CSCI 136:
Data Structures
and
Advanced Programming

Lecture 28

Graphs, part 3

Instructor: Dan Barowy

Williams

Announcements

New majors, welcome.

20 colloquiua to graduate.

Today's speaker: James Lester from NCSU, 2:30-4pm, Wege Auditorium.

Lab 10: choose your own partner.

Two-week lab.

May 8 lab meeting is optional.

Outline

Greedy algorithms

Minimum-weight spanning trees

Directed Acyclic Graphs

Topological ordering

Quiz

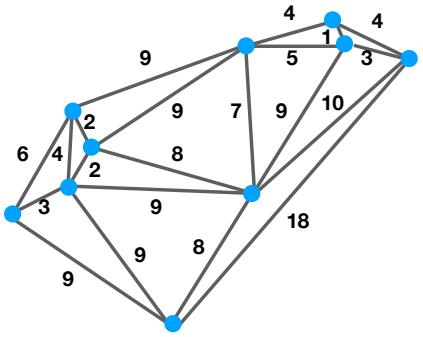
Greedy algorithms

Greedy algorithm

A **greedy algorithm** is a **style of algorithm** that makes **locally-optimal choices** in an attempt to compute a **globally-optimal solution**. Greedy algorithms may or may not find the globally-optimal solution. However, greedy algorithms are usually **fast**, and they often compute a **close approximation** of the globally-optimal solutions.

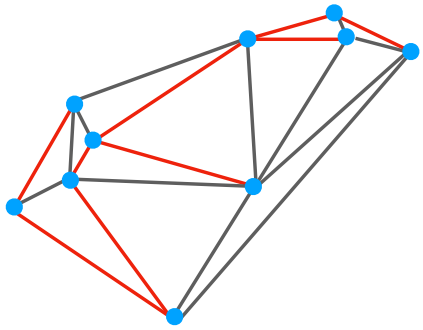
Greedy algorithm: example

Minimum weight spanning tree problem



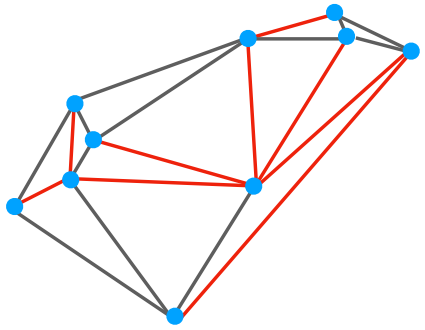
Spanning tree

Given a **connected graph**, a **spanning tree** is a subset of edges that is both a **tree** and **connects all vertices** in the graph.



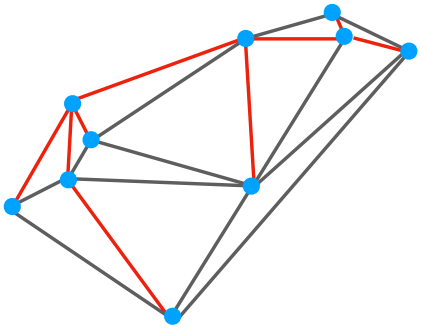
Spanning tree

Given a **connected graph**, a **spanning tree** is a subset of edges that is both a **tree** and **connects all vertices** in the graph.



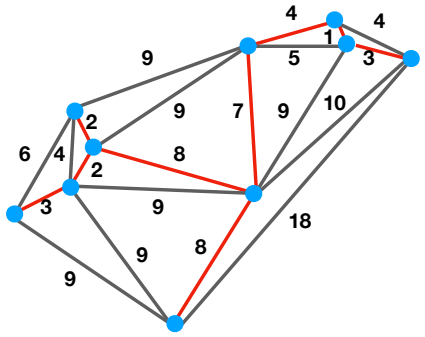
Spanning tree

Given a **connected graph**, a **spanning tree** is a subset of edges that is both a **tree** and **connects all vertices** in the graph.



Minimum-Weight Spanning tree

Given a **connected graph with edge weights**, a **minimum-weight spanning tree** is spanning tree that minimizes the sum of the edge weights.



1 + 2 + 2 + 3 + 3 + 4 + 7 + 8 + 8 = 38

MWST problem

Given a **connected graph with edge weights**, find a **minimum-weight spanning tree**.

