

Tutorial 08 — Graph DS & Traversal

CS2040S Semester 1 2021/2022

By Wu Biao, adapted from previous slides

Set real display name



<https://pollev.com/rezwanarefin430>

Graph Representation

Adjacency Matrix

Adjacency List

Edge List

Graph Representation

Edge List

A list of edges in the entire graph.

Adjacency Matrix

2D Array where `adj_mat[x][y]` stores information about edge $u \rightarrow v$ (or information that it does not exist).

Adjacency List

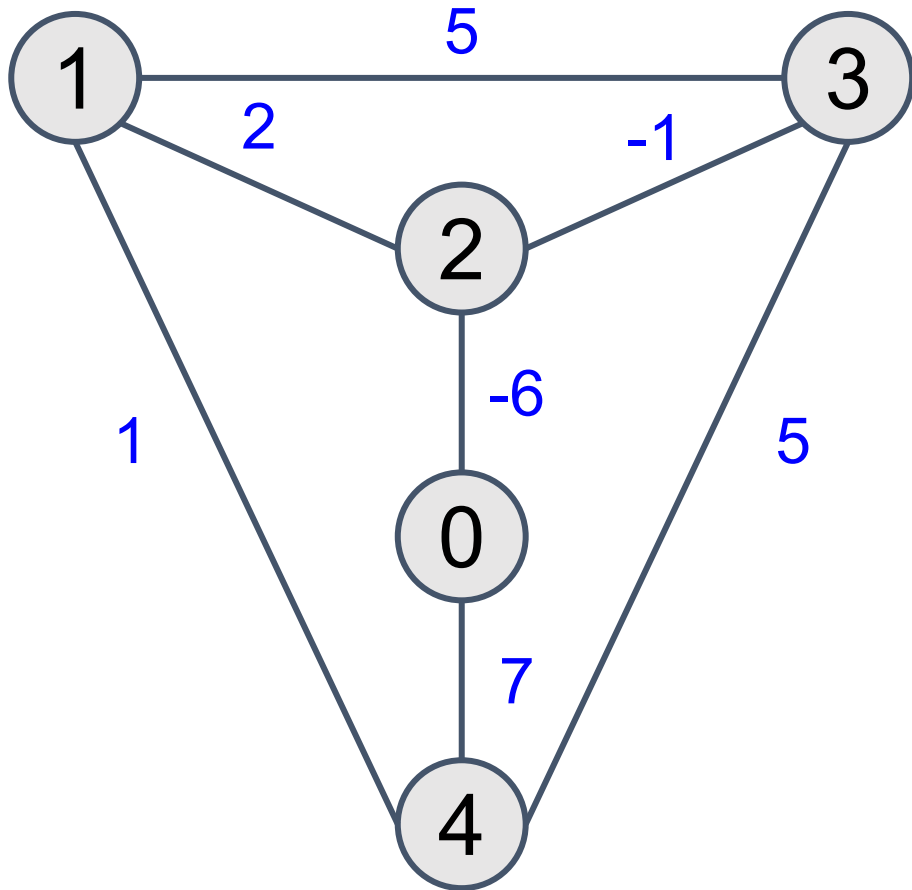
For each vertex, keep a list of vertices it has outgoing edges to

Points to consider

For each of the graph representations, think of the following questions for a graph with V vertices and E edges:

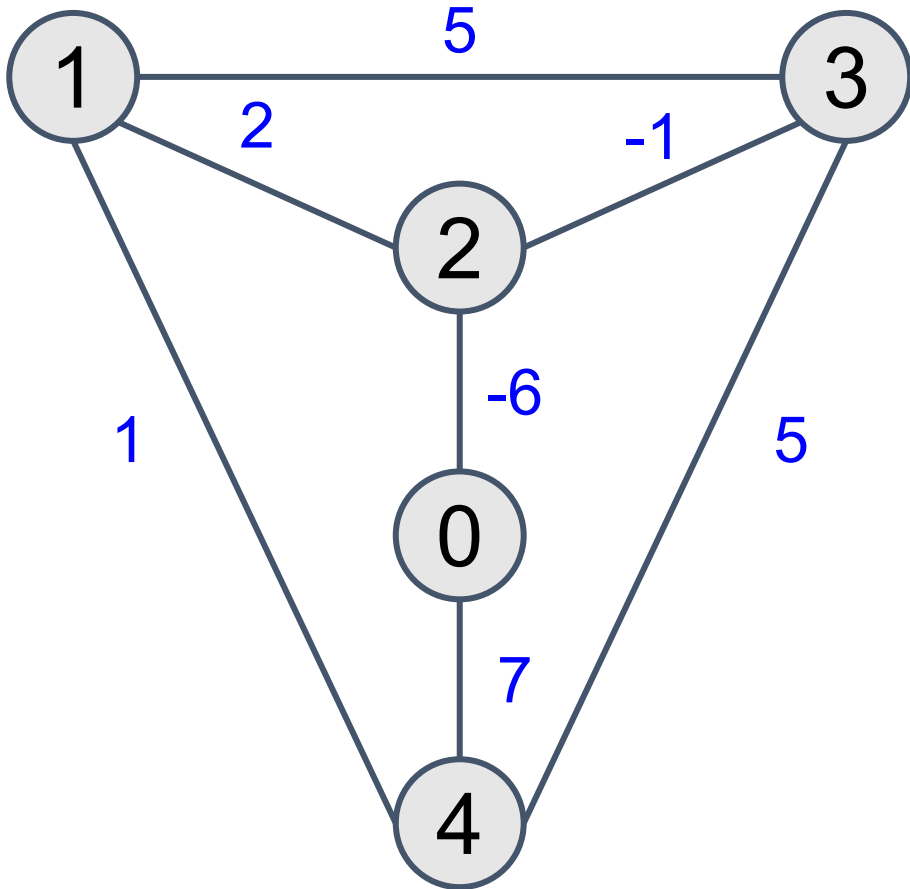
- What's the space complexity?
- What's the time complexity?
 - To verify if an edge exists between u and v .
 - To retrieve a list of all the neighbours of a vertex u .

Graph Representation — Edge List



(Vertex u , Vertex v , Edge weight w)

Graph Representation — Edge List



(Vertex u , Vertex v , Edge weight w)
(0, 2, -6)
(0, 4, 7)
(1, 2, 2)
(1, 3, 5)
(1, 4, 1)
(2, 3, -1)
(3, 4, 5)

Graph Representation — Edge List

- Space complexity: ____
- Time complexity
 - Verify (u, v) is an edge: ____
 - Get neighbours of u : ____

(Vertex u , Vertex v , Edge weight w)
$(0, 2, -6)$
$(0, 4, 7)$
$(1, 2, 2)$
$(1, 3, 5)$
$(1, 4, 1)$
$(2, 3, -1)$
$(3, 4, 5)$

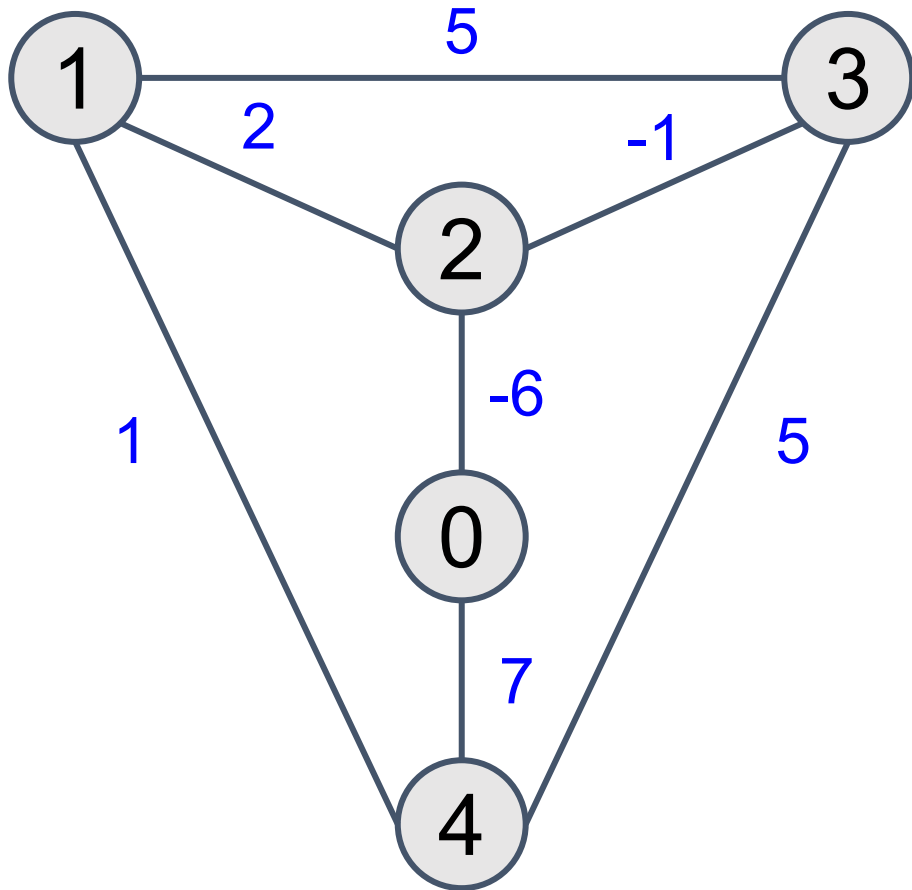
Graph Representation — Edge List

- Space complexity: $O(E)$
- Time complexity
 - Verify (u, v) is an edge: $O(E)$
 - Get neighbours of u : $O(E)$

(Vertex u , Vertex v , Edge weight w)
$(0, 2, -6)$
$(0, 4, 7)$
$(1, 2, 2)$
$(1, 3, 5)$
$(1, 4, 1)$
$(2, 3, -1)$
$(3, 4, 5)$

