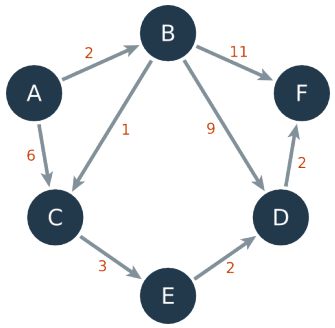
Basic Algorithms — Fall 2020 — Problem Set 3b
Due: Wed, Oct 28, 11am

1. **Dijkstra mechanics.** Run Dijkstra's algorithm on the following graph, starting from vertex A. Whenever there is a choice of vertices, choose the one that is alphabetically first. Show the state of the algorithm (array d, array π, set Q, and set R) just after initialization and at the end of each loop iteration.



2. **Return to Gotham City.** Consider the Gotham City problem from the last problem set. Suppose again that the mayor's claim is not necessarily true. Nevertheless, an emergency arises, and you *must* drive from location s to location t, even though there may not be a legal route from s to t. You want to plan a route that traverses some edges in the graph in the reverse (i.e., illegal) direction. In order to minimize the chance of getting caught by the police, your planned route should minimize the number of illegal edges. The number of legal edges in the route is irrelevant.

   Give a linear-time algorithm that computes a route from s to t that minimizes the number of illegal edges (or reports that no route exists), given the graph G and locations s and t as input.

   Hint: solve this by *reduction* to a standard single-source shortest path problem. Use the fact that if all edge weights are non-negative integers bounded by a constant, then such shortest path problems can be solved in linear time.

3. **Alternating paths, redux.** Consider again the Alternating Paths problem from the last problem set. In this problem, you are given a graph $G = (V, E)$ (in adjacency list representation), a color vector c, and a positive integer k. Also assume that each edge has a *weight*, which is a non-negative number. You are to determine if the graph contains a k-alternating path, and if so, to find a k-alternating path of *minimum weight*. Your algorithm should run in time $O((k|V| + k|E|) \cdot \log(k|V|))$.

   Hint: solve this by *reduction* to a standard single-source shortest path problem.

4. **Patriotic paths.** Let $G = (V, E)$ be a directed graph. Each edge has a *weight*, which is a non-negative number, and a color, which is either *red*, *white*, or *blue*. Let $s, t \in V$. A path from s to t is called *patriotic* if it starts with a sequence of one or more *red* edges, followed by a sequence of one or more *white* edges, and ends with a sequence of one or more *blue* edges.

   Your task is to design an efficient algorithm that finds a patriotic path from s to t of minimum weight (or reports that there is no patriotic path). Your algorithm should run in time $O((|V| + |E|) \log |V|)$.

   Hint: solve this by *reduction* to a standard single-source shortest path problem.

5. **Outbreak!** Buried deep in the heart of the CDC (Center for Disease Control) is a laboratory. In this laboratory there are many animals housed in cages. Each cage contains one animal.

   One day, at 8am, because of a terrible mixup with the animal feed, some of the animals are exposed to and become infected with a deadly virus.

   An animal, once infected with the virus, will eventually become contagious and later die. Once an animal becomes contagious, it will immediately infect the animals in the cages adjacent to its own cage. More specifically, an animal v will become contagious C_v time units after it becomes infected

(the amount of time depends on the animal), and it will die D_v time units after it becomes infected (you may assume that $D_v \geq C_v$).

The question is: which animals will eventually die, and at what time will they drop dead?

You are to design an efficient algorithm to answer this question. Assume the animals are numbered $1, \ldots, n$. As inputs, you are given, for each animal $v \in \{1, \ldots, n\}$:

- the values C_v and D_v (which are positive numbers);
- a list of all animals that are in cages adjacent to v's cage.

You are also given the set $S \subseteq \{1, \ldots, n\}$ of initially infected animals.

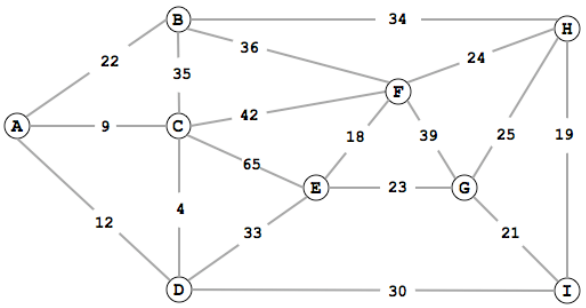
You should also assume that each cage is adjacent to at most 8 other cages.

Your algorithm should run in time $O(n \log n)$.

Hint: solve this by *reduction* to a standard single-source shortest path problem on a *directed* graph.

Disclaimer: no animals were harmed in the production of this exercise.

6. **MST mechanics.** Consider the following weighted, undirected graph:



   (a) Assume we run Prim's MST algorithm starting at vertex A. List the edges that get added to the tree in the order in which the algorithm adds them.

   (b) Now do the same thing for Kruskal's algorithm.

7. **Invasion *[Food for thought (will not be graded)]:*** 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 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. In every step of the algorithm, we greedily choose one city whose status is "undecided" and move it into the "decided" category, assigning it 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 for Dijkstra. 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.