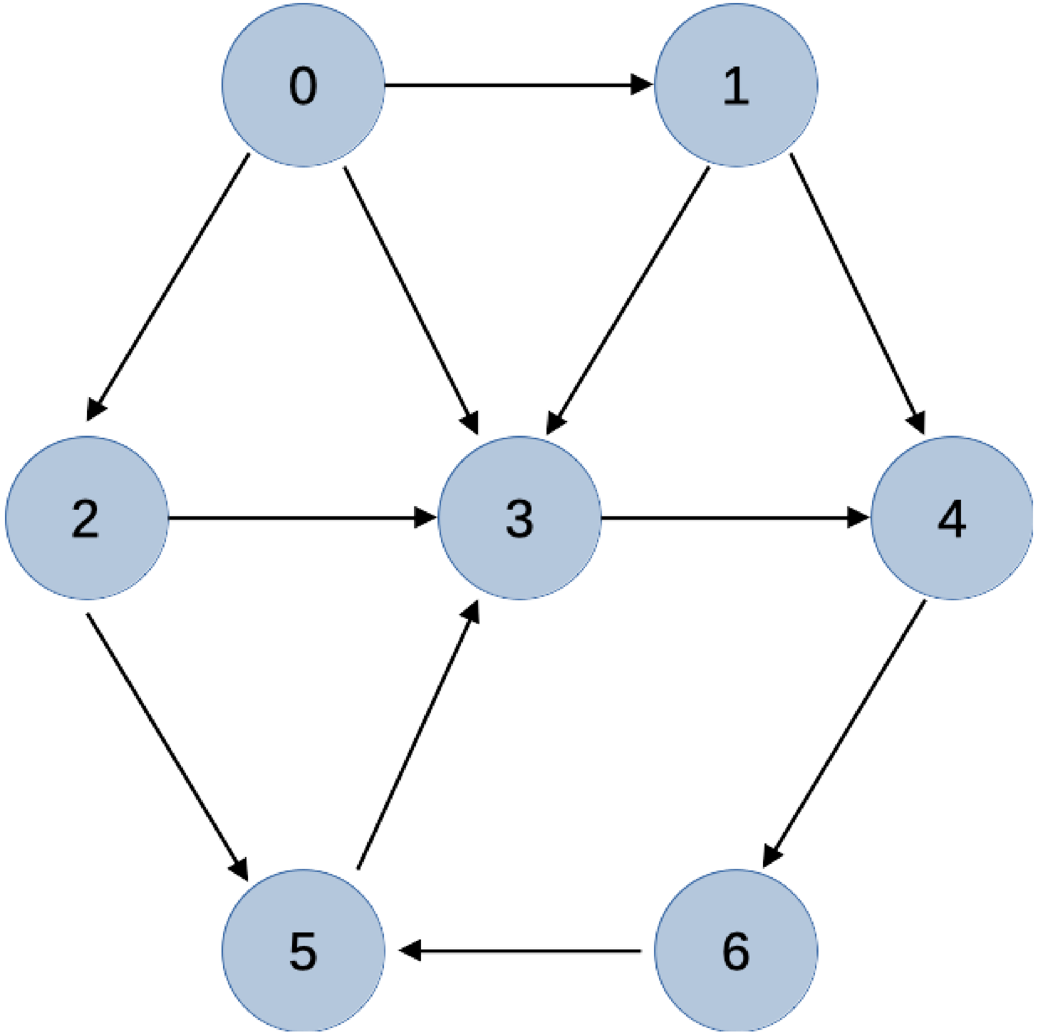
Competitive Programming

Graph Traversal

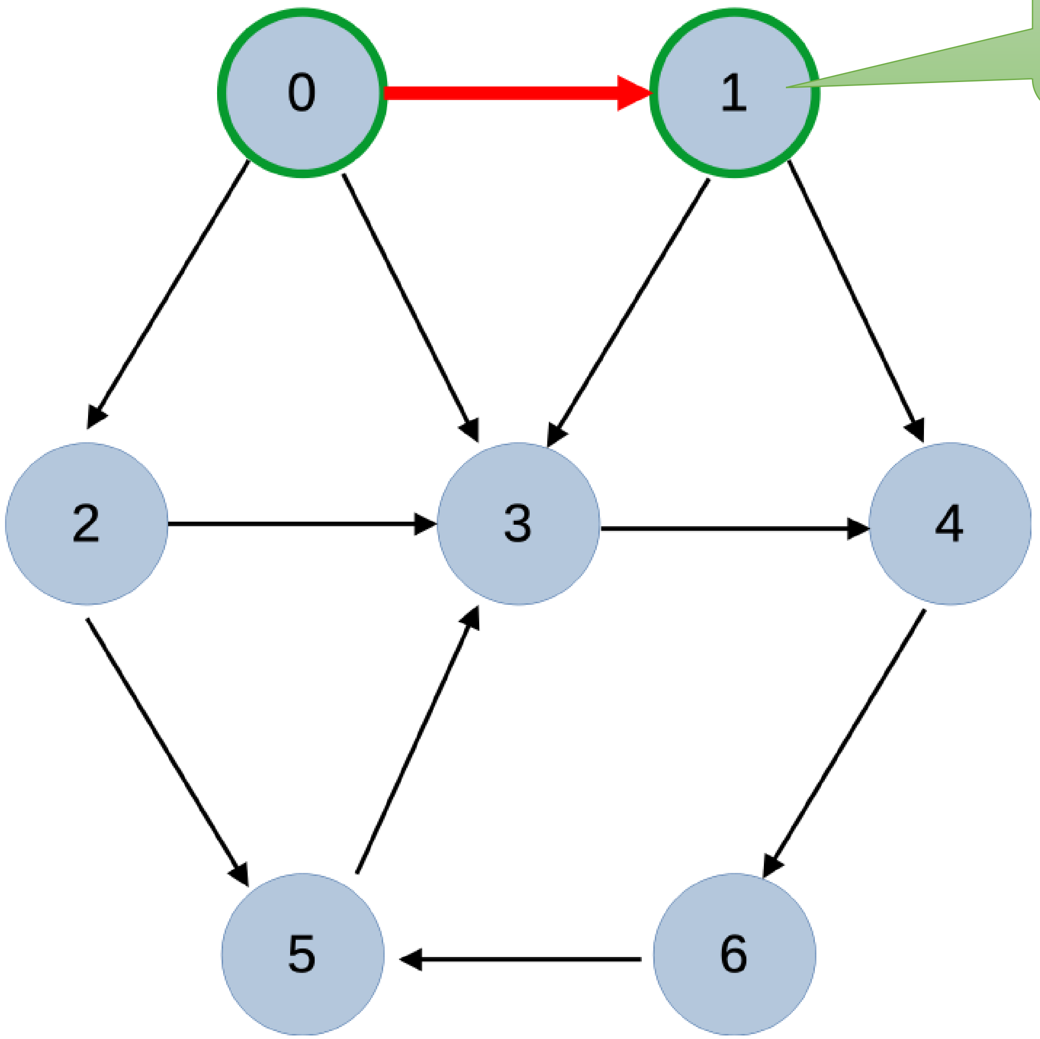
Graph Traversal

- Graph traversal : visit the vertices (in a particular order)
- Depth-first order
  - Easy to implement recursively
  - Good for finding cycles, finding components, topological-sort
- Breadth-first traversal
  - Easy-ish to implement with a queue
  - Good for finding shortest paths (measured in edges)

