Depth First Search and Topological Orderings

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Discuss depth first search for graphs
- Discuss topological orderings

Exercise

- DFS run through

Depth-First Search

- Explore more aggressively, and
- Backtrack when needed
- Linear time algorithm (again O(m + n))

- Computes topological ordering (we'll discuss this today)

API

```
FUNCTION DFS(G, start_vertex)
    found = {v: FALSE FOR v IN G.vertices}
    DFSRecursion(G, start_vertex, found)
    RETURN found


FUNCTION DFSRecursion(G, v, found)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSRecursion(G, vOther, found)
```

Why is this non-recursive function necessary?

```
FUNCTION DFS(G, start_vertex)
    found = {v: FALSE FOR v IN G.vertices}
    DFSRecursion(G, start_vertex, found)
    RETURN found


FUNCTION DFSRecursion(G, v, found)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSRecursion(G, vOther, f
```

Why is this non-recursive function necessary?

```
FUNCTION BFS(G, start_vertex)
    found = {v: FALSE FOR v IN G.vertices}
    found[start_vertex] = TRUE
    visit_queue = [start_vertex]

    WHILE visit_queue.length != 0
        vFound = visit_queue.pop()
        FOR vOther IN G.edges[vFound]
            IF found[vOther] == FALSE
                found[vOther] = TRUE
                visit_queue.add(vOther)

    RETURN found
```

```
FUNCTION DFSIterative(G, v)
    found = {v: FALSE FOR v IN G.vertices}
    found[start_vertex] = TRUE
    visit_stack = [start_vertex]

    WHILE visit_stack.length != 0
        vFound = visit_stack.pop()
        FOR vOther IN G.edges[vFound]
            IF found[vOther] == FALSE
                found[vOther] = TRUE
                visit_stack.push(vOther)

    RETURN found
```

Why is this non-recursive function necessary?

What kind of data structure would we need for an iterative version?

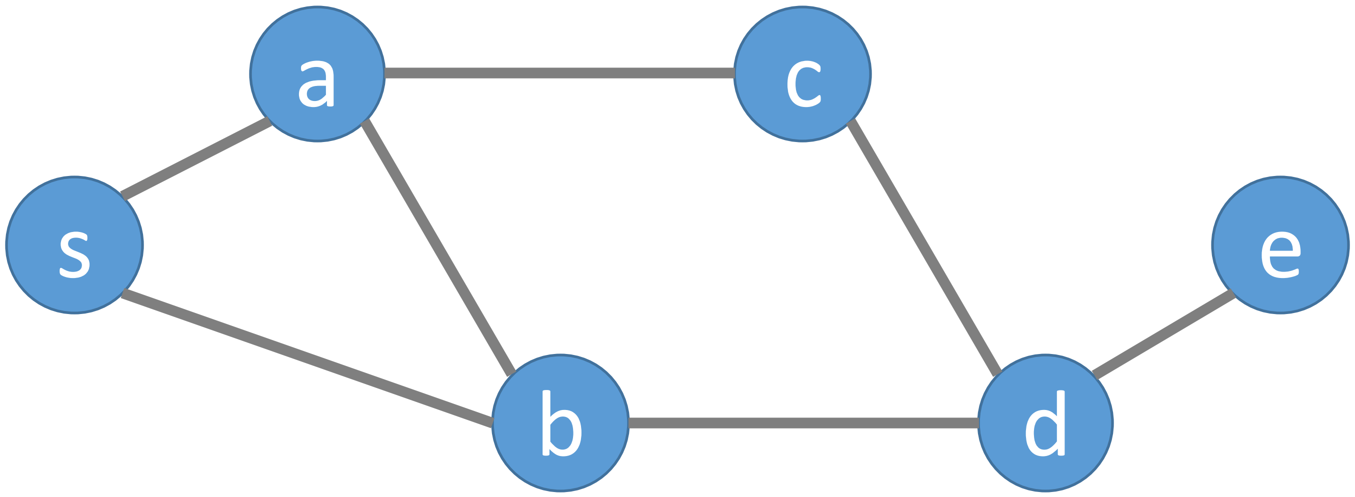
```
FUNCTION BFS(G, start_vertex)
    found = {v: FALSE FOR v IN G.vertices}
    found[start_vertex] = TRUE
    visit_queue = [start_vertex]

    WHILE visit_queue.length != 0
        vFound = visit_queue.pop()
        FOR vOther IN G.edges[vFound]
            IF found[vOther] == FALSE
                found[vOther] = TRUE
                visit_queue.add(vOther)

    RETURN found
```

```
FUNCTION DFSRecursion(G, v, found)
   found[v] = TRUE
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         DFSRecursion(G, vOther, found)
```

```
FUNCTION DFS(G, start_vertex)
   found = {v: FALSE FOR v in G.vertices}
   DFSRecursion(G, start_vertex, found)
   RETURN found
```



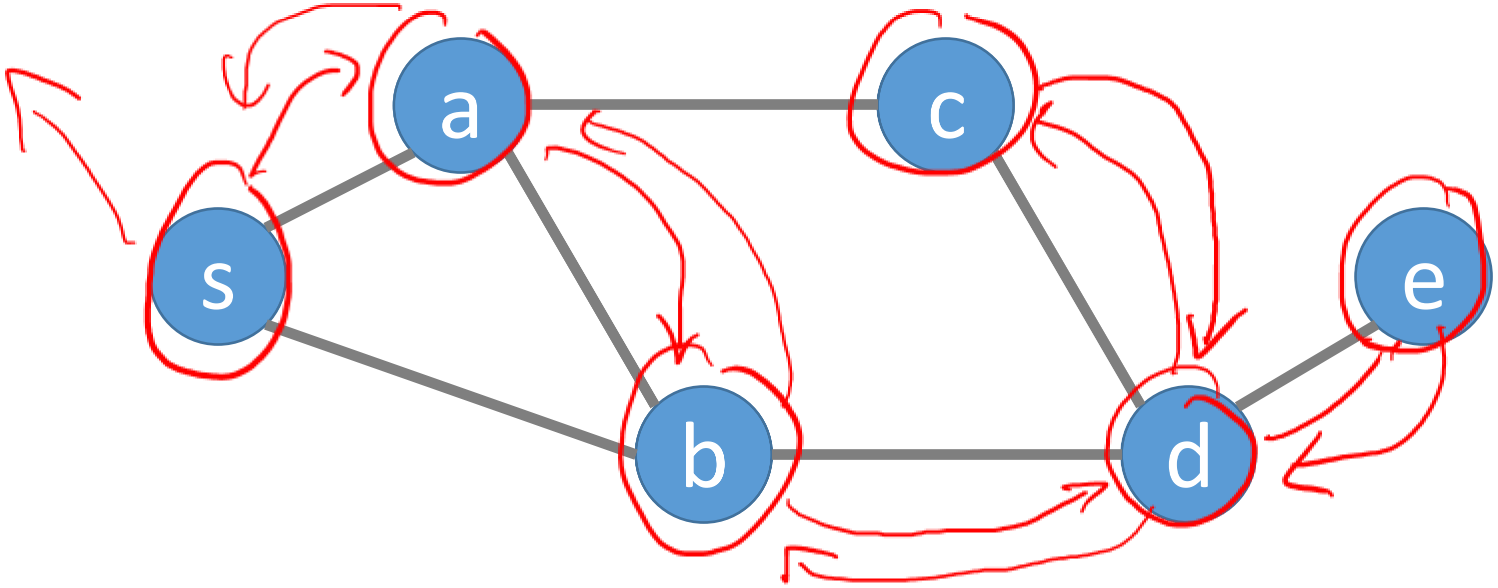
Given a tie, visit edges are in alphabetical order

Exercise

found = [s a b c d e]

DFS(s)
DFS(a)
DFS(b)
DFS(d)
DFS(c)
DFS(e)

```
FUNCTION DFSRecursion(G, v, found)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSRecursion(G, vOther, found)
```



Given a tie, visit edges are in alphabetical order

Running Time

$O(n)$, $O(nm)$, $O(m+n)$

What is the running time?

