Data Structures and Algorithms

Lab 10
Topological sorting

Agenda

- Recall DFS, BFS
- Topological sorting
- Coding exercise

DFS

Which data structure do we use?

What kind of tasks can it solve?

DFS

Which data structure do we use?

- Stack

What kind of tasks can it solve?

- Path from a to b (lexicographical)
- Topological sorting
- Find a cycle
- Find connected components
- ...

DFS pseudocode

Algorithm 1: Recursive DFS

Data: G: The graph stored in an adjacency list
root: The starting node
Result: Prints all nodes inside the graph in the DFS order
$visited \leftarrow \{false\}$;
$DFS(root)$;

Function $DFS(u)$**:**
    if $visited[u] = true$ **then**
        return;
    end
    $print(u)$;
    $visited[u] \leftarrow true$;
    for $v \in G[u].neighbors()$ **do**
        $DFS(v)$;
    end
end

BFS

Which data structure do we use?

What kind of tasks can it solve?

BFS

Which data structure do we use?

- Queue

What kind of tasks can it solve?

- Shortest path from a to b
- Find shortest cycle
- Find connected components
- ...

BFS pseudocode

```
procedure BFS(G,s)

    for each vertex v ∈ V[G] do
        explored[v] ← false
        d[v] ← ∞
    end for
    explored[s] ← true
    d[s] ← 0
    Q:= a queue data structure, initialized with s
    while Q ≠ ϕ do
        u ← remove vertex from the front of Q
        for each v adjacent to u do
            if not explored[v] then
                explored[v] ← true
                d[v] ← d[u] + 1
                insert v to the end of Q
            end if
        end for
    end while

end procedure
```

Cycle in graph

How to find a cycle in given graph?

During DFS let’s “color” vertices:

1. “Unvisited” vertices are **white**
2. “visited” are **gray**

If during DFS we meet gray vertice – means we found a cycle.

Topological sorting

A linear ordering of vertices such that

- for every directed edge $U \to V$
- vertex U comes before V in the ordering

We can apply topological sorting Directed Acyclic Graphs(DAG)

Topological sorting: Example

Given a list of tasks:

- Task 1: Find a client
- Task 2: List requirements
- Task 3: Start working on the project
- Task 4: Build a team
- etc.

Suggest a possible order of execution considering that some tasks depend on result of others.

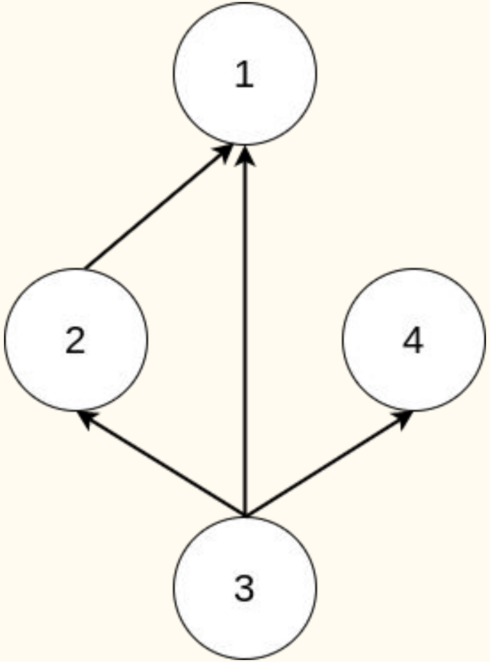
- Task 2 depends on task 1;
- Task 3 depends on task 1, 2 and 4;

Topological sorting: Example

Given a list of tasks, some tasks depend on result of others.

Suggest a possible order of execution.

- Task 2 depends on task 1;
- Task 3 depends on task 1, 2 and 4;



How many ways to execute all of them?

