

FIT2004

Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Companeze, Aamir Cheema, Arun Konagurthu and Lloyd Allison



Faculty of Information Technology, Monash University

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act. Do not remove this notice

Ready?

Agenda

- Directed Acyclic Graph (DAG)

Agenda

- Directed Acyclic Graph (DAG)
- Topological Sort
 - Kahn's algorithm
 - Depth-First Search (DFS) modification

Let us begin...

Directed Acyclic Graph (DAG)

What is it?

- Graphs are very commonly used to model real world scenario
 - One of which is a DAG

Directed Acyclic Graph (DAG)

What is it?

- Graphs are very commonly used to model real world scenario
 - One of which is a DAG
- What is a DAG?

Directed Acyclic Graph (DAG)

What is it?

- Graphs are very commonly used to model real world scenario
 - One of which is a DAG
- What is a DAG?
 - Directed

Directed Acyclic Graph (DAG)

What is it?

- Graphs are very commonly used to model real world scenario
 - One of which is a DAG

- What is a DAG?
 - Directed
 - Acyclic

Directed Acyclic Graph (DAG)

What is it?

- Graphs are very commonly used to model real world scenario
 - One of which is a DAG

- What is a DAG?
 - Directed
 - Acyclic
 - and of course it is a Graph...

Directed Acyclic Graph (DAG)

What is it?

- Graphs are very commonly used to model real world scenario
 - One of which is a DAG
 - Can you give me an example of a real world DAG?

- What is a DAG?
 - Directed
 - Acyclic
 - and of course it is a Graph...

Directed Acyclic Graph (DAG)

What is it?

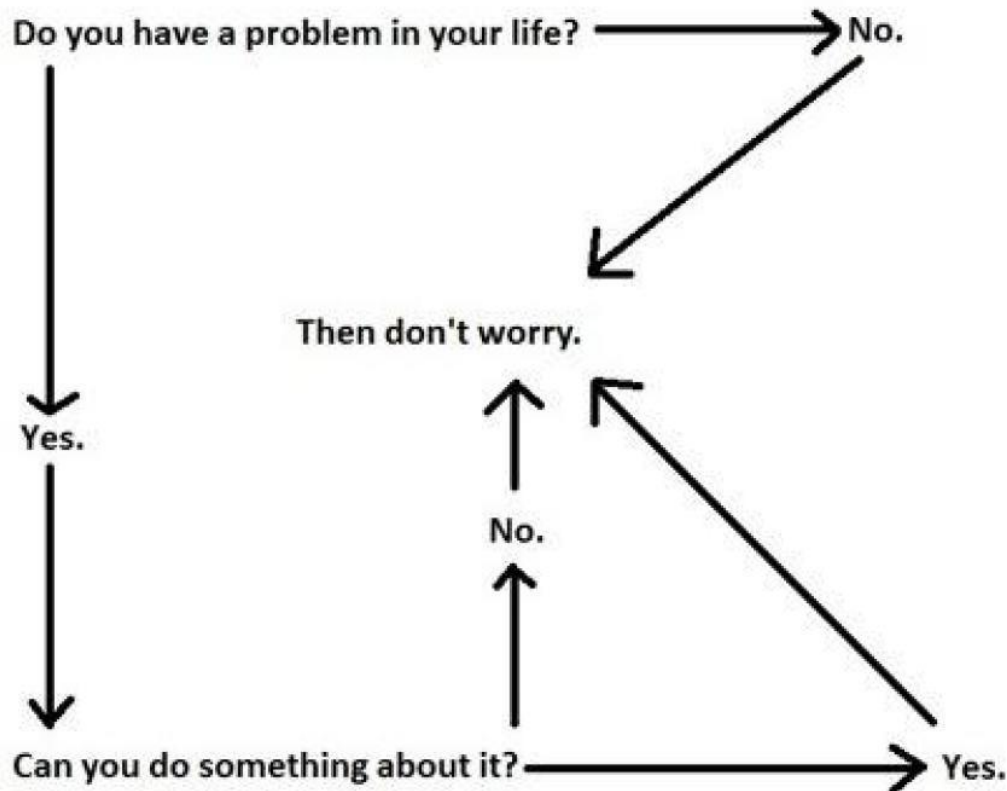
- Graphs are very commonly used to model real world scenario
 - One of which is a DAG
 - Can you give me an example of a real world DAG?
 - Can you give me an example of a real world non-DAG?

- What is a DAG?
 - Directed
 - Acyclic
 - and of course it is a Graph...

Directed Acyclic Graph (DAG)

What is it?

- A real world DAG



Directed Acyclic Graph (DAG)

What is it?

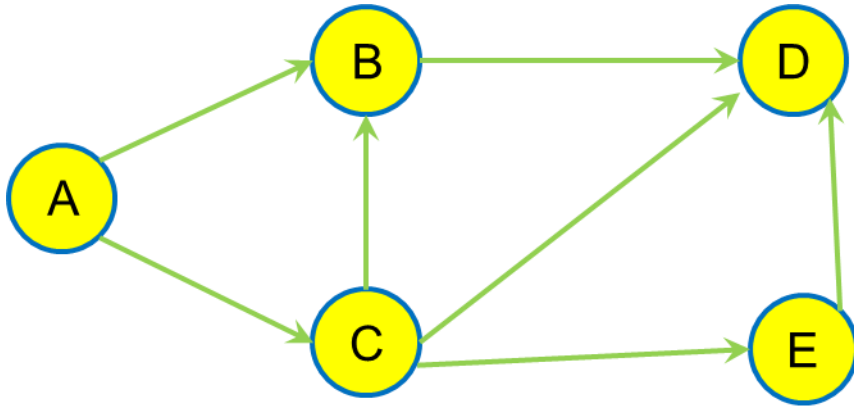
- A real world not DAG



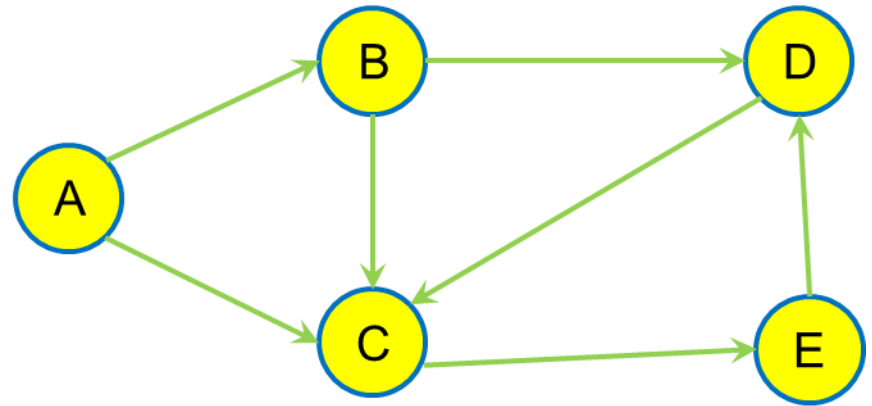
Directed Acyclic Graph (DAG)

What is it?

- Which graph is a DAG?
 - And where is the cycle?



Graph 1



Graph 2

Questions?

Directed Acyclic Graph (DAG)

What is it for?