$O(n+m)$

```
FUNCTION DFS(G, start_vertex)
    found = {v: FALSE FOR v IN G.vertices}
    DFSRecursion(G, start_vertex, found)
    RETURN found

FUNCTION DFSRecursion(G, v, found)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSRecursion(G, vOther, found)
```

What are the lower and upper bounds on m?

$0 \le m \le \binom{n}{2} = \frac{n(n-1)}{2}$

What is the depth of the recursion tree?

$O(m)$, $O(n)$

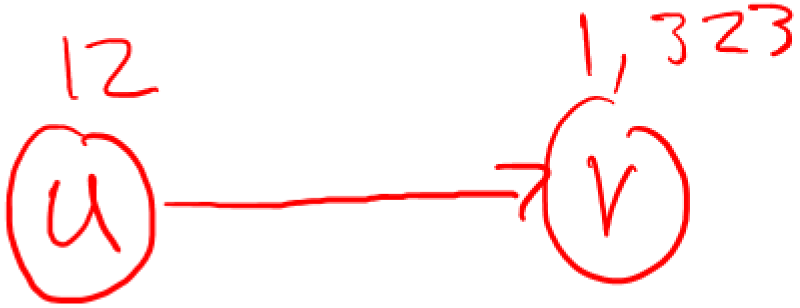
$n = |V|$, $m = |E|$

An example use case for DFS

Topological Orderings

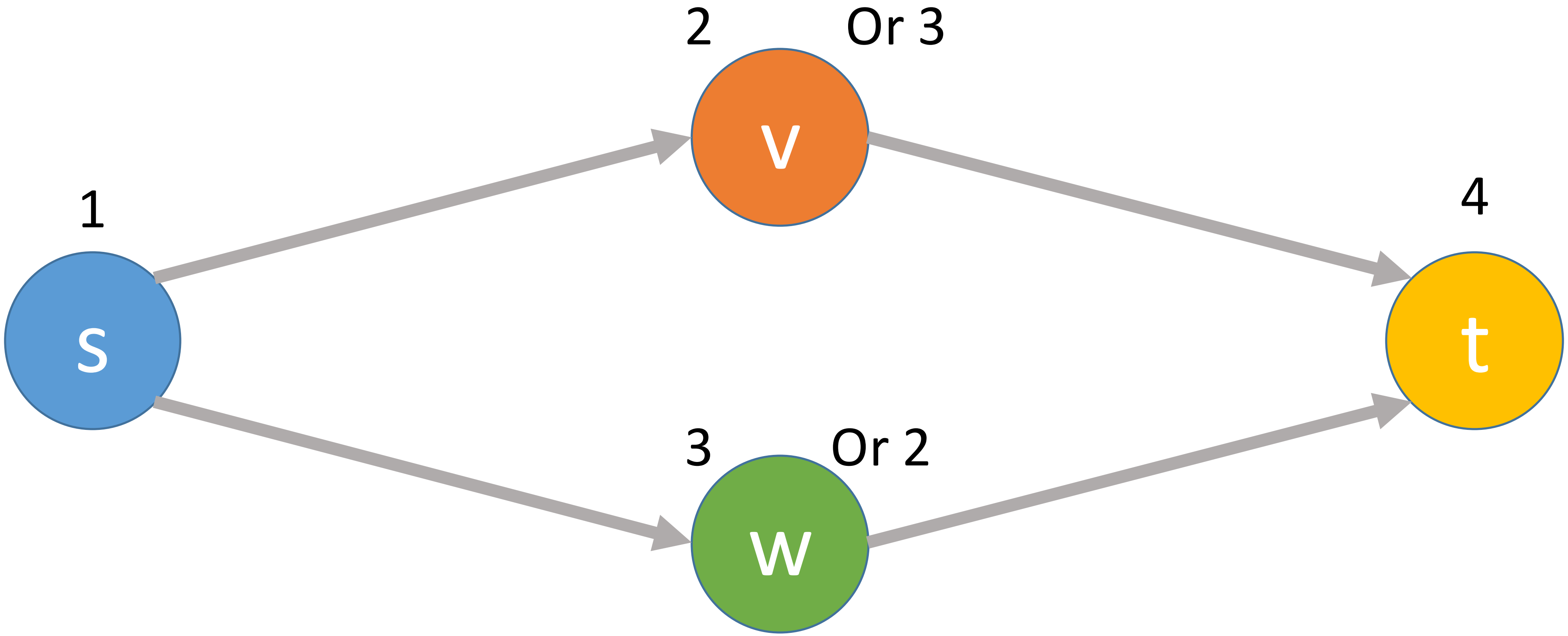
Definition: a topological ordering of a directed acyclic graph (DAG) is a labelling of the graph's vertices with "f-values" such that:

1. The f-values are of the set {1, 2, …, n}
2. For an edge (u, v) of G, $f(u) < f(v)$

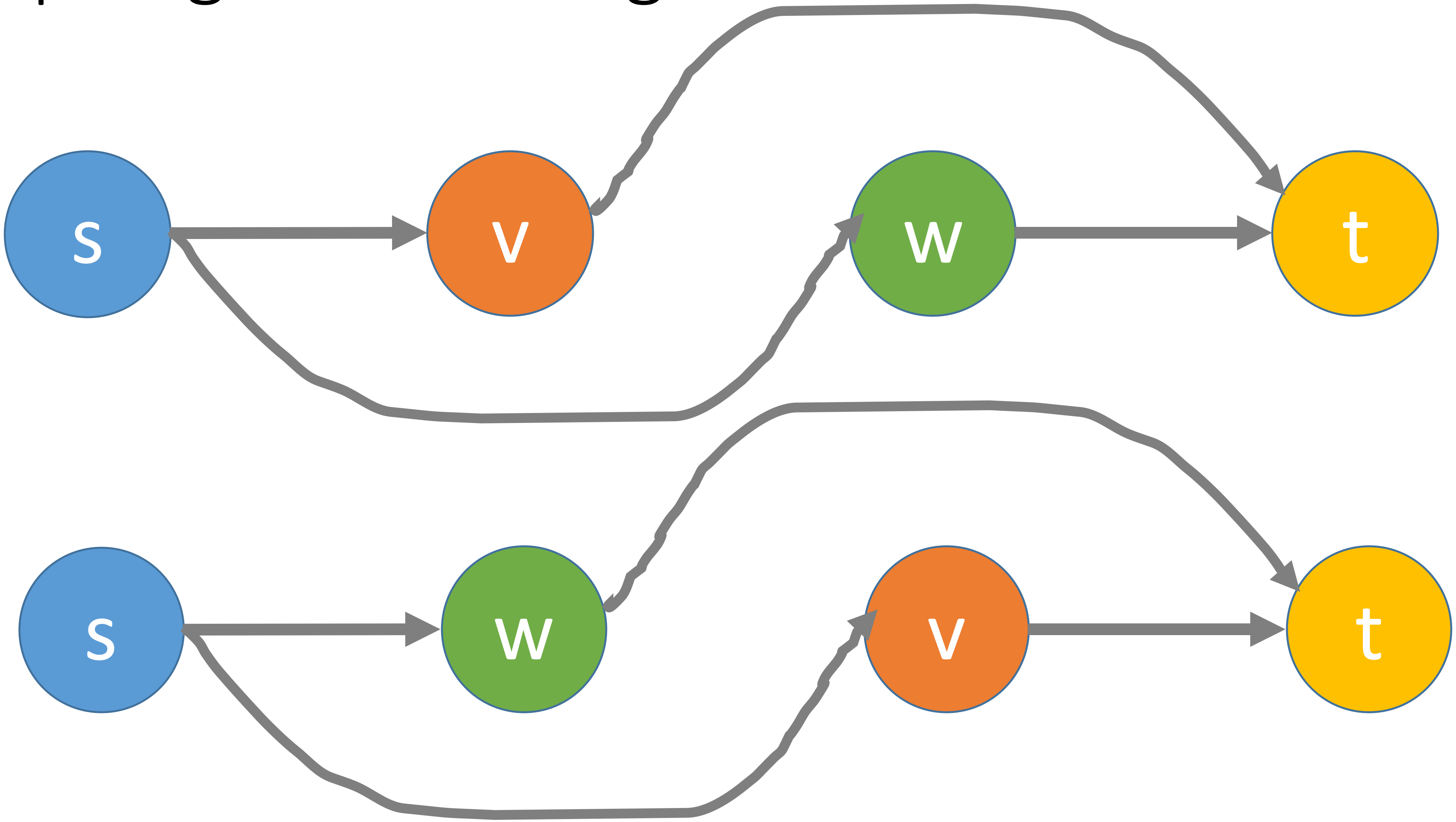


Topological Orderings

1. The f-values are of the set {1, 2, …, n}
2. For an edge (u, v) of G, $f(u) < f(v)$



Topological Orderings



Topological Orderings

Can be used to graph a sequence of tasks while respecting all precedence constraints

- For example, a flow chart for your CS degrees
- I read a funding proposal where they were using topological orderings to schedule robot tasks for building a space station.

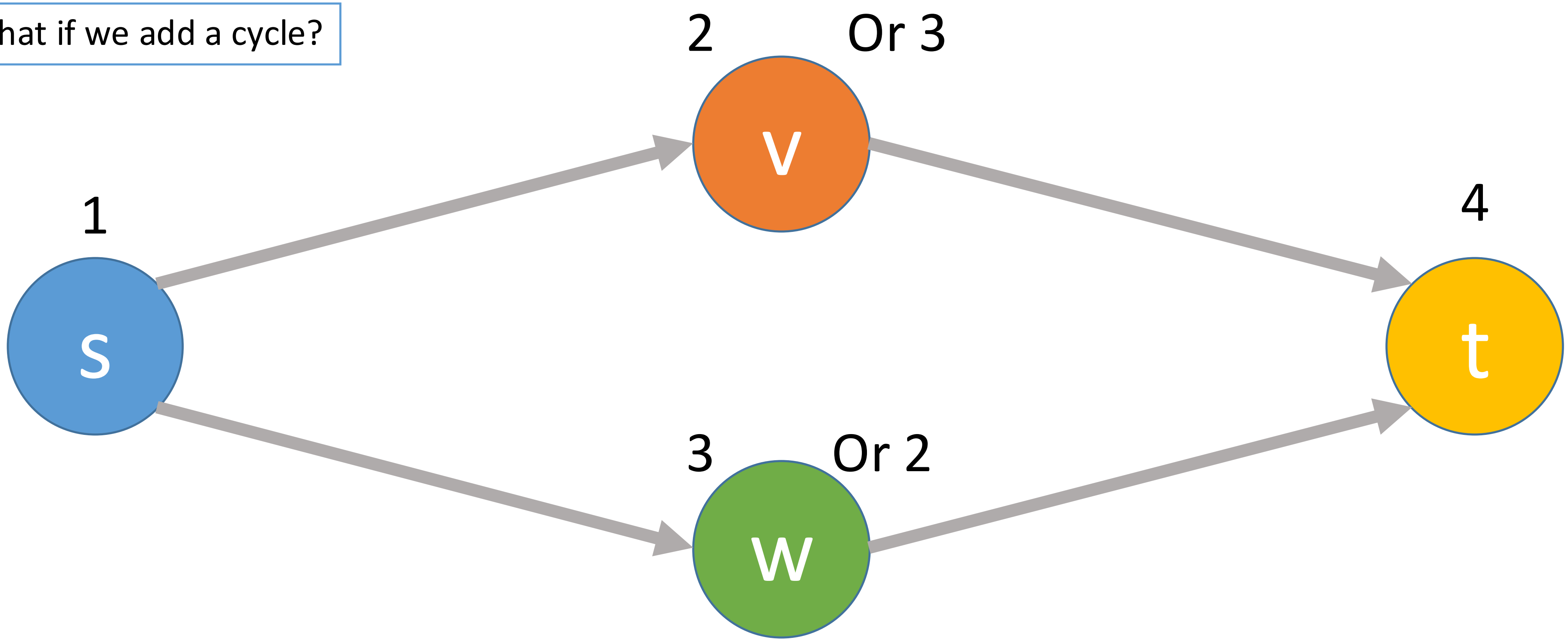
Requires the graph to be acyclic.

- Why?

Topological Orderings

1. The f-values are of the set {1, 2, …, n}
2. For an edge (u, v) of G, f(u) < f(v)

What if we add a cycle?



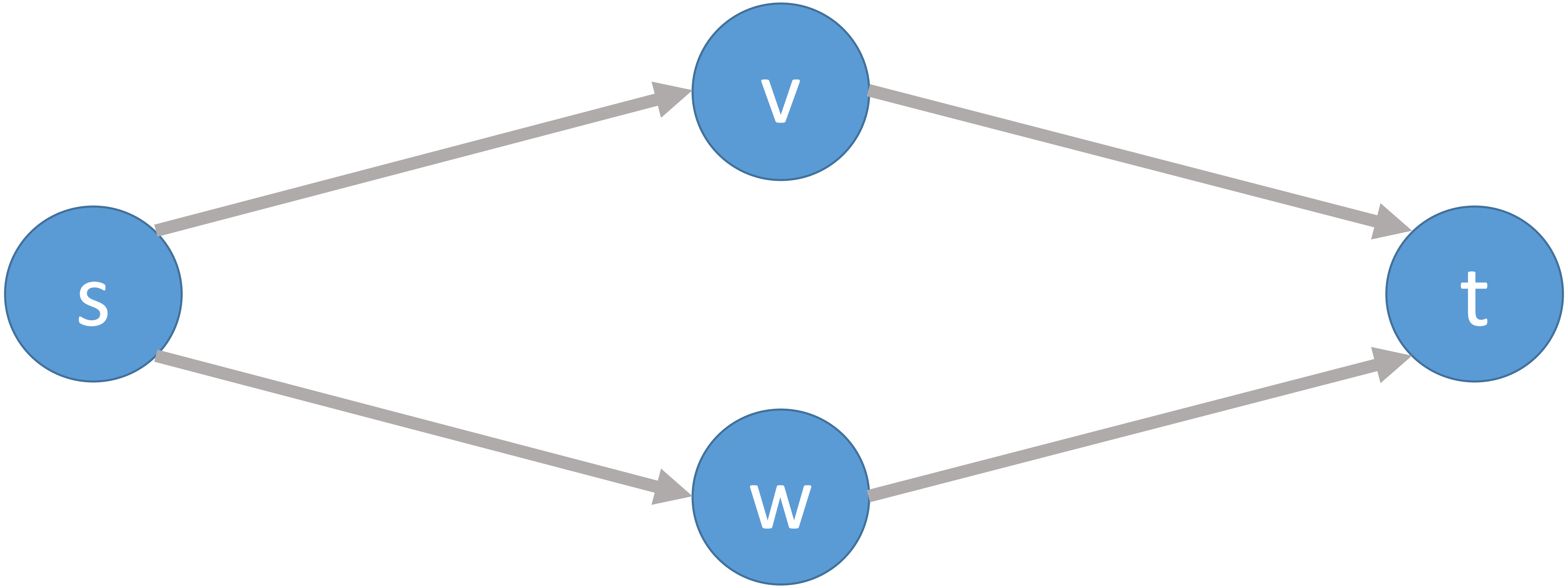
How to Compute Topological Orderings?