Topological sorting: Example

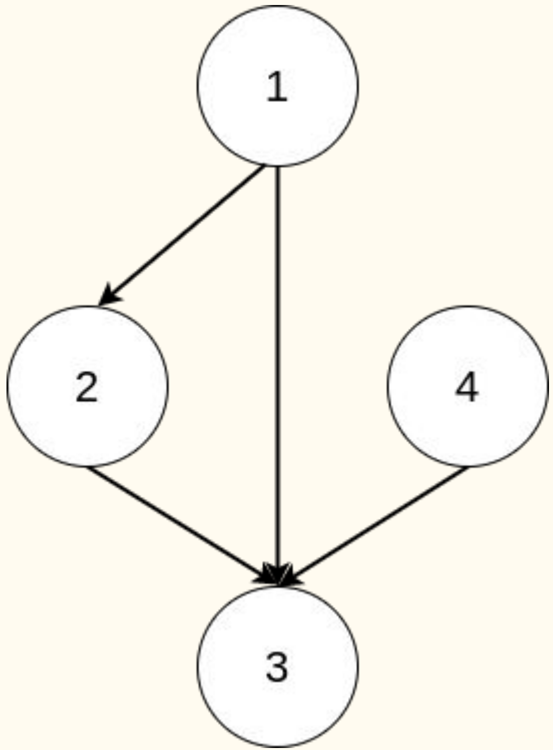
How should we modify the graph?

From:

- Task 2 depends on task 1;
- Task 3 depends on task 1, 2 and 4;

Transpose to:

- Task 1 executes before task 2 and 3
- Task 2 executes before task 3
- Task 4 executes before task 3



Topological sorting

How to order vertices?

- Record when we leave a node
- The node we leave first has the most dependencies.
    - The vertex with the maximum in-degree
- The node we leave last has the least dependencies.
    - The vertex with the minimum in-degree

Topological sorting: implementation

- In DFS, we print a vertex and then recursively call DFS for its adjacent vertices.
- In topological sorting, we need to print a vertex before its adjacent vertices.
- In DFS, we start from a vertex, we first print it and then recursively call DFS for its adjacent vertices.
- In topological sorting, we use a temporary stack. We don't print the vertex immediately, we first recursively call topological sorting for all its adjacent vertices, then push it to a stack.
- Finally, print contents of the stack.
- Note that a vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in the stack.

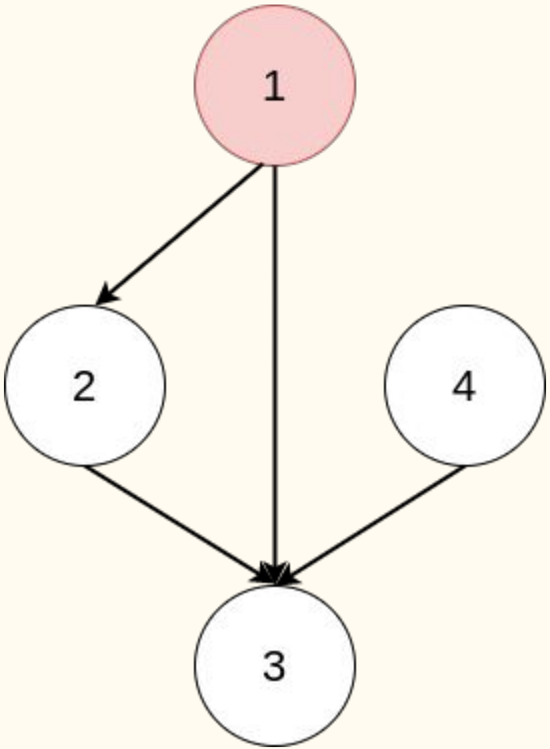
Topological sorting: implementation

Algorithm 3: DFS Algorithm for Topological Sort

```
Data: A DAG G
Result: A topological sort of all vertices in G
Create an empty vertex list L;
Create a visited array to indicate whether a vertex has been visited;
Initialize all elements in visited with false;
foreach vertex u∈G do
    if visited[u] is false then
        topologicalSortRecursive(u);
    end
end
return L;

Function topologicalSortRecursive(u):
    visited[u] = true;
    foreach u's neighboring vertex v do
        if visited[v] is false then
            topologicalSortRecursive(v);
        end
    end
    Add u to the front of list L;
```

