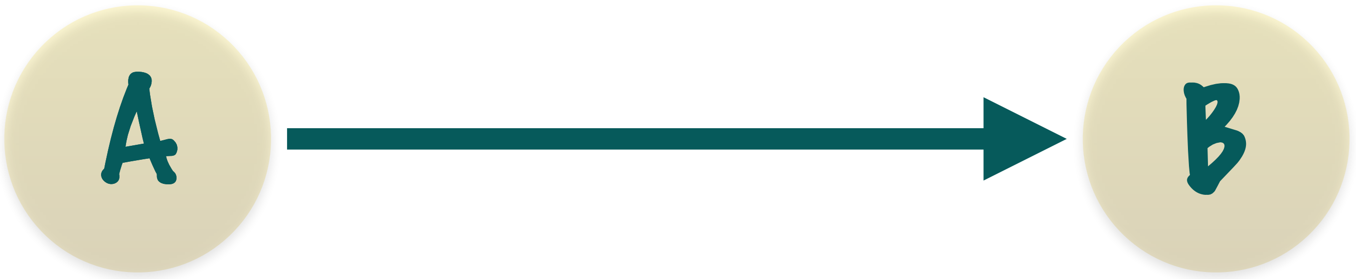
THE GRAPH

TOPOLOGICAL SORT

TOPOLOGICAL SORT

IT IS AN ORDERING OF VERTICES IN A DIRECTED ACYCLIC GRAPH IN WHICH EACH NODE COMES BEFORE ALL THE NODES TO WHICH

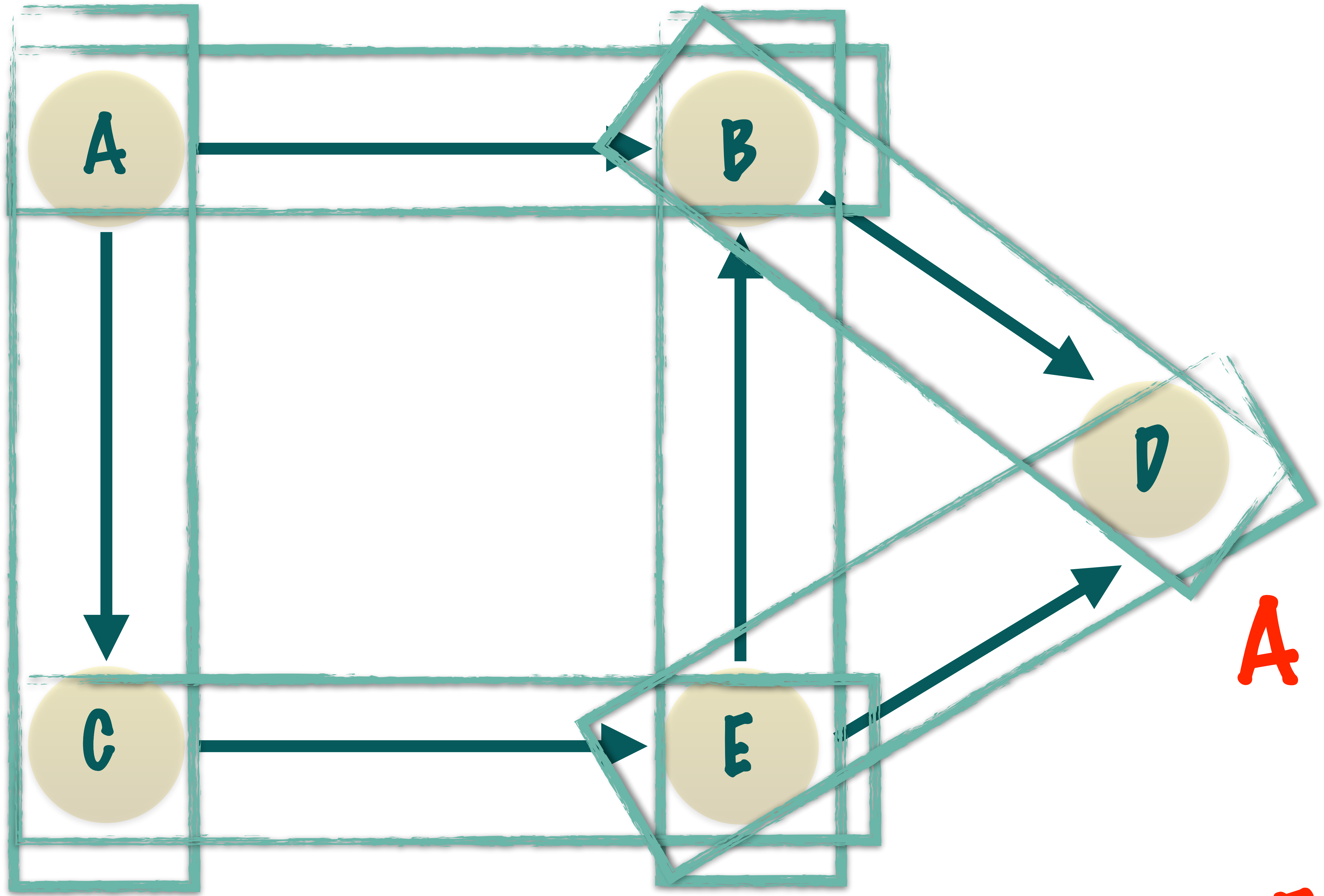
IT HAS OUTGOING EDGES



A SHOULD COME BEFORE B

THE GRAPH

TOPOLOGICAL SORT



IT IS AN ORDERING OF VERTICES IN A DIRECTED ACYCLIC GRAPH IN WHICH EACH NODE COMES BEFORE ALL THE NODES TO WHICH

IT HAS OUTGOING EDGES

A SHOULD COME BEFORE B AND C

C SHOULD COME BEFORE E

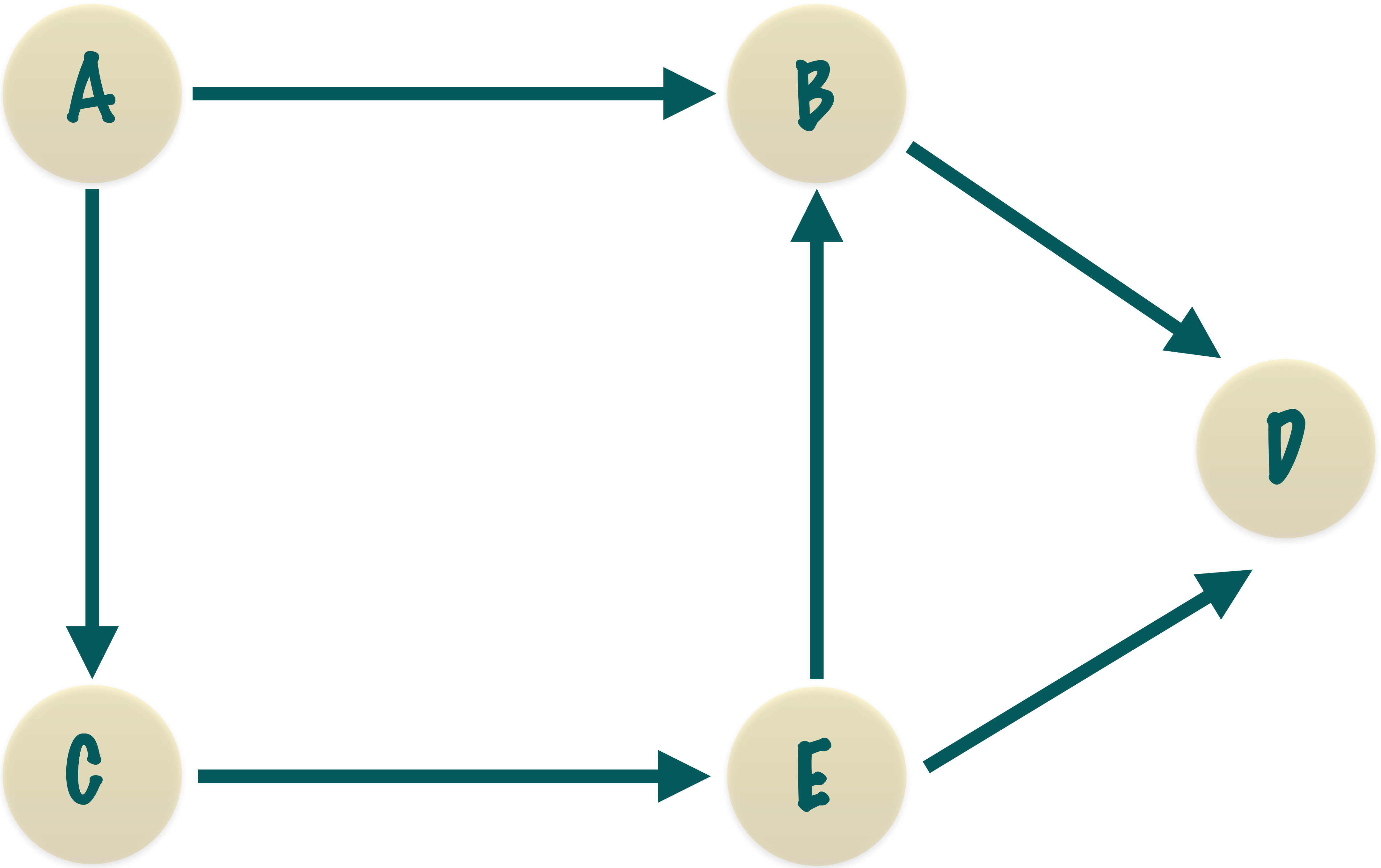
E SHOULD COME BEFORE B AND D

B SHOULD COME BEFORE D

THE GRAPH

TOPOLOGICAL SORT

IT IS AN ORDERING OF VERTICES IN A DIRECTED ACYCLIC GRAPH IN WHICH EACH NODE COMES BEFORE ALL THE NODES TO WHICH IT HAS OUTGOING EDGES



