Ask a Question

Introduction to Topological Sort

Let's go over the Topological Sort pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following...

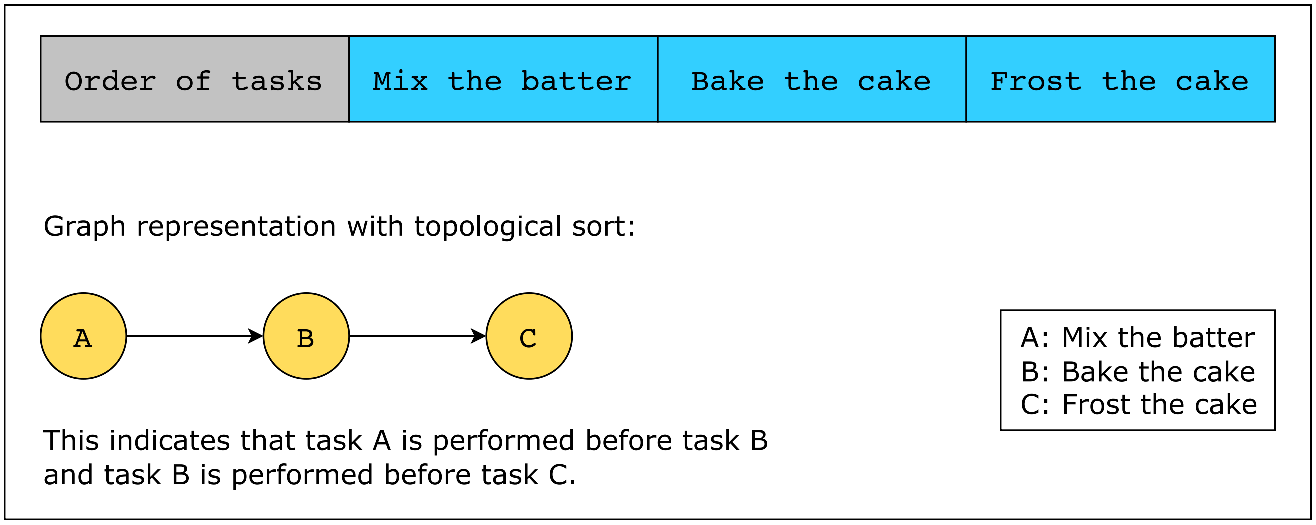
About the pattern

Topological sorting is a technique used to organize a collection of items or tasks based on their dependencies. Imagine there is a list of tasks to complete, but some tasks can only be done after others are finished. There are many such tasks that depend on each other, or there's a specific sequence of actions that must be followed. For example, when baking a cake, there are several steps one needs to follow, and some steps depend on others. We can't frost the cake until it's baked, and we can't bake it until we've mixed the batter. To ensure that we don't frost the cake before baking it or mix the batter after preheating the oven, we need to sort the steps based on their dependencies. This is where topological sorting helps us. It figures out the correct sequence of steps to bake the cake efficiently.

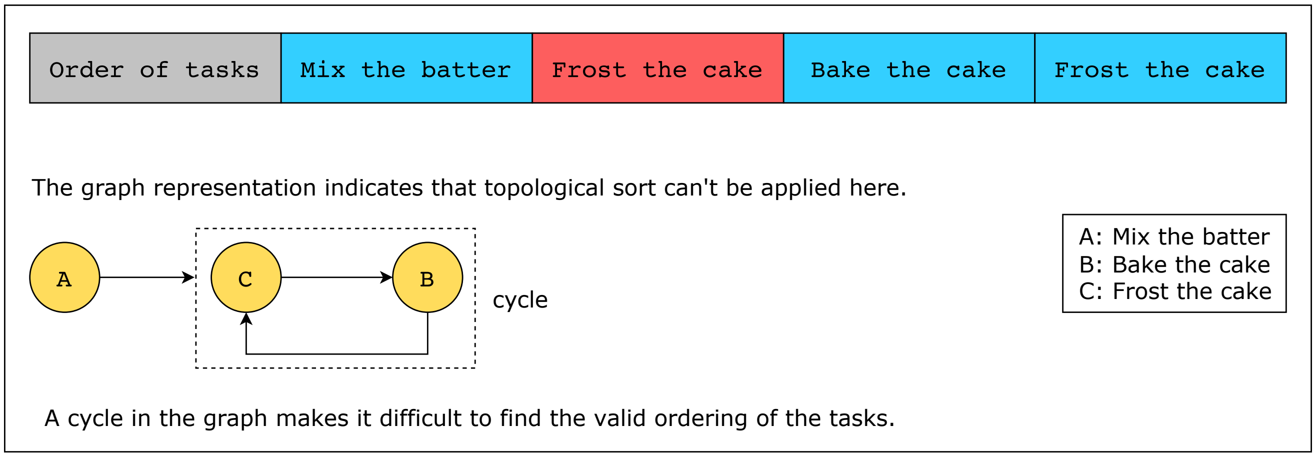
The **topological sort** pattern is used to find valid orderings of elements that have dependencies on or priority over each other. These elements can be represented as the nodes of a graph, so in technical terms, topological sort is a way of ordering the nodes of a directed graph such that for every directed edge $[a, b]$ from node a to node b, a comes befo b in the ordering.

