

DSC 40B

Theoretical Foundations II

Lecture 13 | Part 1

Depth First Search

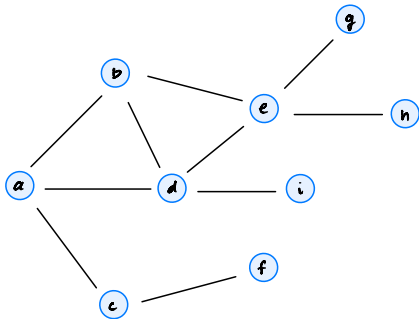
Visiting the Next Node

- ▶ Which node do we process next in a search?
- ▶ BFS: the **oldest** pending node.
- ▶ DFS (today): the **newest** pending node.
 - ▶ Naturally recursive.

```
def dfs(graph, u, status=None):  
    """Start a DFS at `u`."""  
    # initialize status if it was not passed  
    if status is None:  
        status = {node: 'undiscovered' for node in graph.nodes}  
  
    status[u] = 'pending'  
    for v in graph.neighbors(u): # explore edge (u, v)  
        if status[v] == 'undiscovered':  
            dfs(graph, v, status)  
    status[u] = 'visited'
```

Example

```
def dfs(graph, u, status=None):  
    """Start a DFS at `u`."""  
    ...  
    status[u] = 'pending'  
    for v in graph.neighbors(u): # explore edge (u, v)  
        if status[v] == 'undiscovered':  
            dfs(graph, v, status)  
    status[u] = 'visited'
```

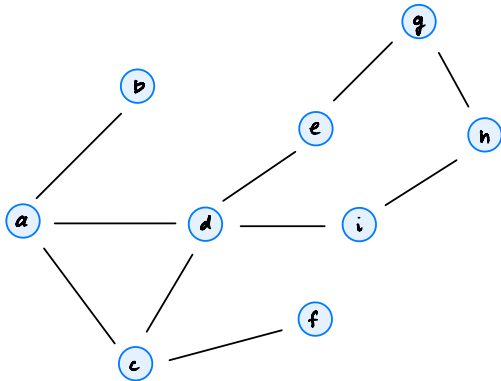


Main Idea

We'll see that the structure of the nested function calls gives us useful information about the graph's structure.

Exercise

Write the nested function calls for a DFS on the graph below.



Full DFS

- ▶ DFS will visit all nodes reachable from source.
- ▶ To visit all nodes in graph, need **full DFS**.

```
def full_dfs(graph):  
    status = {node: 'undiscovered' for node in graph.nodes}  
    for node in graph.nodes:  
        if status[node] == 'undiscovered'  
            dfs(graph, node, status)
```

```
def full_dfs(graph):
    status = {node: 'undiscovered' for node in graph.nodes}
    for node in graph.nodes:
        if status[node] == 'undiscovered':
            dfs(graph, node, status)

def dfs(graph, u, status=None):
    """Start a DFS at `u`."""
    # initialize status if it was not passed
    if status is None:
        status = {node: 'undiscovered' for node in graph.nodes}

    status[u] = 'pending'
    for v in graph.neighbors(u): # explore edge (u, v)
        if status[v] == 'undiscovered':
            dfs(graph, v, status)
    status[u] = 'visited'
```


Time Complexity

- ▶ In a full DFS:
 - ▶ dfs called on each node exactly once.
 - ▶ Like BFS, each edge is explored exactly:
 - ▶ once if directed
 - ▶ twice if undirected
- ▶ Time: $\Theta(V + E)$, just like BFS.

DSC 40B

Theoretical Foundations II

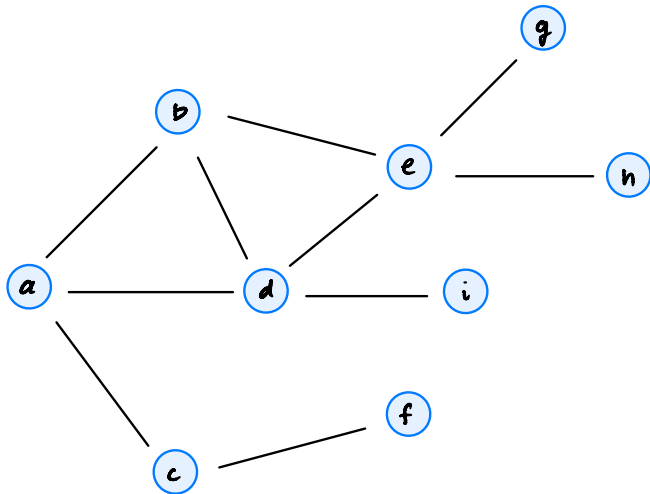
Lecture 13 | Part 2

Nesting Properties of DFS

Key Property of DFS (Informal)

- ▶ Suppose v is reachable from u , and v is **undiscovered** at the time of $\text{dfs}(u)$.
- ▶ If there is a path of undiscovered nodes from u to v at the time of $\text{dfs}(u)$:
 - ▶ $\text{dfs}(v)$ will be run.
 - ▶ v will be marked as **visited**.