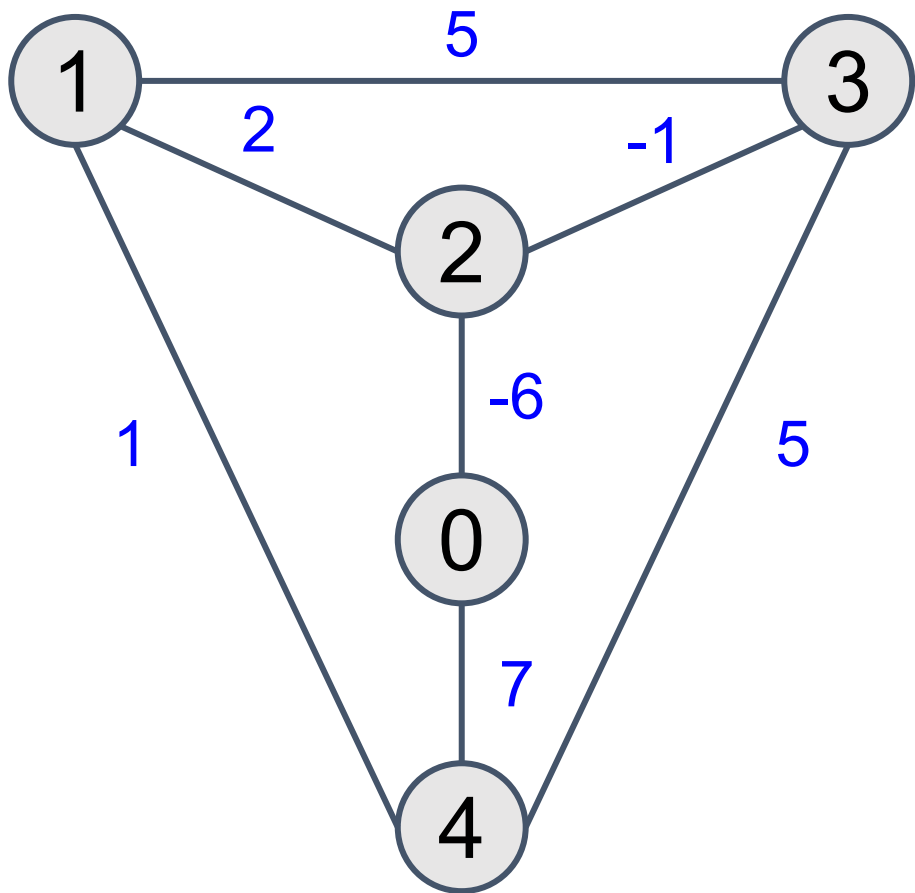
E

Graph Representation — Adjacency Matrix



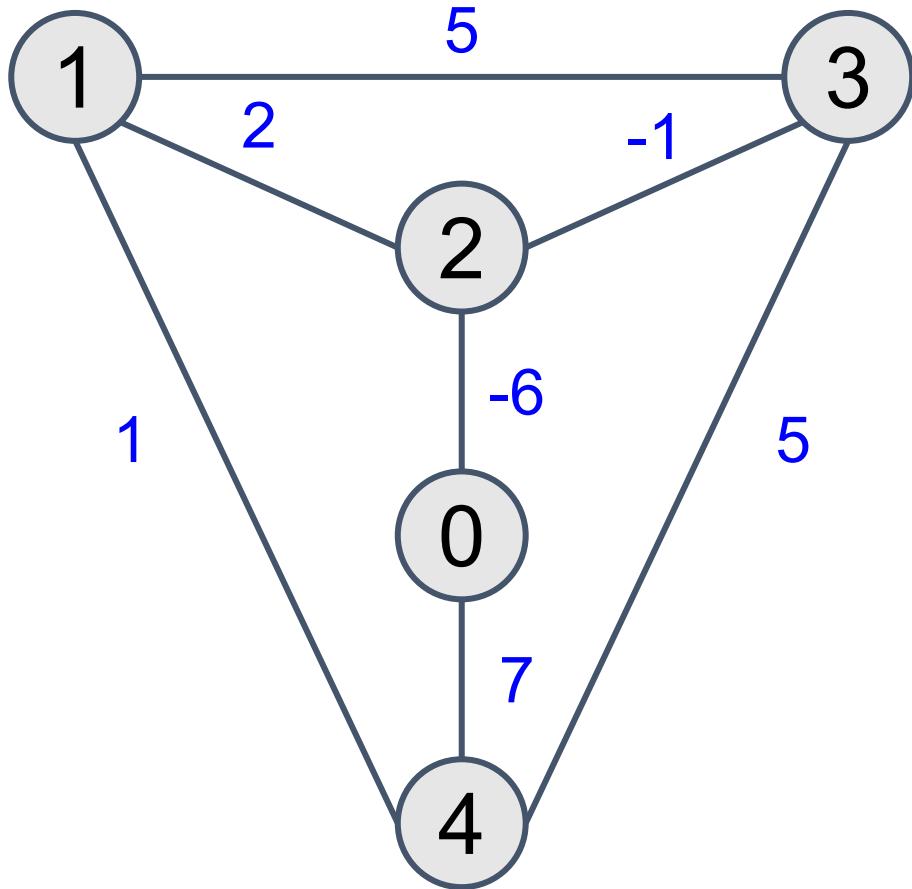
	0	1	2	3	4
0					
1					
2					
3					
4					

Graph Representation — Adjacency Matrix



	0	1	2	3	4
0			-6		7
1			2	5	1
2	-6	2		-1	
3		5	-1		5
4	7	1		5	

Graph Representation — Adjacency Matrix



	0	1	2	3	4
0			-6		7
1			2	5	1
2	-6	2		-1	
3		5	-1		5
4	7	1		5	

Symmetry along diagonal
for bidirectional graphs!

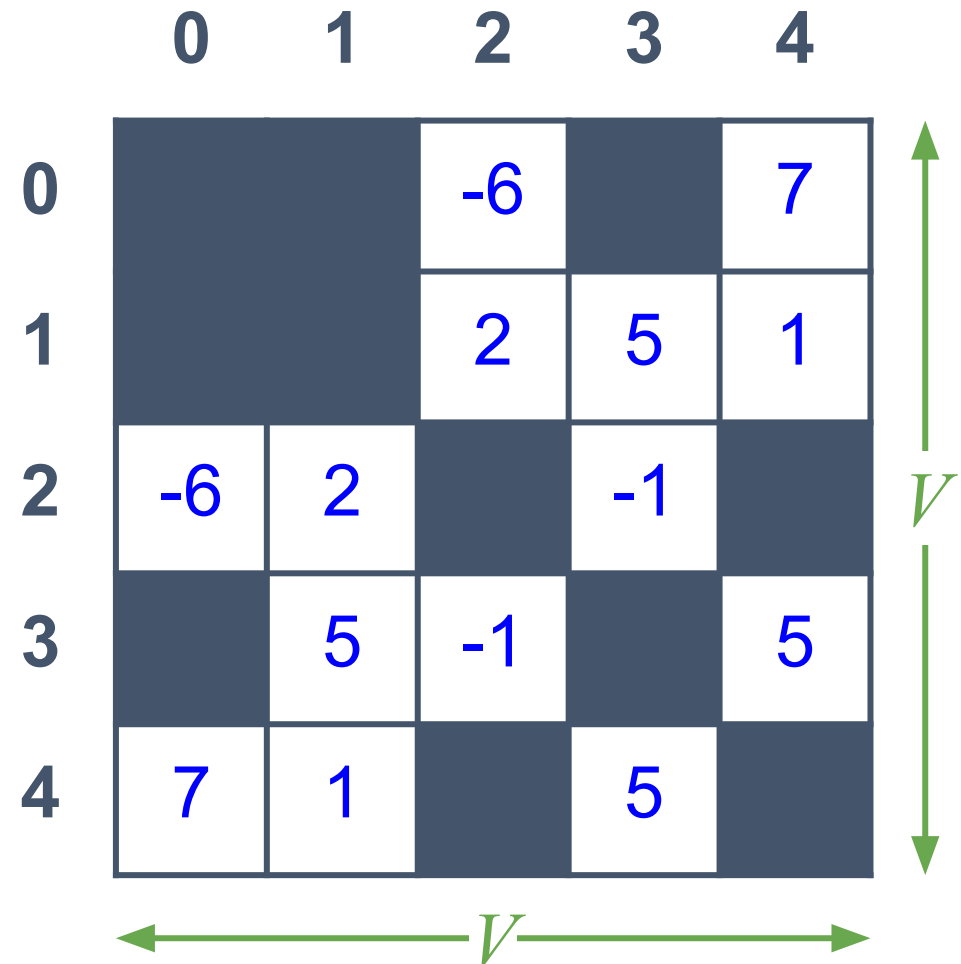
Graph Representation — Adjacency Matrix

- Space complexity: ____
- Time complexity
 - Verify (u, v) is an edge: ____
 - Get neighbours of u : ____

	0	1	2	3	4
0			-6		7
1			2	5	1
2	-6	2		-1	
3		5	-1		5
4	7	1		5	

Graph Representation — Adjacency Matrix

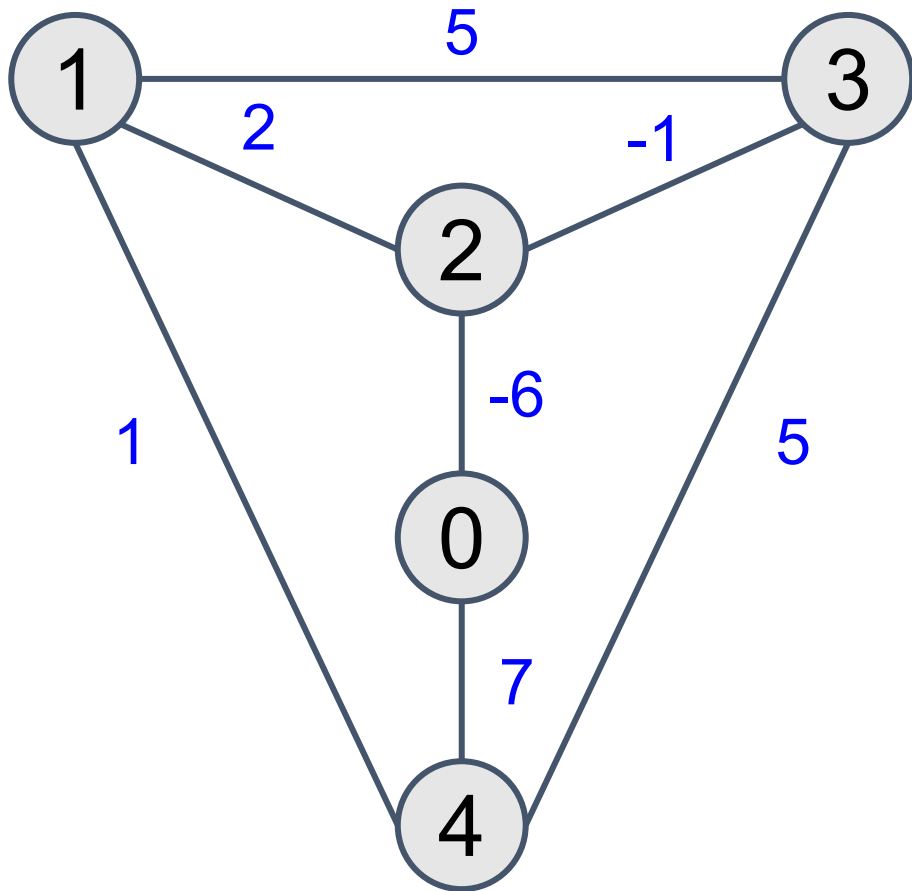
- Space complexity: $O(V^2)$
- Time complexity
 - Verify (u, v) is an edge: $O(1)$
 - Get neighbours of u : $O(V)$



The diagram shows a 5x5 adjacency matrix for a graph with 5 vertices. The rows and columns are indexed from 0 to 4. The matrix is symmetric, indicating an undirected graph. The diagonal elements are all 0. The values in the matrix represent the weights of the edges between vertices. Green arrows indicate the dimensions of the matrix, both labeled as V .