Sorting

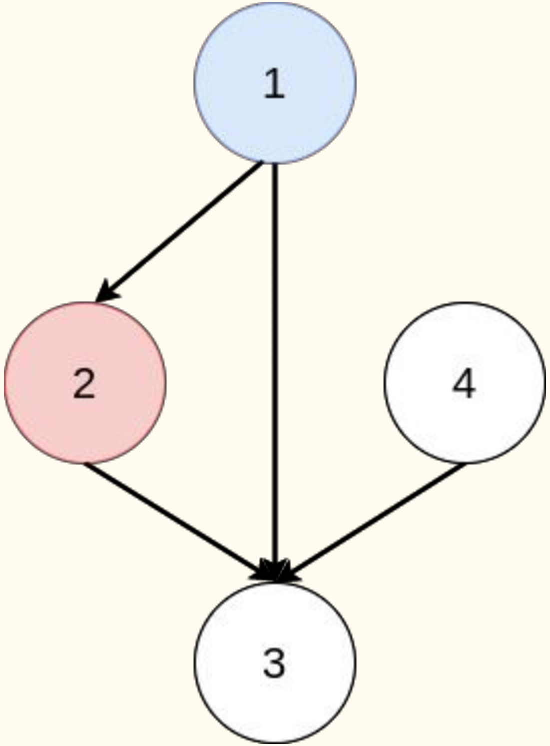


DFS Stack

1			

T.Sort

Sorting

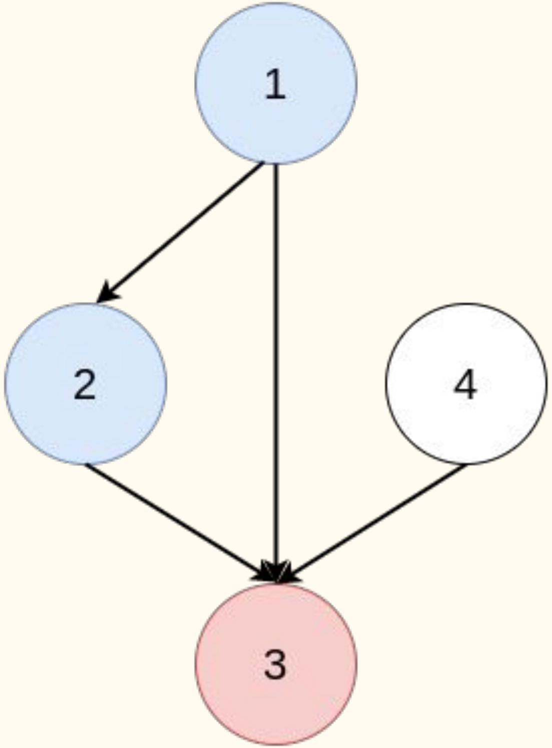


DFS Stack

1	2		

T.Sort

Sorting

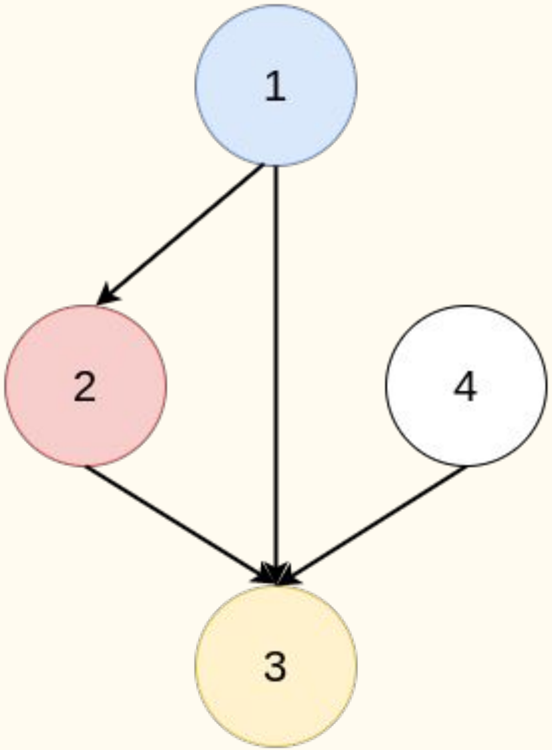