Example Graph

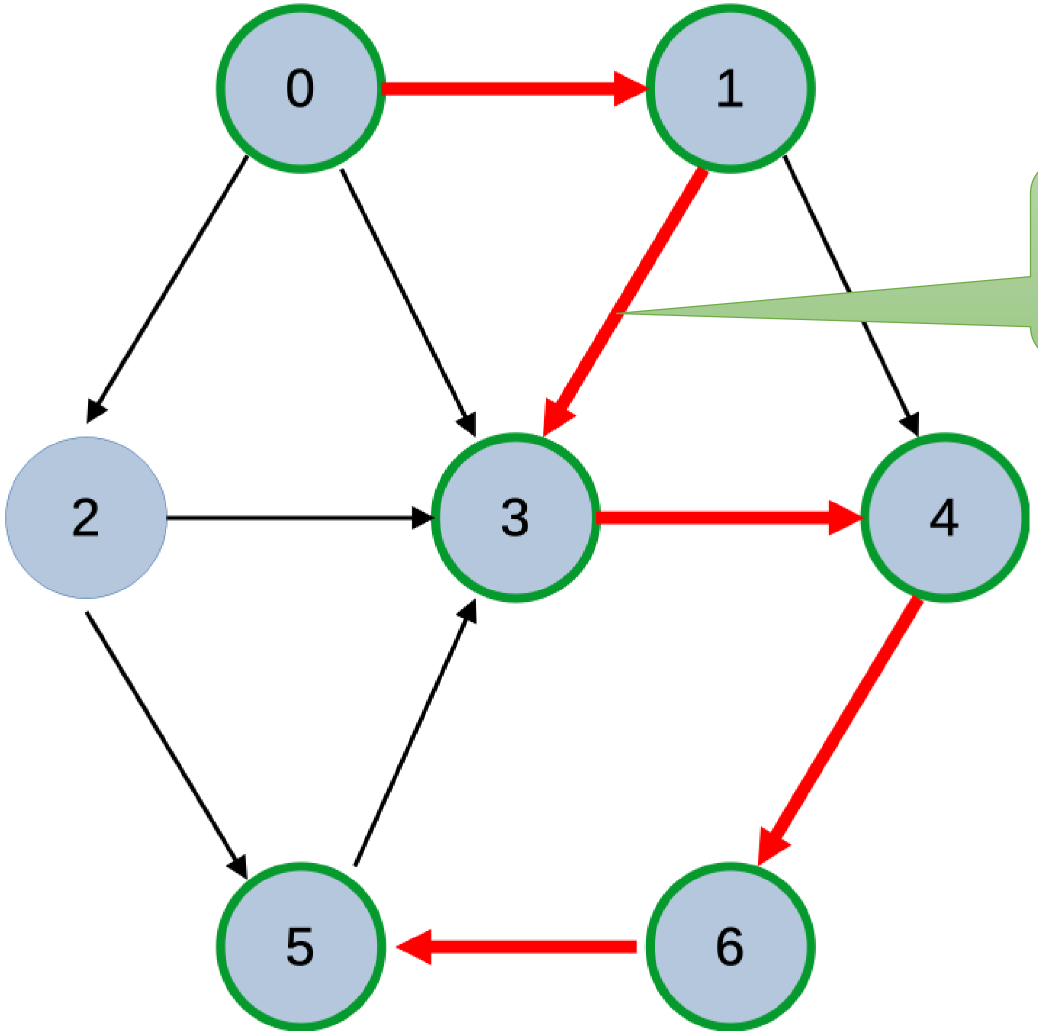


Depth-First Traversal



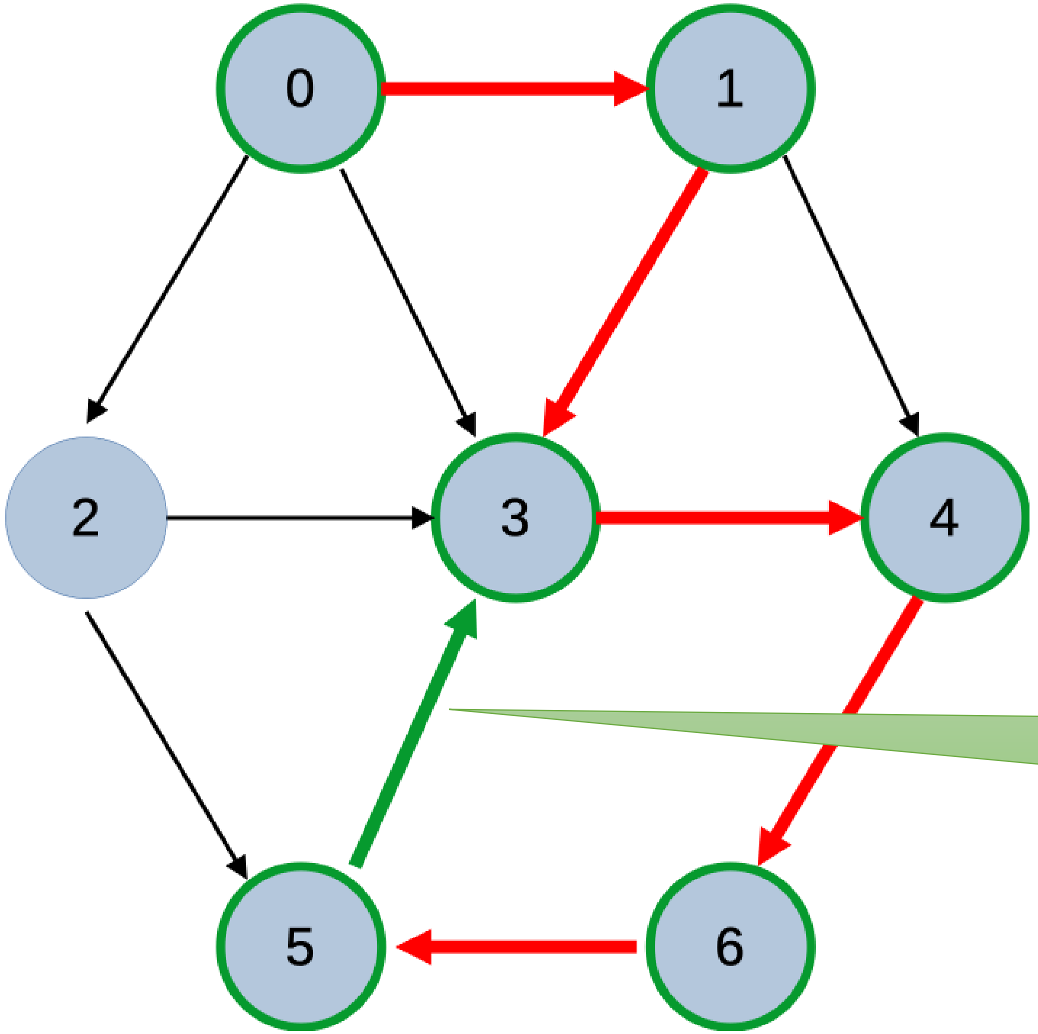
Marking vertices along the current path.