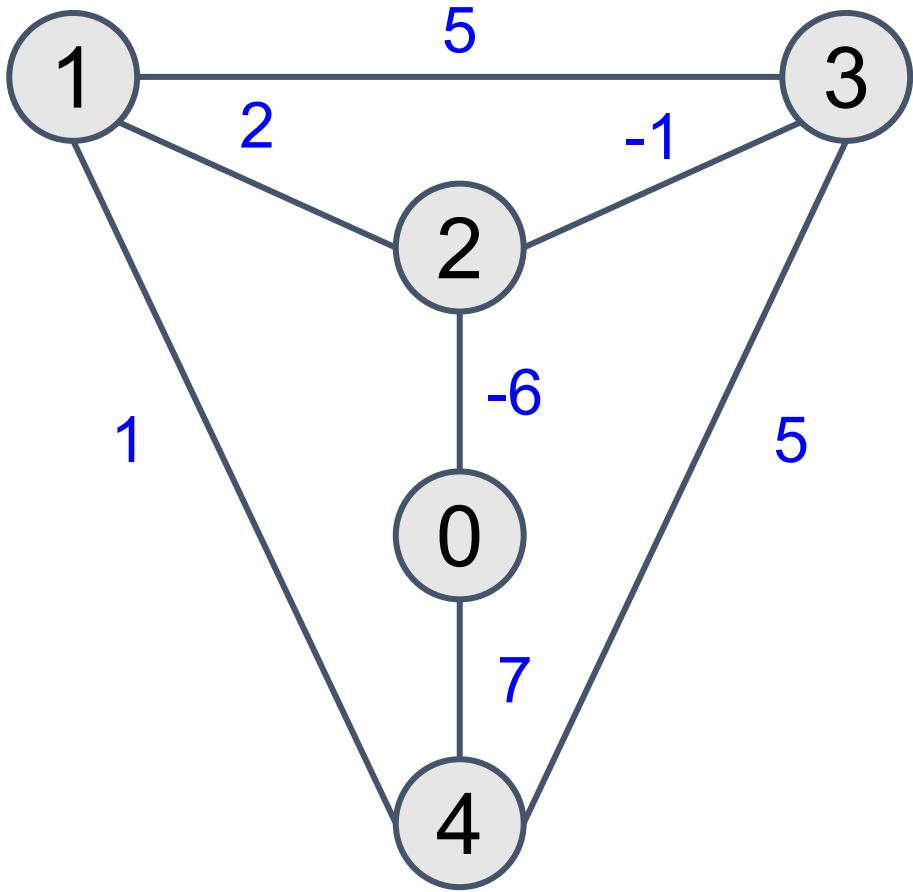
	0	1	2	3	4
0			-6		7
1			2	5	1
2	-6	2		-1	
3		5	-1		5
4	7	1		5	

Graph Representation — Adjacency List



Vertex u	List(vertex v , weight w)
0	
1	
2	
3	
4	

Graph Representation — Adjacency List



Vertex u	List(vertex v , weight w)
0	(2, -6), (4, 7)
1	(2, 2), (3, 5), (4, 1)
2	(0, -6), (1, 2), (3, -1)
3	(1, 5), (2, -1), (4, 5)
4	(0, 7), (1, 1), (3, 5)

Graph Representation — Adjacency List

- Space complexity: ____
- Time complexity
 - Verify (u, v) is an edge: ____
 - Get neighbours of u : ____

Vertex u	List(vertex v , weight w)
0	(2, -6), (4, 7)
1	(2, 2), (3, 5), (4, 1)
2	(0, -6), (1, 2), (3, -1)
3	(1, 5), (2, -1), (4, 5)
4	(0, 7), (1, 1), (3, 5)

Graph Representation — Adjacency List

- Space complexity: $O(V + E)$
- Time complexity
 - Verify (u, v) is an edge: $O(V)$
 - Get neighbours of u : $O(1)$

Vertex u	List(vertex v , weight w)
0	(2, -6), (4, 7)
1	(2, 2), (3, 5), (4, 1)
2	(0, -6), (1, 2), (3, -1)
3	(1, 5), (2, -1), (4, 5)
4	(0, 7), (1, 1), (3, 5)



V

E if directed
 $2E$ if undirected

Graph Representation — Parent Array

Representing a tree

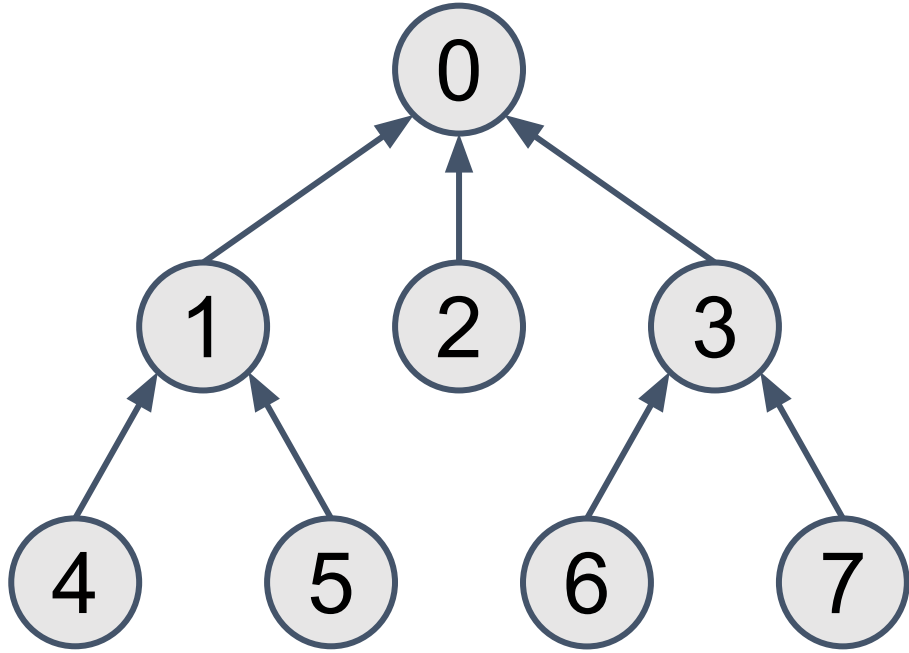
What is the most concise way to capture all the information regarding a tree?

Realize that a tree entails hierarchy!

How many parent does each vertex have?

What if all edges are from child \rightarrow parent?

Graph Representation — Parent Array



Vertex	Parent
0	
1	0
2	0
3	0
4	1
5	1
6	3
7	3

We will revisit parent/predecessor/previous arrays when we learn about SSSP (last topic)

DAG

Directed **A**cyclic **G**raph

Directed Acyclic Graph (DAG)

Definition

- Directed: Edges are **not** bidirectional
- Acyclic: No cycles and no self-loops
- Graph

Directed Acyclic Graph (DAG)

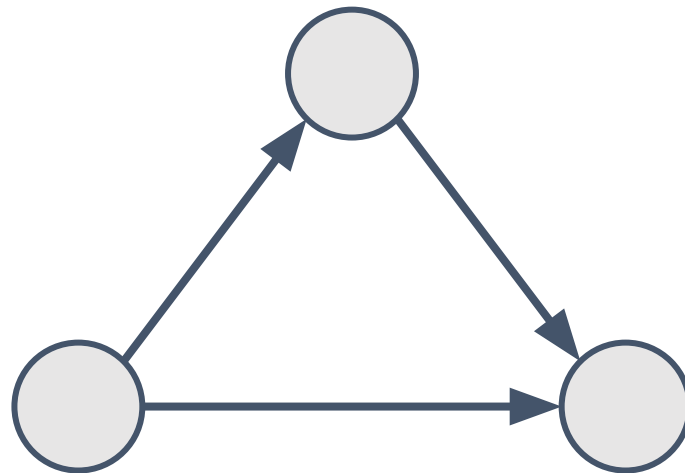
Properties

After traversing an edge from vertex $u \rightarrow v$,

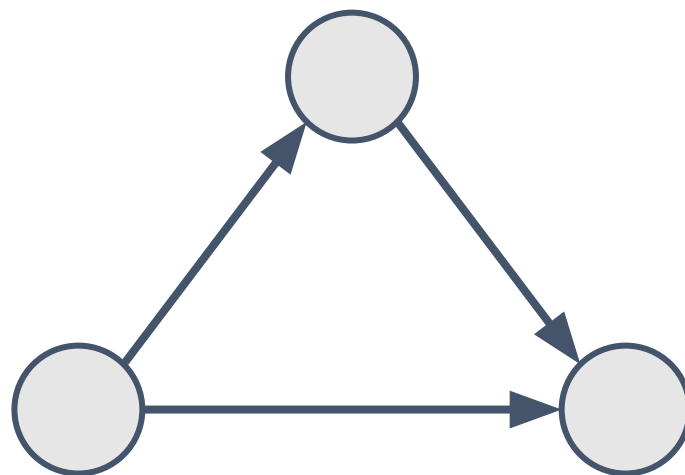
You can never reach vertex u again through any series of directed edges.

Can you prove it? [By contradiction]

Is this a DAG?

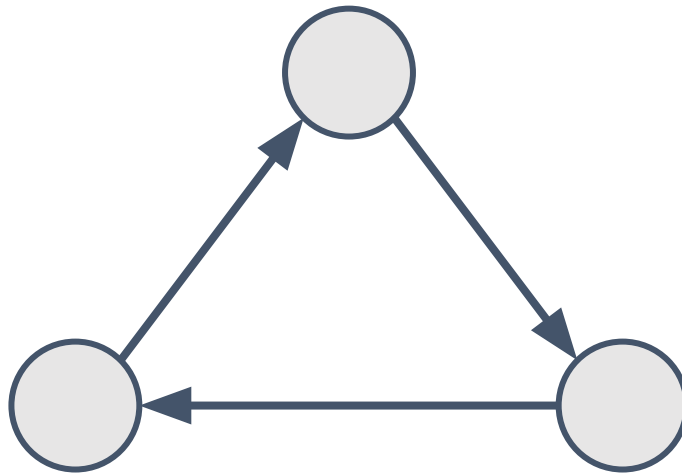


Is this a DAG?