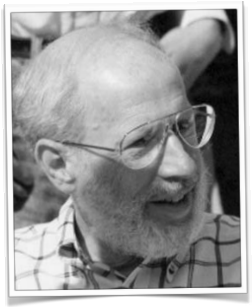
Conveniently, MWST admits a greedy solution.

Kruskal's algorithm:

Invented by Joseph Kruskal in 1956.

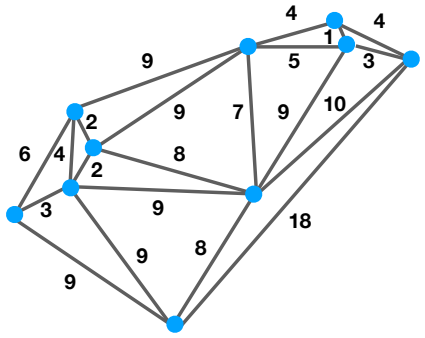
Simple idea:

- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



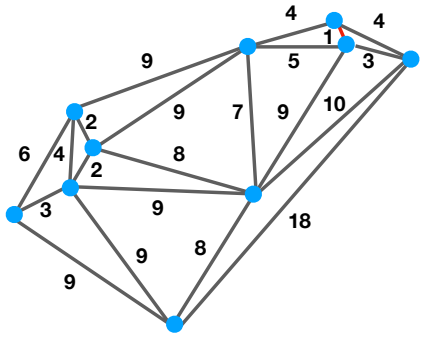
Kruskal's algorithm

- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



Kruskal's algorithm

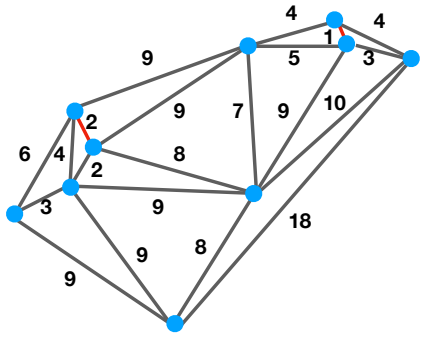
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1

Kruskal's algorithm

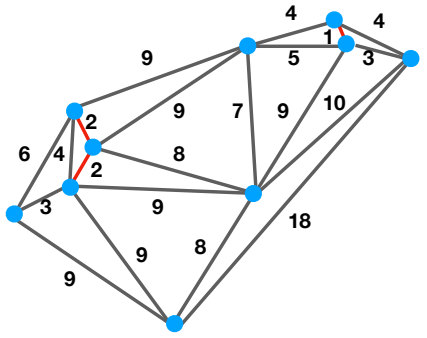
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2

Kruskal's algorithm

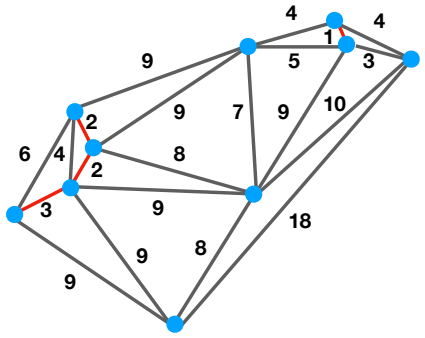
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2 + 2

Kruskal's algorithm

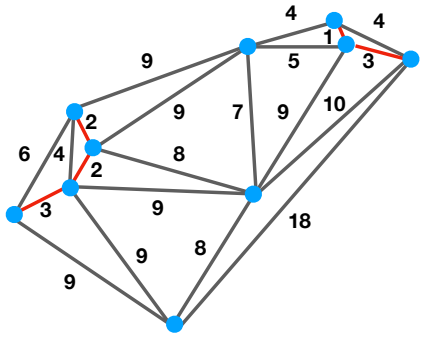
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2 + 2 + 3

Kruskal's algorithm

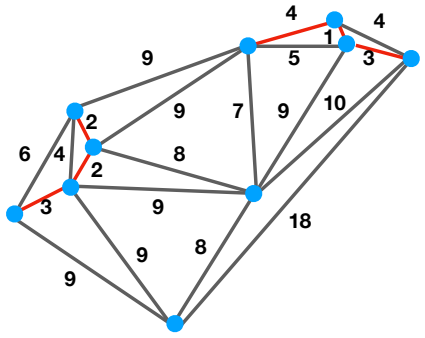
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2 + 2 + 3 + 3