- Prerequisite mapping

Directed Acyclic Graph (DAG)

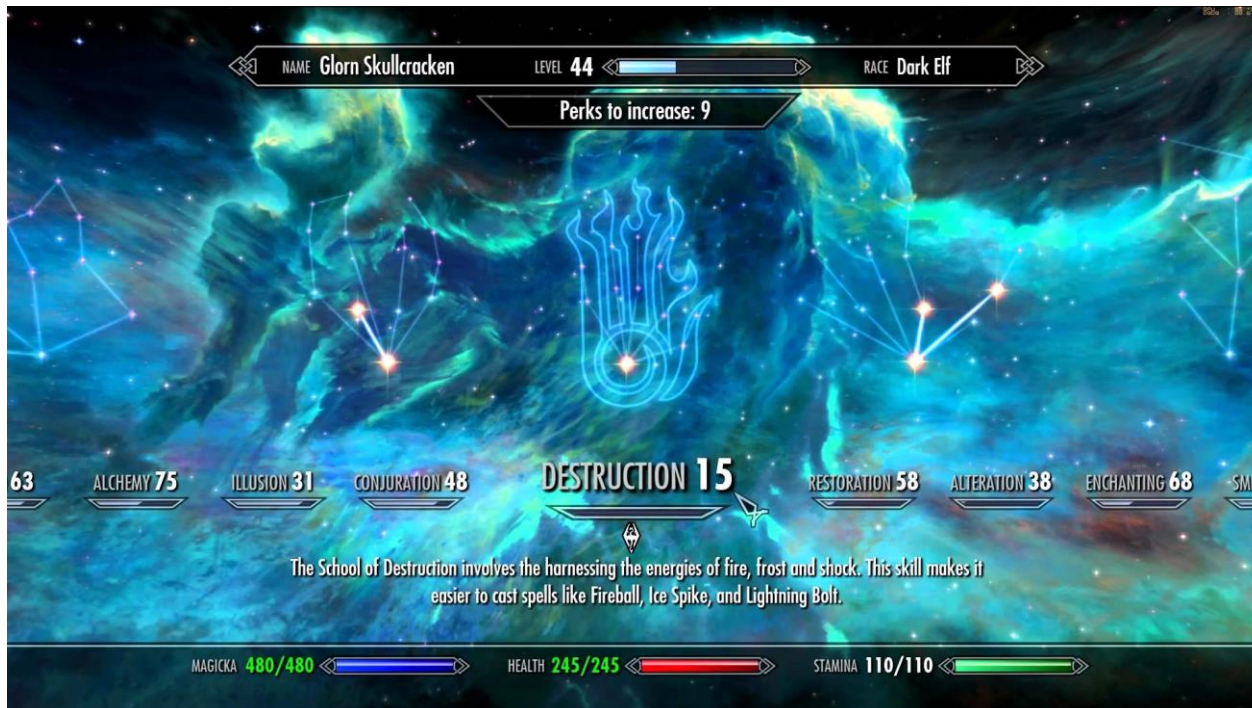
What is it for?

- Prerequisite mapping
 - If I have a directed edge from A to B, this means I need A before B

Directed Acyclic Graph (DAG)

What is it for?

- Prerequisite mapping
 - If I have a directed edge from A to B, this means I need A before B
 - Common in project management
 - Common in talent/skill trees!



Directed Acyclic Graph (DAG)

What is it for?

- Prerequisite mapping
 - If I have a directed edge from A to B, this means I need A before B
 - Common in project management
 - Common in talent/skill trees!
 - Your university units!
 - Pass 1045
 - Pass 1008
 - Pass 2004
 - Pass 3155

Directed Acyclic Graph (DAG)

What is it for?

- So for an edge $\langle A, B \rangle$

Directed Acyclic Graph (DAG)

What is it for?

- So for an edge $\langle A, B \rangle$
 - A is a prerequisite for B
 - A is an ancestor of B
 - A is the subset of B
 - A is ordered before B

Directed Acyclic Graph (DAG)

What is it for?

- So for an edge $\langle A, B \rangle$
 - A is a prerequisite for B
 - A is an ancestor of B
 - A is the subset of B
 - A is ordered before B
 - This enable us to sort a DAG

Questions?

Topological Sort

Ordering of Vertices

- A topological sort
 - Permutation of vertices in a DAG
 - Vertex U will appear before vertex V if we have edge $\langle U, V \rangle$

Topological Sort

Ordering of Vertices

- A topological sort
 - Permutation of vertices in a DAG
 - Vertex U will appear before vertex V if we have edge $\langle U, V \rangle$
 - Vertex U will appear before vertex W if we have edge $\langle U, W \rangle$

Topological Sort

Ordering of Vertices

- A topological sort
 - Permutation of vertices in a DAG
 - Vertex U will appear before vertex V if we have edge $\langle U, V \rangle$
 - Vertex U will appear before vertex W if we have edge $\langle U, W \rangle$
 - But if we don't have edge $\langle V, W \rangle$ then V and W are of the same order

Topological Sort

Ordering of Vertices

- A topological sort
 - Permutation of vertices in a DAG
 - Vertex U will appear before vertex V if we have edge $\langle U, V \rangle$
 - $U < V$
 - Vertex U will appear before vertex W if we have edge $\langle U, W \rangle$
 - $U < W$
 - But if we don't have edge $\langle V, W \rangle$ then V and W are of the same order
 - $V == W$

Topological Sort

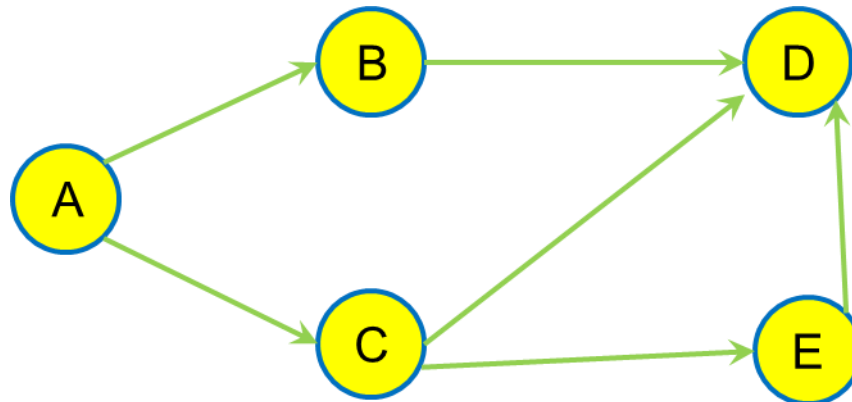
Ordering of Vertices

- A topological sort
 - Permutation of vertices in a DAG
 - Vertex U will appear before vertex V if we have edge $\langle U, V \rangle$
 - $U < V$
 - Vertex U will appear before vertex W if we have edge $\langle U, W \rangle$
 - $U < W$
 - But if we don't have edge $\langle V, W \rangle$ then V and W are of the same order
 - $V == W$
- So we have a DAG of your units
- Topological sort of this DAG gives you the order of units to take!

Topological Sort

Ordering of Vertices

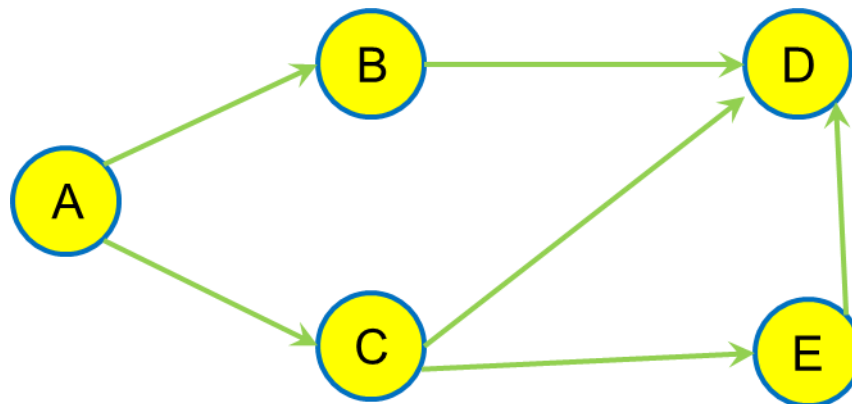
- Which one of these are not a valid topological sort of the DAG?



