

COL106: Data Structures and Algorithms

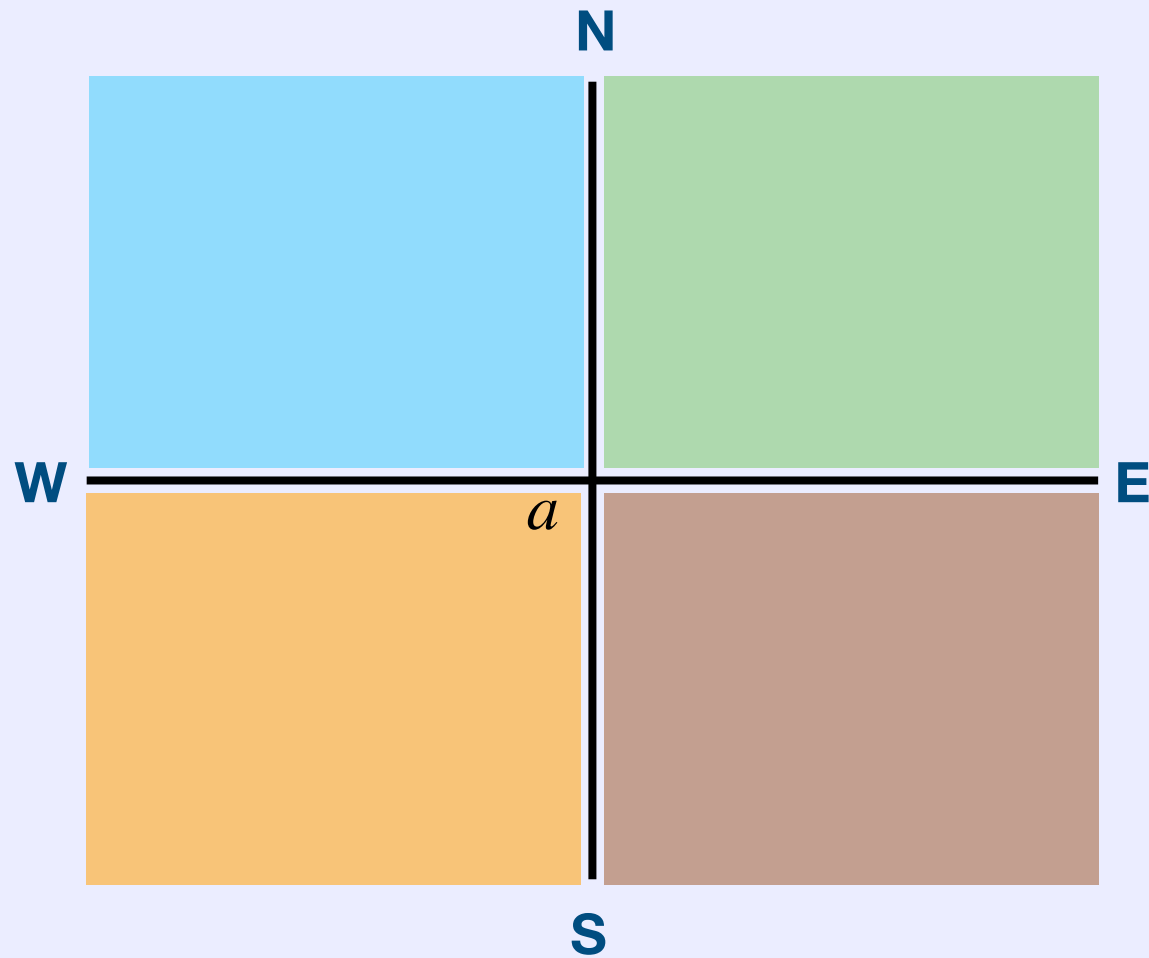
Lecture 32

Keerti Choudhary
Department of Computer Science, IIT Delhi

Today's Lecture

1. Hint for Assignment 5
2. Topological ordering

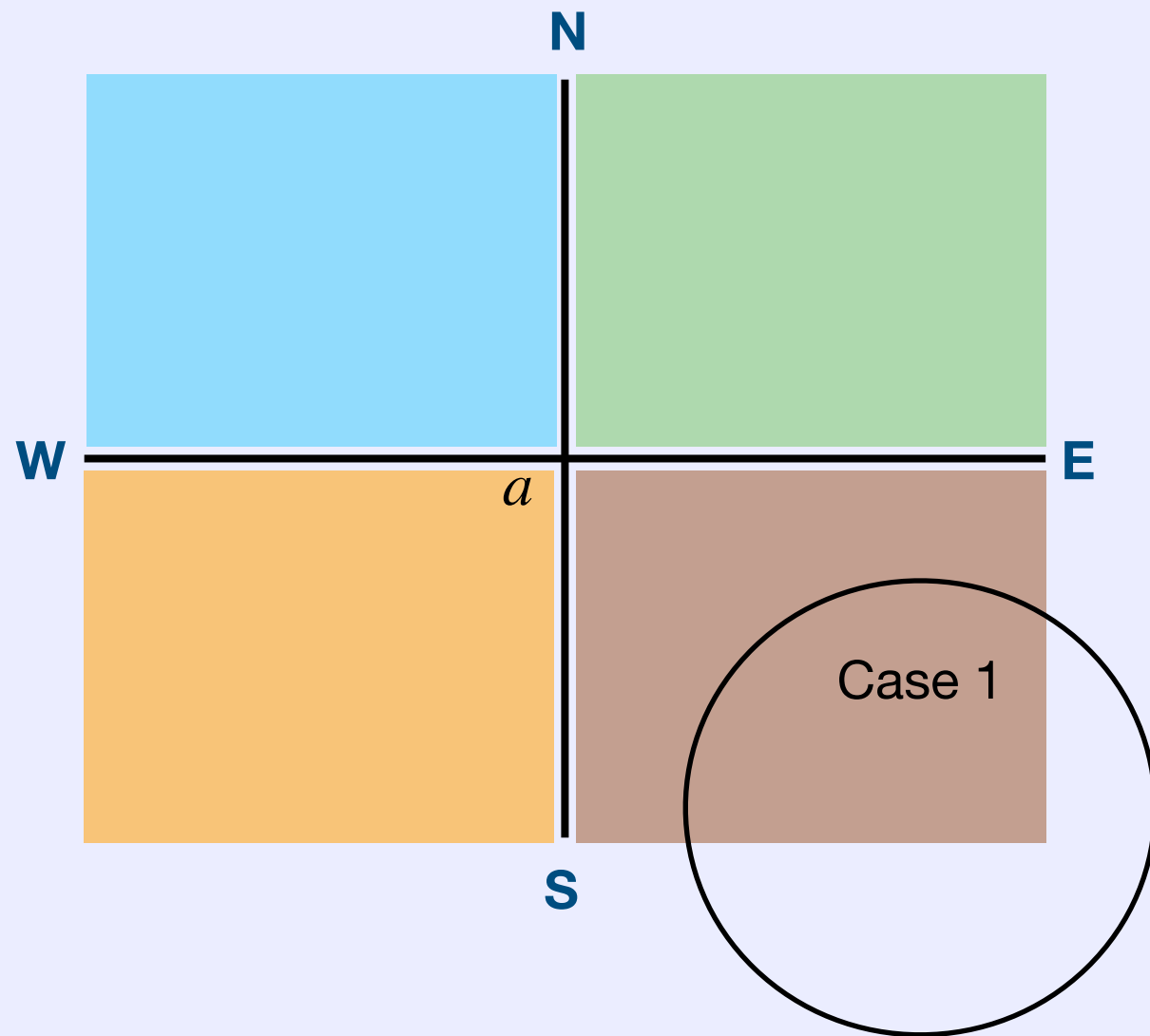
Hint for Assignment 5



Observation:

Each point a in the quad-tree splits the **entire cartesian-plane** into four *quadrants*.