Depth-First Traversal



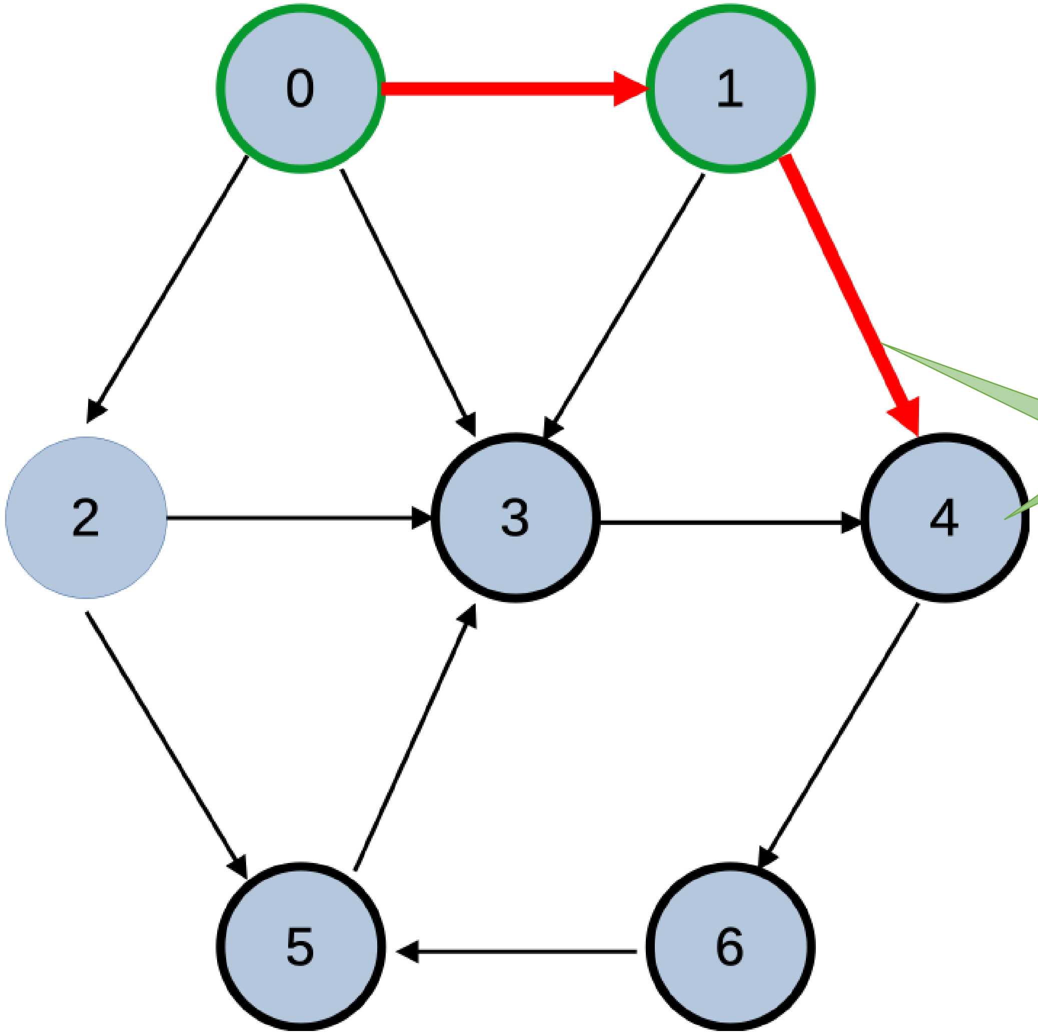
Following edges out of each vertex as we discover it.

Depth-First Traversal



A cycle, an edge back to a vertex being explored.

Depth-First Traversal



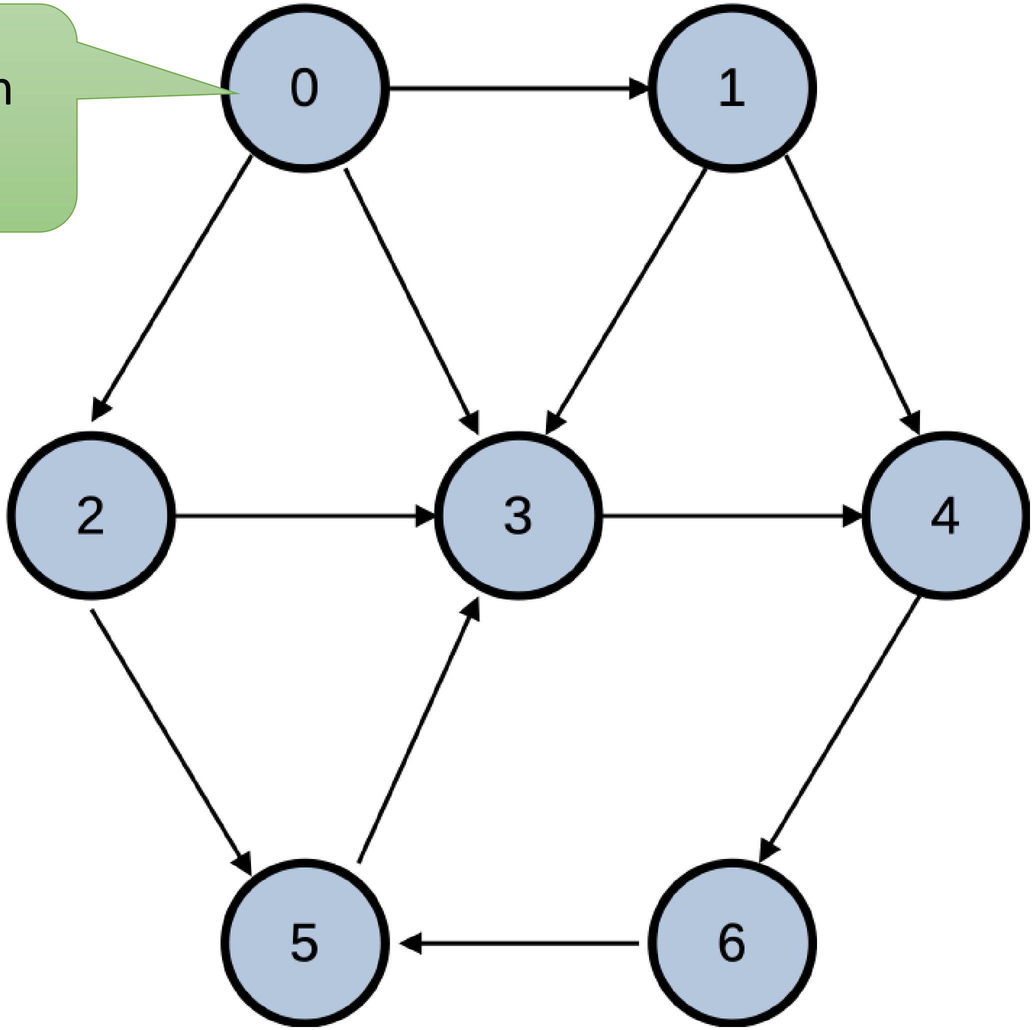
Changing the color of a vertex as we finish exploring it.

