Depth-First Search (DFS) for Both Directed and Undirected Graphs

```
dfs(G = (V, E)):
    for each vertex u in V:
        u.color = white
        u.discoverer = None
    time = 0
    for each vertex u in V:
        if u.color == white:
            dfs_visit(u)
```

```
dfs_visit(u):
    time = time + 1
    u.start_time = time
    u.color = gray
    for each v in G.adj[u]:
        if v.color == white:
            v.discoverer = u
            dfs_visit(v)
    u.color = black
    time = time + 1
    u.finish_time = time
```

This is the pseudo code.

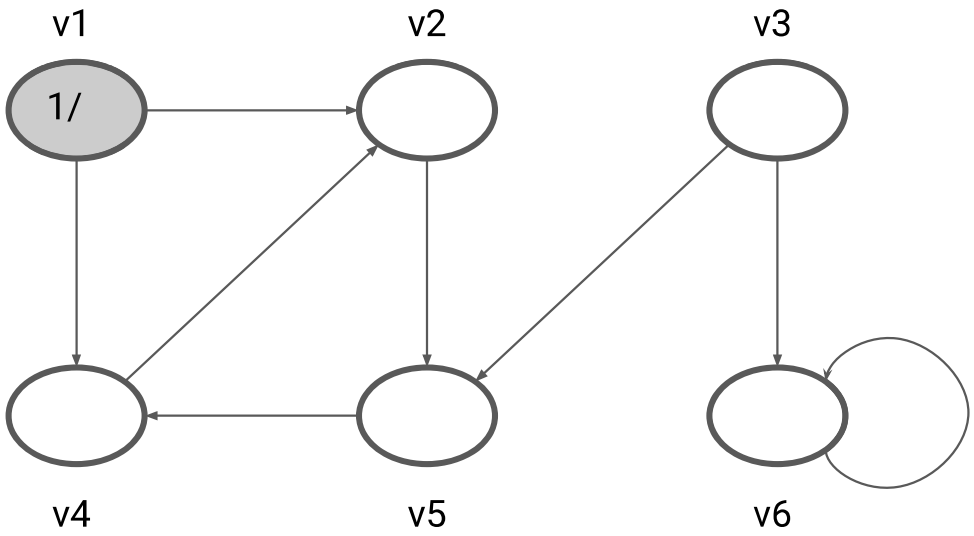
- white means undiscovered
- gray means being processed
- black means finished processing

G.adj[**u**] is the set of vertices adjacent **u**.

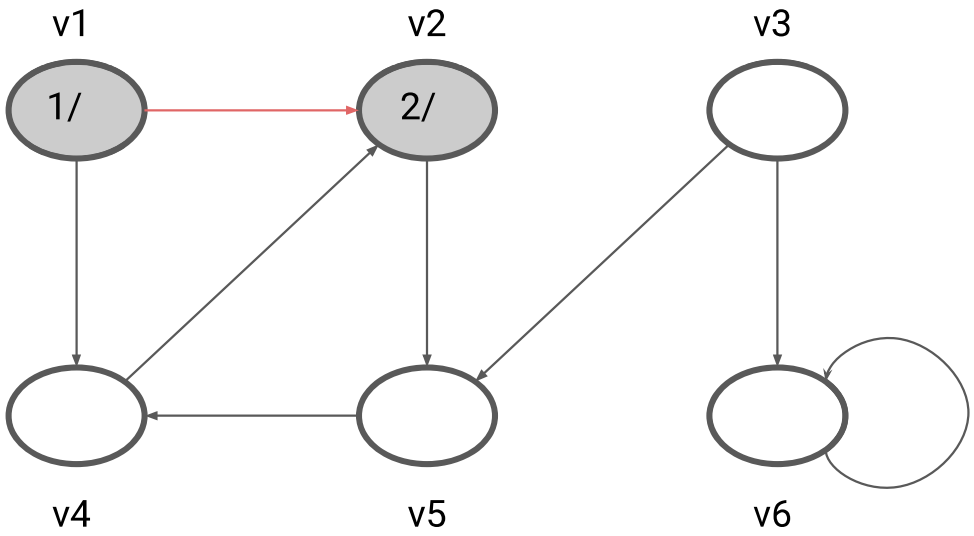
However, a stack should be used in Python to implement DFS due to the interpreter's limit on recursion depth.

DFS Example on a Directed Graph

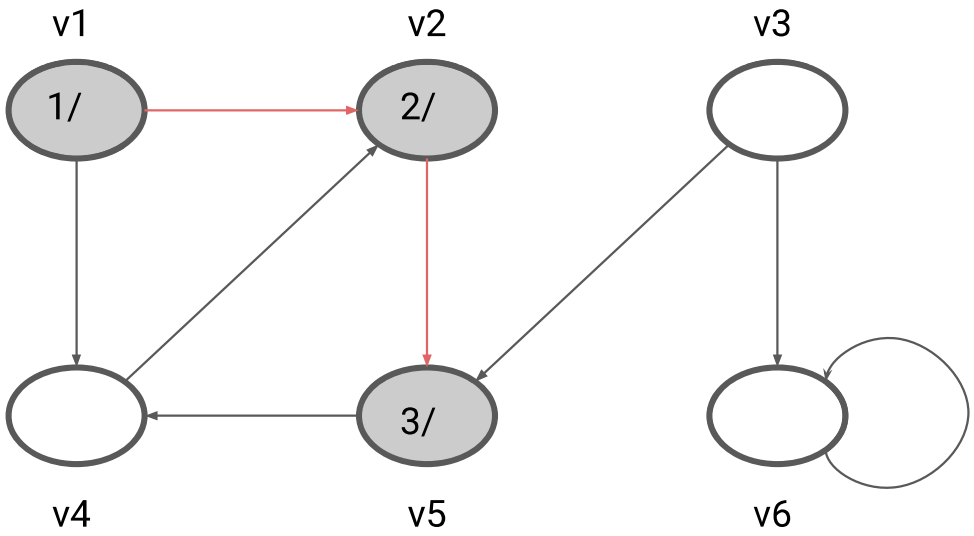
DFS(G)



