Dennis Kuzminer
CSCI-UA 310-001 PS3b

1. After Initialization:
   $d = [\ 0, \infty, \infty, \infty, \infty, \infty\ ]$
   $\pi = [\ Nil, Nil, Nil, Nil, Nil, Nil\ ]$
   $R = \varnothing$
   $Q = A$
   First Loop:
   $d = [\ 0, 2, 6, \infty, \infty, \infty\ ]$
   $\pi = [\ Nil, A, A, Nil, Nil, Nil\ ]$
   $R = A$
   $Q = B, C$
   Second Loop:
   $d = [\ 0, 2, 3, 11, \infty, 13\ ]$
   $\pi = [\ Nil, A, B, B, Nil, B\ ]$
   $R = A, B$
   $Q = C, D, F$
   Third Loop
   $d = [\ 0, 2, 3, 11, 6, 13\ ]$
   $\pi = [\ Nil, A, B, B, C, B\ ]$
   $R = A, B, C$
   $Q = D, F, E$
   Fourth Loop
   $d = [\ 0, 2, 3, 8, 6, 13\ ]$
   $\pi = [\ Nil, A, B, E, C, B\ ]$
   $R = A, B, C, E$
   $Q = D, F$
   Fifth Loop
   $d = [\ 0, 2, 3, 8, 6, 10\ ]$
   $\pi = [\ Nil, A, B, E, C, D\ ]$
   $R = A, B, C, E, D$
   $Q = F$
   Sixth Loop
   $d = [\ 0, 2, 3, 8, 6, 10\ ]$
   $\pi = [\ Nil, A, B, E, C, D\ ]$
   $R = A, B, C, E, D, F$
   $Q = \varnothing$

Dennis Kuzminer
CSCI-UA 310-001 PS3b

2. Using the fact that if all edge weights are non-negative integers bounded by a constant, then we can implement Dijkstra (with buckets) in time $O(nB + m)$, where B is the bound
Because all legal edges are irrelevant we will loop through the graph and assign all legal edges a value of 0. Then construct a reverse edge, which will be denoted as illegal, and assign all illegal edges a value of 1.

for each v ∈ Successor(u) where u ∈ V:
    wt(u, v) ← 0
    wt(v, u) ← 1
Then, run Dijkstra with buckets algorithm, and return a subset of R (the result set) where the first element is s and the last element is t (R[s, t]) without changing the order of R

This algorithm will run in $O(n + m)$ time, as the bound B = 1.

Dennis Kuzminer
CSCI-UA 310-001 PS3b

3. To see if there is a k-alternating path, we can
    a. Making k copies of the graph ( $G^{(0)}$, $G^{(1)}$... $G^{(k-1)}$, $G^{(k)}$ ) connected by k edges with each edge that connects two graphs alternating between black and white.
    b. Particularly, connect two such copies, i and i+1, iff an edge from i to i+1 (using the same connections as in the original graph) changes the current path color (alternates). This will make the "new" connections to the graph alternate in color.
    c. Run Dijkstra from some node in $G^{(0)}$ to some node in $G^{(k)}$ . Record whether Dijkstra has found a path.
    d. Run Dijkstra again from some node in $G^{(0)}$ to some node in $G^{(k)}$ ; however, the starting node this time should have a different color than the staring node in step c. Record whether Dijkstra has found a path. This is to check whether there is a k-alternating path but starting on the opposite color of $G^{(0)}$ .
    e. If Dijkstra has found a path on either of these, then there exists a k-alternating path.

    In this case, because we are making k (with k times as many edges and vertices) copies of the graph, the time complexity of this algorithm is $O((k|V| + k|E|)log(k|V|))$ . Although, we are running Dijkstra twice that should not impact time complexity.

Dennis Kuzminer
CSCI-UA 310-001 PS3b

4. To see if there is a patriotic path, we can
    a. Assume that if t is not connected by any blue edges that there immediately is no patriotic path to t. This could be implemented in less than linear time.
    b. Else, create 4 copies of the original graph: $G^{(0)}$, $G^{(1)}$, $G^{(2)}$, $G^{(3)}$. $O(4|V| + 4|E|)$
        i. $G^{(0)}$ will only the starting point $s^{(0)}$. We can map all red edges to $G^{(1)}$. If no red edges come out of s, then there is no patriotic path from s to t. This is in place so that we ensure that the pathfinding algorithm traverses at least one red edge.
        ii. $G^{(1)}$ will only contain red edges. We can map all white edges that emit from these red edges to $G^{(2)}$.
        iii. $G^{(2)}$ will only contain white edges. We can map all blue edges that emit from these white edges to $G^{(3)}$. To avoid the pathfinding algorithm returning back to a red edge after moving to white, we can exclude all the red edges when constructing this graph. This motivation applies for step iv.
        iv. $G^{(3)}$ will only contain blue edges.
    c. Run Dijkstra. $O((|V| + |E|)log(|V|))$
    d. After Dijkstra, if there is a path from $s^{(0)}$ to $t^{(3)}$, meaning t must lie in $G^{(3)}$, then we can conclude there is a patriotic path from s to t in the original graph. $O(1)$