Topological Sort

Ordering of Vertices

- Which one of these are not a valid topological sort of the DAG?
 1. A, B, C, E, D
 2. A, C, B, E, D
 3. A, C, E, B, D
 4. A, B, E, C, D



Topological Sort

Ordering of Vertices

- Topological sort can be done via
 - Kahn's algorithm
 - A modified DFS

Questions?

Kahn's Algorithm

For topological sort

- What is the concept like?

Kahn's Algorithm

For topological sort

- What is the concept like?
 - Start with vertices without incoming edges

Kahn's Algorithm

For topological sort

- What is the concept like?
 - Start with vertices without incoming edges
 - Delete all outgoing edges from the vertex

Kahn's Algorithm

For topological sort

- What is the concept like?
 - Start with vertices without incoming edges
 - Delete all outgoing edges from the vertex
 - Add in vertices without incoming edges

Kahn's Algorithm

For topological sort

- What is the concept like?
 - Start with vertices without incoming edges
 - Delete all outgoing edges from the vertex
 - Add in vertices without incoming edges
 - Repeat!

Kahn's Algorithm

For topological sort

- Algorithm as follow

```
1  sorted_list = []
2  process = []
3  add all vertices without incoming edges to process
4
5  while len(process) > 0:
6      vertex_u = process.pop()
7      sorted_list.append(vertex_u)
8      for edge in vertex_u.edges:
9          remove edge from graph
10         if edge.vertex_v has no incoming edges:
11             process.append(vertex_v)
12
13  if graph still has edges:
14      print("Error. Not a cycle")
15  else:
16      print(sorted_list)
```


Kahn's Algorithm

For topological sort

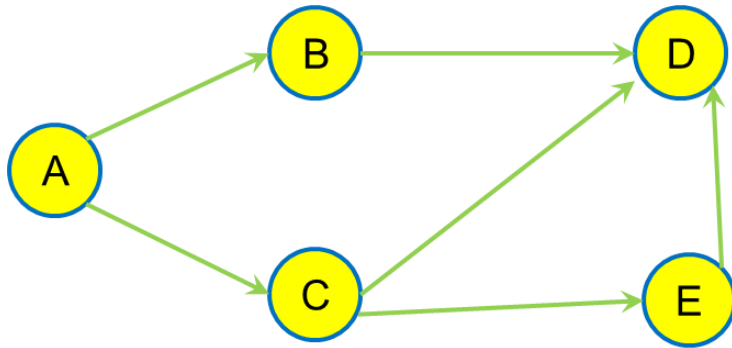
- Algorithm as follow
 - Let us try it out

```
1  sorted_list = []
2  process = []
3  add all vertices without incoming edges to process
4
5  while len(process) > 0:
6      vertex_u = process.pop()
7      sorted_list.append(vertex_u)
8      for edge in vertex_u.edges:
9          remove edge from graph
10         if edge.vertex_v has no incoming edges:
11             process.append(vertex_v)
12
13  if graph still has edges:
14      print("Error. Not a cycle")
15  else:
16      print(sorted_list)
```