Kruskal's algorithm

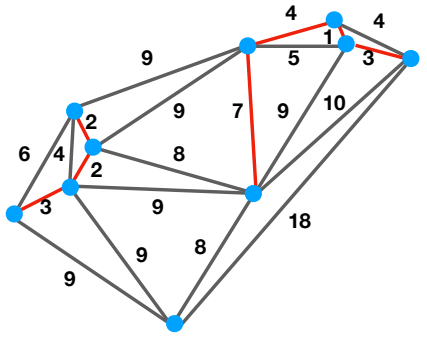
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2 + 2 + 3 + 3 + 4

Kruskal's algorithm

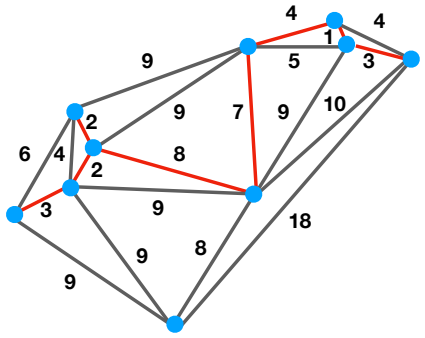
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2 + 2 + 3 + 3 + 4 + 7

Kruskal's algorithm

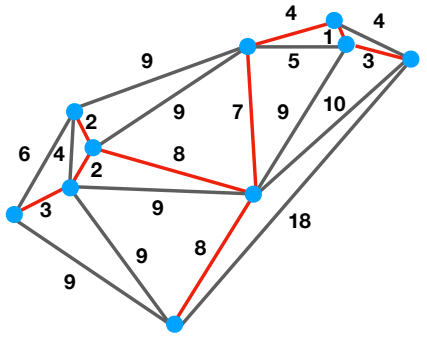
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2 + 2 + 3 + 3 + 4 + 7 + 8

Kruskal's algorithm

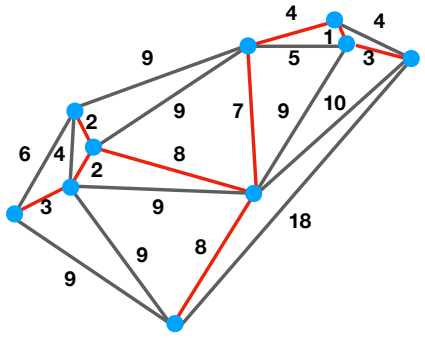
- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2 + 2 + 3 + 3 + 4 + 7 + 8 + 8

Kruskal's algorithm

- Pick the **smallest weight edge** that does not introduce a cycle.
- **Repeat** until the tree includes all vertices.



1 + 2 + 2 + 3 + 3 + 4 + 7 + 8 + 8 = 38

Global optima

Greedy algorithms are **guaranteed** to produce globally-optimal solutions when two conditions hold:

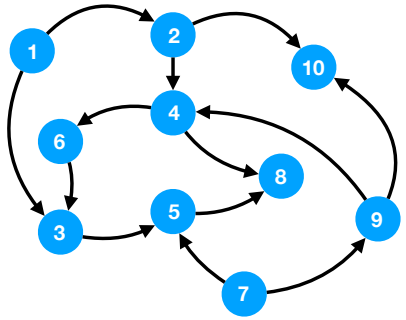
Optimal substructure: the optimal solution is composed of optimal solutions to its subproblems.

Greedy choice property: locally-optimal decisions are sufficient to find optimal solutions; i.e., a greedy algorithm never reconsiders a decision.

Directed Acyclic Graphs

Directed Acyclic Graph

A **directed acyclic graph** (DAG) is a **directed graph** that contains no **directed cycles**.



Directed Acyclic Graph

DAGs are widely used to encode relations that admit a **partial order**.

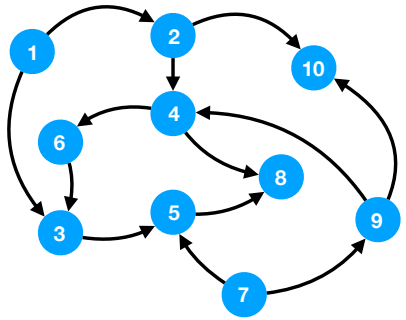
In particular, they are often used in scenarios where is the there is a **dependence relationship**.