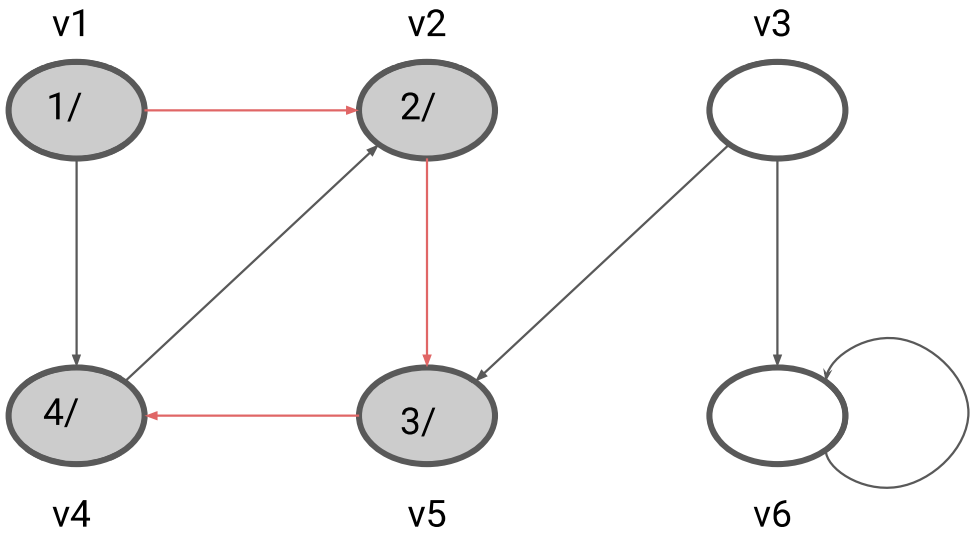
DFS Example on a Directed Graph



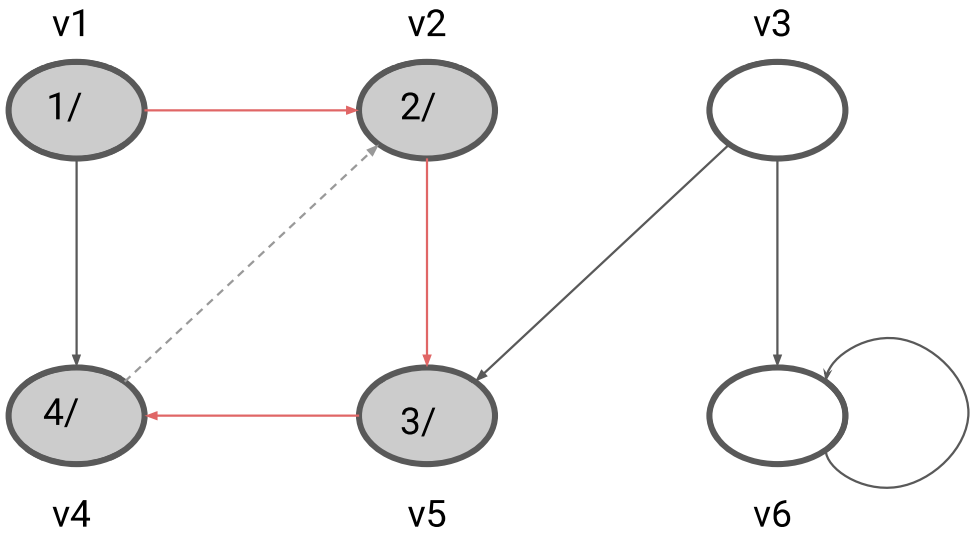
DFS Example on a Directed Graph



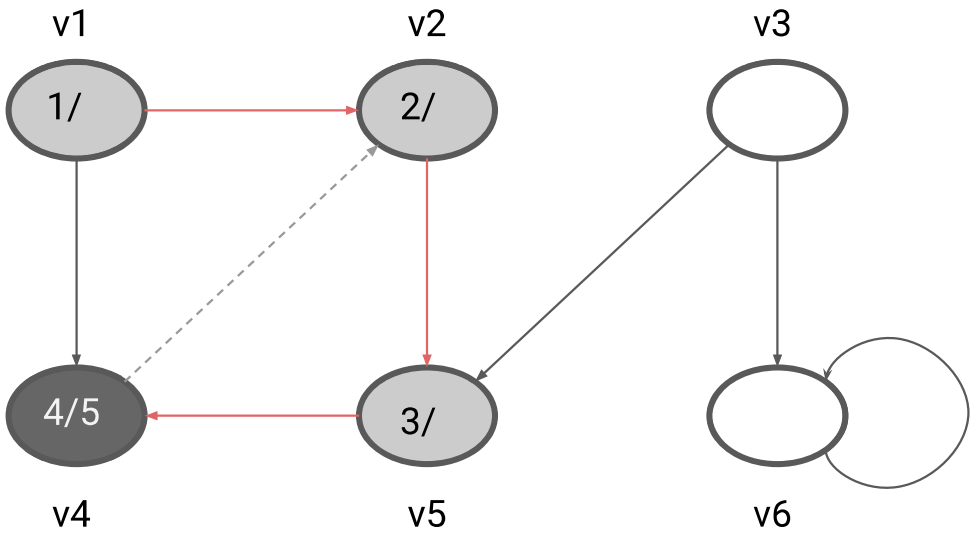
DFS Example on a Directed Graph



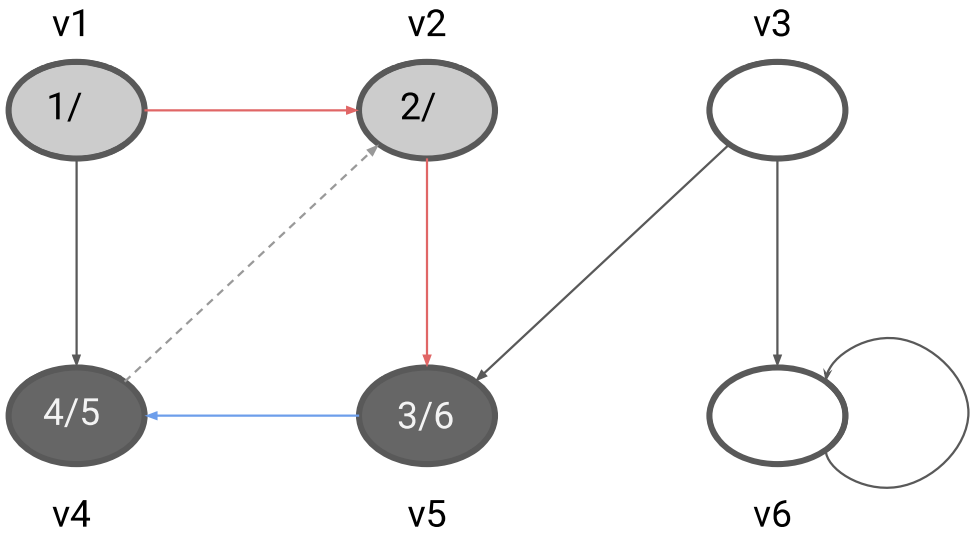
DFS Example on a Directed Graph



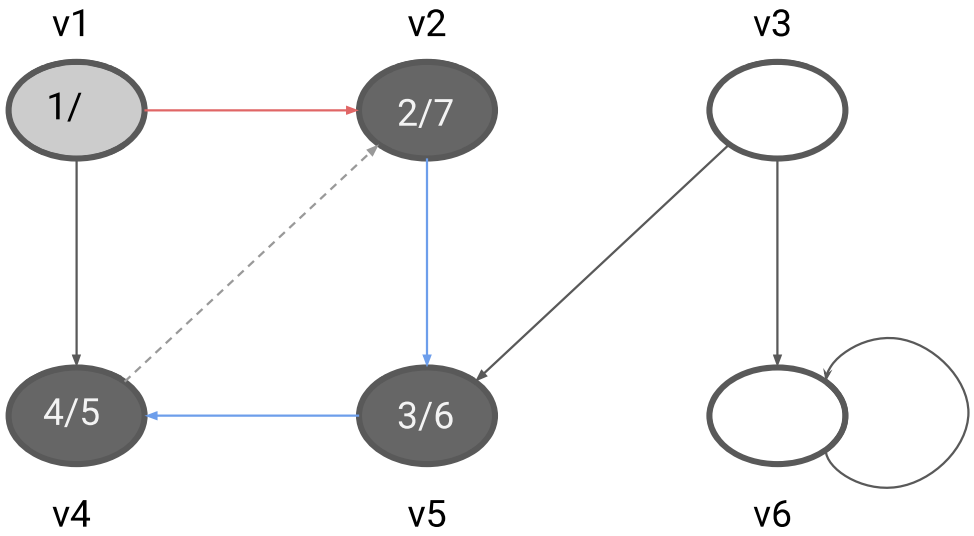
DFS Example on a Directed Graph



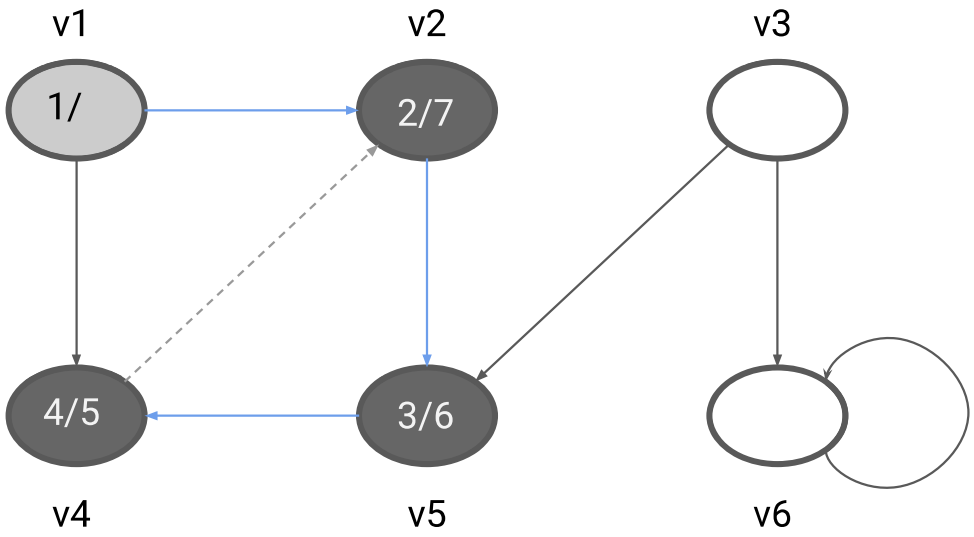
DFS Example on a Directed Graph



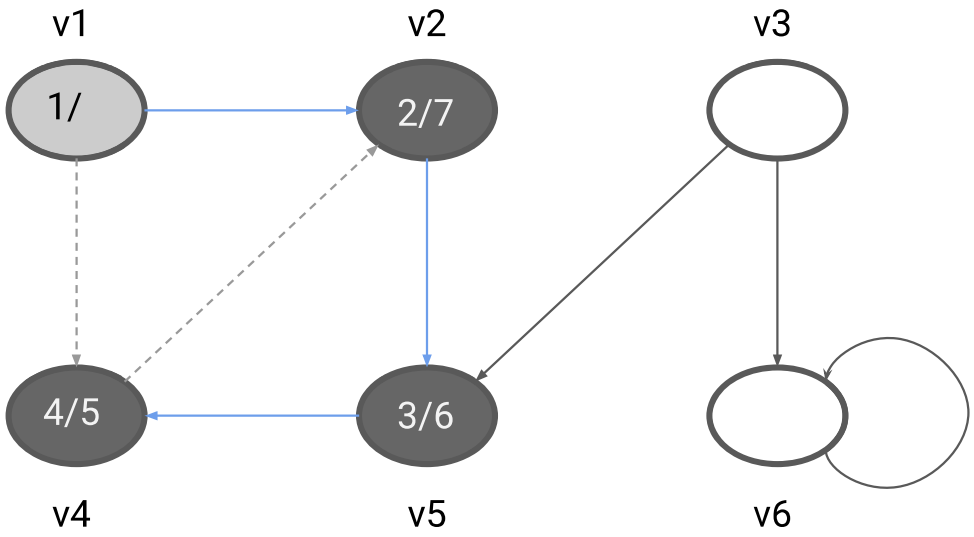