> **Note:** Topological sort is only applicable to directed acyclic graphs (DAGs), meaning there should be no cycles present in the graph.

If we write a recipe for baking a cake, then the list of tasks goes like first mix the batter, then bake the cake, and finally, frost it. These tasks can also be organized in a graph, where each task is a node, and the dependencies between tasks are represented by directed edges.


Order of tasks
Mix the batter
Bake the cake
Frost the cake
Graph representation with topological sort:
A
B
C
A: Mix the batter
B: Bake the cake
C: Frost the cake
This indicates that task A is performed before task B and task B is performed before task C.

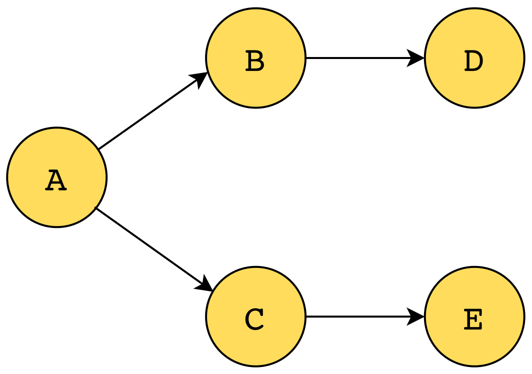
However, if we mistakenly add an additional step in our recipe that contradicts any of the existing steps, it introduces a cycle in our graph. For example, if the recipe goes like mix the batter, frost the cake, bake the cake, and frost the cake, we can't frost a cake that hasn't been baked and can't bake a cake that's already frosted. Similarly, in a graph with cycles, we can't establish a clear order of tasks, making topological sorting impossible. Therefore, topological sorting is only applicable to directed acyclic graphs (DAGs), where tasks are organized in a logical sequence without any contradictions or cycles.


Order of tasks
Mix the batter
Frost the cake
Bake the cake
Frost the cake
The graph representation indicates that topological sort can't be applied here.
A
C
B
cycle
A: Mix the batter
B: Bake the cake
C: Frost the cake
A cycle in the graph makes it difficult to find the valid ordering of the tasks.

Topological sorting is crucial for converting a partial ordering to a complete ordering, especially when certain tasks have defined dependencies while others do not. For instance, consider a research project involving five tasks, labeled as A, B, C, D, and E. Some tasks depend on others: Task A must be completed before Task B and Task C, Task B must be finished before the Task D, and Task C must be done before Task E. However, when it comes to performing Tasks B and C, it's unclear which one should be done first. This is where topological sorting comes in. It helps us convert this partial ordering of tasks into a complete ordering, providing clarity on the sequence of tasks to ensure efficient execution. There can be multiple possible valid ordering of the elements. For example, based on the given dependencies in our example, some valid orderings of the tasks could be:

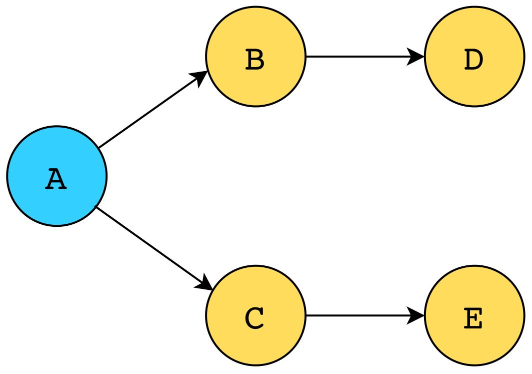
- A, B, C, D, E
- A, C, B, D, E
- A, B, C, E, D

Now, let's have a visual demonstration of the example we discussed above:

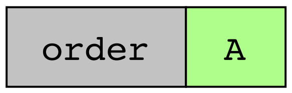


Let's apply topological sorting to find a valid and complete order for performing these tasks.