E.g., Java source code files have a dependence relationship that forms a DAG.

```
import java.util.Iterator;
import structure5.*;
…
```

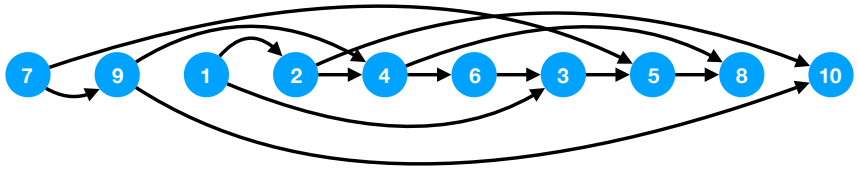
Topological ordering

A **topological ordering** of a **directed acyclic graph** is a **linear ordering of its vertices** such that for every directed edge **u,v** from vertex **u** to vertex **v**, **u** comes before **v** in the ordering.



Topological ordering

A **topological ordering** of a **directed acyclic graph** is a **linear ordering of its vertices** such that for every directed edge **u,v** from vertex **u** to vertex **v**, **u** comes before **v** in the ordering.



Topological ordering

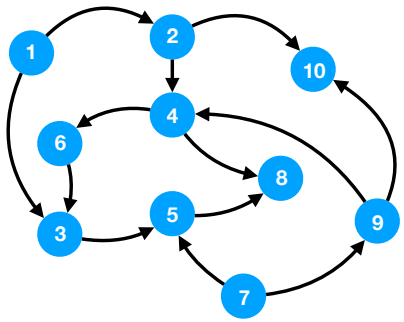
E.g., **how** does Java decide what **source code files** to **compile first**?

`javac` produces a **topological ordering** of the vertices in the file dependence graph.

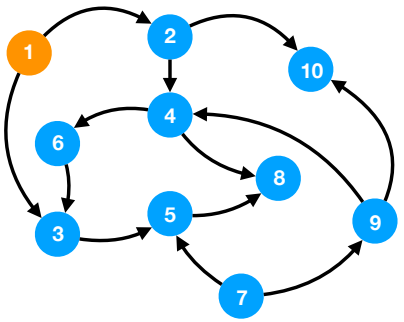
Algorithm: topological sort:

- For each node of the graph (in any order), recursively visit in a depth-first manner. After visiting each node, add it to the head of the list.
- When visiting, return (do not recurse) when:
  - A node has already been visited, or
  - the node has no outgoing edges.

