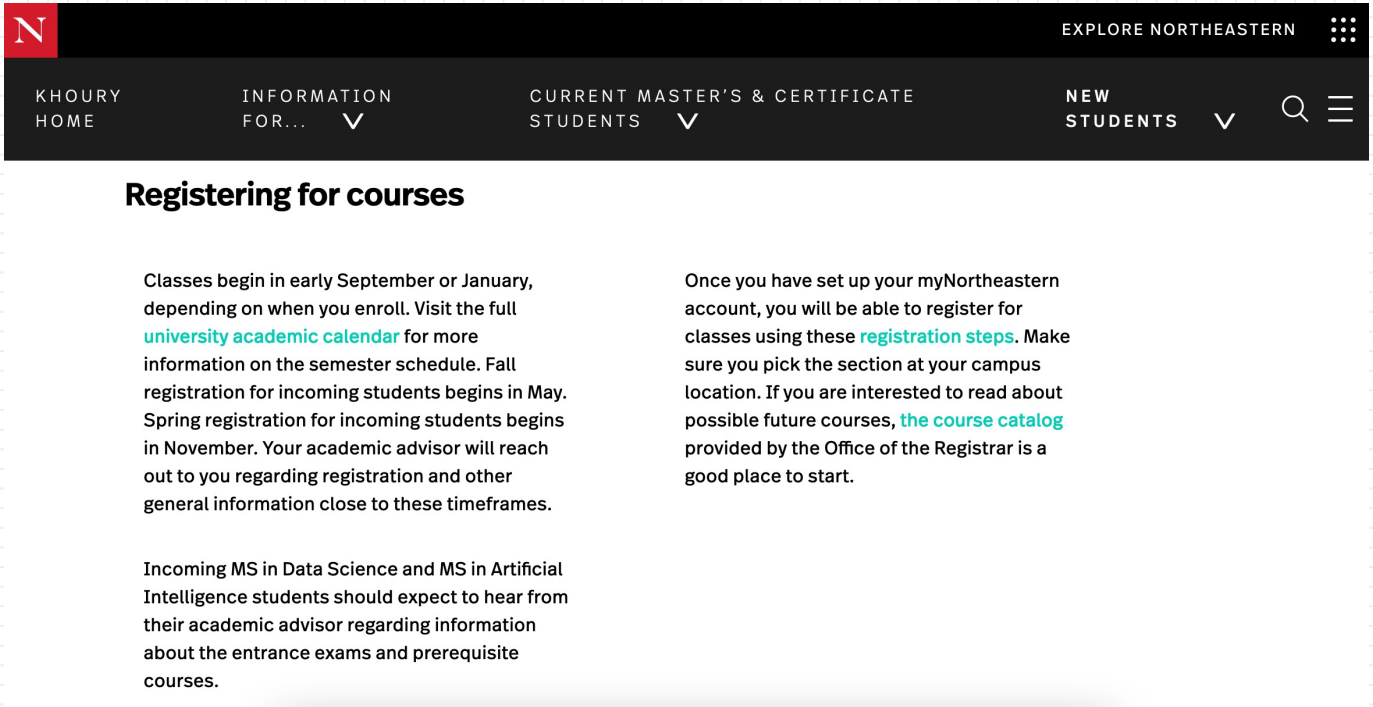
Course Schedule Planning
An Insight of Topological Sort

The Conundrum of Class Registration

A few snapshots of where we are…



Searching the Khoury website for registration steps

A few snapshots of where we are…

Northeastern University
Office of the University Registrar

2022–2023 Graduate Expanded Academic Calendar

Fall 2022

Sep 1 2022 Thu
- First day of "I Am Here" for full-semester and first-half fall classes

Sep 5 2022 Mon
Labor Day (USA), Labour Day (CAN), no classes

Sep 7 2022 Wed
- First day of full-semester and first-half fall classes
- Last day of "I Am Here" for full-semester and first-half fall classes

Sep 20 2022 Tue
- Last day of online class add for full-semester and first-half fall classes
- Last day to drop a first-half fall class without a W grade

Sep 27 2022 Tue
- Last day to drop a full-semester

Oct 24 2022 Mon
- First day of second-half fall classes
- Last day of "I Am Here" for second-half fall classes
- Spring class offerings posted on web

Oct 25 2022 Tue
- Faculty grade deadline at 2:00 p.m. EST for first-half fall classes

Nov 6 2022 Sun
- Last day of online class add for second-half fall classes
- Last day to drop a second-half fall class without a W grade

Nov 10 2022 Thu
- First day of spring class registration

Nov 11 2022 Fri
Veterans Day (USA),

Spring 2023

Jan 3 2023 Tue
- First day of winter intersession classes

Jan 5 2023 Thu
- First day of "I Am Here" for full-semester and first-half spring classes

Jan 6 2023 Fri
- Last day of winter intersession classes

Jan 9 2023 Mon
- First day of full-semester and first-half spring classes

Jan 10 2023 Tue
- Last day of "I Am Here" for full-semester and first-half spring classes

Jan 16 2023 Mon
Martin Luther King, Jr. Day (USA),

Feb 27 2023 Mon
- First day of second-half spring classes
- Last day of "I Am Here" for second-half spring classes

Feb 28 2023 Tue
- Faculty grade deadline at 2:00 p.m. EST for first-half spring classes

Mar 6 2023 Mon
- First day of spring break

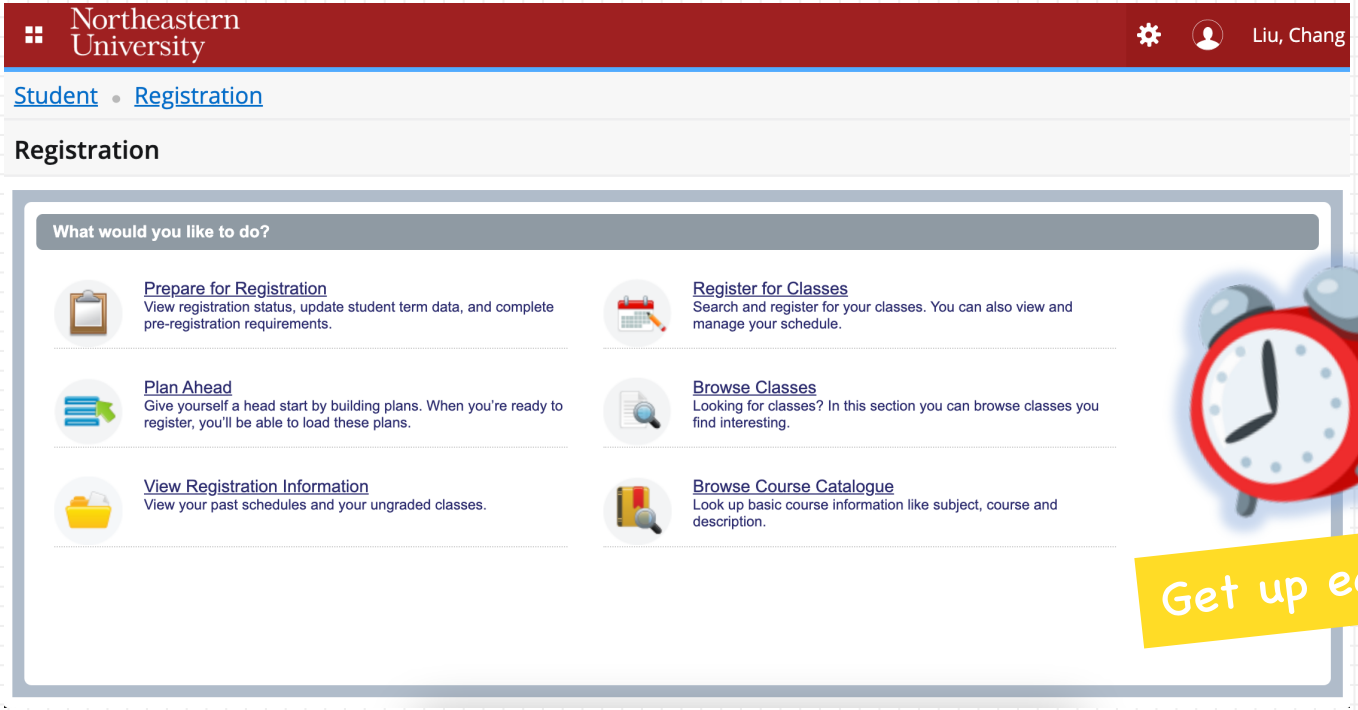
Mar 13 2023 Mon
- Classes resume

Mar 19 2023 Sun
- Last day of online class add for second-half spring classes
- Last day to drop a second-half spring class without a W grade

Mar 20 2023 Mon
- Fall class offerings posted on web

Checking the academic calendar for semester schedule

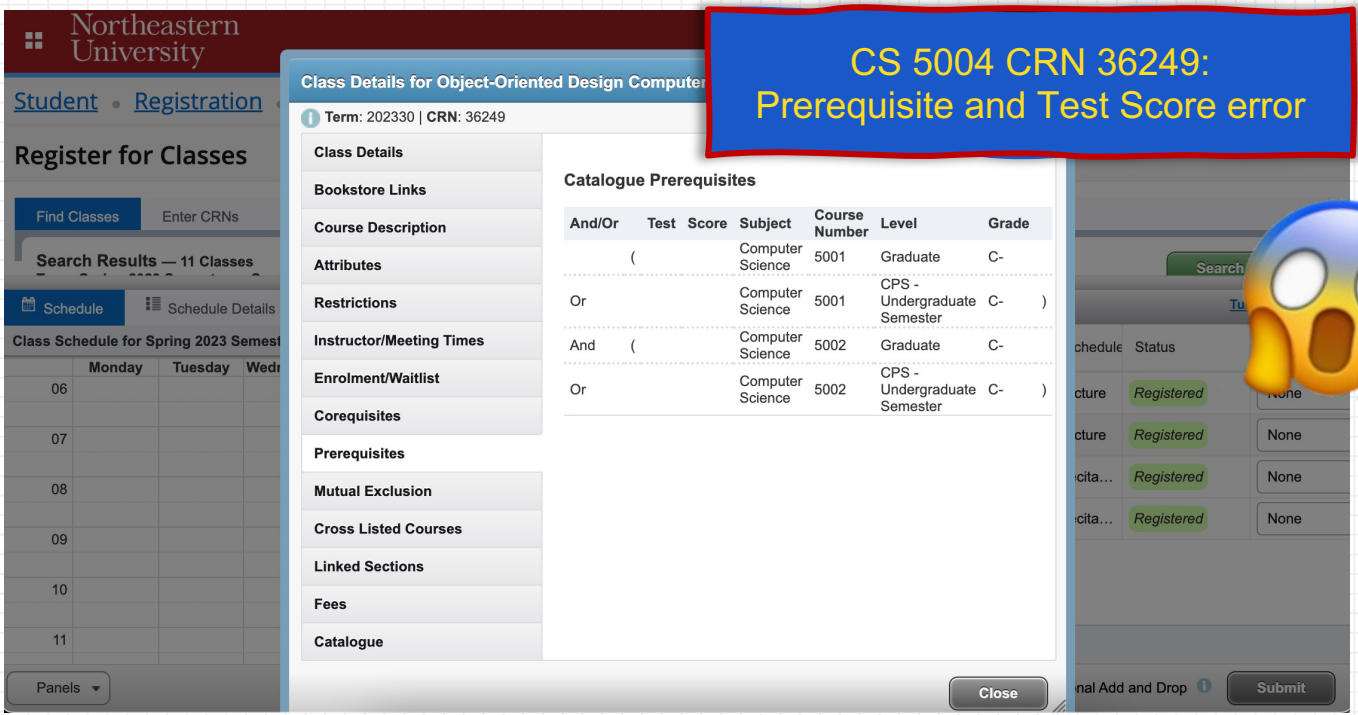
A few snapshots of where we are...



Get up early!!

Doing all homework before registering

Then comes the "oh no!" moment



CS 5004 CRN 36249:
Prerequisite and Test Score error

"What? I had no idea these courses are interrelated! "

AND
LET'S DO MATH !

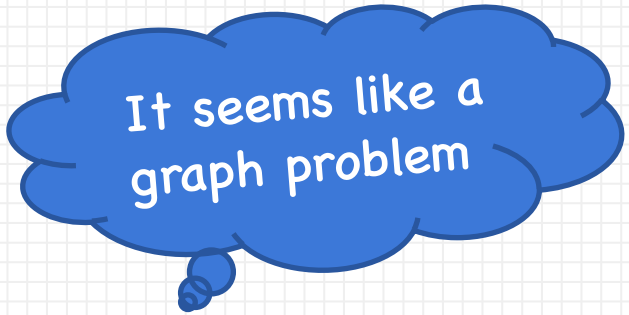
First, let's make an educated guess

Real World

- Courses
- Relations
- The Plan

Math Universe

- Vertices
- Directed edges
- A Graph

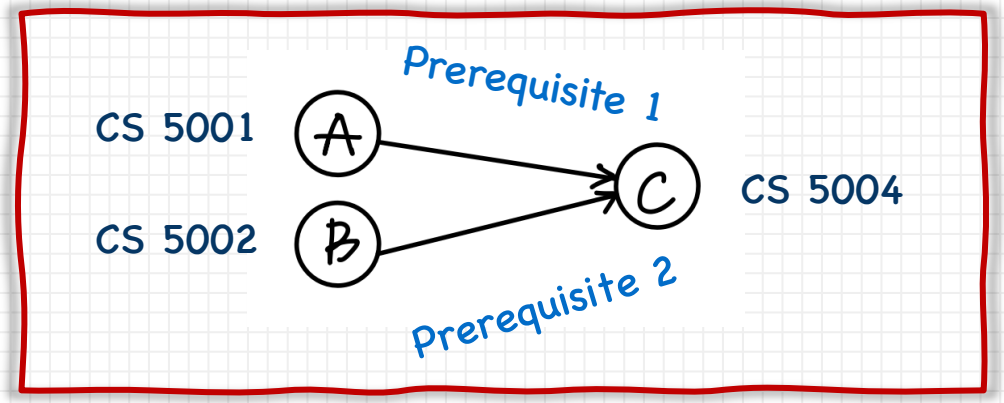


Indeed, it is a graph problem!

Real World		Math Universe
• Courses	———	• Vertices
• Relations	———	• Directed edges
• The Plan	———	• A Graph

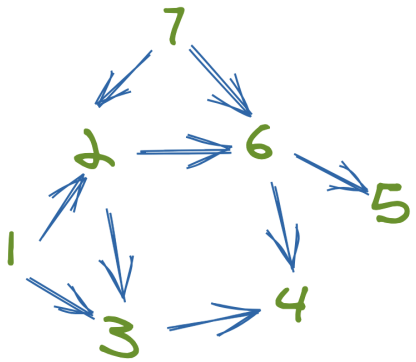
What is the simplest graph we can draw?

Real World		Math Universe
• Courses	———	• Vertices
• Relations	———	• Directed edges
• The Plan	———	• A Graph

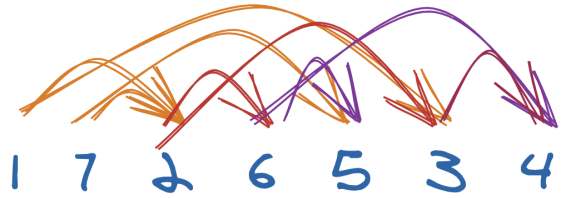


But, what if the problem gets tricker?

Unsorted Graph



Topological Ordering



Disorder → Linear Solution

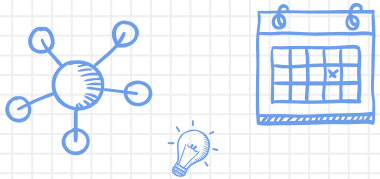
How to sort topologically?

Recall in-class content: how to visit all vertices in a graph?

How to sort topologically?

2 ways to find a valid sequence:

- [x] DFS
- [x] BFS



An obvious method
DFS

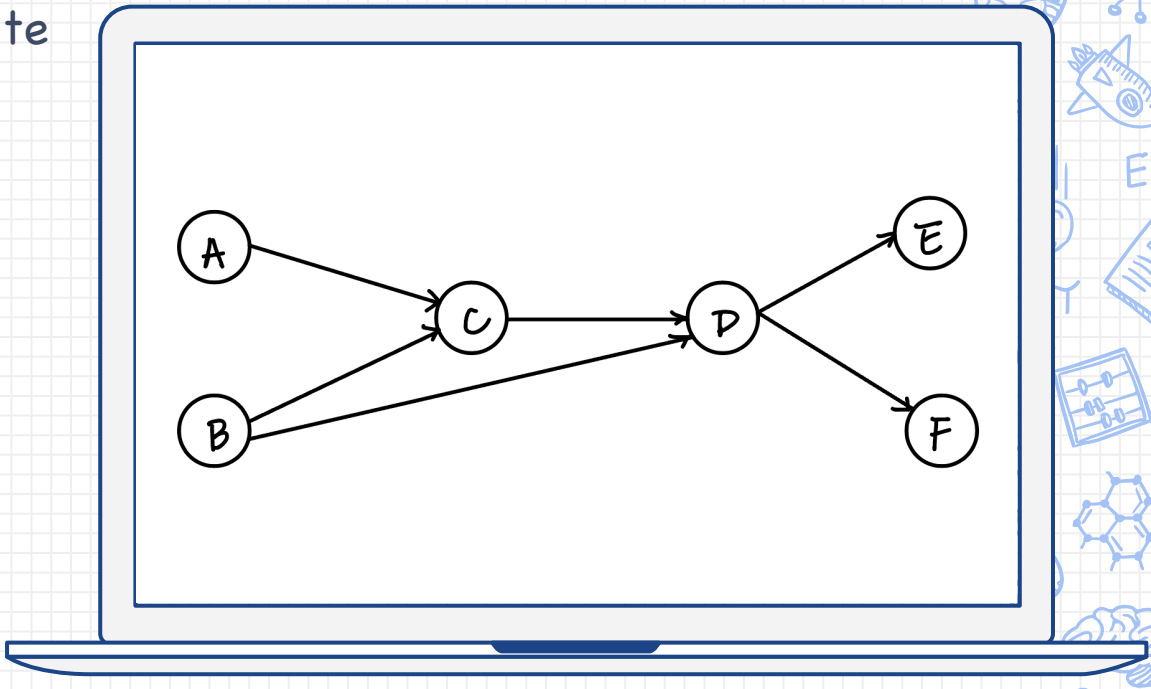
Step 1: Depth-first search

Step 2: Generate a sequence of visited vertices

Step 3: Reverse the sequence

A More Complicated Situation

Course	Prerequisite
A	None
B	None
C	A, B
D	B, C
E	D
F	D



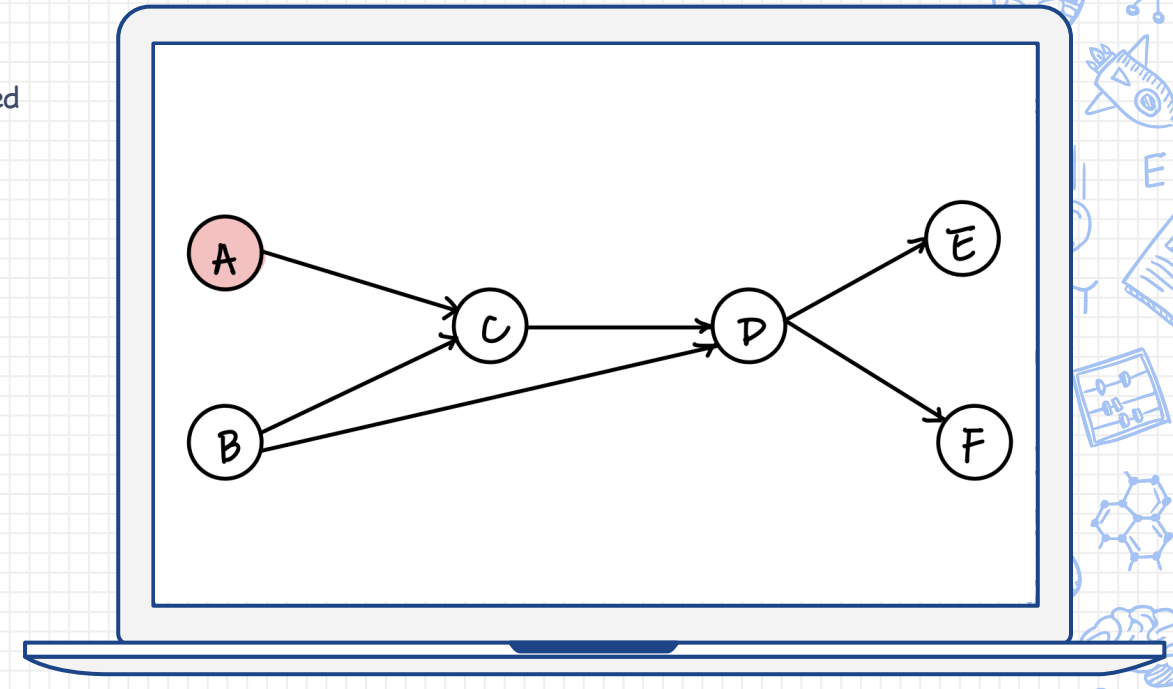
DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.

