1. Run the strongly-connected components algorithm on this graph. Draw a picture of the DFS forest that results from doing DFS on $G^R$, including pre and post numbers. As usual, when faced with a choice among vertices, pick the one that is alphabetically first. Draw the metagraph of the SCC’s.

2. **Sightseeing.** You are a tourist in a city, and you want to take in as many sights as possible on a single tour. Let’s model this as a directed graph $G = (V, E)$, where the vertices in the graph represent the sights, and the edges represent roads between them (the graph is directed, because some roads may be one-way). For a path $p$ in the graph, define the sightseeing value of $p$ to be the number of distinct vertices in $p$. For a vertex $v \in V$, define the sightseeing value of $v$ to be the maximum sightseeing value of any path starting at $v$. Design and analyze an algorithm to find a vertex of maximum sightseeing value. Your algorithm should run in time $O(|V| + |E|)$.

Hint: use a strongly connected components algorithm.

3. Suppose Dijkstra’s algorithm is run on the following graph, starting at vertex $A$. Illustrate the execution by drawing a table of estimated costs of all vertices at each stage of the algorithm, and by drawing the final least-cost-path tree.

4. **Invasion.** Two armies simultaneously invade a country. Let’s call them the “red army,” and the “blue army.” The red army starts out occupying city $a$, and the blue army starts out occupying city $b$. Both armies fan out simultaneously in all directions, and whichever army arrives at a city first, occupies that city, and blocks the other army from either occupying or transiting through that city. The occupying
army leaves a small occupation force at that city, but the remainder of the army continues to fan out
to all neighboring cities. In case of a tie, the city is reduced to rubble, and is occupied by neither army
—and neither army may transit the city. The army that occupies the most cities wins the war. The
question is: which army wins?

Let’s model this problem as a directed graph with positive edge weights. The nodes in the graph
represent the cities, and the edges represent roads between cities. The weight of an edge \((u, v)\) represents
the amount of time required for either army to travel from city \(u\) to city \(v\). Design and analyze an
efficient algorithm to solve this problem. The input is a directed, weighted graph, along with distinct
nodes \(a\) and \(b\). The output is “red wins,” “blue wins,” or “tie.”

**Observations and hints:** You might think that you could just run Dijkstra twice, once starting at \(a\)
and once starting at \(b\). However, the tie-breaking rule means that this won’t work. Why? You might
want to come up with an example graph that shows why this does not work.

To solve this, you will have to modify Dijkstra’s algorithm. The idea is to associate with each city two
pieces of information: a running estimate for red’s best time to reach that city, a running estimate
for blue’s best time to reach that city, and the minimum of these two estimates. In every step of
the algorithm, we greedily choose one city whose status is “undecided” and move it into the “decided”
category, assigning to either red, blue, or neither, and then update the estimates for the other undecided
cities accordingly.

Fill in the details of the above idea, and try to carefully prove the correctness of it using a loop invariant
similar to what was done in class. In your proof, you should identify where we used the fact that the
edge weights are positive. You can use a priority queue in your algorithm, without worrying about
how that is implemented.

**The following is for honors students only.**

H1. In Exercise 2, you only had to find the vertex with highest sightseeing value. Now suppose you want
to find an actual path. Can you give a reasonable bound on the length of the path? Can you give a
reasonable algorithm? Polynomial time in \(|V|\) and \(|E|\) will do.