DFS Example on a Directed Graph



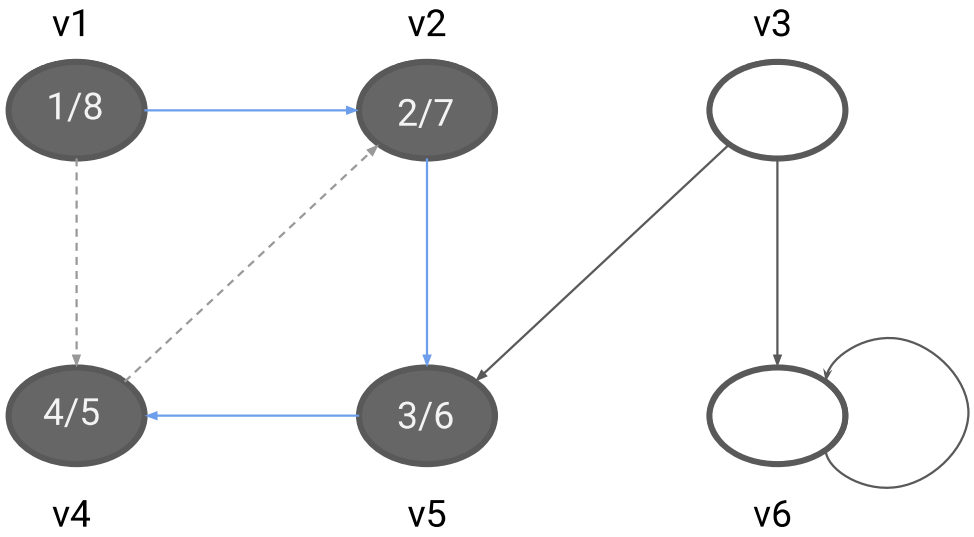
DFS Example on a Directed Graph



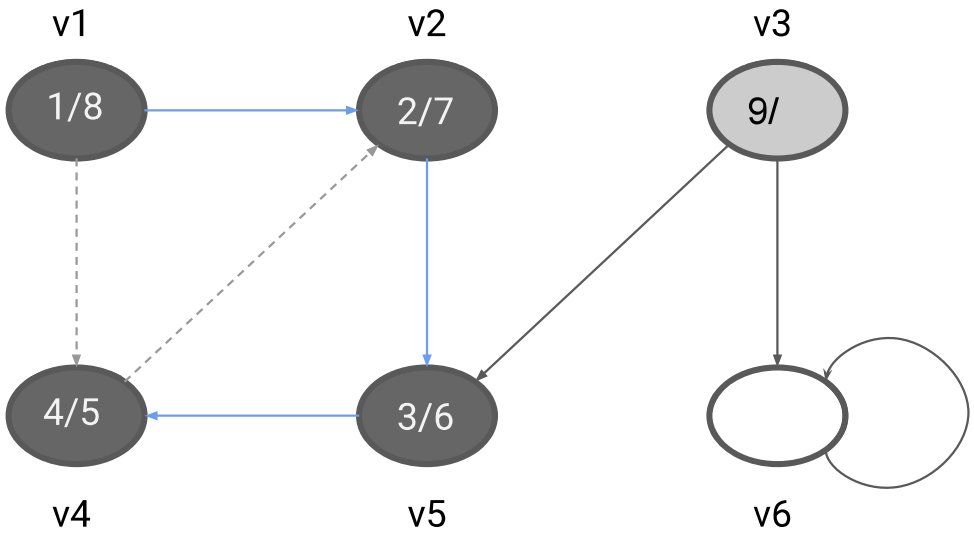
DFS Example on a Directed Graph



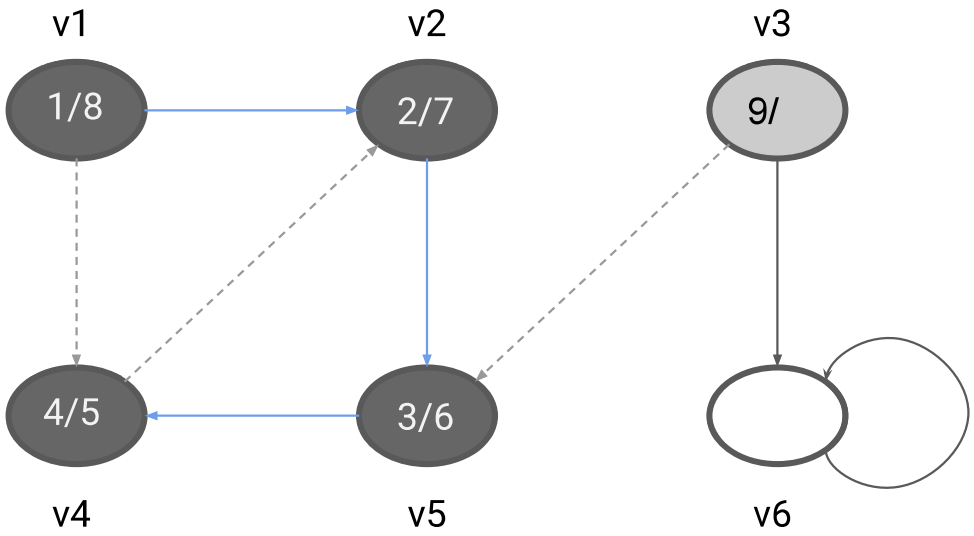
DFS Example on a Directed Graph



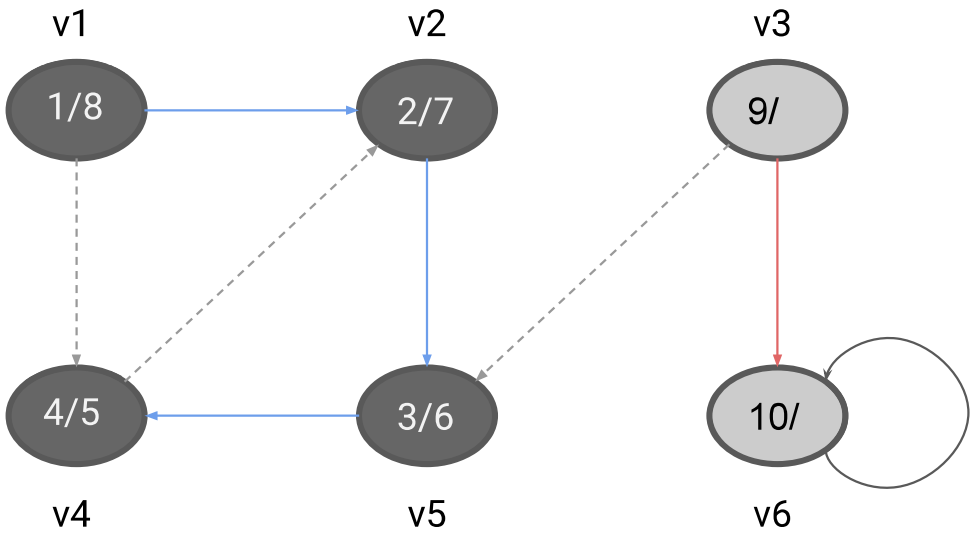
DFS Example on a Directed Graph



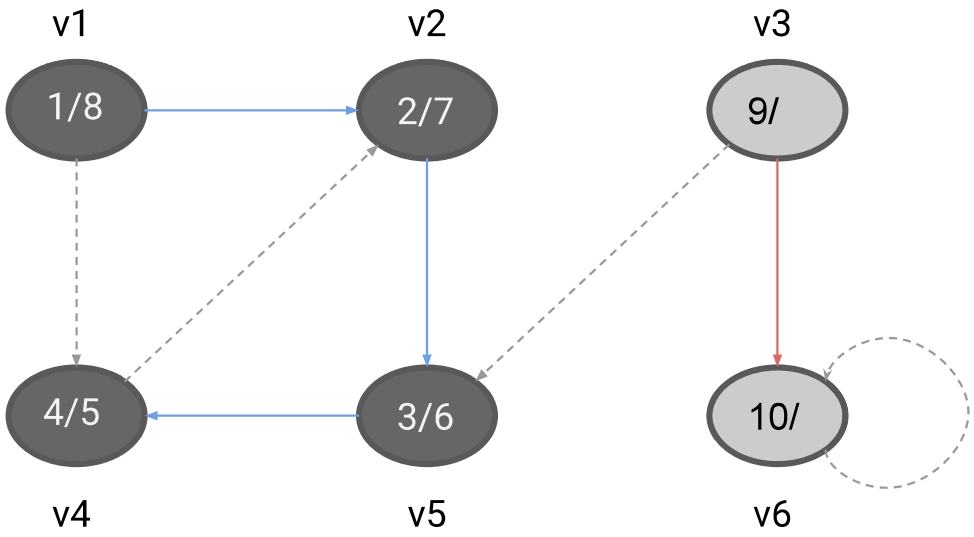
DFS Example on a Directed Graph



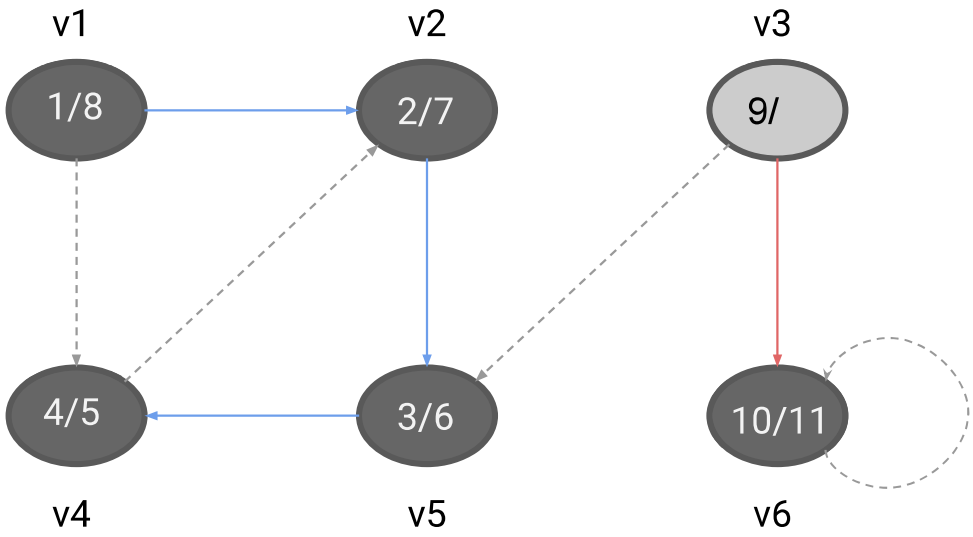
DFS Example on a Directed Graph



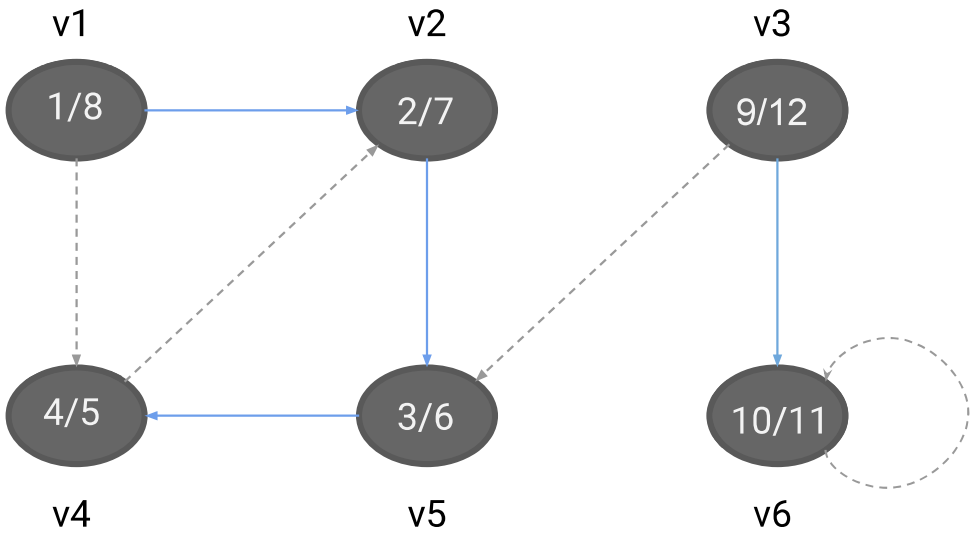