Visited:



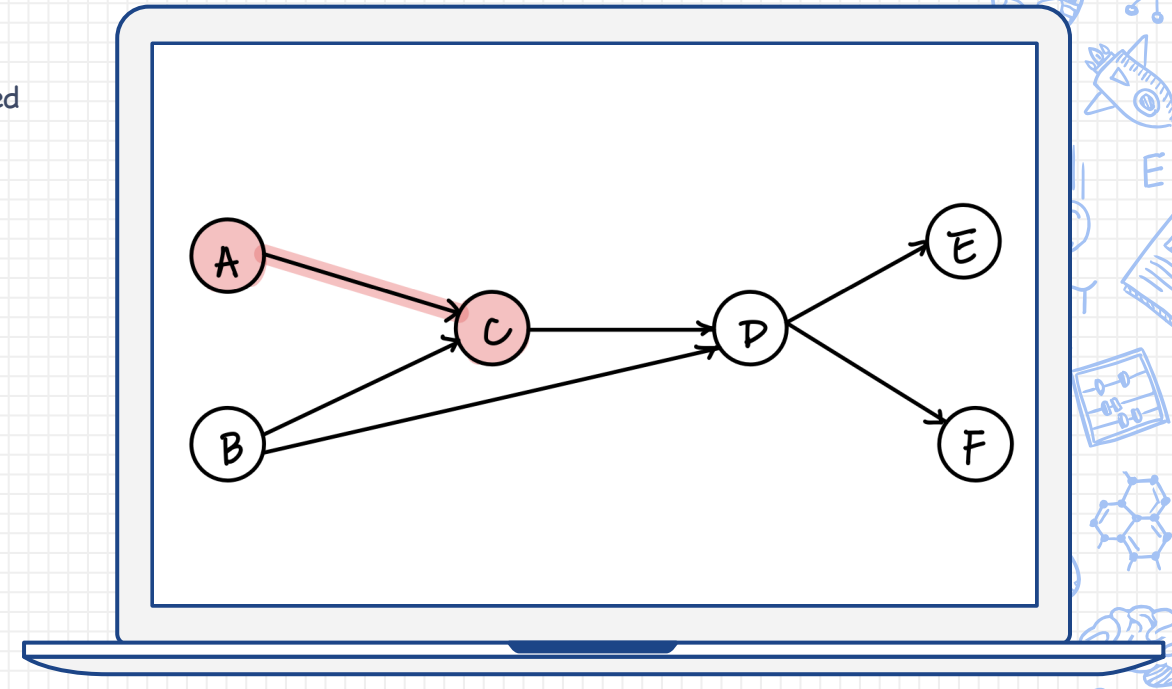
DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.

Visited:



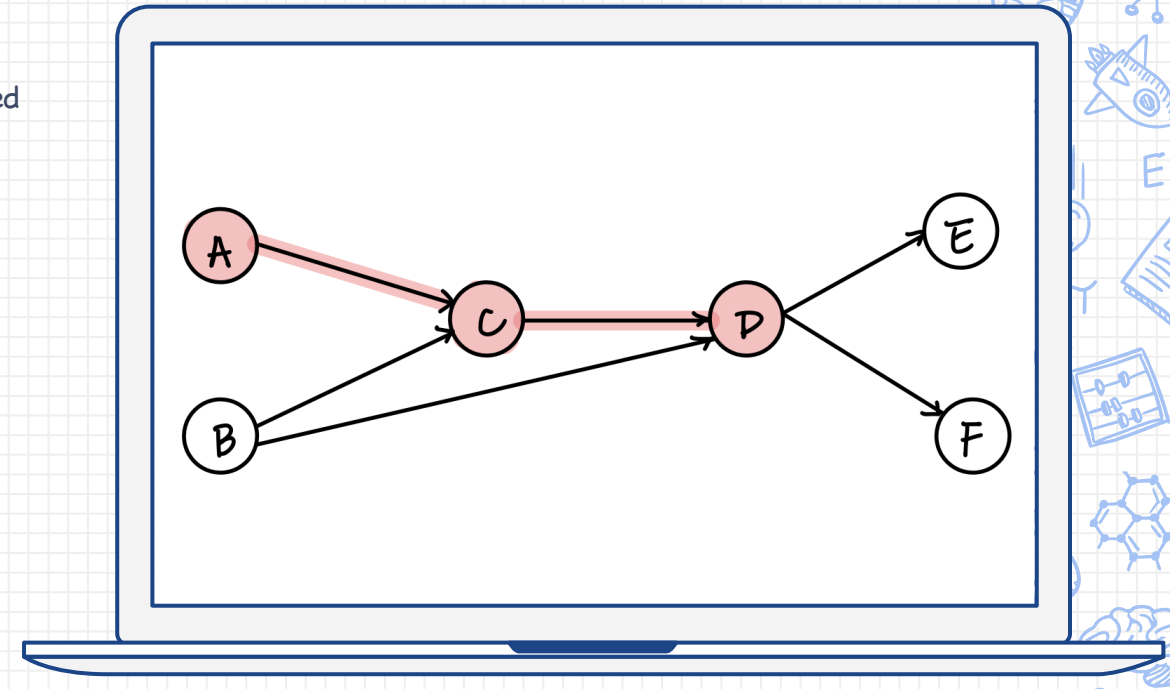
DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.

Visited:



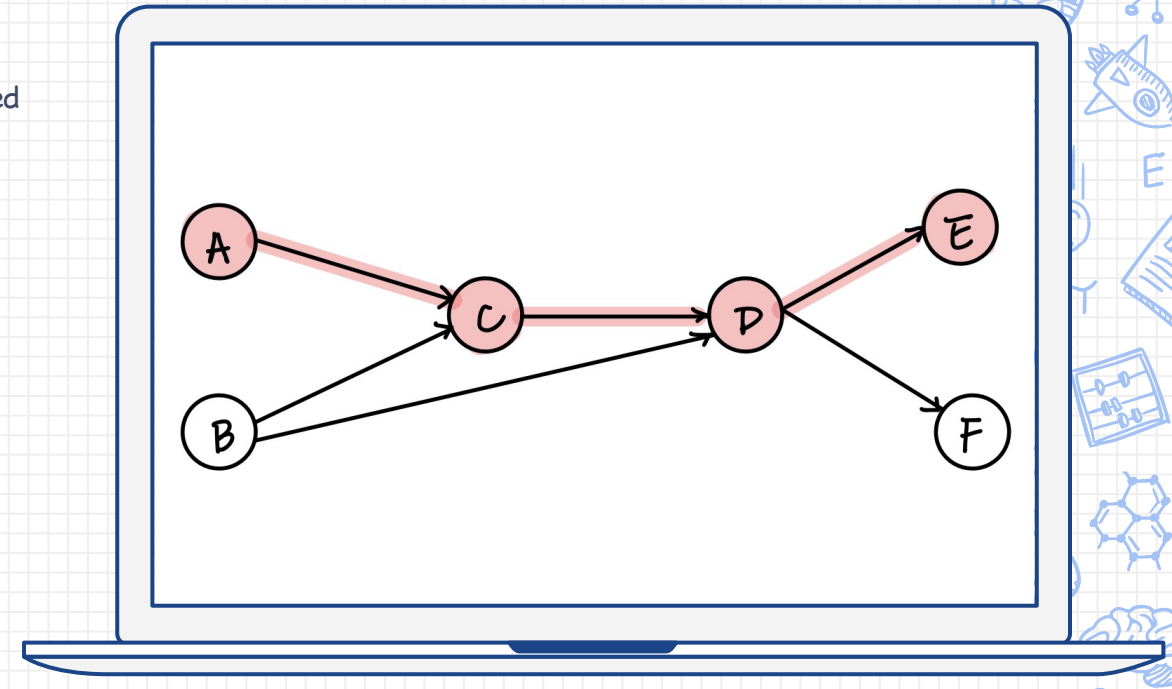
DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.

Visited:

