Solution: Minimize Malware Spread

Let's solve the Minimize Malware Spread problem using the Union Find pattern.

We'll cover the following

Statement

You're given a network of n nodes as an $n \times n$ adjacency matrix graph with the i^{th} node directly connected to the j^{th} node if `graph[i][j] == 1`.

A list of nodes, `initial`, is given, which contains nodes initially infected by malware. When two nodes are connected directly and at least one of them is infected by malware, both nodes will be infected by malware. This spread of malware will continue until every node in the connected component of nodes has been infected.

After the infection has stopped spreading, M will represent the final number of nodes in the entire network that have been infected with malware.

Return a node from `initial` such that, when this node is removed from the graph, M is minimized. If multiple nodes can be removed to minimize M, return the node with the smallest index.

> **Note:** If a node was removed from the initial list of infected nodes, it might still be infected later on due to the malware's spread.

Constraints:

- `graph.length == graph[i].length`
- $2 \leq \text{n} \leq 300$
- `graph[i][j]` is 0 or 1.
- `graph[i][j] == graph[j][i]`
- `graph[i][i] == 1`
- $1 \leq$ `initial.length` $\leq n$
- $0 \leq$ `initial[i]` $\leq n - 1$
- All the integers in the `initial` are unique.

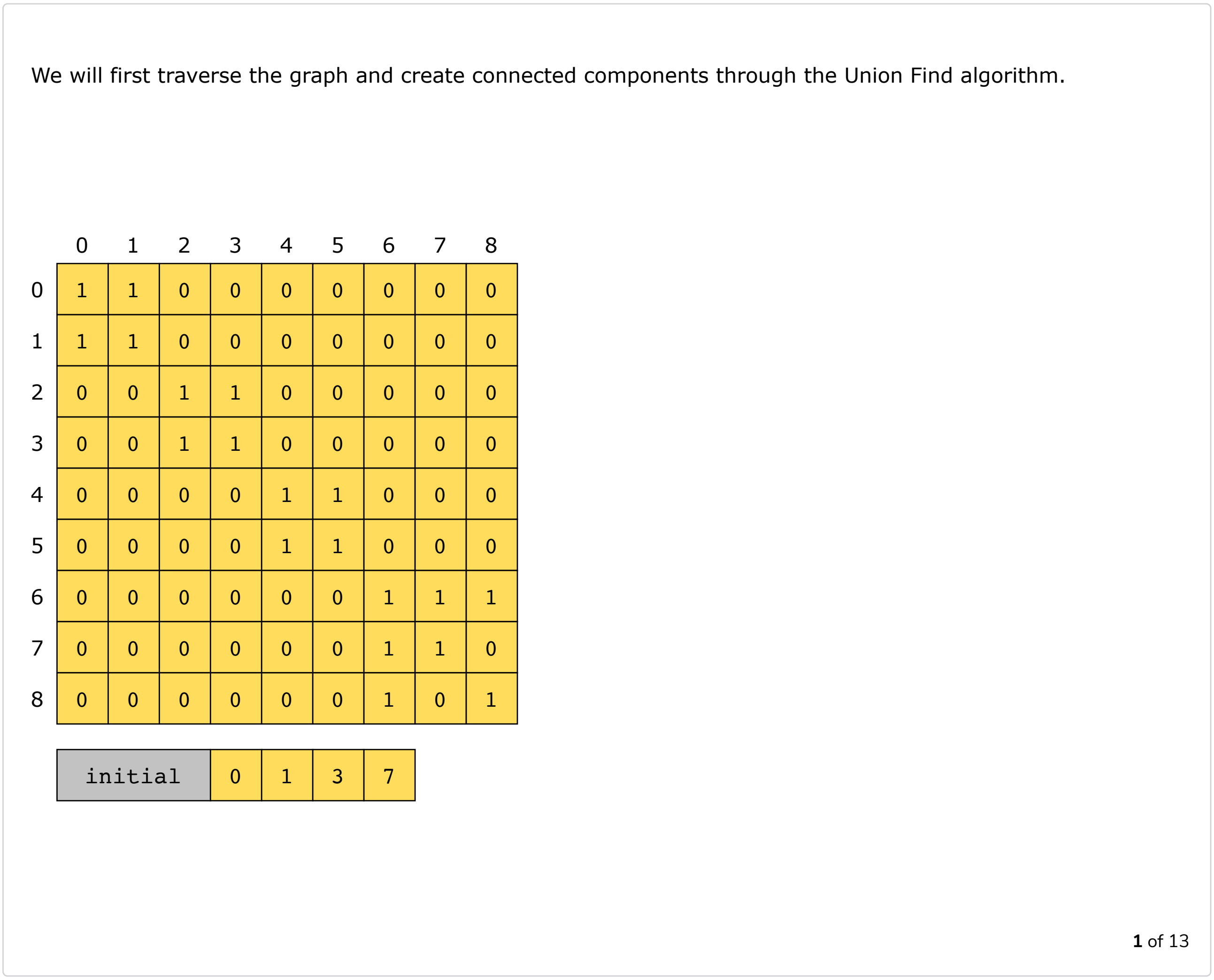
Solution

Since we have to check for infected nodes in each connected component, we need to find all the connected components in a graph. To do this, we use the `UnionFind` data structure that finds all of the graphs's connected components.

The steps of the algorithm are given below:

1. Find all of the graph's connected components using the `UnionFind` data structure.
2. Traverse the `initial` array and store all the connected components with at least one infected node in a hash map, `infected`.
3. Traverse the `initial` array again and calculate the number of infections (from the `infected` hash map) in each connected component containing one or more infected nodes.