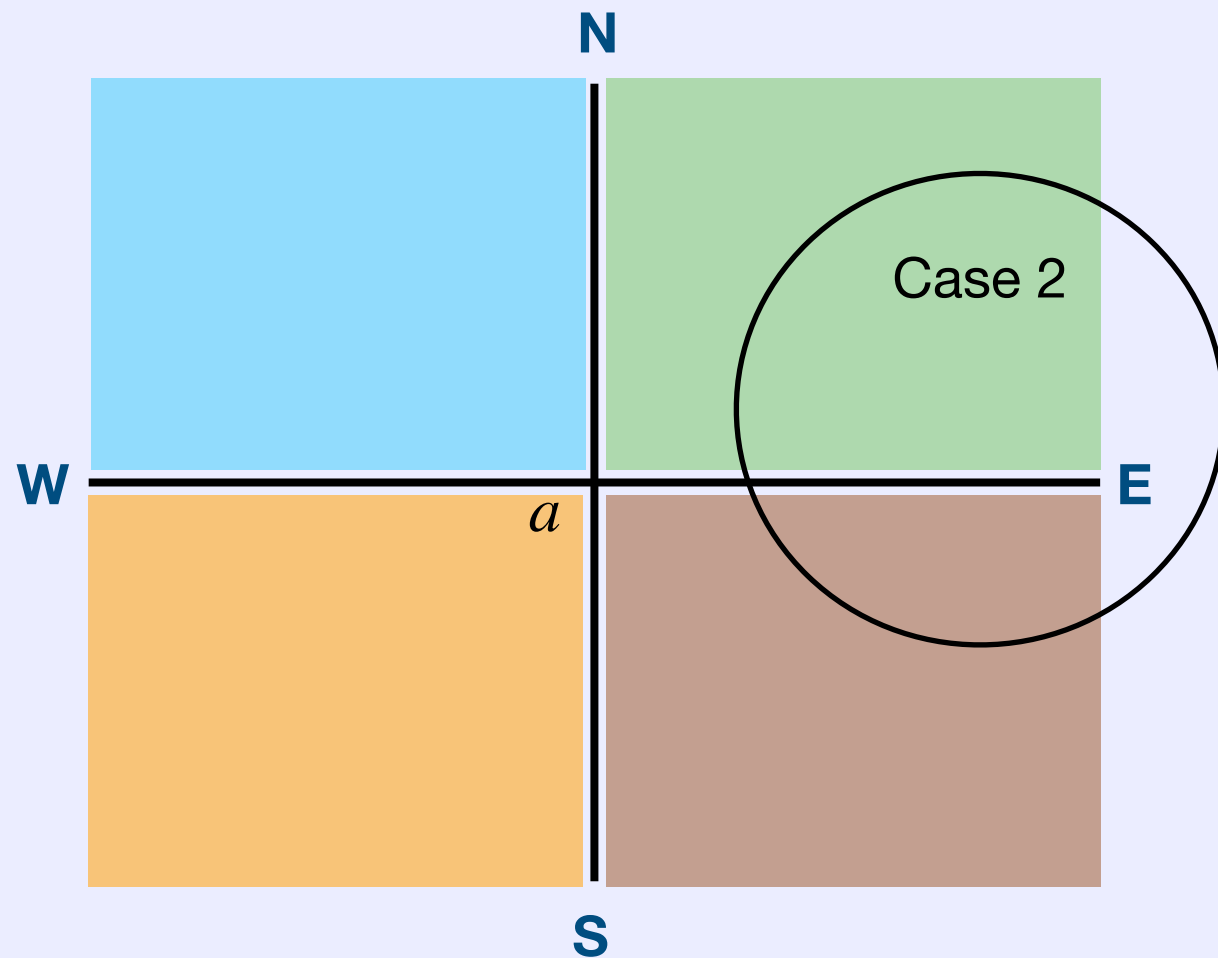
Hint for Assignment 5



Observation:

Each point a in the quad-tree splits the **entire cartesian-plane** into four *quadrants*.

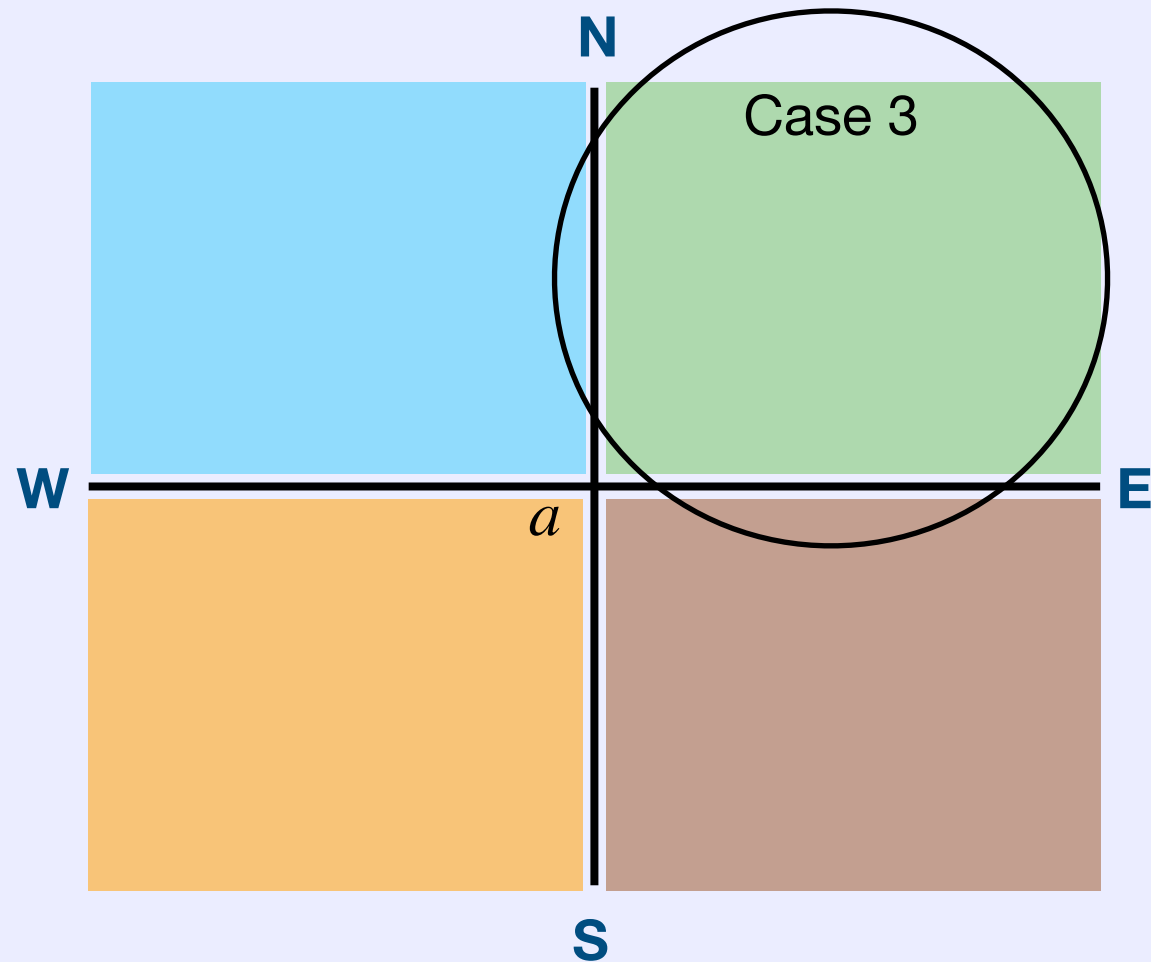
Hint for Assignment 5



Observation:

Each point a in the quad-tree splits the **entire cartesian-plane** into four *quadrants*.

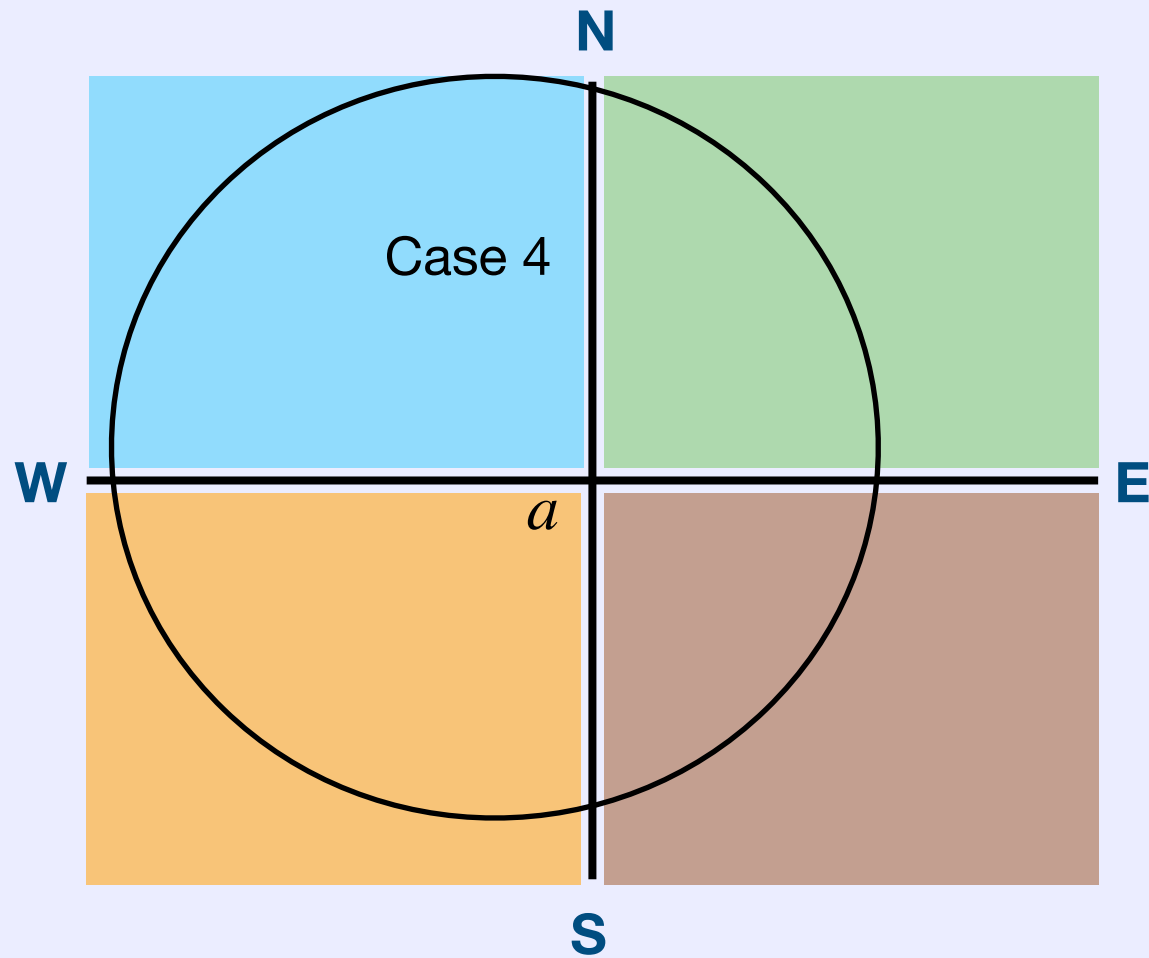
Hint for Assignment 5



Observation:

Each point a in the quad-tree splits the **entire cartesian-plane** into four *quadrants*.