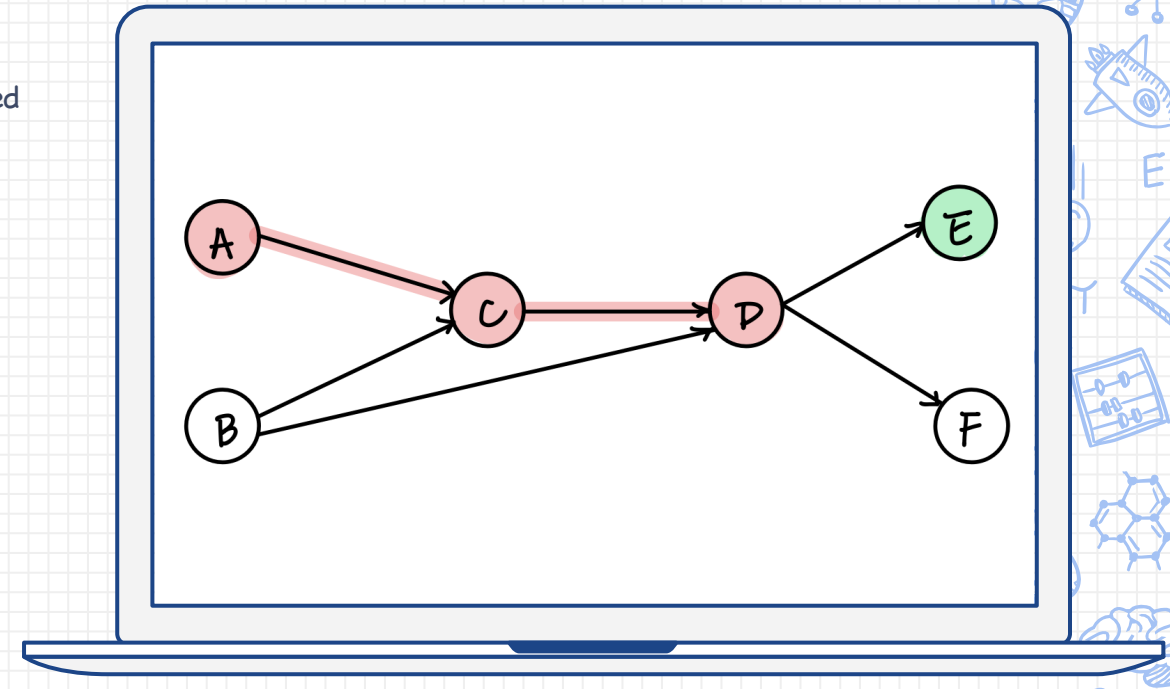


DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

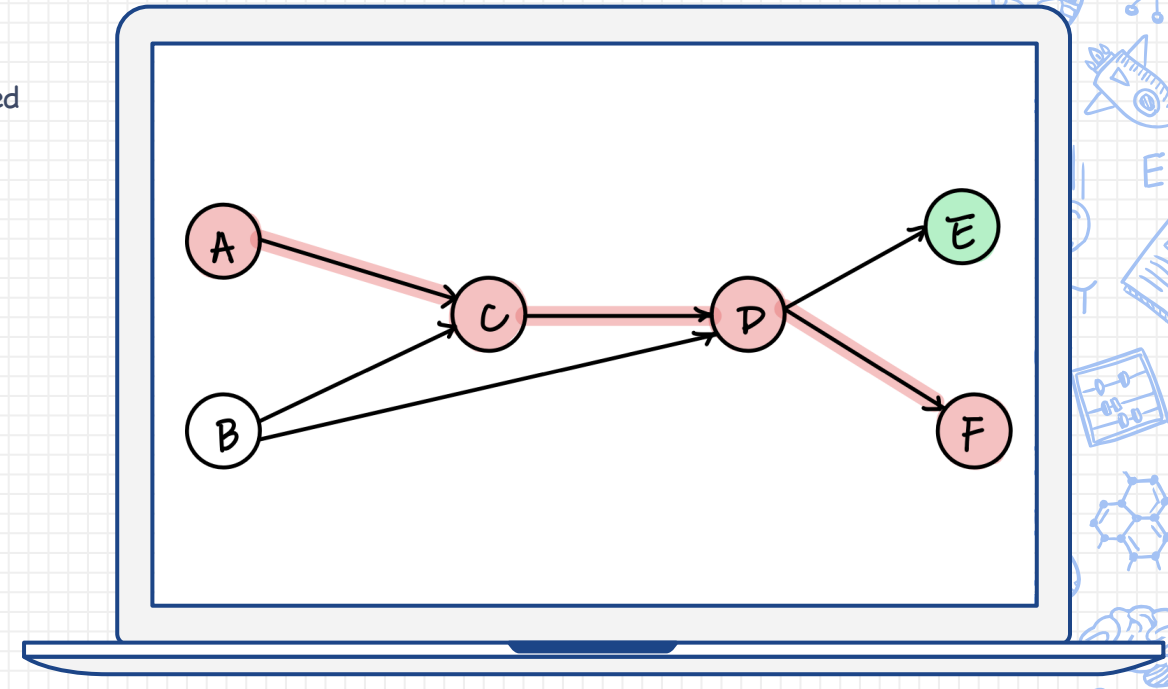
E

DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

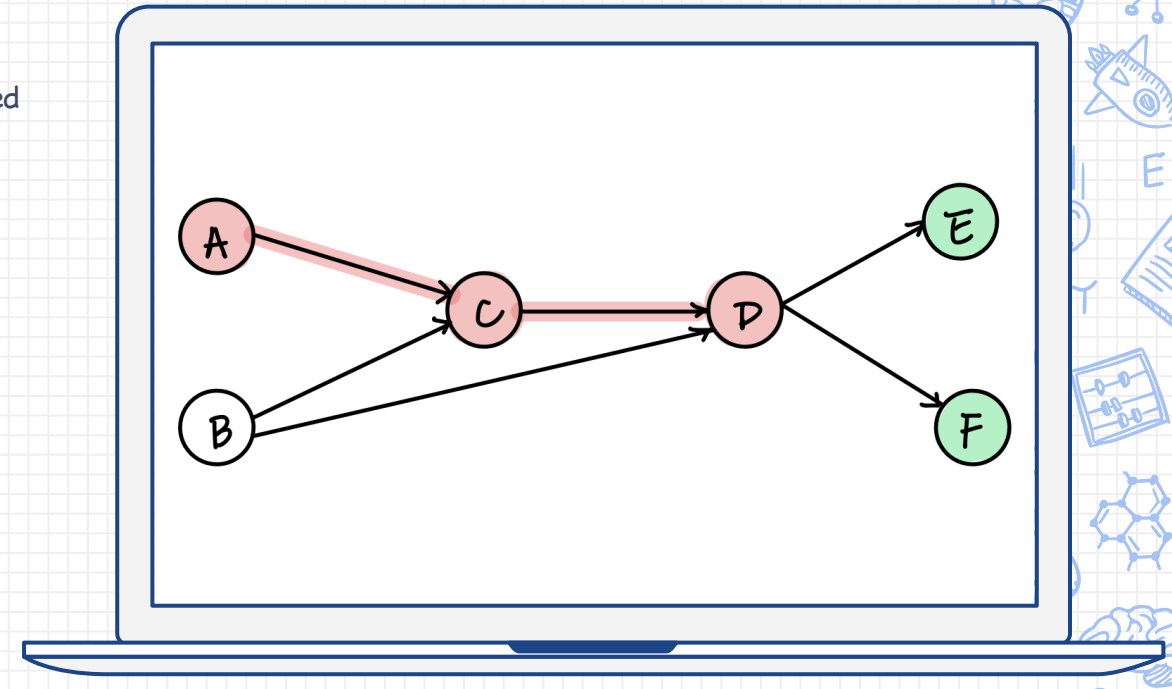
E

DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

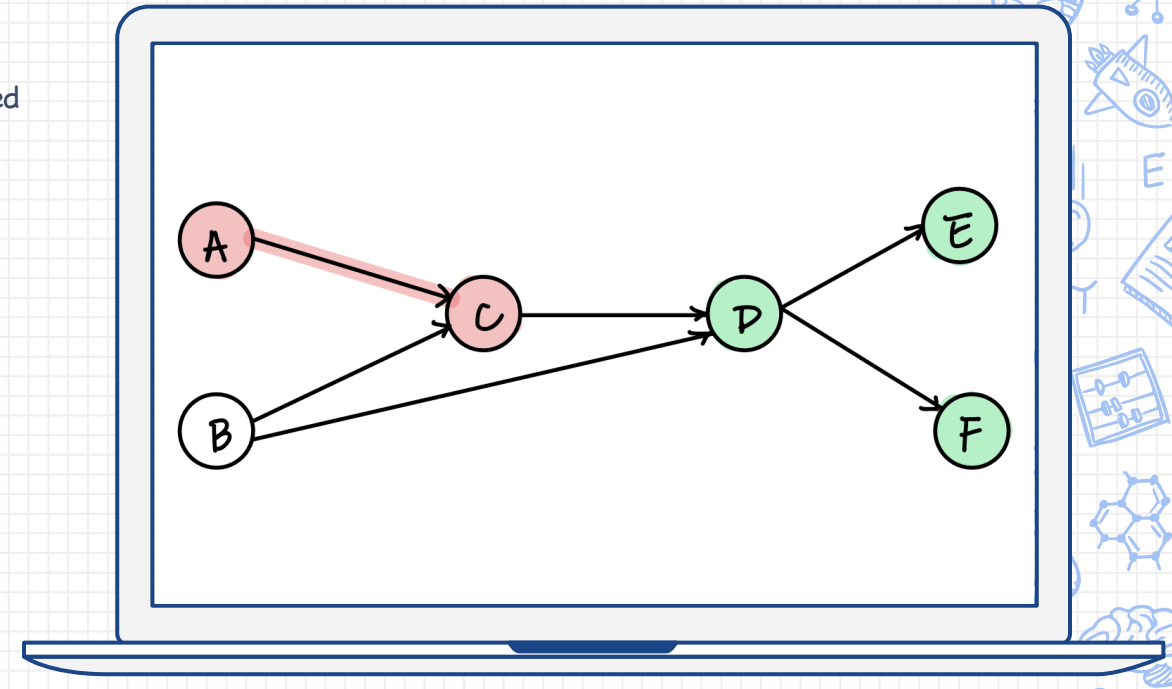
E, F

DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

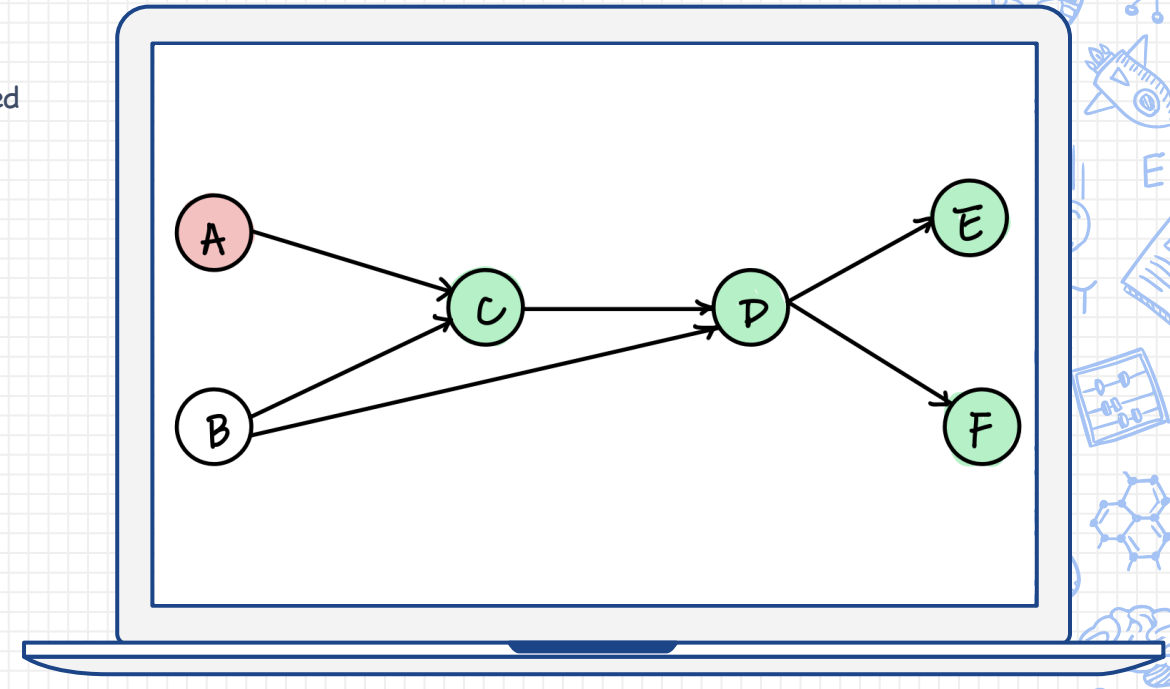
E, F, D

DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

E, F, D, C

DFS to Find a Solution

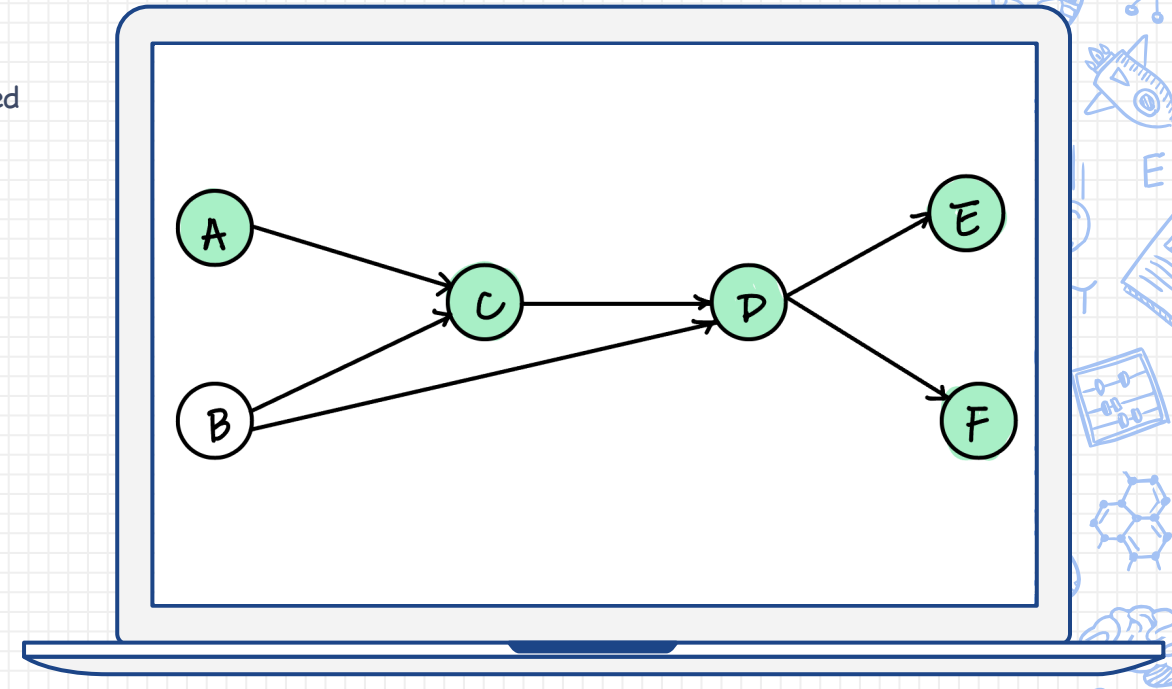
Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.

Visited:

E, F, D, C, A

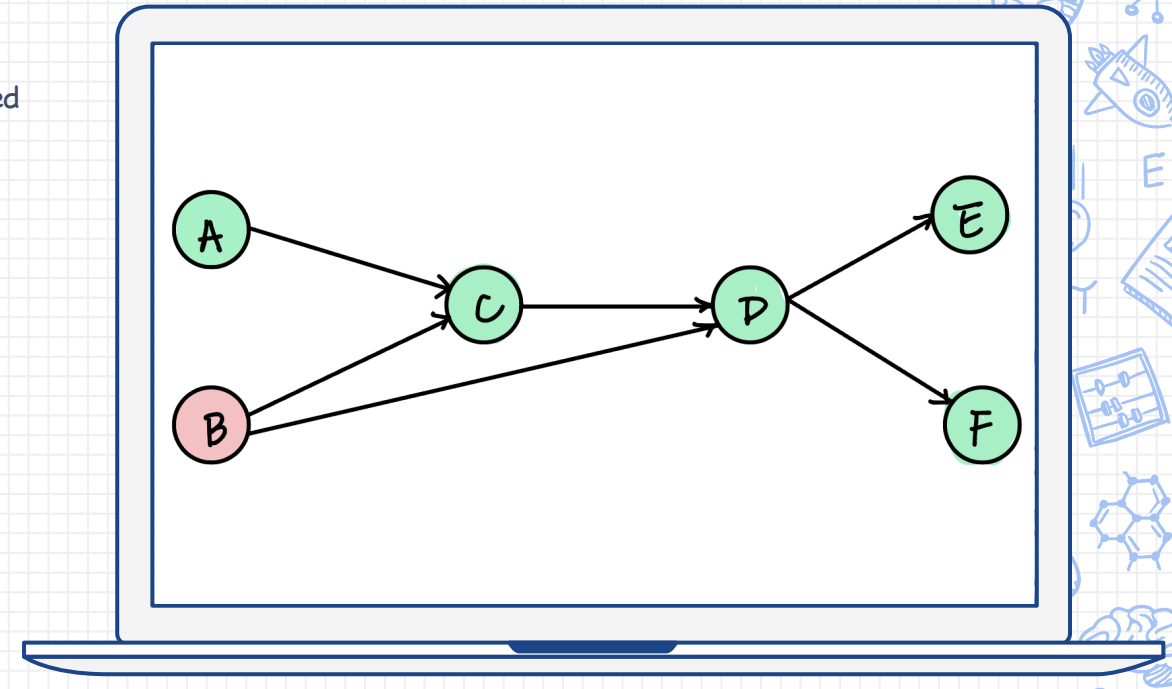


DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

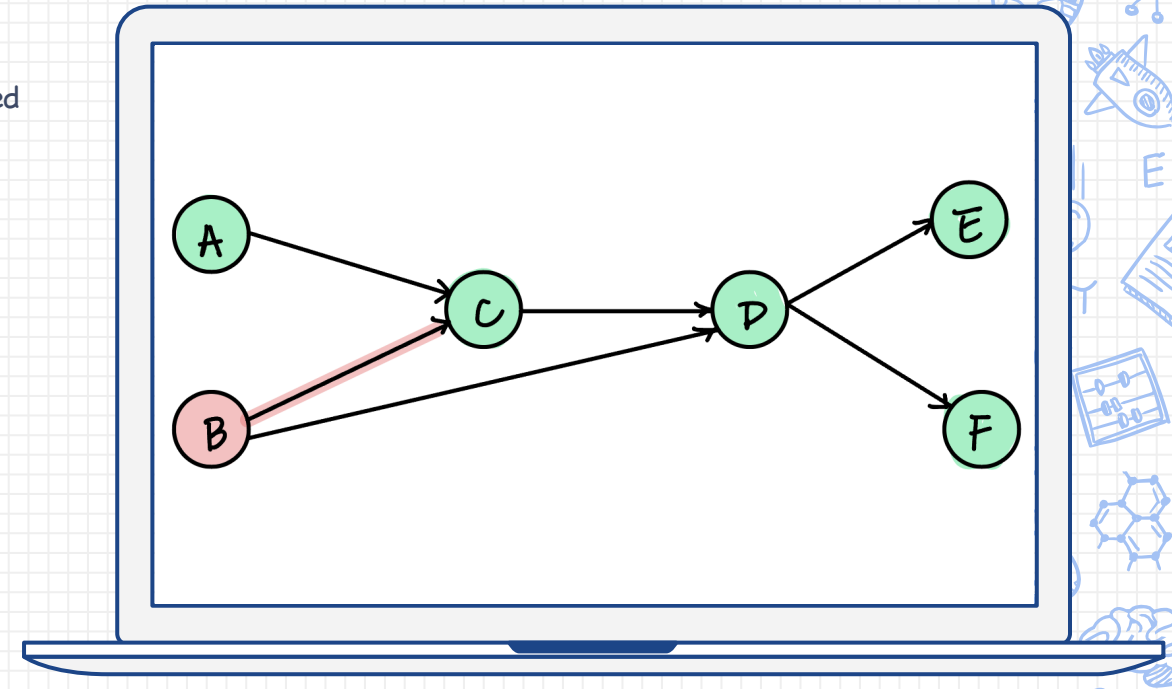
E, F, D, C, A

DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

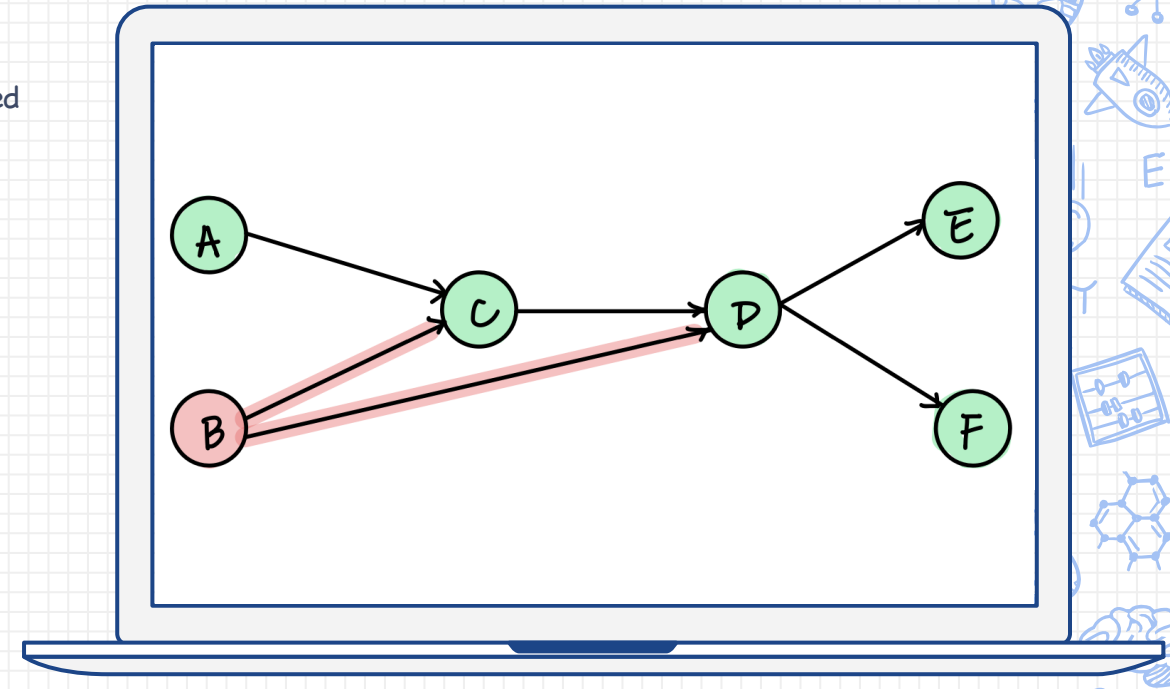
E, F, D, C, A

DFS to Find a Solution

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

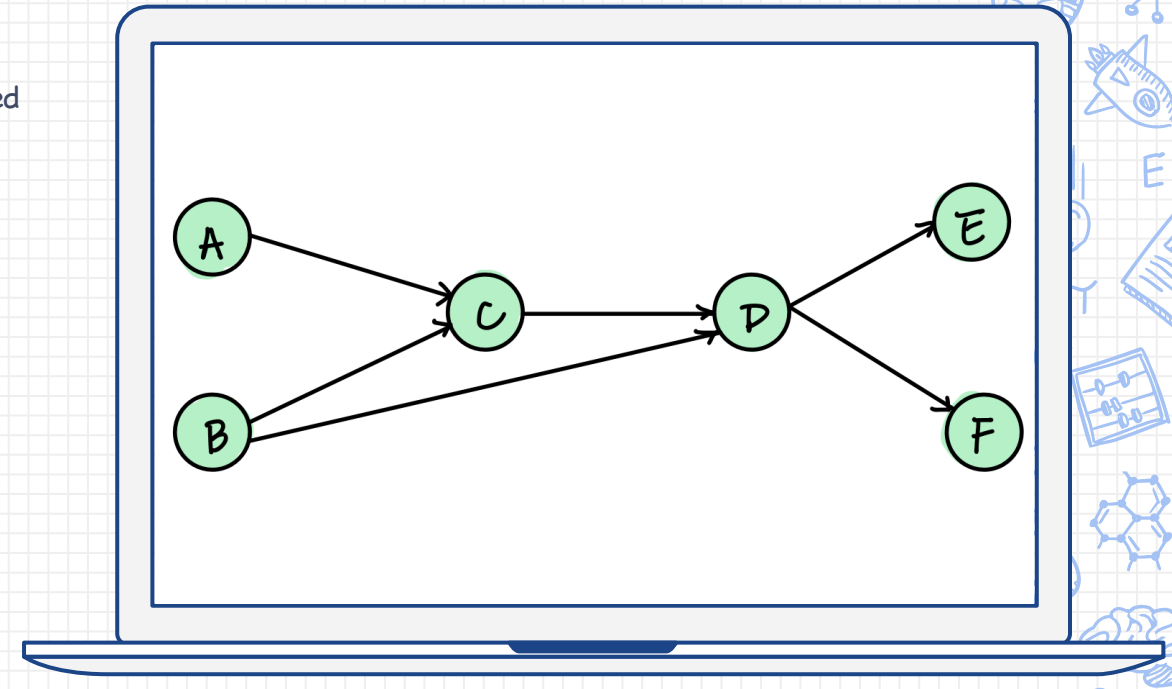
E, F, D, C, A

Then reverse the sequence of visited vertices

Depth-First Search:

Generate a sequence of visited vertices.

Visited vertex: no more un-visited vertices next to the current one.



Visited:

E, F, D, C, A, B

Then reverse the sequence of visited vertices

Depth-First Search:

Generate a sequence of visited vertices.