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



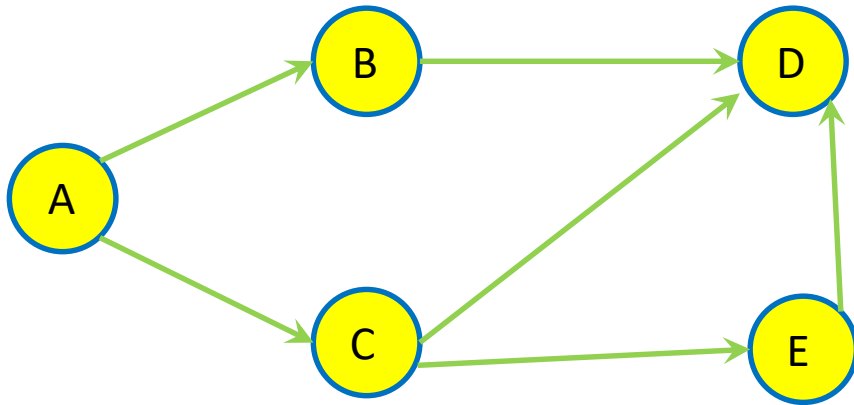
sorted					
process					

```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10    if edge.vertex_v has no incoming edges:
11        process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



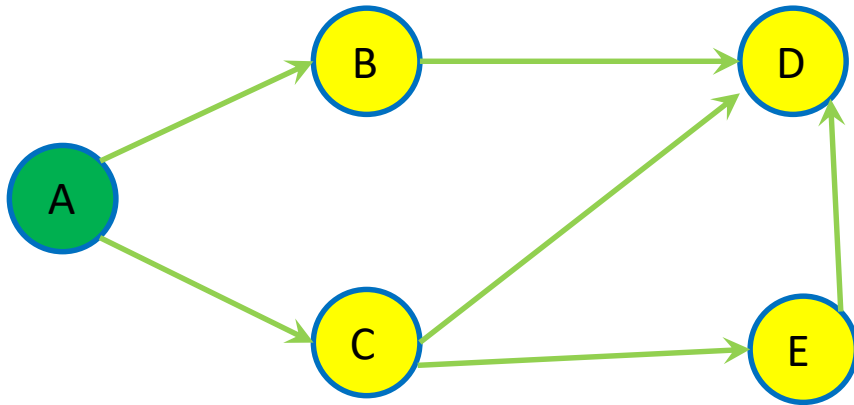
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted					
process	A				

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



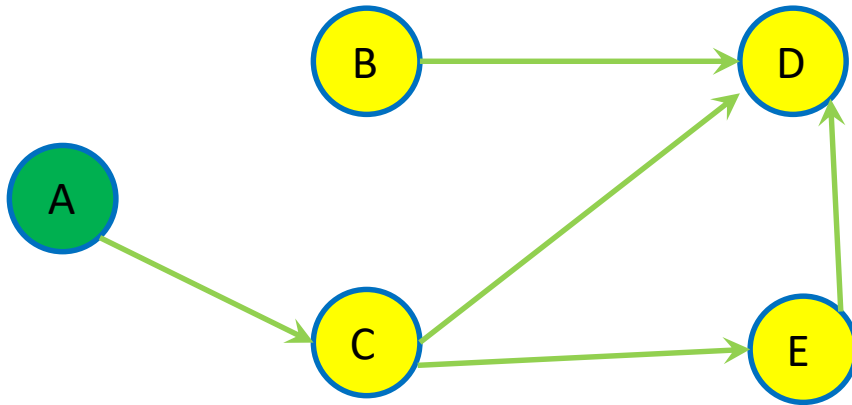
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A				
process					

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



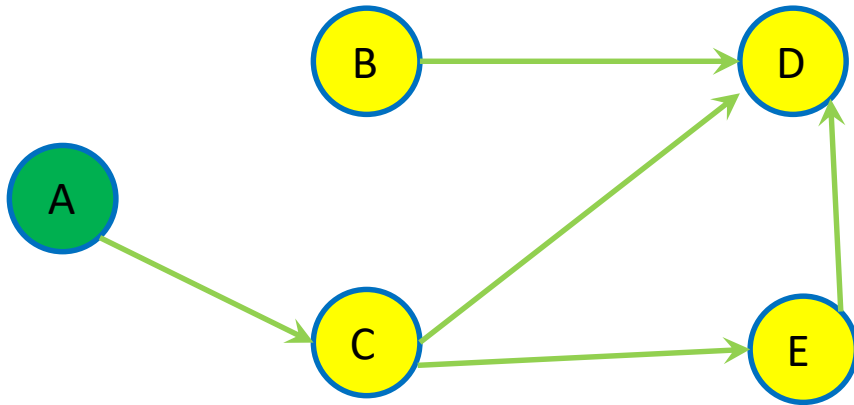
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A				
process					

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



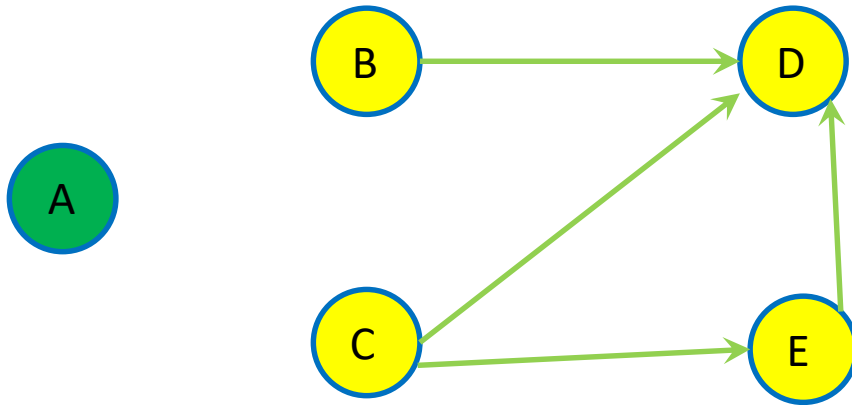
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A				
process	B				

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



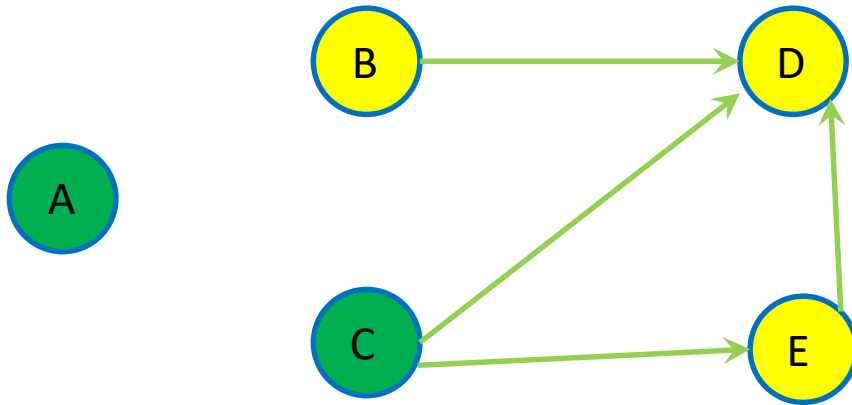
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A				
process	B	C			

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



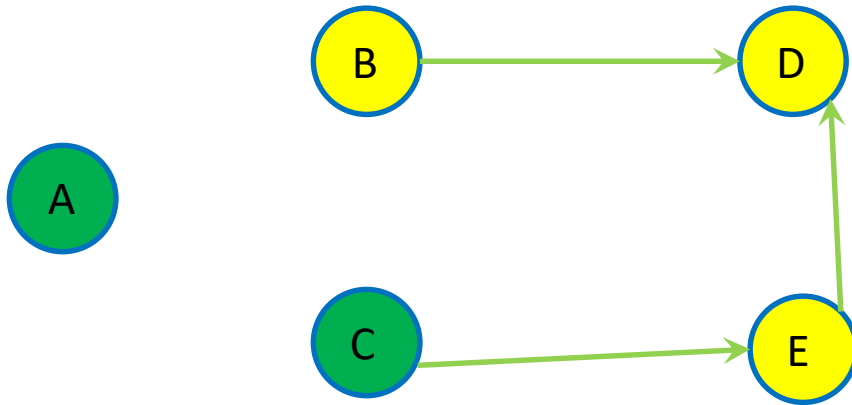
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A	C			
process	B				

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



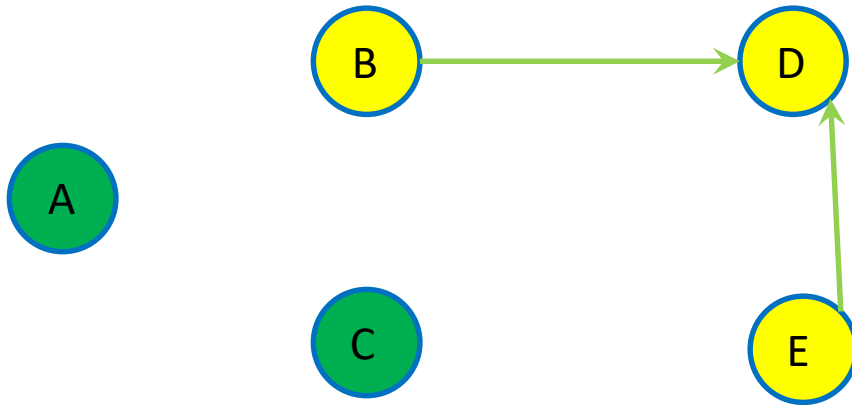
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A	C			
process	B				

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



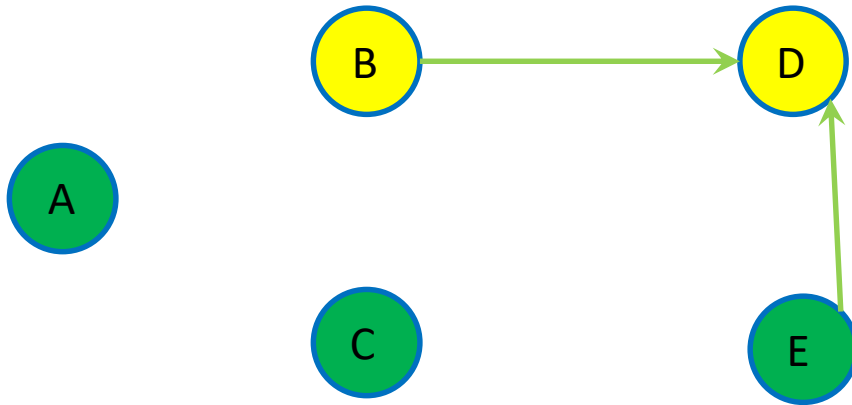
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A	C			
process	B	E			

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



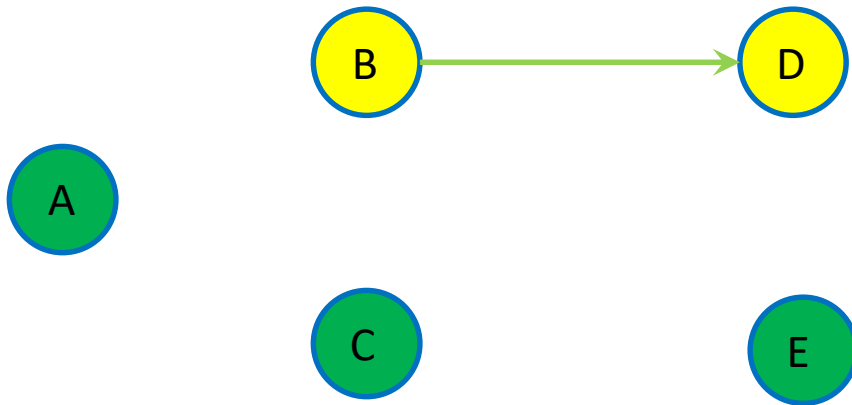
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10        if edge.vertex_v has no incoming edges:
11            process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A	C	E		
process	B				

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



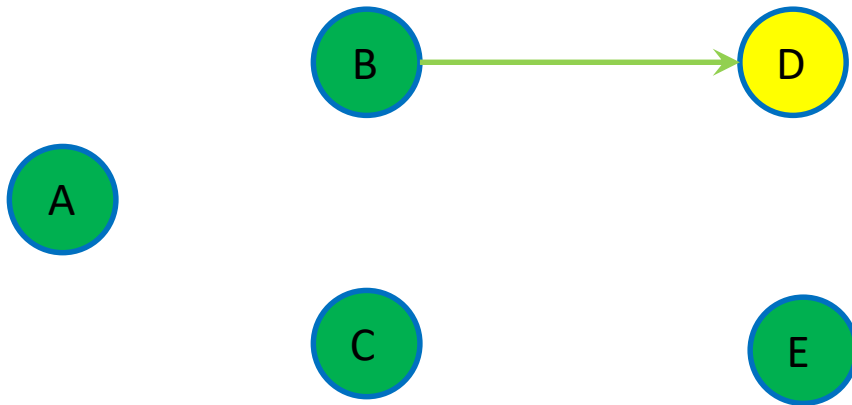
sorted	A	C	E		
process	B				