    - If a connected component contains more than one infected node, ignore it and move to the next iteration of the loop.
    - Otherwise, calculate the size of the connected component.
4. Find the connected component with the greatest size, containing only one initially infected node from step 3.
5. If there are multiple components that are the same size that would count as the largest connected component, choose the component with the smallest index.

The slides below illustrate how we would like the algorithm to run:



We will first traverse the graph and create connected components through the Union Find algorithm.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

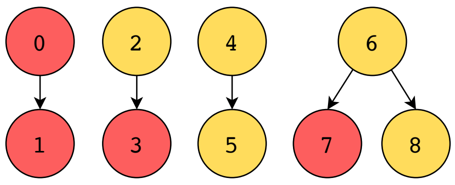
initial	0	1	3	7

1 of 13

This creates a **parent** array containing the root node of each node in the graph and a **rank** array containing the size of each component.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7

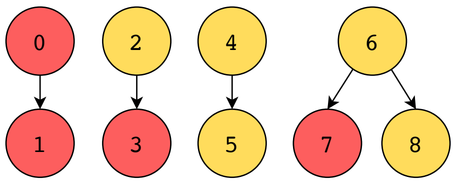


parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

We will now traverse the **initial** array to find out the number of infections in the components that have at least one infected node. We create a hash map, **infected**, to store this information. The key-value pair is of the form: (component root - number of infected nodes in the component).

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7



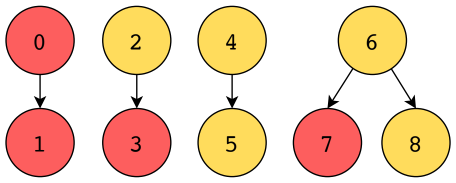
parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

infected

Node **0** has node **1** as its representative. So, we add 1 to the infected nodes of this component and **(1, 1)** is inserted into the hash map.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7


$$\begin{tikzcd}
0 \arrow[d] & 2 \arrow[d] & 4 \arrow[d] & & 6 \arrow[dl] \arrow[dr] & \\
1 & 3 & 5 & 7 & & 8
\end{tikzcd}$$

parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

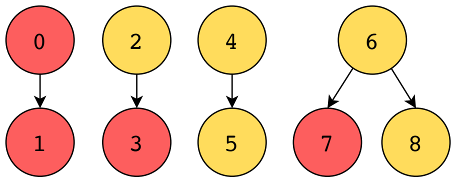
infected
1 : 1

4 of 13

Node **1** has node **1** itself as the component root. So, we add 1 to the infected nodes of this component and the key-value pair **(1, 1)** now becomes **(1, 2)**.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7