Straightforward solution:

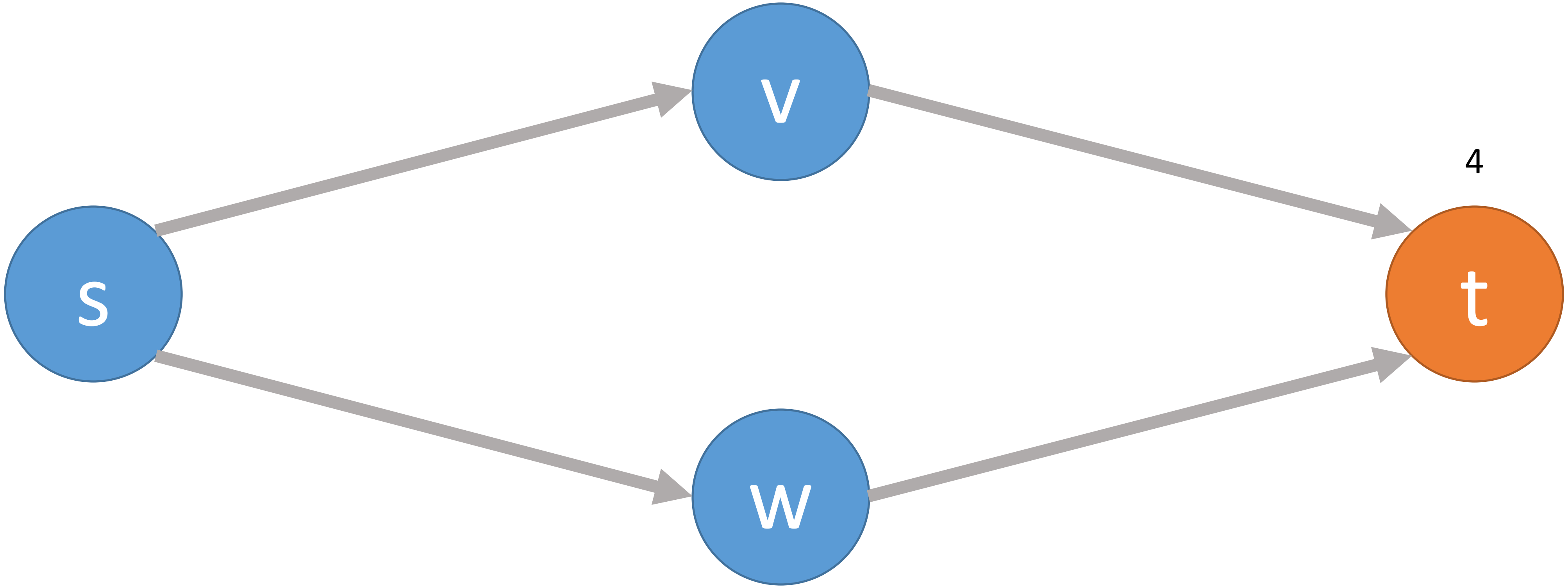
1. Let v be any sink of G
2. Set f(v) = |V| = n
3. Recursively conduct the same procedure on G – {v}

A sink is a vertex without any outgoing edges

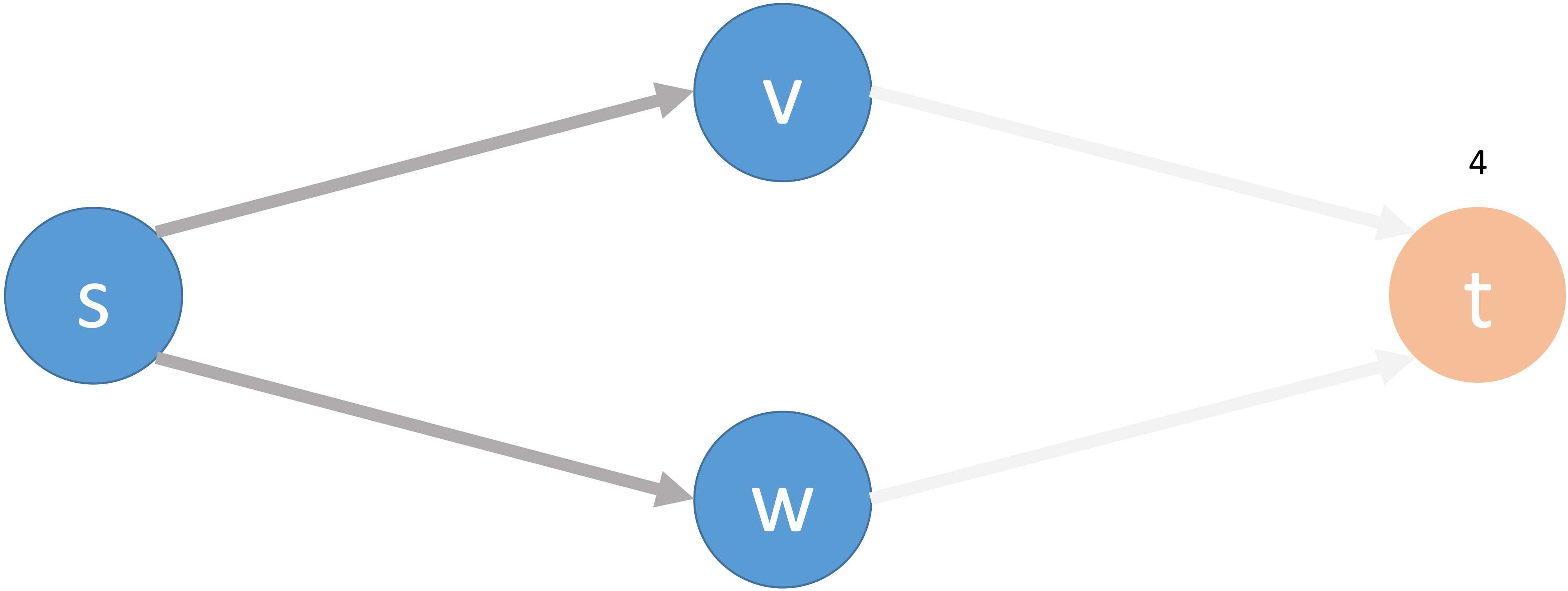
1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}