DFS Example on a Directed Graph



DFS Example on a Directed Graph



DFS Example on a Directed Graph

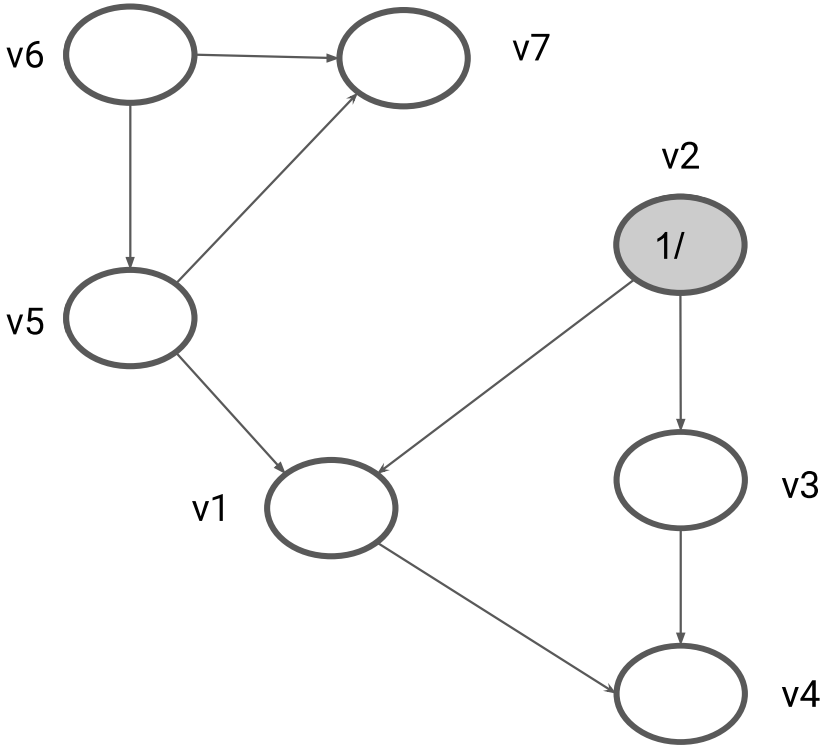


Topological Sort

One way to perform a topological sort on a DAG G is to run DFS on G, and as each vertex is finished being processed, i.e. turned black, append it onto a FIFO queue.

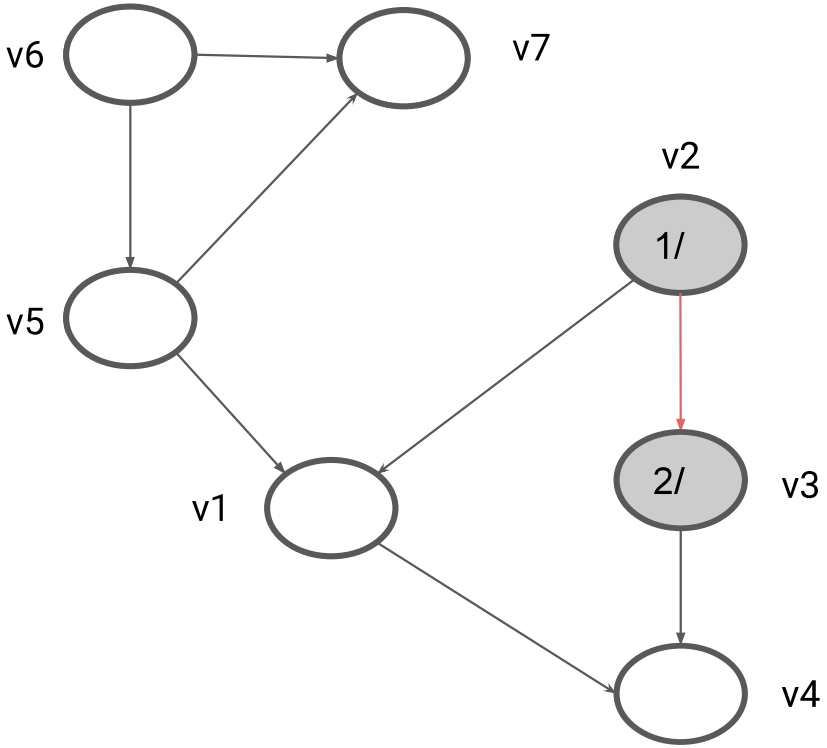
The resulting queue will contain a topological ordering of the vertices, where the first out element is the smallest and so on.