Example



Start and Finish Times

- ▶ Keep a running clock (an integer).
- ▶ For each node, record
 - ▶ **Start time**: time when marked pending
 - ▶ **Finish time**: time when marked visited
- ▶ Increment clock whenever node is marked pending/visited

```

from dataclasses import dataclass

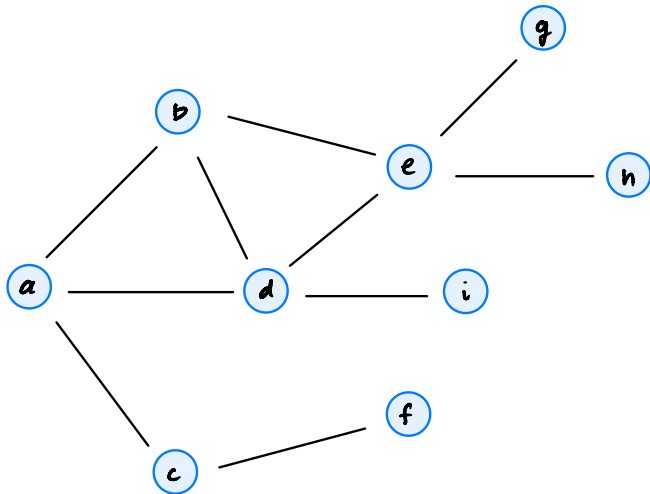
@dataclass
class Times:
    clock: int
    start: dict
    finish: dict

def full_dfs_times(graph):
    status = {node: 'undiscovered' for node in graph.nodes}
    predecessor = {node: None for node in graph.nodes}
    times = Times(clock=0, start={}, finish={})
    for u in graph.nodes:
        if status[u] == 'undiscovered':
            dfs_times(graph, u, status, times)
    return times, predecessor

def dfs_times(graph, u, status, predecessor, times):
    times.clock += 1
    times.start[u] = times.clock
    status[u] = 'pending'
    for v in graph.neighbors(u): # explore edge (u, v)
        if status[v] == 'undiscovered':
            predecessor[v] = u
            dfs_times(graph, v, status, times)
    status[u] = 'visited'
    times.clock += 1
    times.finish[u] = times.clock

```

Example



Key Property

- ▶ Take any two nodes u and v ($u \neq v$).
- ▶ If v is *started* between $\text{start}[u]$ and $\text{finish}[u]$, then v is *finished* between $\text{start}[u]$ and $\text{finish}[u]$.

Key Property

- ▶ Take any two nodes u and v ($u \neq v$).
- ▶ Assume for simplicity that $\text{start}[u] < \text{start}[v]$.
- ▶ Exactly one of these is true:
 - ▶ $\text{start}[u] < \text{start}[v] < \text{finish}[v] < \text{finish}[u]$
 - ▶ $\text{start}[u] < \text{finish}[u] < \text{start}[v] < \text{finish}[v]$

DSC 40B

Theoretical Foundations II

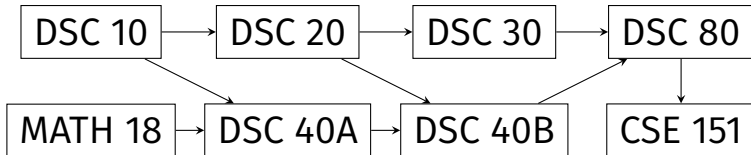
Lecture 13 | Part 3

Topological Sort

Applications of DFS

- ▶ Is node v reachable from node u ?
- ▶ Is the graph connected?
- ▶ How many connected components?
- ▶ What is the shortest path between u and v ? **No.**

Prerequisite Graphs



Goal: find order in which to take classes satisfying prerequisites.

Directed Acyclic Graphs

- ▶ A **directed cycle** is a path from a node to itself with at least one edge.
- ▶ A **directed acyclic graph (DAG)** is a directed graph with no directed cycles.

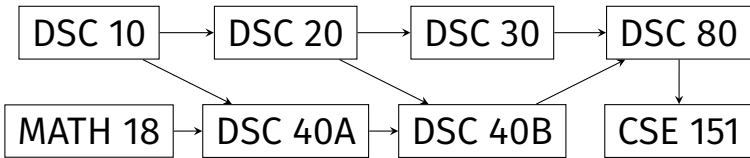
Example

- ▶ Prerequisite graphs are DAGs.
 - ▶ Or at least, they should be!

Topological Sorts

- ▶ **Given:** a DAG, $G = (V, E)$.
- ▶ **Compute:** an ordering of V such that if $(u, v) \in E$, then u comes before v in the ordering
- ▶ This is called a **topological sort** of G .

Example



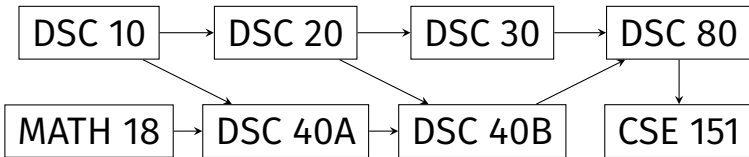
MATH 18, DSC 10, DSC 40A, DSC 20, DSC 40B, DSC 30, DSC 80, CSE 151

Key Property

- ▶ Take any two nodes u and v ($u \neq v$).
- ▶ Assume the graph is a DAG.
- ▶ **Example:** If v is reachable from u , then $\text{finish}[v] < \text{finish}[u]$.

Exercise

Compute start and finish times using DSC 10 as the source.



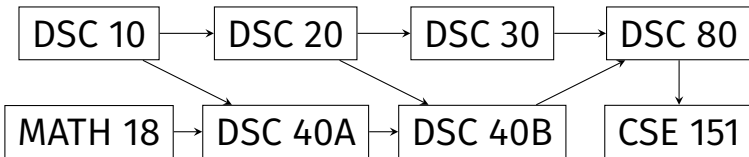
An Algorithm

- ▶ **Recall:** If v is reachable from u , then $\text{finish}[v] \leq \text{finish}[u]$.
- ▶ If v is reachable from u , u should come before v .
- ▶ Idea: nodes with later finish times should come first.

Algorithm

- ▶ To find a topological sort (if it exists):
 - ▶ Compute times with Full DFS.
 - ▶ Sort in **descending** order by finish time.
- ▶ Time complexity:

Example



Note

- ▶ There can be many valid topological sorts!