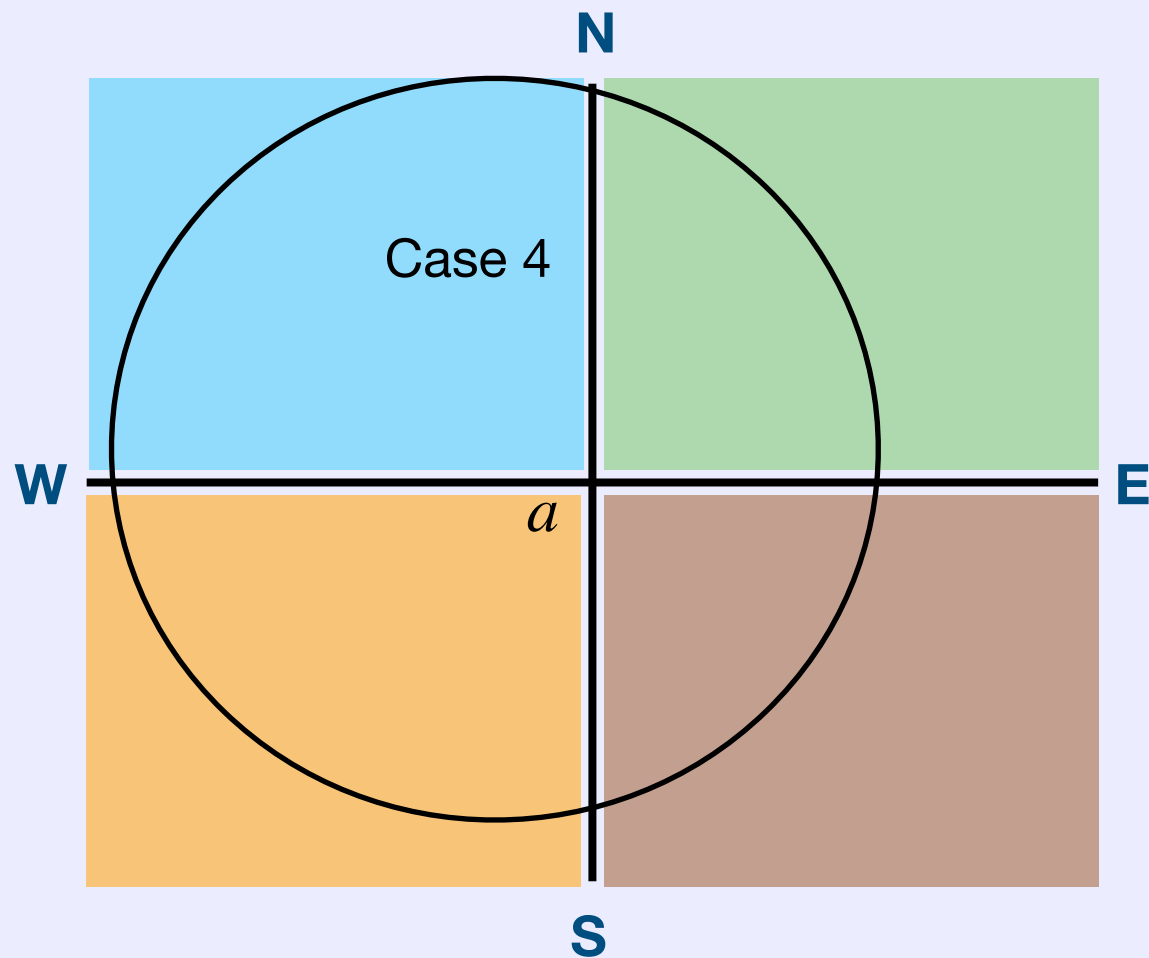
Hint for Assignment 5



Observation:

Each point a in the quad-tree splits the **entire cartesian-plane** into four *quadrants*.

Hint for Assignment 5



Observation:

Each point a in the quad-tree splits the **entire cartesian-plane** into four *quadrants*.

Hint:

You need to design a function to check which of the four *quadrants* intersects the circle.

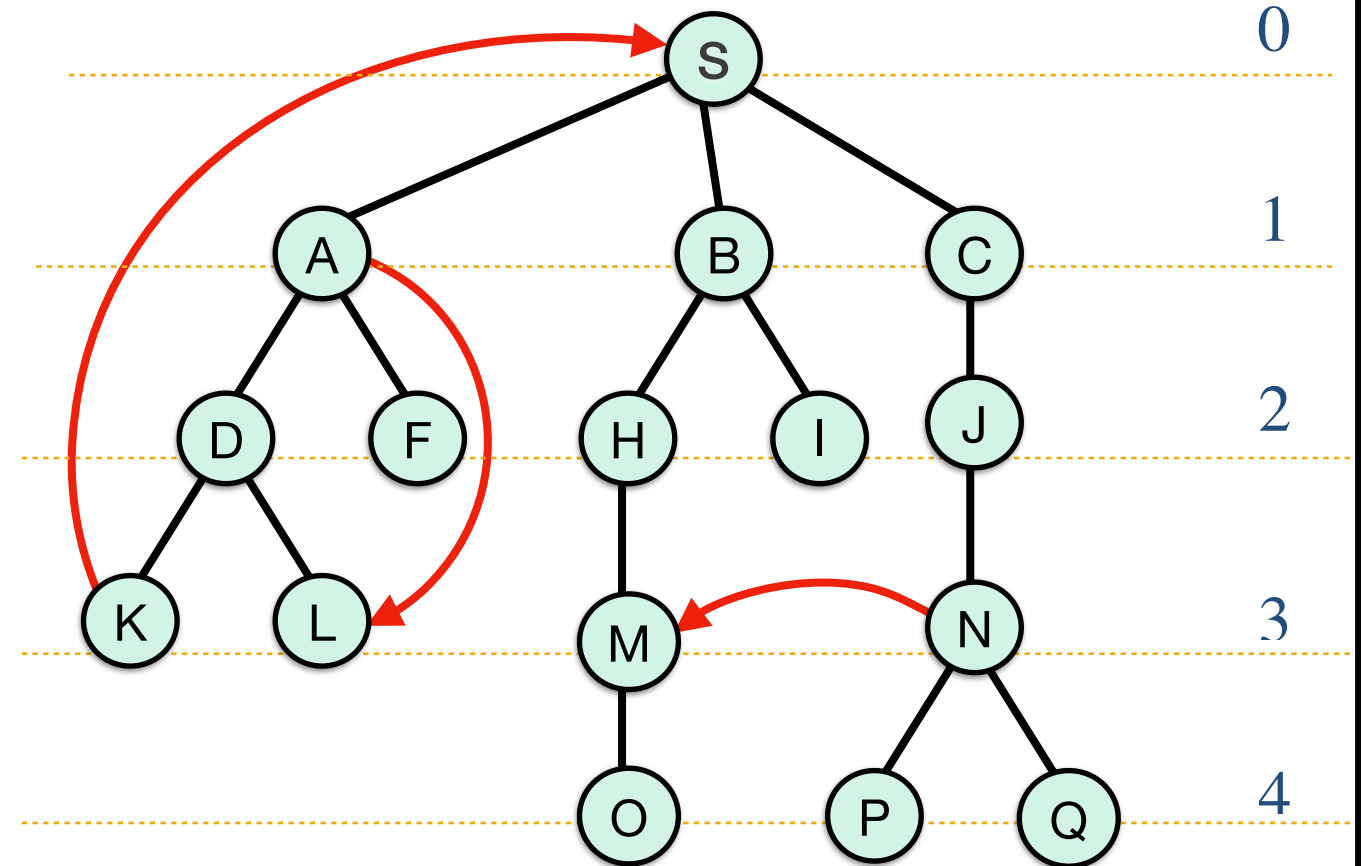
Only the quadrants intersecting the circle should be explored recursively.

DFS Tree in Directed graph

Characterisation of DFS tree

For a **directed** connected graph, a rooted spanning tree is said to be DFS tree if **for each** non-tree edge (xy) , either:

- x is ancestor of y , or
- x is descendant of y or
- (x, y) is a cross edge from right to left.



