CS 112

Part 1: Topological sort using DFS
Part 2: BFS

Priya Govindan

Nov 29, 2016

Rutgers University

Overview

Part 1: Topological sort using DFS

- Revision of DFS
- What is topological sort?
- Algorithm & Code

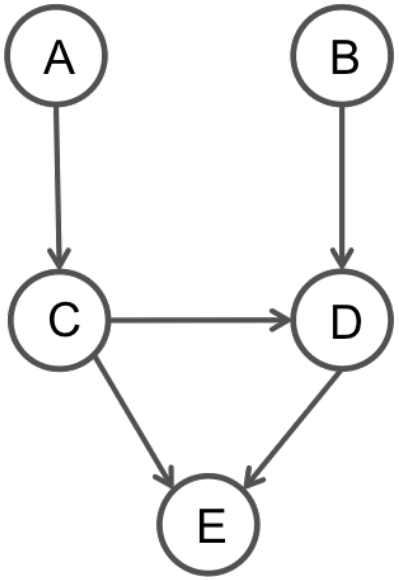
Part 2: BFS

- Algorithm & Code
- Runtime Analysis
- Comparing DFS and BFS

Part 1: Topological sort using DFS

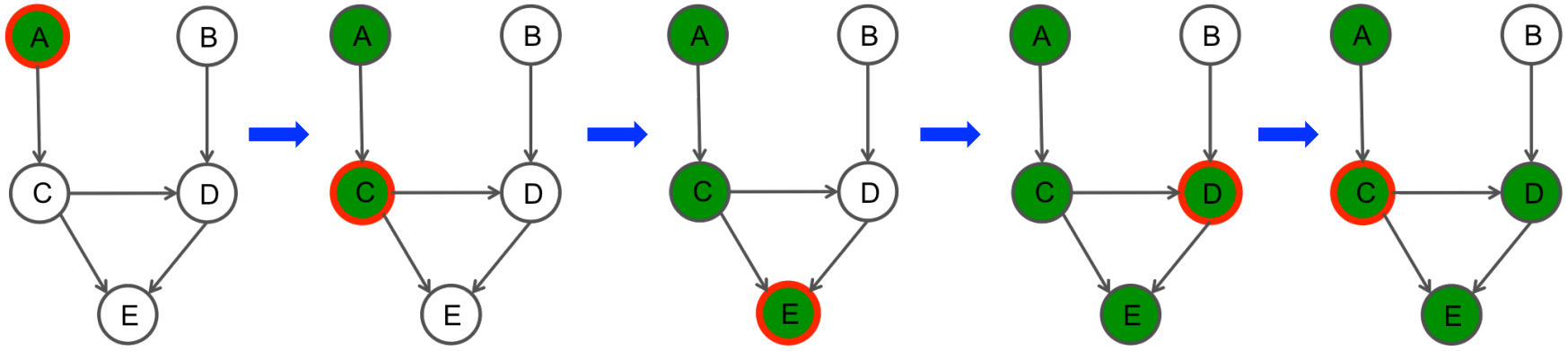
Revision of Depth-First search (DFS)

Given a graph, traverse the graph in a depth-first manner



Note that a graph is more than a tree.

Revision of Depth-First search (DFS)

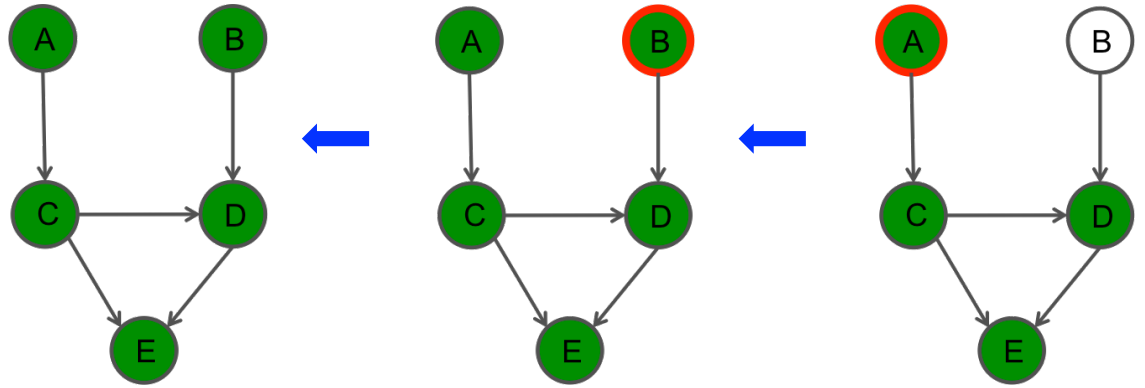


Output: A

Output: A, C

Output: A, C, E

Output: A, C, E, D



Output: A, C, E, D, B

Final Output is
A, C, E, D, B

Not-yet-visited

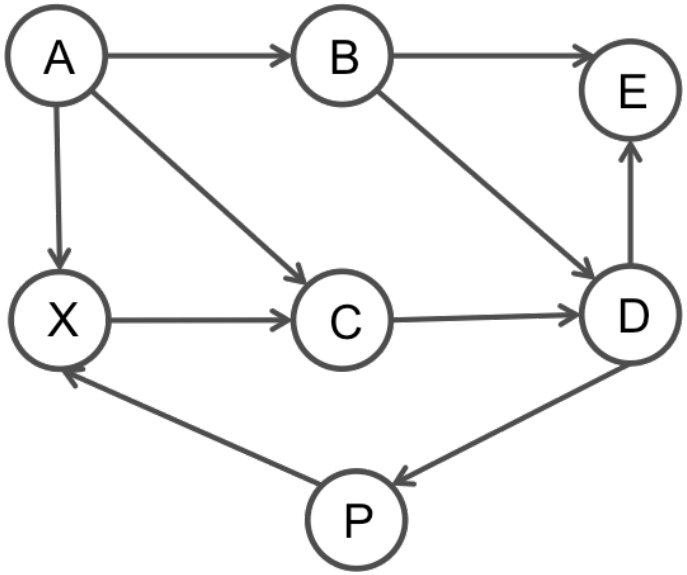
Visited

Current node

Revision of Depth-First search (DFS)