```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10    if edge.vertex_v has no incoming edges:
11        process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



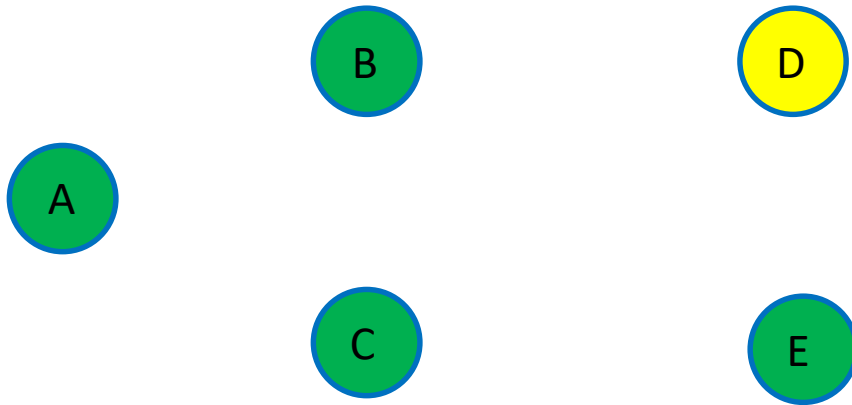
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10    if edge.vertex_v has no incoming edges:
11        process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A	C	E	B	
process					

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



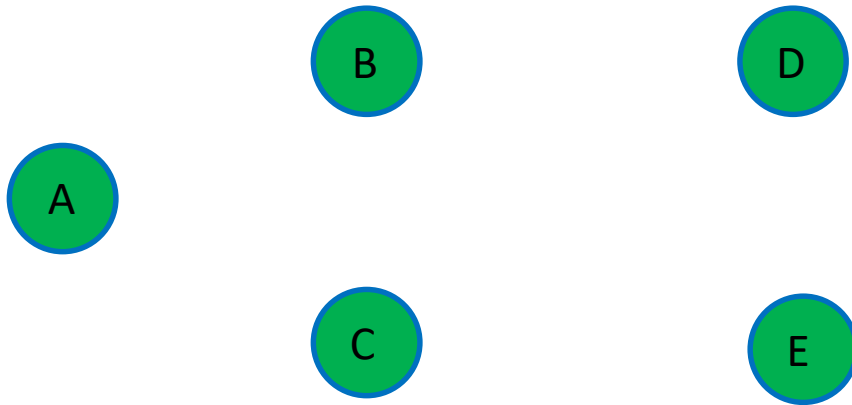
```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10    if edge.vertex_v has no incoming edges:
11        process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A	C	E	B	
process	D				

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Let us try it out



```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10    if edge.vertex_v has no incoming edges:
11        process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```

sorted	A	C	E	B	D
process					

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Seemed simple right?
 - Let us zoomed in to the algorithm more

```
1 sorted_list = []
2 process = []
3 add all vertices without incoming edges to process
4
5 while len(process) > 0:
6     vertex_u = process.pop()
7     sorted_list.append(vertex_u)
8     for edge in vertex_u.edges:
9         remove edge from graph
10    if edge.vertex_v has no incoming edges:
11        process.append(vertex_v)
12
13 if graph still has edges:
14     print("Error. Not a cycle")
15 else:
16     print(sorted_list)
```


Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Seemed simple right?
 - Let us zoomed in to the algorithm more

```
1  # for output
2  sorted_list = []
3  # tracks number of incoming edges
4  incoming_edges = [0] * len(vertices)
5  for edge in graph:          # edge <u,v>
6      incoming_edges[vertex_v] += 1
7  # process queue or stack
8  process = []
9  for vertex_v in incoming_edges:
10     if incoming_edges[vertex_v] == 0:
11         process.append(vertex_v)
12     # kahn's
13     while len(process) > 0:
14         vertex_u = process.pop()
15         sorted_list.append(vertex_u)
16         for edge in vertex_u.edges:
17             incoming_edges[edge.vertex_v] -= 1
18             if incoming_edges[vertex_v] == 0:
19                 process.append(vertex_v)
20     # results
21     if graph still has edges:
22         print("Error. Not a cycle")
23     else:
24         print(sorted_list)
```

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Seemed simple right?
 - Let us zoomed in to the algorithm more
- Complexity?

```
1  # for output
2  sorted_list = []
3  # tracks number of incoming edges
4  incoming_edges = [0] * len(vertices)
5  for edge in graph:          # edge <u,v>
6      incoming_edges[vertex_v] += 1
7  # process queue or stack
8  process = []
9  for vertex_v in incoming_edges:
10     if incoming_edges[vertex_v] == 0:
11         process.append(vertex_v)
12     # kahn's
13     while len(process) > 0:
14         vertex_u = process.pop()
15         sorted_list.append(vertex_u)
16         for edge in vertex_u.edges:
17             incoming_edges[edge.vertex_v] -= 1
18             if incoming_edges[vertex_v] == 0:
19                 process.append(vertex_v)
20     # results
21     if graph still has edges:
22         print("Error. Not a cycle")
23     else:
24         print(sorted_list)
```

Kahn's Algorithm

For topological sort

- Algorithm as follow
 - Seemed simple right?
 - Let us zoomed in to the algorithm more
- Complexity?
 - $O(V+E)$ time space

```
1  # for output
2  sorted_list = []
3  # tracks number of incoming edges
4  incoming_edges = [0] * len(vertices)
5  for edge in graph:          # edge <u,v>
6      incoming_edges[vertex_v] += 1
7  # process queue or stack
8  process = []
9  for vertex_v in incoming_edges:
10     if incoming_edges[vertex_v] == 0:
11         process.append(vertex_v)
12     # kahn's
13     while len(process) > 0:
14         vertex_u = process.pop()
15         sorted_list.append(vertex_u)
16         for edge in vertex_u.edges:
17             incoming_edges[edge.vertex_v] -= 1
18             if incoming_edges[vertex_v] == 0:
19                 process.append(vertex_v)
20     # results
21     if graph still has edges:
22         print("Error. Not a cycle")
23     else:
24         print(sorted_list)
```

Questions?