DFS tree
of a directed graph

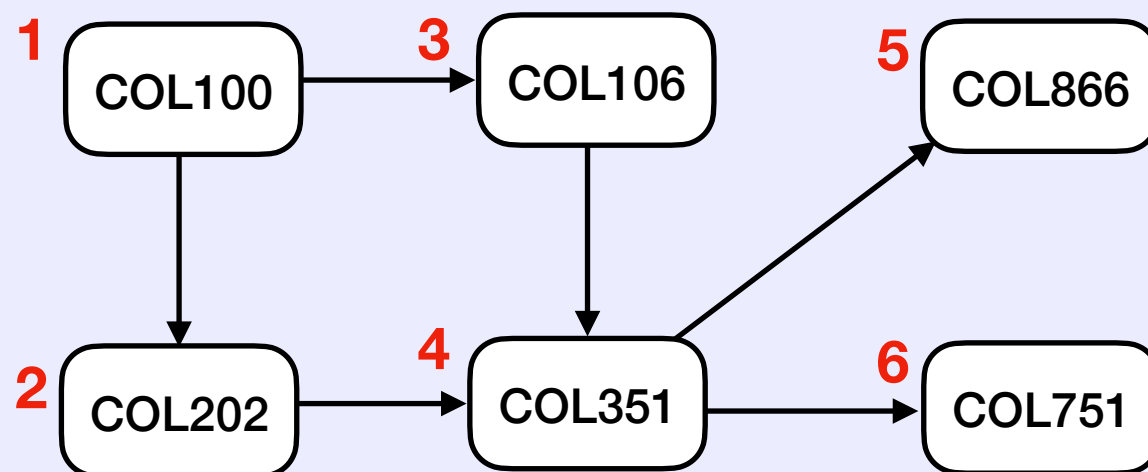
Course Ordering

Given: There is a collection C of n courses you want to do.

For each course $x \in C$, zero or more courses from set C can be **pre-requisites**.

Compute: An **ordering** in which courses must be done, assuming you are allowed to register only one course per semester.

Example:



Assumption:

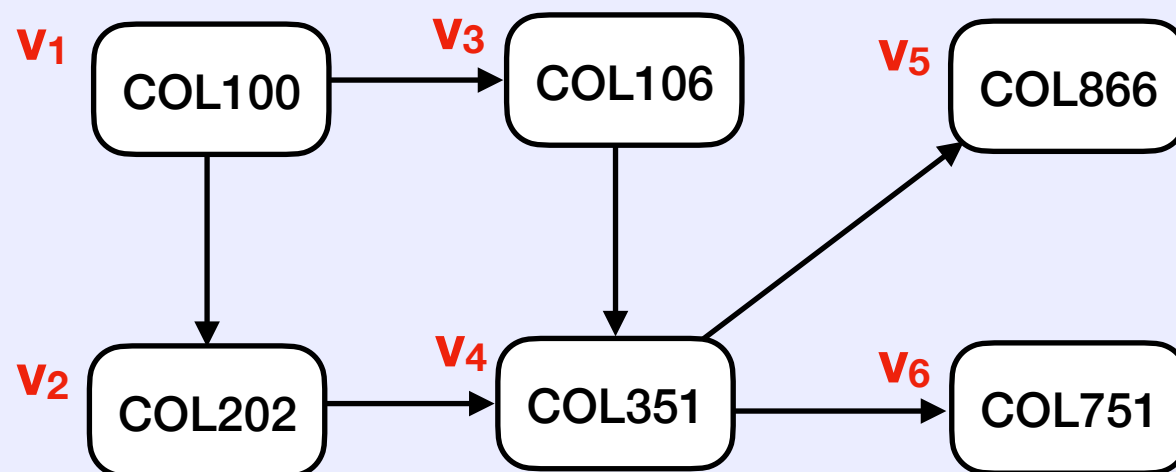
The course prerequisites do not form a Cycle.

Topological Ordering

Given: There is a **directed acyclic graph** (DAG) $G = (V, E)$ of n vertices.

Compute: An **ordering/permutation** of vertices, say $v_1, \dots, v_n$ such that for any edge $(v_i, v_j) \in E(G)$, we have $i < j$

Example:



*What should be
naive **algorithm**?*



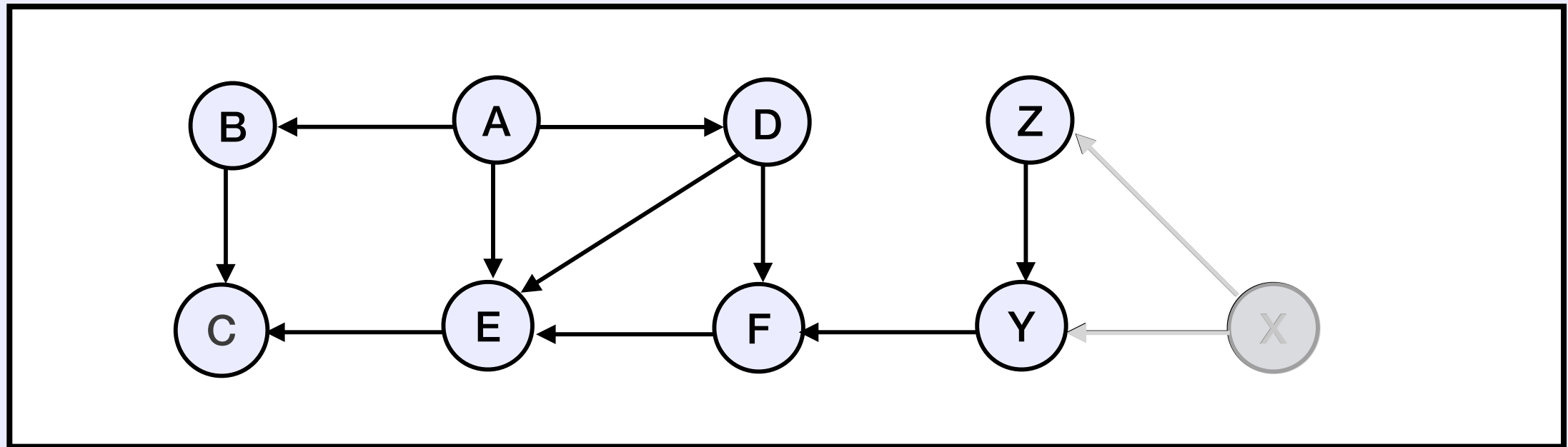
Property of DAGs

Lemma: Any **directed acyclic graph** (DAG) $G = (V, E)$ contains at least one vertex of in-degree zero.

Proof:

Homework (Prove using contradiction).

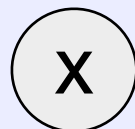
Naive Algorithm



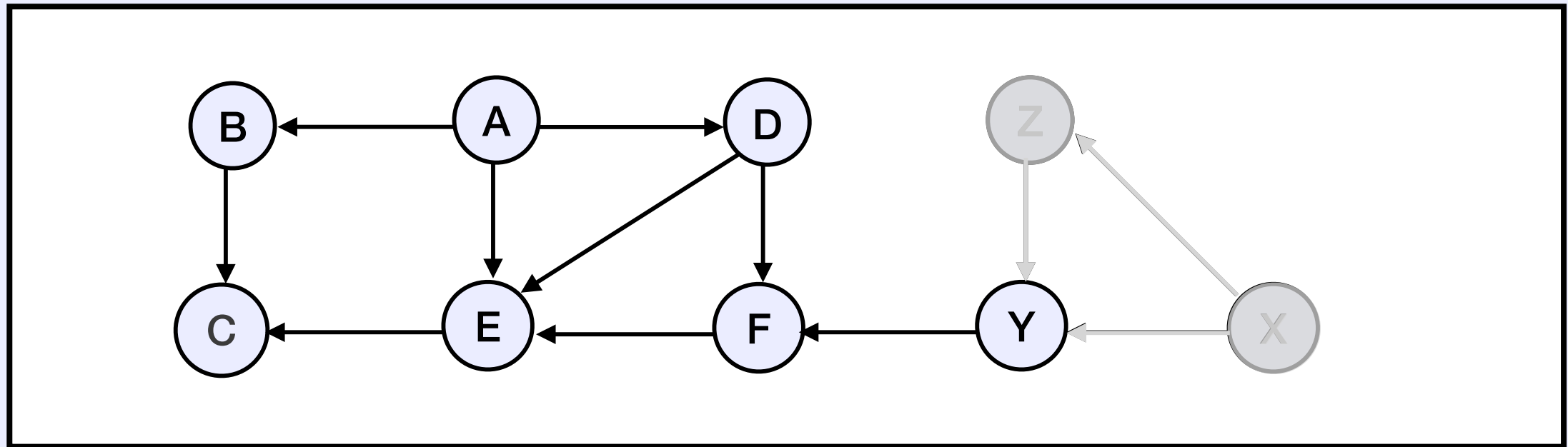
While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

Ordering:



Naive Algorithm



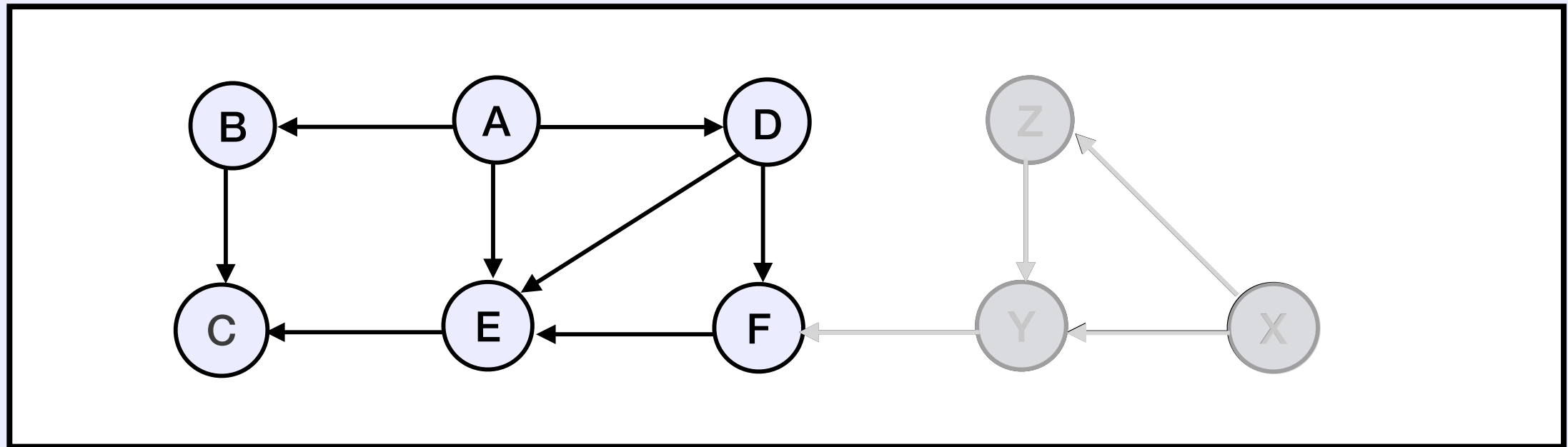
While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

Ordering:



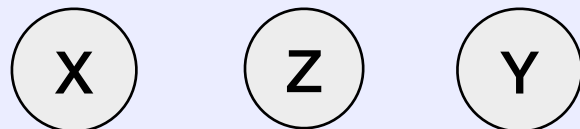
Naive Algorithm



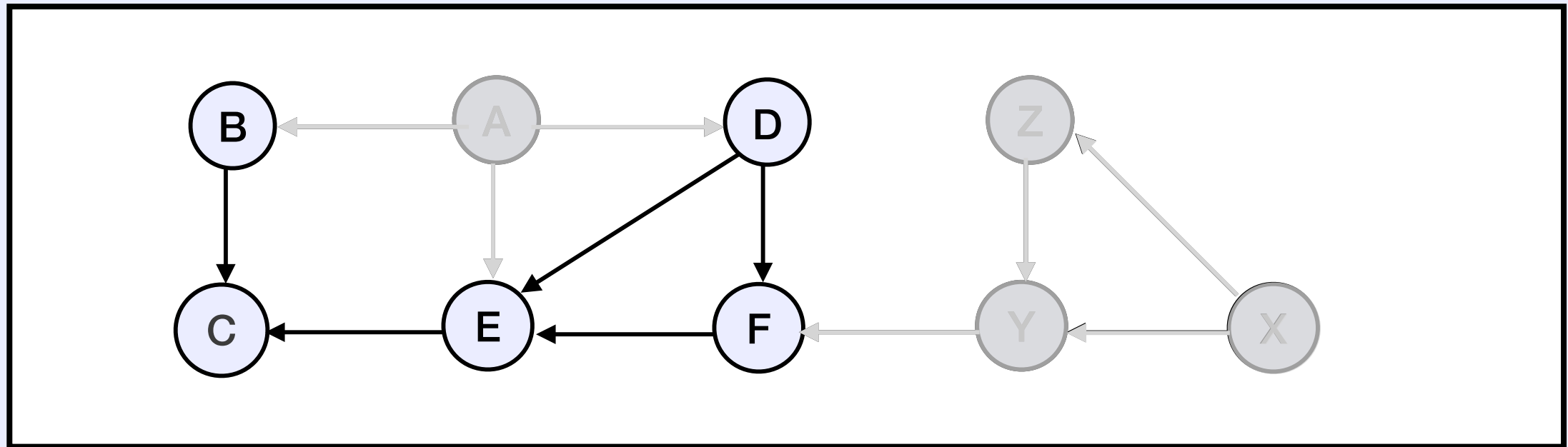
While graph is non-empty:

1. Pick a vertex *w* of zero in-degree, and append it to the output list.
2. Remove *w* and *out-edges(w)* from graph.

Ordering:



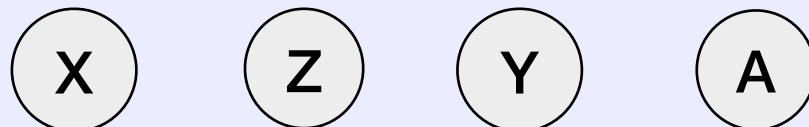
Naive Algorithm



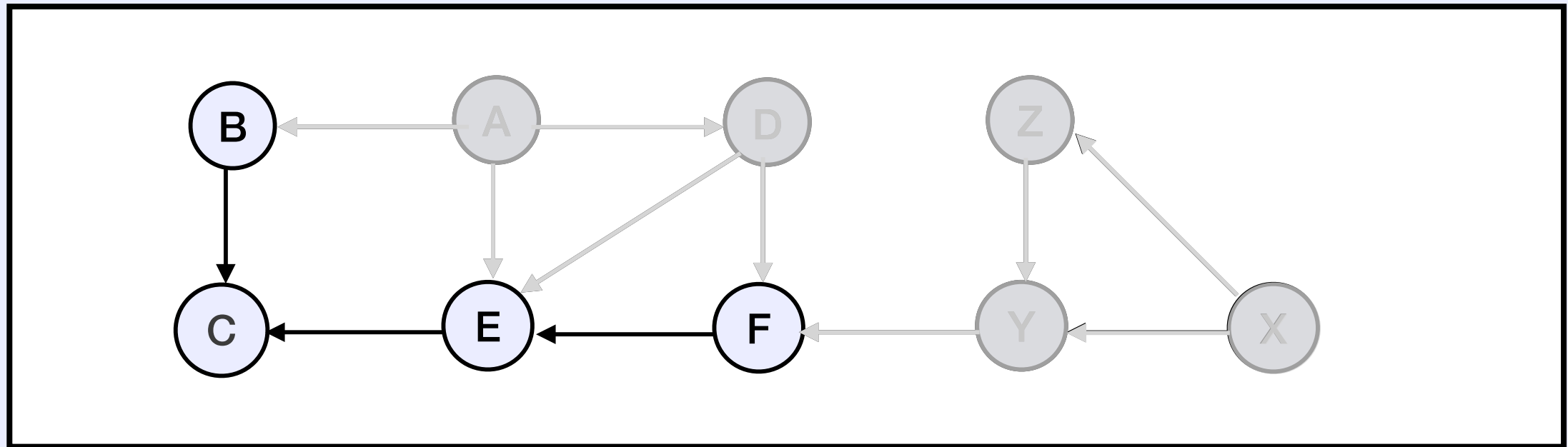
While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

Ordering:



Naive Algorithm



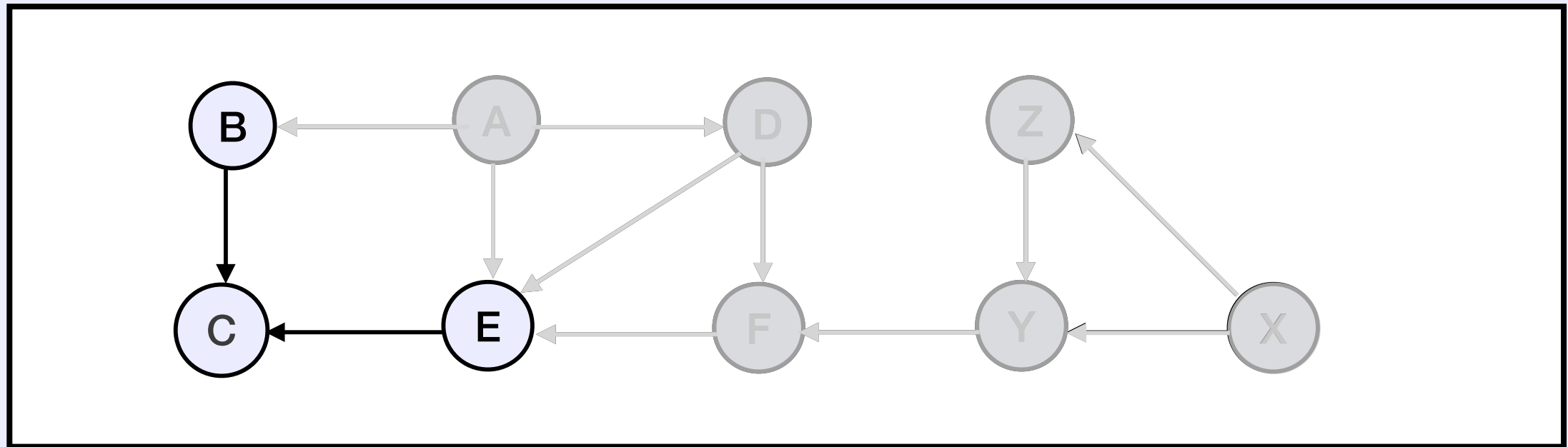
While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