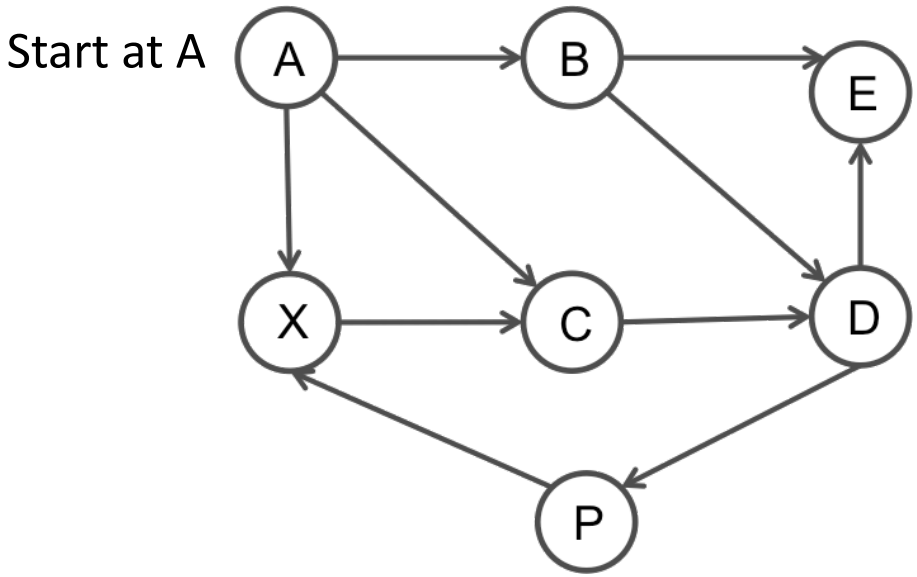
Quiz 1: What is the DFS output for the graph below?

Start at A



Revision of Depth-First search (DFS)

Quiz : What is the DFS output for the graph below?

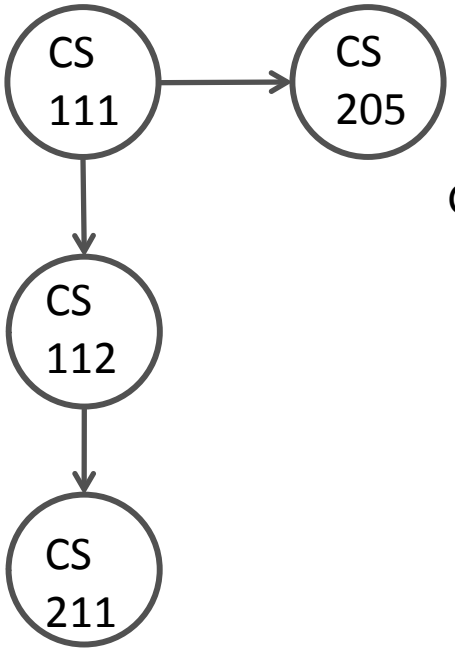


One solution is A, C, D, E, P, X, B

You will get other solutions if you choose to go to B or X from A.

What is topological sort?

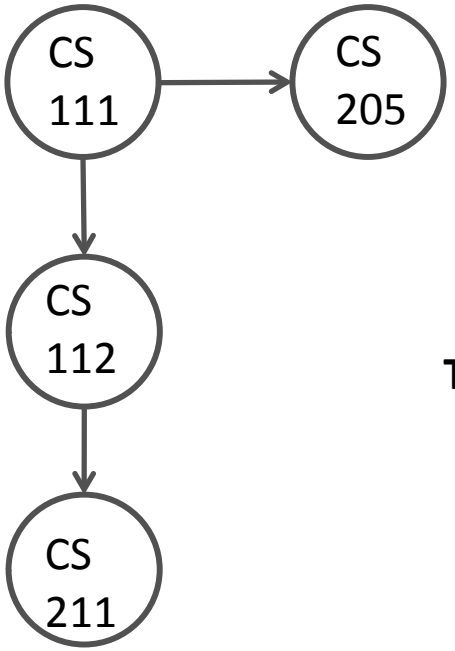
Lets draw a graph of CS courses, such that
Nodes are courses
Edges represent pre-requisite requirement



Given this graph, can you give an order of courses to take?

What is topological sort?

Lets draw a graph of CS courses, such that
Nodes are courses
Edges represent pre-requisite requirement



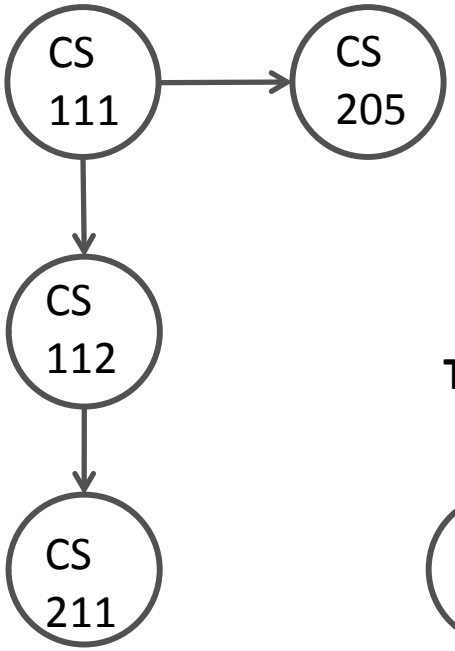
Given this graph, can you give an order of courses to take?
111, 112, 211, 205 OR 111, 205, 112, 211 OR …

This precedence graph is a partial order.
The above ordering is a total order.