DFS Stack

1	2	3	

T.Sort

Sorting



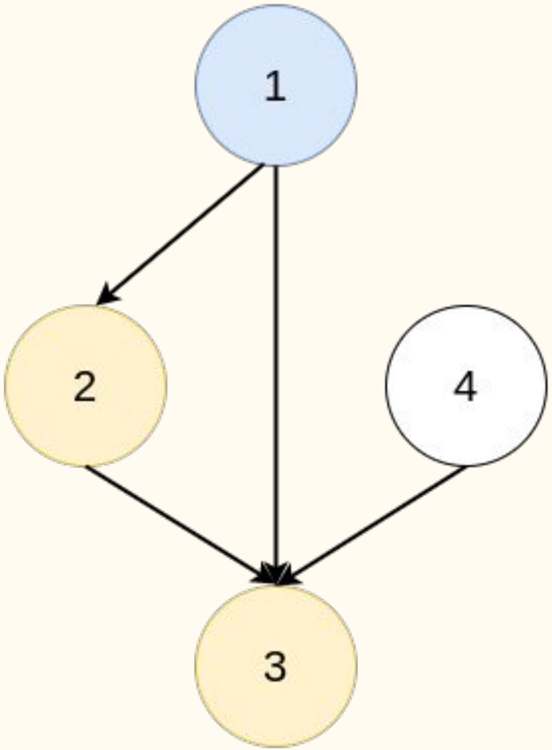
DFS Stack

1	**2**		

T.Sort

3			

Sorting



DFS Stack

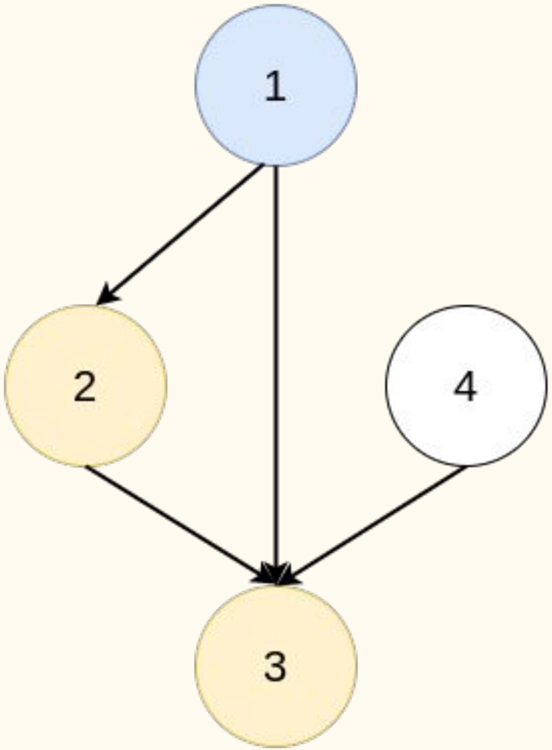
1			

T.Sort

3			

Where to put 2 ?