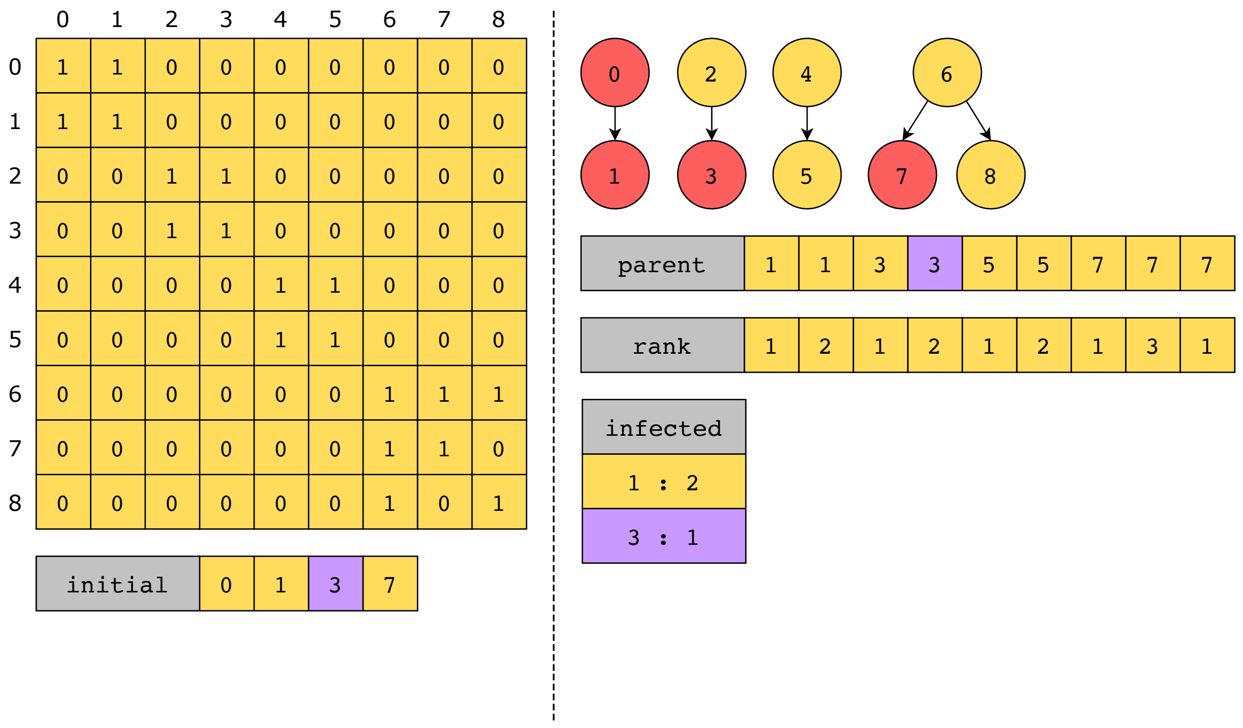

$$\begin{tikzcd}
0 \arrow[d] & 2 \arrow[d] & 4 \arrow[d] & & 6 \arrow[dl] \arrow[dr] & \\
1 & 3 & 5 & 7 & & 8
\end{tikzcd}$$

parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

infected
1 : 2

5 of 13

Node **3** has node **3** itself as the component root. So, we add 1 to the infected nodes of this component and **(3, 1)** is added to the hash map.