Topological sort gives such an ordering of nodes

What is topological sort?

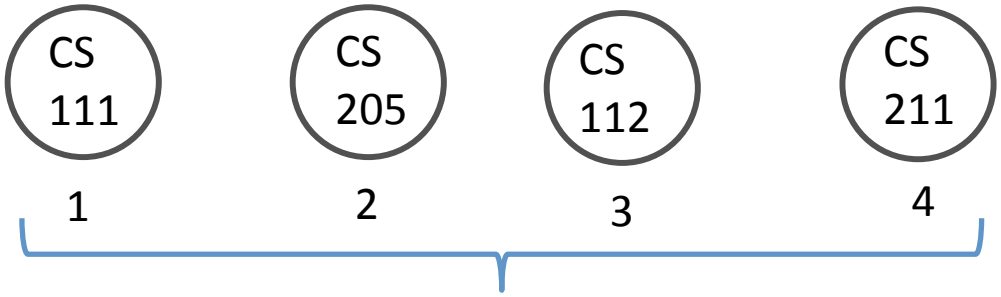
Lets draw a graph of CS courses, such that
Nodes are courses
Edges represent pre-requisite requirement



Given this graph, can you give an order of courses to take?
111, 112, 211, 205 OR 111, 205, 112, 211 OR …

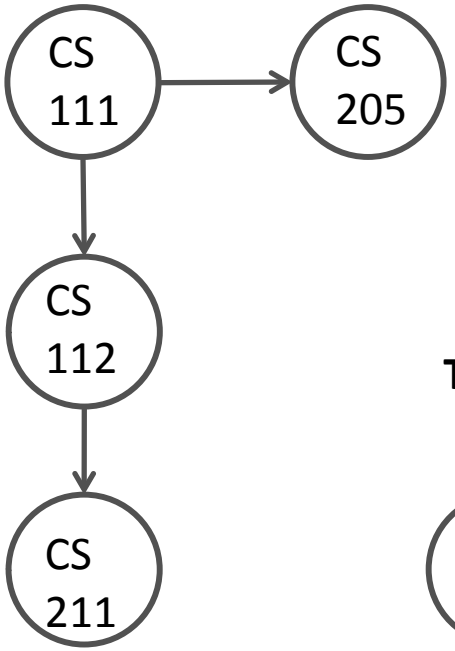
This precedence graph is a partial order.
The above ordering is a total order.

Topological sort gives such an ordering of nodes



What is topological sort?

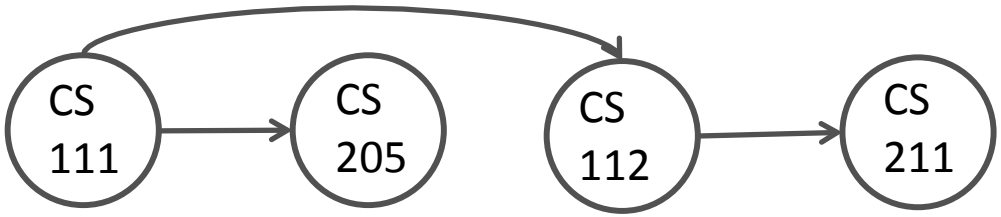
Lets draw a graph of CS courses, such that
Nodes are courses
Edges represent pre-requisite requirement



Given this graph, can you give an order of courses to take?
111, 112, 211, 205 OR 111, 205, 112, 211 OR …

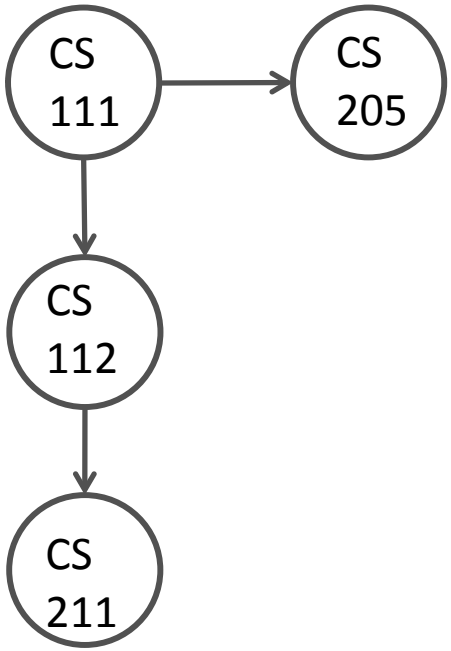
This precedence graph is a partial order.
The above ordering is a total order.

Topological sort gives such an ordering of nodes



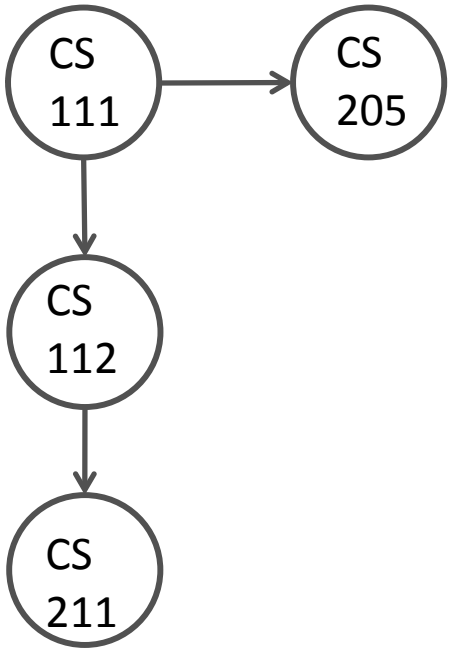
Trick to check correctness of your solution:
In the solution, all edges must go from left to right.

