Exercise 6.5. Let $G = (V, E)$ be a directed graph. We say two vertices $s, t \in V$ are *half-connected* if either s can reach t or t can reach s. We say that the graph G is *half-connected* if every pair of vertices is half-connected. We say that two vertices s and t are *strictly half-connected* if either s can reach t, or t can reach s, but not both. We say that G is *strictly half-connected* if every pair of vertices is strictly half-connected.

Exercise 6.5.1. Consider the problem of deciding if G is half-connected. Design and analyze an algorithm for this problem.

Solution. We take as input the adjacency matrix representation of G, $A[1..n][1..n]$.

```
half-connected($A[1..n][1..n]$):
1. For $k \in [1..n]$:
   A. For $i \in [1..n]$:
      1. For $j \in [1..n]$:
         a. Set $A[i][j] := A[i][j]$ or $(A[i][k]$ and $A[k][j])$
2. For each $(i, j)$ with $i \neq j$,
   A. If $A[i][j] = 0$ and $A[j][i] = 0$:
      1. return **False**
3. return **True**
```

This algorithm iterates over the vertices of G and updates A by using k as an "intermediary" vertex. After n iterations of the outer loop, $A[i][j]$ will be 1 if vertex i can reach vertex j. Next, we check the distinct pairs i, j to confirm that one of the two vertices are able to reach each other. If neither of them are, the algorithm terminates by returning **False**. If it gets through step 2 without terminating, then all pairs of vertices are half-connected and thus G is half-connected.

This algorithm executes in $\mathcal{O}(n^3)$ time. □

Exercise 6.5.2. Consider the problem of deciding if G is strictly half-connected. Design and analyze an algorithm for this problem.

Solution. Again, we take as input the adjacency matrix representation of G, $A[1..n][1..n]$.

```
strictly-half-connected($A[1..n][1..n]$):
1. For $k \in [1..n]$:
   A. For $i \in [1..n]$:
      1. For $j \in [1..n]$:
         a. Set $A[i][j] := A[i][j]$ or $(A[i][k]$ and $A[k][j])$
2. For each $(i, j)$ with $i \neq j$,
   A. If $A[i][j] = A[j][i]$:
      1. return False
3. return True
```

This algorithm is functionally identical to the previous one, but changes the check in step 2.A to confirm that $A[i][j] \neq A[j][i]$, returning false if they are equal. This is because $A[i][j] = A[j][i]$ implies that vertices i, j are either unable to reach each other or are both able to reach each other, and both of these outcomes violate strict half-connectedness. Thus getting to step 3 guarantees the graph is strictly half-connected.

This algorithm executes in $\mathcal{O}(n^3)$ time. □

Exercise 7.1. Each of the following problems describes a graph problem over an unweighted directed graph $G = (V, E)$ with m edges and n vertices. Each problem can be solved by constructing an auxiliary graph and calling a shortest path algorithm from this chapter as a black box. For each problem, (a) design an auxiliary graph and shortest path problem, (b) indicate which algorithm to apply and how, and (c) analyze the running time. No proof of correctness is required.

Exercise 7.1.3. For two vertices s and t, compute the length of the shortest walk from s to t where the number of edges is either a multiple of 2 or a multiple of 3.

Solution. (a) For our auxiliary graph $\hat{G} = (\hat{V}, \hat{E})$, first set $\hat{V} = V \times \{0, 1, 2, 3, 4, 5\}$. The vertex $(v, r) \in \hat{V}$ will represent being at vertex v after ℓ steps, where $r = \ell \mod 6$.

Next, for every edge $(u \to w) \in E$ and for every remainder $r \in \{0, \dots, 5\}$, add an edge $(u, r) \to (w, (r + 1) \bmod 6)$ to $\hat{E}$. Then each new edge will have an effective 'cost' of 1 in this auxiliary graph.

(b) two-or-three(s, t):

```
1. Run BFS($\hat{G}$, $(s, 0)$)
2. Find the smallest dist$((s, 0), (t, r))$ for $r \in \{0, 2, 3, 4\}$.
3. If $\exists r \in \{0, 2, 3, 4\}$ such that dist$((s, 0), (t, r)) \neq \infty$:
    A. Return dist$((s, 0), (t, r))$
4. Else: return $\infty$
```

(c) We know the auxiliary graph contains $6n$ vertices and $\leq 6m$ edges. Thus running BFS takes $\mathcal{O}(6m+6n) = \mathcal{O}(m + n)$ time.

Thus our final running time is $\mathcal{O}(m + n)$.