Sorting



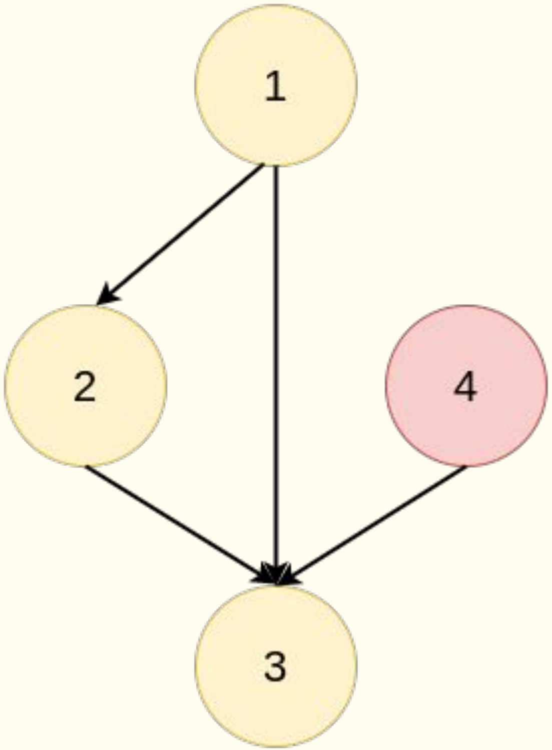
DFS Stack

1			

T.Sort

3	**2**		

Sorting

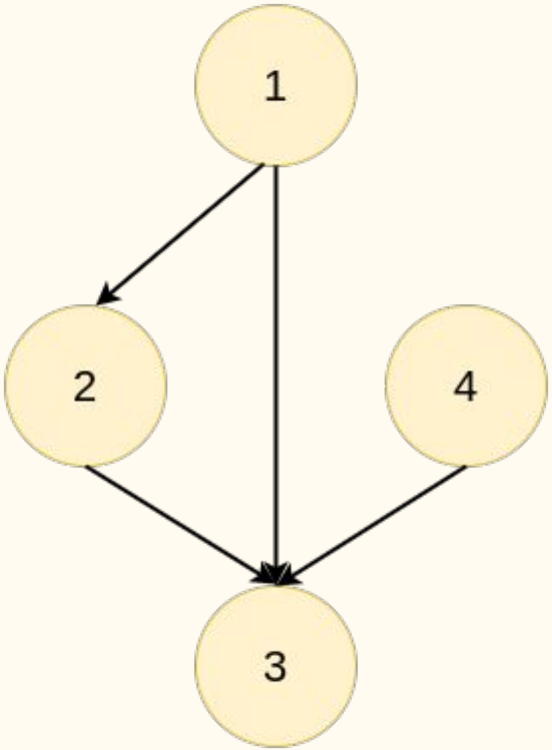


DFS Stack

T.Sort

3	2	1	

Sorting

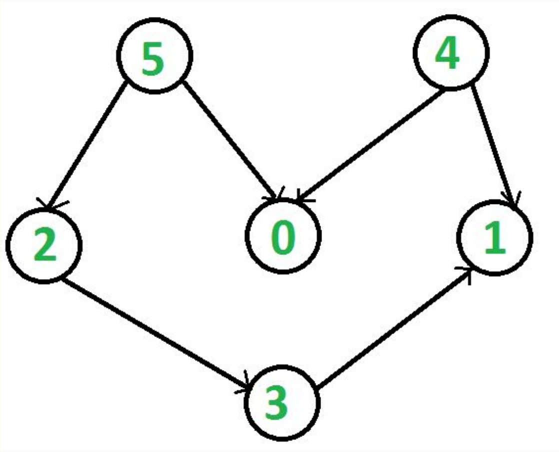


DFS Stack

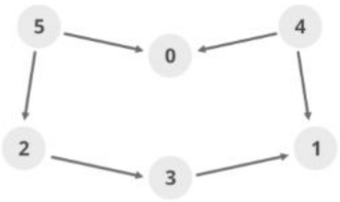
T.Sort

3	2	1	4

Another example



- A topological sorting of the following graph is “5 4 2 3 1 0”.
- There can be more than one topological sorting for a graph.
- For example, another topological sorting of the following graph is “4 5 2 3 1 0”. The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no incoming edges).



Adja cent list (G)
0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

	0	1	2	3	4	5
visited	false	false	false	false	false	false

Stack(empty)

Step 1: Topological Sort(0), visited[0] = true
↓
List is empty. No more recursion call.
Stack | 0 |

Step 2: Topological Sort(1), visited[1] = true
↓
List is empty. No more recursion call.
Stack | 0 | 1 |

Step 3: Topological Sort(2), visited[2] = true
↓
Topological Sort(3), visited[3] = true
↓
'1' is already visited. No more recurrsion call
Stack | 0 | 1 | 3 | 2 |

Step 4: Topological Sort(4), visited[4] = true
↓
'0' , '1' are already visited. No more recurrsion call
Stack | 0 | 1 | 3 | 2 | 4 |

Step 5: Topological Sort(5), visited[5] = true
↓
'2' , '0' are already visited. No more recurrsion call
Stack | 0 | 1 | 3 | 2 | 4 | 5 |

Step 6: Print all elements of stack from top to bottom

Topological Sorting: Implementation

Implement topological sorting

See You next week!