Depth-First Search (DFS)

Modified for topological sorting

- We saw the complexity of Kahn's being $O(V+E)$

Depth-First Search (DFS)

Modified for topological sorting

- We saw the complexity of Kahn's being $O(V+E)$
- Can we modify DFS to do the same?

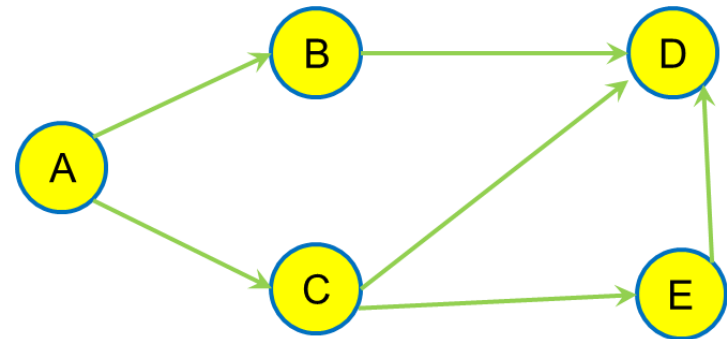
```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Let us run DFS on the following graph and see

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

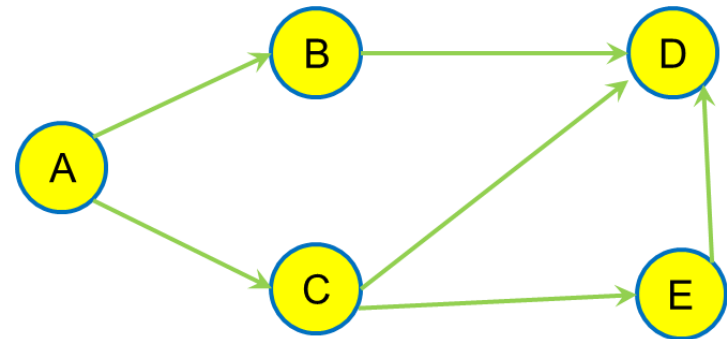


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Let us run DFS on the following graph and see
 - Start from A

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

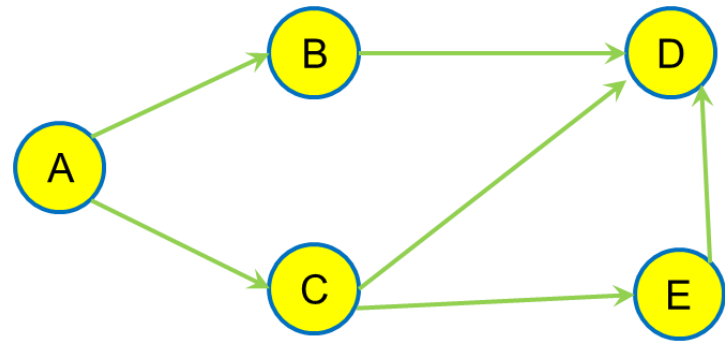


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Let us run DFS on the following graph and see
 - Start from A
 - Go to B

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```



Depth-First Search (DFS)

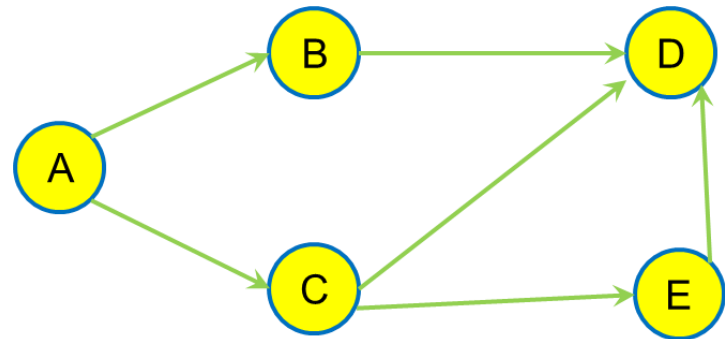
Modified for topological sorting

- Can we modify DFS to do the same?

- Let us run DFS on the following graph and see

- Start from A
- Go to B
- Go to D

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```



Depth-First Search (DFS)

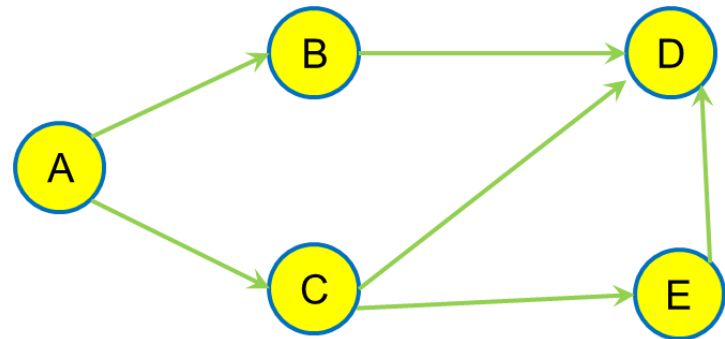
Modified for topological sorting

- Can we modify DFS to do the same?

- Let us run DFS on the following graph and see

- Start from A
- Go to B
- Go to D
- Go to C

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```



Depth-First Search (DFS)

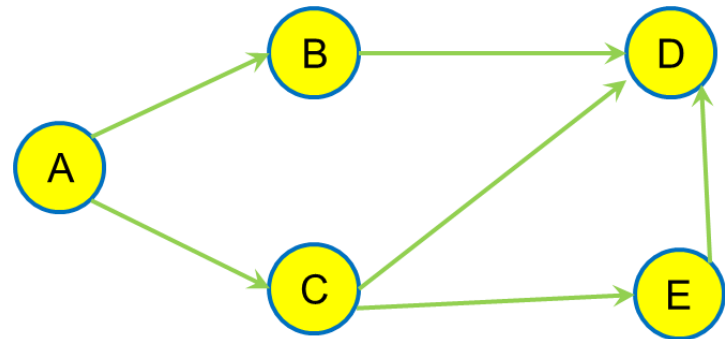
Modified for topological sorting

- Can we modify DFS to do the same?

- Let us run DFS on the following graph and see

- Start from A
- Go to B
- Go to D
- Go to C
- Go to E

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

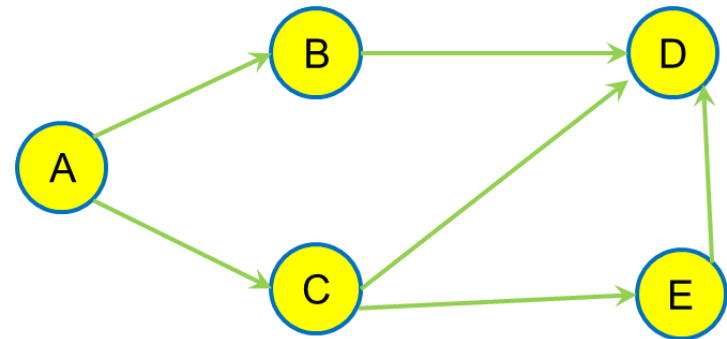


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Let us run DFS on the following graph and see
 - Start from A
 - Go to B
 - Go to D
 - Go to C
 - Go to E
 - So we have A, B, D, C, E

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

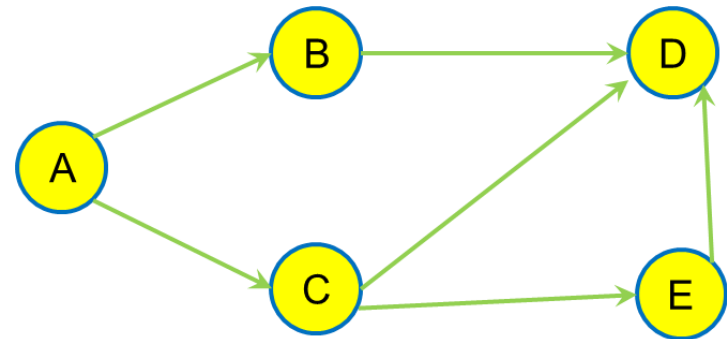


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Let us run DFS on the following graph and see
 - Start from A
 - Go to B
 - Go to D
 - Go to C
 - Go to E
 - So we have A, B, D, C, E
 - Any other DFS order?

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

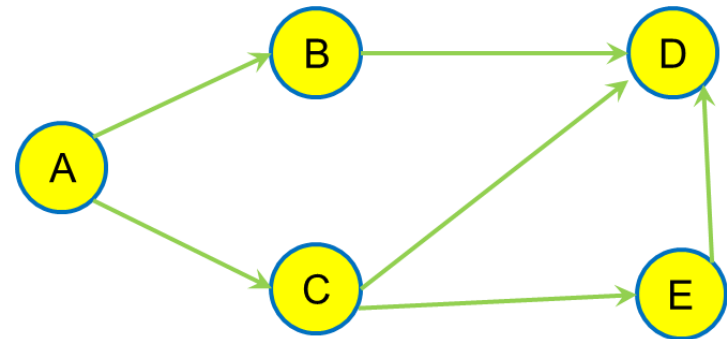


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Let us run DFS on the following graph and see
 - Start from A
 - Go to B
 - Go to D
 - Go to C
 - Go to E
 - So we have A, B, D, C, E
 - Any other DFS order?
 - A, C, D, E, B

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

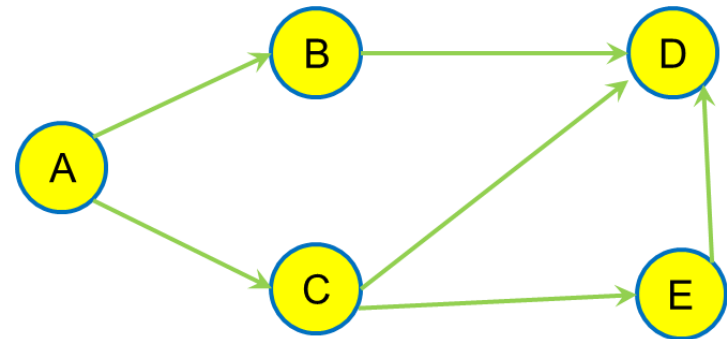


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Let us run DFS on the following graph and see
 - Start from A
 - Go to B
 - Go to D
 - Go to C
 - Go to E
 - So we have A, B, D, C, E
 - Any other DFS order?
 - A, C, D, E, B
 - A, C, E, D, B

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

