CS 361 Data Structures & Algs Lecture 13

Prof. Tom Hayes
University of New Mexico

10-5-2010

Last Time

Spanning Trees

BFS

DFS

Testing Bipartiteness

Today

- Inside BFS
  - Running Time
  - Implementation
  - Degrees & Degree sums
  - Properties of BFS and DFS trees
- Identifying trees in the wild

Breadth-First Search

BFS finds the vertices in the connected component of the start vertex s one "level" at a time.

Level L_i = {vertices whose "distance" to s equals i}

distance(v,w) = number of edges in the shortest path from v to w.

Question: how to implement?

Plan of Attack

We will first talk about a general search framework (GSF), broad enough to include BFS and some other search algorithms.

Then focus on BFS in more detail.

GSF (includes BFS)

Each node has 3 different “states”: U, A, I

Initially: everything marked U (unfound).

Change s to A (active).

```
for (v : A)
    for (w : Adj[v])
        if (w is U) change w to A
    change v to I (inactive)
```

Analysis of GSF

Each node has 3 different "states": U, A, I

Initially: everything marked U (unfound).

Change s to A (active).

```
for (v : A)
    for (w : Adj[v])
        $\boxed{\text{if (w is U) change w to A}}$
    change v to I (inactive)
```

How many times?

Analysis of GSF

Each node has 3 different "states": U, A, I

Initially: everything marked U (unfound).

Change s to A (active).

```
for (v : A)
    for (w : Adj[v])
        $\boxed{\text{if (w is U) change w to A}}$
    change v to I (inactive)
```

How many times?

2 (#edges)

Analysis of GSF

Lemma: Each node is initially U, then A for some time, then I, and never goes back to an earlier label.

Proof. By inspection, there is no line of code which could turn a node marked A or I into U, or I into A.

Corollary: Loop variable v never repeats an earlier value.

Degree of a node

Let v be a vertex in a graph G=(V,E). The degree of v equals the number of neighbors of v, that is, the length of the adjacency list of v.

Answer: What is the sum,

$$\sum_{v \in V} \mathrm{degree}(v)$$

Degree of a node

Let v be a vertex in a graph G=(V,E). The degree of v equals the number of neighbors of v, that is, the length of the adjacency list of v.

Theorem:

$$\sum_{v \in V} \mathrm{degree}(v) = 2|E|$$

Analysis of GSF

Each node has 3 different “states”: U, A, I

Initially: everything marked U (unfound). (#nodes)

Change s to A (active). 1

```
for (v : A)
    for (w : Adj[v])
        if (w is U) change w to A        2 (#edges)
    change v to I (inactive)             (#nodes)
```

Total: O(#nodes + #edges)

Linear Time Algorithms

We say an algorithm runs in linear time if the running time is O(input size).

For a graph in adjacency list format, the input size is Θ(#nodes + #edges).

So, for algorithms where the input is a graph, linear time means $O(|V| + |E|)$.

In particular, BFS is a linear-time algorithm.

From GSF to BFS

BFS wants to explore level by level.

GSF iterates through active set A in any order. BFS wants to explore from all nodes in level i before any in level (i+1).

Insight: BFS also discovers all nodes in level i before any in level (i+1).

Solution: Store A in a queue (FIFO)!
In Java: AbstractQueue.

GSF (includes BFS)

Each node has 3 different “states”: U, A, I

Initially: everything marked U (unfound).

Change s to A (active).

```
for (v : A)
    for (w : Adj[v])
        if (w is U) change w to A
    change v to I (inactive)
```

BFS

```
Initially: A = empty queue.
Initially: U[each vertex] = true
A.add(s)                          (enqueue s)
while (A nonempty)
    v = A.remove()
    for (w : Adj[v])
        if (U[w]) {U[w]=false, A.add(w)}
```

Notes

Originally, state U,A,I, was a single attribute of each node. Helped prevent conflicts (a state marked both U and A).

When A became a Queue, we made U into a boolean array, then dropped I entirely. Care needed to avoid conflicts.

Didn’t have to keep track of current Level explicitly, since Queue keeps the nodes in the right order. Suppose we want to. How can we do it?

Edges in BFS

Theorem: Each time BFS looks at an edge, it is either:

(a) joining a found node (level L_i) to an unfound node (level L_{i+1}). Becomes an edge in the tree, or

(b) joining a found node (level L_i) to an already found node (level L_i or L_{i+1}).

Why only these?

Rephrased:

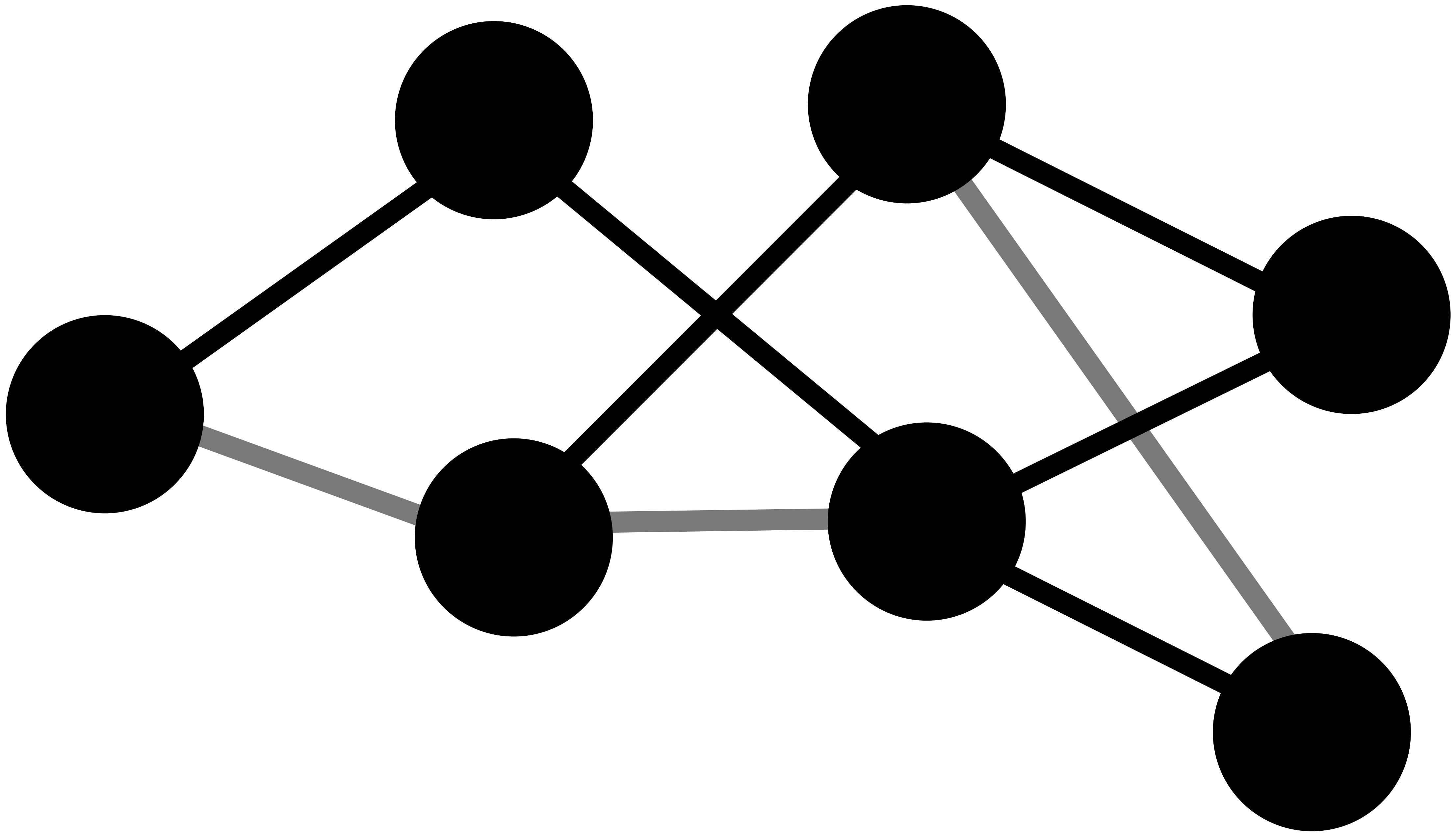
Theorem: If T is a BFS tree for a graph G, then every edge in G either:

(a) is in T, and joins adjacent levels, or

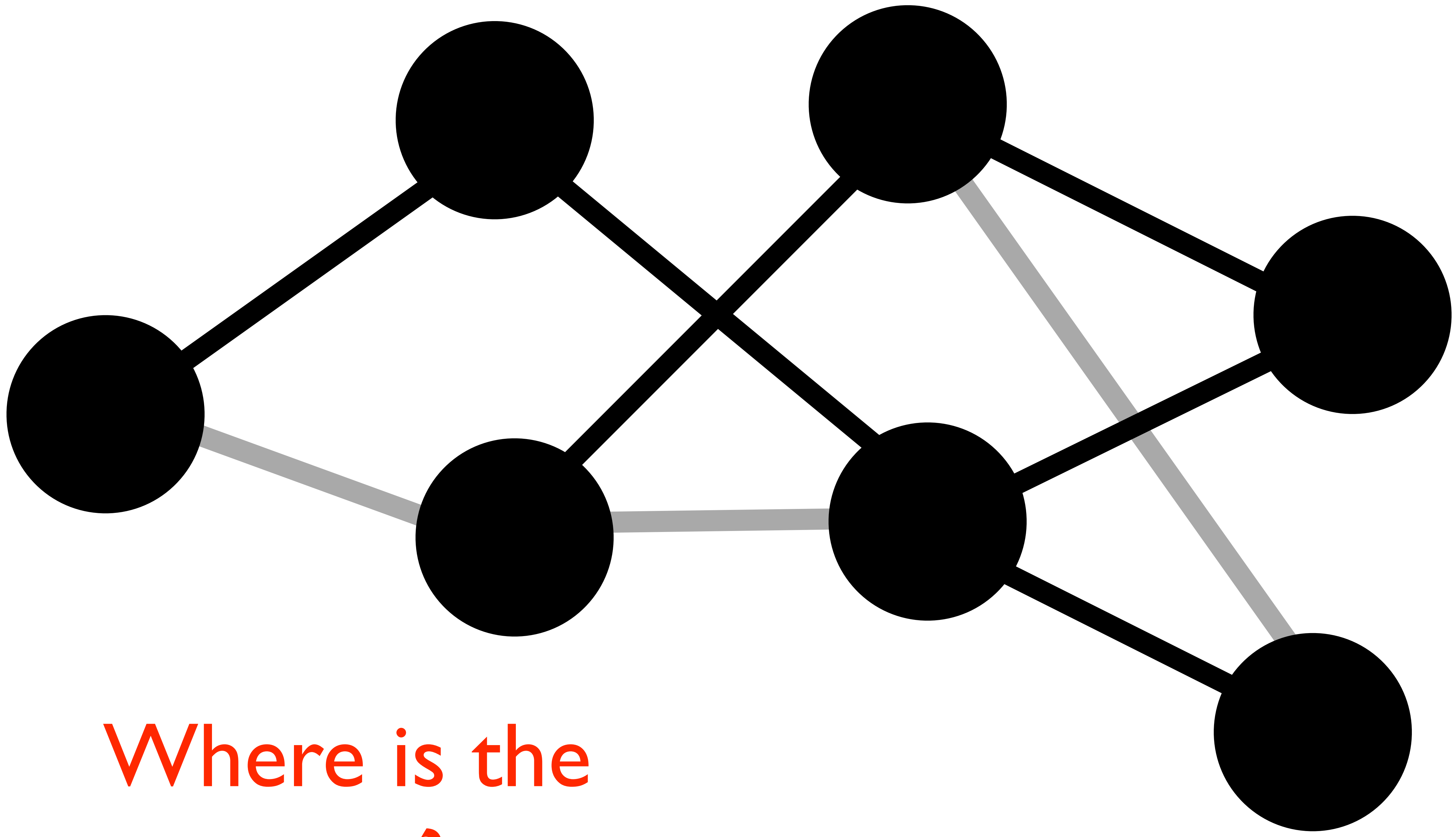
(b) is not in T, and joins nodes in the same or adjacent levels.

Level: equal distance from the root.

Is this a BFS tree?

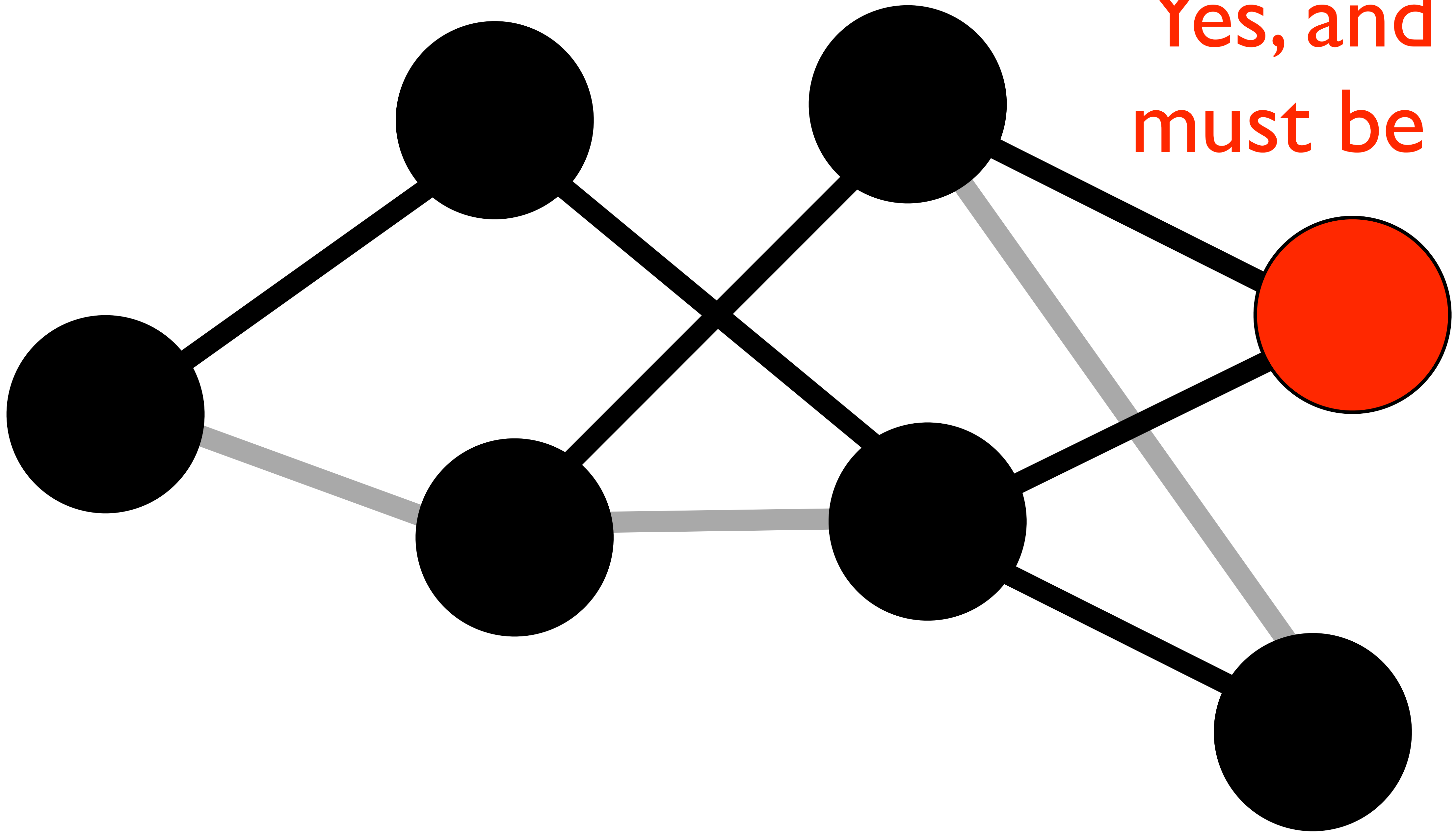


Is this a BFS tree?