TOPOLOGICAL SORT FOR THIS GRAPH WILL BE:

A, C, E, B, D

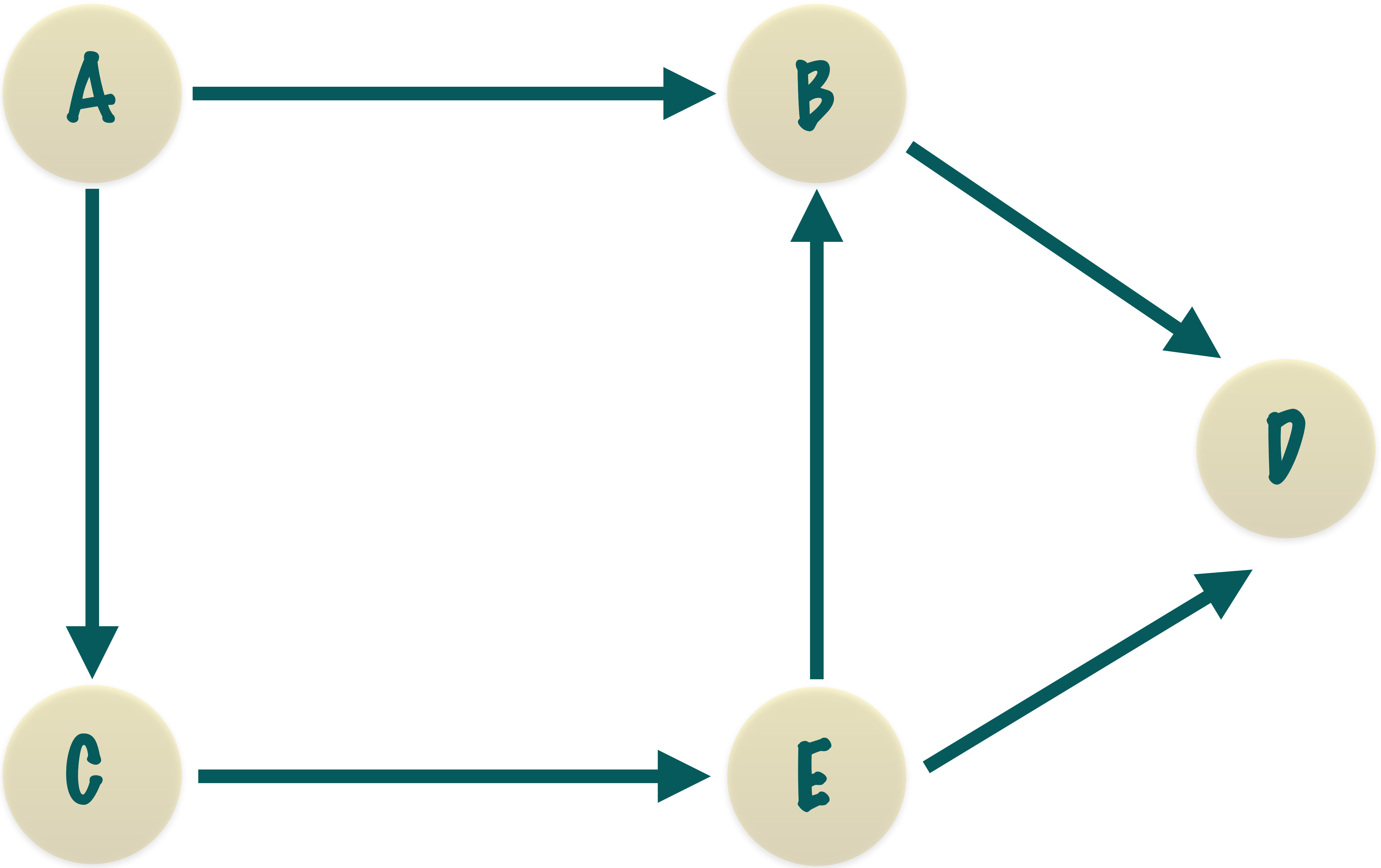
HOW?

*A GRAPH CAN HAVE MULTIPLE TOPOLOGICAL SORTS BUT THIS ONE HAS ONLY ONE

THE GRAPH

TOPOLOGICAL SORT

IT IS AN ORDERING OF VERTICES IN A DIRECTED ACYCLIC GRAPH IN WHICH EACH NODE COMES BEFORE ALL THE NODES TO WHICH IT HAS OUTGOING EDGES



TOPOLOGICAL SORT FOR THIS GRAPH WILL BE:

A, C, E, B, D

HOW?

A SHOULD COME BEFORE B AND C

C SHOULD COME BEFORE E

E SHOULD COME BEFORE B AND D

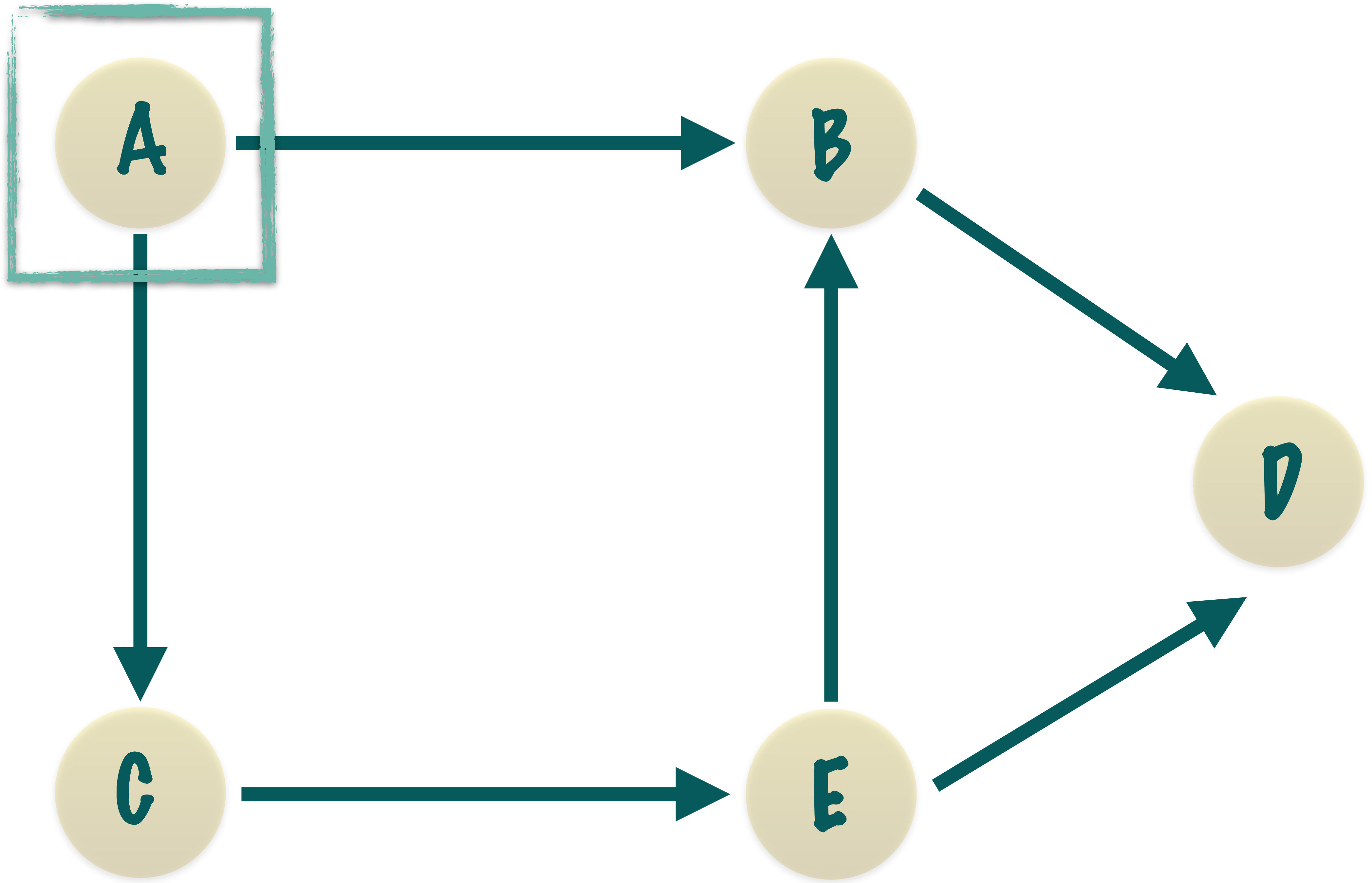
B SHOULD COME BEFORE D

THE GRAPH

TOPOLOGICAL SORT

IT IS AN ORDERING OF VERTICES IN A DIRECTED ACYCLIC GRAPH IN WHICH EACH NODE COMES BEFORE ALL THE NODES TO WHICH IT HAS OUTGOING EDGES

TOPOLOGICAL SORT FOR THIS GRAPH WILL BE: A,C,E,B,D



WE FIRST FIND A VERTEX WHICH HAS NO INCOMING EDGE

(IT IS THE DESTINATION OF NO EDGE)
(NO ARROW POINTS TO IT)

A IS THE ONLY VERTEX WITH NO INCOMING EDGE - THIS IS THE FIRST ELEMENT OF THE SORT!