Ordering:



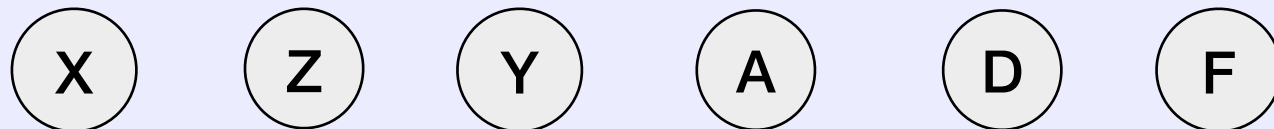
Naive Algorithm



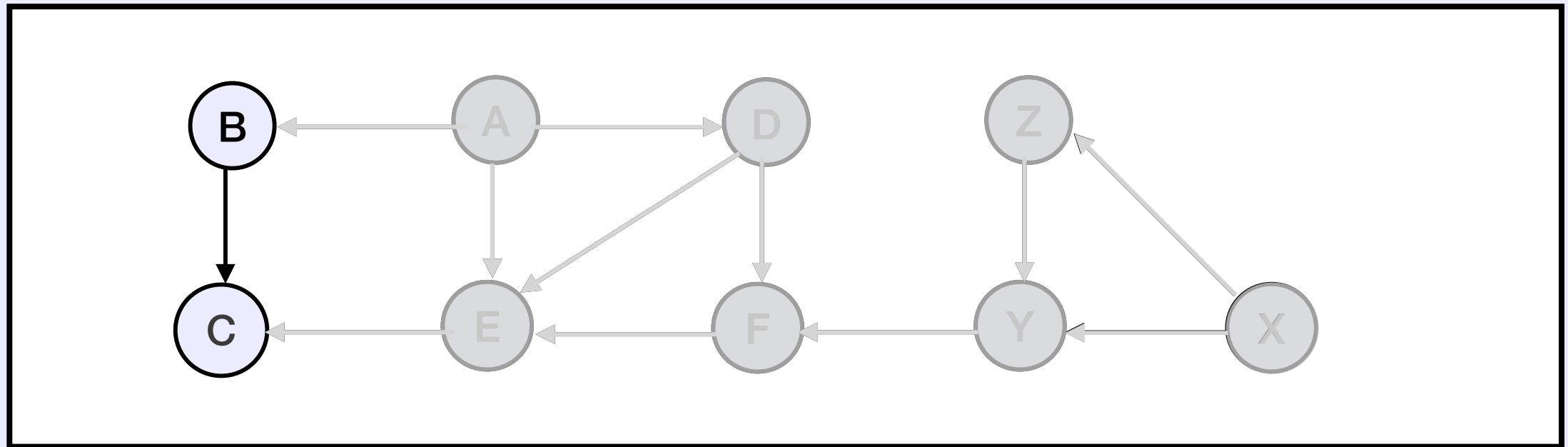
While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

Ordering:



Naive Algorithm

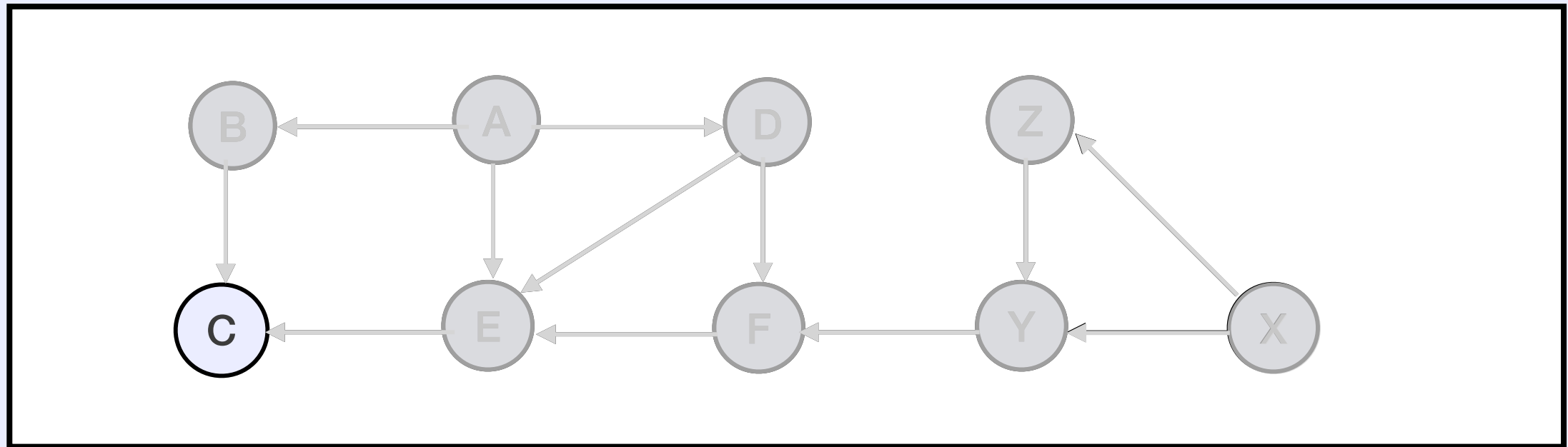


While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

Ordering: X Z Y A D F E

Naive Algorithm



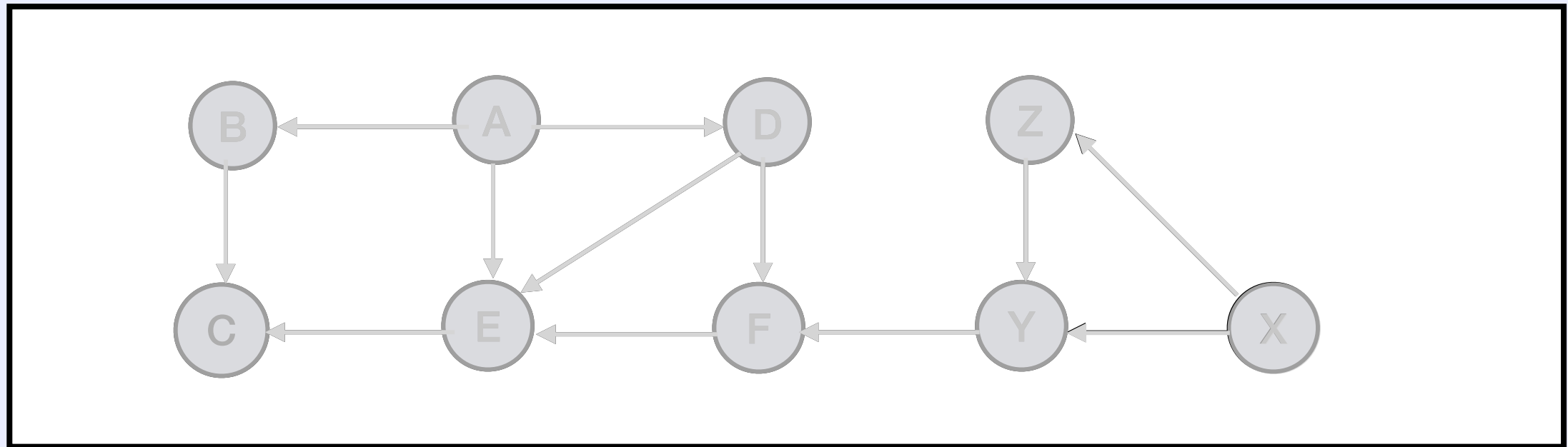
While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

Ordering:



Naive Algorithm



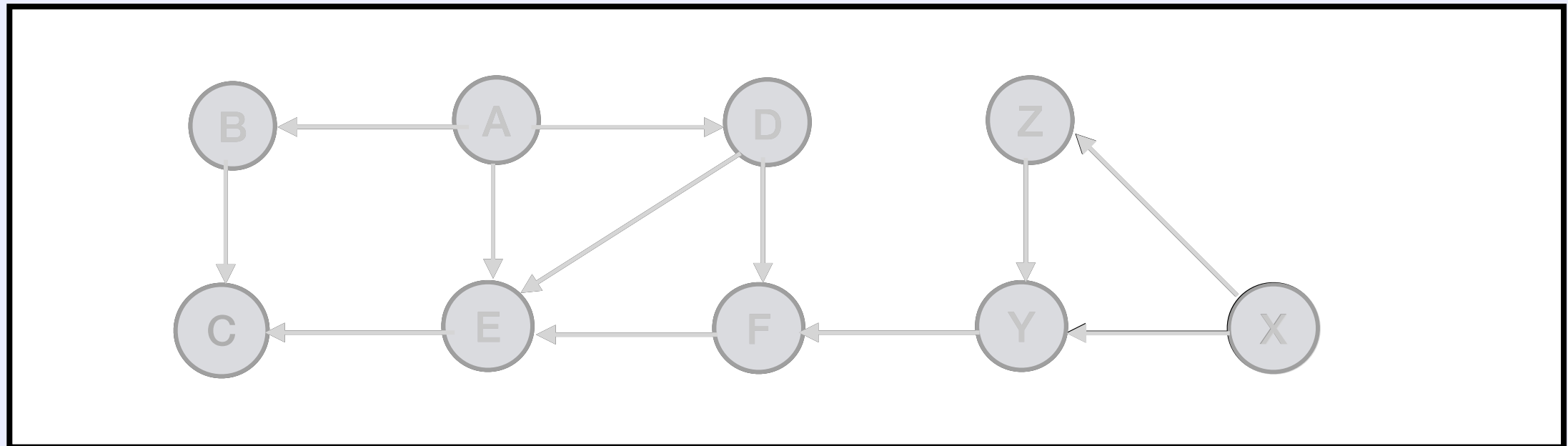
While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

Ordering:



Naive Algorithm



While graph is non-empty:

1. Pick a vertex w of zero in-degree, and append it to the output list.
2. Remove w and $\text{out-edges}(w)$ from graph.