Topological Sort Example



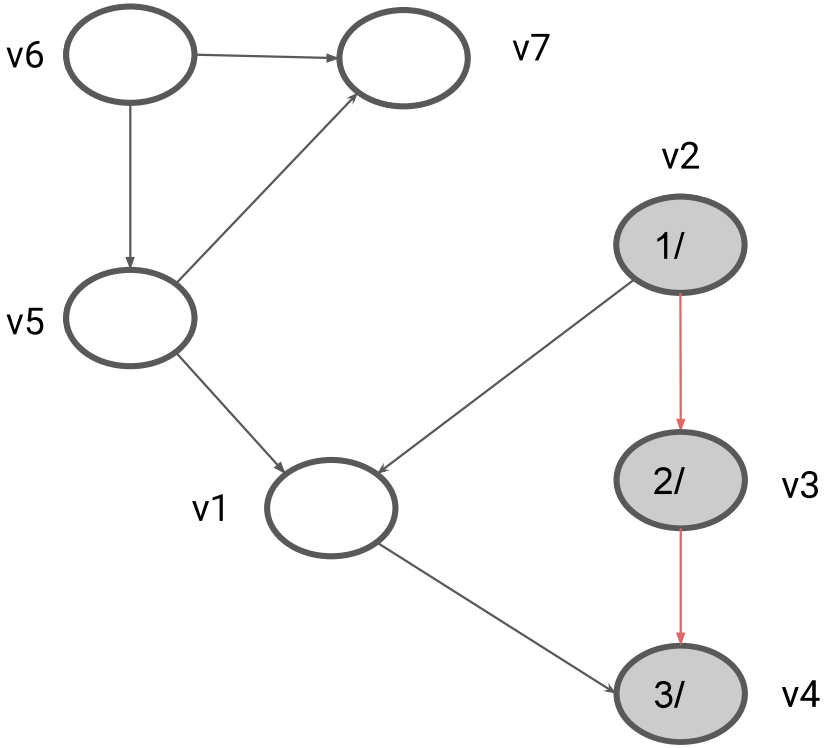
topologically ordered vertices:
[ ]

Topological Sort Example



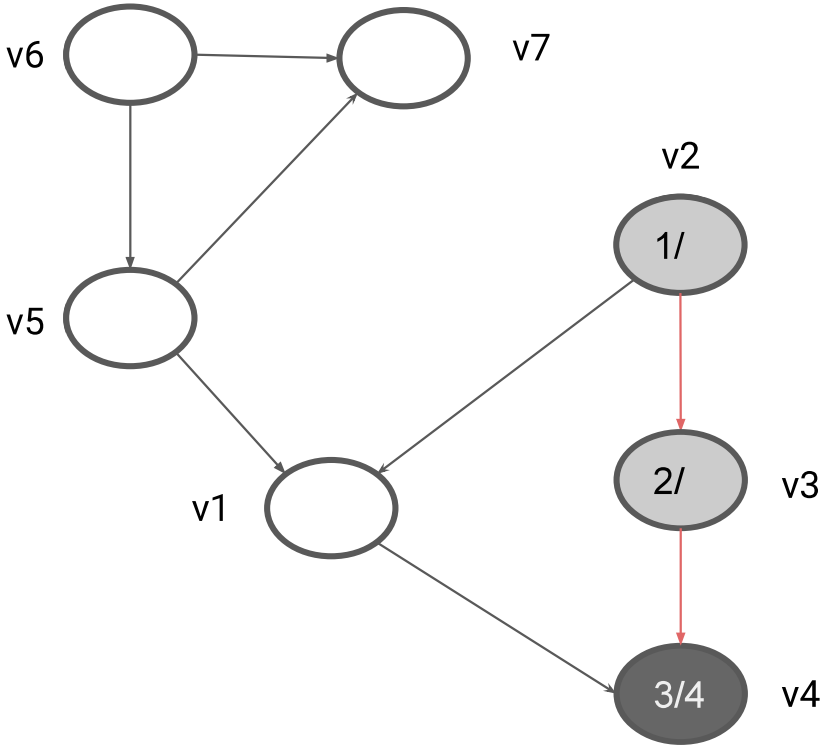
topologically ordered vertices:
[ ]

Topological Sort Example



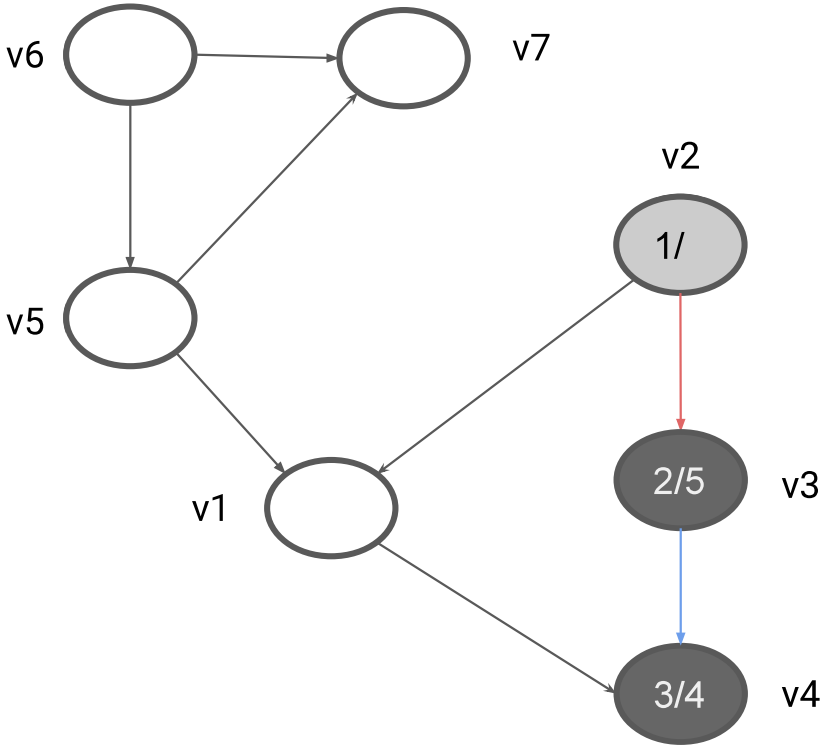
topologically ordered vertices:
[ ]

Topological Sort Example



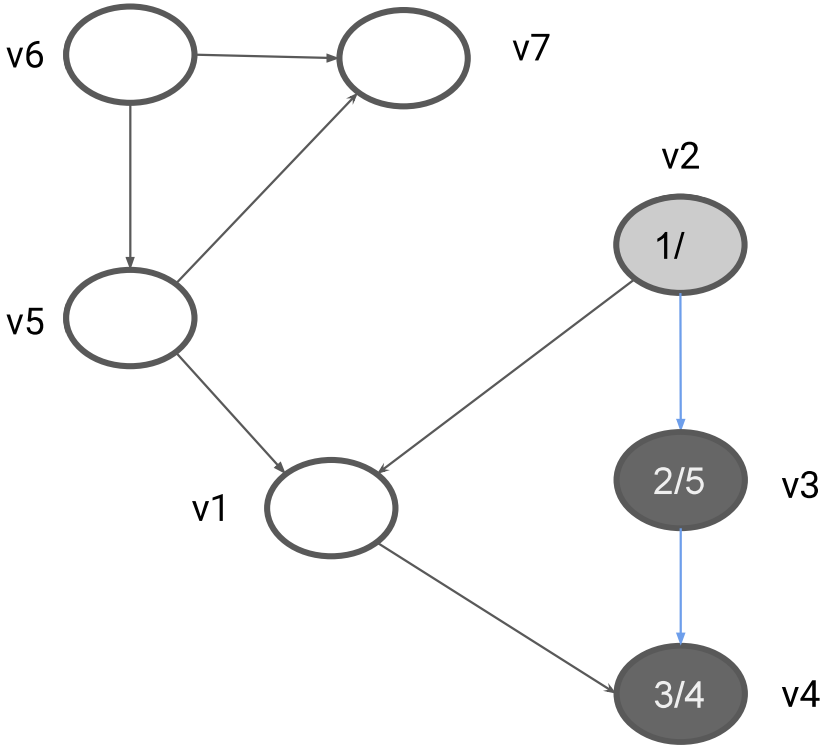
topologically ordered vertices:
[v4]

Topological Sort Example



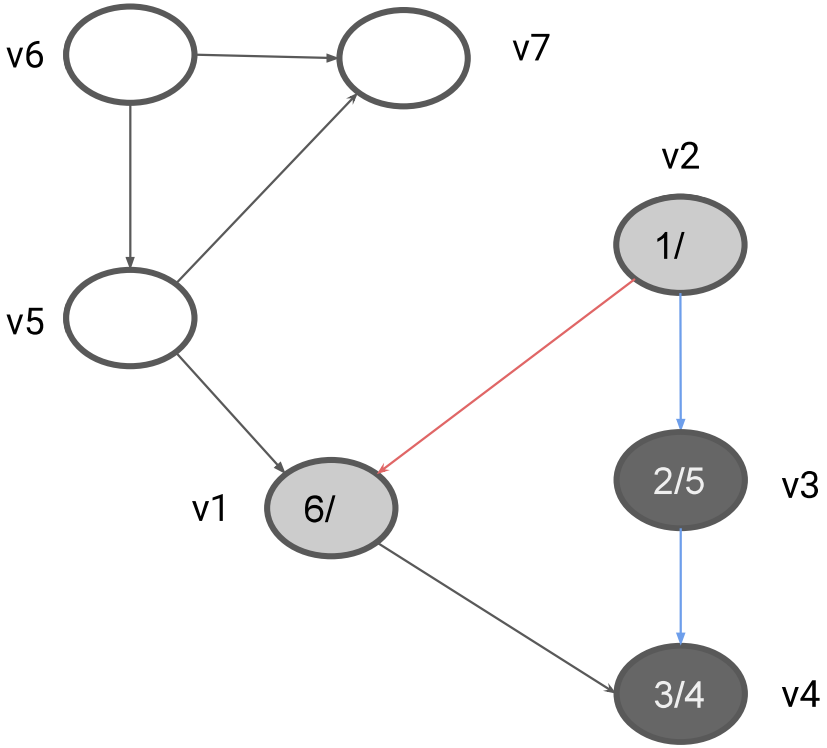
topologically ordered vertices:
`[v4, v3]`