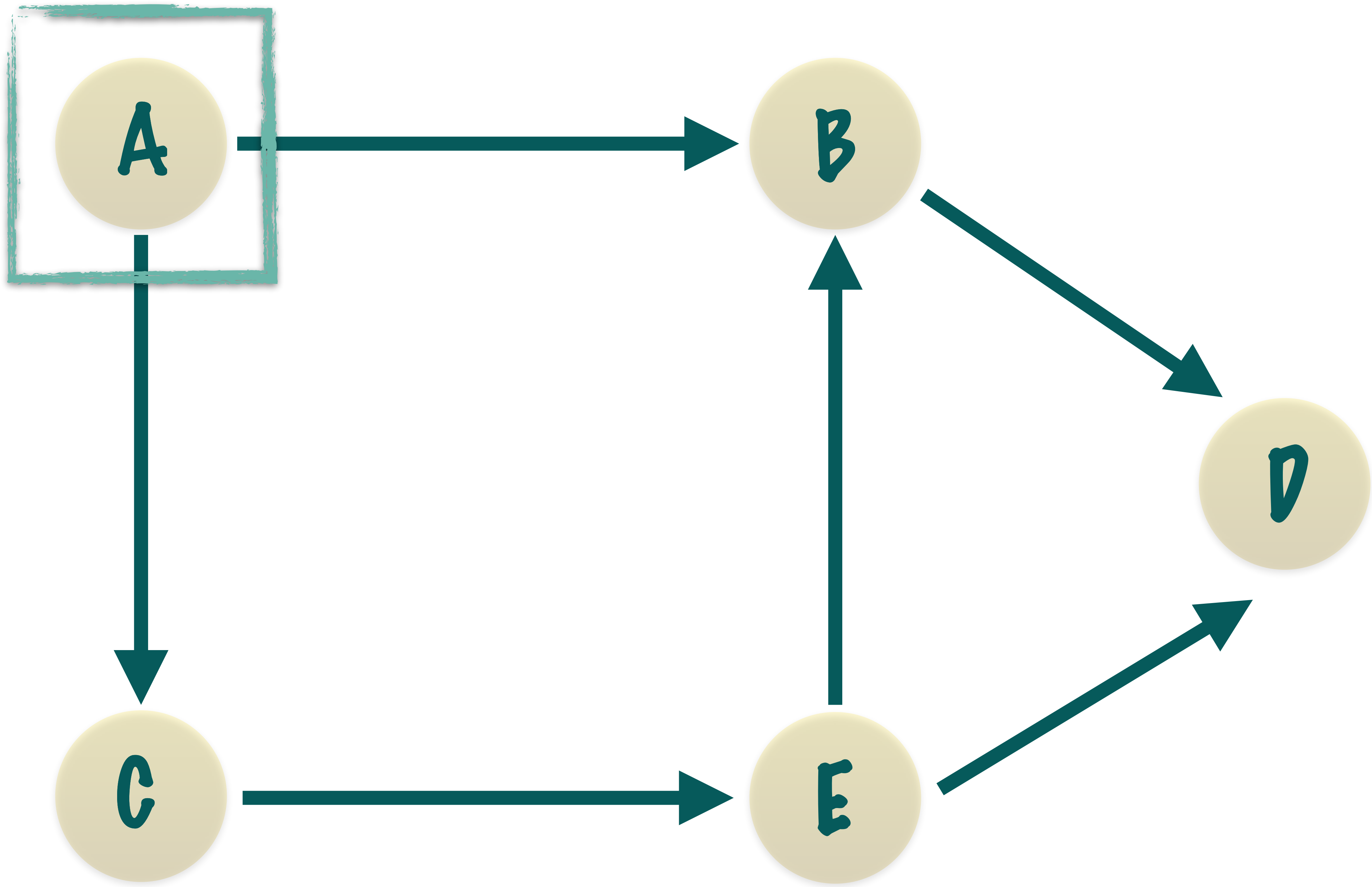
THE GRAPH

TOPOLOGICAL SORT

IT IS AN ORDERING OF VERTICES IN A DIRECTED ACYCLIC GRAPH IN WHICH EACH NODE COMES BEFORE ALL THE NODES TO WHICH IT HAS OUTGOING EDGES

TOPOLOGICAL SORT FOR THIS GRAPH WILL BE: A,C,E,B,D

A IS THE ONLY VERTEX WITH NO INCOMING EDGE - THIS IS THE FIRST ELEMENT OF THE SORT!



INDEGREE

NUMBER OF INWARD DIRECTED GRAPH EDGES FOR A GIVEN GRAPH VERTEX

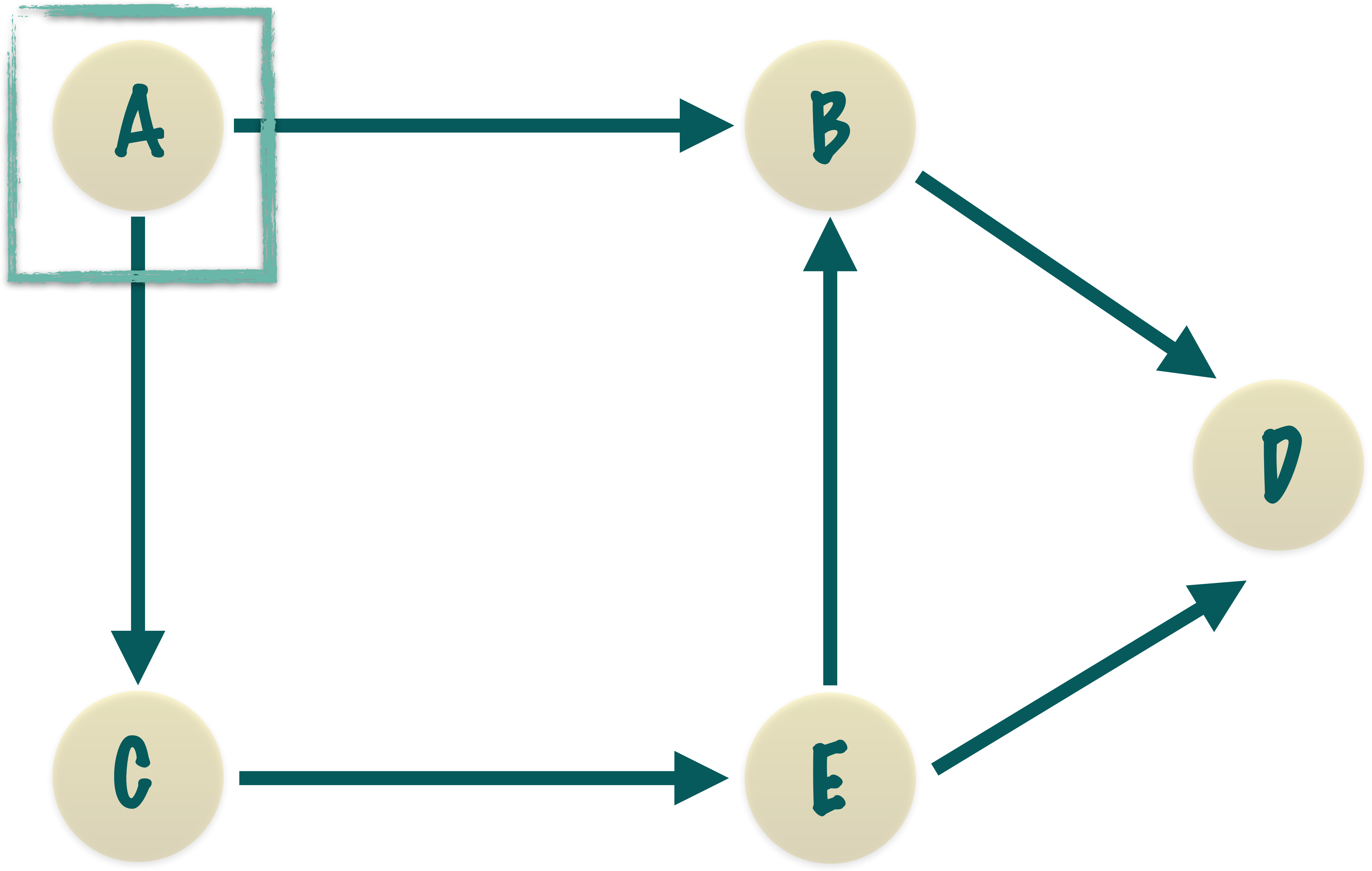
INDEGREE OF A IS 0!