	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

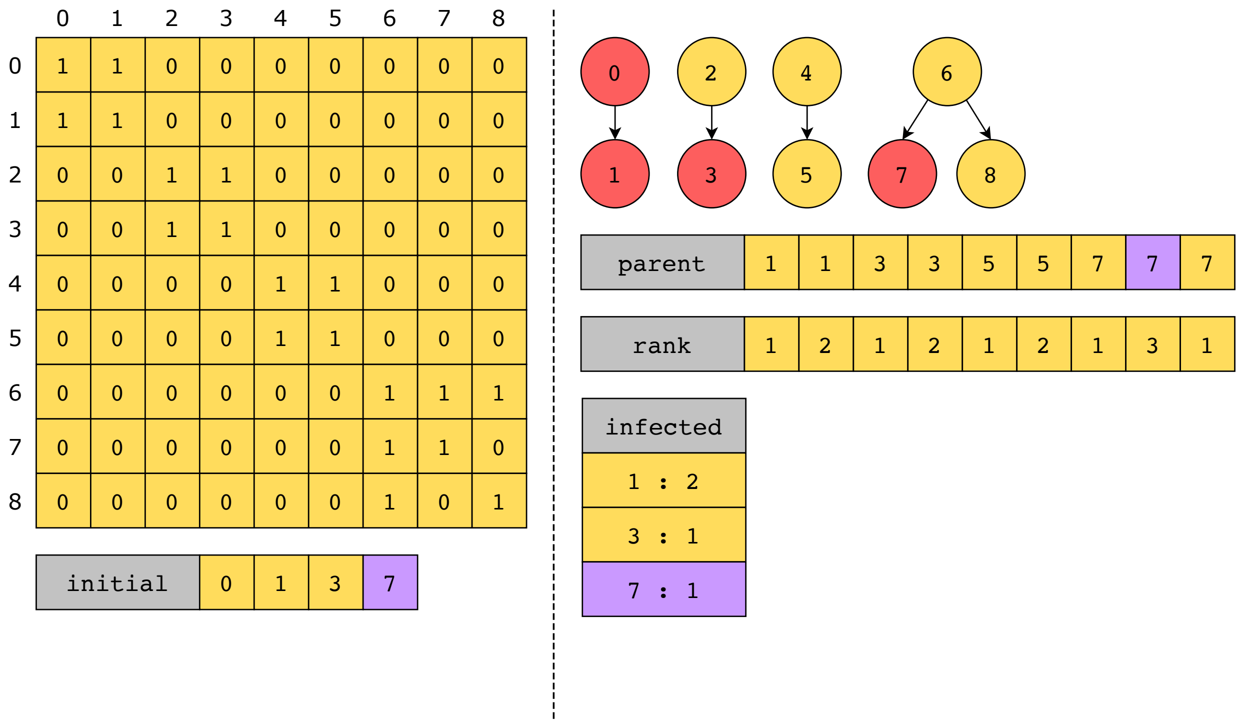
initial	0	1	3	7

$$\begin{tikzcd}
0 \arrow[d] & 2 \arrow[d] & 4 \arrow[d] & & 6 \arrow[dl] \arrow[dr] & \\
1 & 3 & 5 & 7 & & 8
\end{tikzcd}$$

parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

infected
1 : 2
3 : 1

Node **7** has node **7** itself as the component root. So, we add 1 to the infected nodes of this component. **(7, 1)** is added to the hash map.