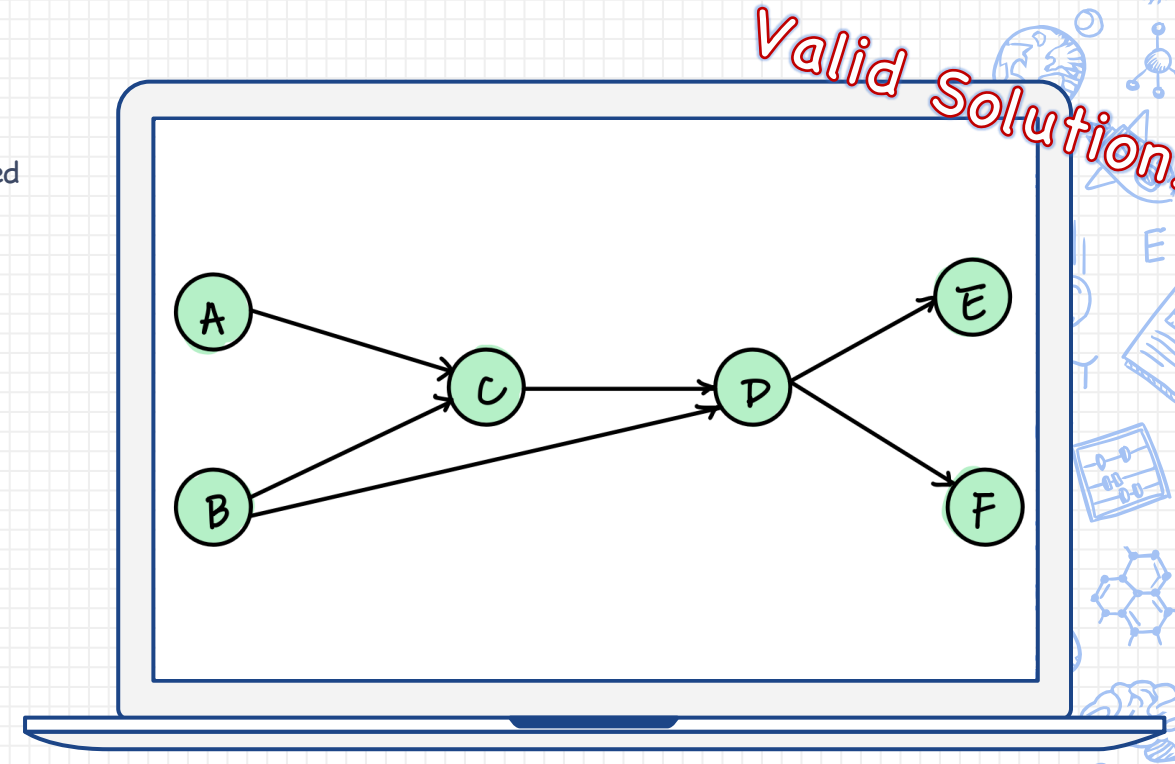
Visited vertex: no more un-visited vertices next to the current one.

Visited:

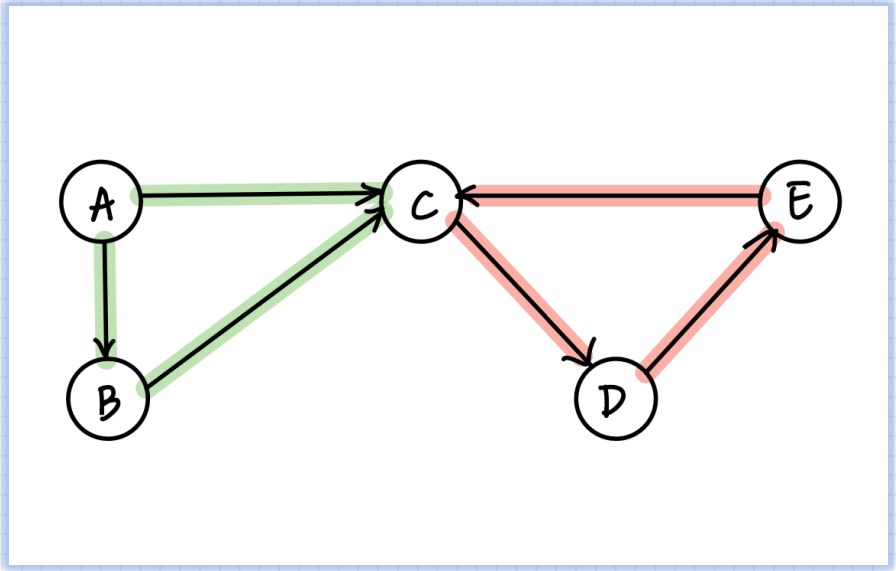
E, F, D, C, A, B

Reverse it:

B→A→C→D→F→E



However, DFS cannot deal with cycles in graph

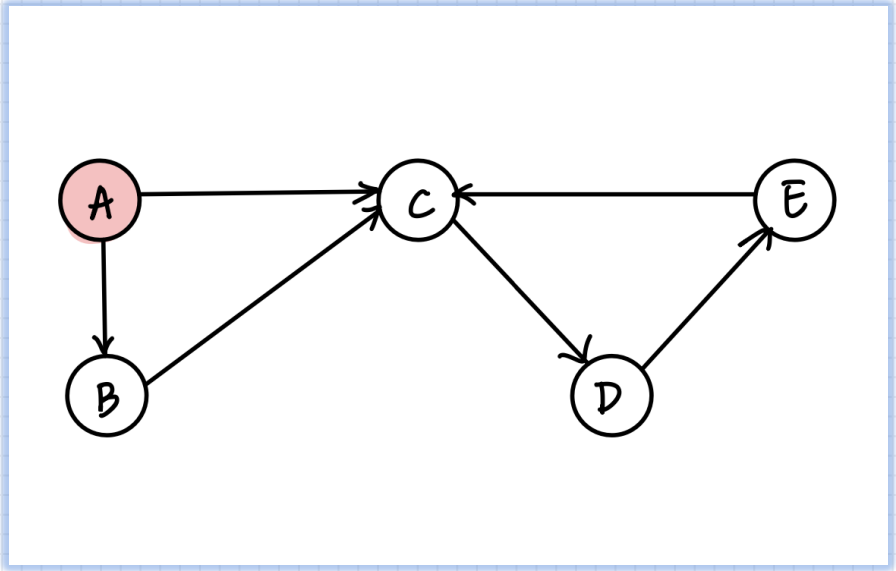


A, B, C: not a cycle

C, D, E: a cycle

No solution here

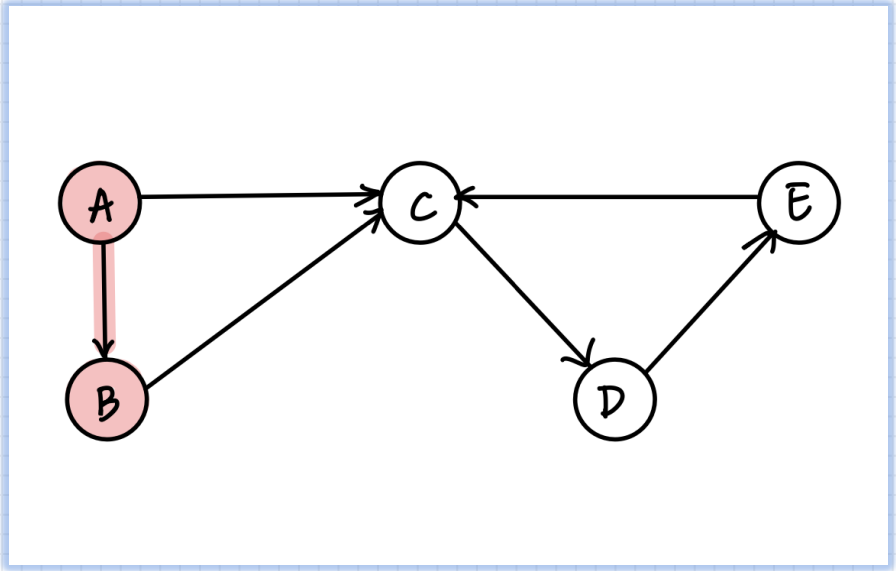
DFS still generates a sequence when there is a cycle



Visited:

No solution here

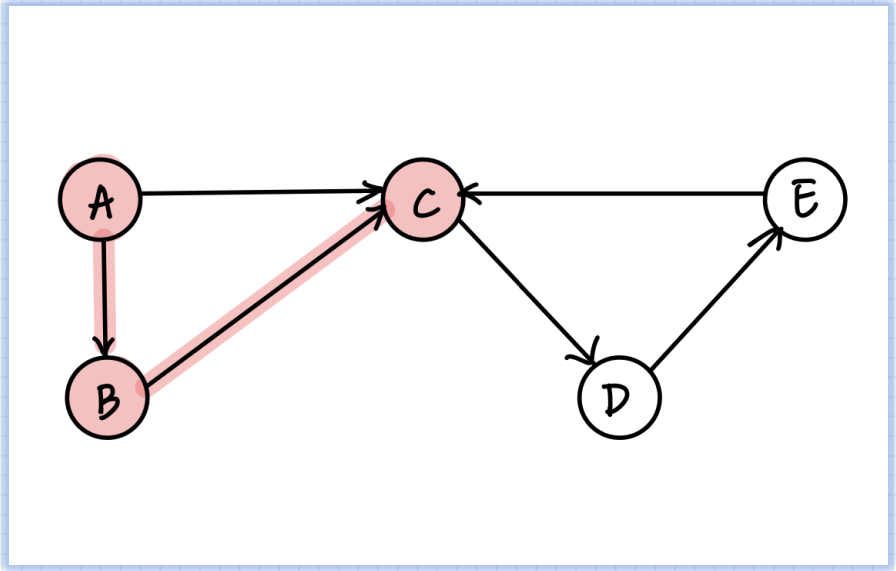
DFS still generates a sequence when there is a cycle



Visited:

No solution here

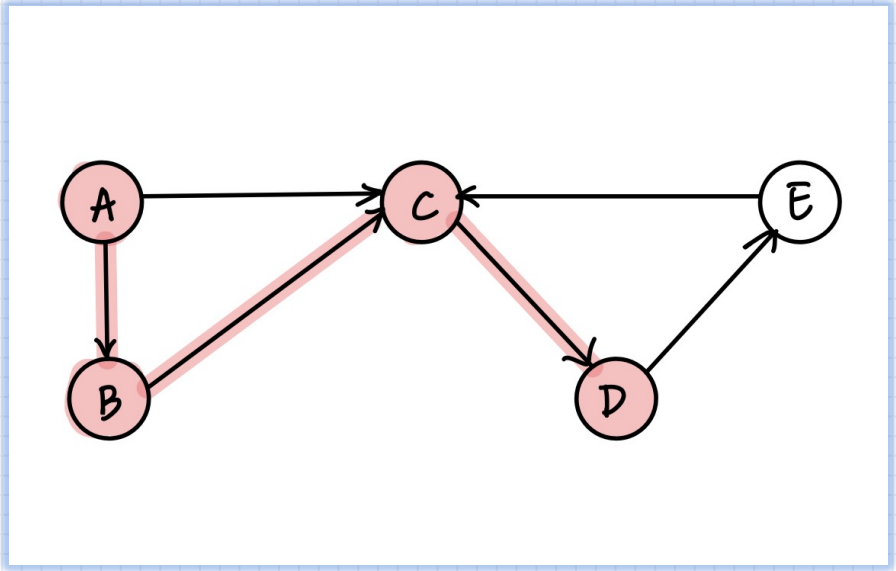
DFS still generates a sequence when there is a cycle



Visited:

No solution here

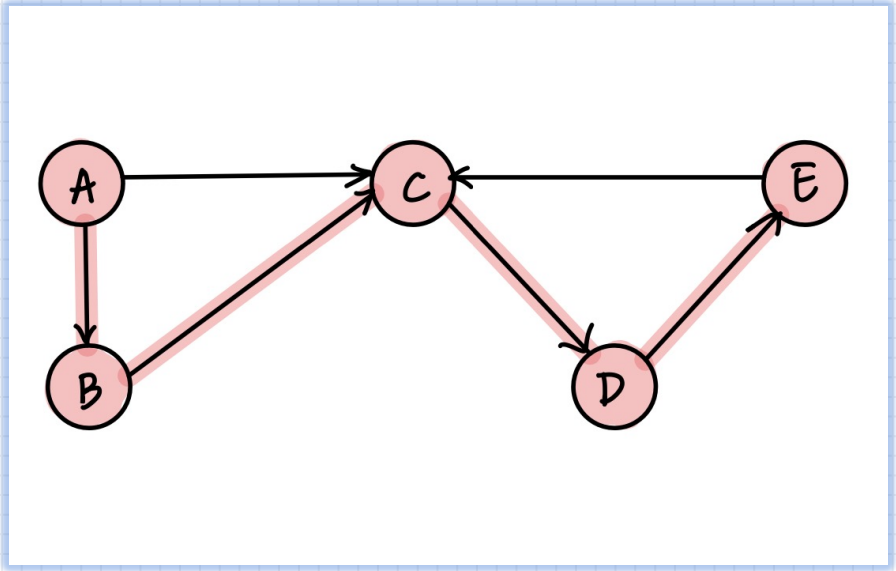
DFS still generates a sequence when there is a cycle



Visited:

No solution here

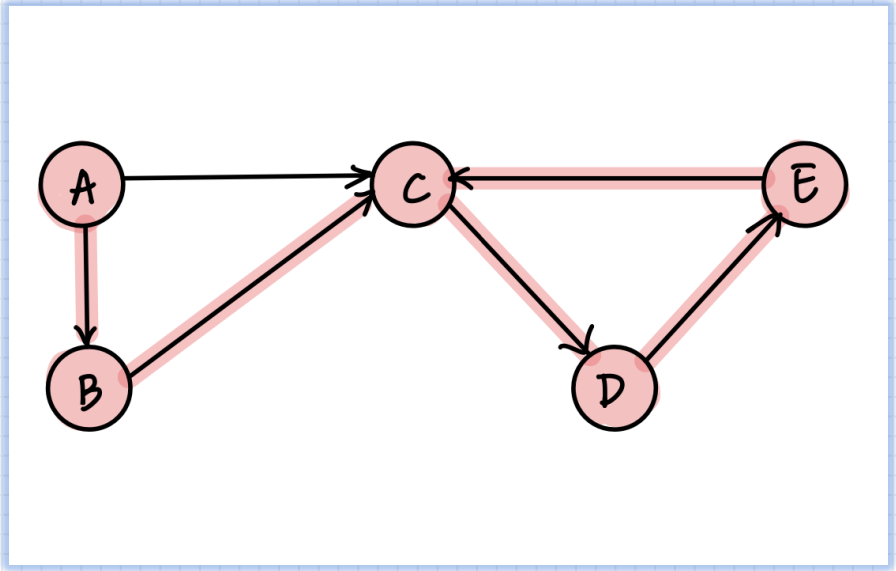
DFS still generates a sequence when there is a cycle



Visited:

No solution here

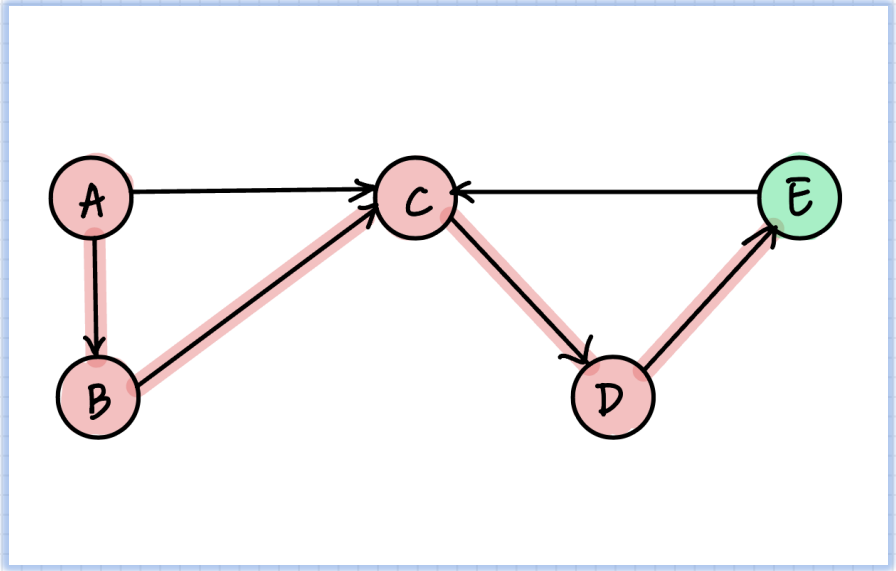
DFS still generates a sequence when there is a cycle



Visited:

No solution here

DFS still generates a sequence when there is a cycle

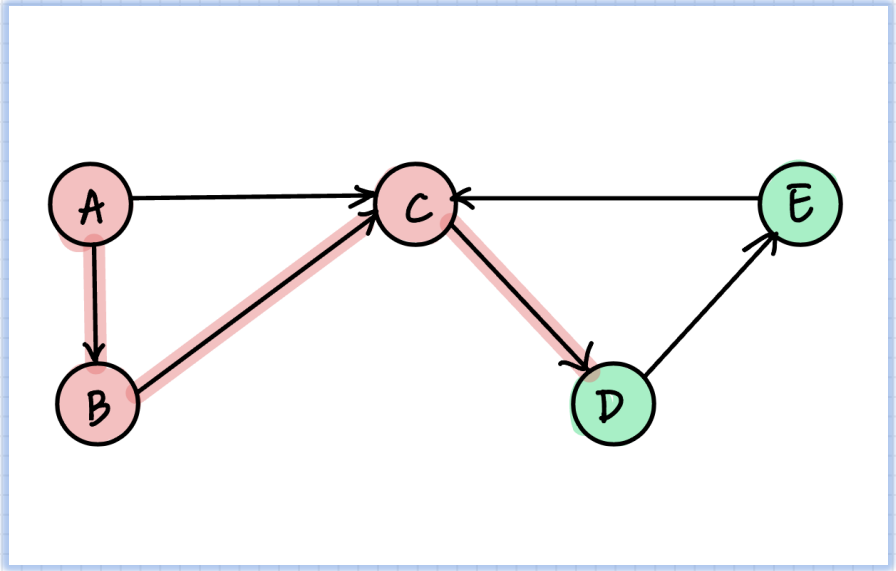


Visited:

E

No solution here

DFS still generates a sequence when there is a cycle

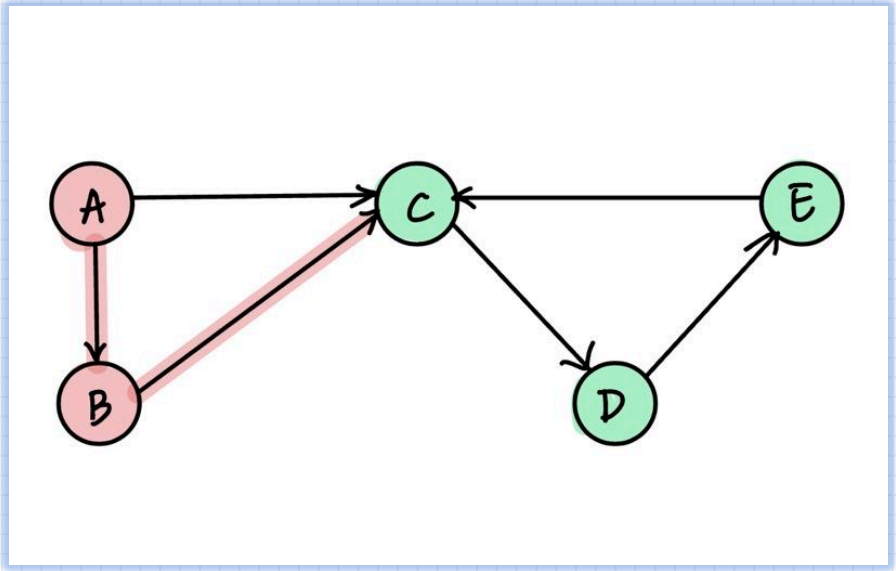


Visited:

E, D

No solution here

DFS still generates a sequence when there is a cycle

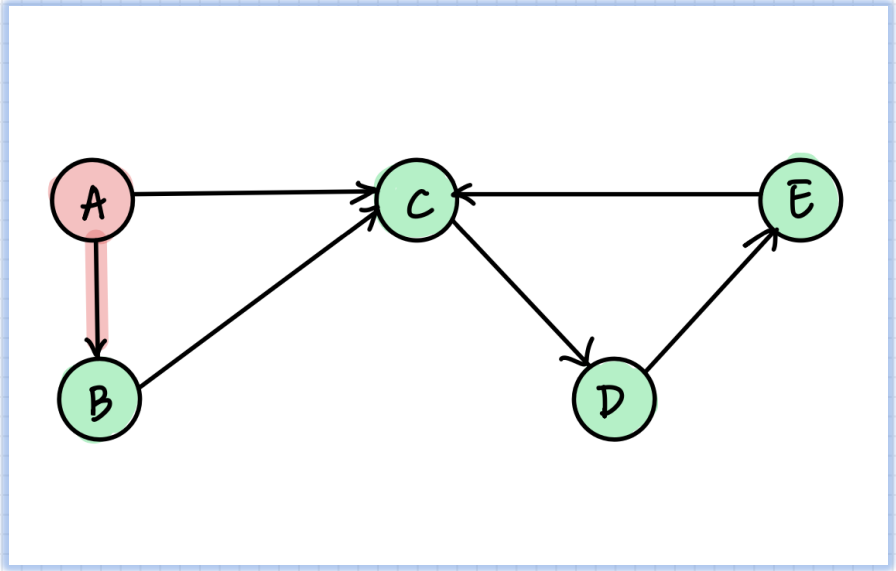


Visited:

E, D, C

No solution here

DFS still generates a sequence when there is a cycle

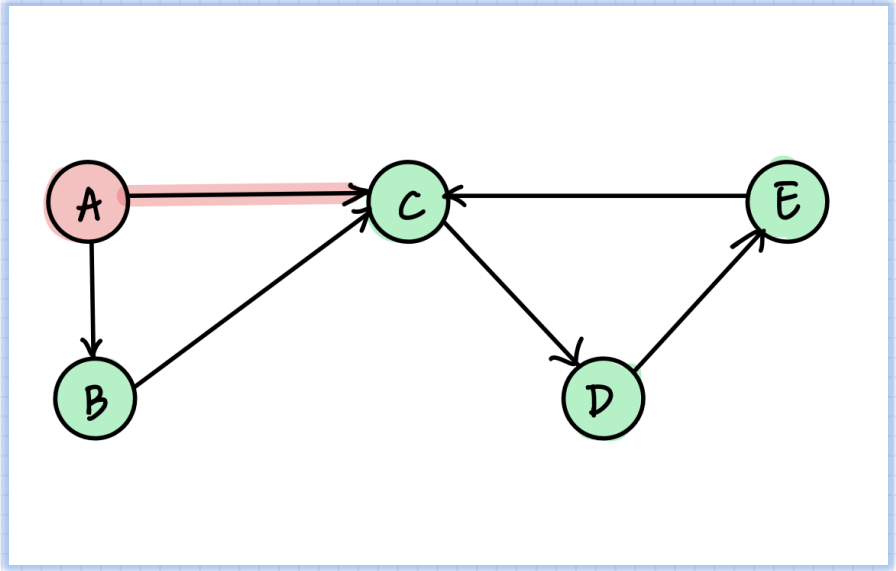


Visited:

E, D, C, B

No solution here

DFS still generates a sequence when there is a cycle

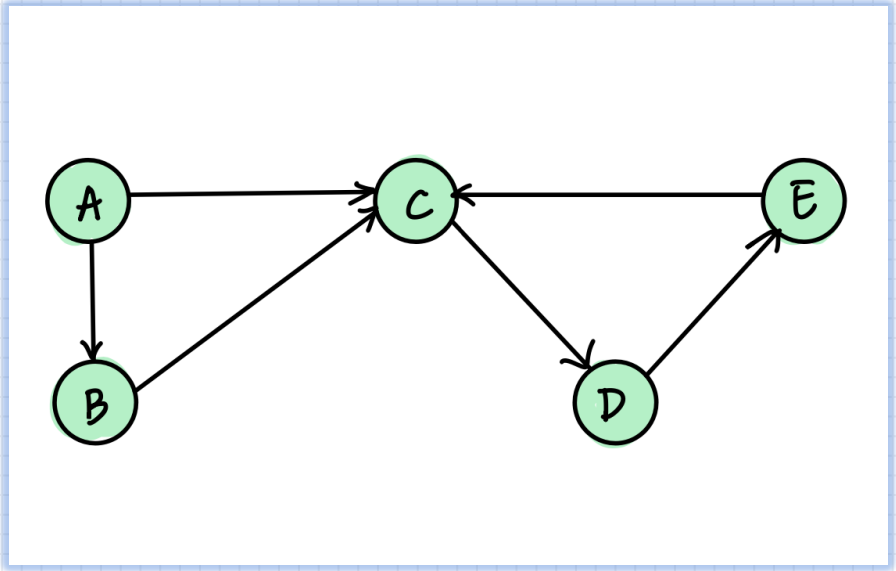


Visited:

E, D, C, B

No solution here

DFS still generates a sequence when there is a cycle



Visited:

E, D, C, B, A

Reverse it:

A→B→C→D→E

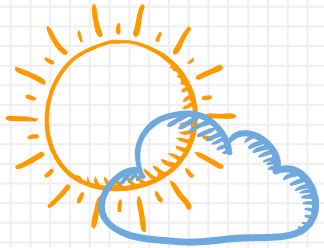
Not Valid Solution!

No solution here

Topological Sort Using DFS

NO CIRCLE IN GRAPH

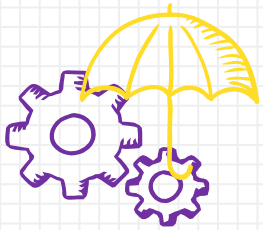
Generate a valid sequence for course taking.

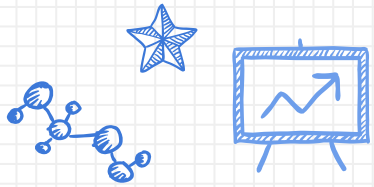


CIRCLE IN GRAPH

Still generate a sequence for course taking.

However, the sequence is invalid, and we won't even know!





A simpler method: BFS

Step 1: Calculating Each Vertex's **In-degree**

Step 2: Take the course of which the in-degree is 0

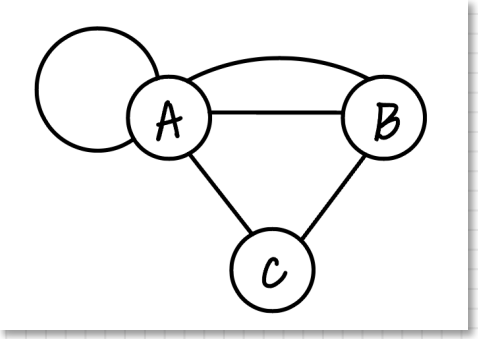
Step 3: Remove the course and its edges

Step 4: Repeat 1-3 until no more in-degree becomes 0

What is in-degree?

Degree:

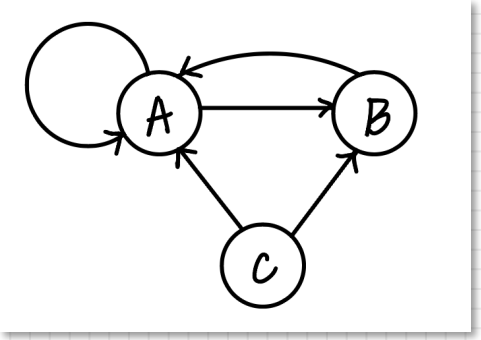
- ✗ In un-directed graph
- ✗ Number of edges connected to a vertex



D(A) = 5
D(B) = 3
D(C) = 2

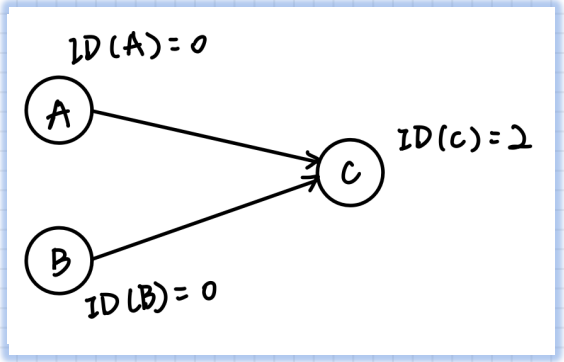
In-Degree:

- ✗ In directed graph
- ✗ Number of edges pointed to a vertex



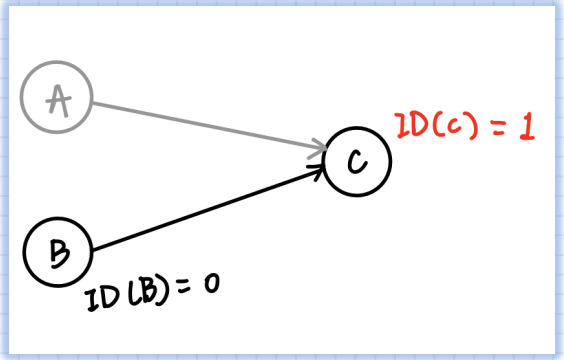
ID(A) = 3
ID(B) = 2
ID(C) = 0

Simple example of BFS



Step 1: Calculate all vertices' in-degree.

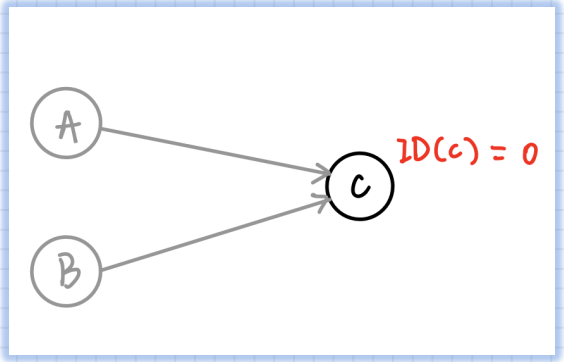
Step 2: Take the course with in-degree as 0.



Step 3: Remove the vertex and its edges.

Step 4: Re-calculate all vertices' in-degree.

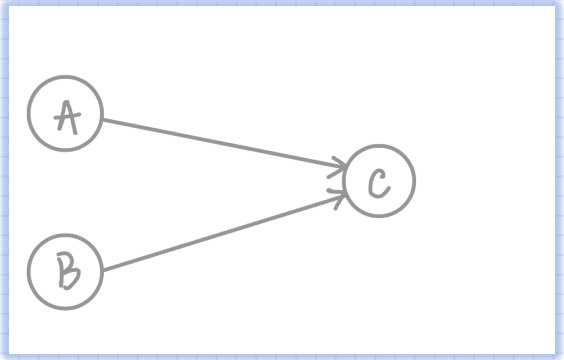
Simple example of BFS



Step 4: Then repeat.

Taking sequence:

A → B → C



In-degree = 0 means:

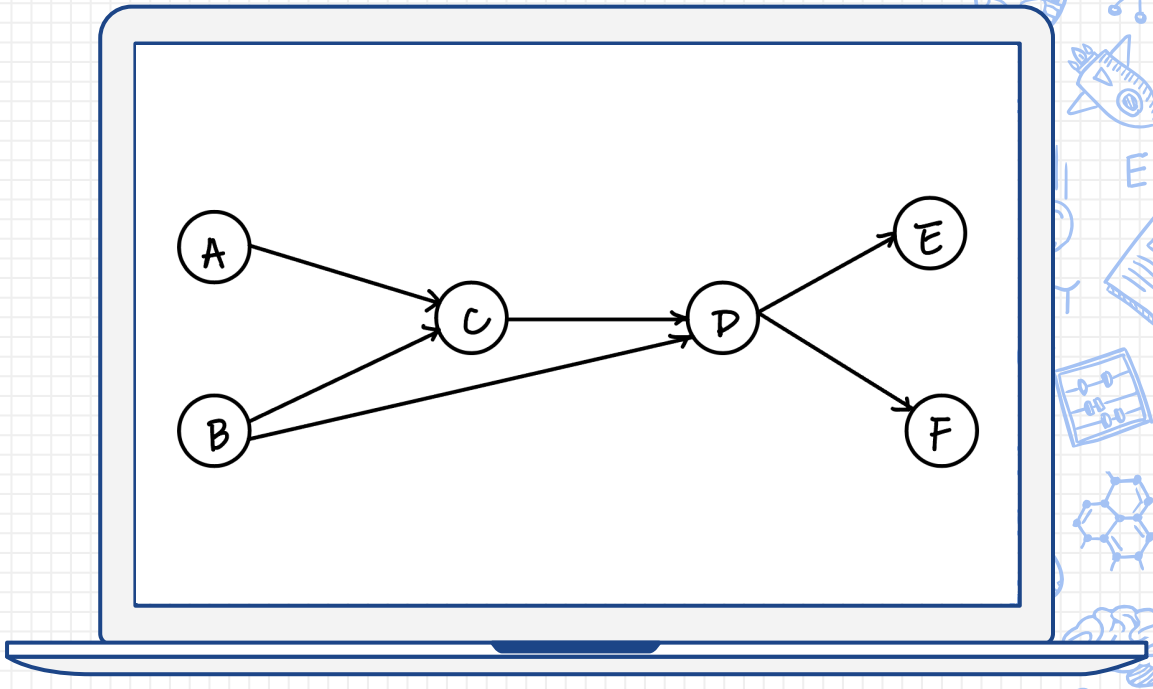
There is no prerequisites

OR

prerequisites are fulfilled.

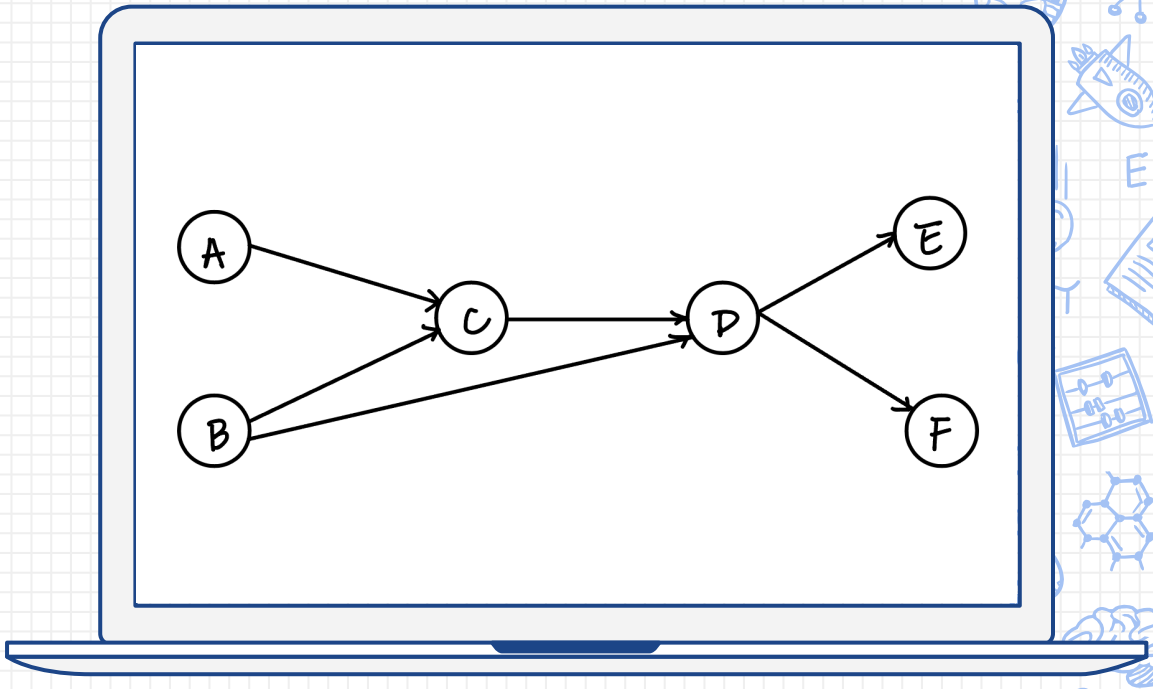
BFS using previous example

Course	Prerequisite
A	None
B	None
C	A, B
D	C
E	D
F	D



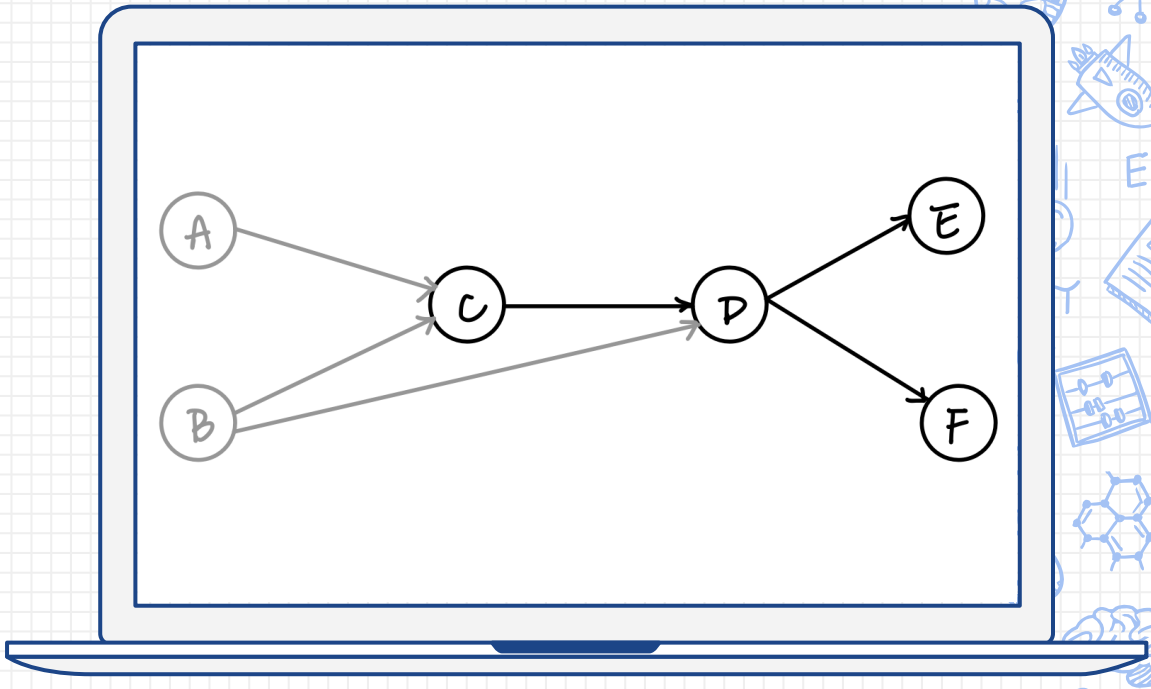
Topological sort: calculate all in-degree