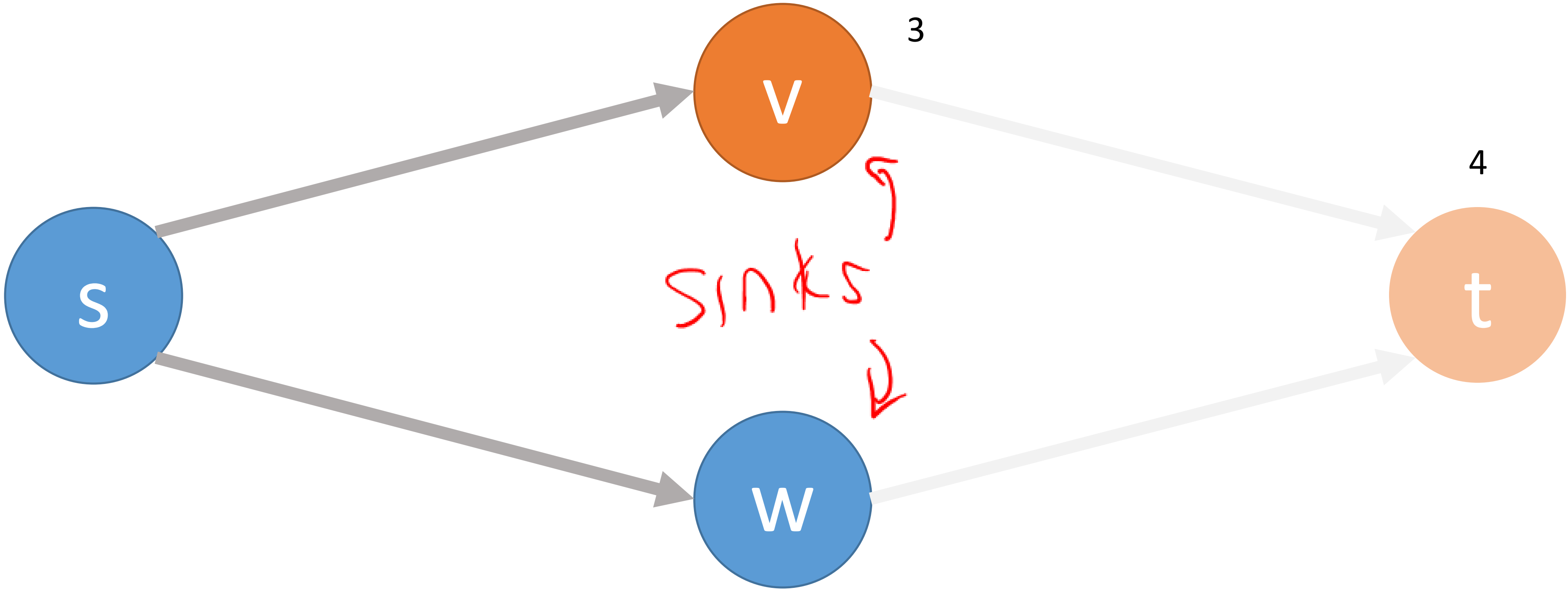
1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}



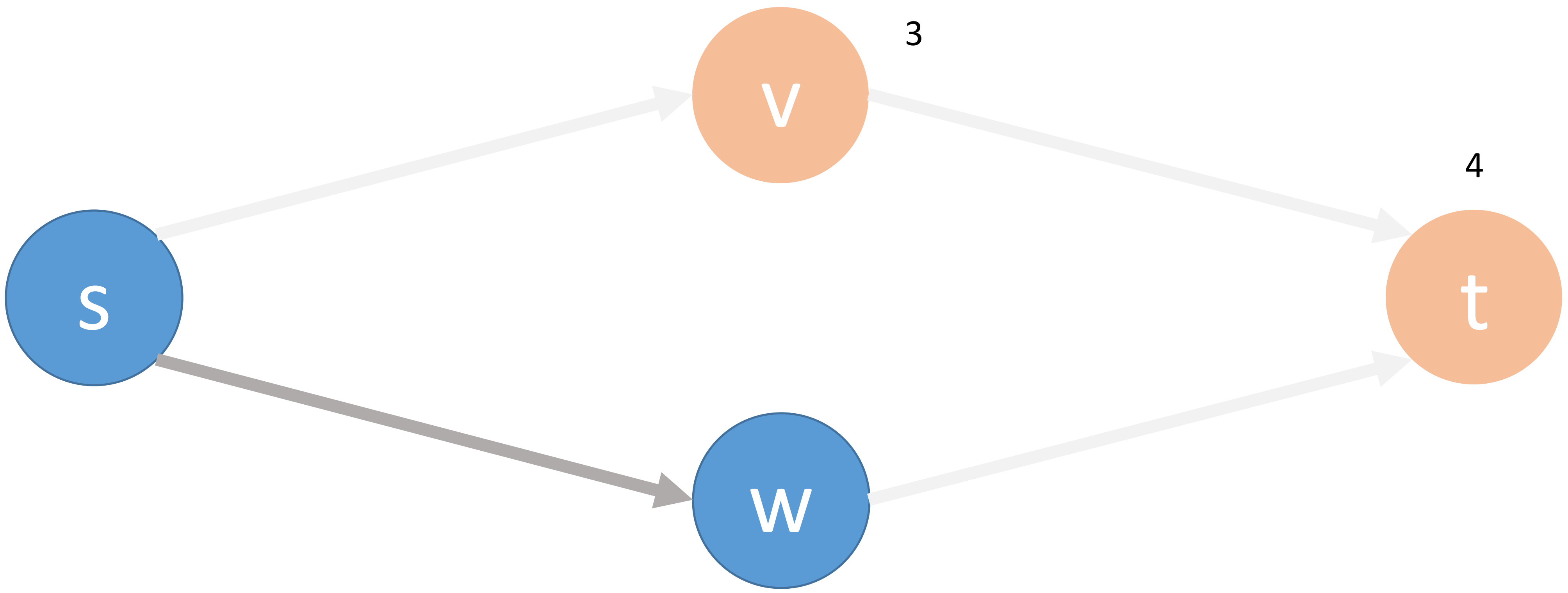
1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}



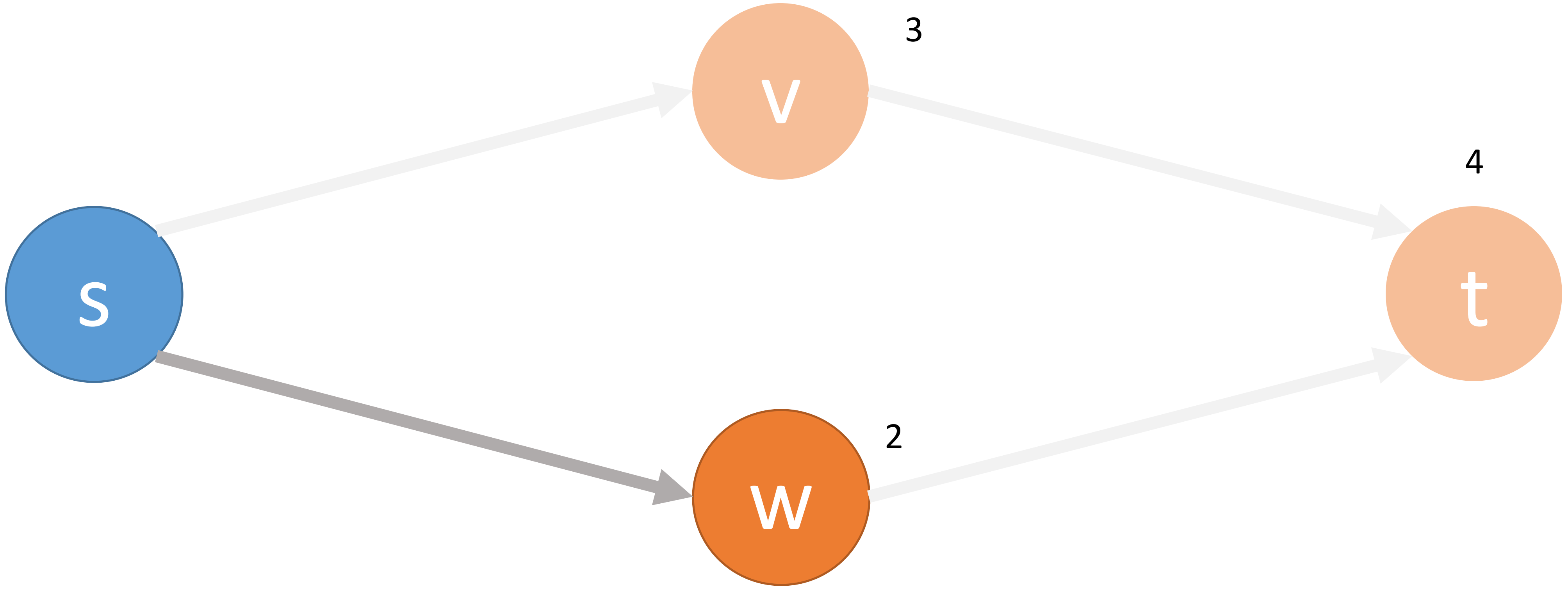
1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}