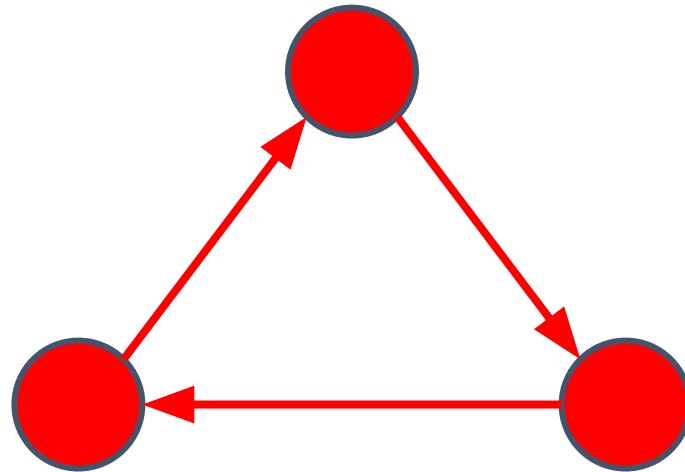


Yes

Is this a DAG?

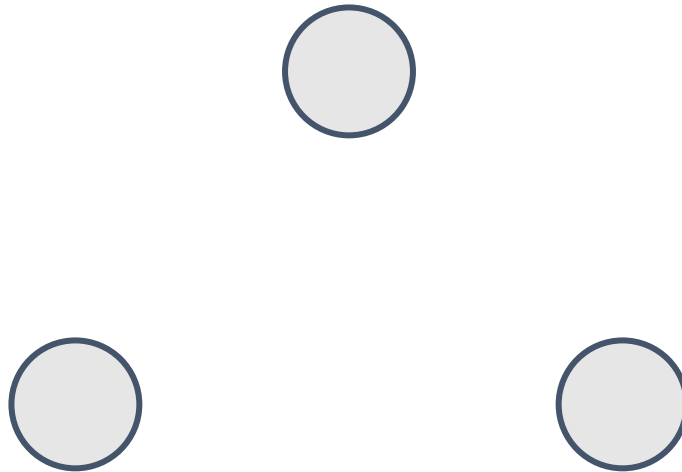


Is this a DAG?

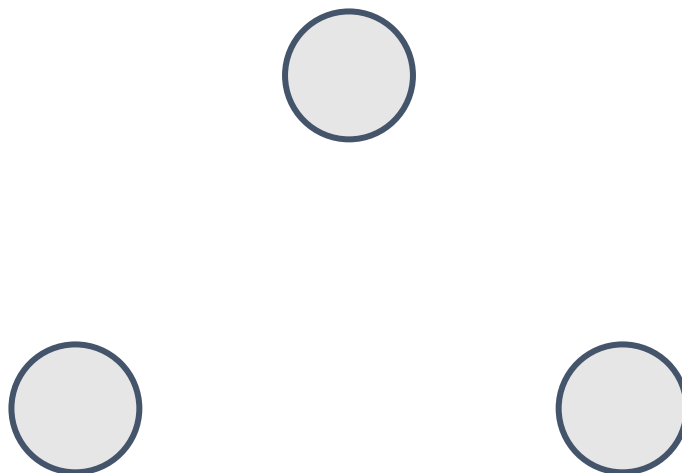


No

Is this a DAG?

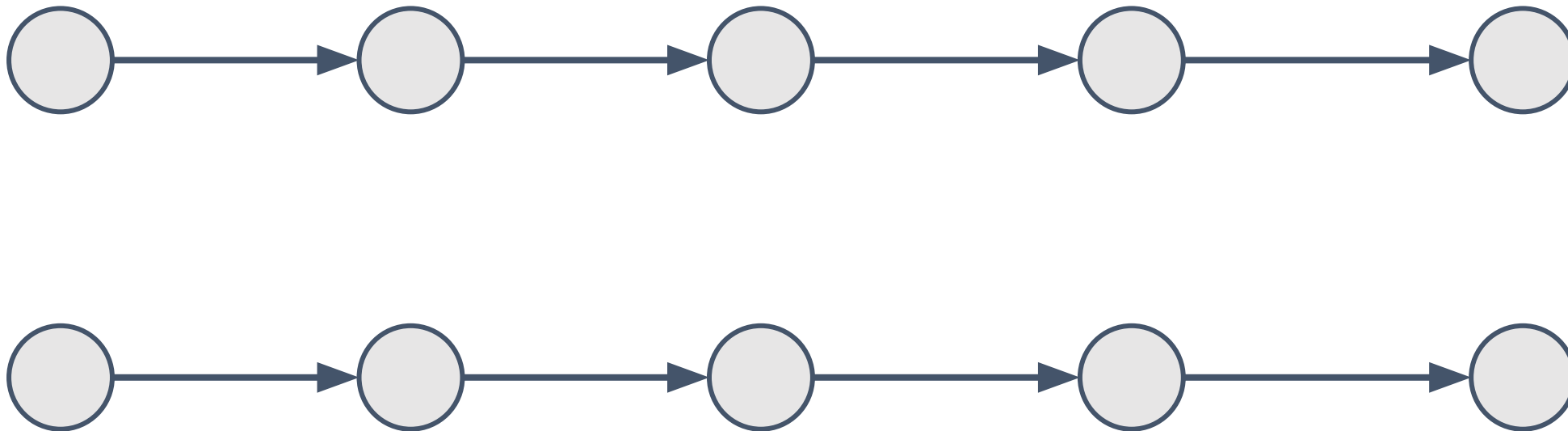


Is this a DAG?



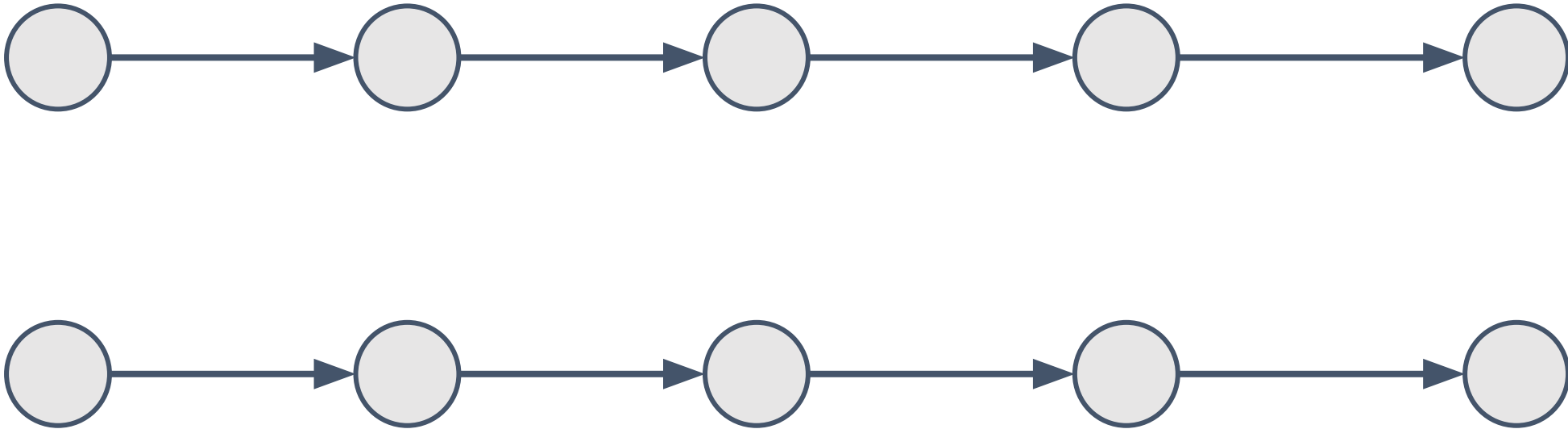
Yes

Is this a DAG?

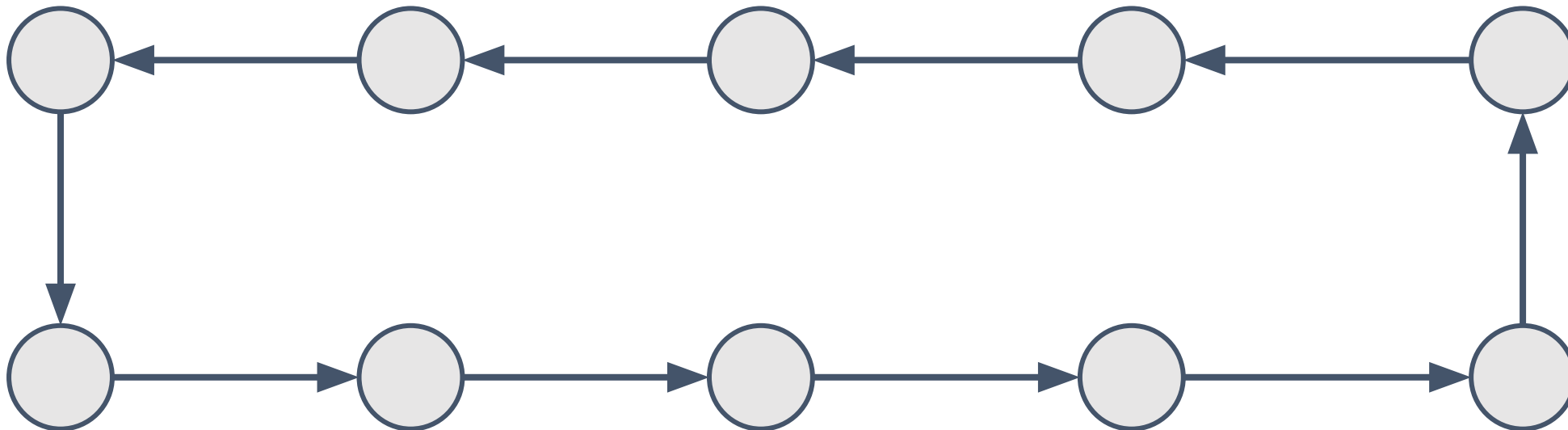


Is this a DAG?

Yes

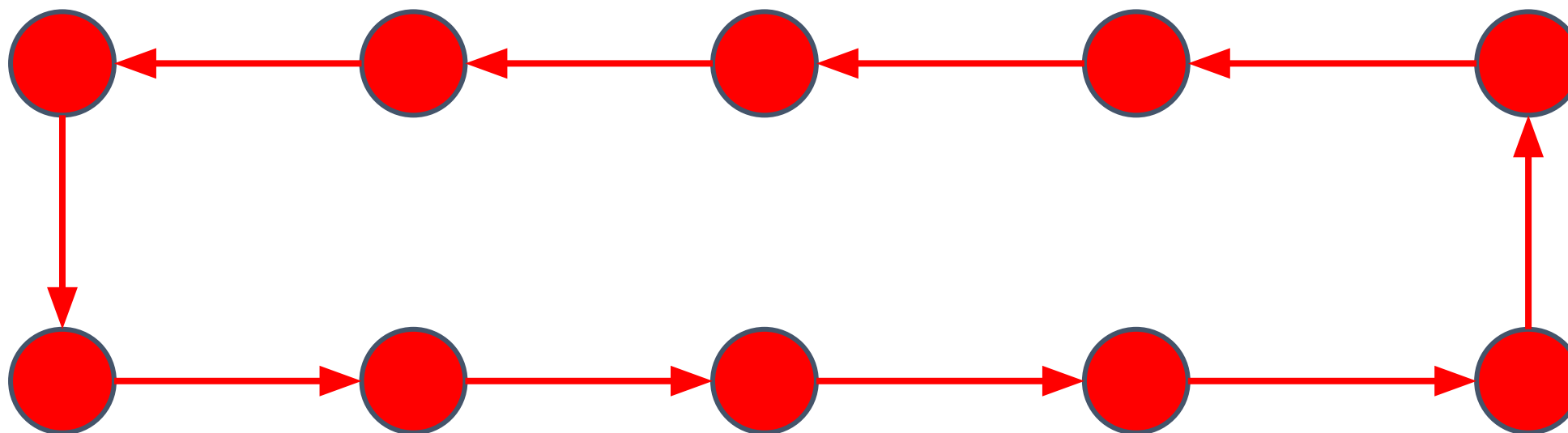


Is this a DAG?

