. You do not need to redraw the graph this time. Include the initial state, as set up by the initialization loop in the Floyd-Warshall implementation given in class. Use the same format shown in the support material for the Floyd-Warshall trace.



Question 3 (2 pt.)

Consider the multithreaded implementation of the computation of Fibonacci numbers shown in class. Draw the computation *dag* (directed, acyclic graph) for the execution of `Fib(4)`. Calculate the work, the span, and the maximum theoretical speedup of this implementation when running on a machine with an unlimited number of processors.

Question 4 (2 pt.)

Run the C++ code given in the support material for the multithreaded producer-consumer model. Replace the declaration of global variable `x` (currently defined as `std::atomic<int> x`) with just `int x`.

a) (1 pt.) What is the difference in the output? Why does it differ?

b) (1 pt.) Run the program with the non-atomic version of `x` several times. Explain the different final values of `x` obtained across multiple executions of the program.