Vertices	In-degree
A	0
B	0
C	2
D	2
E	1
F	1



Topological sort: take the vertices with 0 in-degree

Vertices	In-degree
A	0
B	0
C	2
D	2
E	1
F	1



Taking:

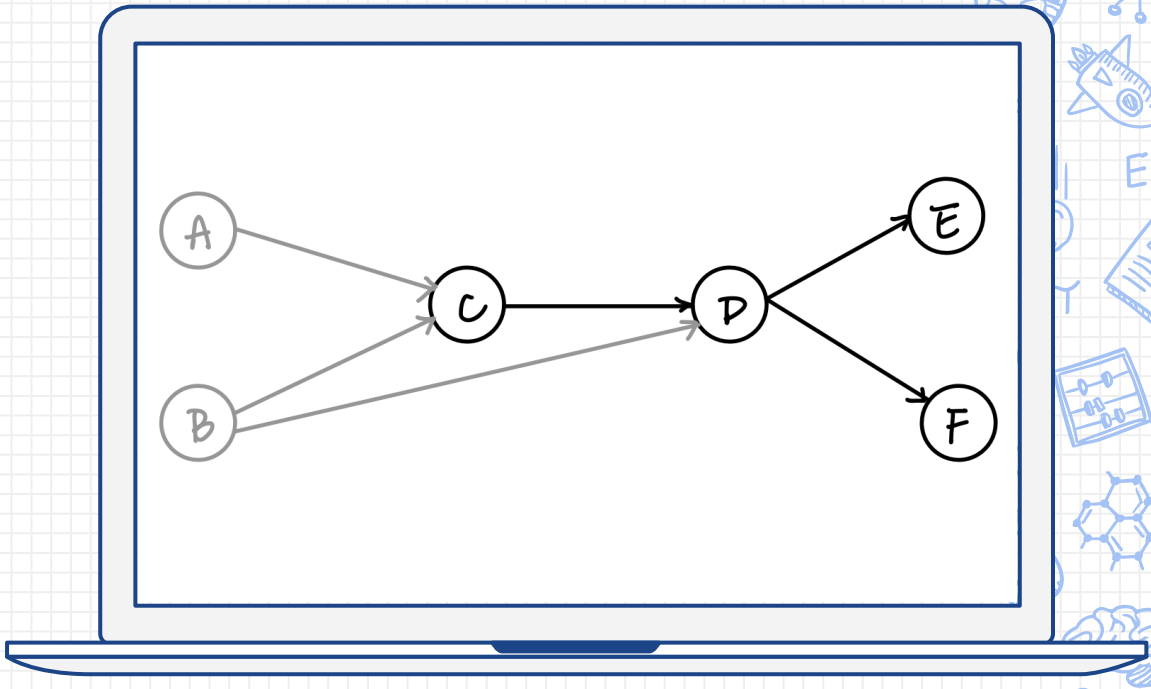
A → B / B → A

Topological sort: remove and update

Vertices	In-degree
A	0
B	0
C	0
D	1
E	1
F	1

Taking:

A → B

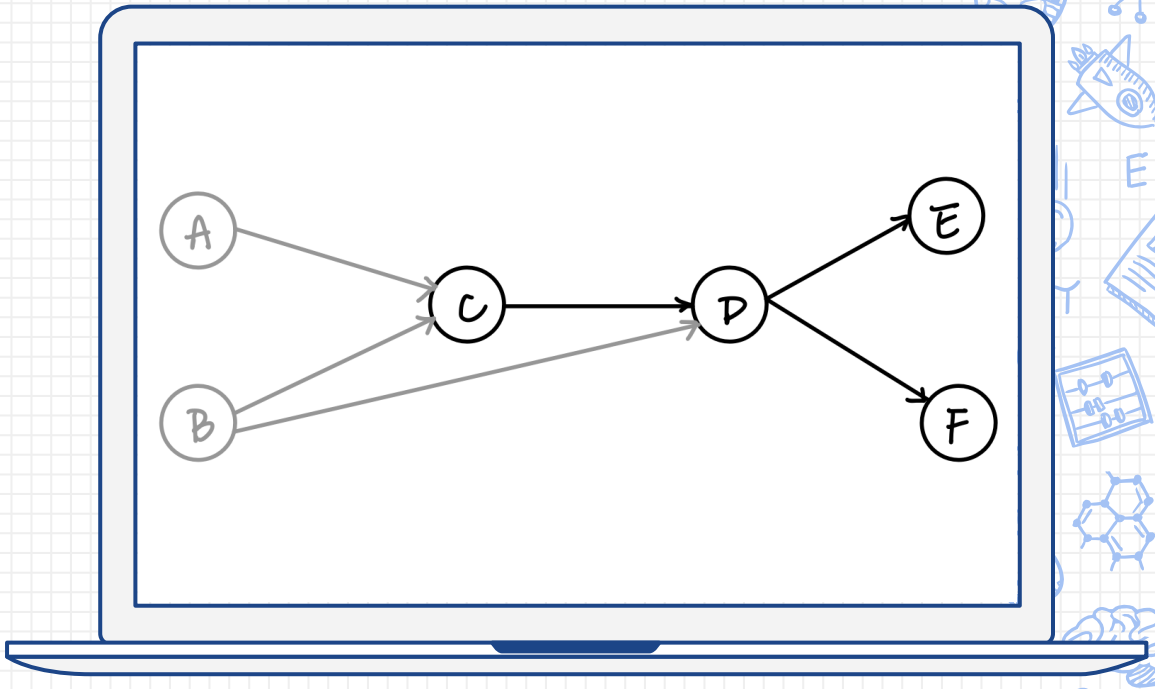


Topological sort: take the vertex with 0 in-degree

Vertices	In-degree
A	0
B	0
C	0
D	1
E	1
F	1

Taking:

A → B → C

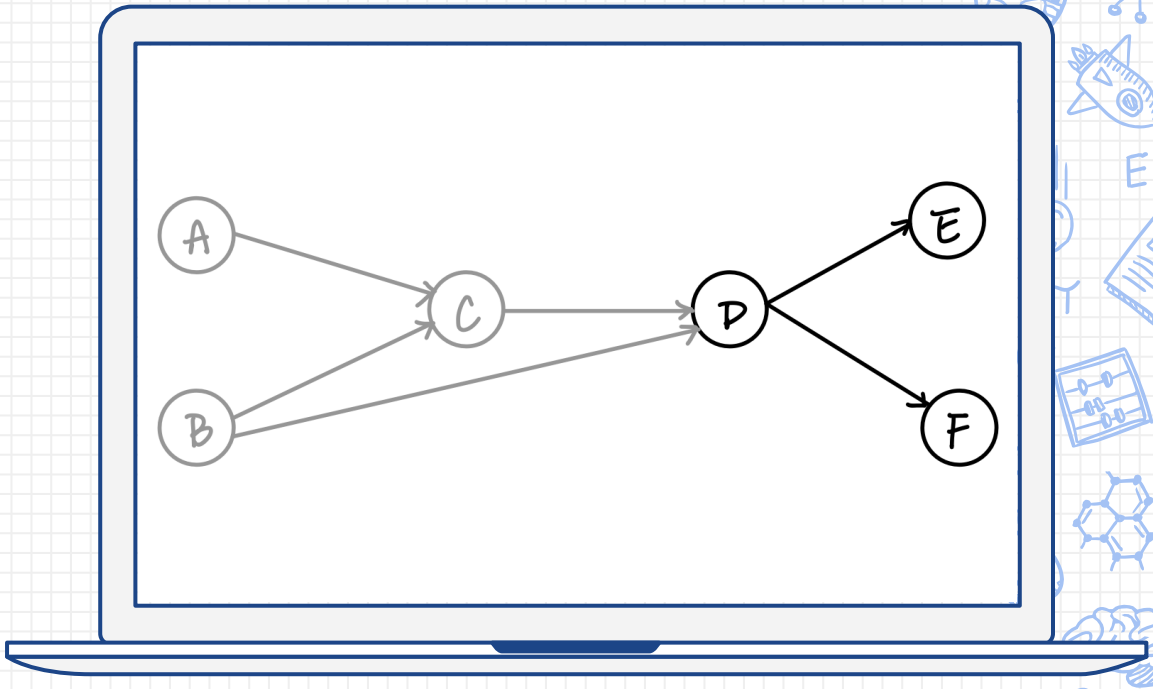


Topological sort: remove and update

Vertices	In-degree
A	0
B	0
C	0
D	0
E	1
F	1

Taking:

A → B → C

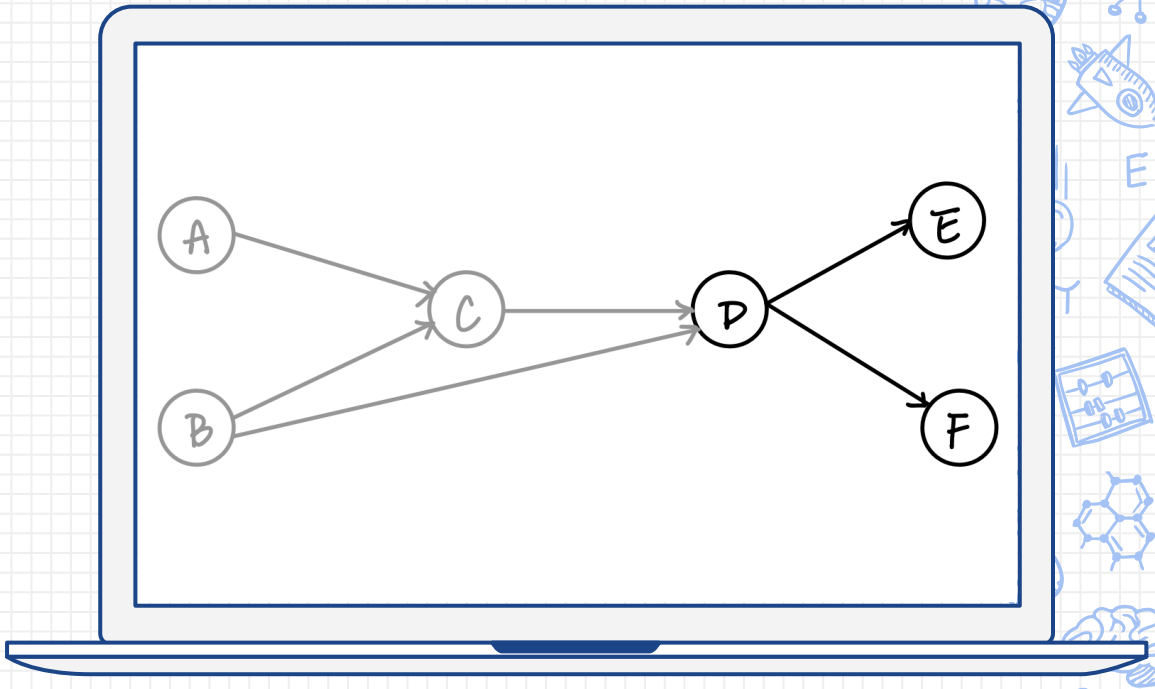


Topological sort: take the vertex with 0 in-degree

Vertices	In-degree
A	0
B	0
C	0
D	0
E	1
F	1

Taking:

A → B → C → D

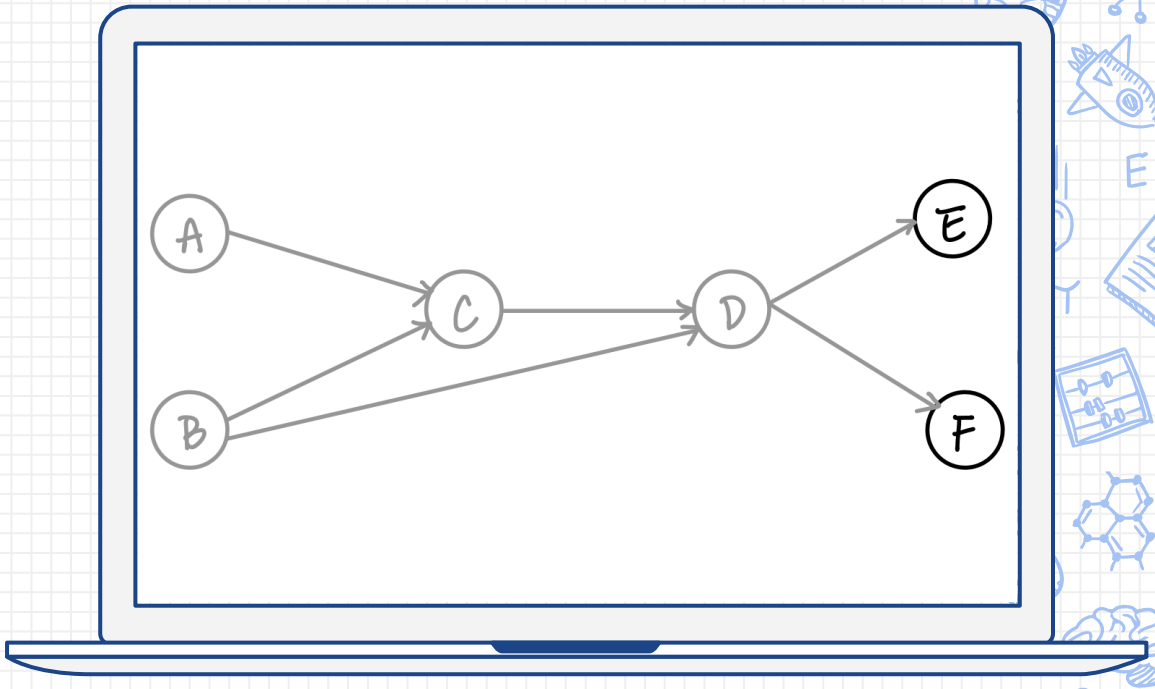


Topological sort: remove and update

Vertices	In-degree
A	0
B	0
C	0
D	0
E	0
F	0

Taking:

A → B → C → D

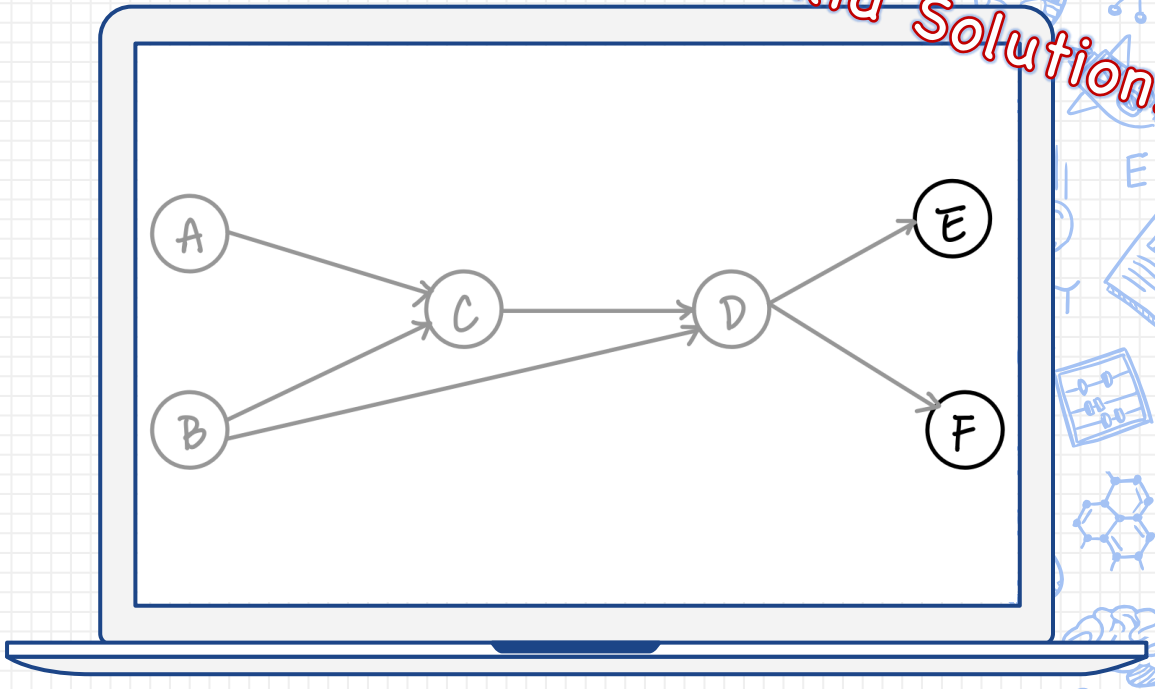


Topological sort: take the vertices with 0 in-degree

Vertices	In-degree
A	0
B	0
C	0
D	0
E	0
F	0

Taking:

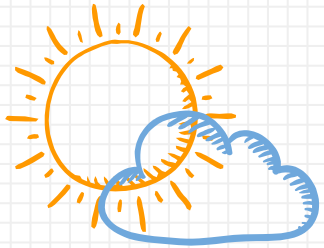
A → B → C → D → E → F



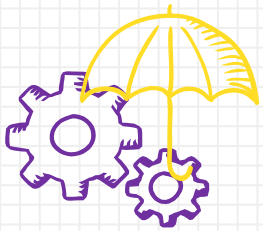
Topological Sort Using BFS

NO CYCLE IN GRAPH

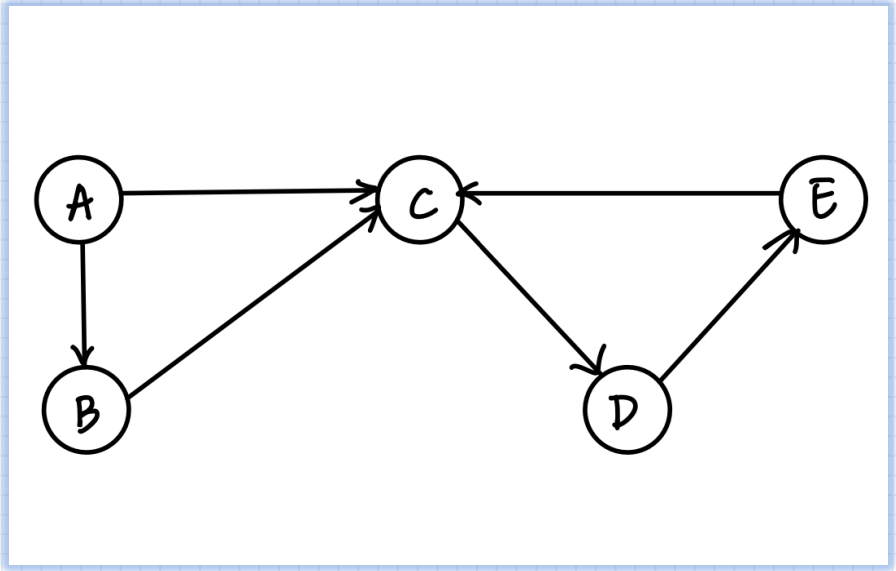
Generate a valid sequence for course taking.