THE GRAPH

TOPOLOGICAL SORT

IT IS AN ORDERING OF VERTICES IN A DIRECTED ACYCLIC GRAPH IN WHICH EACH NODE COMES BEFORE ALL THE NODES TO WHICH IT HAS OUTGOING EDGES

TOPOLOGICAL SORT FOR THIS GRAPH WILL BE: A,C,E,B,D



INDEGREE OF A IS 0!

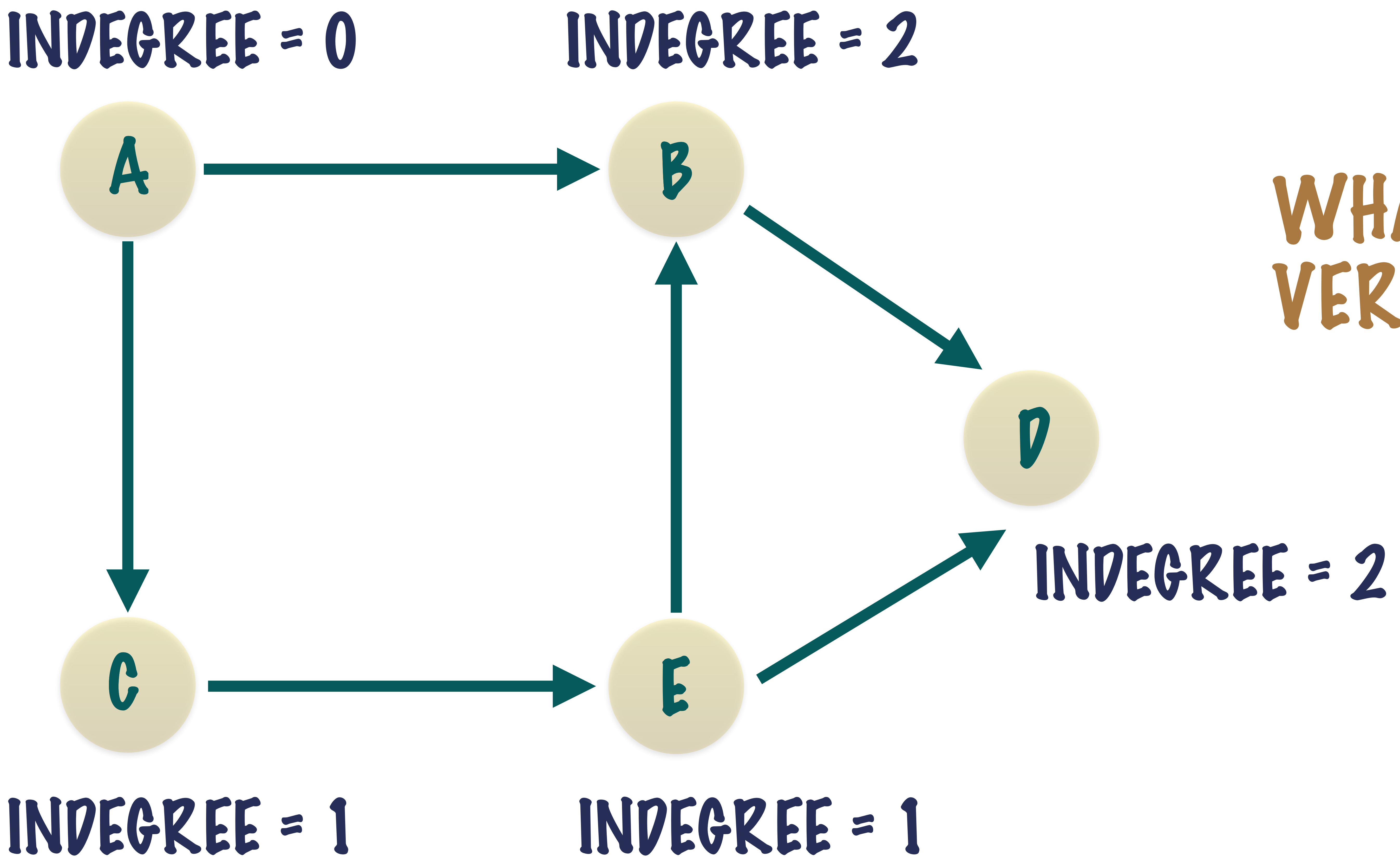
IF THERE WERE NO VERTICES WITH 0 INDEGREE, THEN THERE WOULD HAVE BEEN NO TOPOLOGICAL SORT

THE GRAPH HAS A CYCLE!

THE GRAPH

TOPOLOGICAL SORT

A,C,E,B,D

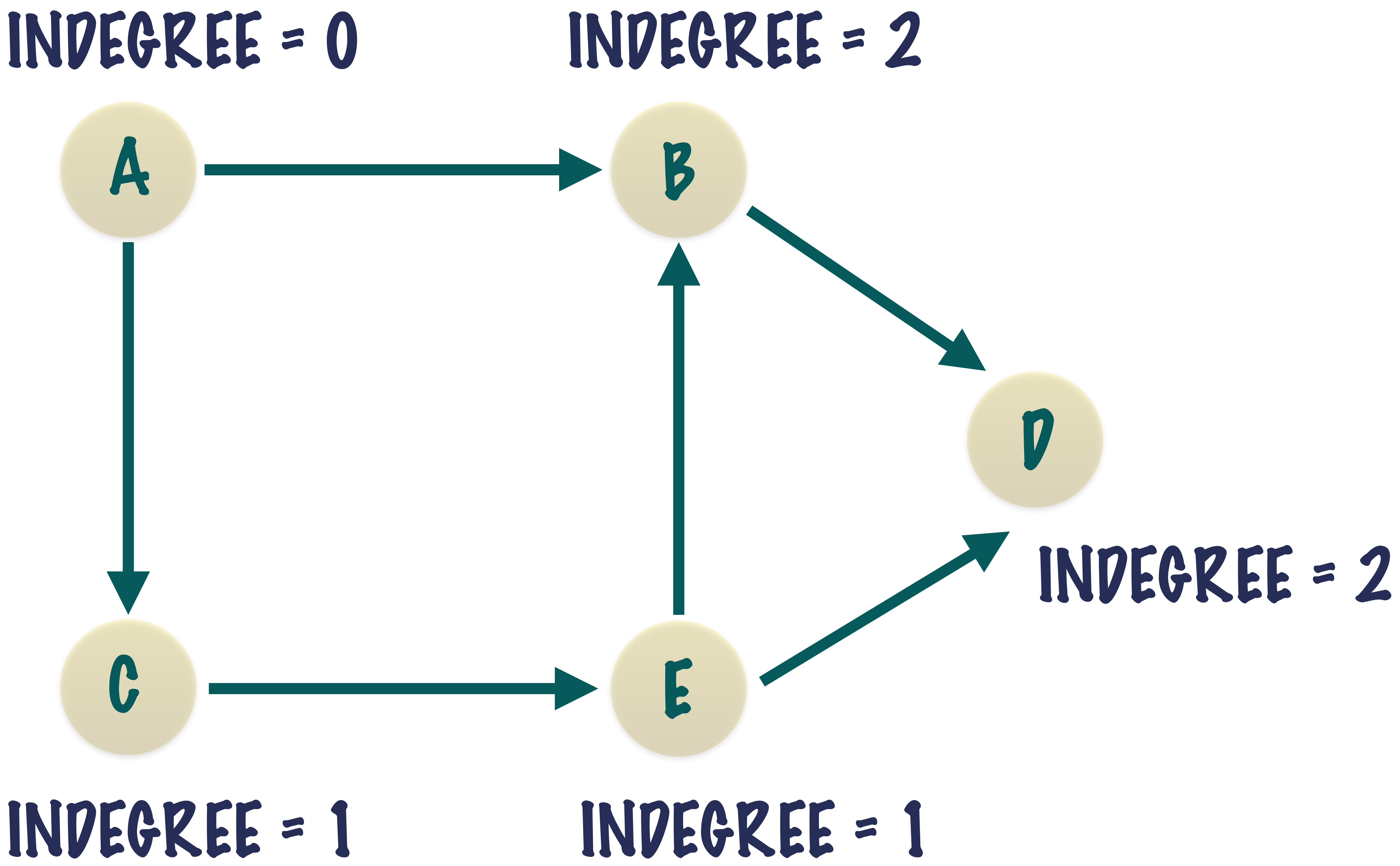


WHAT IS THE INDEGREE OF EACH VERTEX IN THIS GRAPH?