Topological Sort Example



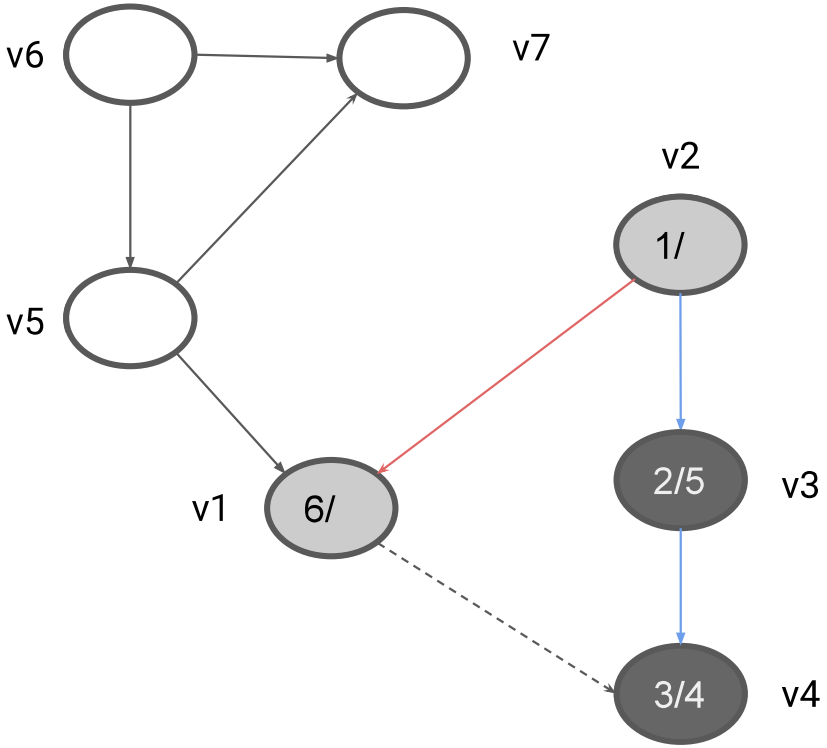
topologically ordered vertices:
`[v4, v3]`

Topological Sort Example



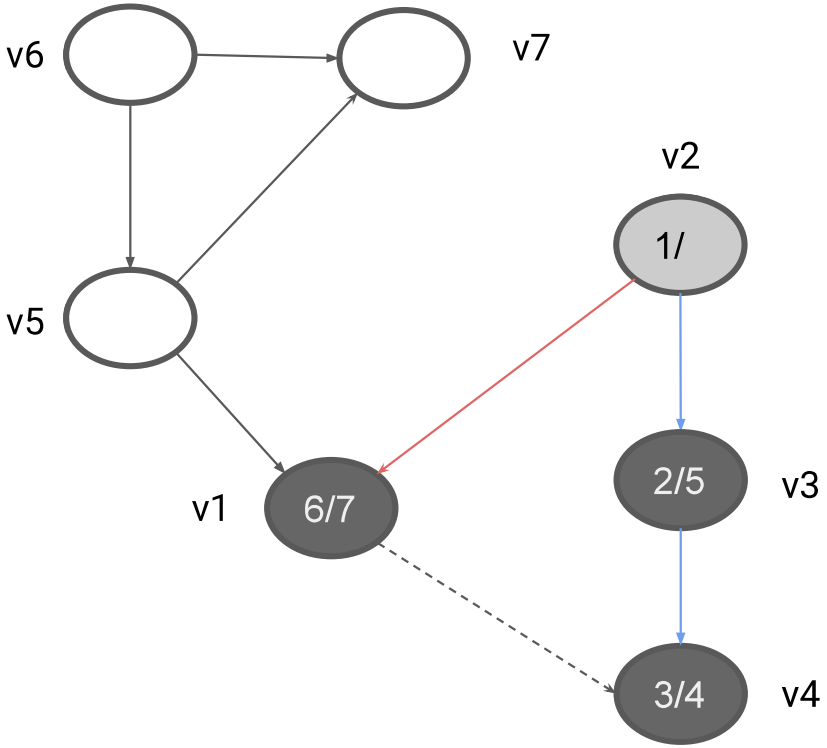
topologically ordered vertices:
`[v4, v3]`

Topological Sort Example



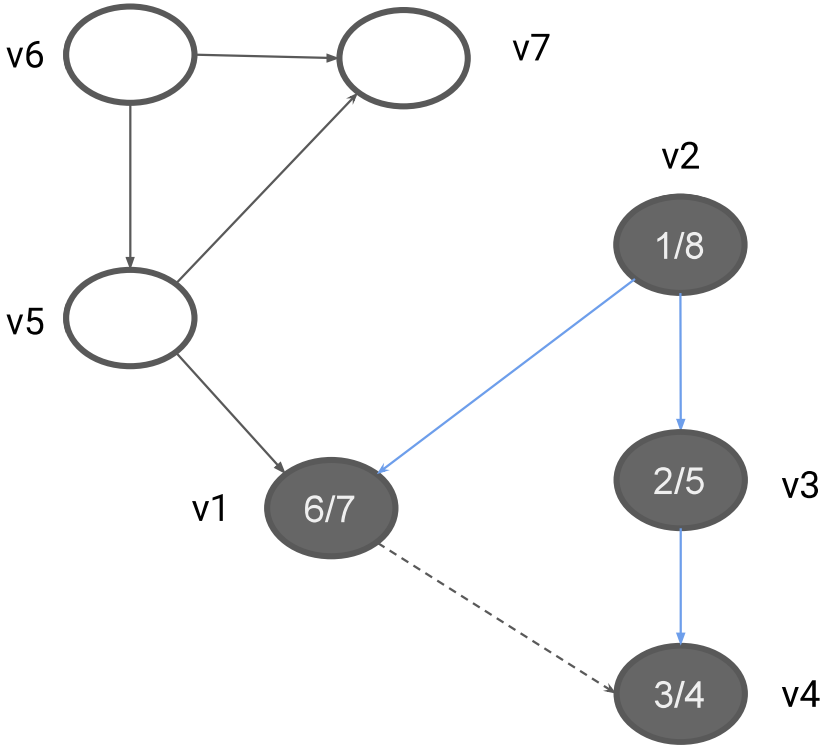
topologically ordered vertices:
`[v4, v3]`

Topological Sort Example



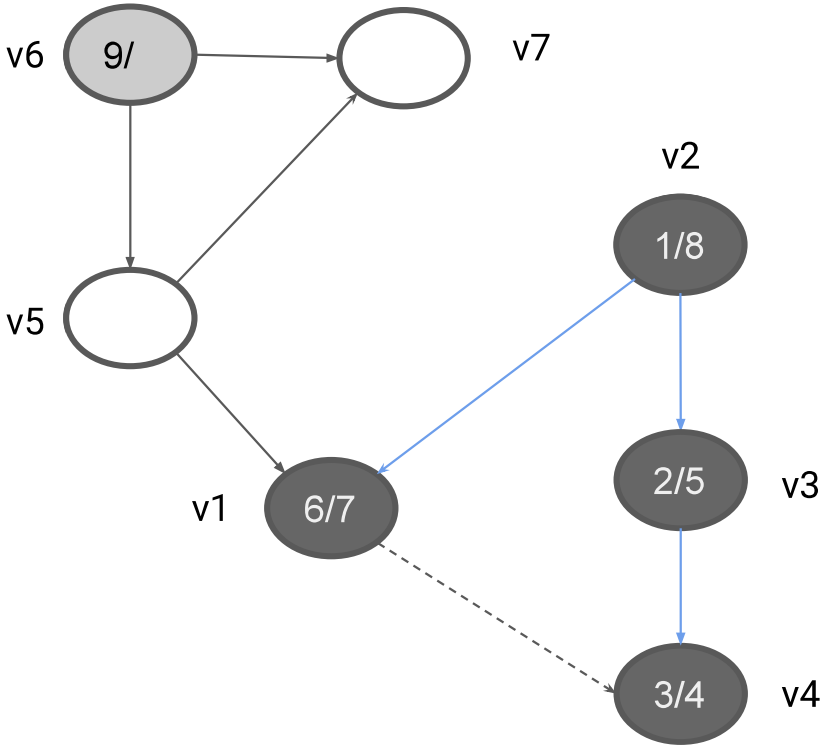
topologically ordered vertices:
`[v4, v3, v1]`