TopSort algorithm using DFS



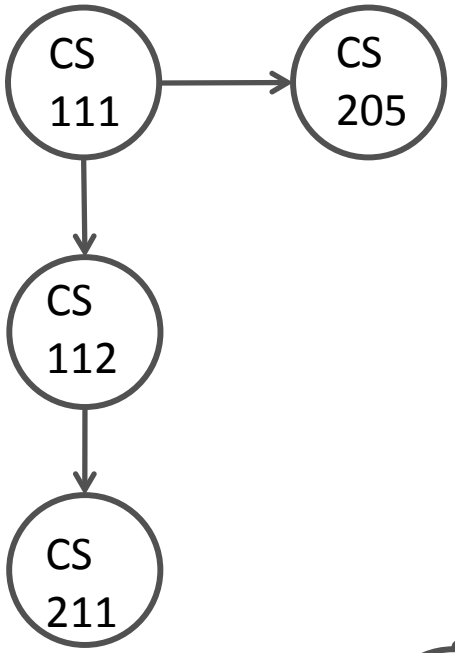
Output from DFS: ?

TopSort algorithm using DFS



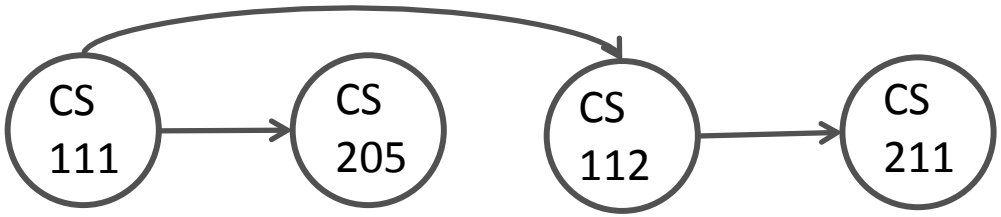
Output from DFS: 111, 205, 112, 211

TopSort algorithm using DFS

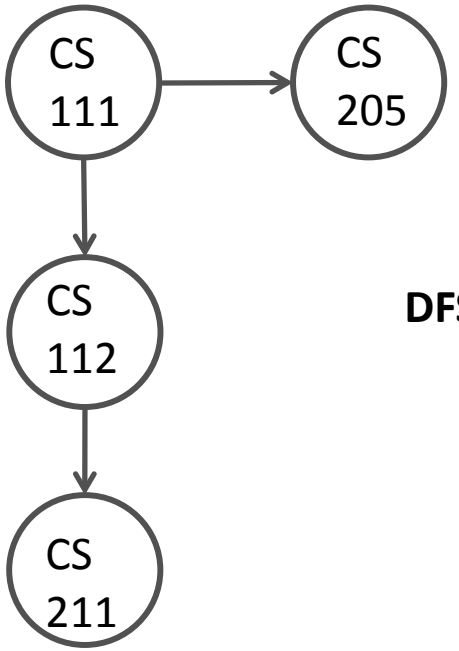


Output from DFS: 111, 205, 112, 211

A legitimate course ordering



TopSort algorithm using DFS

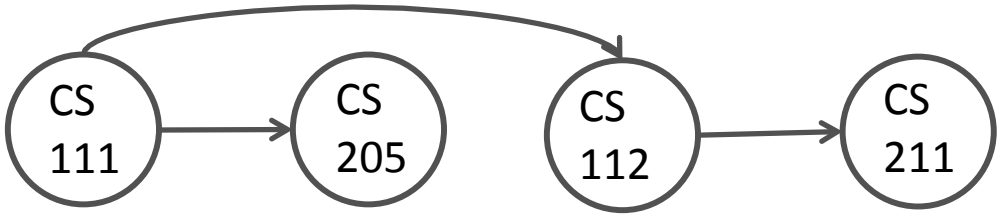


Output from DFS: 111, 205, 112, 211

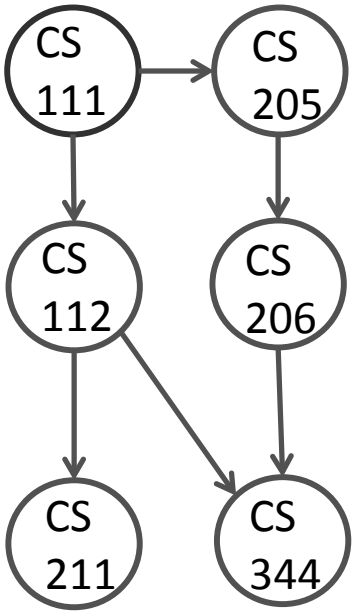
DFS output here gives a topological sort!

Does DFS output always give a topSort? **NO!**
So, how do we use DFS algo to get a topSort?

A legitimate course ordering

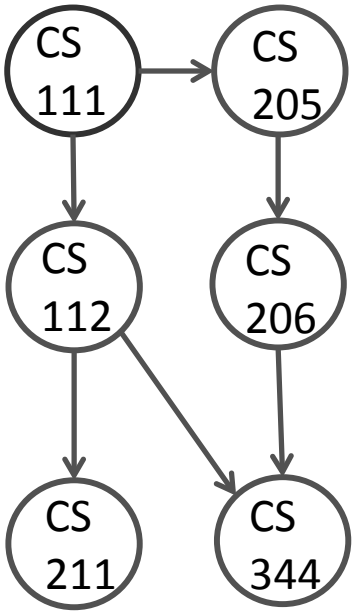


TopSort algorithm using DFS



Output from DFS: 111, 112, 211, 344, 205, 206
Is this also a topological sort?

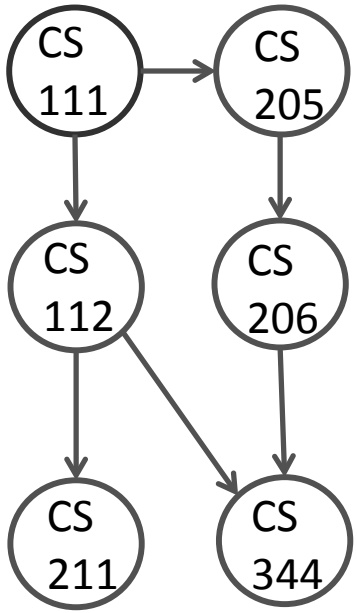
TopSort algorithm using DFS



Output from DFS: 111, 205, 206, 344, 112, 211

This is NOT a correct topological sort.
112 should be before 344.