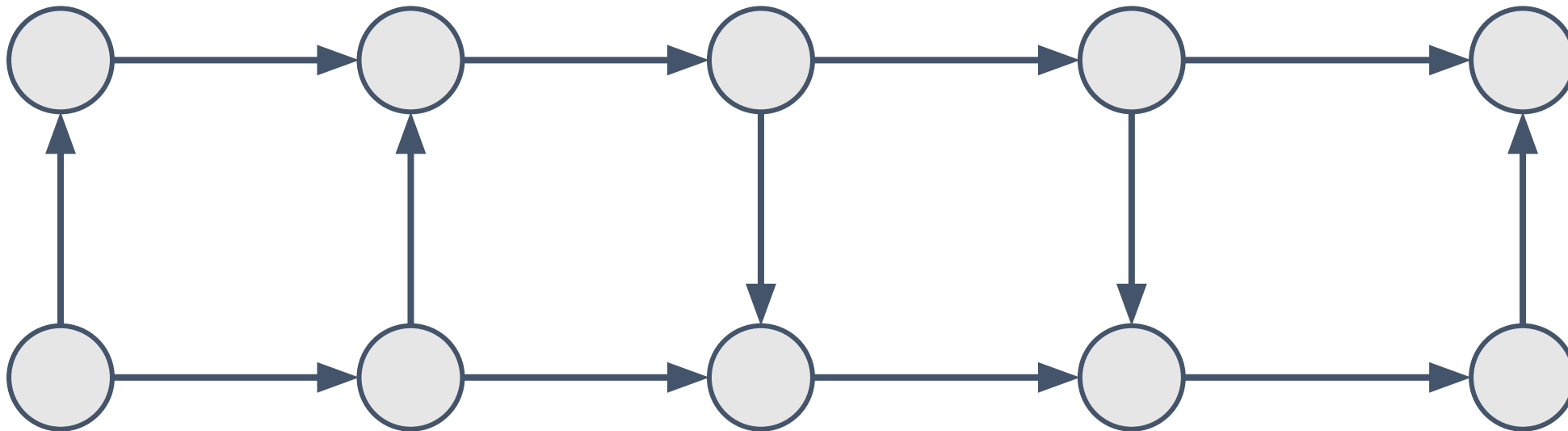


Is this a DAG?

No

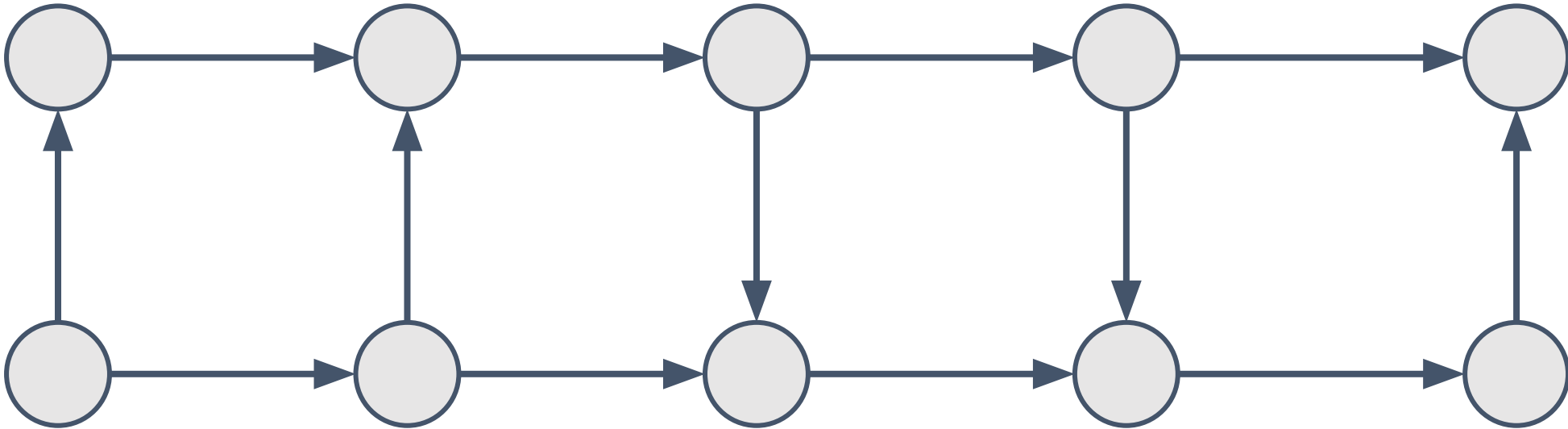


Is this a DAG?

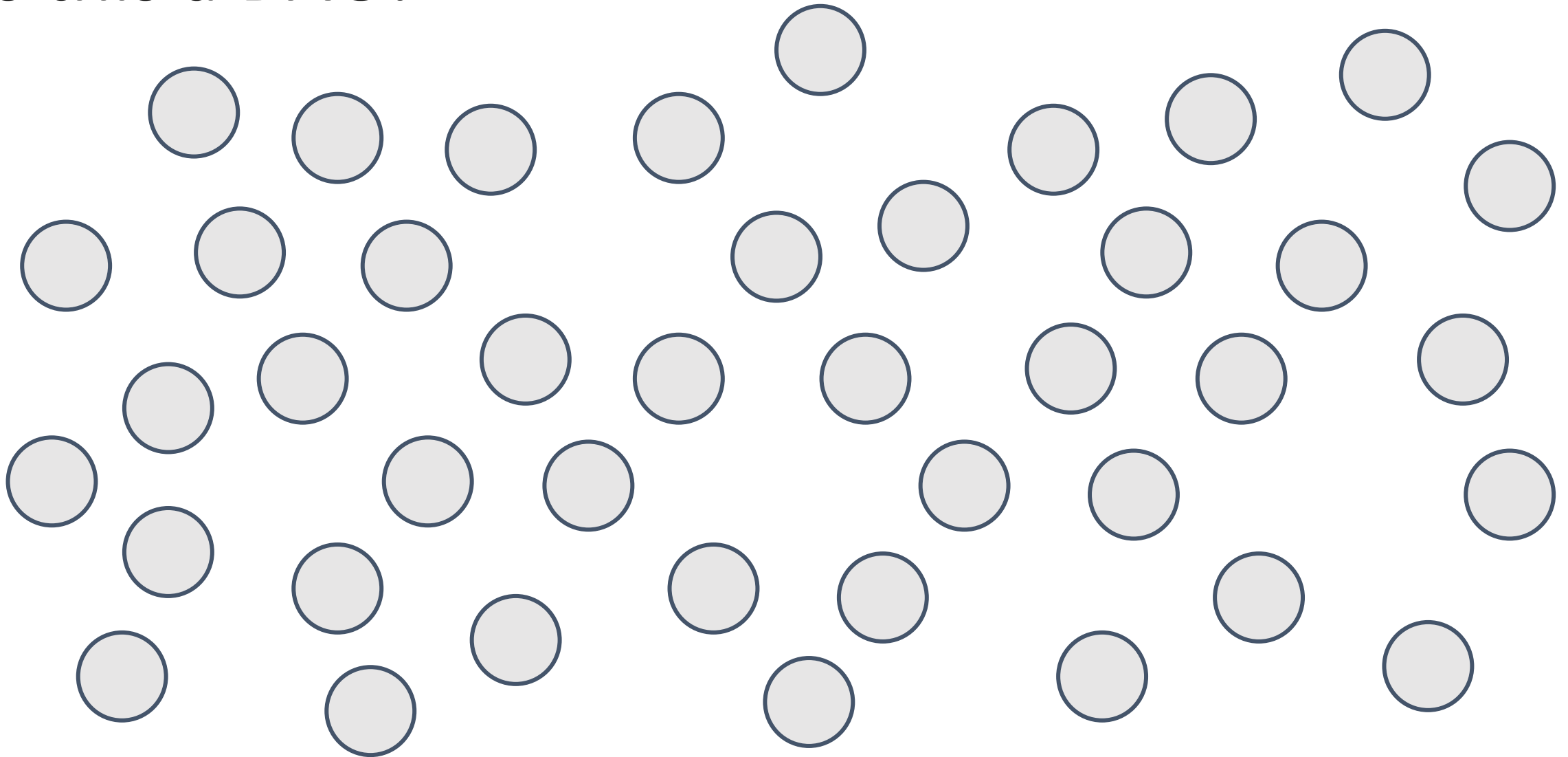


Is this a DAG?

