Assume the graph $G = (V, E)$ isn't $(f+1)$-connected. Then there exists a set of f processes which if removed would disconnect the graph. Let G' be the graph after these processes are removed. Let p_i be a process in G', and let V_1 (respectively, V_2) be the set of processes in G' which do (respectively, do not) have a path to p_i in G'.

If there exists a path between a process $p_k \in V_2$ and a process $p_j \in V_1$, then the concatenation of that path with the path from p_j to p_i would be a path from p_k to p_i, which is a contradiction because $p_k \in V_2$. Therefore no messages can pass between V_1 and V_2.

Assume there exists a consensus algorithm that works in this case. If all processes in V_1 have input 0, then the algorithm must require them to output 0, because of the possibility (which is impossible for V_1 to refute) that all processes in V_2 also have input 0. Similarly, all processes in V_2 would have to output 1 if all of their inputs are 1. But then the algorithm would fail if all processes in V_1 have input 0 and all processes in V_2 have input 1, so no valid algorithm exists.

Now assume the graph $G = (V, E)$ is $(f+1)$-connected. For each process p_i with input x_i, let m_i be a message saying that p_i's input value is x_i.

Algorithm 1 p_i's Instructions in Round r

if r=1 **then** broadcast m_i.
else broadcast all messages m_j that you just received for the first time.
end if
$n \leftarrow$ the number of processes p_j for which you've received m_j.
if $r > n + 10$ **then** output the minimum value of x_j that you've seen.
end if ▷ The +10 is insurance against off by one errors in my analysis.

First I assume that each process executes the above instructions even after outputting a value, and then I prove that this assumption isn't necessary.

Lemma 1. *Let p_s and p_t be processes that don't crash. If a message m_i reaches p_s, then m_i will also reach p_t.*

Proof. Since G is $(f+1)$-connected and there are at most f crashes, there exists a path P from p_s to p_t in which no processes crash. If m_i doesn't propagate along P after reaching p_s, it's because m_i found a faster way to reach p_t, and no process sends the same message twice. □

Lemma 2. *Let p_t be a process that outputs a value. p_t will receive no messages after doing so.*

Proof. Let S be the set of n processes that p_t knows of when it outputs a value. Assume there exists $p_s \in V - S$ such that there exists a path P from p_s to p_t along which m_s can propagate (i.e. no process in P crashes before it can send m_s at the appropriate time.) Without loss of generality assume that P is non-self-intersecting, and therefore finite. Let p_μ be the last process in P in $V - S$, and let P' be the subpath of P from p_μ to p_t. All but one element of P' is in S, so $|P'| \leq 1 + |S| = 1 + n$. Since P' is traversable when m_s reaches

m_μ, P' is also traversable at the beginning of the algorithm (except perhaps in some cases where there's a shorter path). Therefore m_μ will reach p_t by Round $n+2$ or so, which is a contradiction, because p_t outputs a value around Round $r+10$ and $p_\mu \notin S$. □

Lemma 3. *The algorithm performs the same, regardless of whether a processes continues to run its pseudocode after outputting a value.*

Proof. This follows from Lemma 2, the fact that no process sends a message several rounds after receiving it, and the fact that a process which terminates won't have received any messages for the last few rounds (because of the $+10$). □

Theorem 1. *The algorithm terminates.*

Proof. Each process waits a finite number of rounds before outputting a value or learning about a new process, and there are finitely many processes. □

Theorem 2. *All processes that don't crash output the same value.*

Proof. Let p_i be a process that doesn't crash, and let S be the set of processes that p_i knows about when outputting a value. S is nonempty because $p_i \in S$. By Lemmas 1-3, the value of S doesn't depend on the choice of p_i. All processes output $\min_{p_j \in S} x_j$. □

Theorem 3. *If all processes have the same input value, then no other value is output.*

Proof. Trivial. □