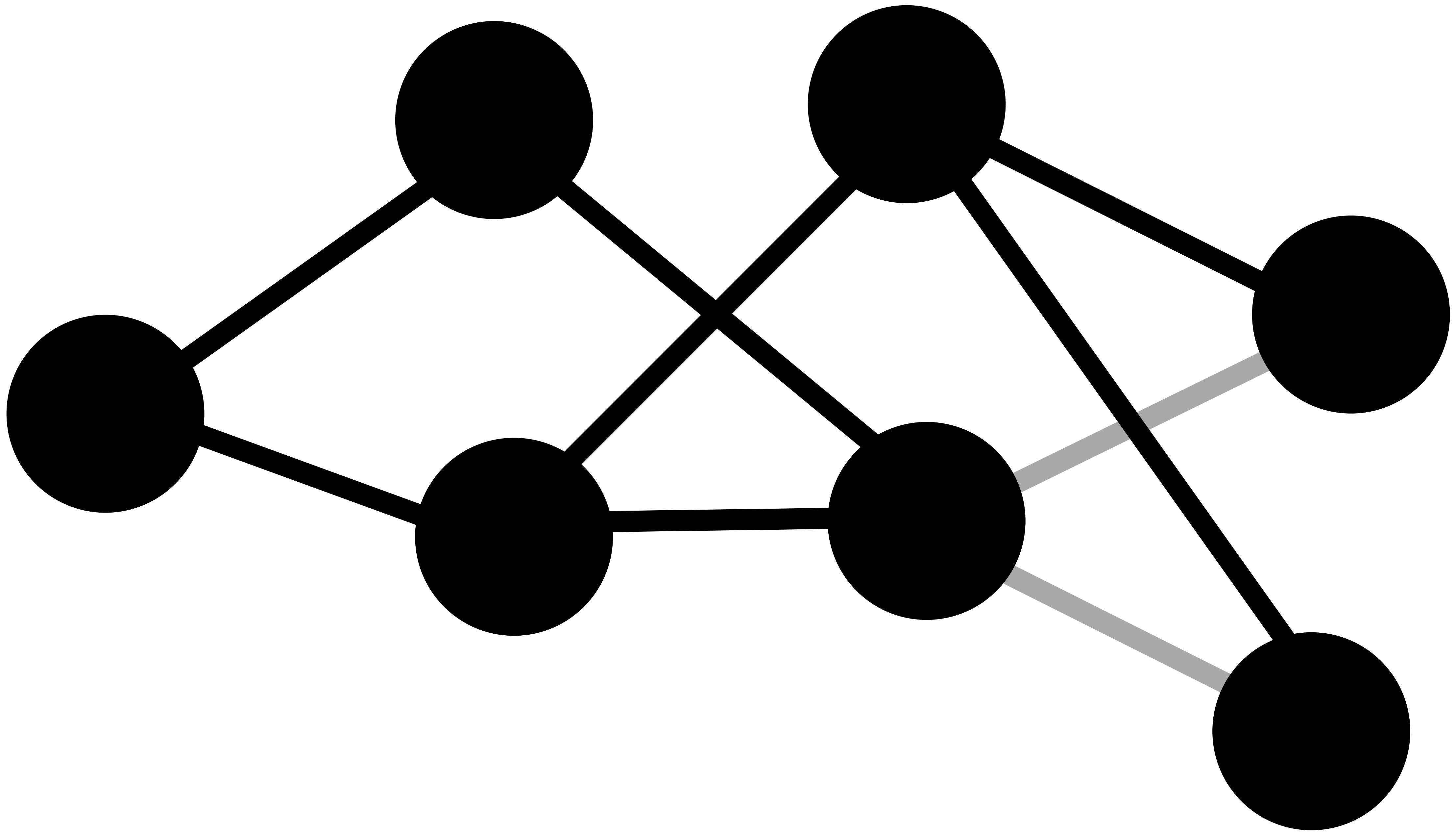


Where is the root?

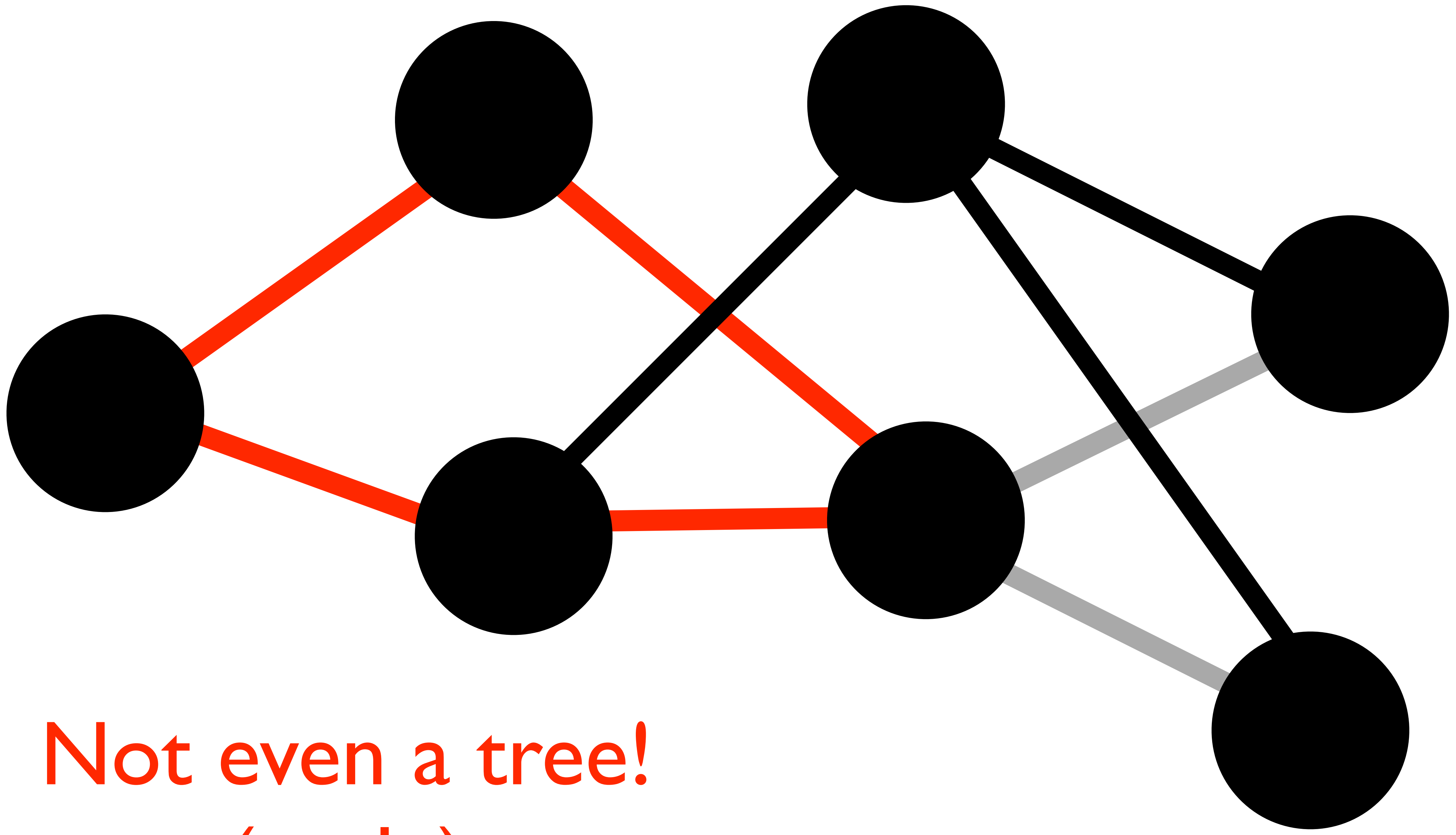
Is this a BFS tree?



Is this a BFS tree? 2

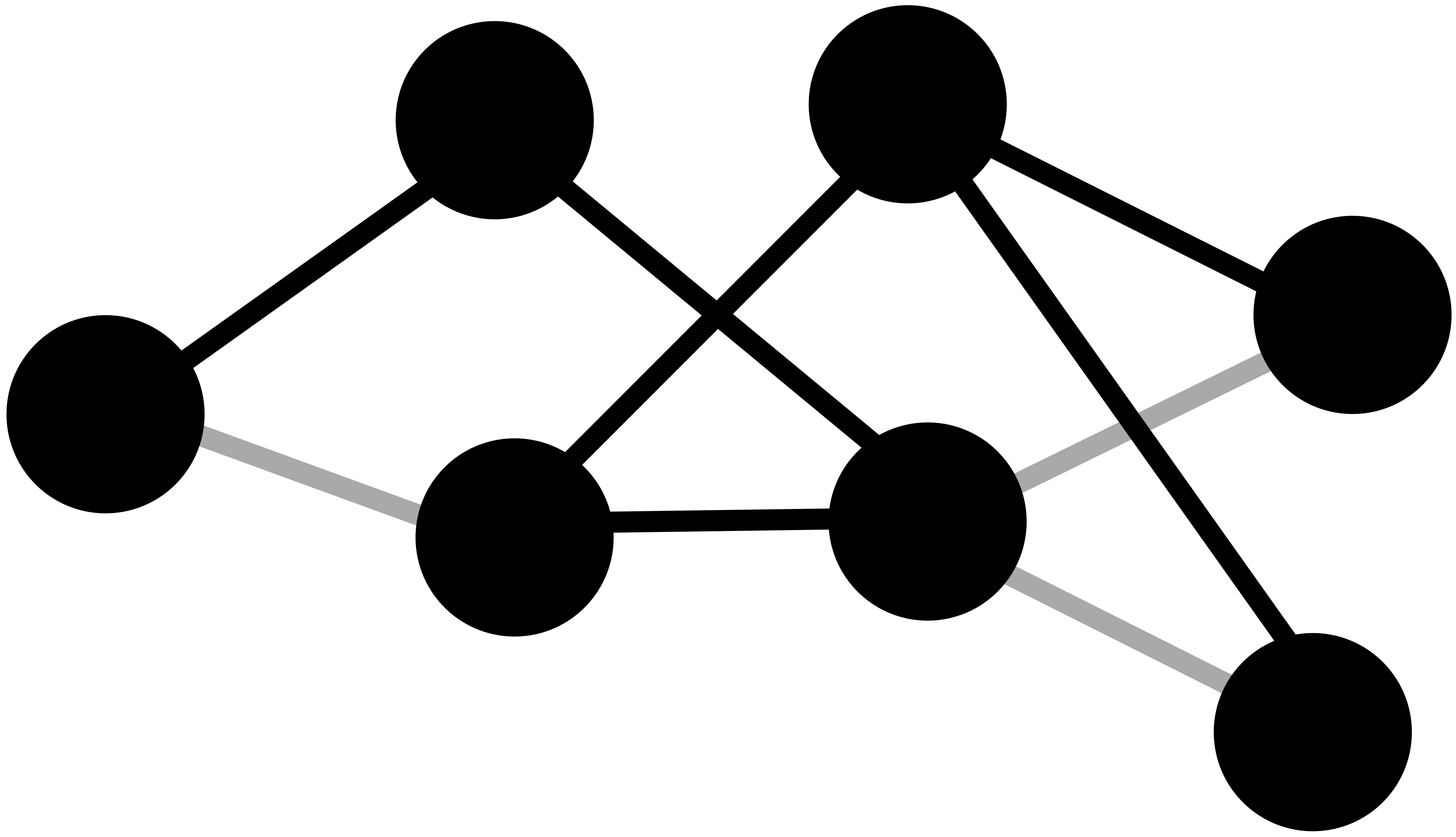


Is this a BFS tree? 2

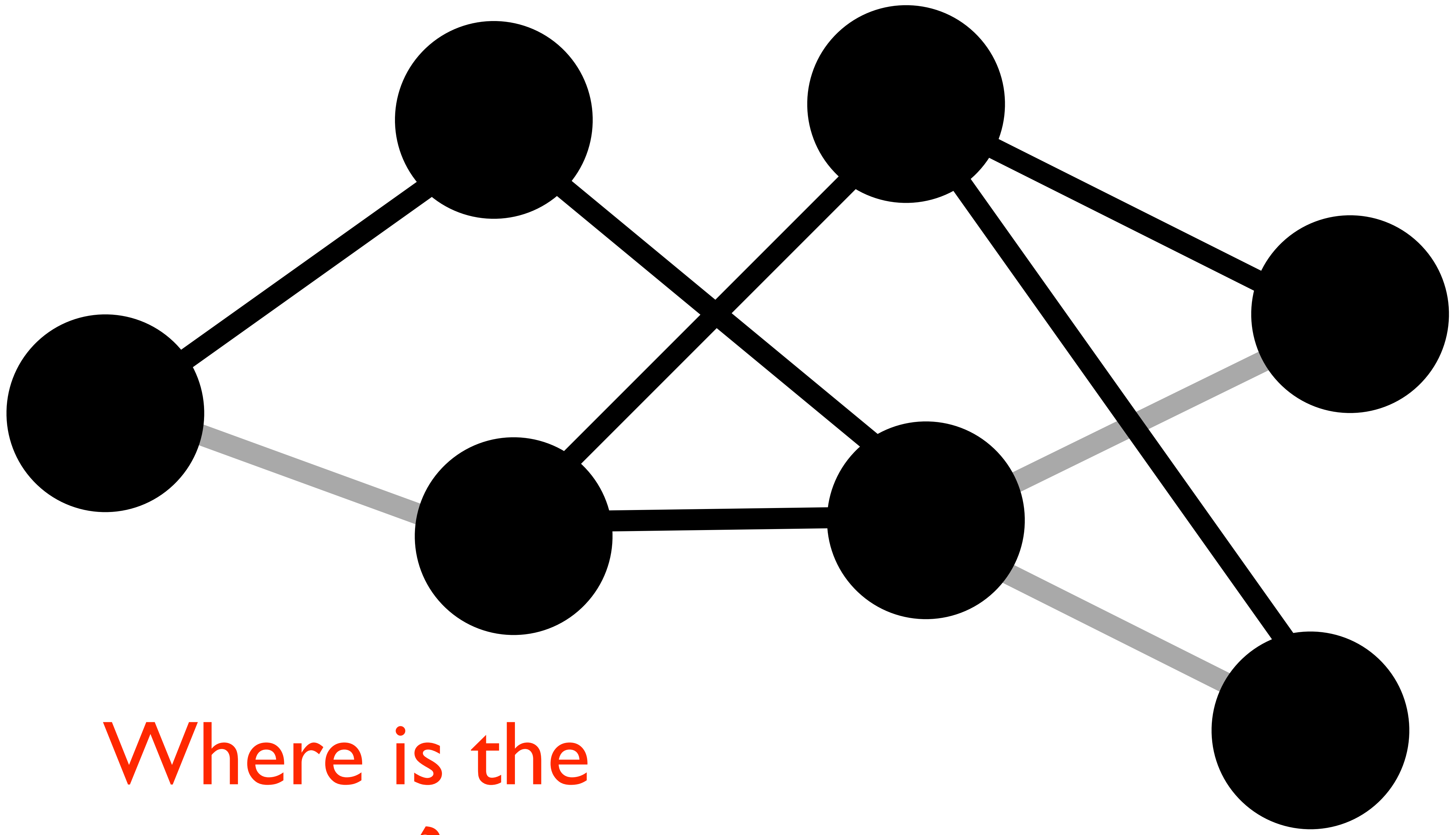


Not even a tree!
(cycle)