THE GRAPH

TOPOLOGICAL SORT

A,C,E,B,D



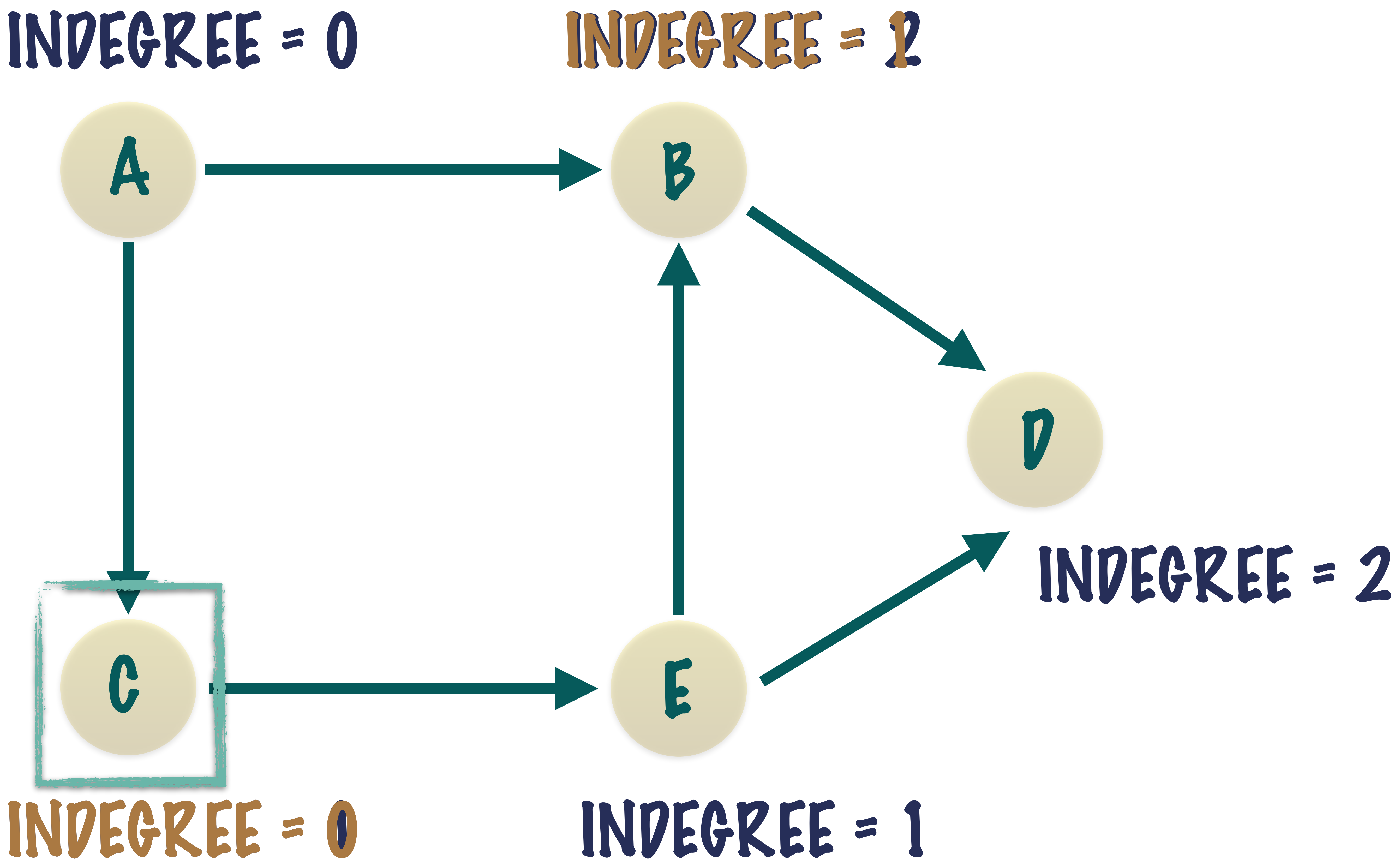
WE NOW KNOW A IS THE FIRST ELEMENT IN OUR SORT

IF WE "REMOVE" A FROM THIS GRAPH, WE HAVE TO REDUCE THE INDEGREE OF ALL ITS IMMEDIATE NEIGHBOURS

TOPOLOGICAL SORT

THE GRAPH

A,C,E,B,D



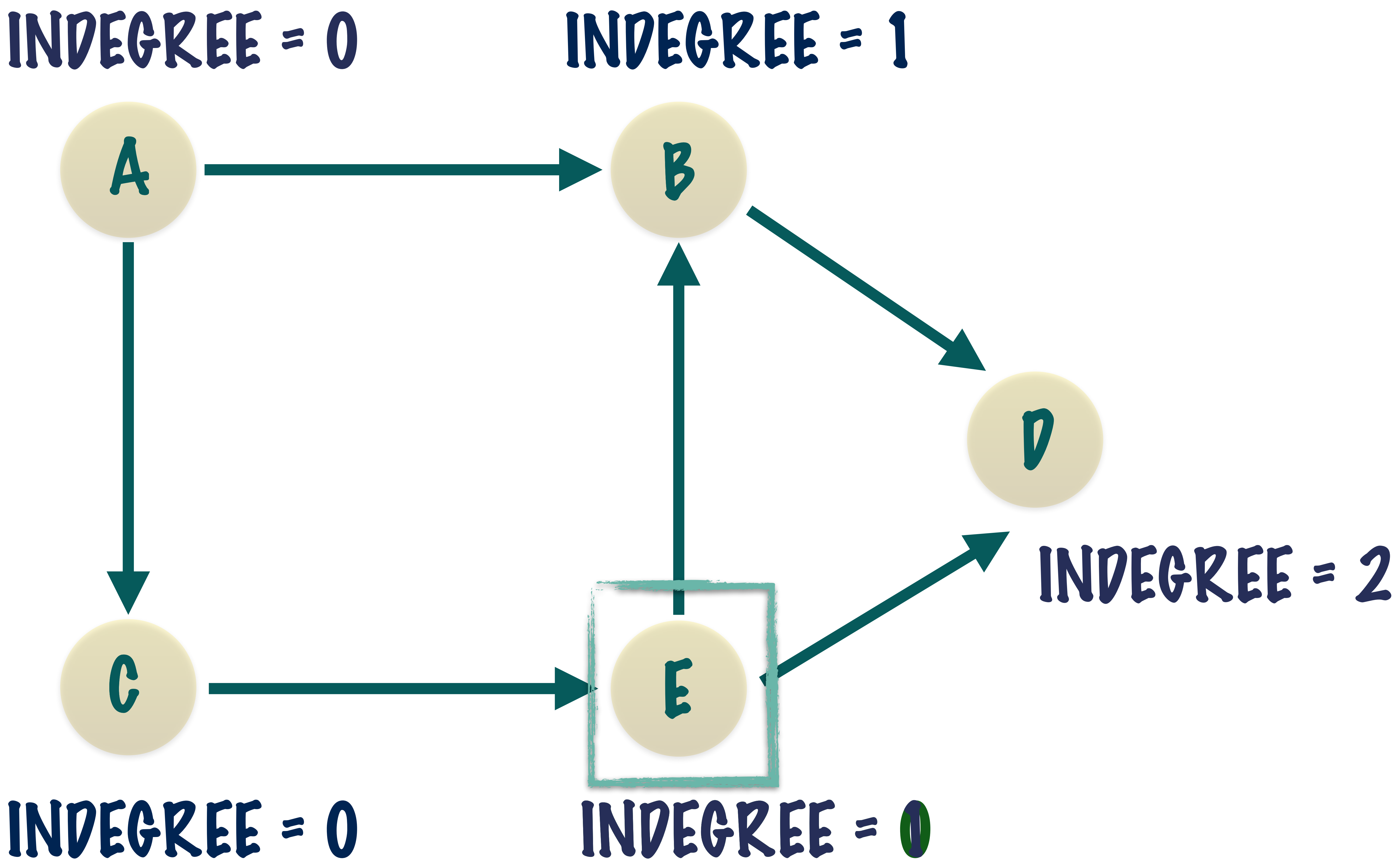
AFTER FINDING "A", WE DECREMENT INDEGREE OF ITS NEIGHBOURS BY 1

THE NEXT VERTEX IN THIS SORT THE ONE WITH INDEGREE 0

C IS THE NEXT ELEMENT!