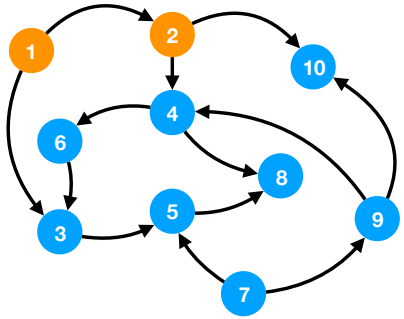
Topological sort



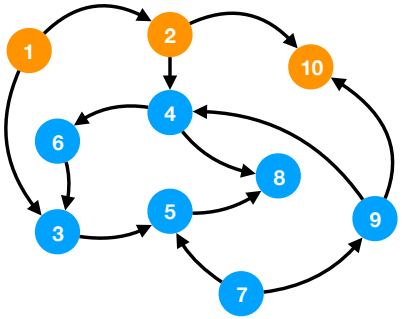
Topological sort



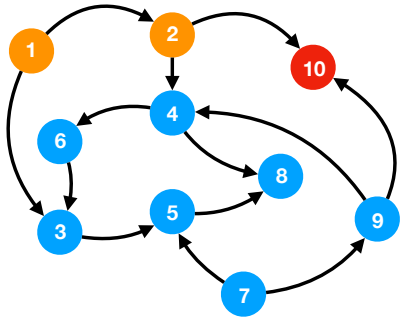
Topological sort



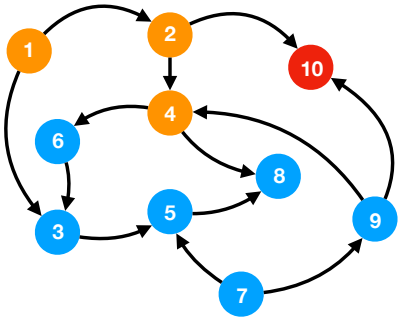
Topological sort



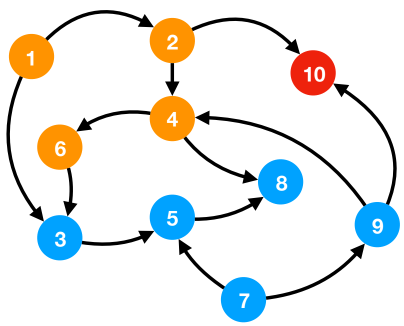
Topological sort



Topological sort

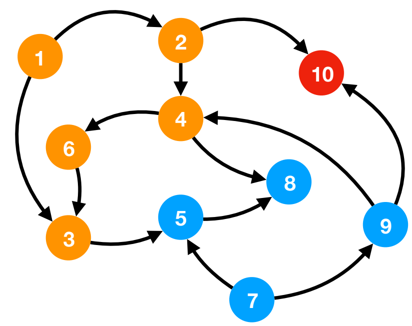


Topological sort



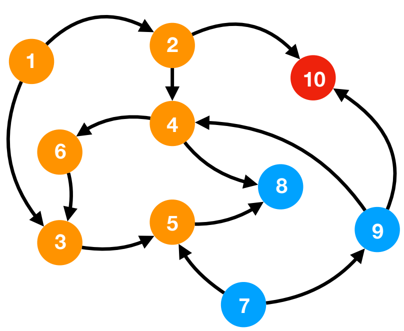
10

Topological sort



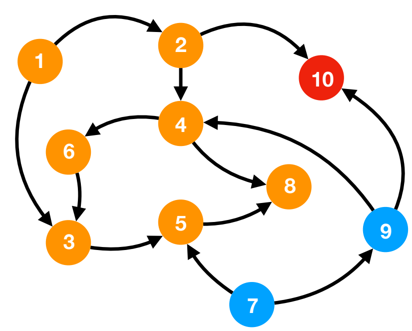
10

Topological sort



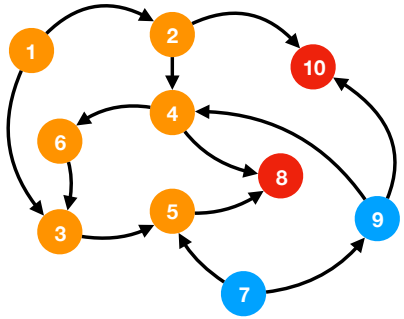
10

Topological sort



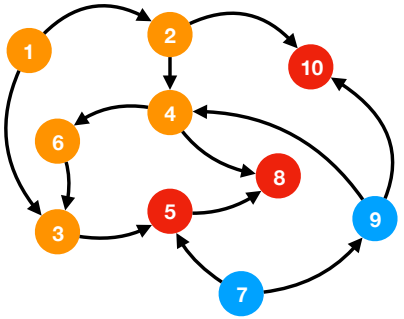
10

Topological sort



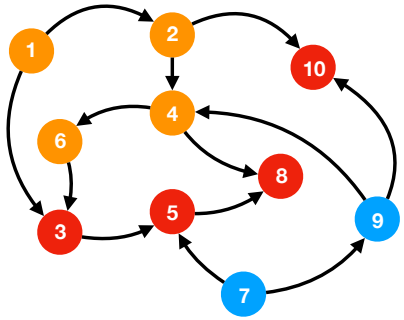
8, 10

Topological sort



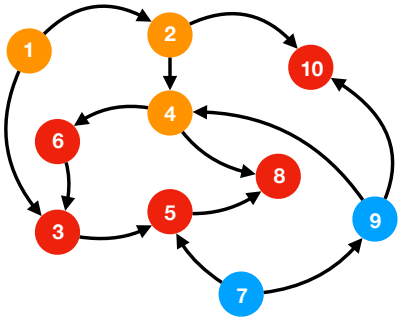
5, 8, 10

Topological sort