Is this a BFS tree? 3

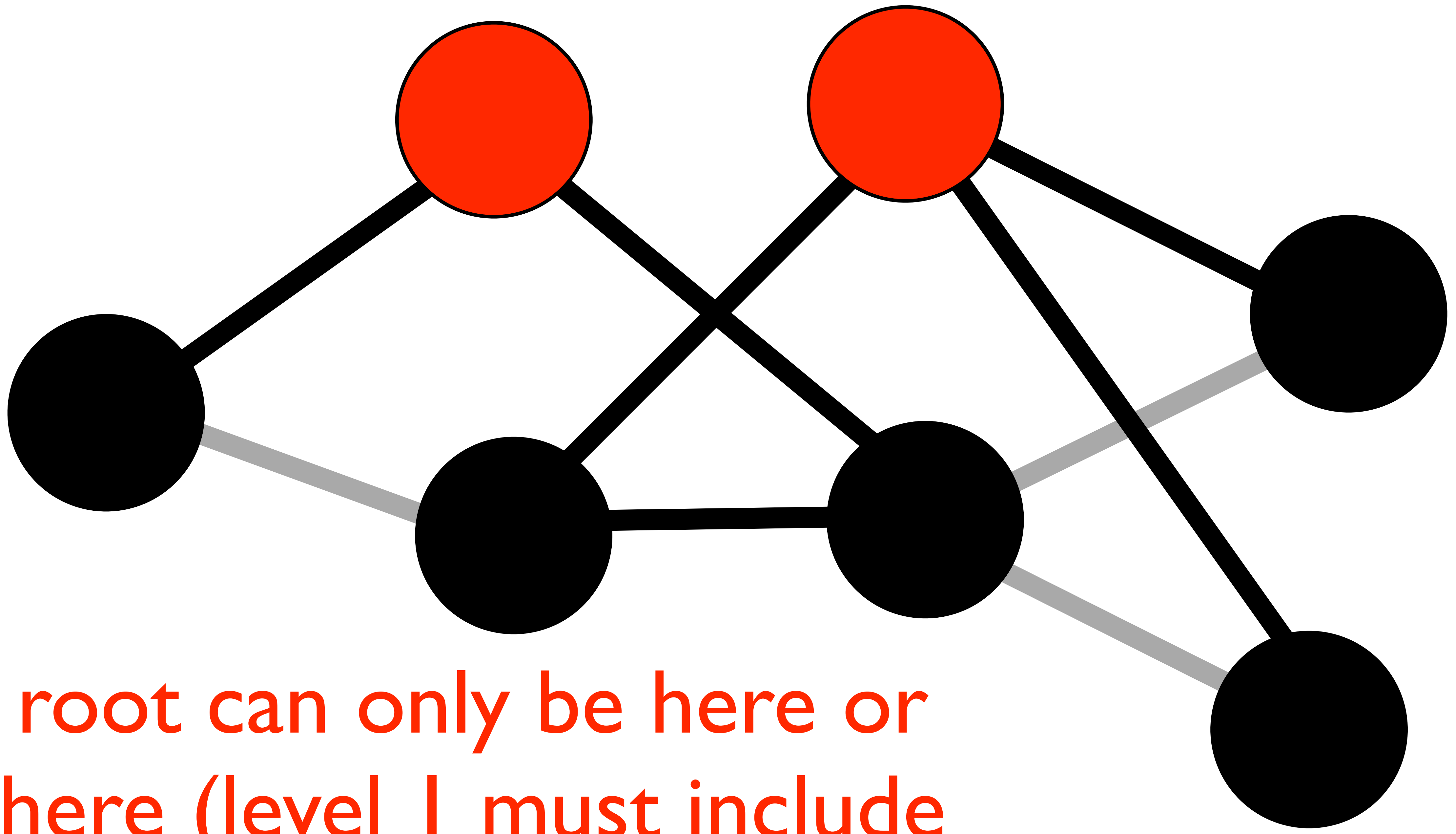


Is this a BFS tree? 3



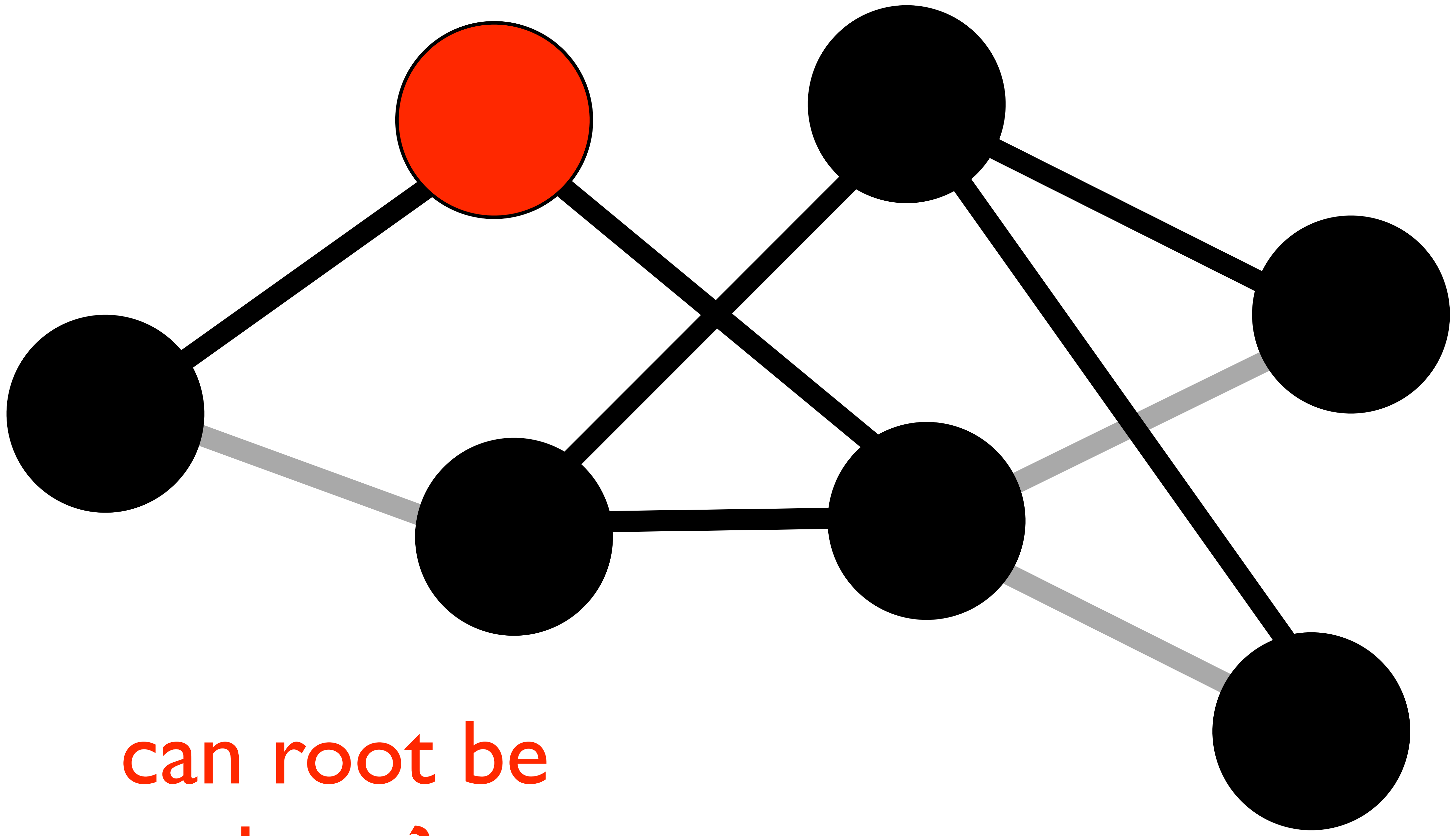
Where is the root?

Is this a BFS tree? 3



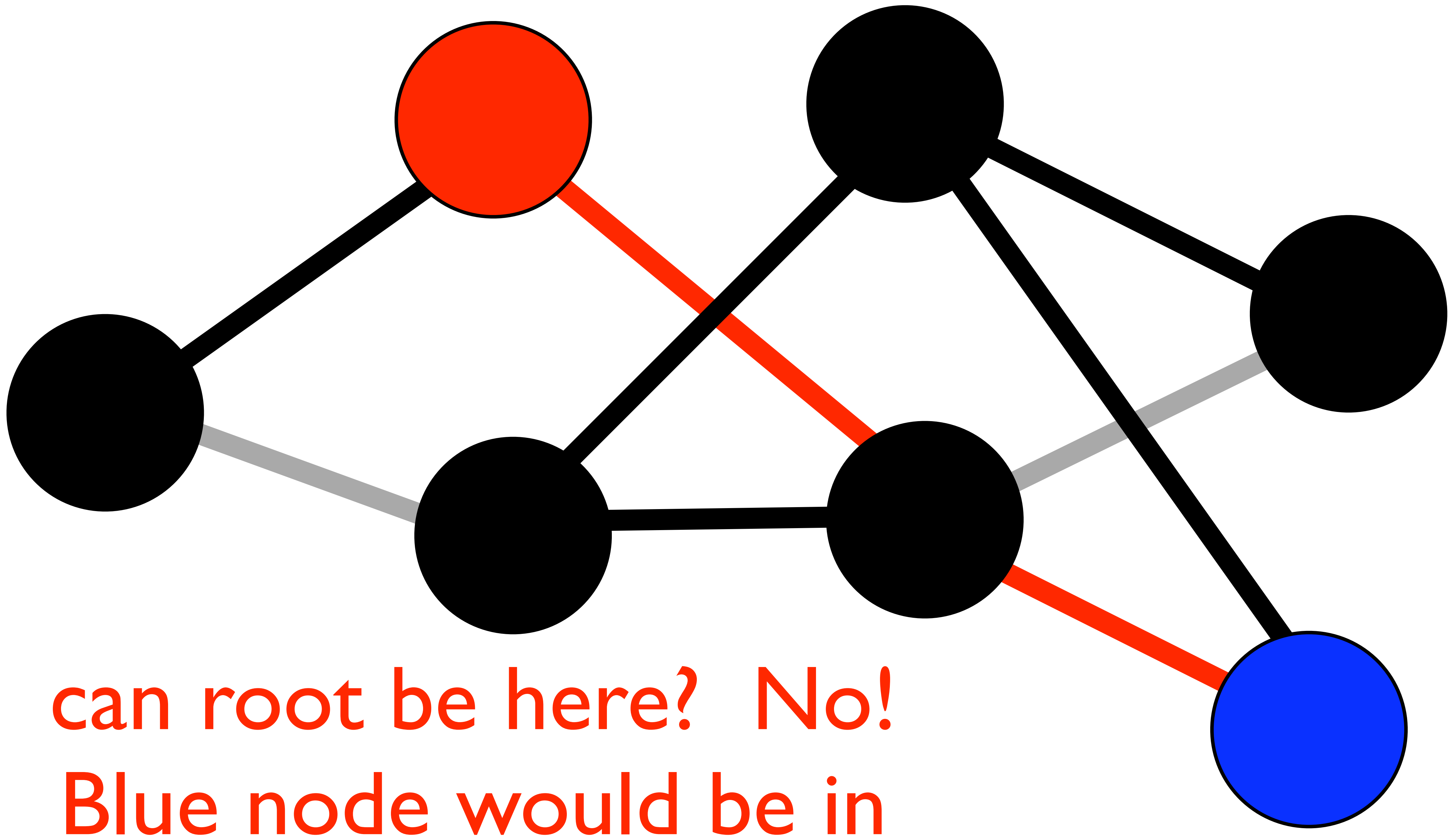
root can only be here or here (level 1 must include all neighbors of root)

Is this a BFS tree? 3



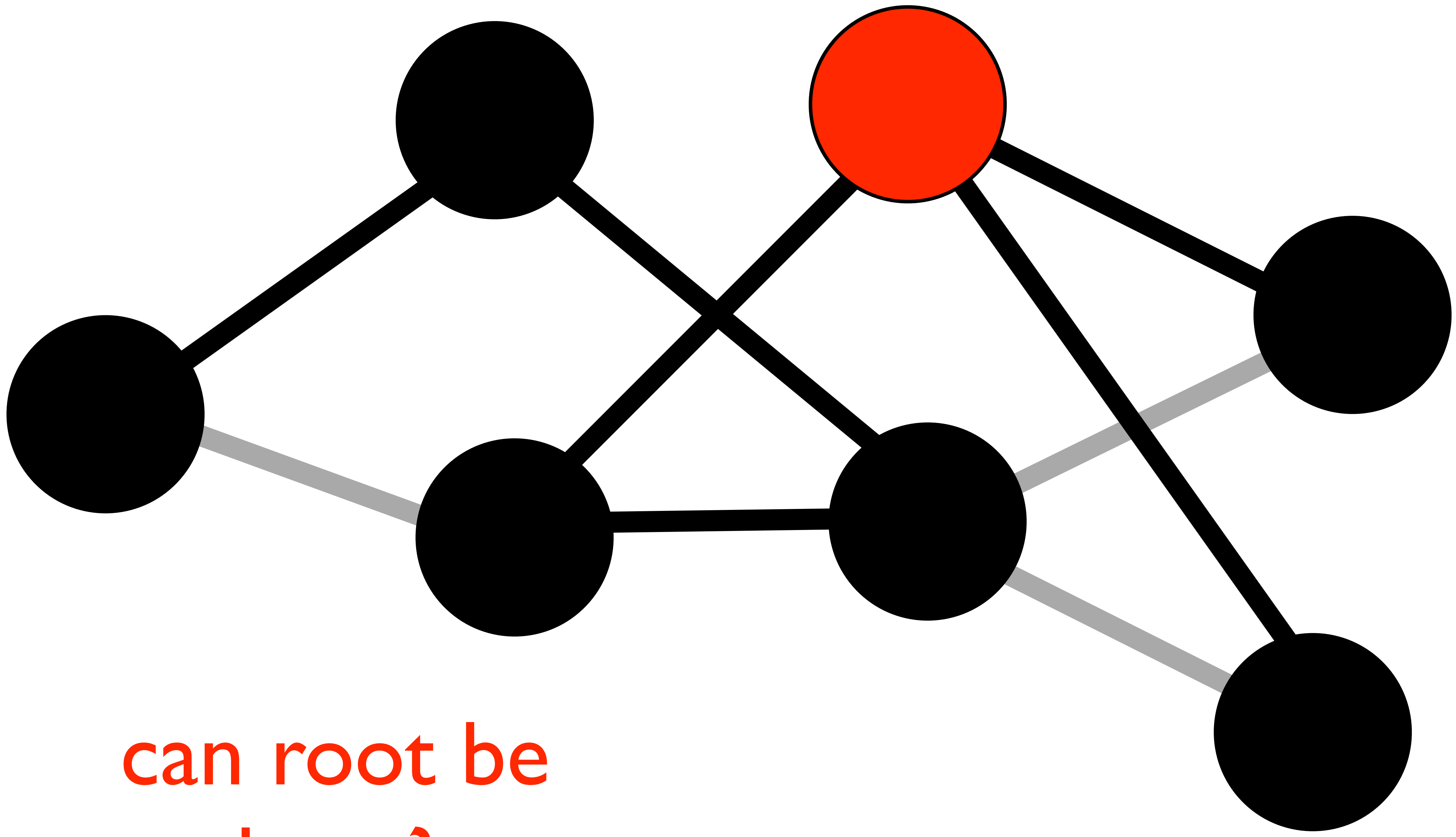
can root be here?

Is this a BFS tree? 3



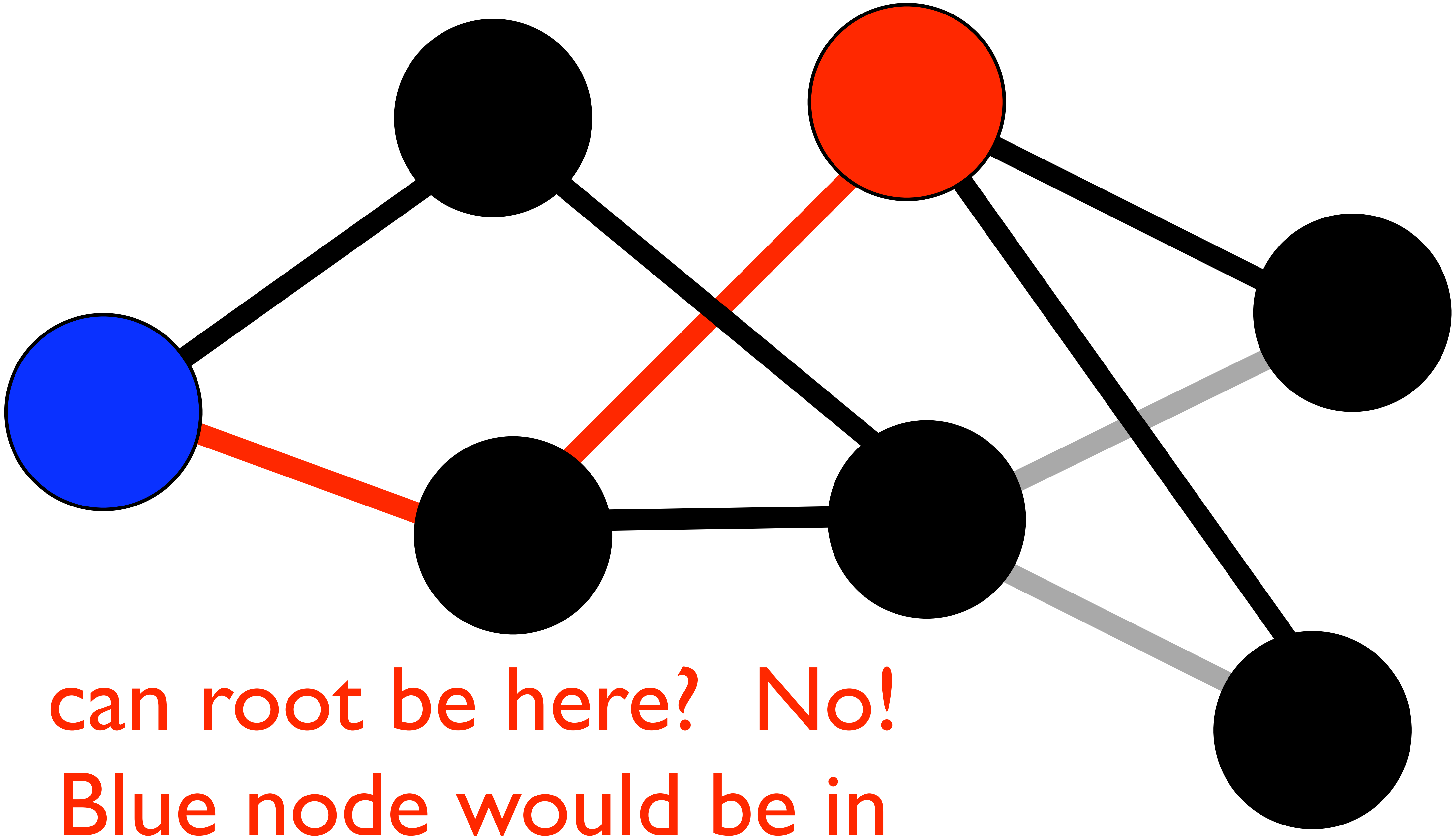
can root be here? No! Blue node would be in level 2 of tree.