TopSort algorithm using DFS



Output from DFS: 111, 205, 206, 344, 112, 211

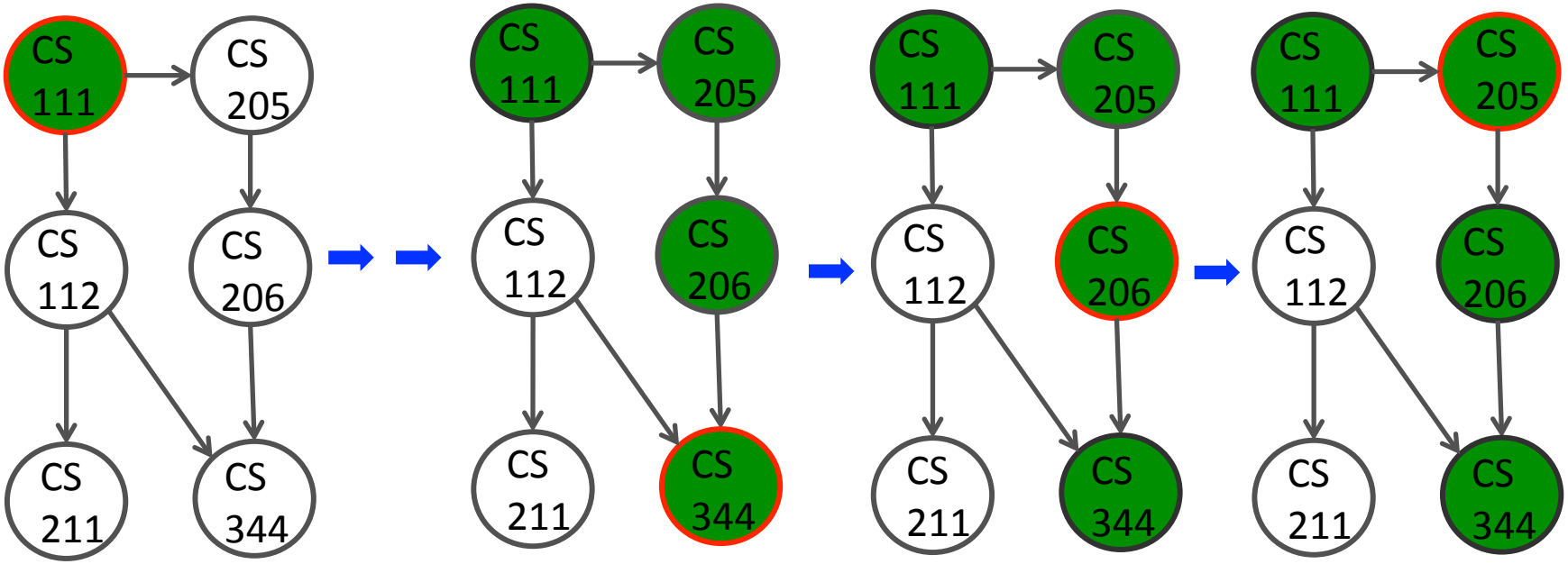
This is NOT a correct topological sort.
112 should be before 344.

Two simple rules can fix this. Do a DFS, but

1. Assign topological number is descending order

2. Number a vertex just before backing out

TopSort algorithm using DFS



Start DFS
Start numbering at 6 (6 = #nodes)

No outgoing edge from CS344.
So assign 6 to CS344

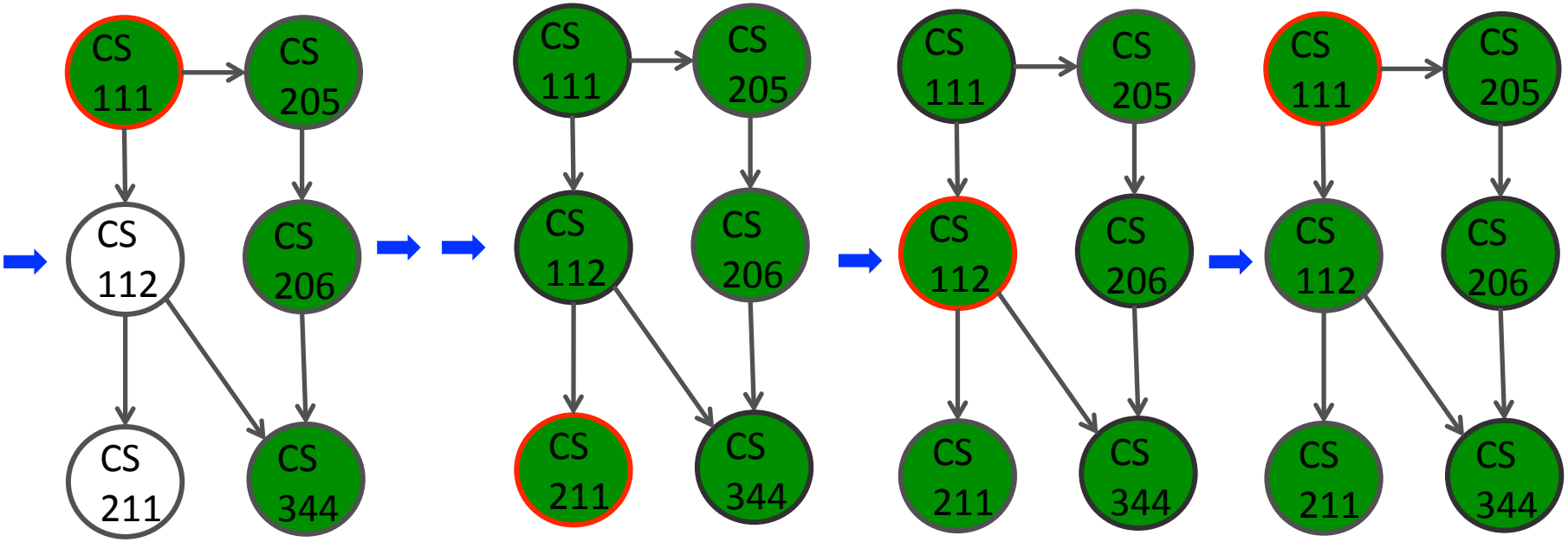
No other outgoing edge from CS206.
So assign 5 to CS206