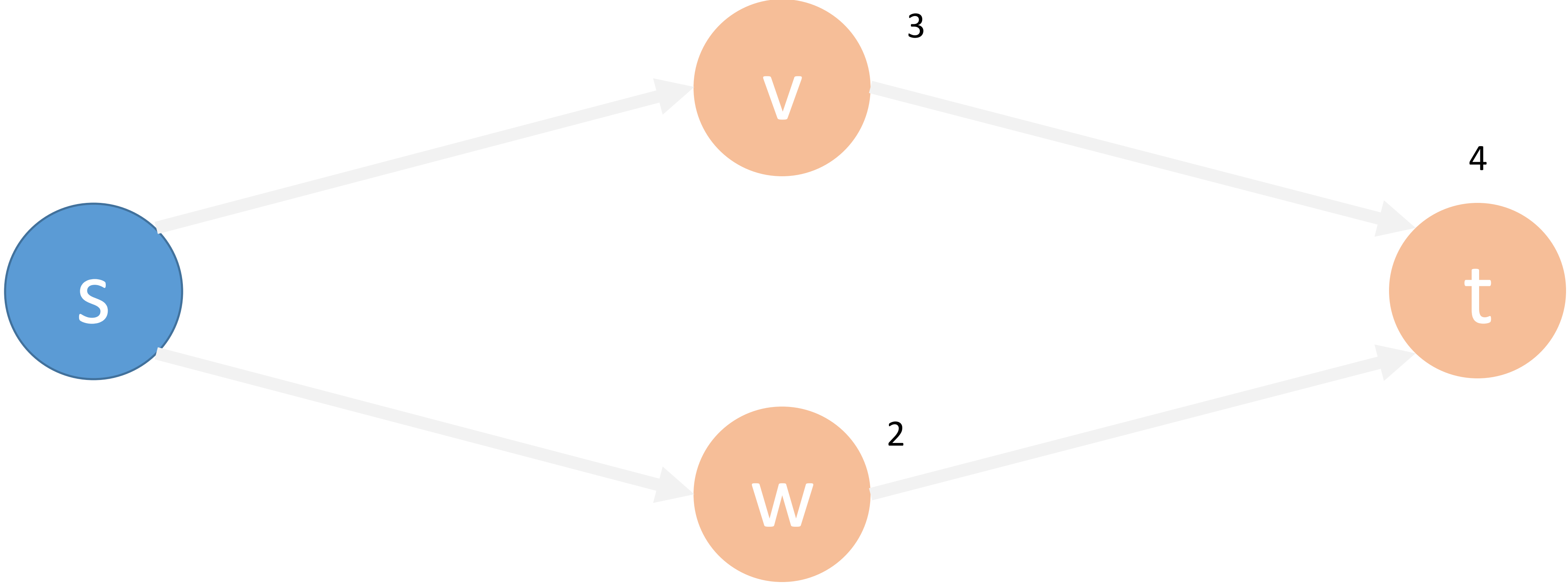
1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}



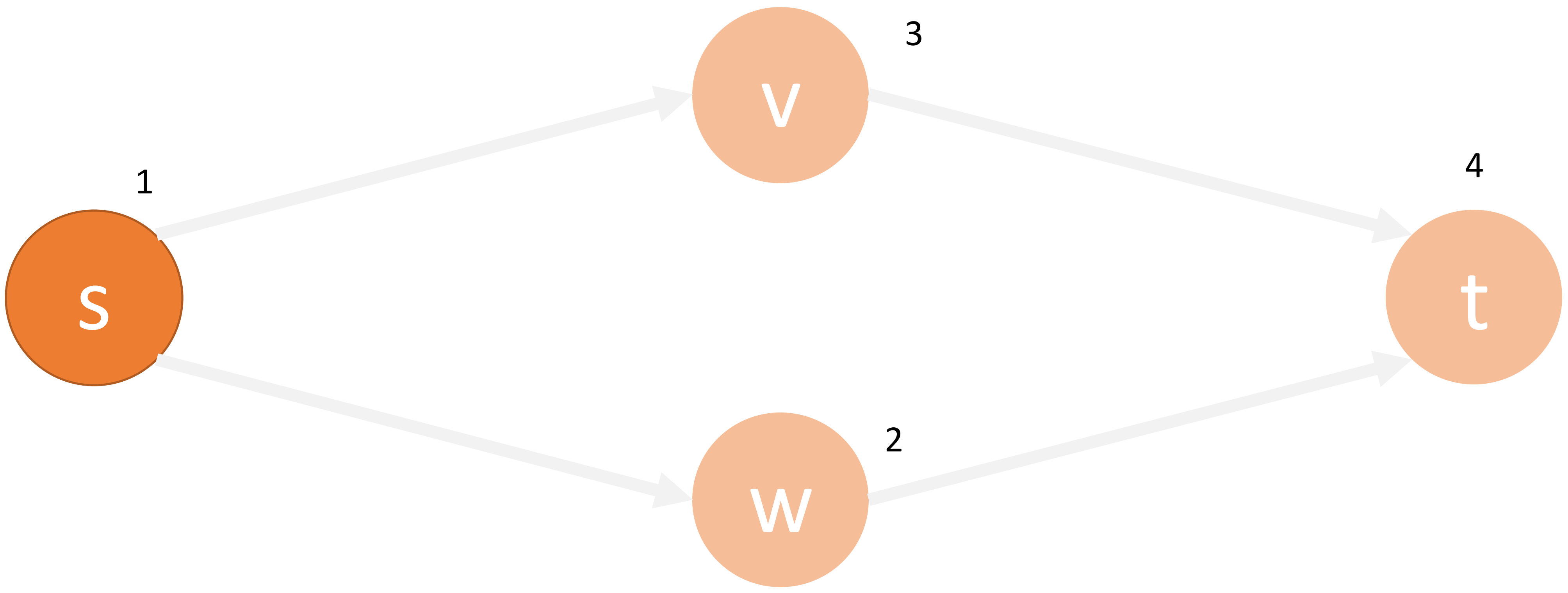
1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}