□

Exercise 7.1.4. For two vertices s and t, compute the length of the shortest walk from s to t where the number of edges is of the form $5x + 8y$ for $x, y \in \mathbb{Z}_{\geq 0}$

Solution. (a) For our auxiliary graph $\overline{G} = (\overline{V}, \overline{E})$, first set $\overline{V} = V \times \{0,1,2,3,4\} \times \{0,1,2,3,4,5,6,7\}$. The vertex $(v, r_1, r_2) \in \overline{V}$ will represent being at vertex v after ℓ steps, where $r_1 = \ell \mod 5$ and $r_2 = \ell \mod 8$.

For each edge $(u \to w) \in E$ and for each (r_5, r_8), add the edge $(u, r_5, r_8) \to (w, (r_5 + 1) \bmod 5, (r_8 + 1) \bmod 8)$ to $\overline{E}$.

(b) `five-eight`(s,t):

```
1. Run BFS$(s, 0, 0)$.
2. Find the smallest $d = \text{dist}((s,0,0),(t,r_5,r_8))$ such that $d = 5x + 8y$ for some $x, y \in \mathbb{N}$.
3. If such an $(r_5, r_8)$ exists, return $\text{dist}((s,0,0),(t,r_5,r_8))$.
4. Otherwise, return $\infty$ (no valid walk exists).
```

(c) We know the auxiliary graph contains $(5 \cdot 8)n = 40n$ vertices and $\leq (5 \cdot 8)m = 40m$ edges. Thus running BFS takes $\mathcal{O}(40m + 40n) = \mathcal{O}(m + n)$ time.

Thus our final running time is $\mathcal{O}(m + n)$.

□

Exercise 7.1.5. Suppose G is a patriotic directed graph where all the edges are colored either red, white, or blue. An *American walk* is a walk where the edges alternate in color in the order of the American dream: red, white, blue, red, white, blue... Here the first edge could be any color and then the colors have to cycle through the American dream thereafter. Compute the length of the shortest American walk from s to t.

Solution. (a) Create an auxiliary graph $G' = (V', E')$.

For each original vertex $v \in V$ and each color $c \in \{\mathrm{R}, \mathrm{W}, \mathrm{B}\}$, create a state $(v, c) \in G'$ to denote that we are at vertex v after an edge of color c. Create an extra starting node and call it $(s, null)$ to denote not having crossed an edge yet.

Add an edge to E' from $(s, null) \to (u, c) \iff \exists$ some c-colored edge $(s, u) \in E$.

Add an edge to E' from $(v, \mathrm{R}) \to (u, \mathrm{W}) \iff \exists$ a red edge $(v, u) \in E$. Repeat for white $\to$ blue, and blue $\to$ red.

(b) american(s, t):

```
1. Run BFS($G'$, $(s, null)$)
2. Return $\min_{c \in \{\mathrm{R},\mathrm{G},\mathrm{B}\}} \{dist((s, null), (t, c))\}$
```

(c) The auxiliary graph has $3n + 1$ vertices and $\mathcal{O}(m)$ edges, so BFS runs in $\mathcal{O}(m + (3n + 1))$ time.

Thus our final time complexity is $\mathcal{O}(m + n)$

□

Exercise 7.1.6. Suppose G is again a patriotic directed graph where all the edges are colored either red, white, or blue. An *un-American walk* is a walk that never cycles through the American dream; that is, a walk where no three consecutive edges in the walk have colors that form any of the following three sequences: (red, white, blue), (white, blue, red), or (blue, red, white). Compute the length of the shortest un-American walk from s to t.

Solution. (a) Create an auxiliary graph $G' = (V', E')$.

For each vertex $v \in V$ and each pair $(c_1, c_2) \in \{\text{R,W,B}, \mathit{null}\} \times \{\text{R,W,B}, \mathit{null}\}$, add $(v, c_1, c_2) \in V'$. Add an extra starting node $(s, \mathit{null}, \mathit{null})$.

Add an edge to E' from $(s, \mathit{null}, \mathit{null})$ to (u, null, c) if $(s, u) \in E$ has color c.

Add an edge to E' from (v, c_1, c_2) to (u, c_2, c_3) iff $(v, u) \in E$ has color c_3, and the triple $(c_1, c_2, c_3) \notin \{(\text{R, W, B}), (\text{W, B, R}), (\text{B, R, W})\}$.

(b) `unamerican(`G', G`):`

```
1. Run BFS($G'$, $(s, \mathit{null}, \mathit{null})$)
2. Return $\min_{c_1, c_2 \in \{\text{R,W,B},\mathit{null}\}} \{\text{dist}((s, \mathit{null}, \mathit{null}), (t, c_1, c_2))\}$
```

(c) In G', there are at most $(3+1)^2 \cdot n = 16n$ states, giving us $\mathcal{O}(n)$ vertices. Each original edge generates at most a constant number of edges in G', whence G' has $\mathcal{O}(m)$ edges. We know BFS takes $\mathcal{O}(m+n)$ time to complete.