Topological Sort Example



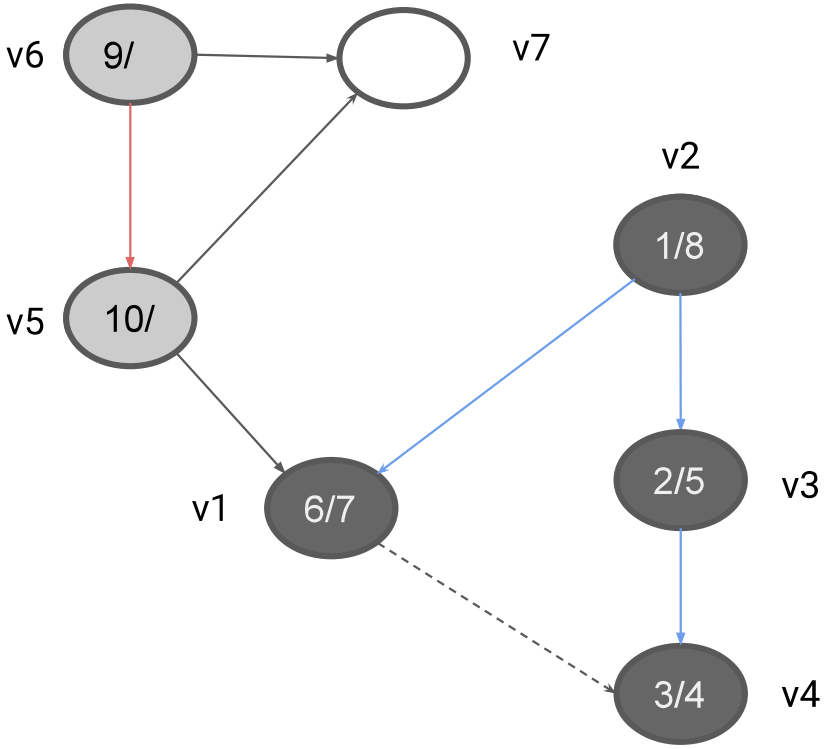
topologically ordered vertices:
`[v4, v3, v1, v2]`

Topological Sort Example



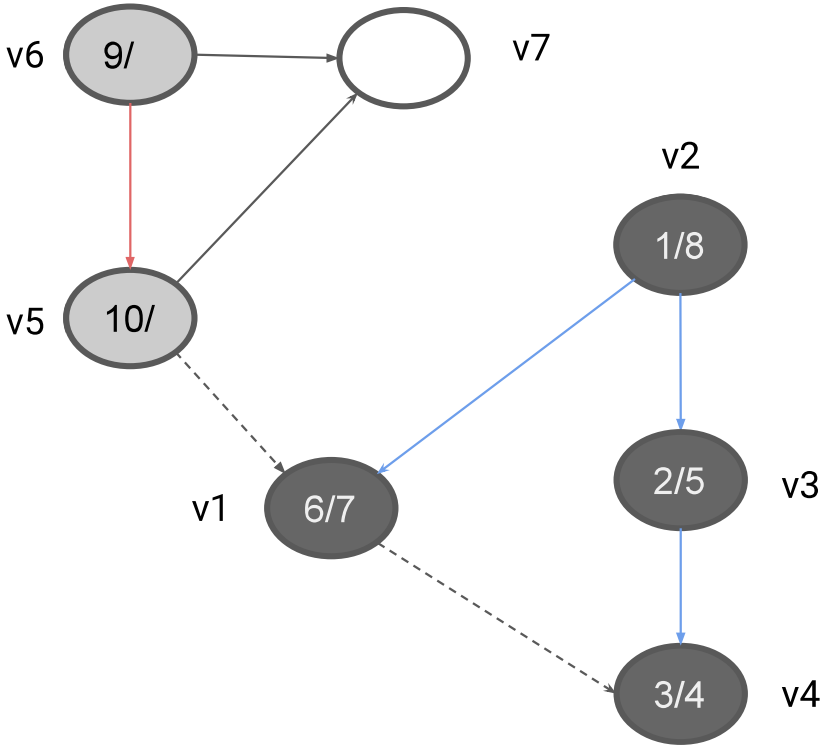
topologically ordered vertices:
`[v4, v3, v1, v2]`

Topological Sort Example



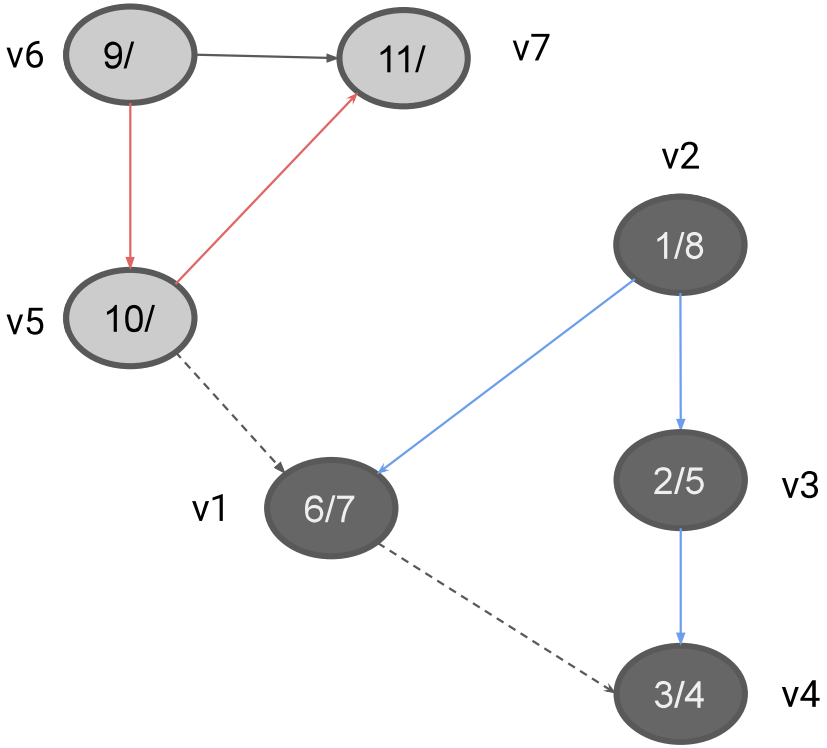
topologically ordered vertices:
`[v4, v3, v1, v2]`

Topological Sort Example



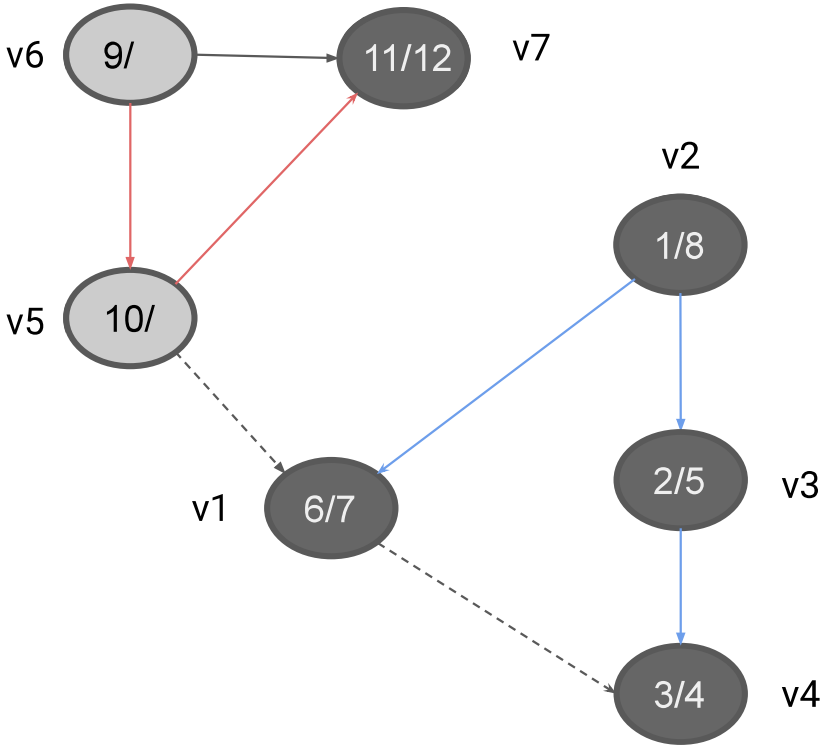
topologically ordered vertices:
`[v4, v3, v1, v2]`

Topological Sort Example



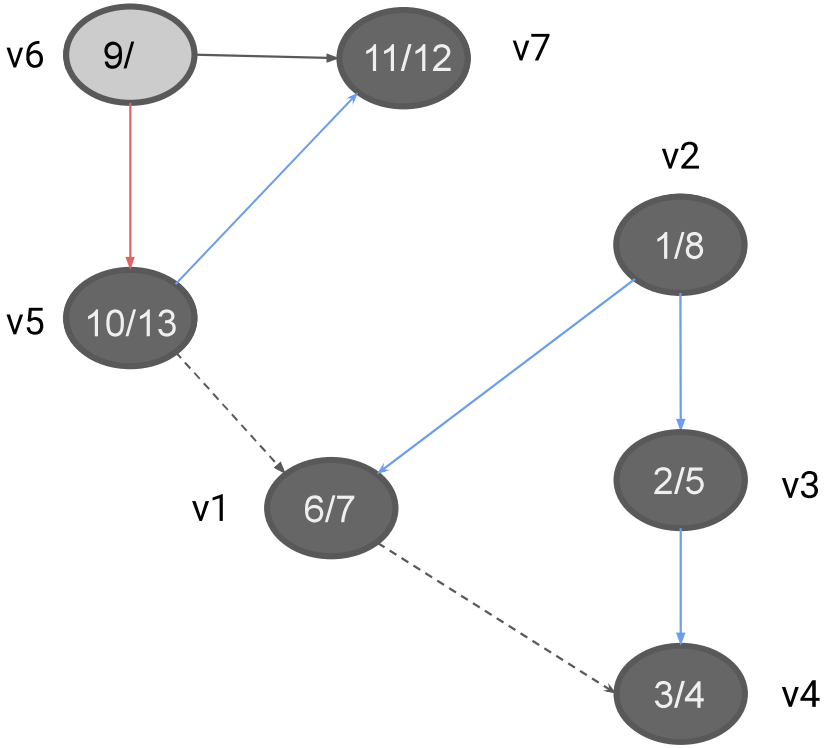
topologically ordered vertices:
`[v4, v3, v1, v2]`