	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7

$$\begin{tikzcd}
0 \arrow[d] & 2 \arrow[d] & 4 \arrow[d] & & 6 \arrow[dl] \arrow[dr] & \\
1 & 3 & 5 & 7 & & 8
\end{tikzcd}$$

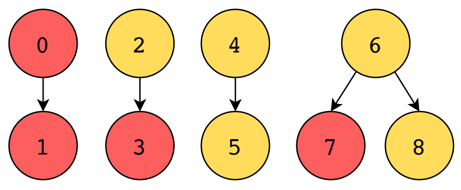
parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

infected
1 : 2
3 : 1
7 : 1

We will now traverse the **initial** array again to find the node that minimizes the final infected nodes. If a component containing the infected node contains more than one infected node, we ignore it. Otherwise, the candidate node to be removed is updated if the component in which it is present has more nodes than the previous component.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7



parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

infected
1 : 2
3 : 1
7 : 1

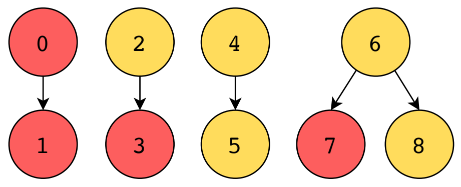
infectionCount	
componentSize	
maximumSize	0
candidateNode	0

8 of 13

The component containing node **0** has two infected nodes in its component and has a rank of **2**. Since the number of infected nodes is greater than 1, we do not check whether the candidate node needs to be changed and proceed to the next element.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7



parent	1	1	3	3	5	5	7	7	7

infected
1 : 2
3 : 1
7 : 1

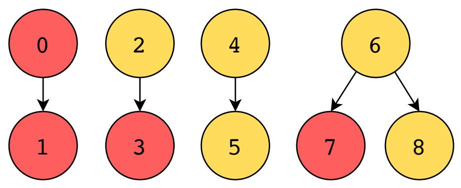
infectionCount	2
componentSize	2
maximumSize	0
candidateNode	0

9 of 13

The component containing node **1** has two infected nodes in its component and has a rank of **2**. Since the number of infected nodes is greater than 1, we do not check whether the candidate node needs to be changed and proceed to the next element.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7



parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

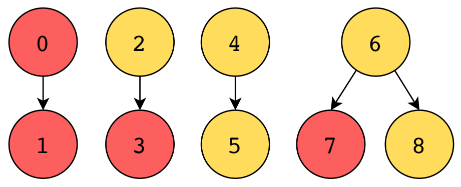
infected
1 : 2
3 : 1
7 : 1

infectionCount	2
componentSize	2
maximumSize	0
candidateNode	0

The component containing node **1** has two infected nodes in its component and has a rank of **2**. Since the number of infected nodes is greater than 1, we do not check whether the candidate node needs to be changed and proceed to the next element.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7



parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

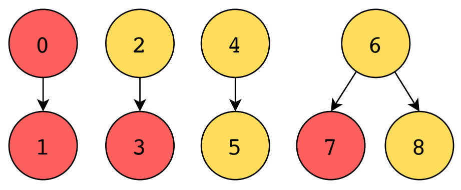
infected
1 : 2
3 : 1
7 : 1

infectionCount	1
componentSize	2
maximumSize	2
candidateNode	0

The component containing node **7** has one infected node in its component and has a rank of **3**. Since the number of infected nodes is equal to 1, we check if the rank of this component is greater than the rank stored in the maximumSize variable. Since it is, we update this variable and the candidate node is now changed to 7.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7



parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

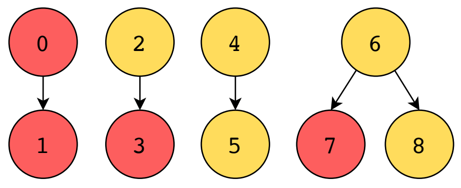
infected
1 : 2
3 : 1
7 : 1

infectionCount	1
componentSize	3
maximumSize	3
candidateNode	7

We have completed the traversal and return **7**, since removing node **7** will result in the minimum number of nodes that are not infected.

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	1	0	1

initial	0	1	3	7



parent	1	1	3	3	5	5	7	7	7
rank	1	2	1	2	1	2	1	3	1

infected
1 : 2
3 : 1
7 : 1

infectionCount	
componentSize	
maximumSize	3
candidateNode	7