What if there is a CYCLE in graph?



BFS when there is a cycle



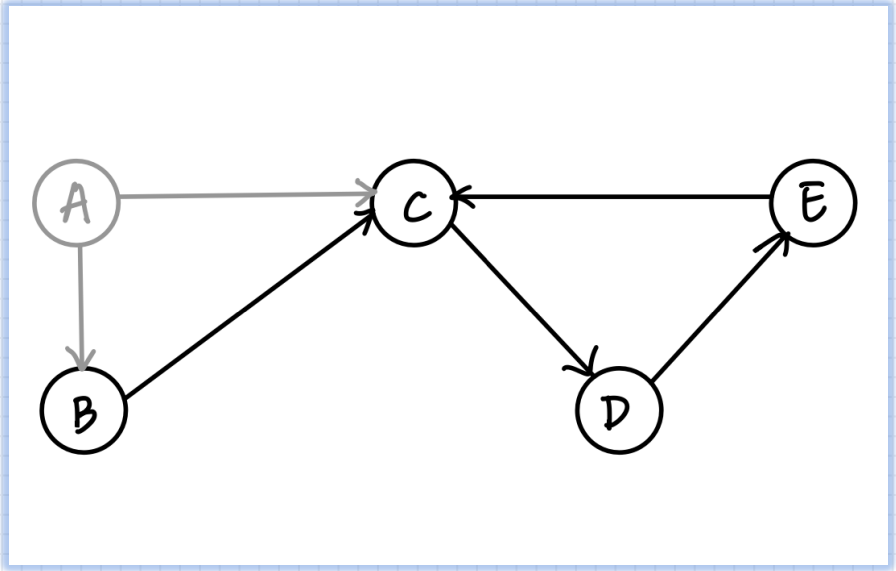
Vertices	In-degree
A	0
B	1
C	3
D	1
E	1

Taking:

A

No solution here

BFS: remove and update



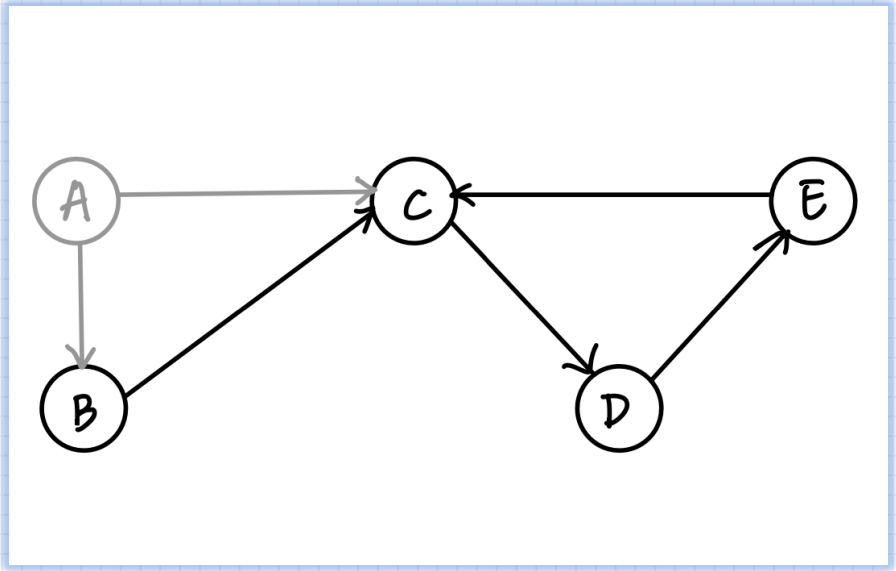
Vertices	In-degree
A	0
B	0
C	2
D	1
E	1

Taking:

A

No solution here

BFS: take the vertex with 0 in-degree



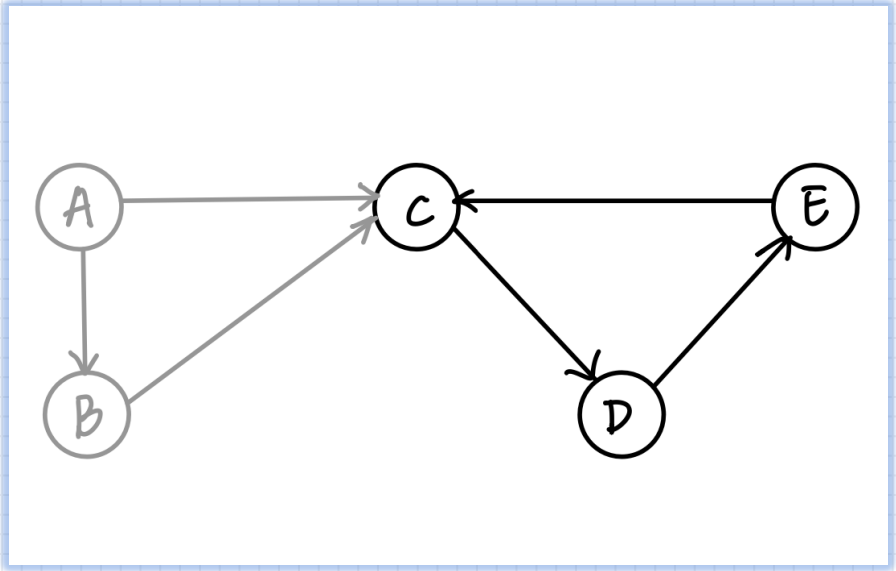
Vertices	In-degree
A	0
B	0
C	2
D	1
E	1

Taking:

A → B

No solution here

BFS: remove and update



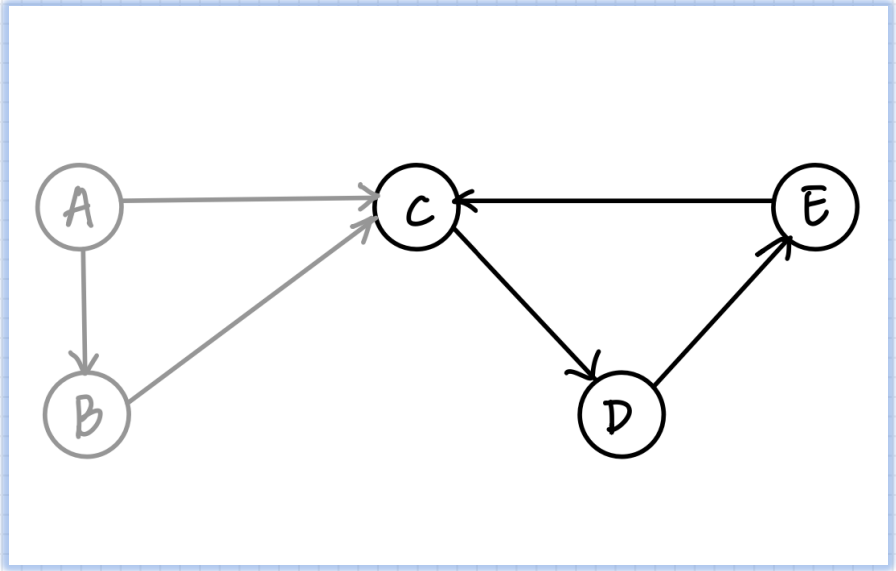
Vertices	In-degree
A	0
B	0
C	1
D	1
E	1

Taking:

A → B

No solution here

BFS: no more in-degree becomes 0



No solution here

Vertices	In-degree
A	0
B	0
C	1
D	1
E	1

Taking:

A → B

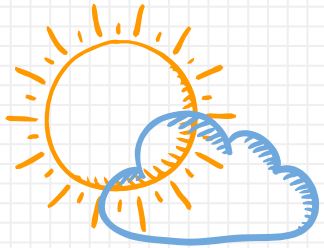
Cannot Proceed!

Topological Sort Using BFS

NO CYCLE IN GRAPH

Generate a valid sequence for course taking.

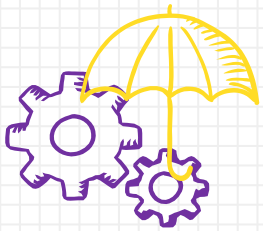
We Know as the length of sequence equals the number of courses.

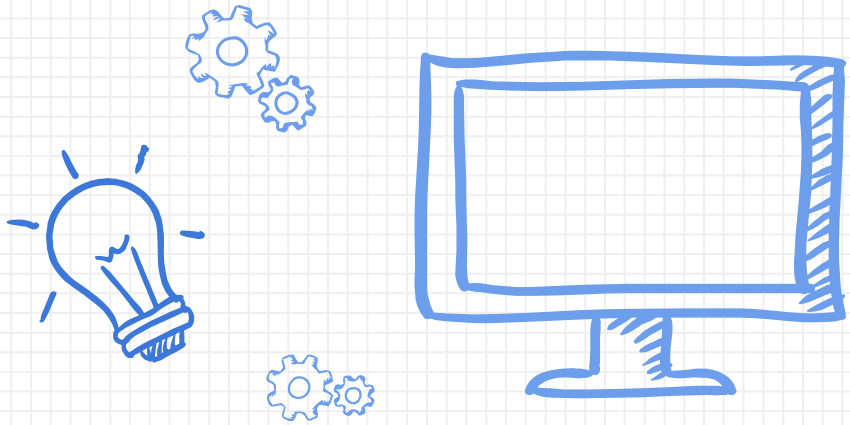


CYCLE IN GRAPH

Generate a invalid sequence for course taking.

We Know as the length of sequence is smaller than the number of courses.





Hello, computer scientists! Let's write

CODES!

Build a Graph -> Find a Path by Depth-First Search(DFS)

Build a graph as an adjacency list
V: numCourses
E: prerequisites

Start from v,
Mark v as visited

Recursively visit all unvisited vertices u pointing from v

Build a graph as an adjacency list

Global variable edges

prerequisites -- (course, prerequisite) pairs

Initialize edges

Add edges

```
List<List<Integer>> edges;

public int[] dfs(int numCourses,
                 int[][] prerequisites) {


    edges = new ArrayList<List<Integer>>();
    for (int i = 0; i < numCourses; ++i){
        edges.add(new ArrayList<Integer>());
    }


    for (int[] pair : prerequisites) {
        edges.get(pair[0]).add(pair[1]);
    }
```

Start DFS from an unvisited vertex

Initialize variables for DFS

Valid -- true if the graph has no cycle

Start DFS from unvisited vertices

If these is a cycle, return an empty list;

Else return result in reverse order

```
int[] visited = new int[numCourses];
List<Integer> result = new ArrayList<>();

boolean valid = true;

for (int v = 0; v < numCourses; ++v) {
    if (visited[v] == 0) {
        dfs(v, visited, result);
    }
}

if (!valid) {
    return new int[0];
}
return result.reverse();
```

Recursively visit all adjacent unvisited u

```
public void dfs(int v,
                int[] visited,
                int[] result) {
    visited[v] = 1;
    for (int u: edges.get(v)) {
        if (visited[u] == 0) {
            dfs(u, visited, result);
            if (!valid) { return; }
        } else if (visited[u] == 1) {
            valid = false;
            return;
        }
    }
    visited[v] = 2;
    result.add(v);
}
```

status:
0 unvisited
1 searching
2 visited

Mark v as searching

Visit all adjacent unvisited u

If there is a cycle, quit immediately.

Mark v as visited

Add v to result

Build a Graph -> Find a Path by

Breadth-First Search(BFS)

Build a graph as an adjacency list
V: numCourses
E: prerequisites

Count in-degrees of all vertices

Visit vertices with 0 in-degree

Build a graph as an adjacency list

Global variable edges

prerequisites -- (course, prerequisite) pairs

Initialize edges

Add edges

```
List<List<Integer>> edges;

public int[] dfs(int numCourses,
                 int[][] prerequisites) {


    edges = new ArrayList<List<Integer>>();
    for (int i = 0; i < numCourses; ++i){
        edges.add(new ArrayList<Integer>());
    }


    for (int[] pair : prerequisites) {
        edges.get(pair[0]).add(pair[1]);
    }
```

Count initial in-degrees & Add v to queue if in-degree == 0

Initialize variables for BFS

Count initial indegrees

Add vertices to the queue if indegree == 0

```
int[] indegree = new int[numCourses];
List<Integer> result = new ArrayList<>();

for (int[] info : prerequisites) {
    indegree[info[1]]++;
}


Queue<Integer> queue = new LinkedList<>();
for (int v = 0; v < numCourses; ++v) {
    if (indegree[v] == 0) {
        queue.offer(v);
    }
}
```

Visit vertices in the queue

Visit v in the queue

Add v to result

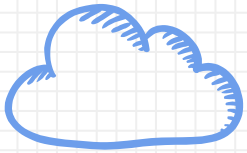
Update in-degrees of adjacent vertices

```
while (!queue.isEmpty()) {

    int v = queue.poll();

    result.add(v);

    for (int u: edges.get(v)) {
        indegree[u]--;
        if (indeg[v] == 0) {
            queue.offer(v);
        }
    }
}
```

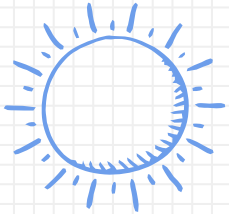
If not all vertices visited, there must be cycle

If not all vertices in the result, then there is a cycle

```
    if (result.size() != numCourses) {
        return new int[0];
    }

    return result;
}
```



lines of code of



lines of code of

DFS = BFS × 2

It's time to do some math...

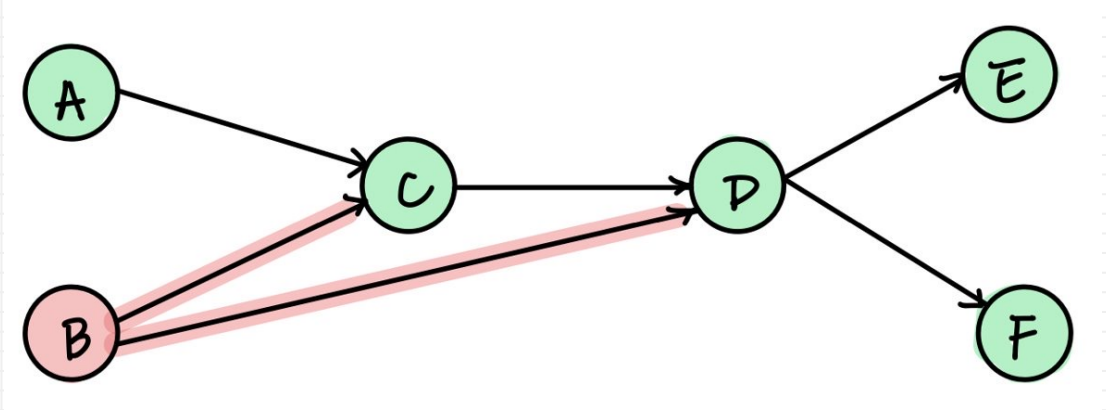
What is the

Time Complexity?

Time Complexity: O(V + E)

V: number of vertices

E: number of edges

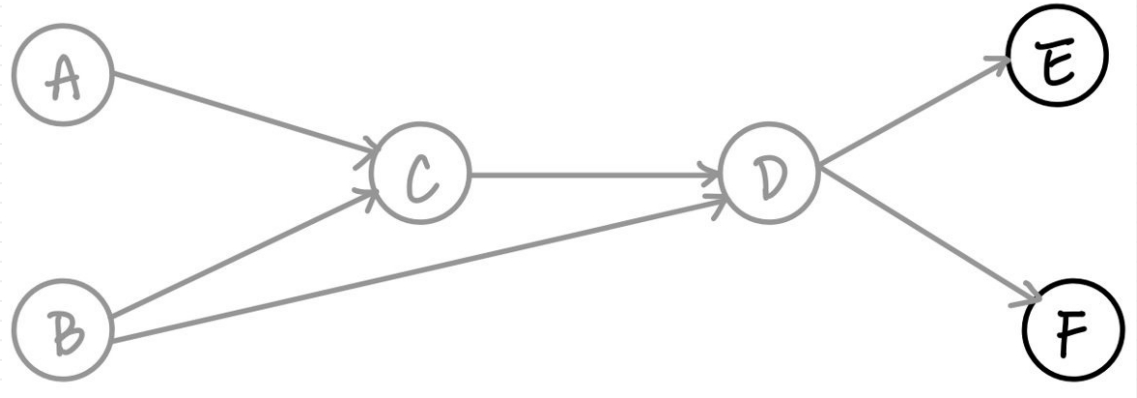


Visit each vertex and each edge **once and only once**.

Time Complexity: O(V + E)

V: number of vertices

E: number of edges



Visit each vertex and each edge once and only once.