No other outgoing edge from CS205.
So assign 4 to CS205

Two simple rules can fix this. Do a DFS, but

1. Assign topological number is descending order
2. Number a vertex just before backing out

TopSort algorithm using DFS



Back at CS111.
Visit CS 112,
then CS 211.

No outgoing edge from CS211.
So assign 3 to CS211

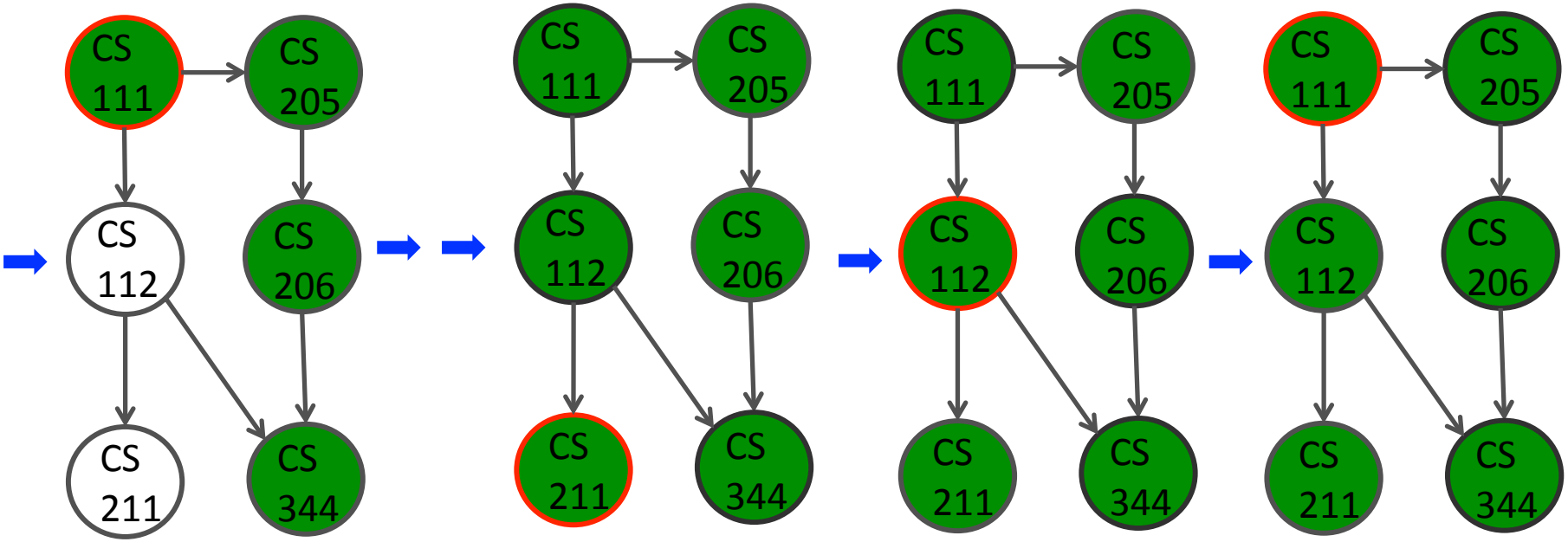
Backing out from CS112.
So assign 2 to CS112

Backing out from CS111.
So assign 1 to CS111

Two simple rules can fix this. Do a DFS, but

1. Assign topological number is descending order
2. Number a vertex just before backing out

TopSort algorithm using DFS



Back at CS111.
Visit CS 112,
then CS 211.

No outgoing edge
from CS211.
So assign 3 to
CS211

Backing out
from CS112.
So assign 2 to
CS112

Backing out
from CS111.
So assign 1 to
CS111

Node	CS 111	CS 112	CS 211	CS 205	CS 206	CS 344
Topological number	1	2	3	4	5	6

Topological sorting and DFS

- Output of DFS is order of visiting nodes.

  Output of TopSort is reverse order of backing out from nodes

- Note: Topological Sort would not work if there are cycles! Only works on Directed Acyclic Graphs (DAG)

Topological sort using DFS: Code

Algorithm DFStopsortdriver

```
topnum ← n
for each vertex v in the graph do
    if (v is not visited) then
        DFStopsort( v, topnum )
    endif
endfor
```

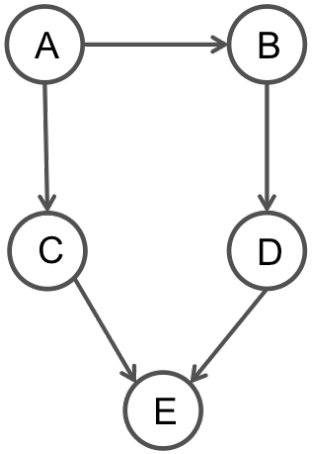
Algorithm DFStopsort(v, topnum)

```
visit v and mark v as visited
for each neighbor w of v do
    if (w is not visited) then
        DFStopsort(w, topnum)
    endif
endfor
number v with topnum
topnum ← topnum − 1
```

From the textbook, page 441

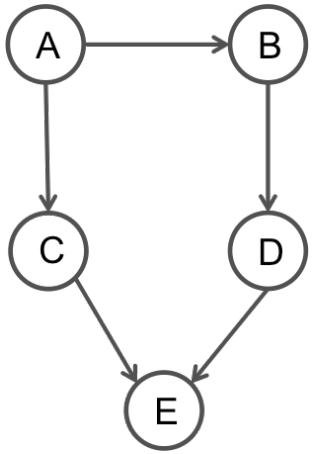
Part 2: Breadth-First Search (BFS)

What is BFS?