Thus the total time to find the shortest un-American walk is $\mathcal{O}(m+n)$.

□

Exercise 7.1.7. Let $s, t \in V$ be two vertices. Consider the following game with two people Alice and Bob. Initially, Alice is on s, and Bob is on t. Each round, Alice and Bob are on two vertices, and they simultaneously take an outgoing edge to another vertex. The goal is to guide Alice and Bob to the same vertex with the minimum number of rounds or declare that it is impossible to have them meet. Compute the minimum number of rounds needed to have Alice and Bob meet at the same vertex.

Solution. (a) Create an auxiliary graph $G' = (V', E')$.

For each pair of vertices $(a, b) \in V \times V$, add a node $(a, b) \in V'$, representing the state in which Alice is at vertex a and Bob is at vertex b.

Add a directed edge from (a, b) to (a', b') in E' iff $(a, a') \in E$ and $(b, b') \in E$, meaning Alice moves from $a \to a'$ and Bob moves from $b \to b'$ in the same round.

We begin at (s, t), and want to end at some (x, x) for $x \in V$.

(b)
```
alice-bob($G'$, $G$):
  1. Run BFS($G'$, $(s, t)$)
  2. Return $\min_{x \in V}\{\text{dist}((s,t),(x,x))\}$
```

(c) The auxiliary graph G' has $|V| \times |V| = n^2$ vertices, and each vertex in G' can have up to m^2 out-edges.

We know BFS takes $\mathcal{O}(m + n)$ time to complete, giving us a final time complexity of $\mathcal{O}(n^2 + m^2)$.

□

Exercise 7.3. Suppose you drive to campus but you are running late for the CS580 midterm, and in this extreme circumstance you are willing to speed up to 25% over the speed limit. However, you don't want to run too high a risk of getting a ticket, so you decide you are willing to speed for a limited number k of streets. The goal is to get to the exam as fast as possible, going 25% over the speed limit for up to k streets, and obeying the speed limit everywhere else.

Formally, the input consists of a directed graph $G = (V, E)$, positive edge weights $\ell : E \to \mathbb{R}_{>0}$, vertices $s, t \in V$, and an integer $k \in \mathbb{N}$. The *k-speeding distance* from s to t is defined as the minimum total length of any (s, t)-walk, except there the k largest edges are decreased by a factor of $4/5$. For example, if $k = 3$, and you have an (s, t)-walk with edge lengths 6, 3, 4, 6, 9, 3, 6, 9, 7, 1, then the total length including the k "speedups" becomes

$$6 + 3 + 4 + 6 + \frac{4}{5} * 9 + 4 + 6 + \frac{4}{5} * 9 + \frac{4}{5} * 7 + 1 = 50.$$

Design and analyze an algorithm computing the k-speeding distance from s to t. Your running time may depend on k. (You may assume $k < n$ since any shortest walk should be a path anyway).

Solution. `k-speeding`$(G = (V, E), \ell : E \to \mathbb{R}_>0, s \in V, t \in V, k \in \mathbb{N})$:

```
1. Let $L$ be an auxiliary graph
2. Let $V_L = (v, i) \mid v \in V, 0 \leq i \leq k$
   A. For each edge $(u, v) \in E$
      1. For each $i \in [0..k]$:
         a. Add the edge $((u, i), (v, i))$ to $E_L$ with length $\ell(u, v)$
         b. If $i < k$:
            1. Add the edge $((u, i), (v, i + 1))$ to $E_L$ with length $\frac{4}{5}\ell(u, v)$
3. Run Dijsktra$(L, \ell, (s, 0))$ and denote the distance from $(s, 0)$ to $(v, i)$ by $\text{dist}(v, i)$
4. Return $\min_{i \in [0..k]} \{\text{dist}(t, i)\}$
```

By using a layered construction, each path from $(s, 0)$ to (t, i) encodes a walk from $s \to t$ in G spending exactly $i \leq k$ speedups, and Dijkstra's algorithm finds the minimal possible total length among those paths. Hence we take the minimum over $i = 0, \ldots, k$ to allow up to k speedups.

The graph L has $(k + 1)|V|$ vertices and up to $(k + 1)|E|$ edges. Thus Dijkstra's algorithm runs in $\mathcal{O}((nk + mk)\log(nk)) = \mathcal{O}((m + n)k\log(nk))$ time.

□