Is this a BFS tree? 3



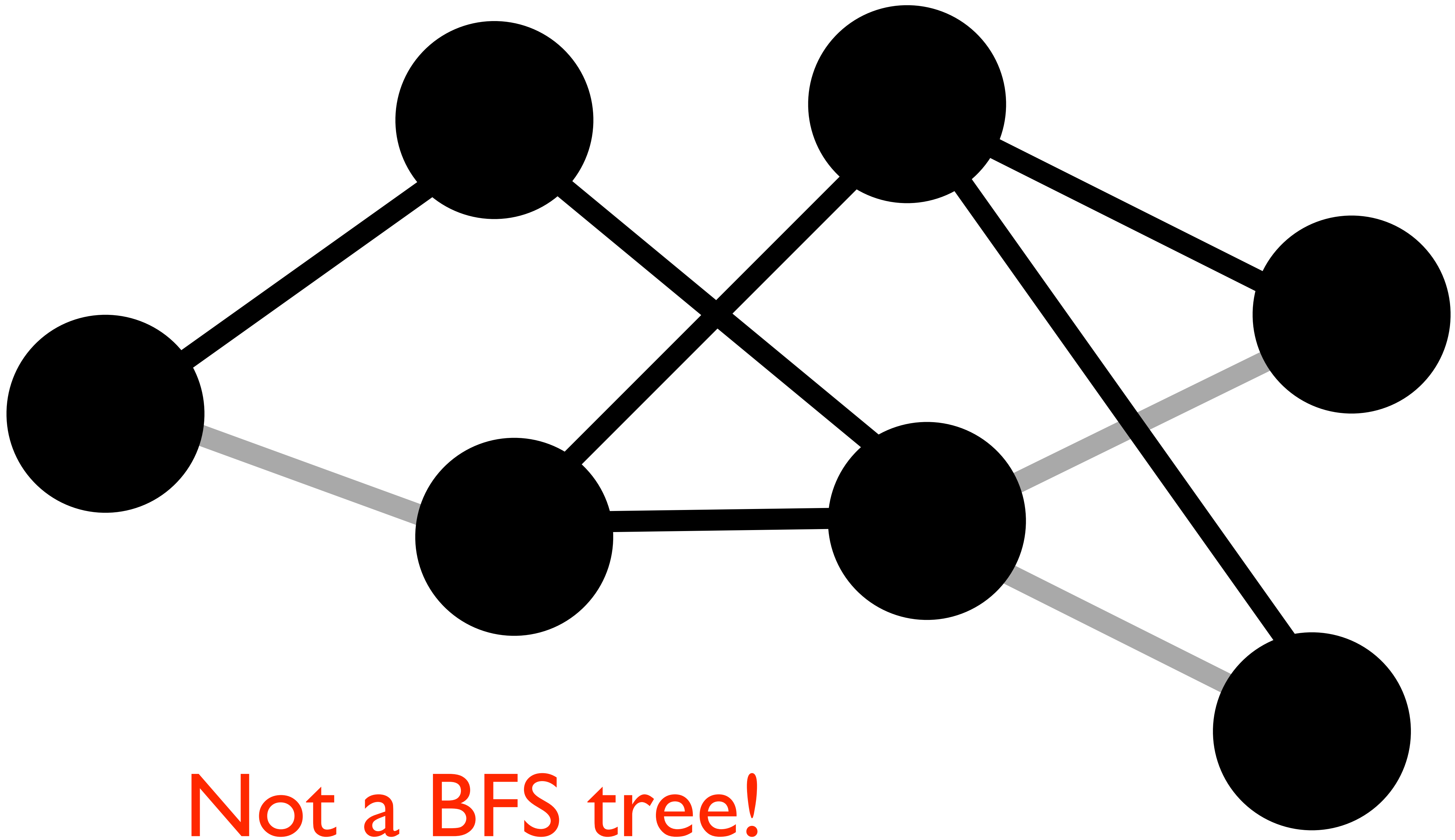
can root be here?

Is this a BFS tree? 3



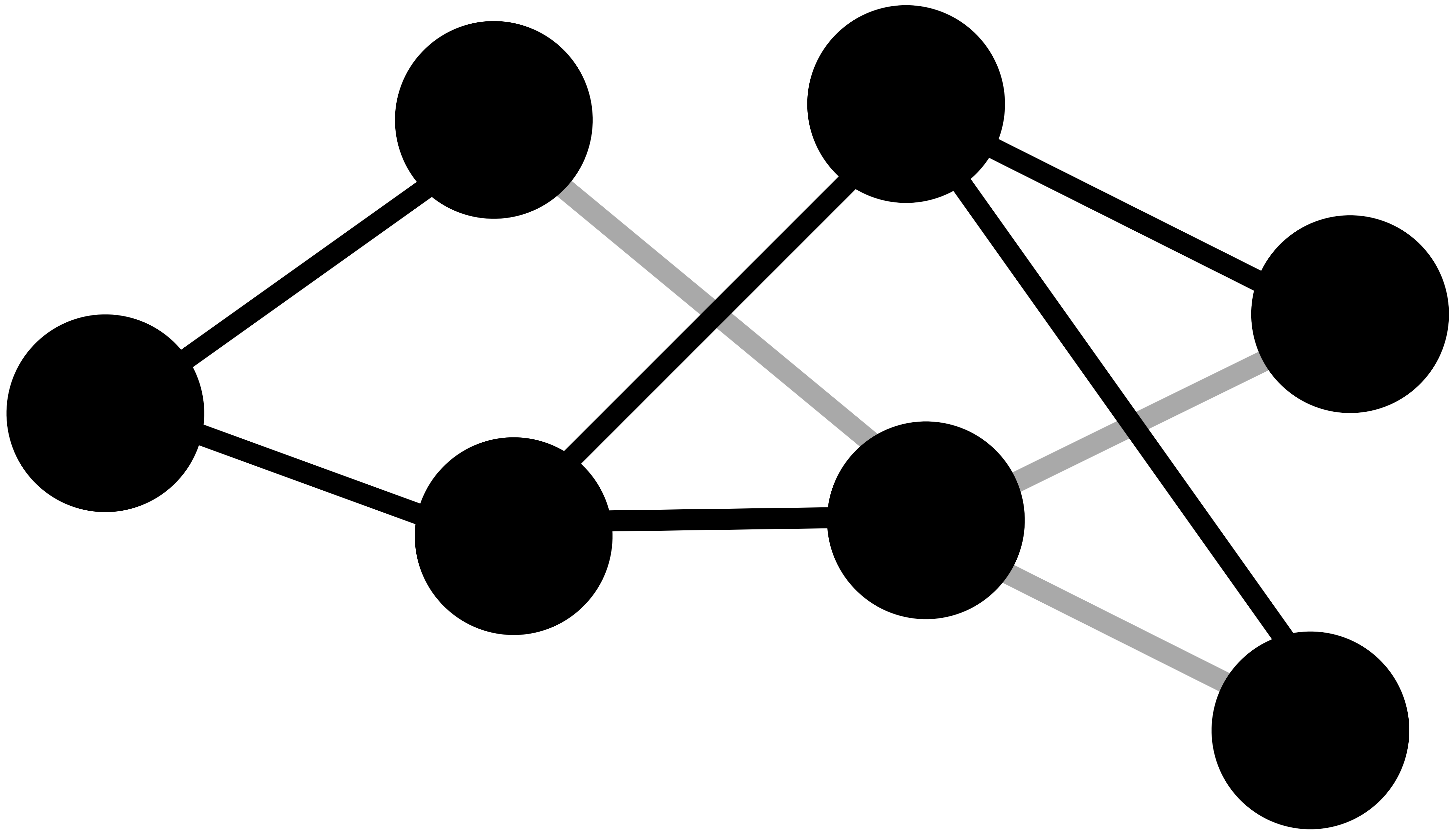
can root be here? No!
Blue node would be in
level 2 of tree.

Is this a BFS tree? 3

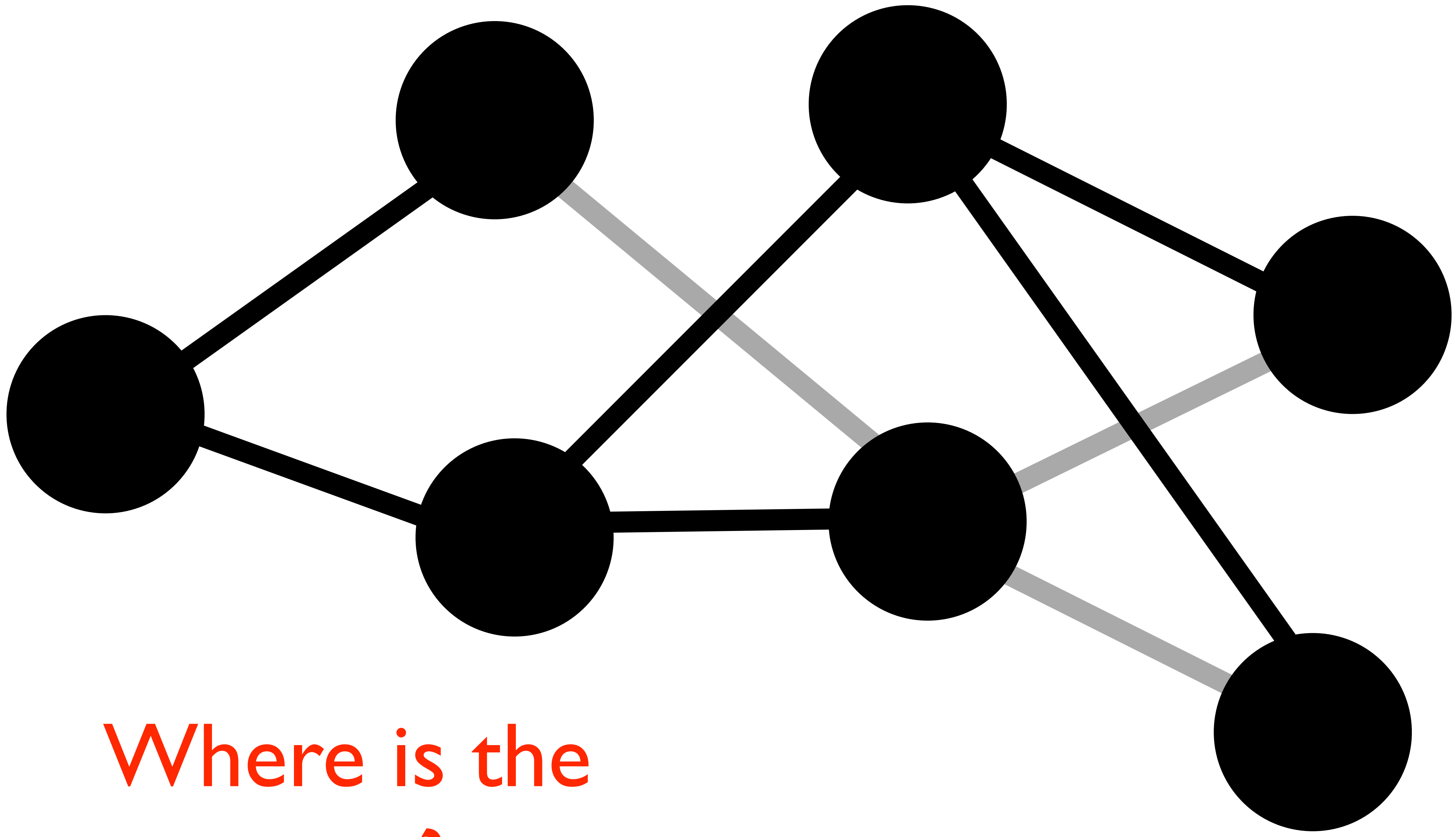


Not a BFS tree!

Is this a BFS tree? 4

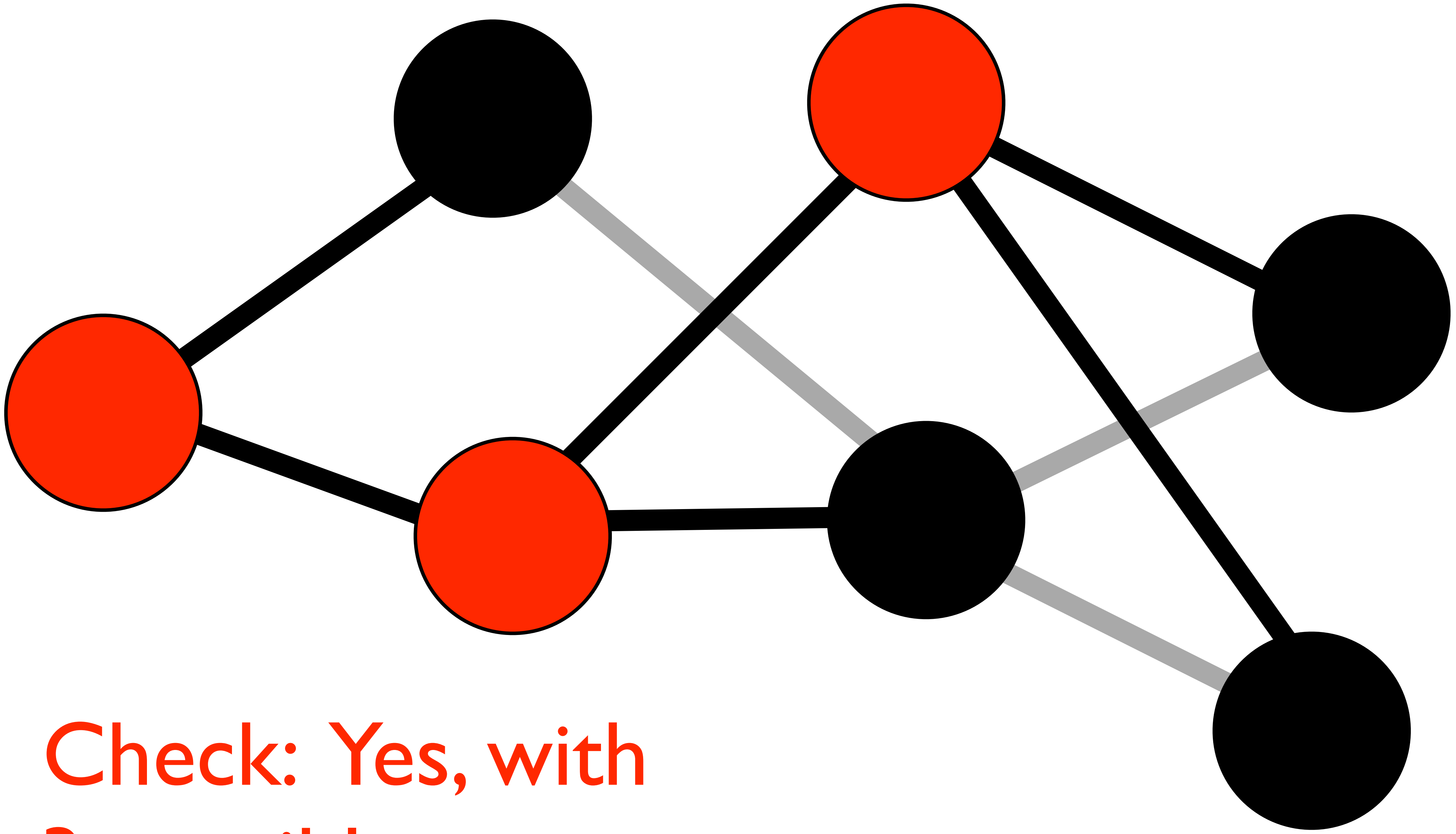


Is this a BFS tree? 4



Where is the root?

Is this a BFS tree? 4



Check: Yes, with
3 possible roots.

Depth-First Search

DFS: explores fully from each vertex, before backing up to try another one.

```
DFS(s): Mark s as found.
For each unfound neighbor v of s:
    add edge (v,s) to T.
    DFS(v)
```

DFS Trees

DFS: explores fully from each vertex, before backing up to try another one.