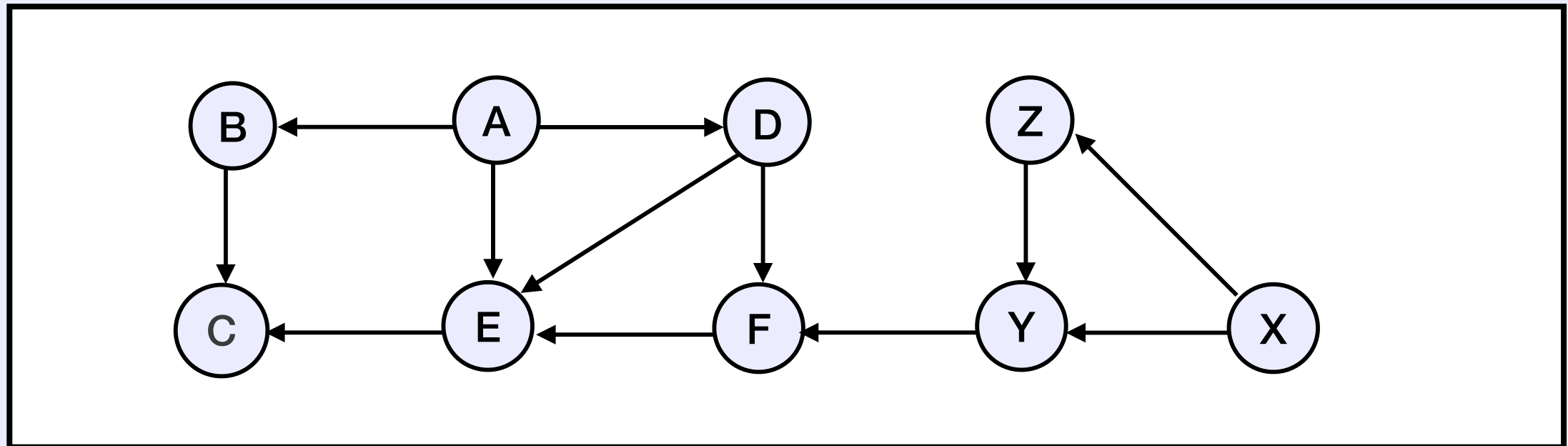
Time complexity:

$$O(m + n^2)$$

Ordering:



Improved Algorithm using DFS Trees



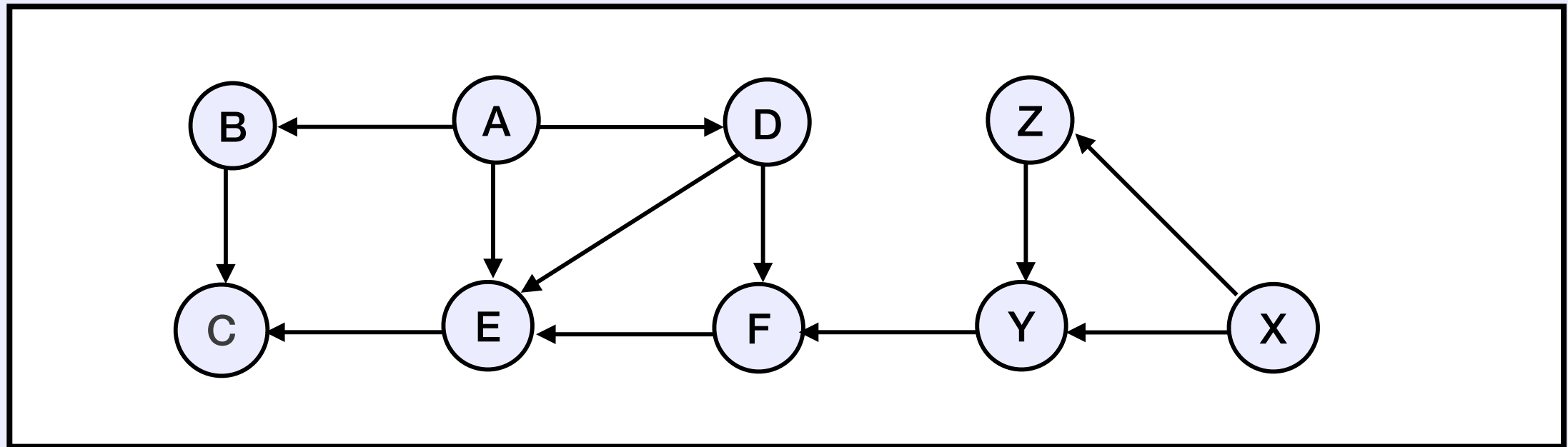
Algorithm:

1. For each $v \in V$:
 $visited(v) = false$
2. For each $v \in V$:
If ($visited(v) = false$)
 Invoke DFS(v)

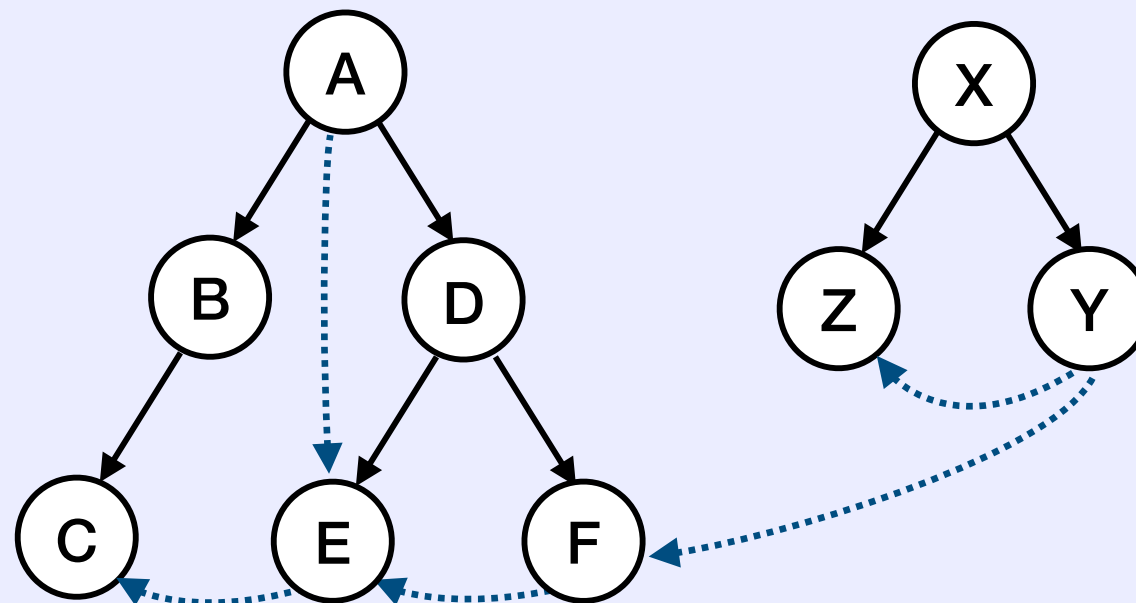
DFS(x):