Prevents repeated exploring of the same part of the graph (exponential time).

Depth-First Traversal

```
void traverse( int i, int state[ n ] ) {
  if ( state[ i ] == 2 )    // Are we finished with this vertex?
    return;

  if ( state[ i ] == 1 )    // Are we currently exploring this vertex?
    // this is a cycle.
    return;

  state[ i ] = 1;           // OK.  Now, we’re exploring from this vertex.

  for ( j : neighbors[ i ] )  // Recursively explore its neighbors.
    traverse( j, state );

  state[ i ] = 2;           // Now, we’re done with this vertex.
}
```

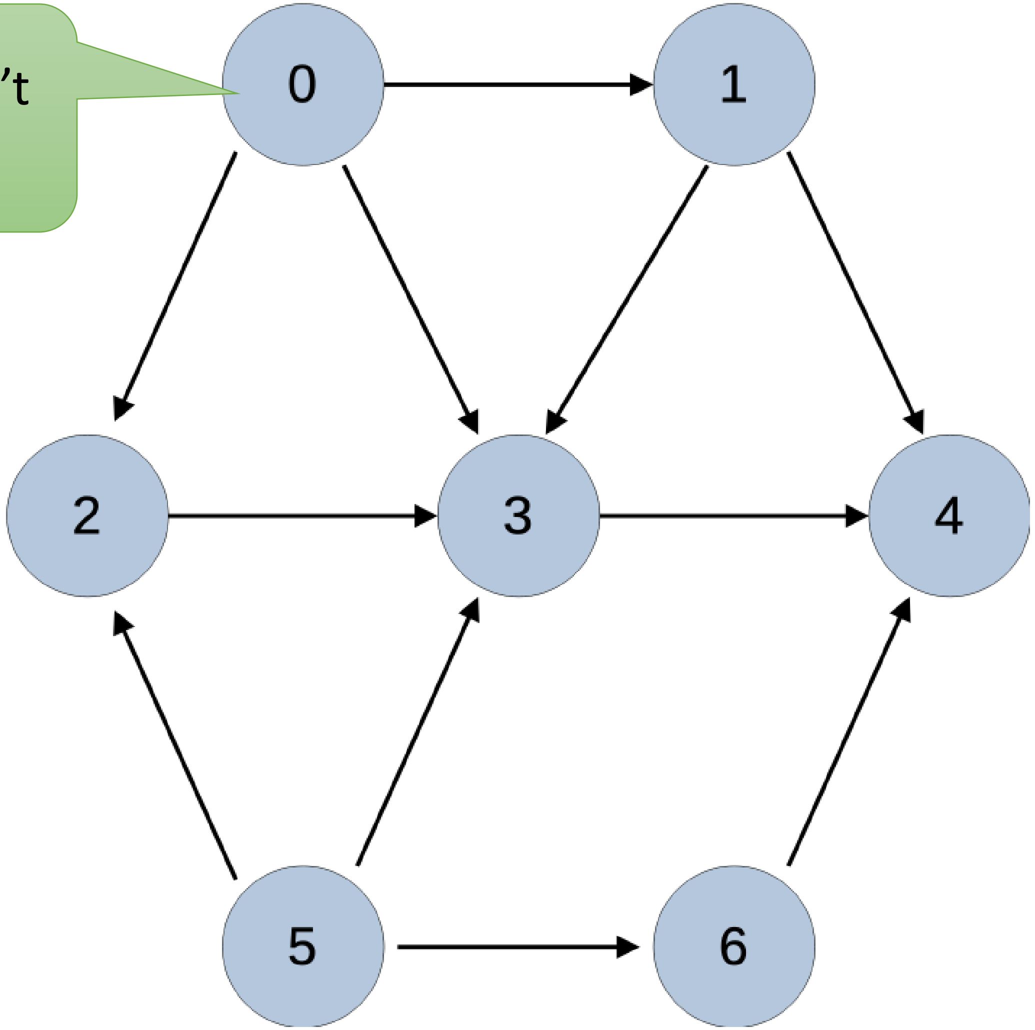
Depth-First Traversal



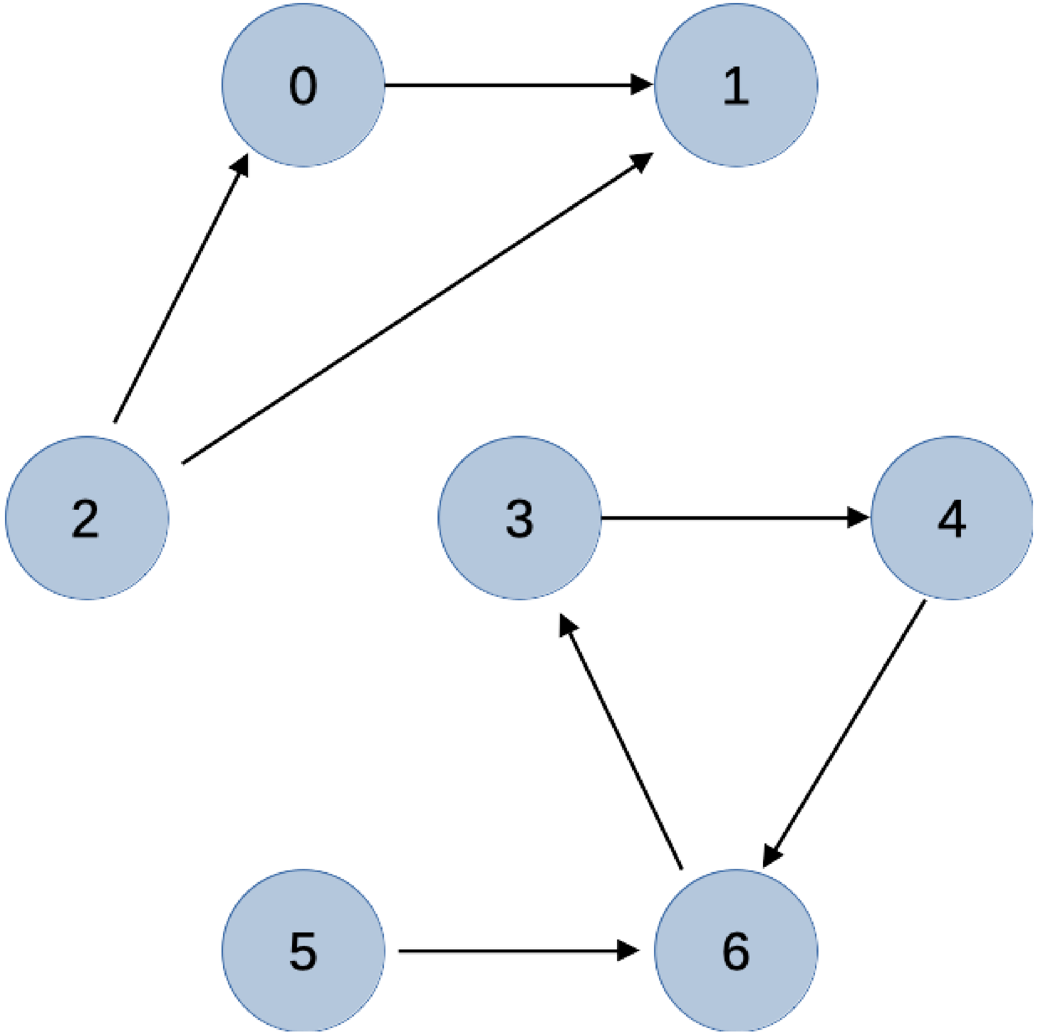
Starting here will reach every vertex.

Where to start?

Now, starting here won't reach 5 and 6.