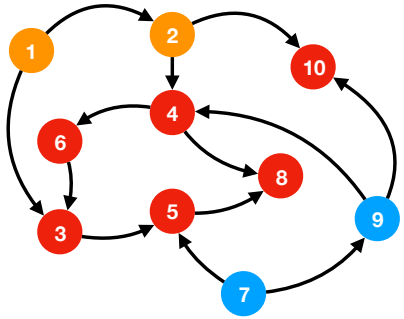
3, 5, 8, 10

Topological sort



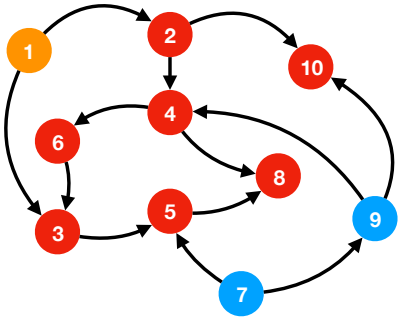
6, 3, 5, 8, 10

Topological sort



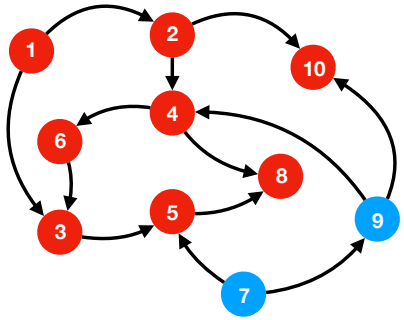
4, 6, 3, 5, 8, 10

Topological sort



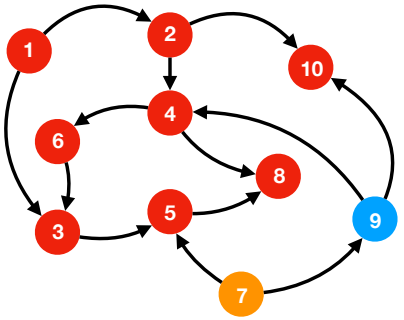
2, 4, 6, 3, 5, 8, 10

Topological sort



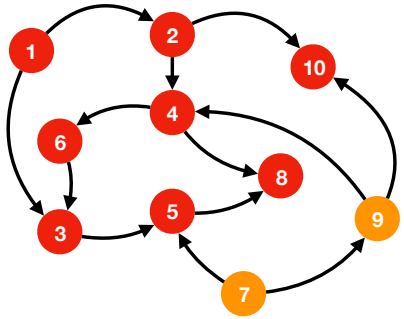
1, 2, 4, 6, 3, 5, 8, 10

Topological sort



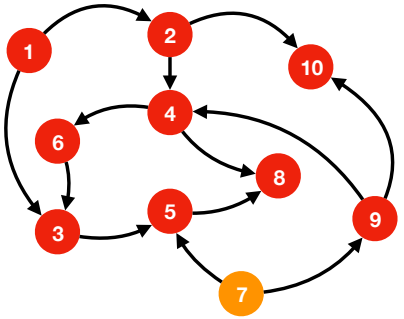
1, 2, 4, 6, 3, 5, 8, 10

Topological sort



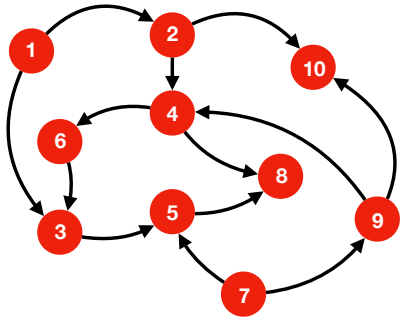
1, 2, 4, 6, 3, 5, 8, 10

Topological sort



9, 1, 2, 4, 6, 3, 5, 8, 10

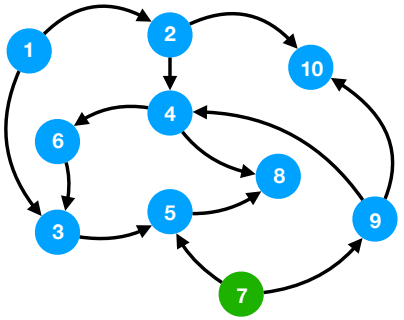
Topological sort



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

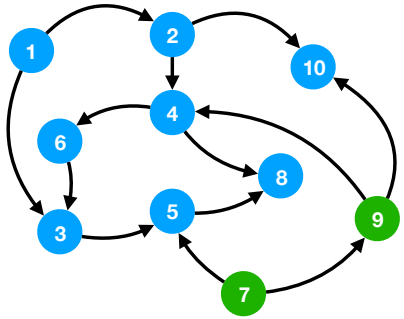
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