1. Set $visited(x) = true$
2. For each $y \in N(x)$:
 If ($visited(y) = false$):
 Invoke DFS(y)

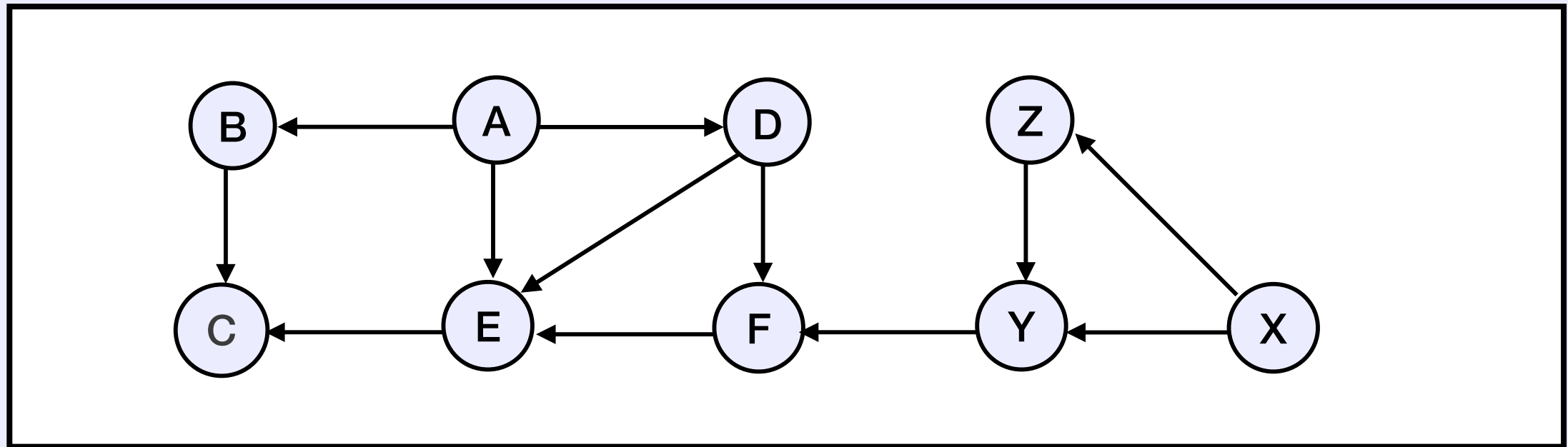
Improved Algorithm using DFS Trees



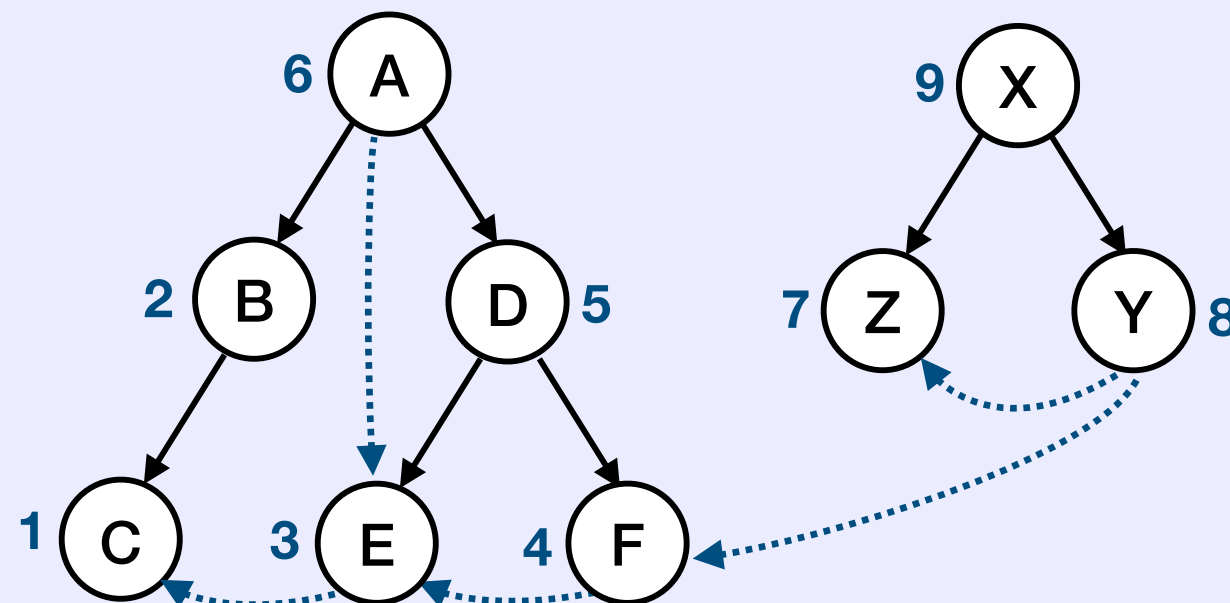
DFS Tree:



Improved Algorithm using DFS Trees



DFS Tree:



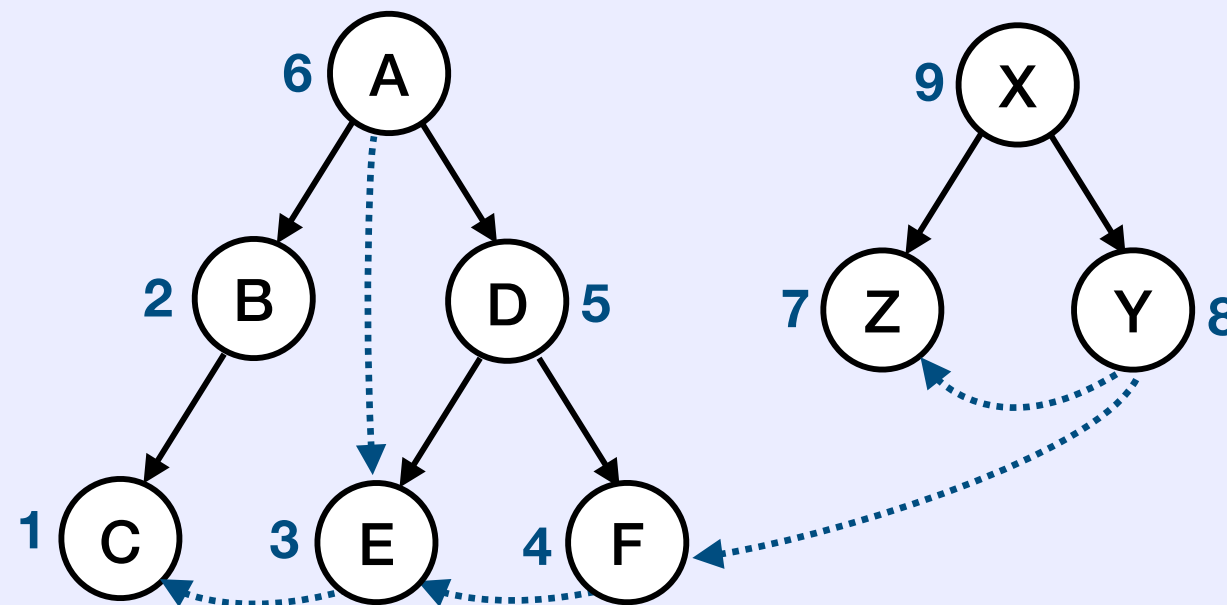
*Order in
which DFS
finishes
(or postorder
sequence):*

C B E F D A Z Y X

Property of Finish Time

Time complexity:

$$O(m + n)$$



Order in
which DFS
finishes
(or postorder
sequence):

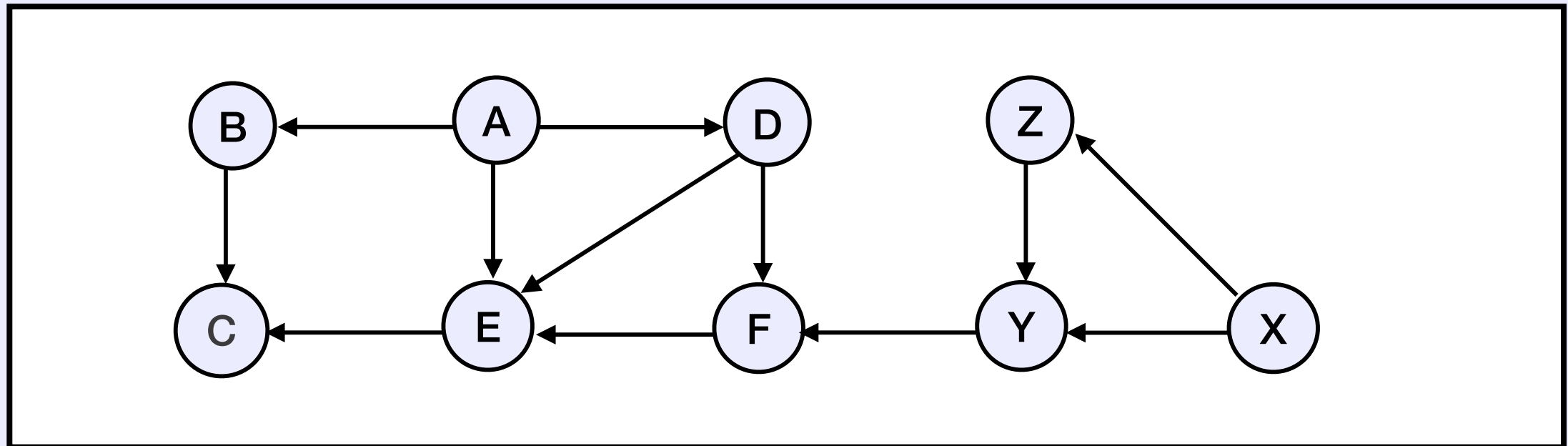
C B E F D A Z Y X

Lemma:

Let $G = (V, E)$ be a DAG, and for each $x \in V$, let $FNT(x)$ denote finish time of $DFS(x)$.

Then, for any edge (x, y) , we have $\underline{FNT(x) > FNT(y)}$.

Computing Finish Time



Algorithm:

1. counter = 1
2. For each $v \in V$:
 $visited(v) = false$
3. For each $v \in V$:
 If ($visited(v) = false$)
 Invoke DFS(v)

DFS(x):

1. Set $visited(x) = true$
2. For each $y \in N(x)$:
 If ($visited(y) = false$):
 Invoke DFS(y)
3. $FNT(x) = counter++$

Homework Question

Let G be a **directed acyclic graph**.

Design an $O(m + n)$ time algorithm without DFS to compute an **ordering**, $v_1, \dots, v_n$ of $V(G)$ satisfying:

for any edge $(v_i, v_j) \in E(G)$, we have $i < j$