TOPOLOGICAL SORT

THE GRAPH



A,C,E,B,D

WE "REMOVE" C FROM THE GRAPH AND DECREMENT THE INDEGREE OF ITS IMMEDIATE NEIGHBOURS

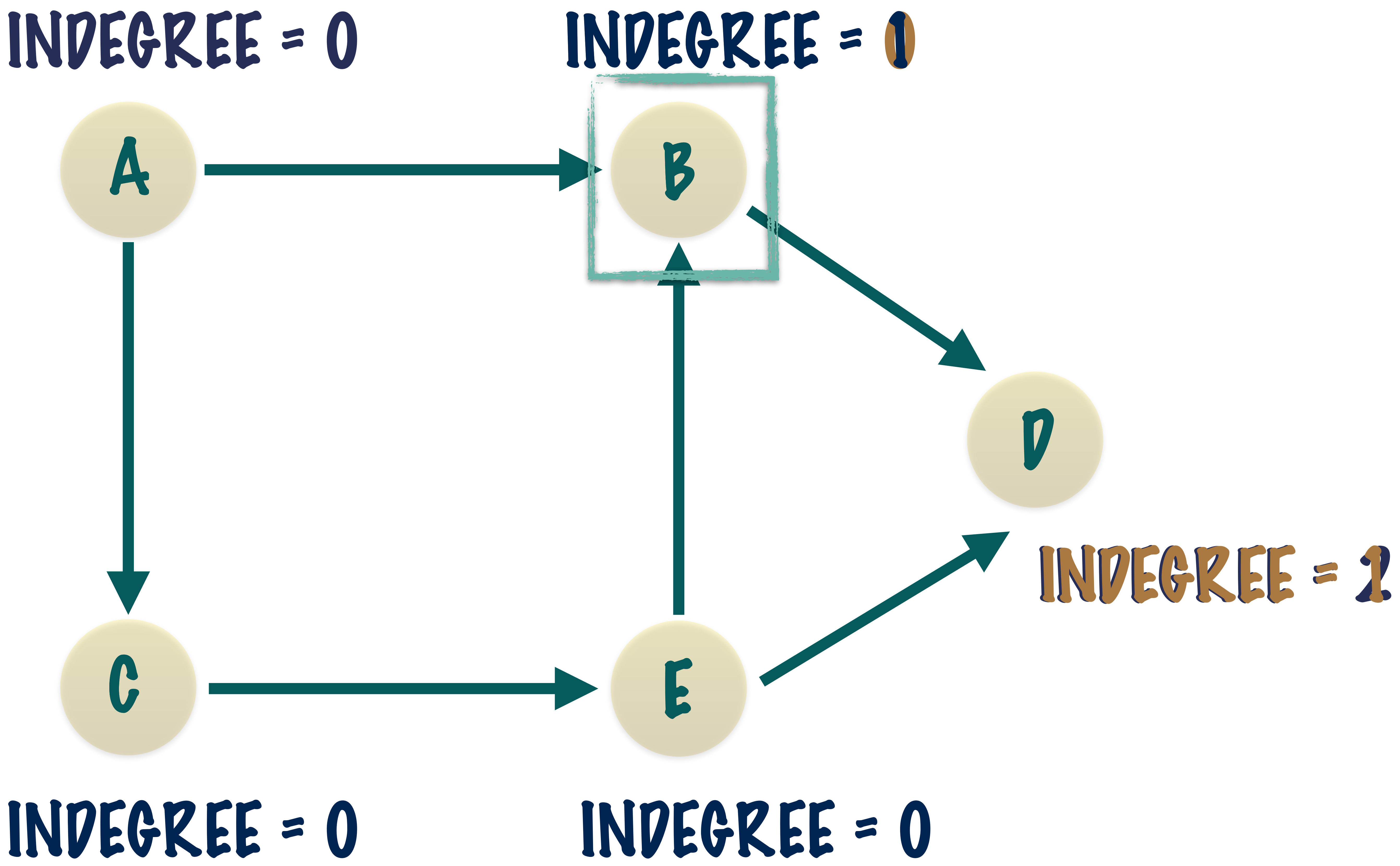
THE NEXT VERTEX IN THIS SORT THE ONE WITH INDEGREE 0

THE NEXT ELEMENT IS E!

THE GRAPH

TOPOLOGICAL SORT

A,C,E,B,D

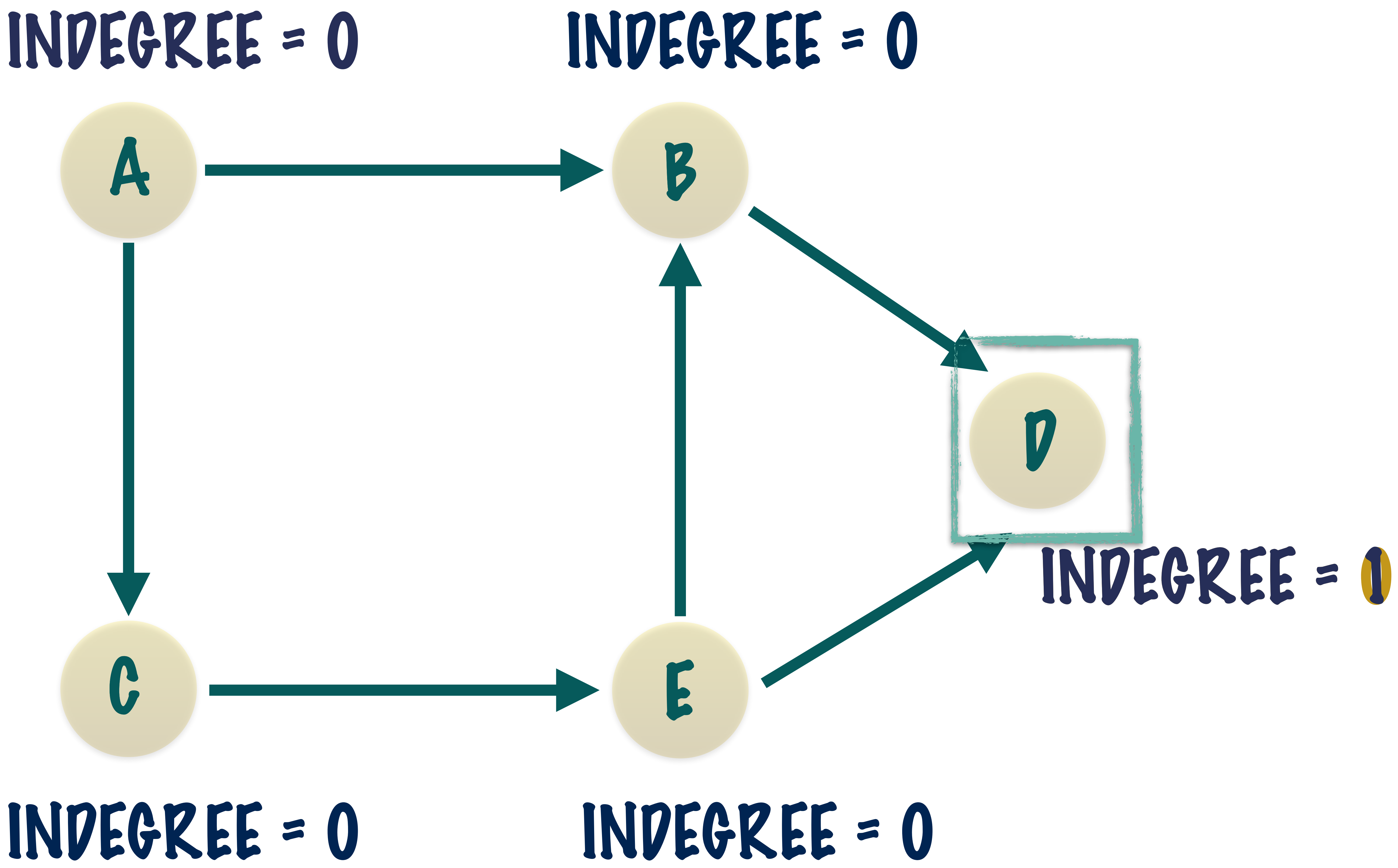


THE NEXT ELEMENT IS B!