The best programs

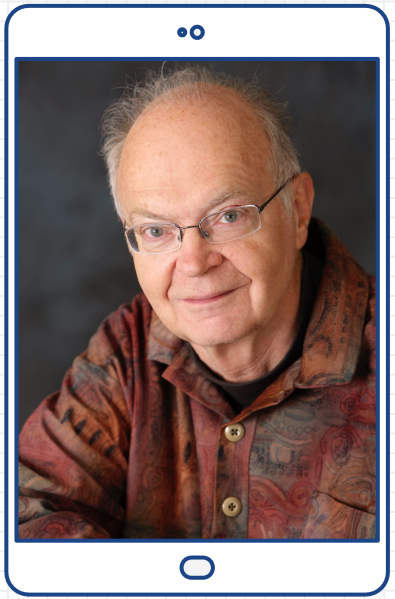
are written so that

computing machines can perform them quickly

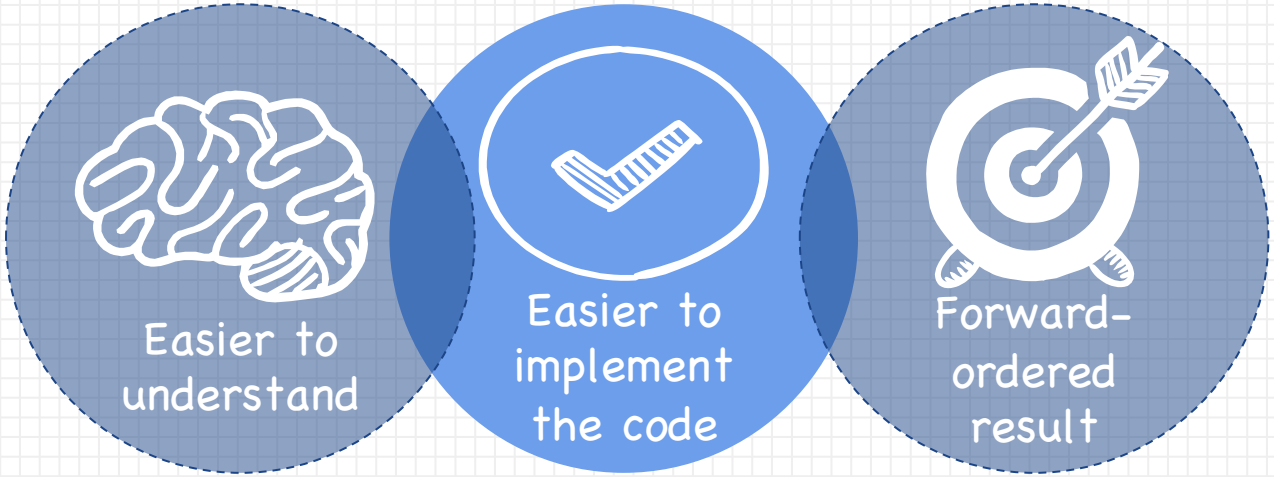
and so that

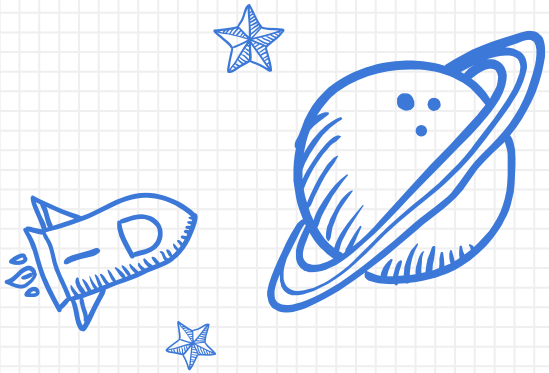
human beings can understand them clearly.

Donald Knuth



For Topological Sorting, BFS > DFS because:





APPLICATION!

Topological sort: Application

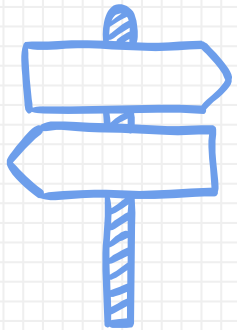
In computer science

- ✗ Operation system deadlock detection
- ✗ Serializing data
- ✗ Instruction Scheduling

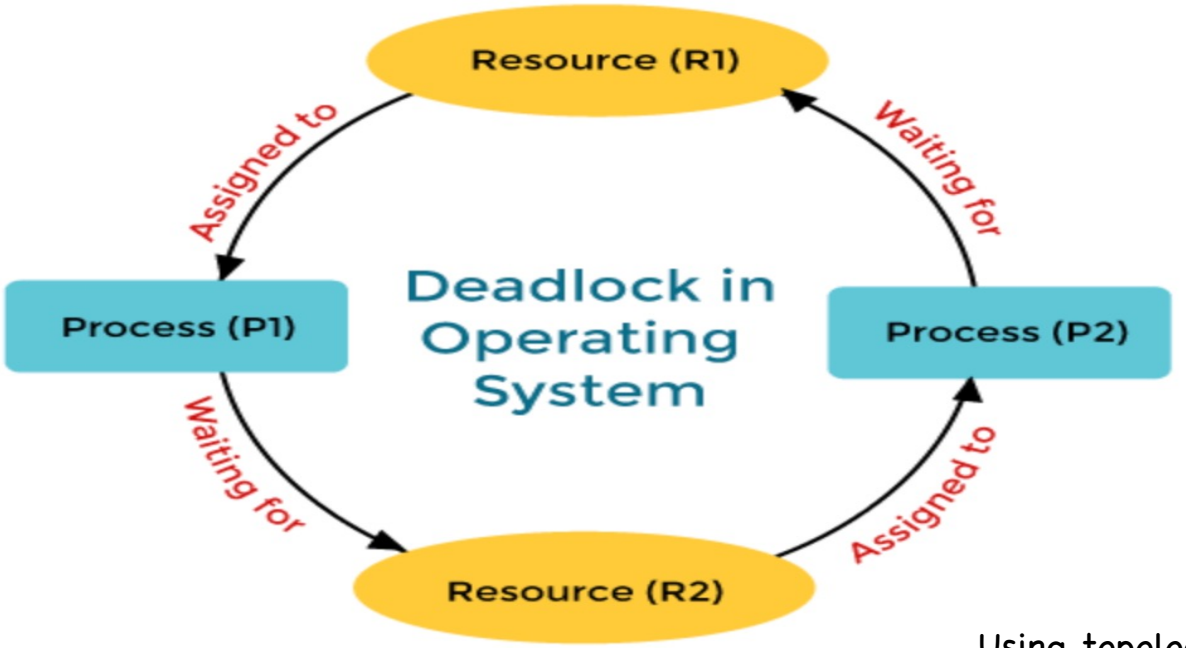
In real life

- ✗ Making pancakes
- ✗ Creating vaccine schedule
- ✗ Creating manufacturing workflow

In Computer Science



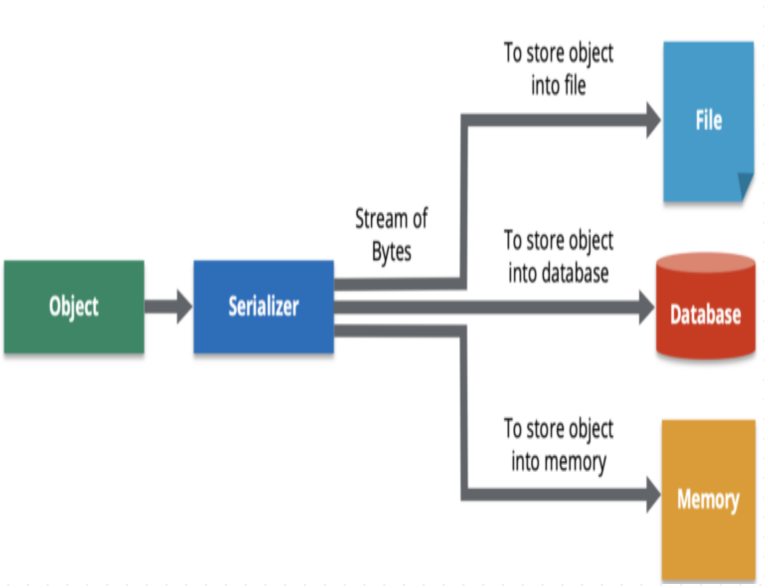
Operation system deadlock detection



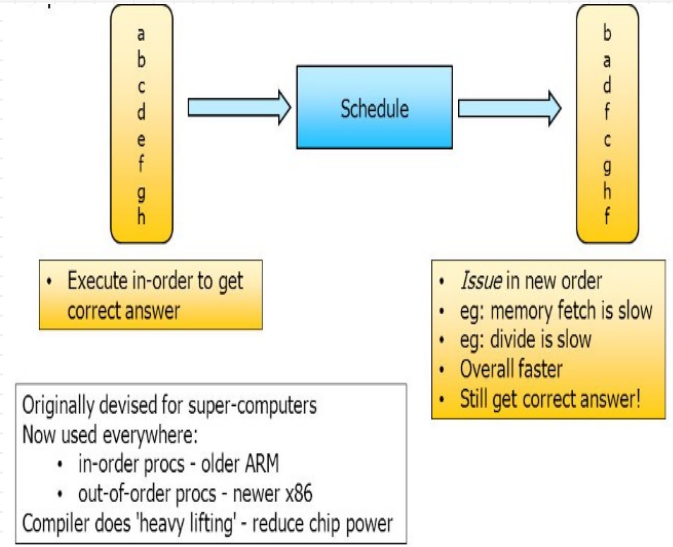
Using topological sort for cycle detection

Serializing data &Instruction Scheduling

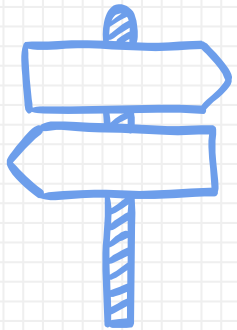
Serializing data



Instruction Scheduling

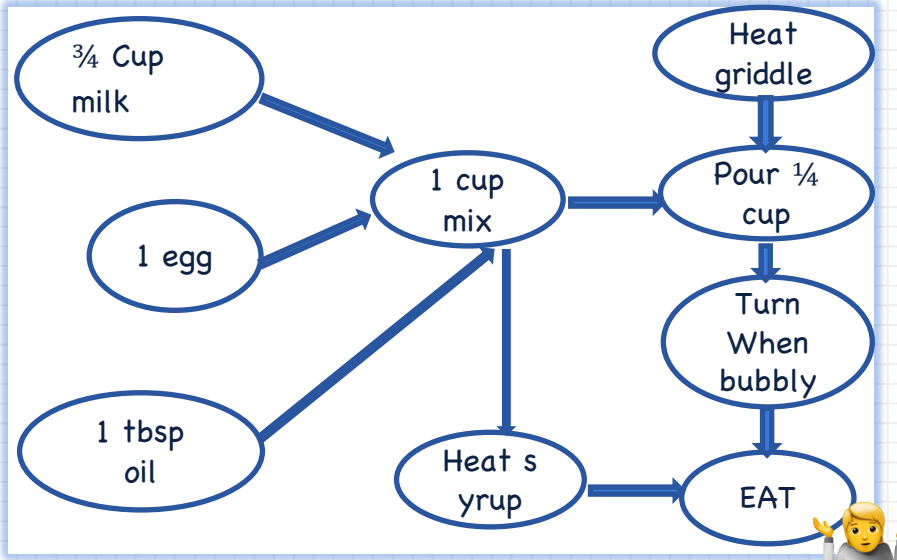


In real life

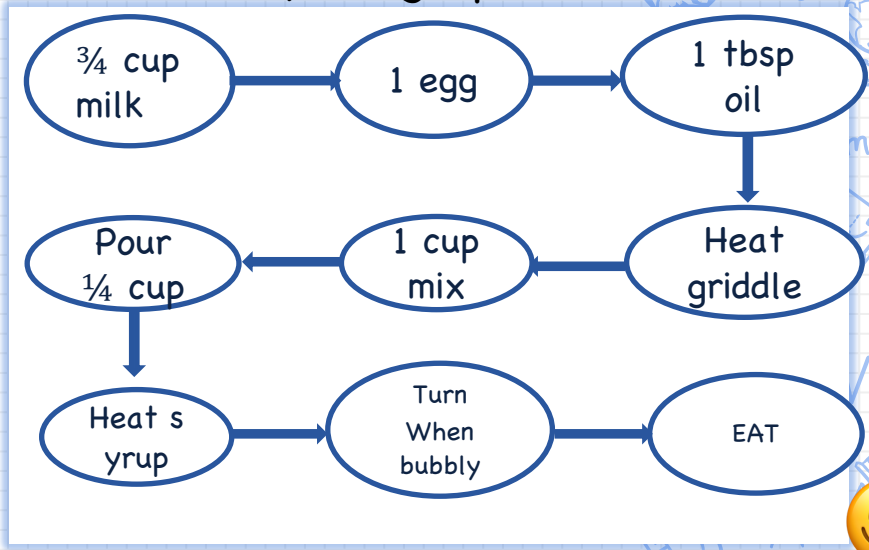


Making Pancakes

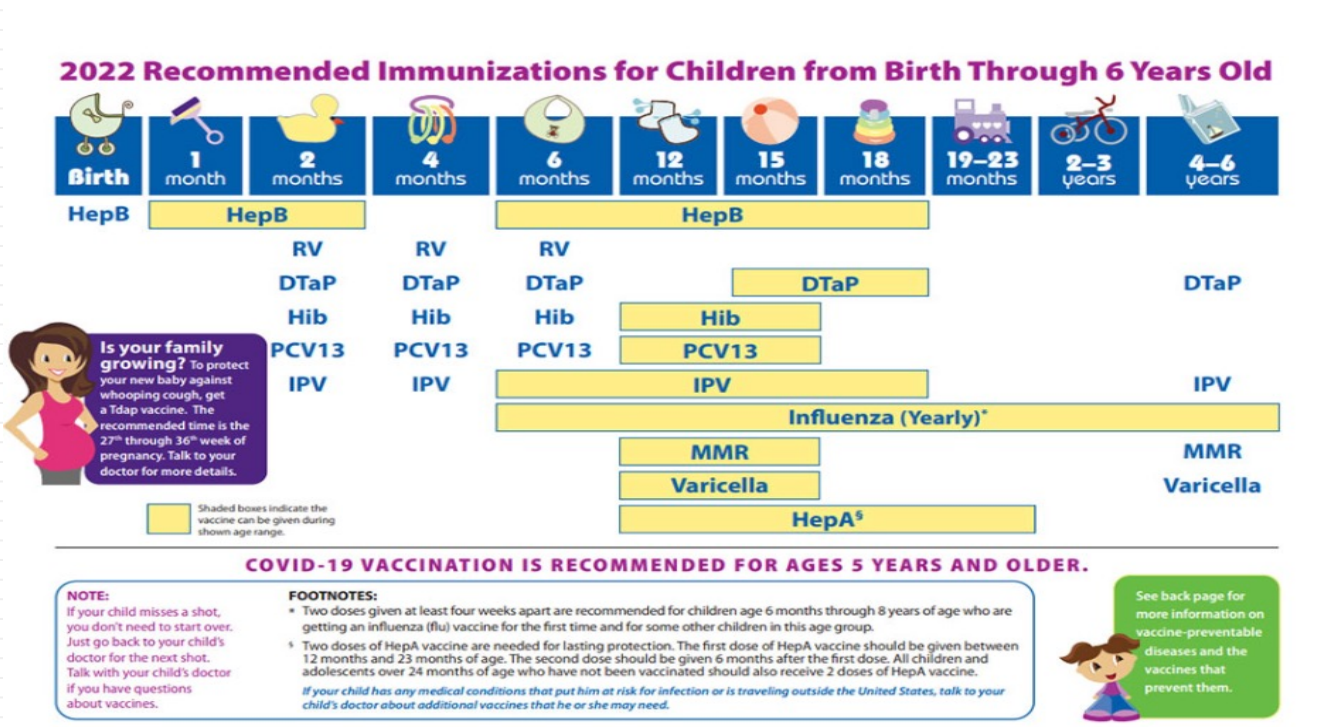
Steps for making pancakes



Result of topological sort on directed acyclic graph

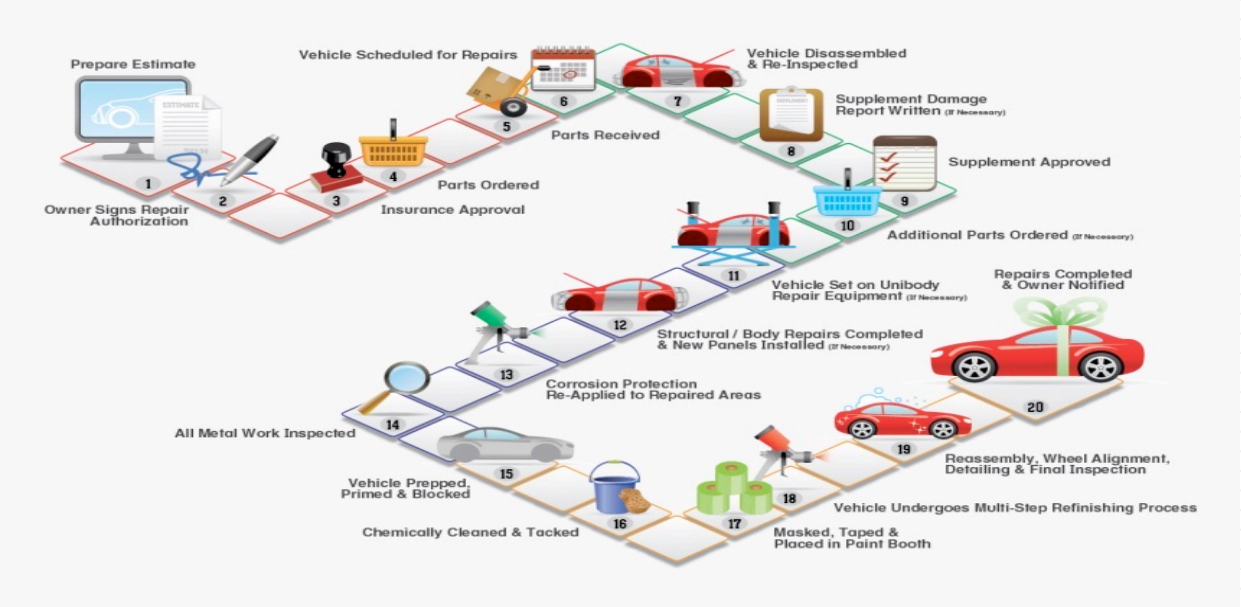


Creating vaccine schedule



https://www.cdc.gov/vaccines/parents/schedules/index.html

Manufacture workflow



Create workflow by using topological sorting: clear and more efficient!

https://www.pinterest.com/pin/602708362630983548/

THANKS! We're group #6

Chang Liu (Kayla)
002682838
Introduction
liu.chang31@northeastern.edu

Quan Yuan (Kate)
002703792
Visualize Solution
yuan.qua@northeastern.edu

Yidan Cong (Danielle)
002728816
Code
cong.yi@northeastern.edu

Lingjing Liu (Olivia)
002663069
Application
ling.lingj@northeastern.edu