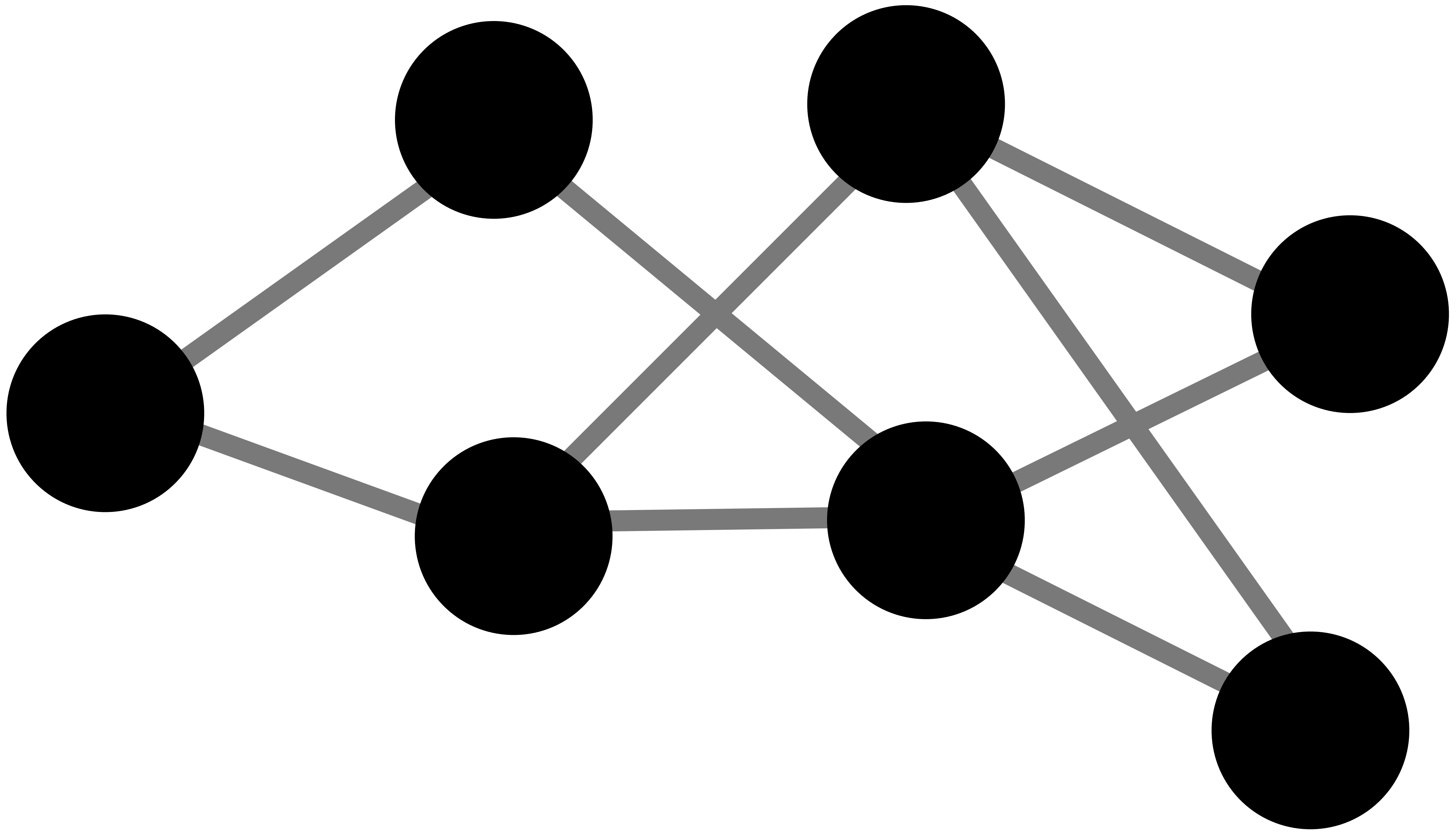
Theorem: Each time DFS looks at an edge, it either:

(a) is added to the tree, or

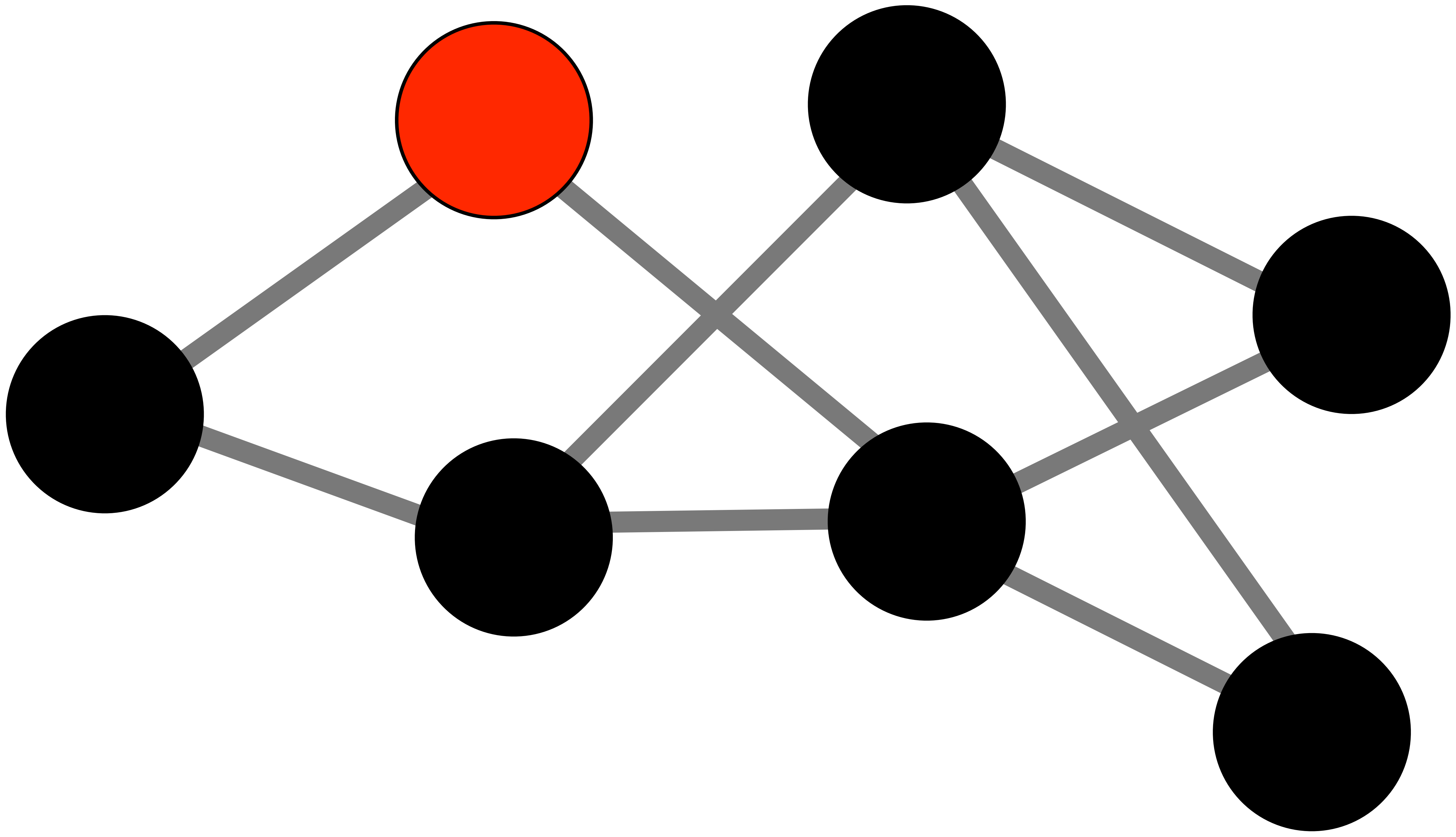
(b) is a back-edge to an ancestor of the current node. (see (3.7) on page 85)

Lemma: Active nodes are always a path from the root.