Yes

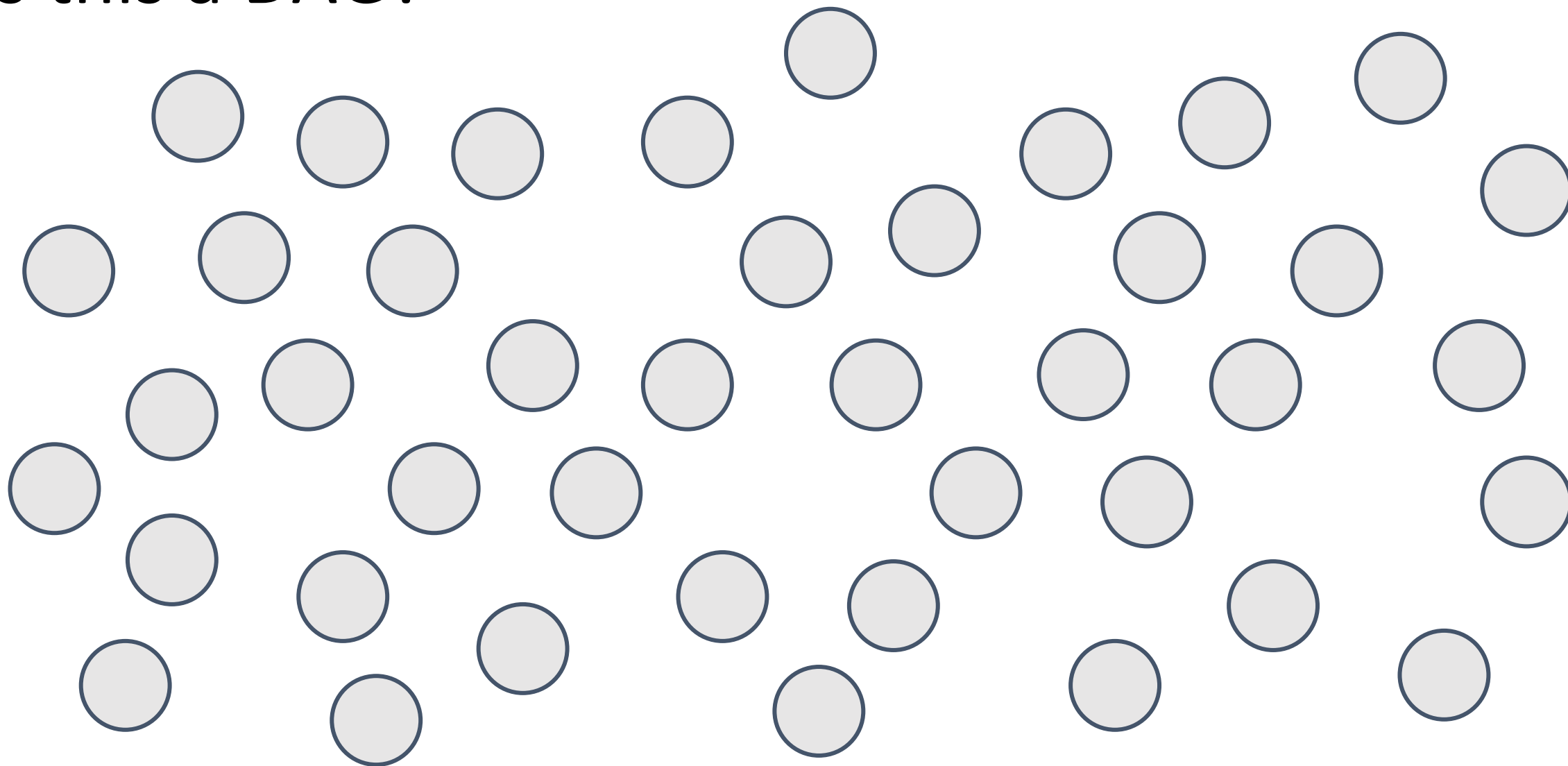


Is this a DAG?

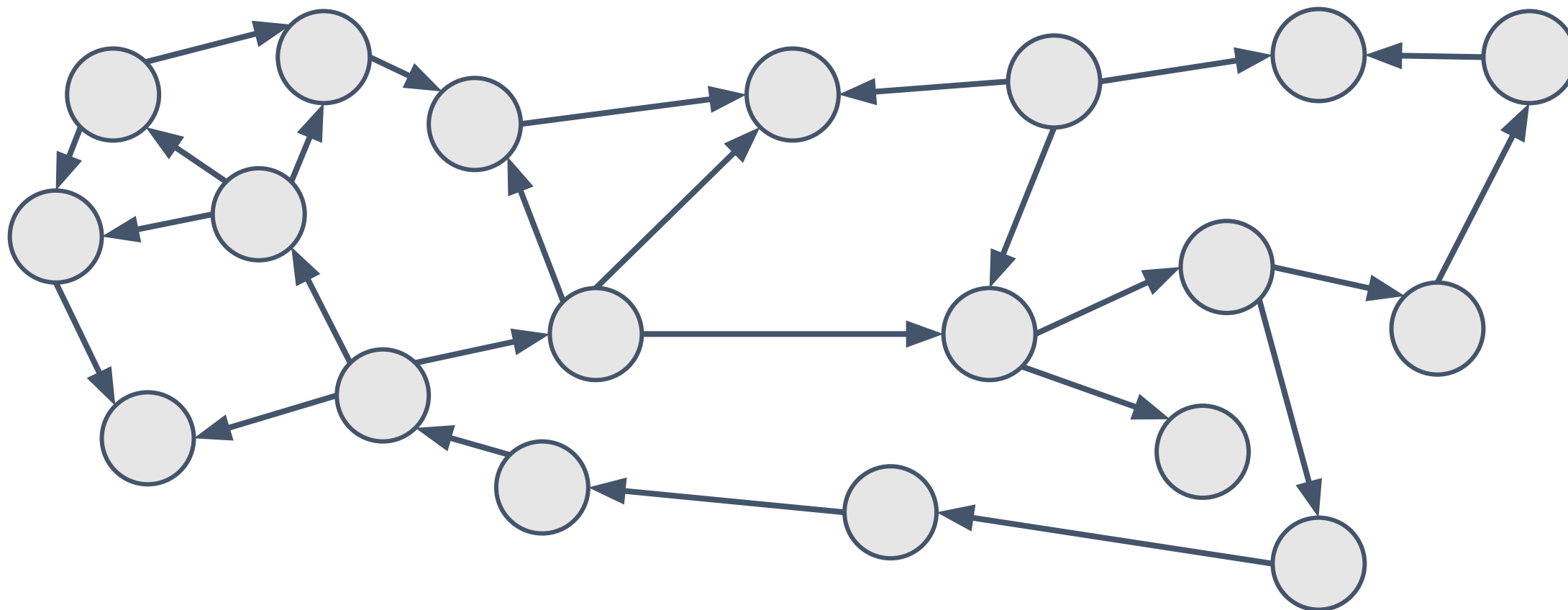


Is this a DAG?

Yes

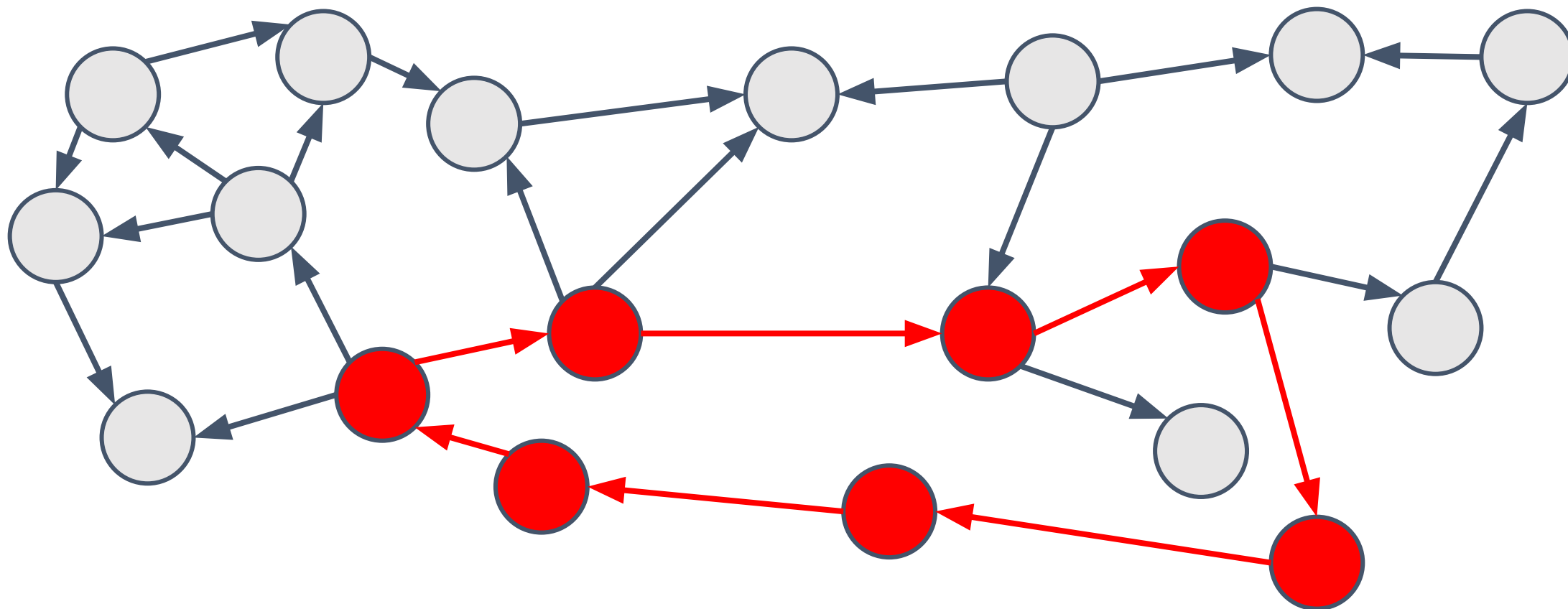


Is this a DAG?

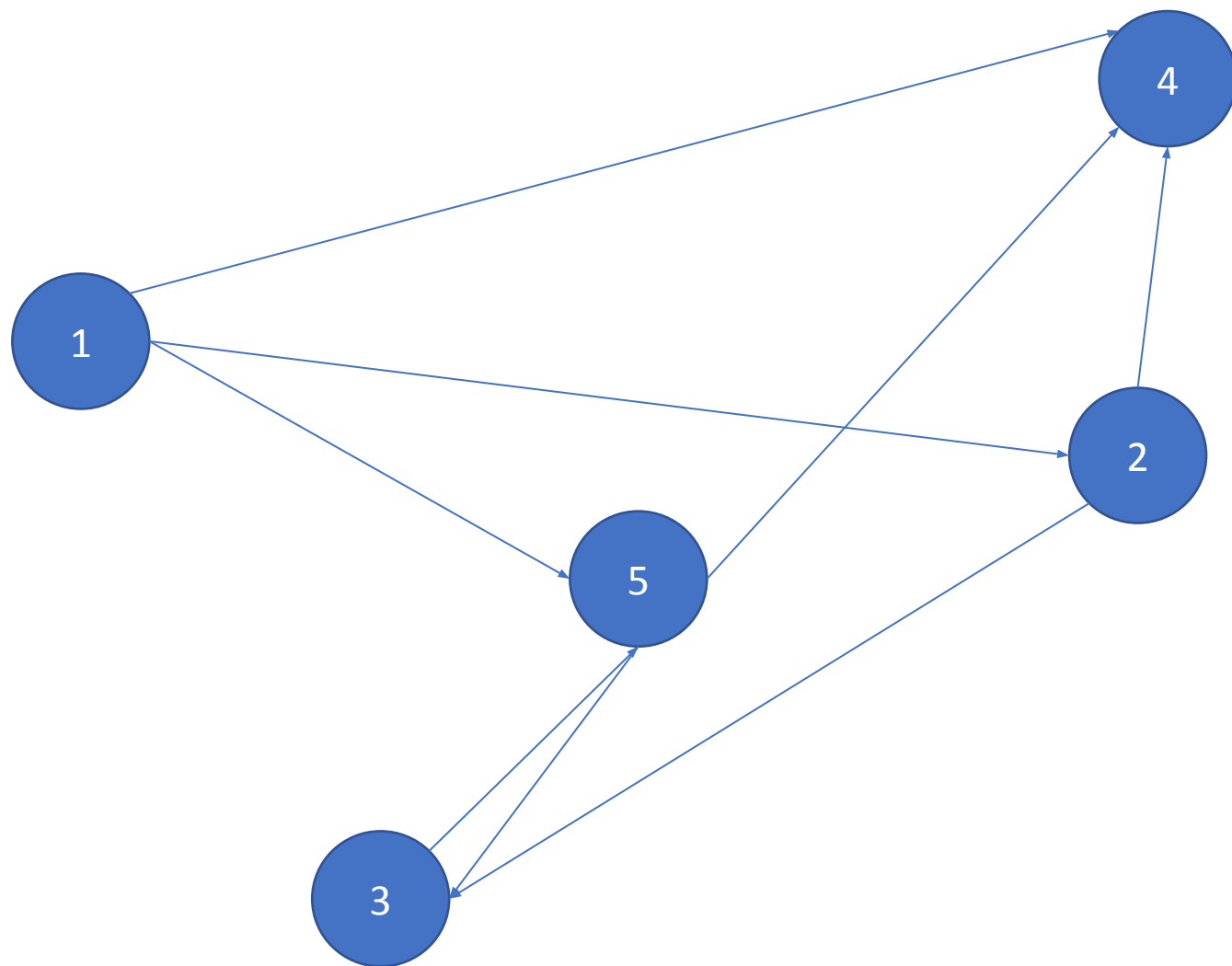


Is this a DAG?