THE GRAPH

TOPOLOGICAL SORT

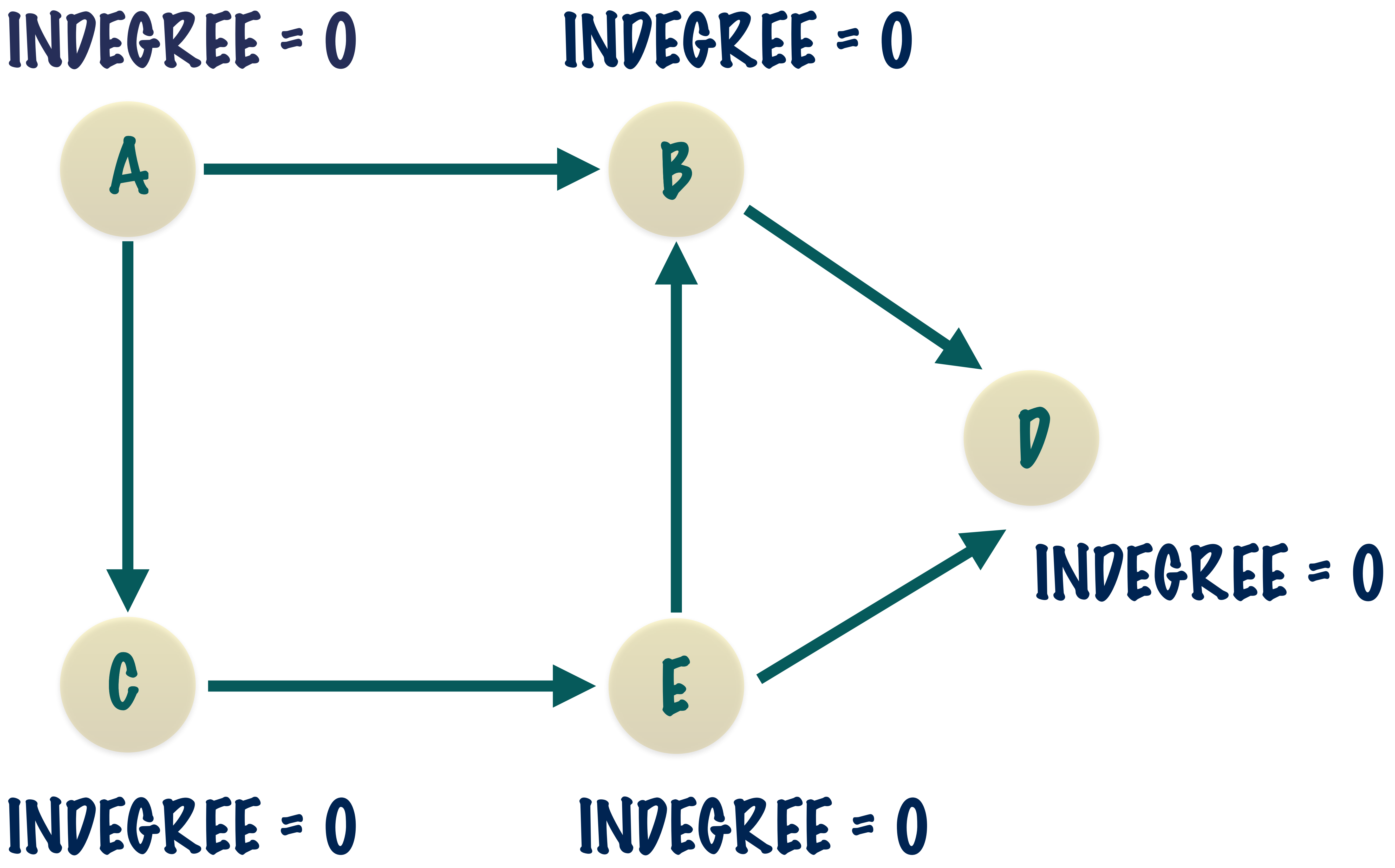
A,C,E,B,D



THE NEXT ELEMENT IS D!

TOPOLOGICAL SORT

THE GRAPH



A,C,E,B,D

RUNNING TIME FOR TOPOLOGICAL SORT IS $O(V+E)$

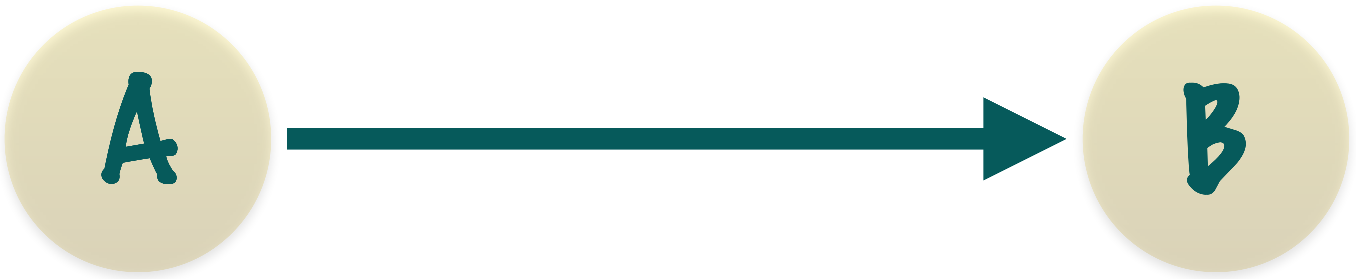
EVERY EDGE AND EVERY VERTEX IS VISITED ONCE