Multiple Components

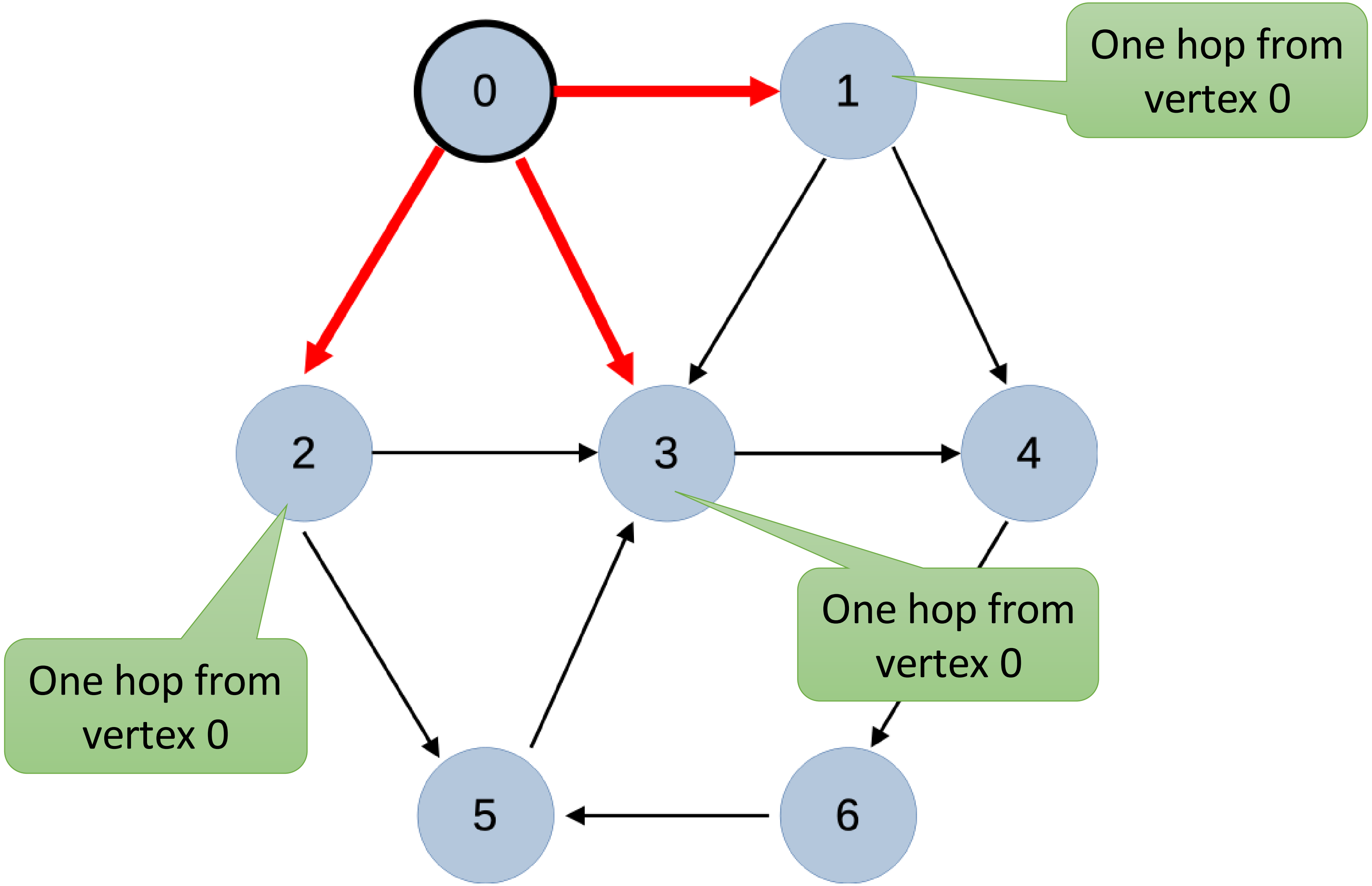


Visiting Multiple Components

- Reaching the whole graph may require multiple traversals.
- Maintaining the state will let us avoid duplicate traversal.

```
state = [ 0, 0, … 0 ];     // Haven’t explored any vertex yet.

for ( i = 0; i < n; i++ )  // Try a traversal starting from any vertex.
  traverse( i, state );
```