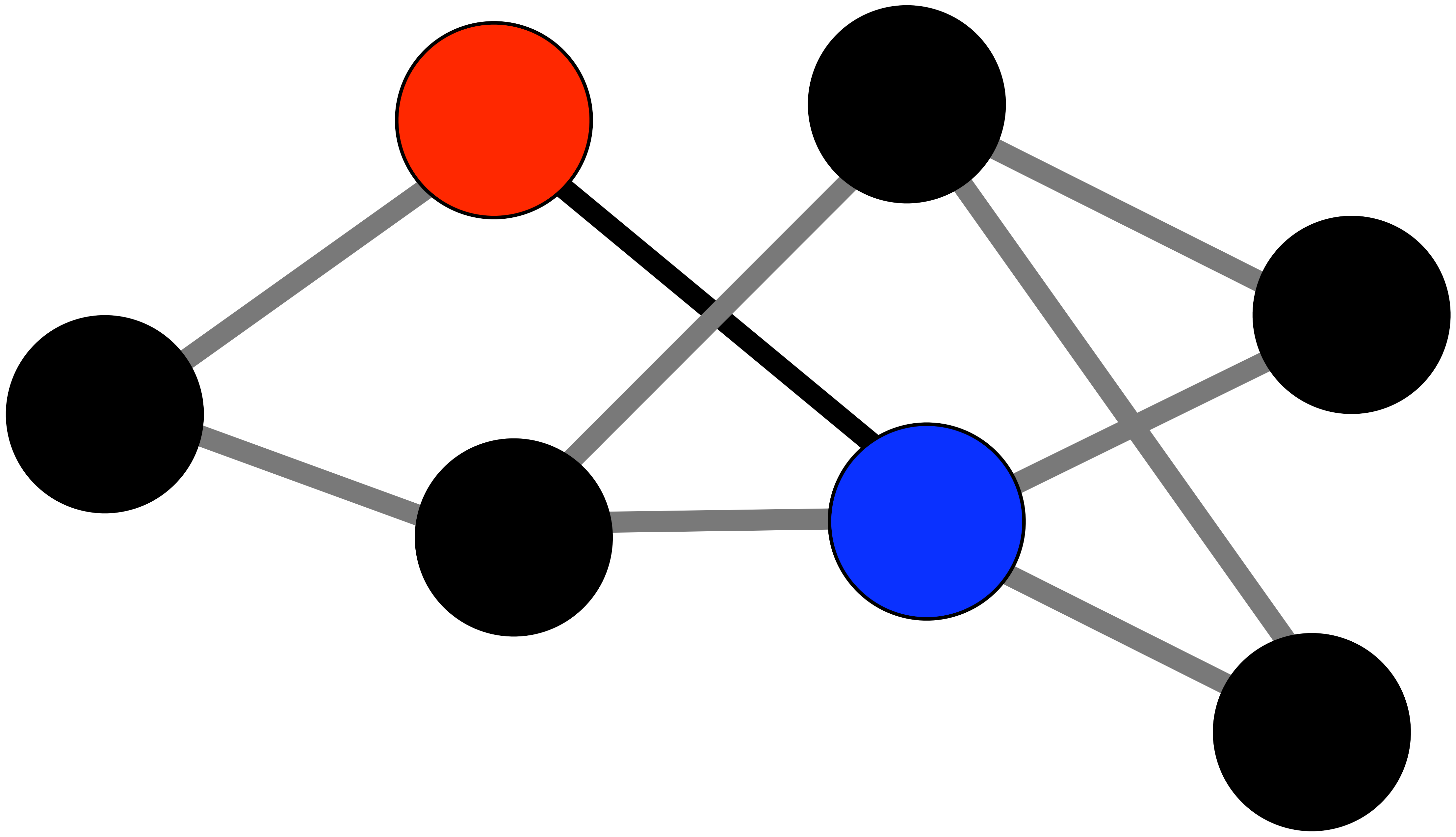
DFS example



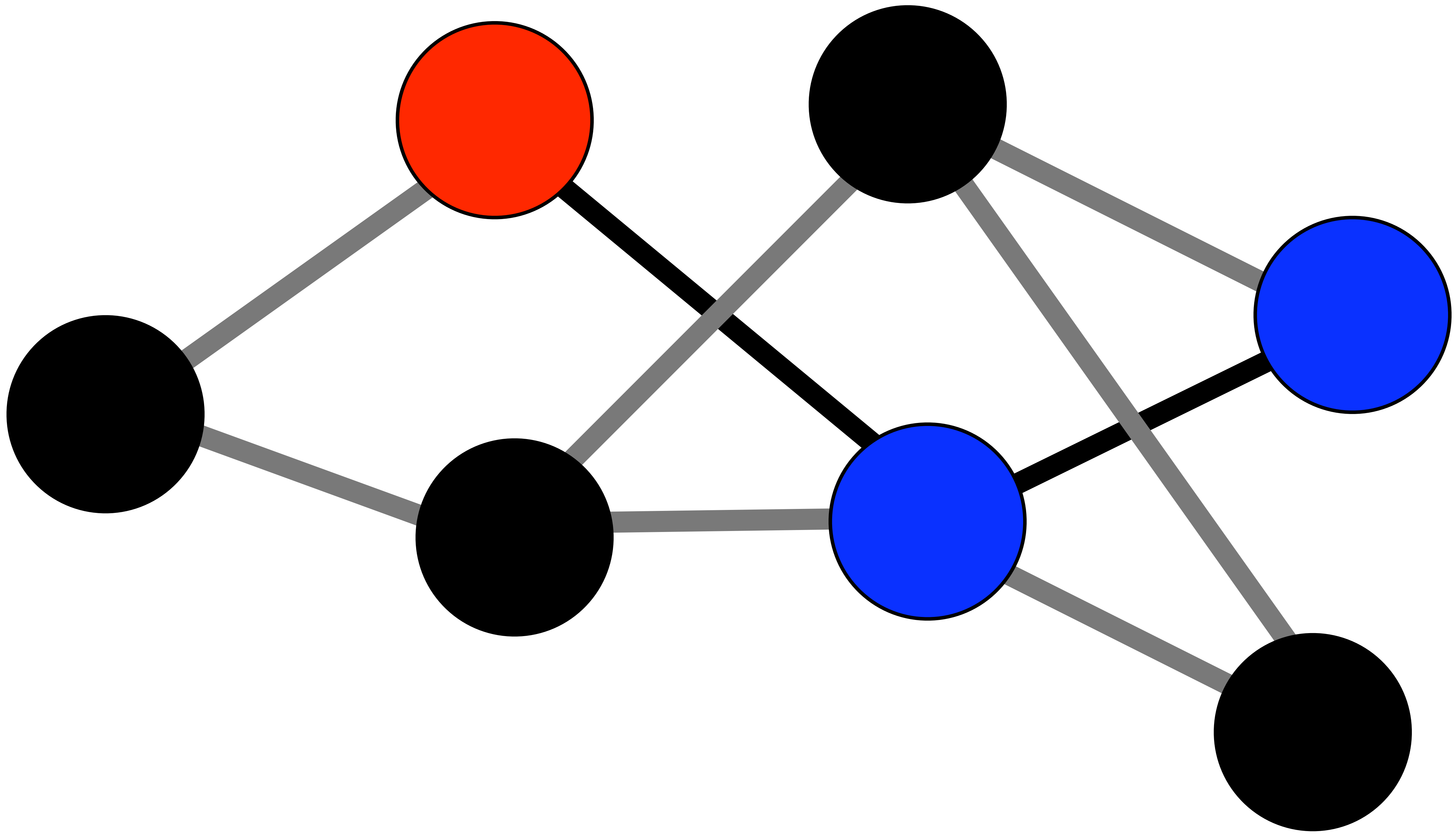
DFS example



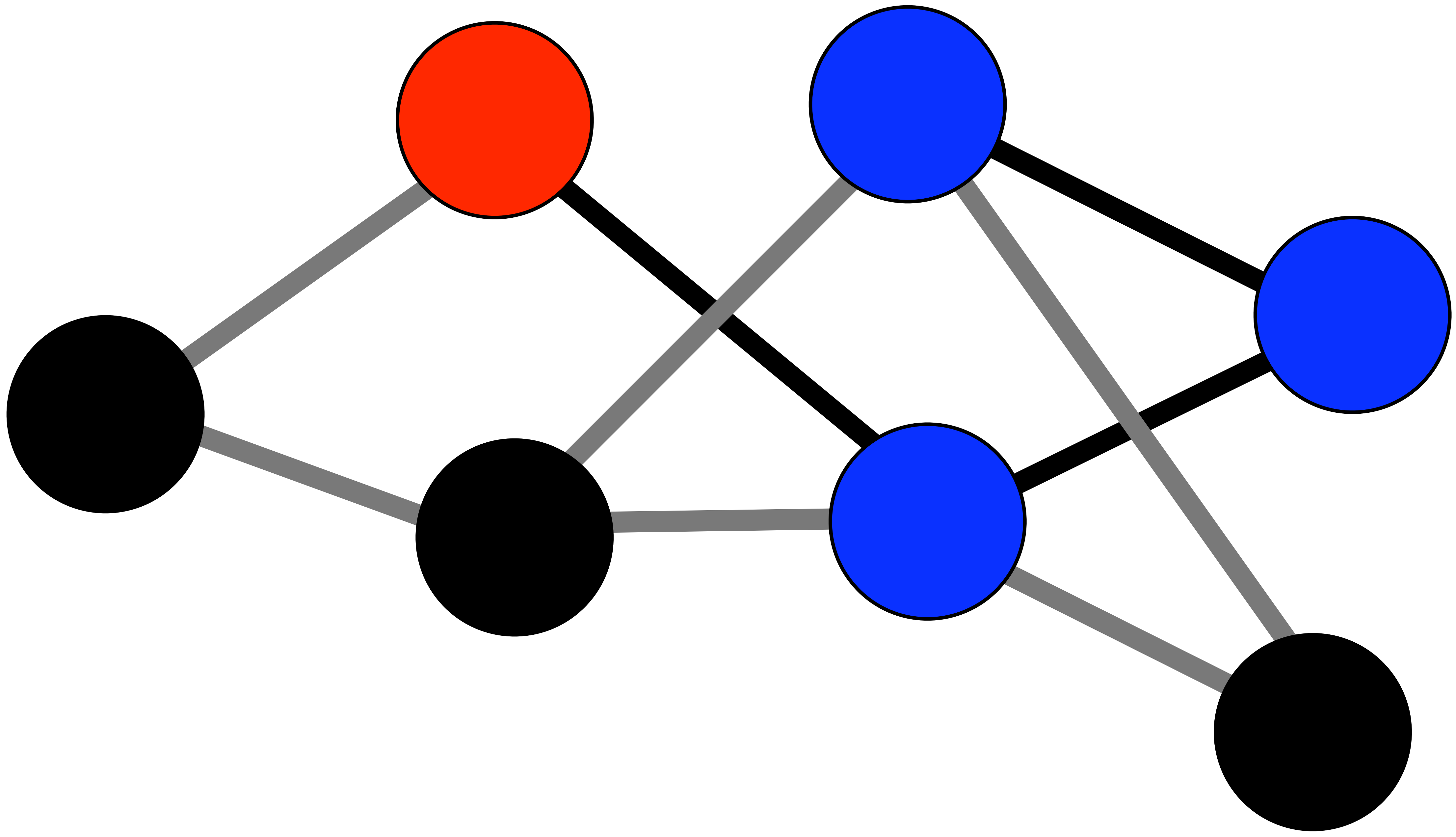
DFS example



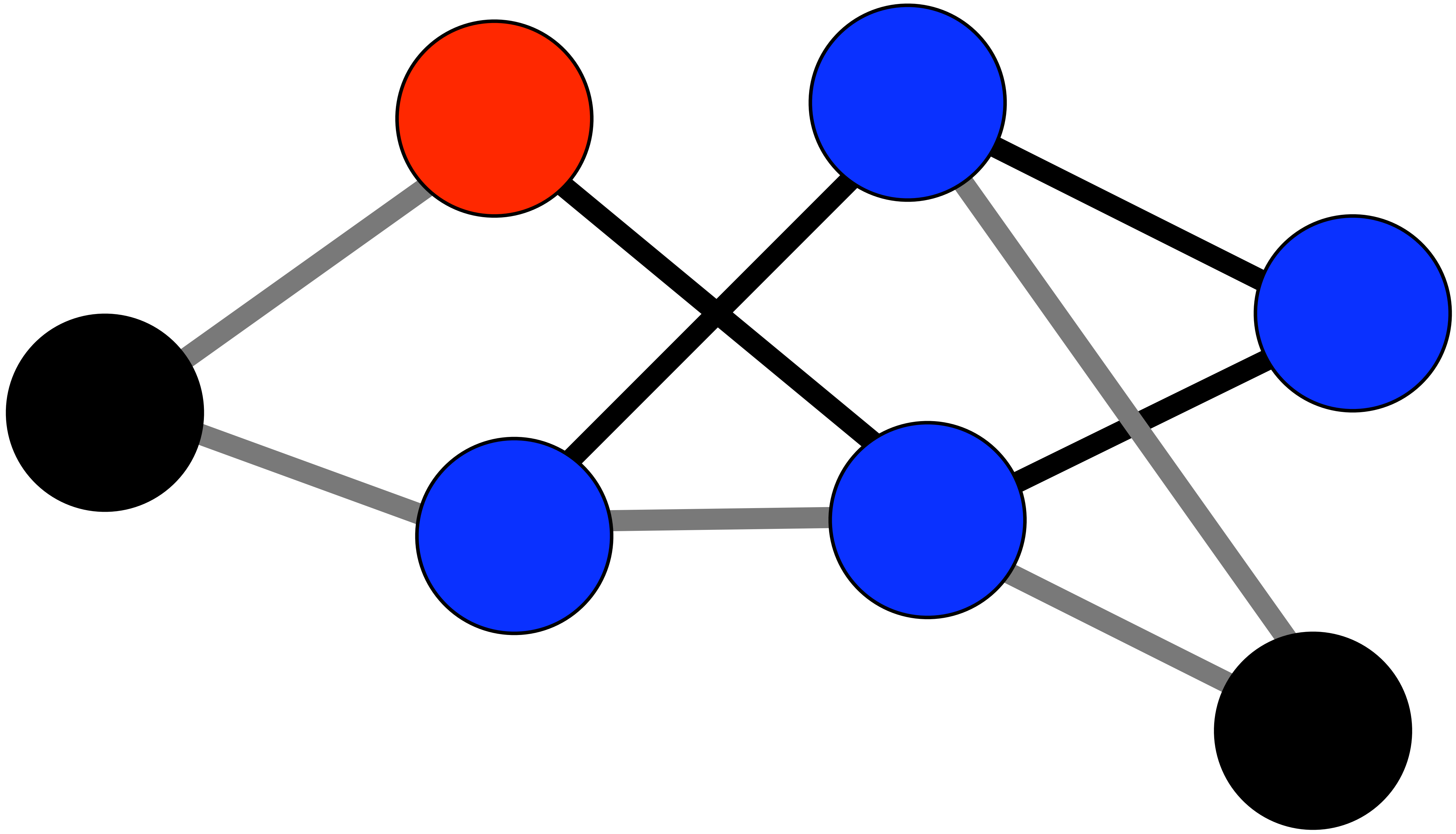
DFS example



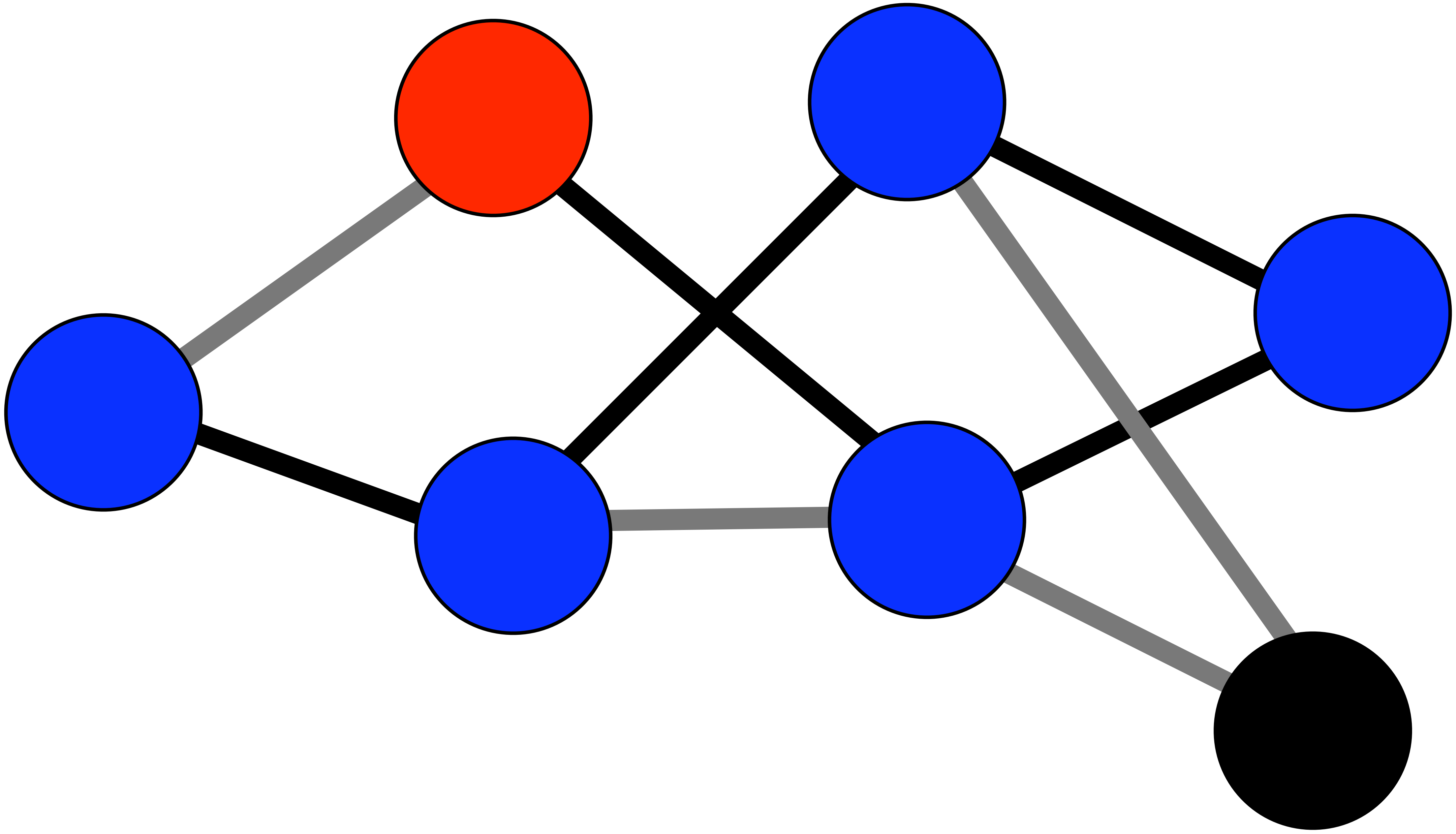
DFS example



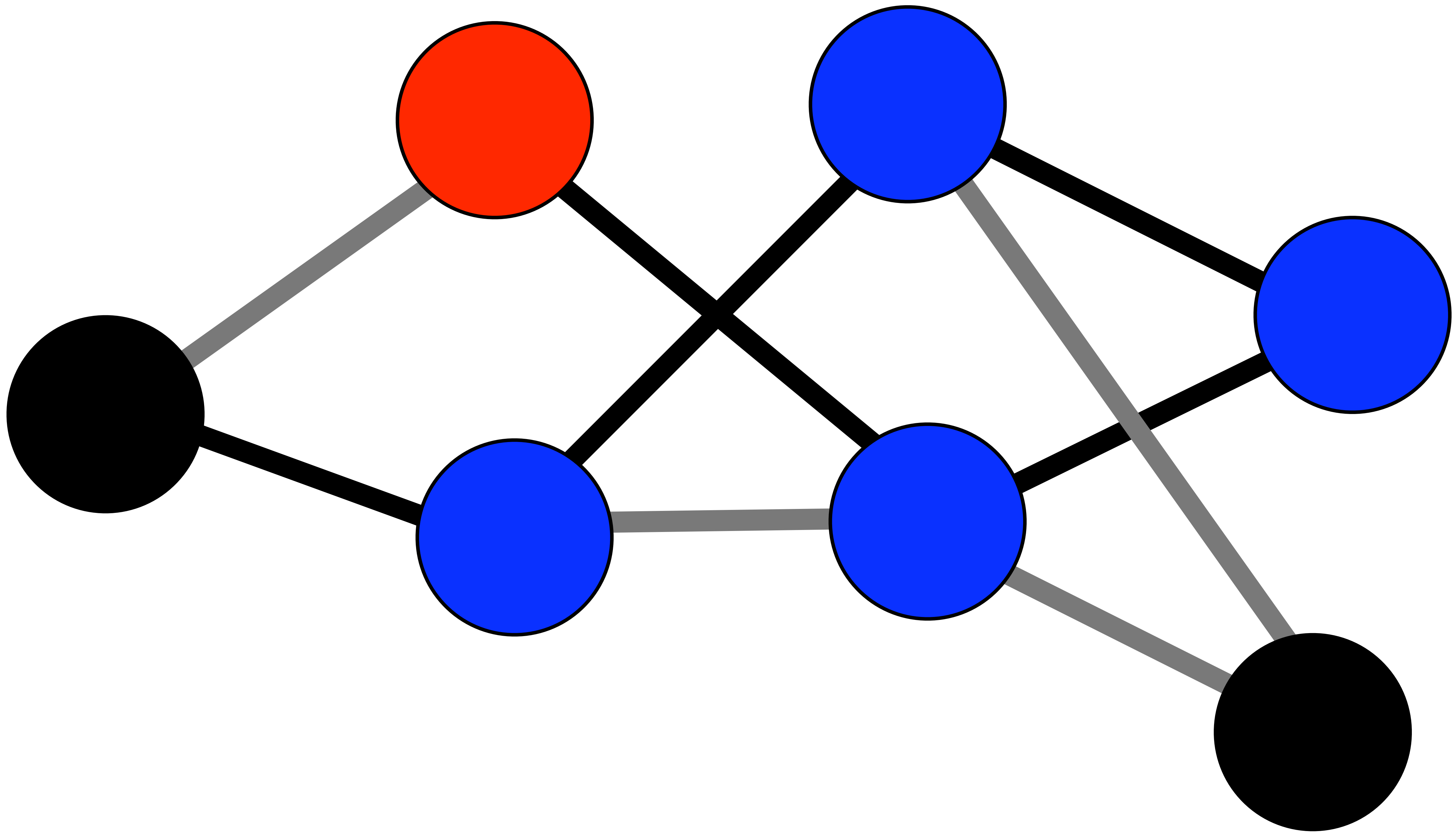
DFS example