THE GRAPH

TOPOLOGICAL SORT

IT IS AN ORDERING OF VERTICES IN A DIRECTED ACYCLIC GRAPH IN WHICH EACH NODE COMES BEFORE ALL THE NODES TO WHICH

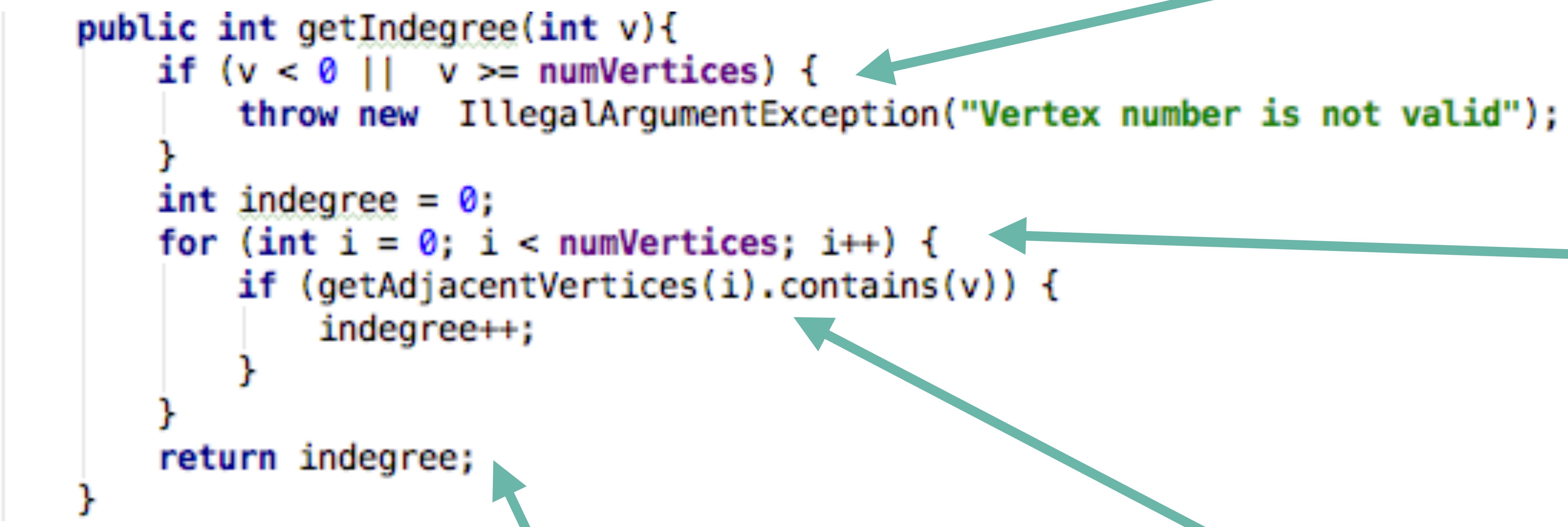
IT HAS OUTGOING EDGES



A SHOULD COME BEFORE B

INDEGREE IN ADJACENCY LIST

CHECK THAT THE VERTEX IS VALID



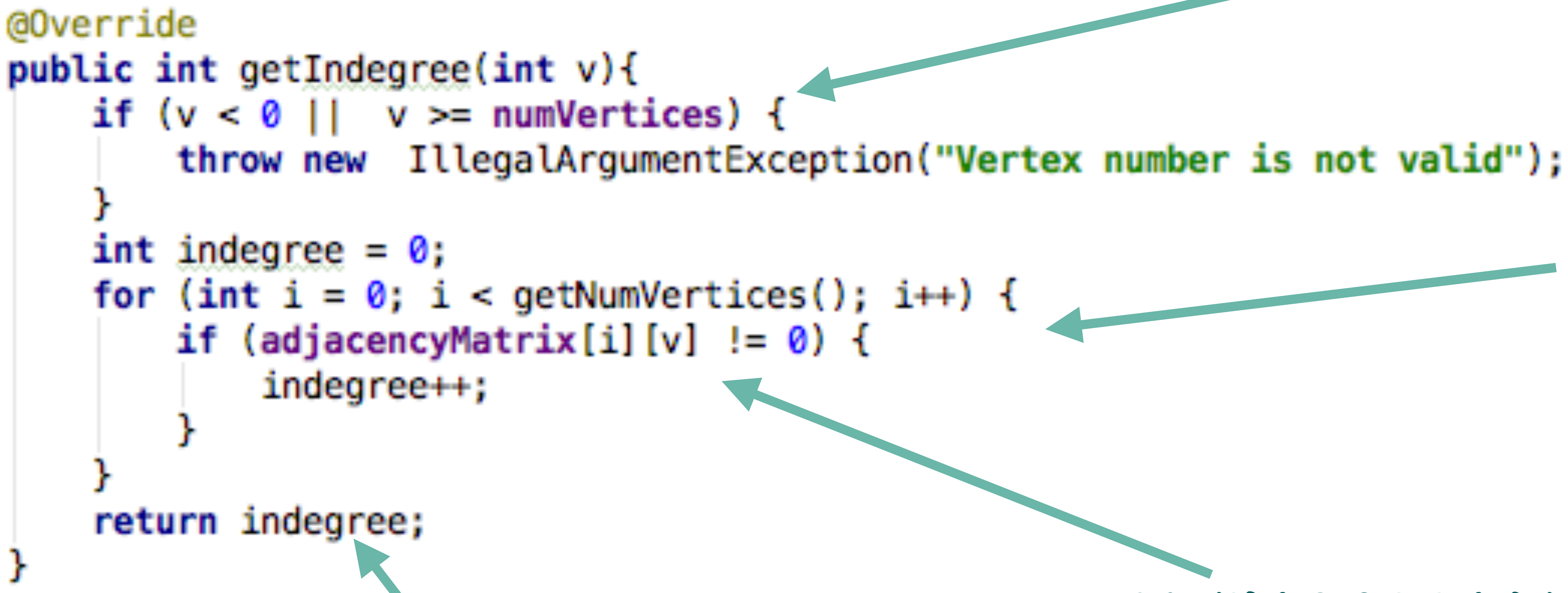
```java
public int getIndegree(int v){
    if (v < 0 ||  v >= numVertices) {
        throw new  IllegalArgumentException("Vertex number is not valid");
    }
    int indegree = 0;
    for (int i = 0; i < numVertices; i++) {
        if (getAdjacentVertices(i).contains(v)) {
            indegree++;
        }
    }
    return indegree;
}
```

ITERATE THROUGH ALL THE VERTICES IN THE GRAPH

IF THE CURRENT VERTEX IS PRESENT AS AN ADJACENT VERTEX FOR ANY OTHER VERTEX THEN INCREMENT THE INDEGREE COUNT FOR THE CURRENT VERTEX

RETURN THE INDEGREE COUNT

INDEGREE IN ADJACENCY GRAPH



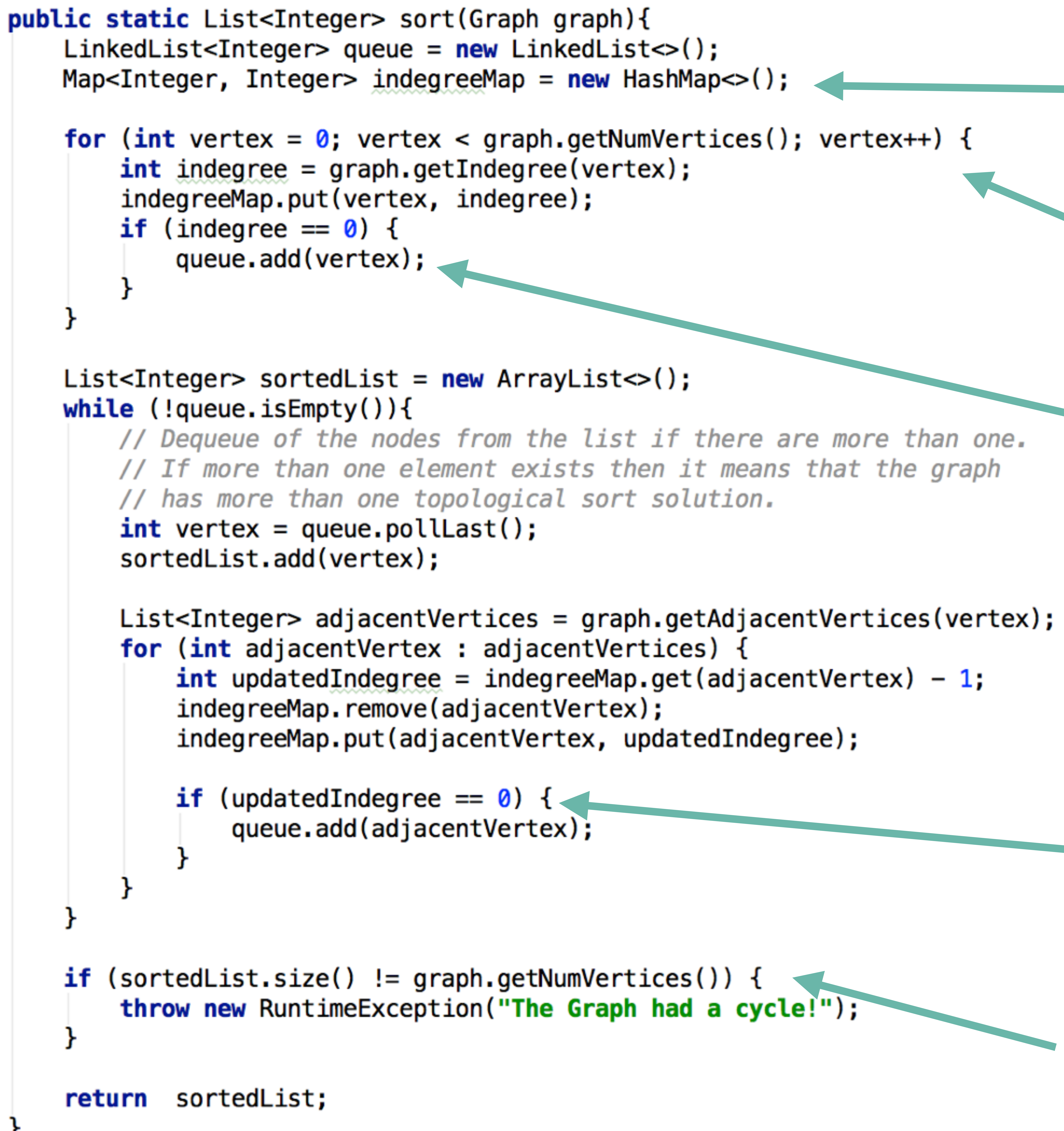
```java
@Override
public int getIndegree(int v){
    if (v < 0 ||  v >= numVertices) {
        throw new  IllegalArgumentException("Vertex number is not valid");
    }
    int indegree = 0;
    for (int i = 0; i < getNumVertices(); i++) {
        if (adjacencyMatrix[i][v] != 0) {
            indegree++;
        }
    }
    return indegree;
}
```

CHECK THAT THE VERTEX IS VALID

ITERATE THROUGH ALL THE VERTICES IN THE GRAPH

IF THE CURRENT VERTEX IS PRESENT AS AN ADJACENT VERTEX FOR ANY OTHER VERTEX THEN INCREMENT THE INDEGREE COUNT FOR THE CURRENT VERTEX

RETURN THE INDEGREE COUNT



```java
public static List<Integer> sort(Graph graph){
    LinkedList<Integer> queue = new LinkedList<>();
    Map<Integer, Integer> indegreeMap = new HashMap<>();

    for (int vertex = 0; vertex < graph.getNumVertices(); vertex++) {
        int indegree = graph.getIndegree(vertex);
        indegreeMap.put(vertex, indegree);
        if (indegree == 0) {
            queue.add(vertex);
        }
    }

    List<Integer> sortedList = new ArrayList<>();
    while (!queue.isEmpty()){
        // Dequeue of the nodes from the list if there are more than one.
        // If more than one element exists then it means that the graph
        // has more than one topological sort solution.
        int vertex = queue.pollLast();
        sortedList.add(vertex);

        List<Integer> adjacentVertices = graph.getAdjacentVertices(vertex);
        for (int adjacentVertex : adjacentVertices) {
            int updatedIndegree = indegreeMap.get(adjacentVertex) - 1;
            indegreeMap.remove(adjacentVertex);
            indegreeMap.put(adjacentVertex, updatedIndegree);

            if (updatedIndegree == 0) {
                queue.add(adjacentVertex);
            }
        }
    }

    if (sortedList.size() != graph.getNumVertices()) {
        throw new RuntimeException("The Graph had a cycle!");
    }

    return  sortedList;
}
```

STORES A MAPPING OF A VERTEX TO ITS INDEGREE

INITIALIZE THE INDEGREE MAP BY ITERATING THROUGH ALL VERTICES

ADD ALL VERTICES WITH INDEGREE = 0 TO THE QUEUE OF VERTICES TO EXPLORE

GET THE ADJACENT VERTICES OF THE CURRENT ONE AND DECREMENT THEIR INDEGREES BY 1

FOR EVERY VERTEX WHICH NOW HAS INDEGREE = 0 IT'S A POTENTIAL NEXT NODE FOR THE TOPOLOGICAL SORT

IF THE FINAL SORTED LIST IS NOT EQUAL TO THE NUMBER OF VERTICES IN THE GRAPH THERE IS A CYCLE