No



Question 1: Graph DS Demo



Question 2(a)

Problem statement

Draw a DAG with V vertices and $E = V(V - 1)/2$ directed edges.

Observations

How to start?

Here's an idea: Observe how the graph property when we change the number of vertices by 1.

So let's consider the cases when V is $k-1$, k and $k+1$

Observations

$k-1$ vertices: $(k-1)(k-2)/2$ edges

k vertices: $(k)(k-1)/2$ edges

$k+1$ vertices: $(k+1)(k)/2$ edges

Observations

When we increased V from $(k-1)$ to k :

- E increased by $(k-1)$

When we increased V from k to $(k+1)$:

- E increased by k

Observations

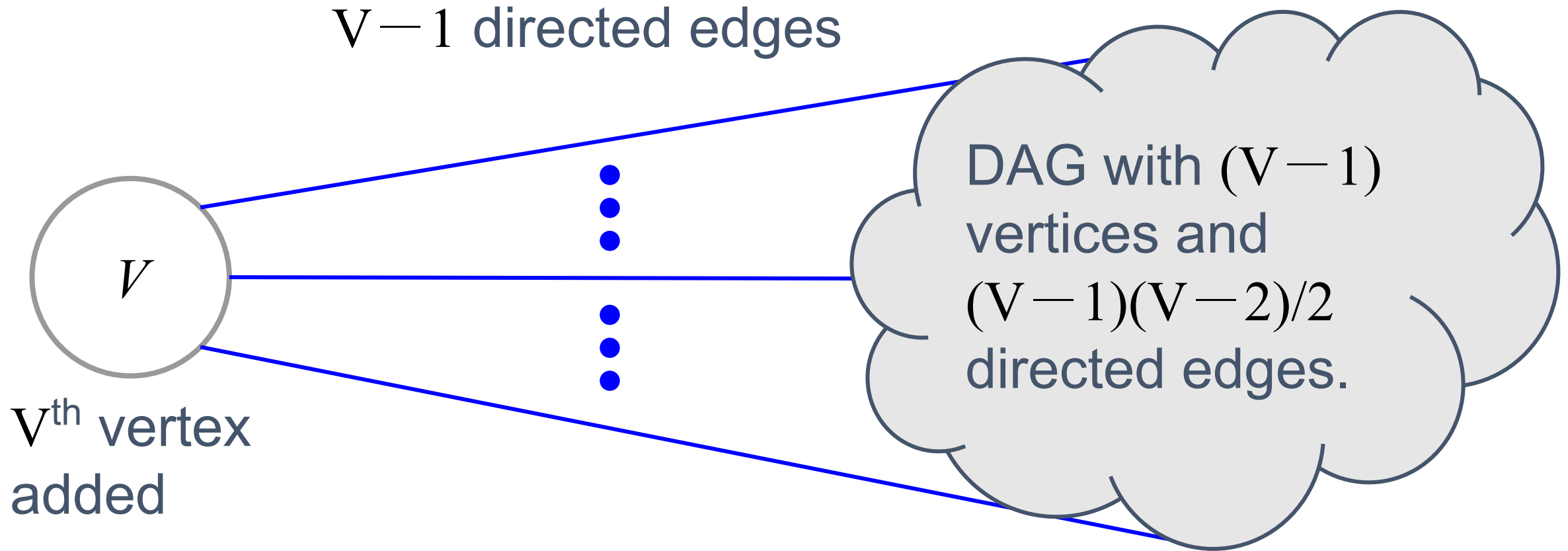
$V - 1$ vertices: $(V - 1)(V - 2)/2$ edges

V vertices: $(V)(V - 1)/2$ edges

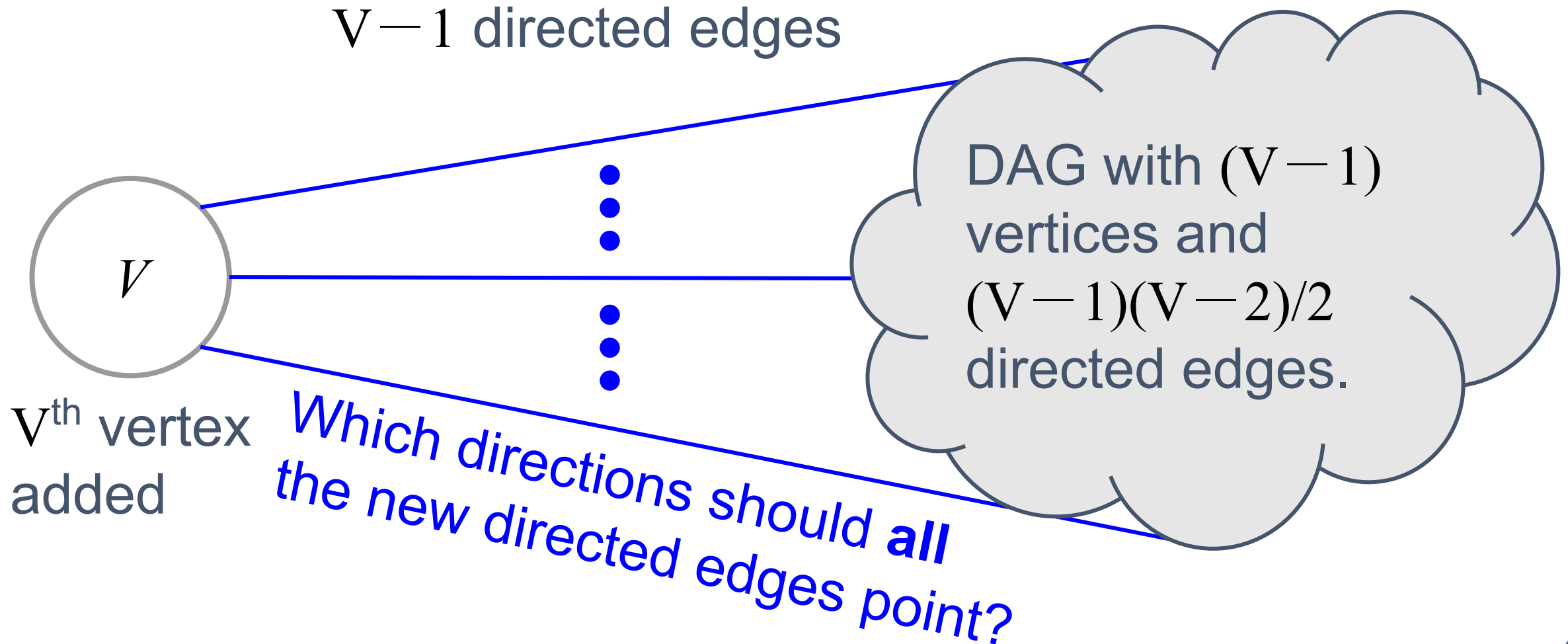
Assuming we can draw a graph with $V - 1$ vertices, we can construct a solution by:

- Adding 1 vertex and $V - 1$ directed edges to it
- While maintaining DAG property

Observations



Observations



Construction

Let's start by drawing a graph with only 1 vertex and 0 edges

Note that it is a DAG.

It has $(V)(V-1)/2=0$ directed edges.

Construction

We can now inductively construct a solution with the following algorithm:

We label the first vertex as 1.

For each vertex u from 2 to V ,

- Draw a directed edge from vertex u to vertex $v < u$



Test yourself!

On the edge

Can we have more than $V(V-1)/2$ directed edges for a DAG with V vertices?



Test yourself!

On the edge

Can we have more than $V(V-1)/2$ directed edges for a DAG with V vertices?

No!

With $V(V-1)/2$ directed edges, there is already exactly 1 edge between every pair of vertices. Adding any more will form a bi-directional edge.

Question 2(b)

Problem statement

Draw a Bipartite Graph with V vertices (assume that V is even) and $V^2/4$ undirected edges.

Solution

Say we have a vertices on 1 side, and b vertices on the other. What can we conclude?

Solution

Think this: say we have a vertices on 1 side, and b vertices on the other.
What can we conclude?

We have $a+b=V$, and number of edges are $a*b$ at most (why?)

Solution

Now we want $V^2/4$ edges, this can be achieved if $a=b=V/2$

Can the number of edges $> V^2/4$?

Question 2(c)

Problem statement

Draw a Tree with V vertices (and $E = V - 1$ edges) that is not a Bipartite graph.

Solution

Draw a Tree with V vertices (and $E = V - 1$ edges) that is not a Bipartite graph.

No Way!

All trees are bipartite. Assume even and odd depth vertices are on different sides.

Question 3

Problem Statement

Show what is the best (fastest) way to convert a graph currently stored in graph data structure 1 into graph data structure 2.

Q3a). From Adjacency Matrix (AM) to Adjacency List (AL)

Q3b). From AM to Edge List (EL)

Q3c). From AL to AM

Q3d). From AL to EL

Q3e). From EL to AM

Q3f). From EL to AL

Q3a

From Adjacency Matrix (AM) to Adjacency List (AL)

Q3a

From Adjacency Matrix (AM) to Adjacency List (AL)

Read all cells in $O(V^2)$, for each edge (u, v) , push back v to the back of $AL[u]$.

Q3b

From AM to Edge List (EL)

Q3b

From AM to Edge List (EL)

$O(V^2)$, same as above, but this time, push back (u, v) to the back of EL.