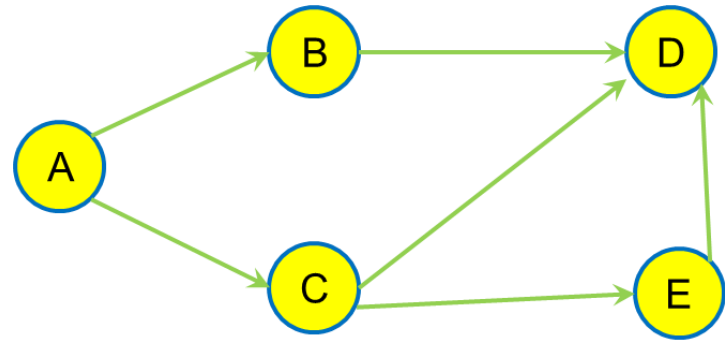


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Let us run DFS on the following graph and see
 - Start from A
 - Go to B
 - Go to D
 - Go to C
 - Go to E
 - So we have A, B, D, C, E
 - Any other DFS order?
 - A, C, D, E, B
 - A, C, E, D, B
 - A, C, E, B, D?

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

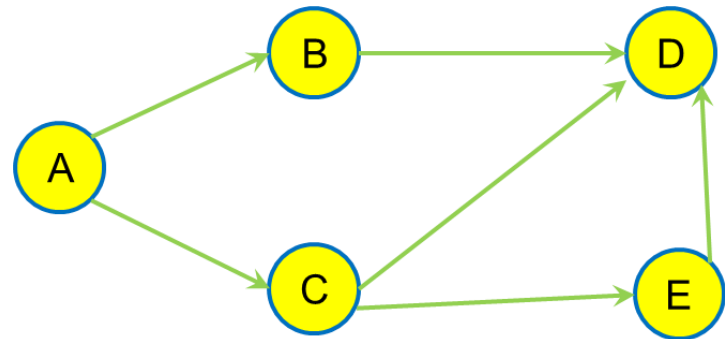


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Any other DFS order?
 - A, B, D, C, E
 - A, C, D, E, B
 - A, C, E, D, B

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

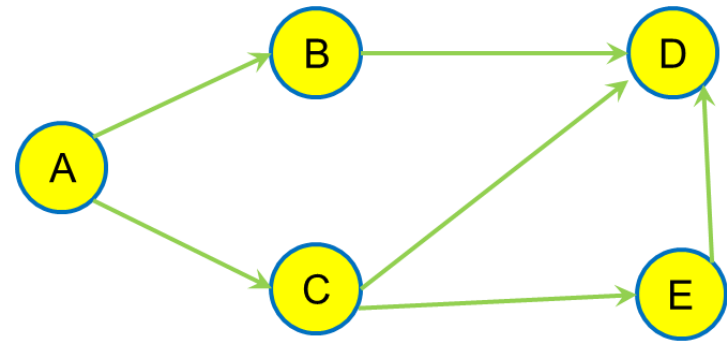


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Any other DFS order?
 - A, B, D, C, E
 - A, C, D, E, B
 - A, C, E, D, B
 - A possible topological sort
 - A, B, C, E, D
 - A, C, B, E, D

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```

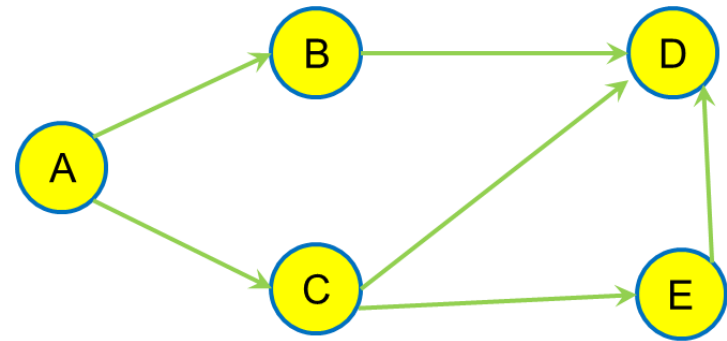


Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?
 - Any other DFS order?
 - A, B, D, C, E
 - A, C, D, E, B
 - A, C, E, D, B
 - A possible topological sort
 - A, B, C, E, D
 - A, C, B, E, D
 - Notice something?

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```



Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?

- Any other DFS order?

- A, B, D, C, E
- A, C, D, E, B
- A, C, E, D, B

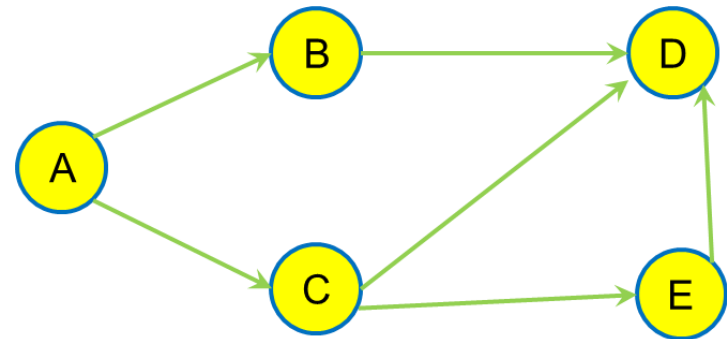
- A possible topological sort

- A, B, C, E, D
- A, C, B, E, D

- Notice something?

- When we reach the end of the DFS, we go back to an earlier vertex but this vertex should be early in topological sort (such as vertex B or C)

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```



Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?

- Any other DFS order?

- A, B, D, C, E
- A, C, D, E, B
- A, C, E, D, B

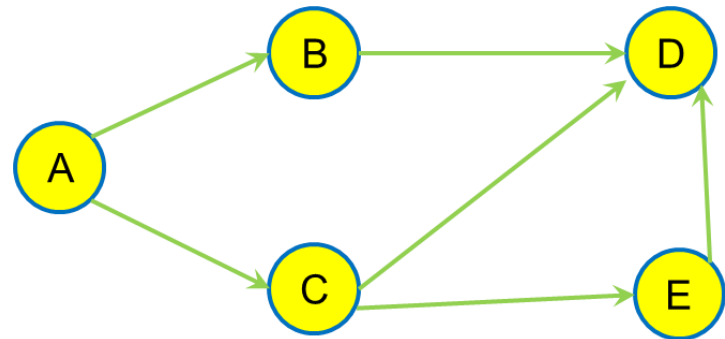
- A possible topological sort

- A, B, C, E, D
- A, C, B, E, D

- Notice something?

- When we reach the end of the DFS, we go back to an earlier vertex but this vertex should be early in topological sort (such as vertex B or C)

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```



Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?

- Any other DFS order?

- A, B, D, C, E
- A, C, D, E, B
- A, C, E, D, B

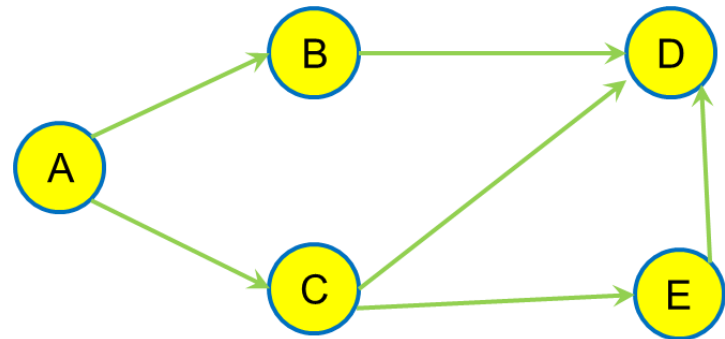
- A possible topological sort

- A, B, C, E, D
- A, C, B, E, D

- Notice something?

- When we reach the end of the DFS, we go back to an earlier vertex but this vertex should be early in topological sort (such as vertex B or C)

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```



Depth-First Search (DFS)

Modified for topological sorting

- Can we modify DFS to do the same?

- Any other DFS order?

- A, B, D, C, E
- A, C, D, E, B
- A, C, E, D, B

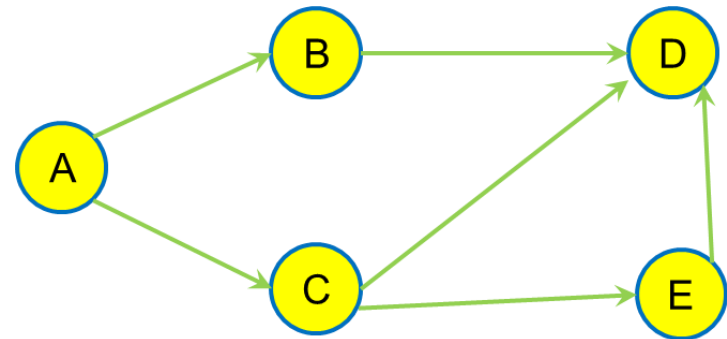
- A possible topological sort

- A, B, C, E, D
- A, C, B, E, D

- Notice something?

- When we reach the end of the DFS, we go back to an earlier vertex but this vertex should be early in topological sort (such as vertex B or C)
- So how do we arrange it?