We can see that this runs in $O((|V| + |E|)log(|V|))$.

Dennis Kuzminer
CSCI-UA 310-001 PS3b

5. To see when a particular animal dies, we can
    a. Construct a graph connecting all the animals [1...n], with edge weights of $C_v$. (Linear time). More specifically,
        i. We have 2 states: the alive/just infected state and the contagious state.
        ii. We can treat the alive/just infected state as a node $a_i$ and the contagious state $b_i$, where $i \in [1...n]$.
        iii. Then, connect $a_i$ to $b_i$ with an edge weight of $C_v$, showing the transition to contagion.
        iv. Connect $b_i$ to $a_{i+1}$ with an edge of weight 0. This is because once an animal gets contagious, its neighbors will immediately become infected.
        v. We can simplify this graph by just connecting all the animals [1...n], with edge weights of $C_v$.
    b. Create some source node $s$ that connects to all $S \in [1...n]$. Make the edge weights of these 0. (Less or perhaps equal to than linear time)
    c. Run Dijstraka, so we see which animals will die.
        i. Runs in $O((|V| + |E|)log(|V|))$; however, we assume each cage/animal is surrounded by at most 8 other animals/cages. We can restate the runtime: $O((|V| + 8|V|)log(|V|)) \rightarrow O((9|V|)log(|V|)) \rightarrow O(nlog(n))$
    d. Dijkstra tracks the shortest path to some node $v$ using the array $d[v]$. Each entry in this array will be the sum of the weights $(C_v)$. Whatever $d[v]$ is for some node, this will be the time of contagion of this animal.
    e. Death of an animal happens $D_v - C_v$ after $C_{v\,(aggregate)}$, where $C_{v\,(aggregate)} = d[v]$.
    f. Therefore, death of each animal happens $d[v] + (D_v - C_v)$.

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

Dennis Kuzminer
CSCI-UA 310-001 PS3b

6.

a. $\{ AC, CD, AB, DI, IH, IG, GE, EF \}$
b. $\{ CD, AC, EF, IH, IG, AB, GE, DI \}$

Irrelevant trash ---->
4

i. $G^{(0)}$ will only the starting point $s^{(0)}$. We can map all red edges to $G^{(1)}$. If no red edges come out of s, then there is no patriotic path from s to t.

$G^{(0)}$ will only contain red edges and the starting point $s^{(0)}$. Each red edge $u \rightarrow v$ in $G$ maps to an edge $u^{(i)} \rightarrow v^{(i+1)}$. This is in place so that we ensure that the pathfinding algorithm traverses at least one red edge.

ii. $G^{(1)}$ will only contain red edges. We can map all white edges that emit from these red edges to $G^{(2)}$.
$G^{(1)}$ will only contain red edges and the white "fringe" edges of any given nodes connected by a red edge. Each white edge $u \rightarrow v$ in $G$ maps to an edge $u^{(i)} \rightarrow v^{(i+1)}$.

iii. $G^{(2)}$ will only contain white edges and the blue "fringe" edges of any given nodes connected by a white edge. Each blue edge $u \rightarrow v$ in $G$ maps to an edge $u^{(i)} \rightarrow v^{(i+1)}$. To avoid the pathfinding algorithm returning back to a red edge after moving to white, we can exclude all the red edges when constructing this graph.

iv. $G^{(3)}$ will only contain blue edges. To avoid the pathfinding algorithm returning back to a white edge after moving to blue, we can exclude all the white edges when constructing this graph.

2

```
for each v ∈ V:
        d[v] ← ∞
        π[v] ← Nil
d[s] ← 0
R ← ∅
Q ← {s}
while Q not empty:
        remove u from Q with minimal d[u]
        add u to R
        for each v ∈ Successor(u) do:
                if d[u] + wt(u, v) < d[v] then
                        if d[v] = ∞ then add v to Q
                        d[v] ← d[u] + wt(u, v)
                        π[v] ← u
```