Prim's

Here is a video walkthrough of the solutions.

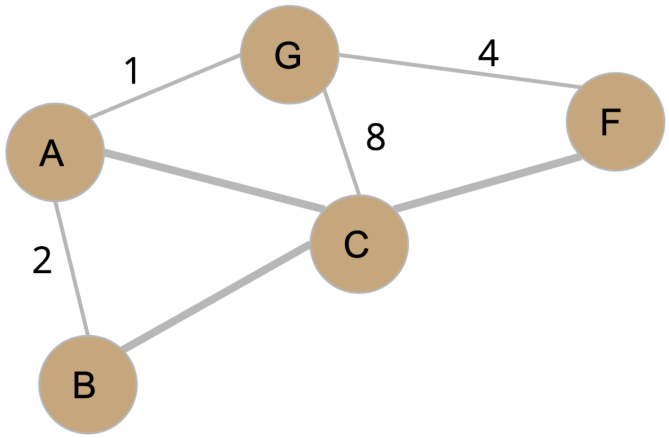
(a) In an arbitrary graph, Prim's can change the priority of a vertex `v` in the priority queue a **maximum** of _____ times and a **minimum** of _____ times. Assume **v** is not the start vertex and the graph is connected and undirected. Give **tight** bounds specific to **v**. Assume we set all priorities to infinity initially.

Solution:
In an arbitrary graph, Prim's can change the priority of a vertex `v` in the priority queue a **maximum** of `degree(v)` times and a **minimum** of 1 times.

Explanation:
Recall that the `degree(v)` of `v` is the neighbors `v` has. It's possible that we *every* neighbor finds a **better** way of getting to `v`, as the second part shows. As such, `degree(v)` is the **maximum** number of change priority operations we can call for `v` since we can only call `changePriority(v, ...)` when are at a neighbor of v.

Next, since all vertices start at priority infinity and the graph is connected, the final priority of each vertex has to change. As such, 1 is the **minimum** number of times. For an example, every time we run Prim's, the first vertex we visit after the start vertex has its priority changed exactly once. Convince yourself why this is the case.

(b) Suppose we run Prim's from A on the graph below.



Fill in the missing edges in the graph to the right so that

1. The priority of C is changed the **maximum** number of times, i.e. the first blank from above.

Solution:
There are many possible solutions to this part. One such solution is setting `AC` to 9, `BC` to 7, and `CF` to 6. To generalize, a valid solution simply

needs to satisfy the following inequalities:

(a) $\texttt{BC} > \texttt{4}$

(b) $\texttt{AC} > \texttt{GC} > \texttt{BC} > \texttt{FC}$

Explanation:
The intuition for why these inequalities must hold is twofold. First, we need C to be the *last* vertex visited. To ensure this is the case, edges AC and BC must be greater than 4, which is accounted for in the check $\texttt{BC} > \texttt{4}$. The reason they must be greater than 4 is to ensure that we visit F before C.

Second, we need to ensure that every edge we consider finds a better way of getting to C. To ensure this is the case, notice that we visit vertices in this order: $\texttt{A} \rightarrow \texttt{G} \rightarrow \texttt{B} \rightarrow \texttt{F}$, assuming that C is the last vertex visited. Accordingly, we must impose the following inequality between the adjacent edges of C: $\texttt{AC} > \texttt{GC} > \texttt{BC} > \texttt{FC}$.

2. The priority of every vertex is changed the **minimum** number of times, i.e. the second blank from above.

Solution:
There are many possible solutions to this part. One such solution is setting AC to 1, BC to 100, and CF to 101. To generalize, a valid solution simply needs AC to be *lighter* than BC, GC, and FC or AC to be *lighter* than AG and AB.

Explanation:
The intuition for the first solution (make AC *lighter* than BC, GC, and FC) is that we want to change the priority of C exactly once. The first edge we consider adjacent to C is AC, so this must be the smallest adjacent edge.

The second, equally valid solution (make AC *lighter* than AG and AB) works because C becomes the first vertex we visit, and we change its priority only once.