Topological Sort Example



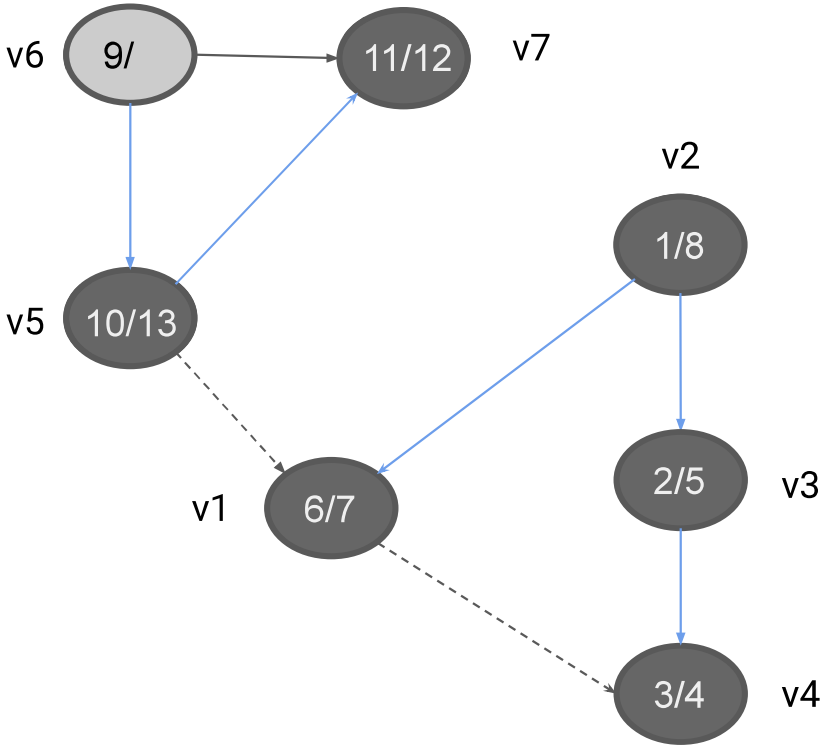
topologically ordered vertices:
`[v4, v3, v1, v2, v7]`

Topological Sort Example



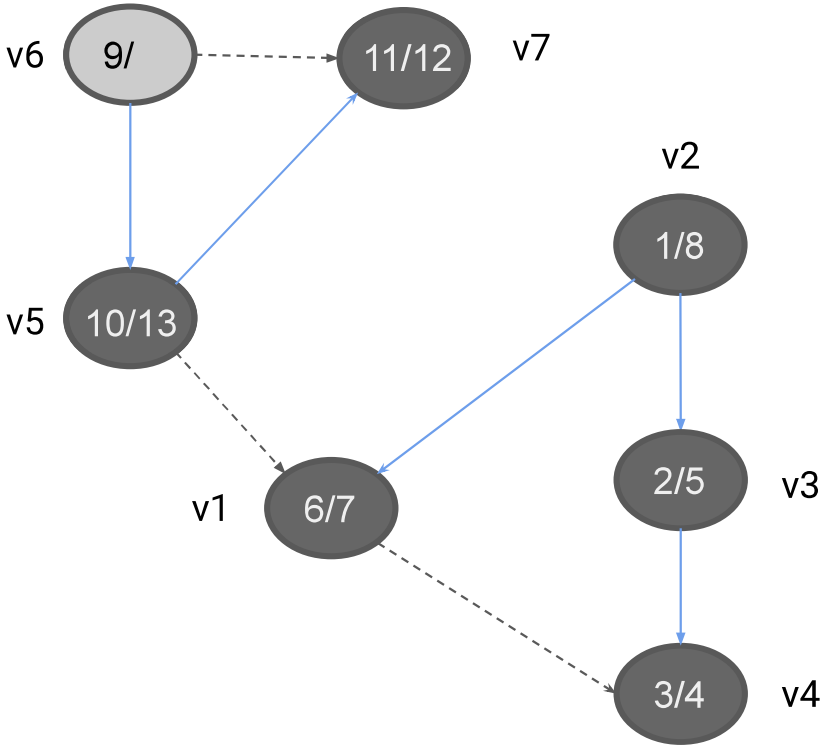
topologically ordered vertices:
`[v4, v3, v1, v2, v7, v5]`

Topological Sort Example



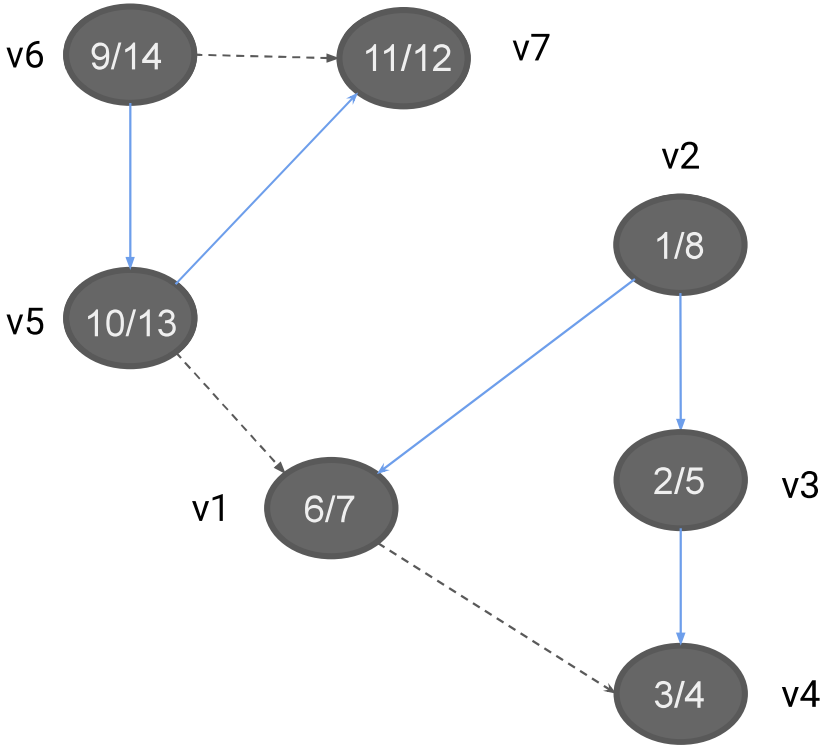
topologically ordered vertices:
`[v4, v3, v1, v2, v7, v5]`

Topological Sort Example



topologically ordered vertices:
`[v4, v3, v1, v2, v7, v5]`

Topological Sort Example



topologically ordered vertices:
`[v4, v3, v1, v2, v7, v5, v6]`

DFS Implementation Using a Stack

A normal DFS implementation on a directed graph might go like this:

```
visited = set()
# the second element in the tuple keeps track of the path leading up to the current vertex
stack = [(vertex, [])]

while stack:
    v, path = stack.pop()
    if v in visited:
        continue
    visited.add(v)

    # do something ...

    for child in v.children:
        if child not in visited:
            stack.append((child, path + [v]))
```

Topo Sort Implementation Using Stacks

However, for topological sorting we also have to keep track of the finish time.

When a vertex is popped off of the stack in the previous implementation we will lose track of it, which leads us to think about using another stack, called **gray_stack**, to model the evolution of the gray vertices.

Go back to the two pictorial animations for DFS and topo sort and look at how the gray patterns evolve.

The idea is that when the current vertex is no longer a descendent of the top of the **gray_stack**, it means that the traversal has jumped to a sibling vertex, which means that the outstanding vertices on top of the **gray_stack** should now become black vertices.

```
def get_topo_ordered_vertices(vertex_id_to_nodes, root_vertices):
    """
    Assumes a DAG, does not check for cycles.
    What do you have to do if you want to check for cycles? Hint: use black_nodes as a set.
    Time complexity O(|V| + |E|)
    vertex_id_to_nodes: maps vertex_id to vertex objects
    """
    order = []
    visited = set() # visited is the union of the gray and black vertices
    gray_stack = []
    stack = list(root_vertices)

    while stack:
        v = stack.pop()
        what do you do if v has already been visited?

        visited.add(v)

        while v is not a child of the vertex on the top of the gray stack
            g = gray_stack.pop()
            order.append(g)
        gray_stack.append(v)

        for c in vertex_id_to_nodes[v].children:
            what do you do if v has already been visited?
            stack.append(c)

    add the rest of the gray stack into order

    return order
```