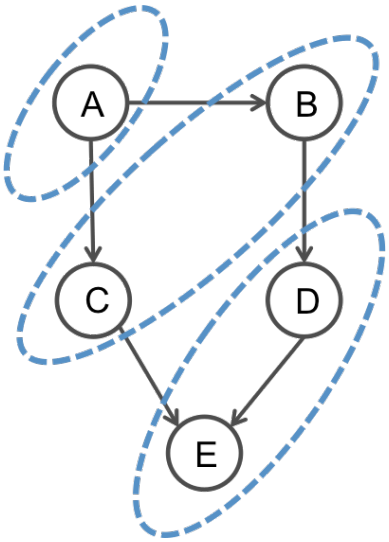


Output of DFS (starting at A): ?

What is BFS?



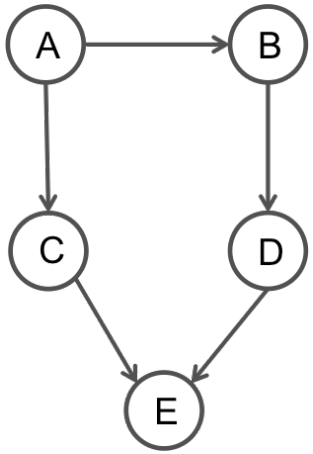
Output of DFS (starting at A): A, B, D, E, C



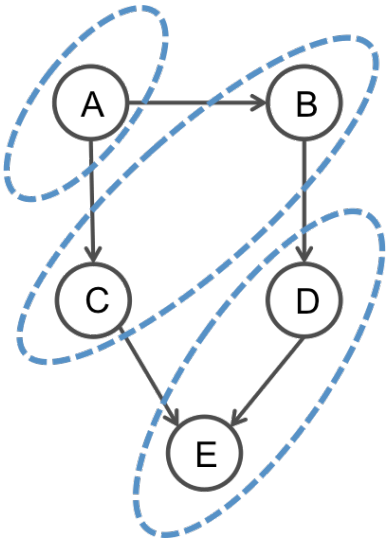
Output of BFS (starting at A): A, B, C, D, E

Nodes are visited in “waves”

What is BFS?



Output of DFS (starting at A): A, B, D, E, C



Output of BFS (starting at A): A, B, C, D, E

Nodes are visited in “waves”

How do we write a non-recursive algorithm to do a BFS?

BFS: Pseudocode

Algorithm BFS(v)

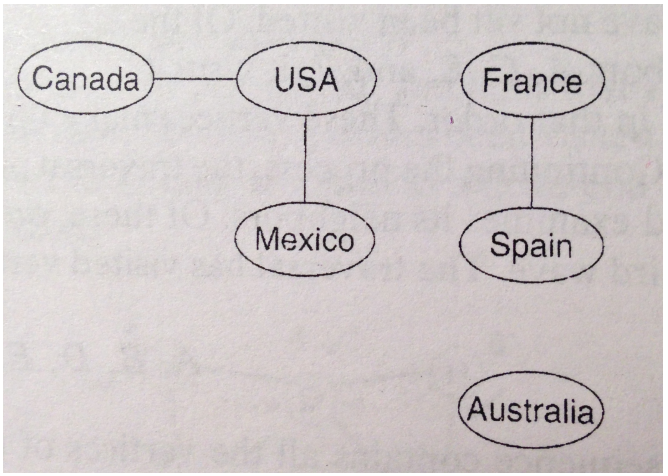
```
visit v and mark v as visited
add v to the queue
while the queue is not empty do
        w ← vertex at the front of the queue
        delete w from the queue
        for each neighbor p of w do
            if (p is not visited) then
                visit p and mark p as visited
                add p to the queue
            endif
        endfor
endwhile
```

From textbook, page 434

BFS: Pseudocode

```
Algorithm BFS(v)

visit v and mark v as visited
add v to the queue
while the queue is not empty do
        w ← vertex at the front of the queue
        delete w from the queue
        for each neighbor p of w do
            if (p is not visited) then
               visit p and mark p as visited
               add p to the queue
            endif
        endfor
endwhile
```

But, what if the graph is disconnected?



From textbook, page 434

BFS: Pseudocode

Algorithm BFS(v)

```
visit v and mark v as visited
add v to the queue
while the queue is not empty do
        w ← vertex at the front of the queue
        delete w from the queue
        for each neighbor p of w do
            if (p is not visited) then
              visit p and mark p as visited
              add p to the queue
            endif
        endfor
endwhile
```

Algorithm XFSdriver

```
for each vertex v in the graph do
    if (v is not visited) then
      XFS ( v )
    endif
endfor
```