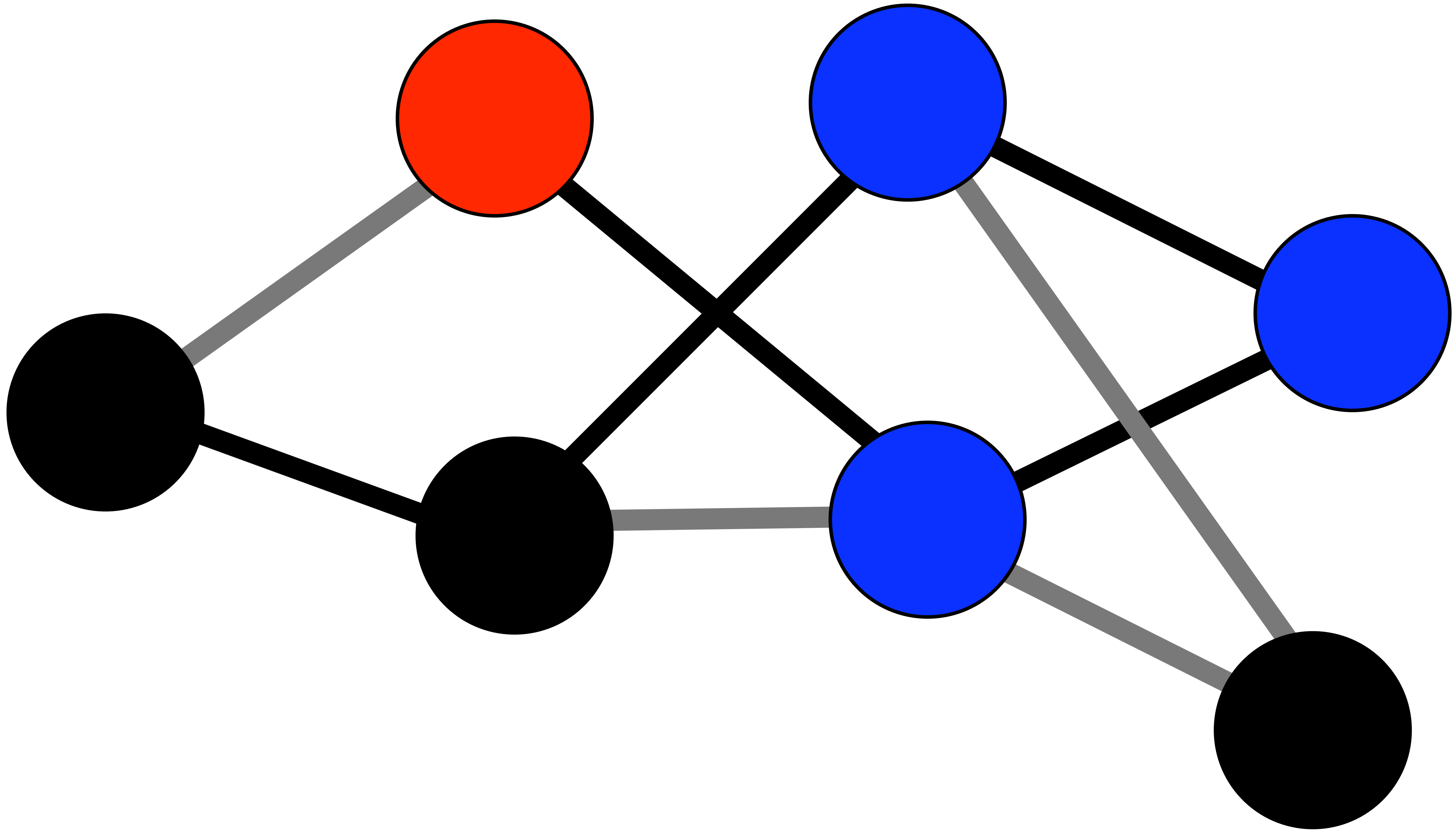
DFS example



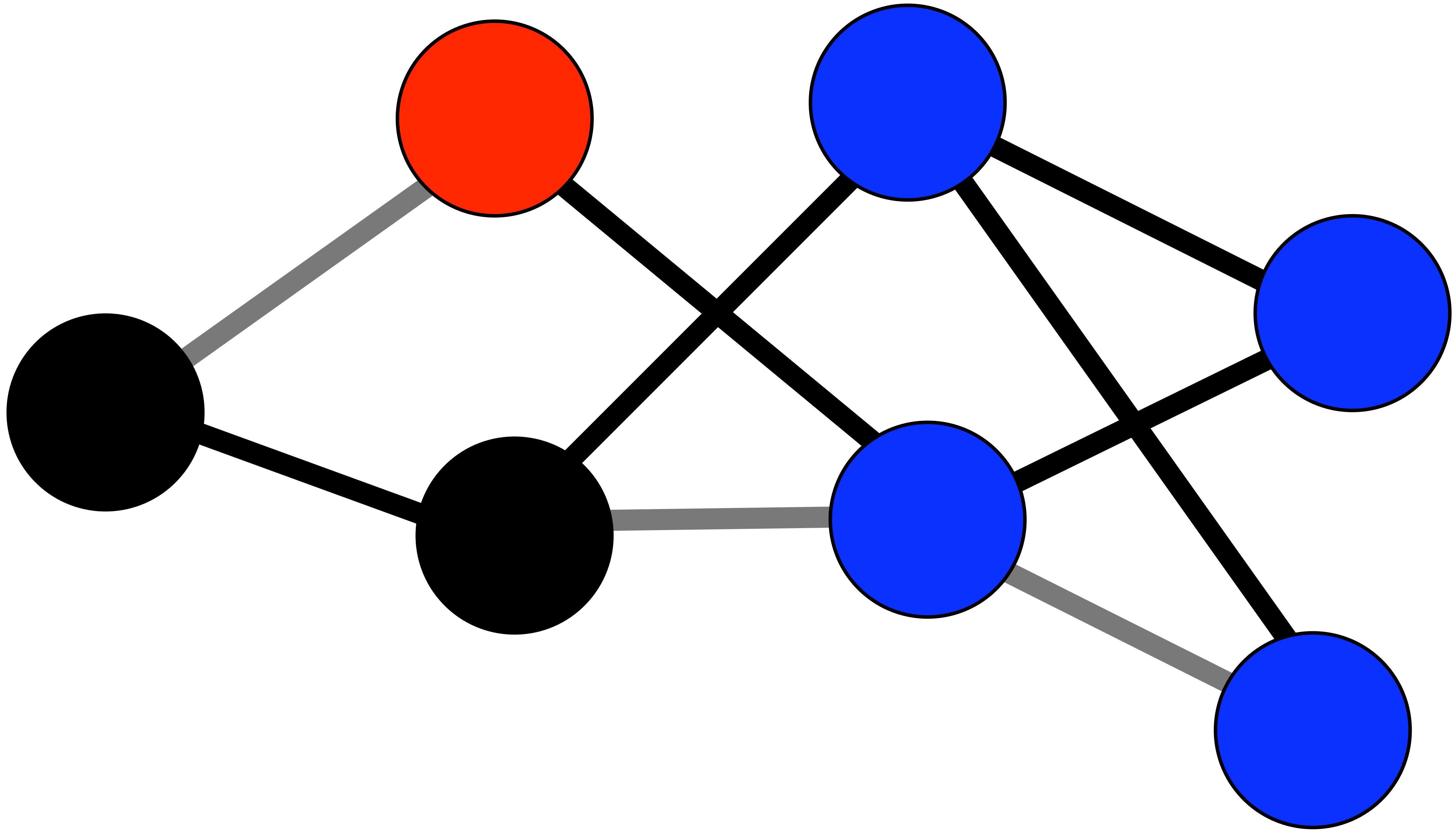
DFS example



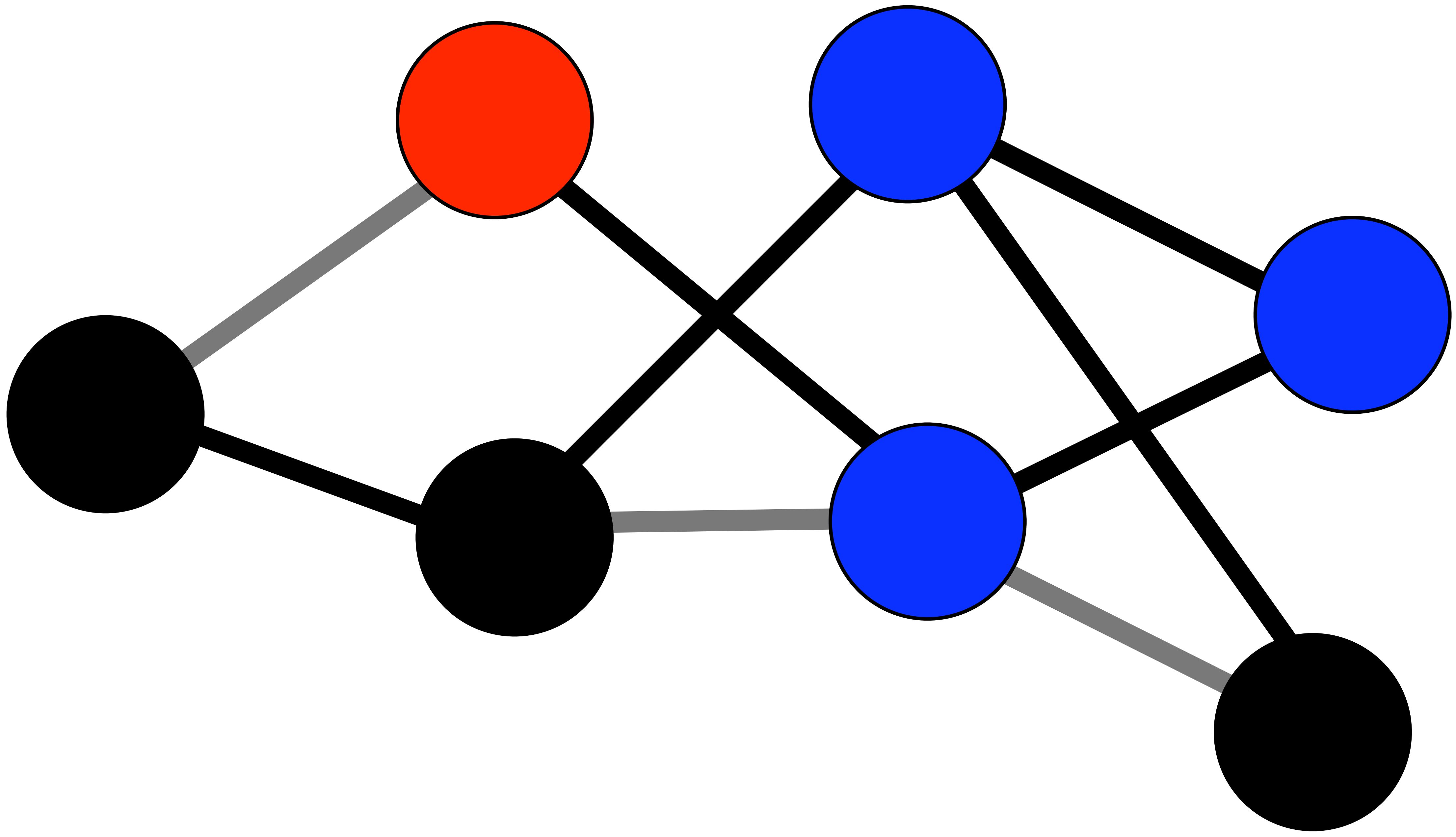
DFS example



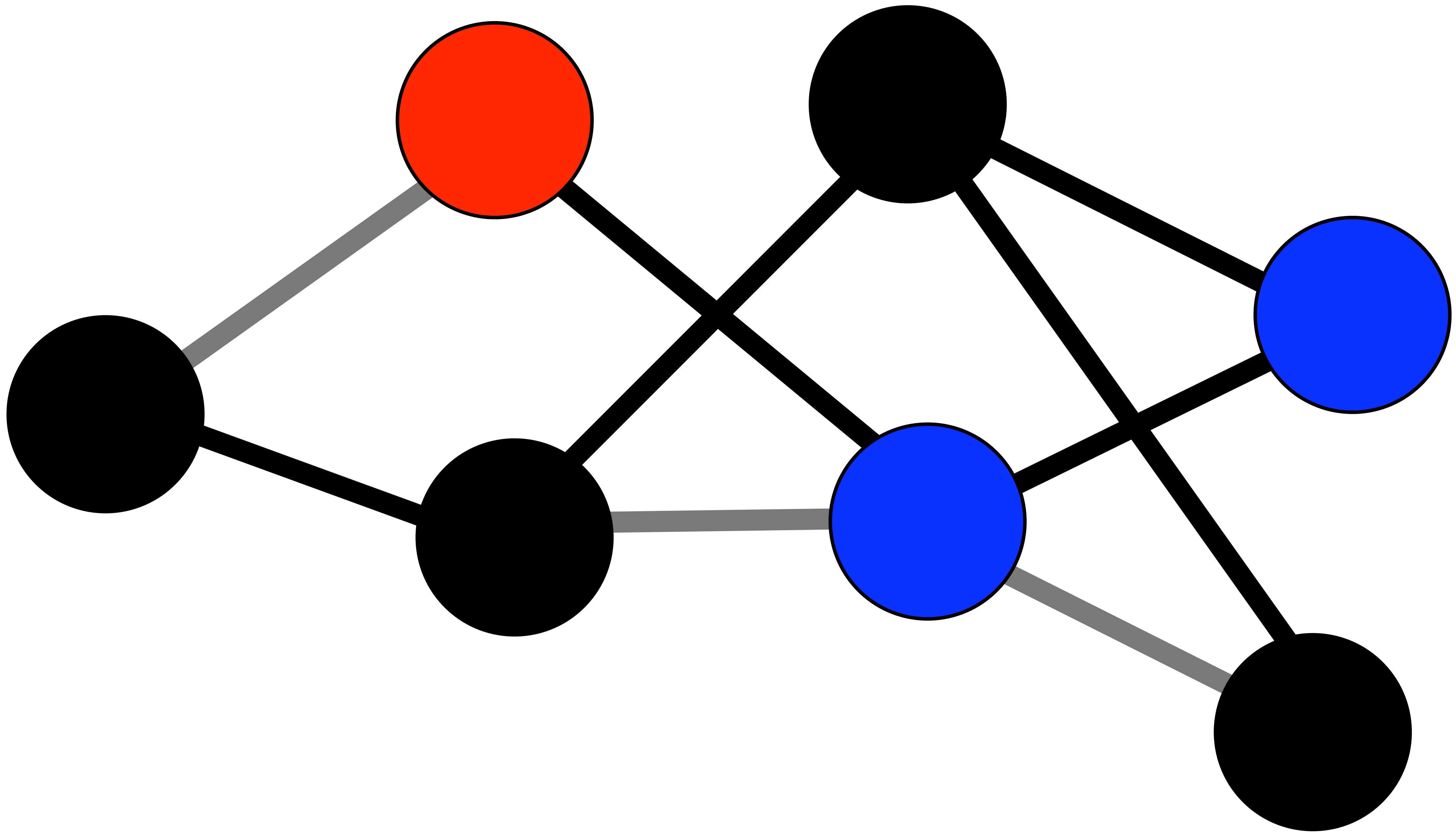
DFS example



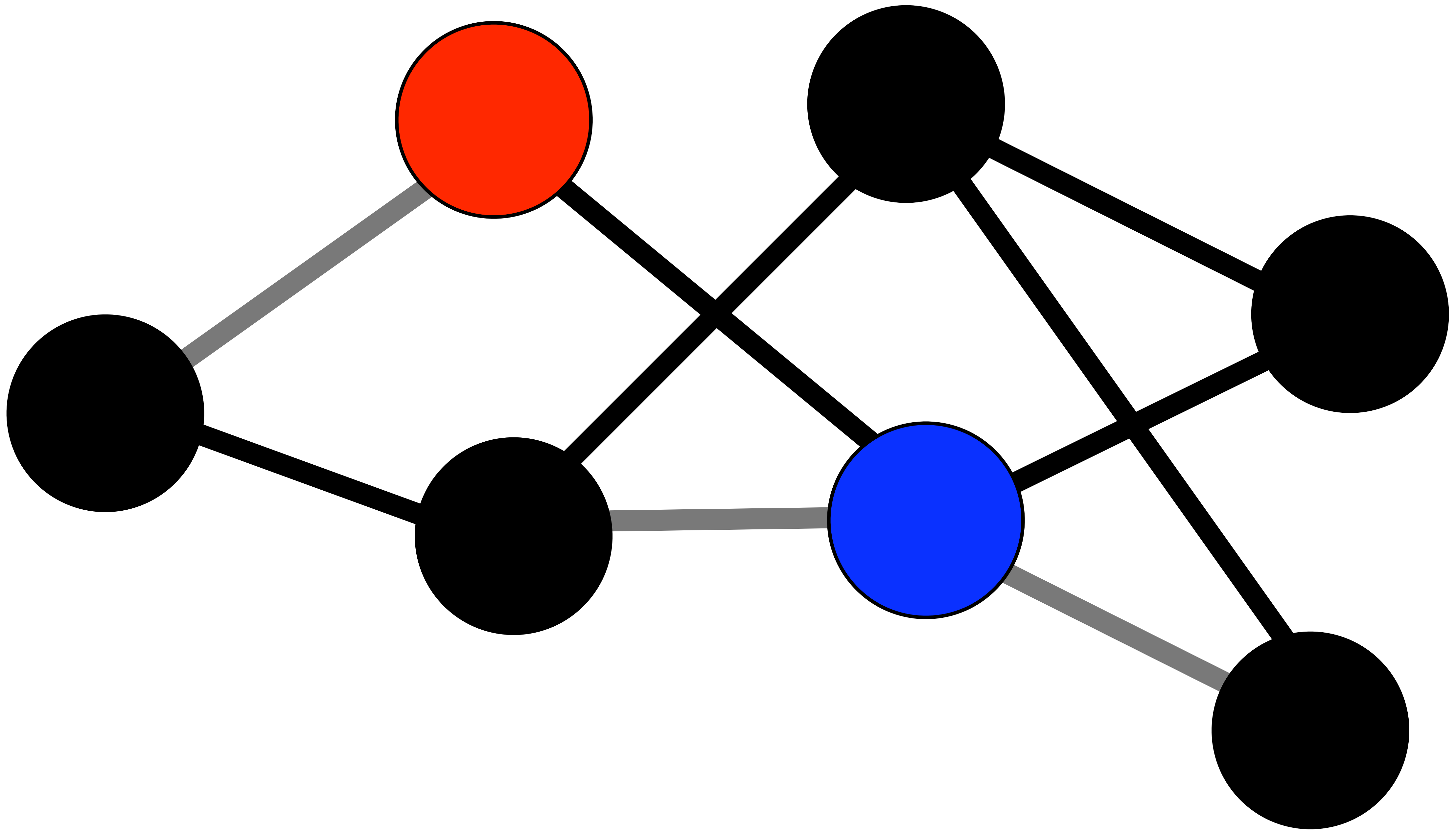
DFS example



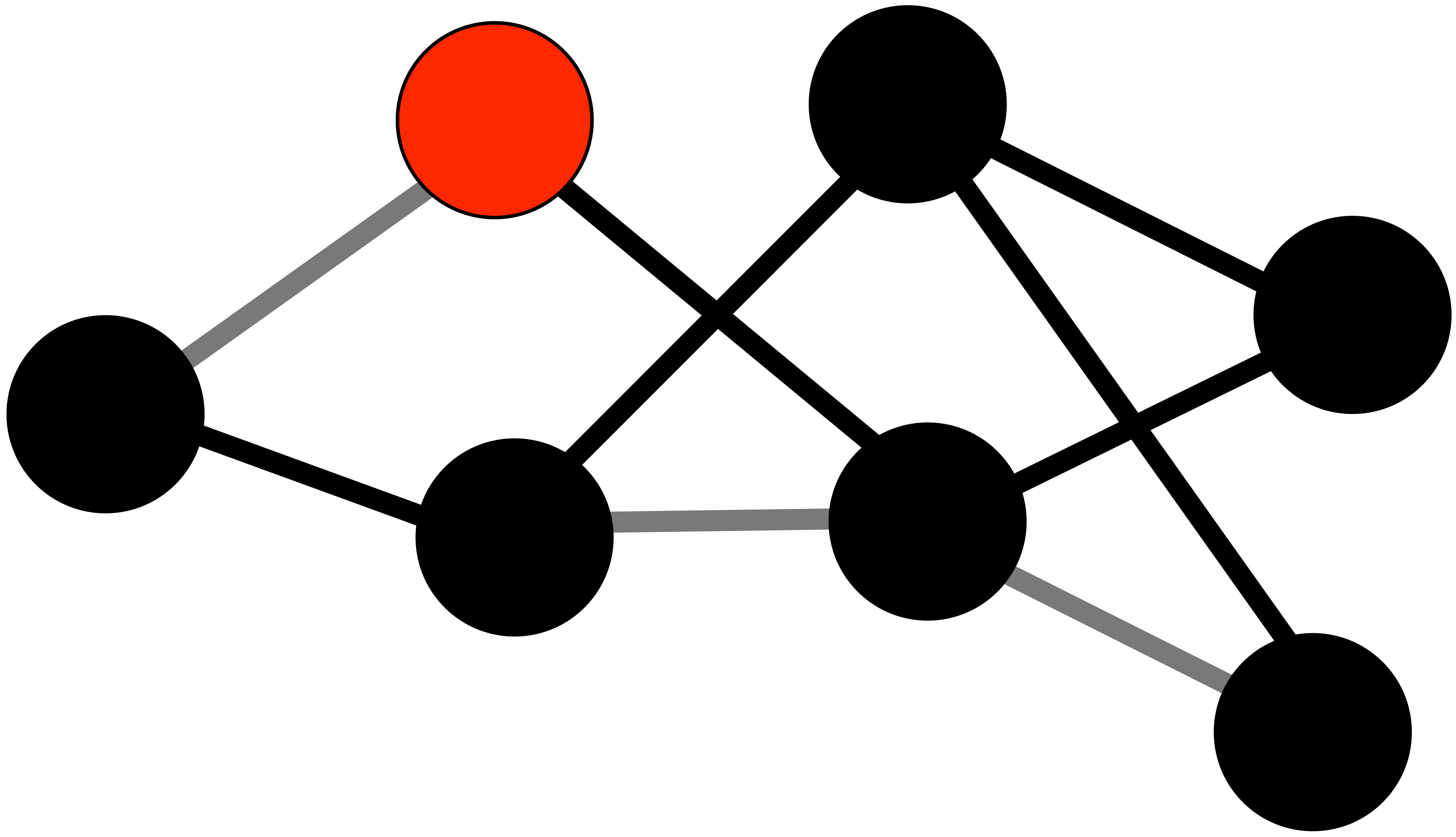
DFS example



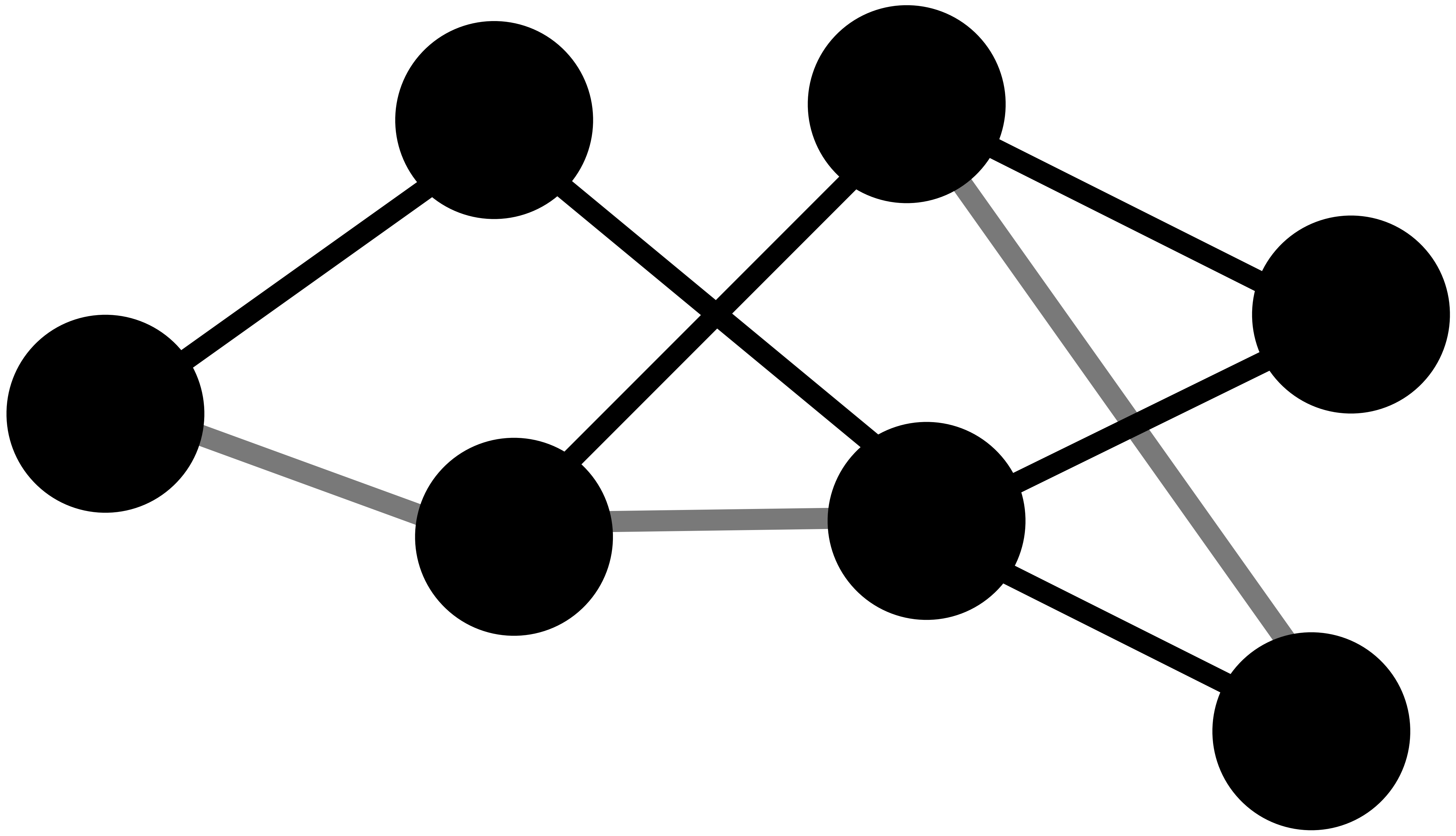