1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}



1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}

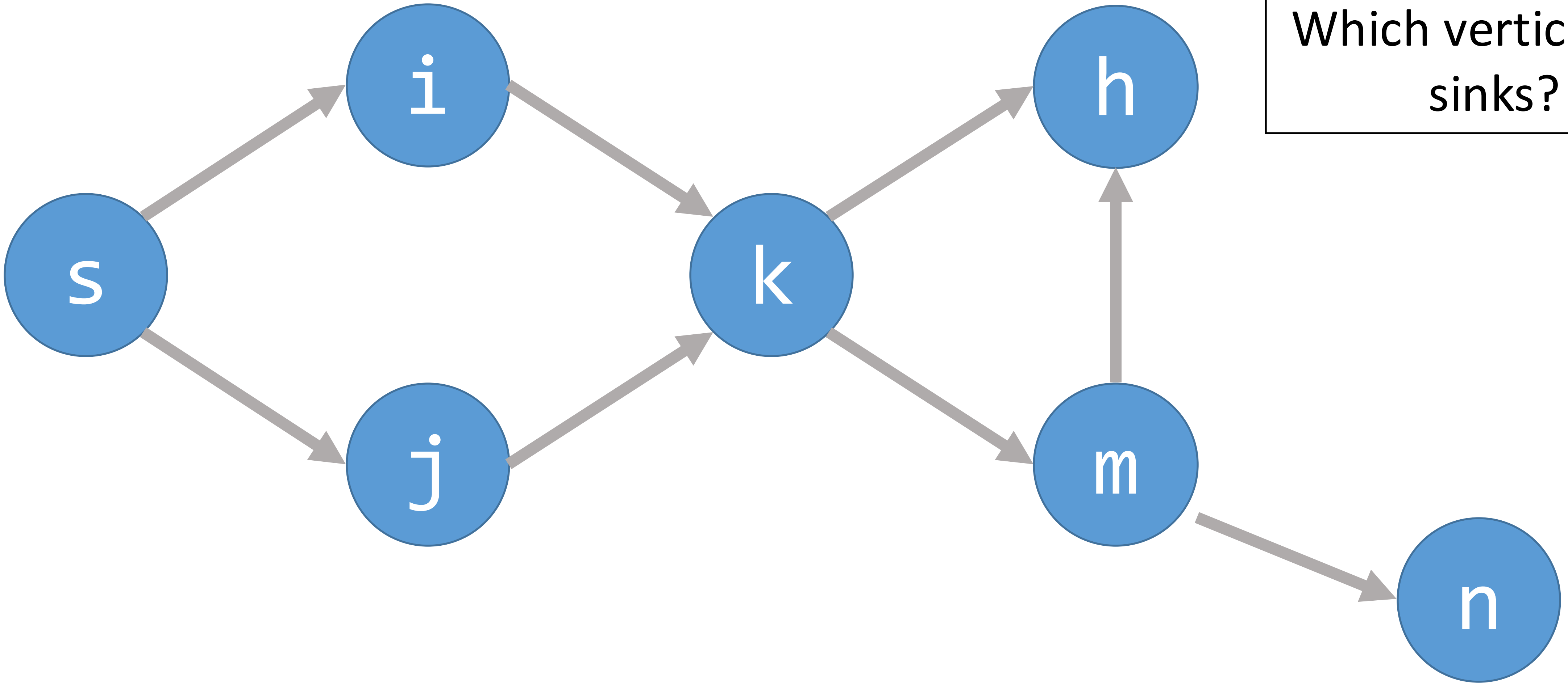


How to Compute Topological Orderings?

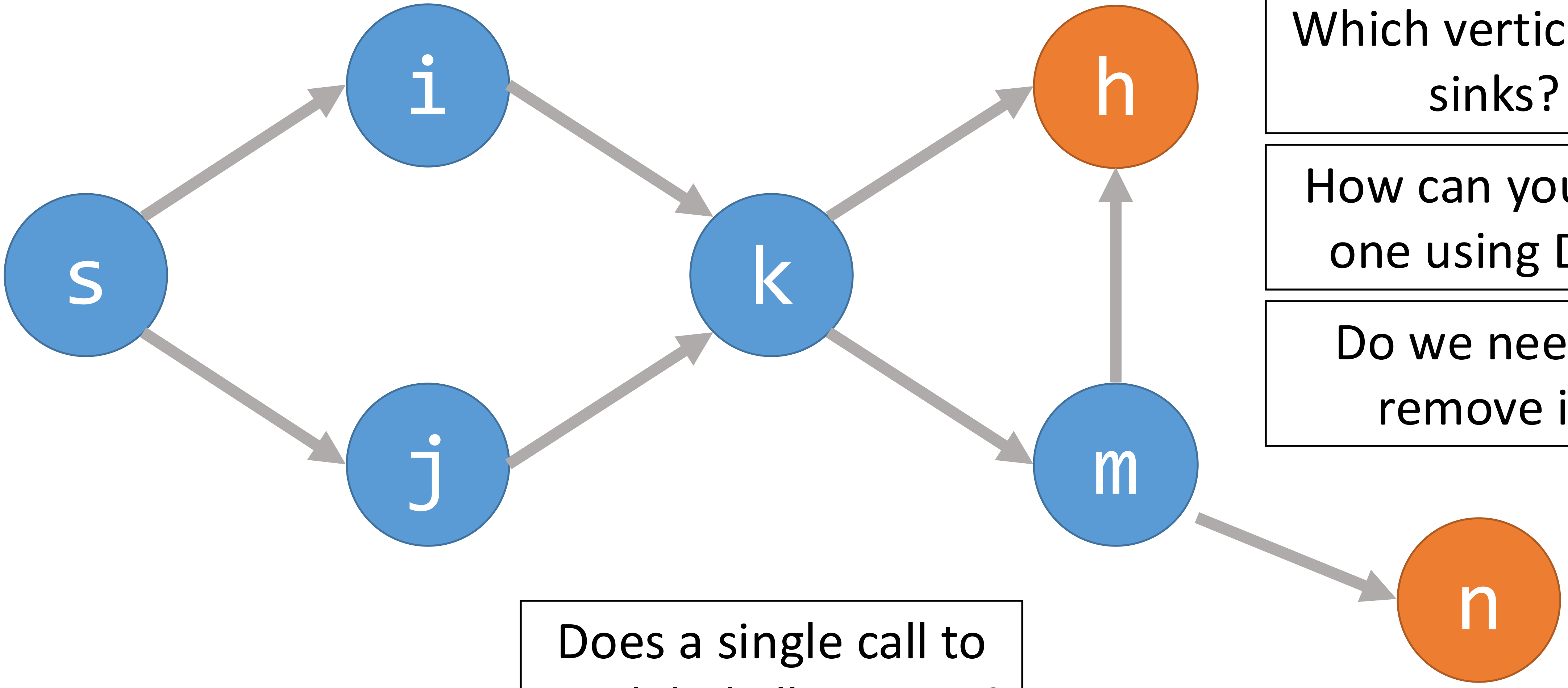
Straightforward solution:

1. Let v be any sink of G
2. Set f(v) = |V|
3. Recursively conduct the same procedure on G – {v}

How can we do this with our DFS algorithm if we don't know which vertices are sinks?



Which vertices are sinks?



Which vertices are sinks?

How can you find one using DFS?

Do we need to remove it?