Q3c

From AL to AM

From AL to AM

$O(V + E)$, read all edges in $O(V + E)$, insert at row u , column v (assuming the V^2 cells already set to all zeroes beforehand).

Q3d

AL to EL

Q3d

AL to EL

$O(V + E)$, same as above, but this time, push back (u, v) to the back of EL.

Q3e

EL to AM

Q3e

EL to AM

$O(E)$, read all edges in $O(E)$, insert at row u , column v (assuming the V^2 cells already set to all zeroes beforehand).

Q3f

EL to AL

Q3f

EL to AL

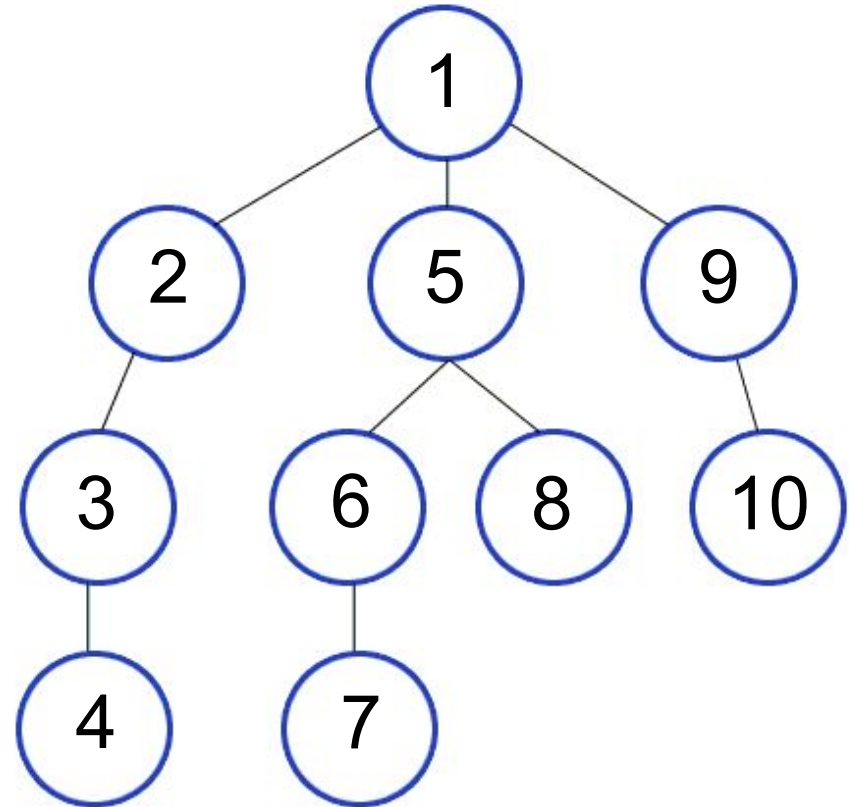
$O(E)$, same as above, but this time push back v to the back of $AL[u]$.

Question 4: DFS Review

Depth First Search (DFS)

Likes to go *deeper*!

It will only backtrack if there is no other way to continue deeper.



Note: Order of visitation is marked by number within vertex.

Depth First Search — Pre-order Traversal

Recall that previously we introduced DFS for **binary trees** via the following pre-order traversal algorithm?

```
void dfs(vertex u) {  
    cout << u.id << endl; // Visit operation  
    if (u.left)  dfs(u.left);  
    if (u.right) dfs(u.right);  
}
```

How can we generalize this for **graphs**? i.e. When vertices are no longer restricted to having just 2 neighbours

Depth First Search — Pre-order Traversal

You may be tempted to just update it to the following. This is a classic mistake which will lead to infinite recursion! Why?

```
void dfs(Integer u) {  
    System.out.println(u); // Visit operation  
    for (Integer v : adj[u]){  
        dfs(v);  
    }  
}
```

Recurse down every neighbour

Use any graph and trace out the code!

Depth First Search — Pre-order Traversal

We should only recurse further on unvisited vertices! Just ignore those which have been visited before!

```
void dfs(Integer u) {  
    if (visited[u]) return;  
    visited[u] = true;  
    System.out.println(u);  
    for (Integer v : adj[u]) {  
        dfs(v);  
    }  
}
```

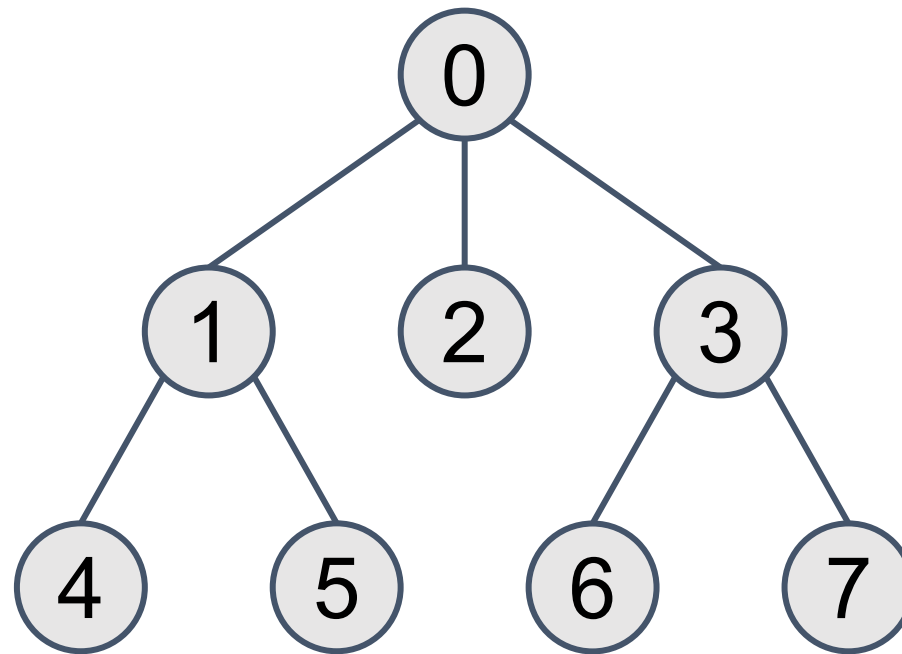
Just fix by adding
these 2 new lines!

Depth First Search — Pre-order Traversal

There are other ways of writing the logic for visitation tracking, such as this:

```
void dfs(Integer u) {  
    visited[u] = true;  
    System.out.println(u);  
    for (Integer v : adjList[u]) {  
        if (visited[v]) continue;  
        dfs(v);  
    }  
}
```

Depth First Search — Pre-order Traversal



Output:

0, 1, 4, 5, 2, 3, 6, 7

Challenge yourself!

Implement DFS *without* recursion.

It is ok to not try this!

Hint: What DS have you learnt that exhibits recursive property?

Question 5: UFDS Revisited



Question 5: UFDS Revisited

Given an undirected graph. Count number of connected components.

How to do this with UFDS?



Question 5: UFDS Revisited

Given an undirected graph. Count number of connected components.

How to do this with UFDS?

- For each edge (u, v) : $\text{Union}(u, v)$
- Count number of i such that $p[i] == i$.

How to do this with DFS?



Question 5: UFDS Revisited

Given an undirected graph. Count number of connected components.

How to do this with UFDS?

- For each edge (u, v) : $\text{Union}(u, v)$
- Count number of i such that $p[i] == i$.

How to do this with DFS?

- $\text{DFS}(i)$ visits all nodes in i 's component. So:
- For each node i : `if (!visited[i]) { dfs(i); ++count; }`

Questions?
Attendance
Break

[https://github.com/stevenhalim/cpbook-code/
blob/master/ch2/ourown/graph_ds.java](https://github.com/stevenhalim/cpbook-code/blob/master/ch2/ourown/graph_ds.java)

[https://github.com/stevenhalim/cpbook-code/
blob/master/ch4/traversal/dfs_cc.java](https://github.com/stevenhalim/cpbook-code/blob/master/ch4/traversal/dfs_cc.java)

<https://visualgo.net/training?diff=Medium&n=5&tl=5&module=graphds,dfsdfs>

PS5 Discussion

PS5 [/kannafriendship](#)

Store disjoint segments in a BBST, sorted by start point.

Suppose we are inserting $[l, r]$.

- Remove all segments intersecting with $[l, r]$ from BBST.
 - Find the first segment that has start point $\geq l$.
 - Check next segments as long as it intersects with $[l, r]$.
 - Check previous segments as long as it intersects with $[l, r]$.
- Insert $[l, r]$ union the removed segments back to the BBST.
- Keep a global sum of length variable.
 - Increase it by L whenever a segment of length L is added to BBST.
 - Decrease it by L whenever a segment of length L is removed from BBST.

PS5 [/traveltheskies](#)

- For each airport i , maintain $C[i]$ = how many people are left here.
- Simulate each day:
 - For each airport i :
 - $C[i] +=$ number of people arrive at airport i from home.
 - Note that $C[i]$ possibly also has left over people from previous days.
 - Try to push $C[i]$ to all outgoing flights from airport i .
 - If we could not use all flights then print “suboptimal” and return.
- Print “optimal”.
- Small constraints allow using any reasonable data structure.
 - One possibility:
 - Adjacency list for each day.
 - 2D array for people arriving from home.

Hands-On Session: [/hermits](#)

Hands-On: [/hermits](#)

There are N streets. On street i , a_i people live. Find the street where total number of people on this street and adjacent streets are minimum.

Hands-On: [/hermits](#)

There are N streets. On street i , a_i people live. Find the street where total number of people on this street and adjacent streets are minimum.

Hint 5 min:

Hint 10 min:

Hint 15 min: Take minimum of the values calculated.

Hands-On: [/hermits](#)

There are N streets. On street i , a_i people live. Find the street where total number of people on this street and adjacent streets are minimum.

Hint 5 min: Streets are node. Edges are intersections. Use an Adjacency List.

Hint 10 min:

Hint 15 min:

Hands-On: [/hermits](#)

There are N streets. On street i , a_i people live. Find the street where total number of people on this street and adjacent streets are minimum.

Hint 5 min: Streets are node. Edges are intersections. Use an Adjacency List.

Hint 10 min: For each street, calculate total number of people in adjacent streets by a loop. Why is this fast?

Hint 15 min:

Hands-On: [/hermits](#)

There are N streets. On street i , a_i people live. Find the street where total number of people on this street and adjacent streets are minimum.

Hint 5 min: Streets are node. Edges are intersections. Use an Adjacency List.

Hint 10 min: For each street, calculate total number of people in adjacent streets by a loop. Why is this fast?

- Each edge is iterated twice in total; once from each direction.

Hint 15 min: Take minimum of the values calculated.

(Actually the 10 min hint part could be calculated while reading input, without storing an Adjacency List)

Thank You!

Anonymous Feedback:

<https://forms.gle/MkETeXdUT53Vhh896>