DFS example



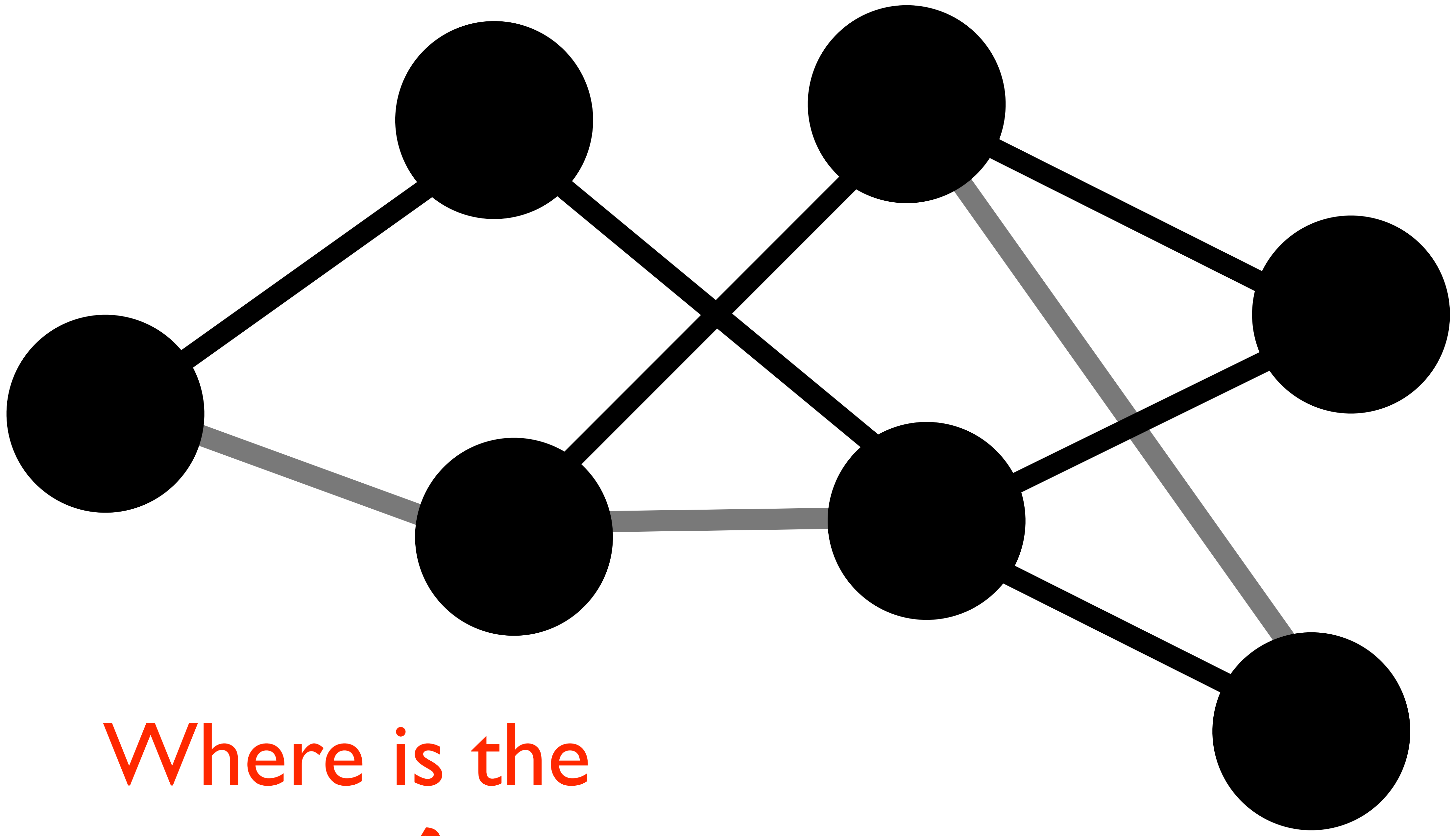
DFS example



Is this a DFS tree?

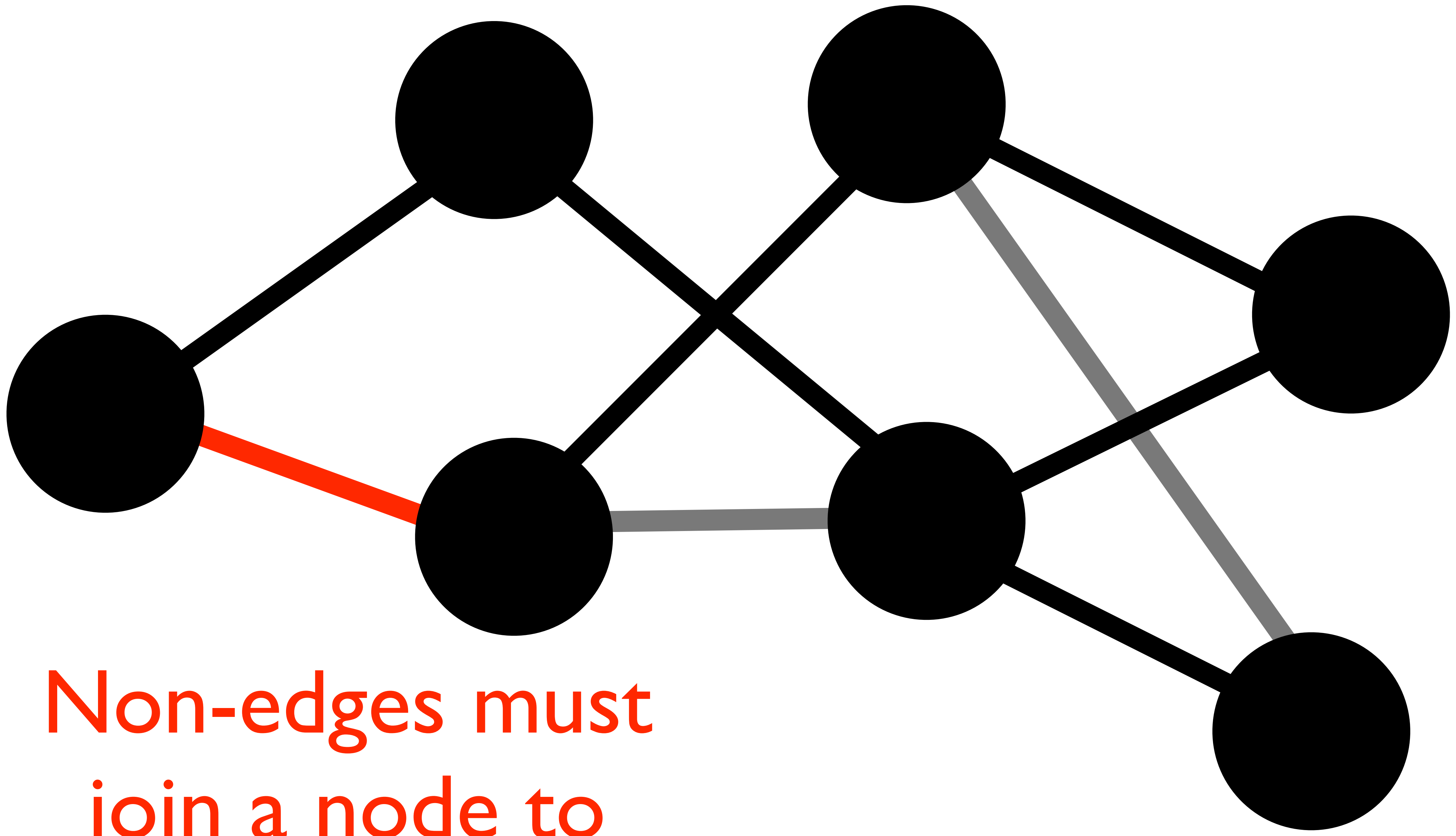


Is this a DFS tree?



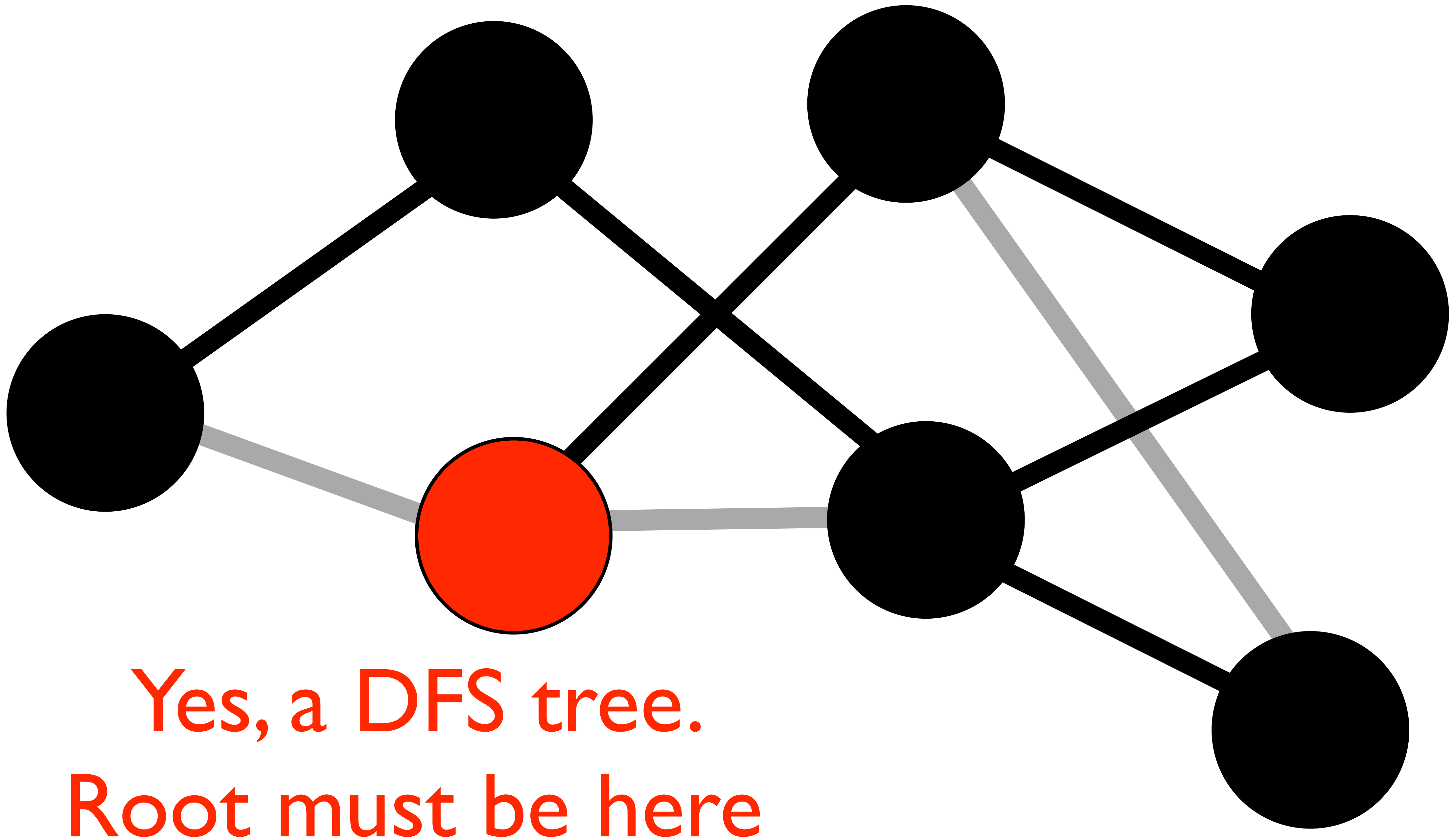
Where is the root?

Is this a DFS tree?



Non-edges must join a node to its ancestor.

Is this a DFS tree?



Yes, a DFS tree.
Root must be here