From textbook, page 434, 435

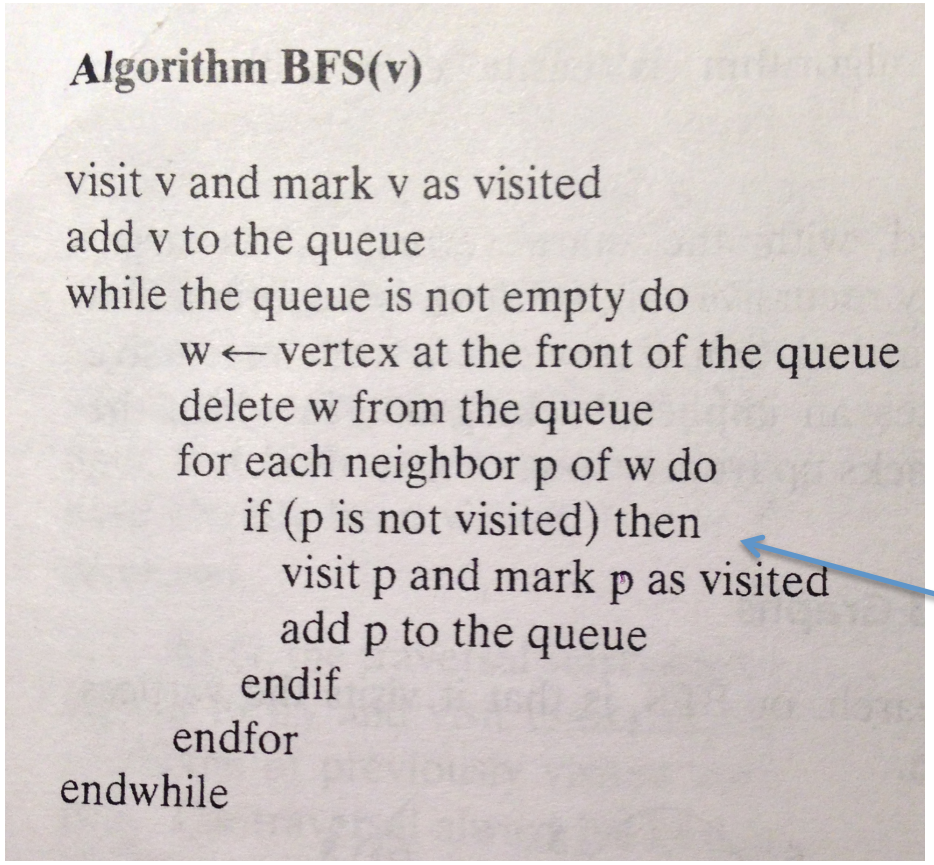
BFS: Analysis

```
Algorithm BFS(v)

visit v and mark v as visited
add v to the queue
while the queue is not empty do
        w ← vertex at the front of the queue
        delete w from the queue
        for each neighbor p of w do
            if (p is not visited) then
                visit p and mark p as visited
                add p to the queue
            endif
        endfor
endwhile
```

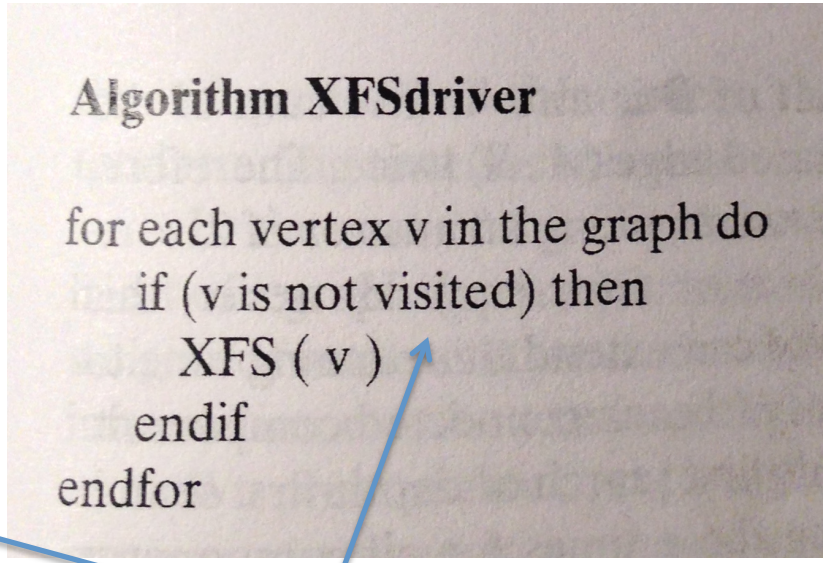
```
Algorithm XFSdriver

for each vertex v in the graph do
    if (v is not visited) then
        XFS ( v )
    endif
endfor
```

Each node is visited once. So that's *n*

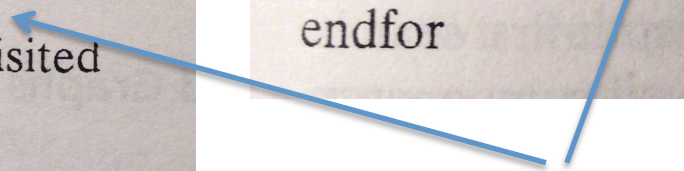
BFS: Analysis



```
Algorithm BFS(v)

visit v and mark v as visited
add v to the queue
while the queue is not empty do
        w ← vertex at the front of the queue
        delete w from the queue
        for each neighbor p of w do
            if (p is not visited) then
              visit p and mark p as visited
              add p to the queue
            endif
        endfor
endwhile
```



```
Algorithm XFSdriver

for each vertex v in the graph do
    if (v is not visited) then
      XFS ( v )
    endif
endfor
```



Each node is visited once. So that's *n*

Each node is inspected as many times as its degree (# neighbors).
Sum of degrees of nodes = 2 * Number of edges (How?)

BFS: Analysis

```
Algorithm BFS(v)

visit v and mark v as visited
add v to the queue
while the queue is not empty do
        w ← vertex at the front of the queue
        delete w from the queue
        for each neighbor p of w do
            if (p is not visited) then
              visit p and mark p as visited
              add p to the queue
            endif
        endfor
endwhile
```

```
Algorithm XFSdriver

for each vertex v in the graph do
    if (v is not visited) then
      XFS ( v )
    endif
endfor
```

Each node is visited once. So that's *n*

Each node is inspected as many times as its degree (# neighbors).
Sum of degrees of nodes = 2 * Number of edges
So, total running time is n + 2e. i.e O(n + e)

Thank you!

Extra slides

Comparison between DFS and BFS

Similarities:

- Runtime is the same O(n + e)
- Both works on directed & undirected graphs, with and without cycles

Differences:

- Traversal and hence output is different.
- BFS uses queues. Non-recursive DFS needs stacks.