Breadth-First Order

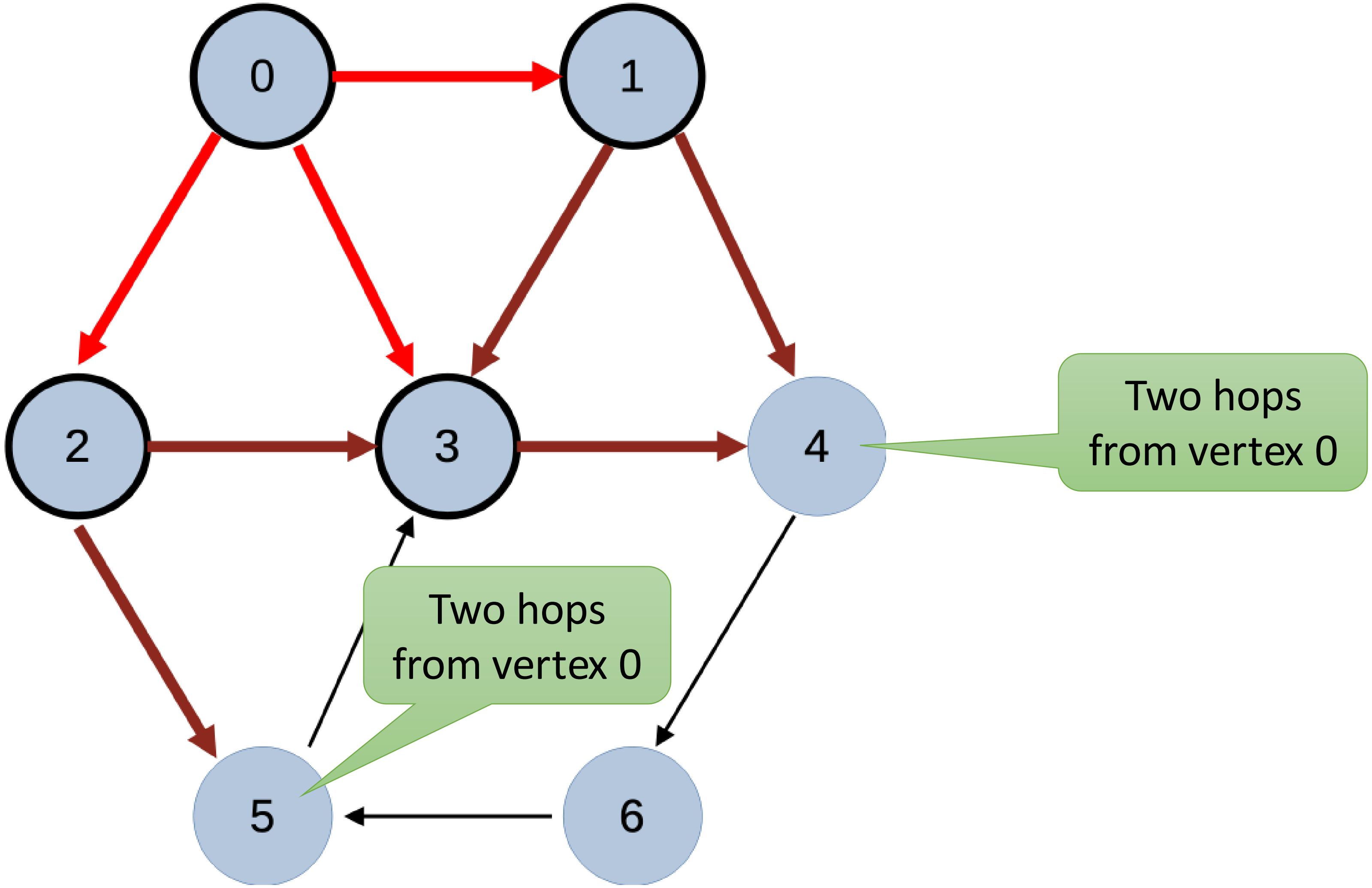


One hop from vertex 0

One hop from vertex 0

One hop from vertex 0

Breadth-First Order



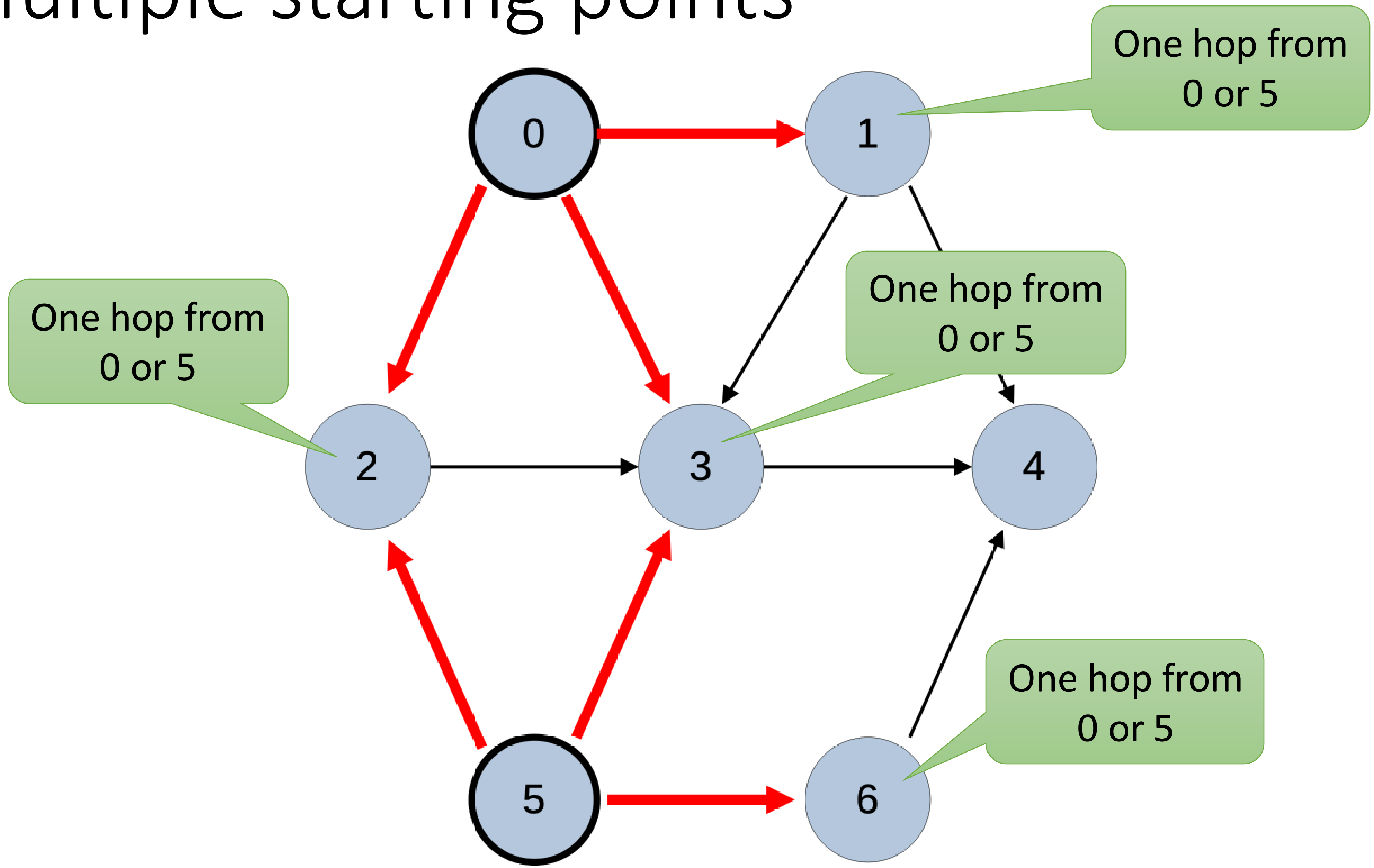
Two hops from vertex 0

Two hops from vertex 0

Breadth-First Traversal

```
distance = [ -1, -1, -1 … -1 ];     // Distance to each vertex we reach.

queue< int > Q;

Q.add_back( 0 );                    // Start at vertex 0
distance[ 0 ] = 0;

while ( Q.size() ) {
  int i = Q.remove_front();
  for ( j : neighbors[ i ] )
    if ( distance[ j ] < 0 ) {      // First time we’ve found this vertex?
      distance[ j ] = distance[ j ] + 1;  // One hop farther than i.
      Q.add_back( j );                    // Explore from j (later)
    }
}
```

Multiple starting points



One hop from 0 or 5

One hop from 0 or 5

One hop from 0 or 5

One hop from 0 or 5