Does a single call to DFS label all vertices?

```
FUNCTION DFS(G, start_vertex)
    found = {v: FALSE FOR v IN G.vertices}
    DFSRecursion(G, start_vertex, found)
    RETURN found


FUNCTION DFSRecursion(G, v, found)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSRecursion(G, vOther, found)
```

```
FUNCTION DFS(G, start_vertex)
    found = {v: FALSE FOR v IN G.vertices}

    fValues = {v: INFINITY FOR v IN G.vertices}

    f = G.vertices.length

    FOR v IN G.vertices

        IF found[v] == FALSE

            DFSRecursion(G, start_vertex, found)
    RETURN found


FUNCTION DFSRecursion(G, v, found)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSRecursion(G, vOther, found)

    fValues[v] = f

    f = f - 1
```

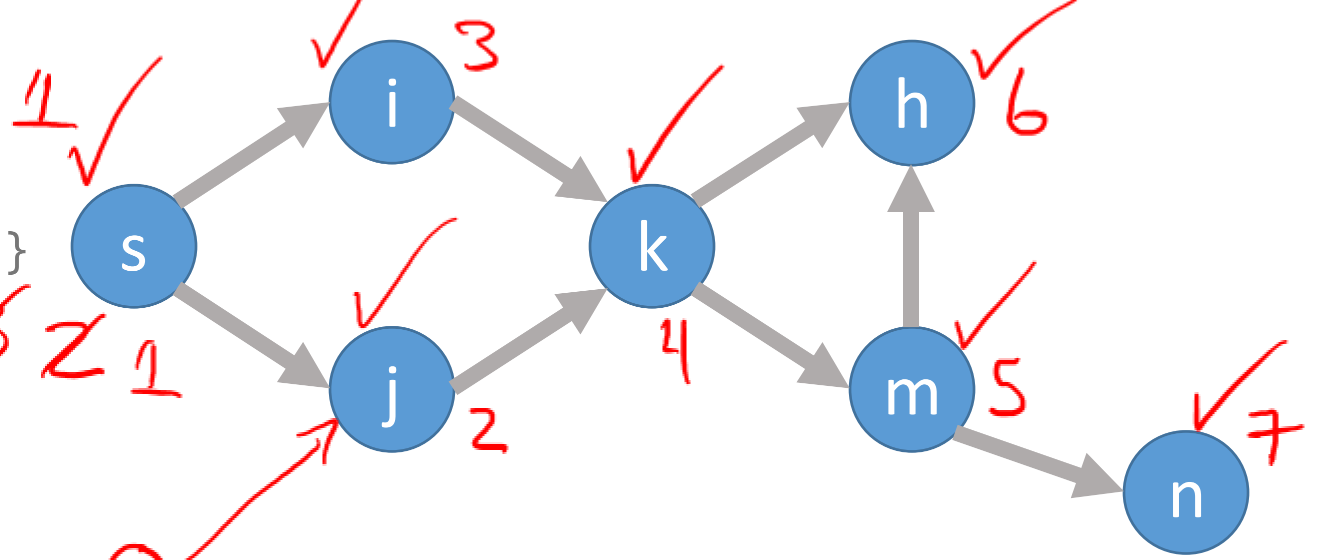
Topological Ordering with DFS

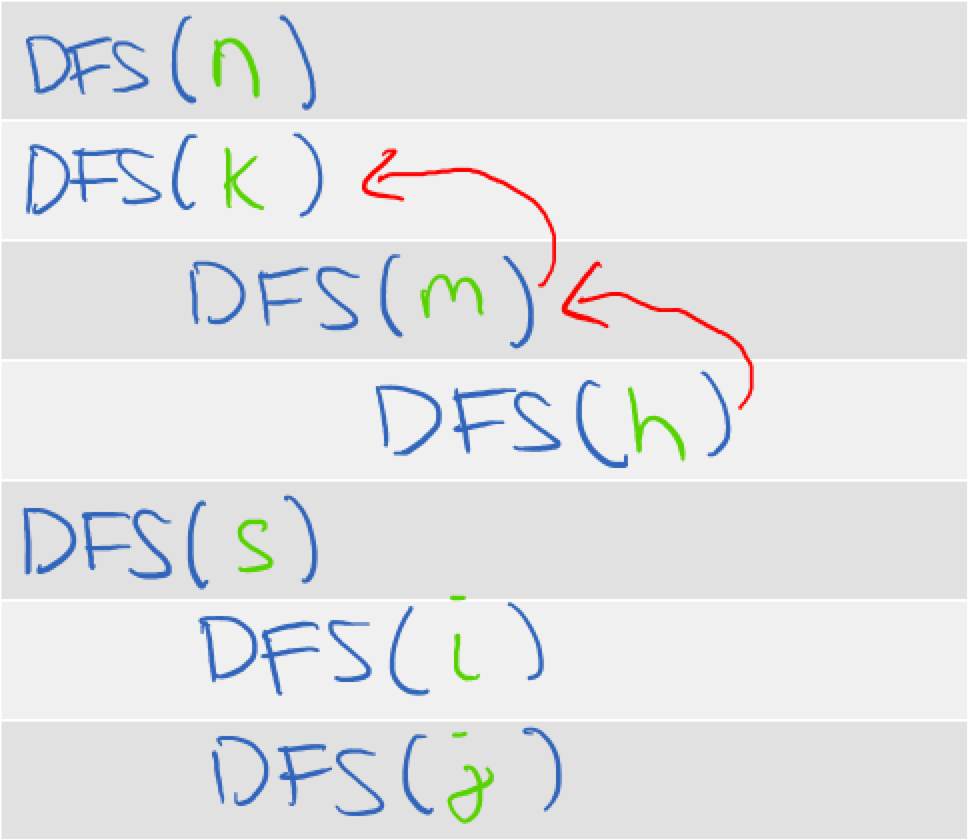
```
FUNCTION TopologicalOrdering(G)
    found = {v: FALSE FOR v IN G.vertices}
    fValues = {v: INFINITY FOR v IN G.vertices}
    f = G.vertices.length
    FOR v IN G.vertices
        IF found[v] == FALSE
            DFSTopological(G, v, found, f, fValues)
    RETURN fValues
```

```
FUNCTION DFSTopological(G, v, found, f, fValues)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSTopological(G, vOther, found, f, fValues)
    fValues[v] = f
    f = f - 1
```

```
FUNCTION TopologicalOrdering(G)
    found = {v: FALSE FOR v IN G.vertices}
    fValues = {v: INFINITY FOR v IN G.vertices}
    f = G.vertices.length
    FOR v IN G.vertices
        IF found[v] == FALSE
            DFSTopological(G, v, found, f, fValues)
    RETURN fValues


FUNCTION DFSTopological(G, v, found, f, fValues)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSTopological(G, vOther, found, f, fValues)
    fValues[v] = f
    f = f - 1
```





Running Time

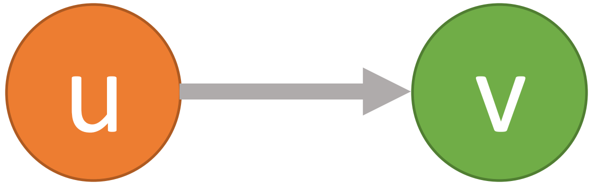
Again, this algorithm is O(n + m)

We only consider each vertex once, and

We only consider each edge once (twice if you consider backtracking)

Correctness of DFS Topological Ordering

We need to show that for any (u, v) that f(u) < f(v)



1. Consider the case when u is visited first
   1. We recursively look at all paths from u and **label those vertices first**
   2. So, f(u) must be less than f(v)

2. Now consider the case when v is visited first
   1. There is **no path back** to u, so v gets labeled before we explore u
   2. Thus, f(u) must be less than f(v)

```
FUNCTION DFSTopological(G, v, found,
                        f, fValues)
  found[v] = TRUE
  FOR vOther IN G.edges[v]
    IF found[vOther] == FALSE
      DFSTopological(G, vOther,
        found, f, fValues)
  fValues[v] = f
  f = f - 1
```

How do we know that there is no path from v to u?

Topological Ordering

- We can use DFS to find a topological ordering since a DFS will search as far as it can until it needs to backtrack

- It only needs to backtrack when it finds a sink

- Sinks are the first values that must be labeled