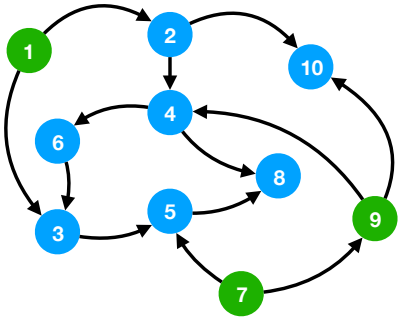
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

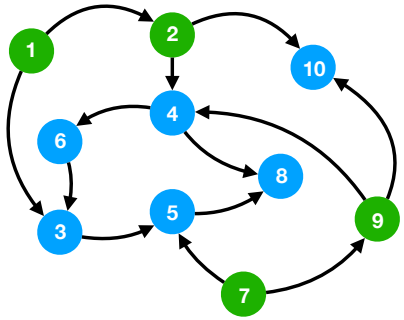
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

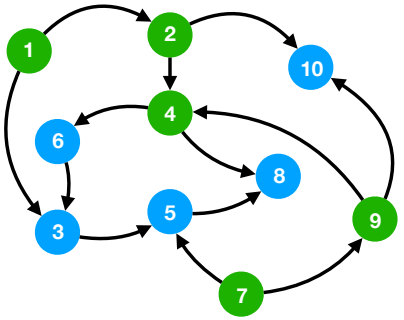
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

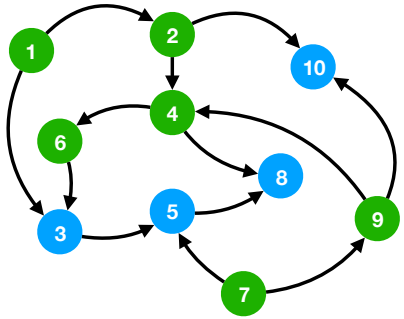
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

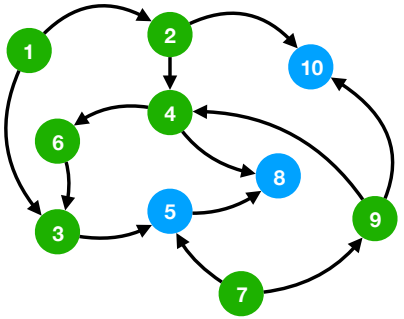
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

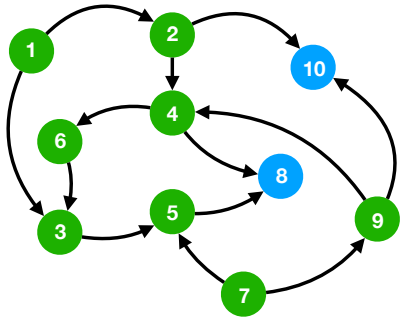
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

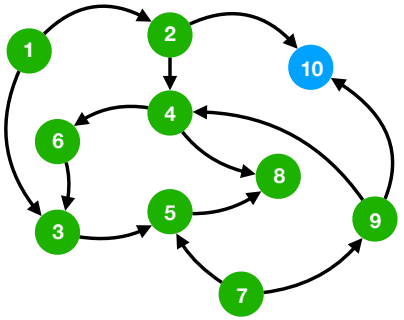
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

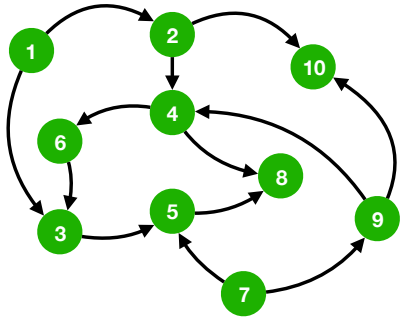
Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Topological sort: check

Are we always only following directed edges?



7, 9, 1, 2, 4, 6, 3, 5, 8, 10

Yes!

Recap & Next Class

Today we learned:

- Greedy algorithms
- Minimum-weight spanning trees
- DAGs
- Topological order

Next class:

- Finish topological sorting algorithm
- Hash tables