```
28 def dfs_topological(vertex_u):  
29     vertex_u.visited = True  
30     for edge in vertex_u.edges:  
31         if edge.vertex_v.visited == False:  
32             dfs_topological(vertex_v)
```



Depth-First Search (DFS)

Modified for topological sorting

- Here's how we modify with a stack!

```
28 def dfs_topological(vertex_u):
29     # start for result
30     stack = []
31     # run DFS
32     vertex_u.visited = True
33     for edge in vertex_u.edges:
34         if edge.vertex_v.visited == False:
35             dfs_topological_aux(vertex_v, stack)
36     # output
37     print(stack)
38
39 def dfs_topological_aux(vertex_u, stack):
40     vertex_u.visited = True
41     for edge in vertex_u.edges:
42         if edge.vertex_v.visited == False:
43             dfs_topological_aux(vertex_v)
44     # add to stack
45     stack.push(vertex_u)
```

Depth-First Search (DFS)

Modified for topological sorting

- Complexity?

```
28 def dfs_topological(vertex_u):
29     # start for result
30     stack = []
31     # run DFS
32     vertex_u.visited = True
33     for edge in vertex_u.edges:
34         if edge.vertex_v.visited == False:
35             dfs_topological_aux(vertex_v, stack)
36     # output
37     print(stack)
38
39 def dfs_topological_aux(vertex_u, stack):
40     vertex_u.visited = True
41     for edge in vertex_u.edges:
42         if edge.vertex_v.visited == False:
43             dfs_topological_aux(vertex_v)
44     # add to stack
45     stack.push(vertex_u)
```

Depth-First Search (DFS)

Modified for topological sorting

- Complexity?
 - $O(V+E)$ since we only added a stack

```
28 def dfs_topological(vertex_u):
29     # start for result
30     stack = []
31     # run DFS
32     vertex_u.visited = True
33     for edge in vertex_u.edges:
34         if edge.vertex_v.visited == False:
35             dfs_topological_aux(vertex_v, stack)
36     # output
37     print(stack)
38
39 def dfs_topological_aux(vertex_u, stack):
40     vertex_u.visited = True
41     for edge in vertex_u.edges:
42         if edge.vertex_v.visited == False:
43             dfs_topological_aux(vertex_v)
44     # add to stack
45     stack.push(vertex_u)
```

Questions?

Thank You