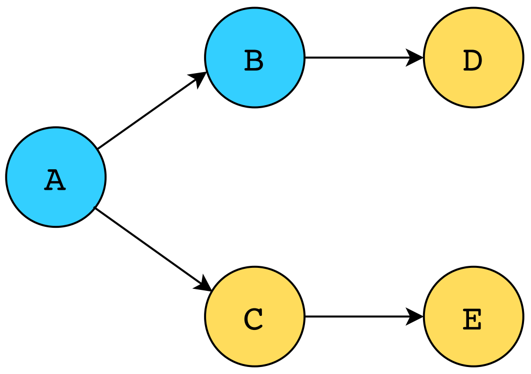
1 of 7



First, task A will be performed, so add it the order list.



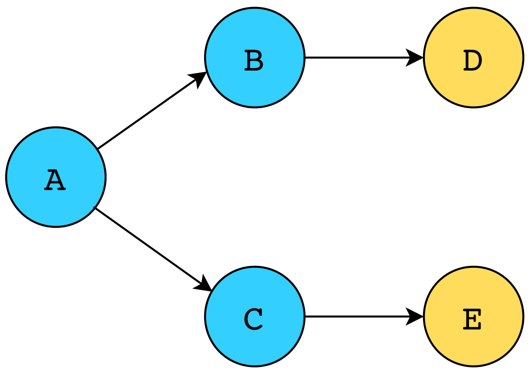
2 of 7



Next, any of the tasks B or C can be performed, so add one of it to the order list.

order	A	B

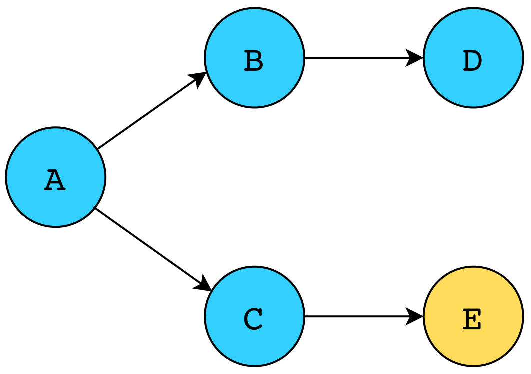
3 of 7



Similarly, any of the tasks C or D can be performed, so add one of it to the order list.

order	A	B	C

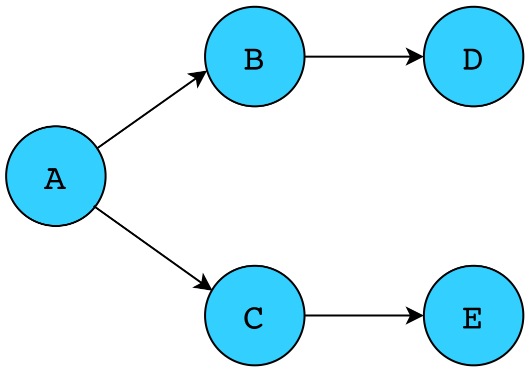
4 of 7



Next, any of the tasks D or E can be performed, so add one of it to the order list.

order	A	B	C	D

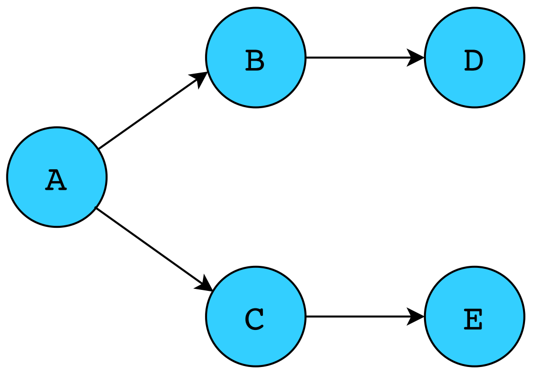
5 of 7



Now, add task E to the order list.

order	A	B	C	D	E

6 of 7



We now have a defined order for performing these tasks.

Note: This is one of many possible valid orderings.

order	A	B	C	D	E

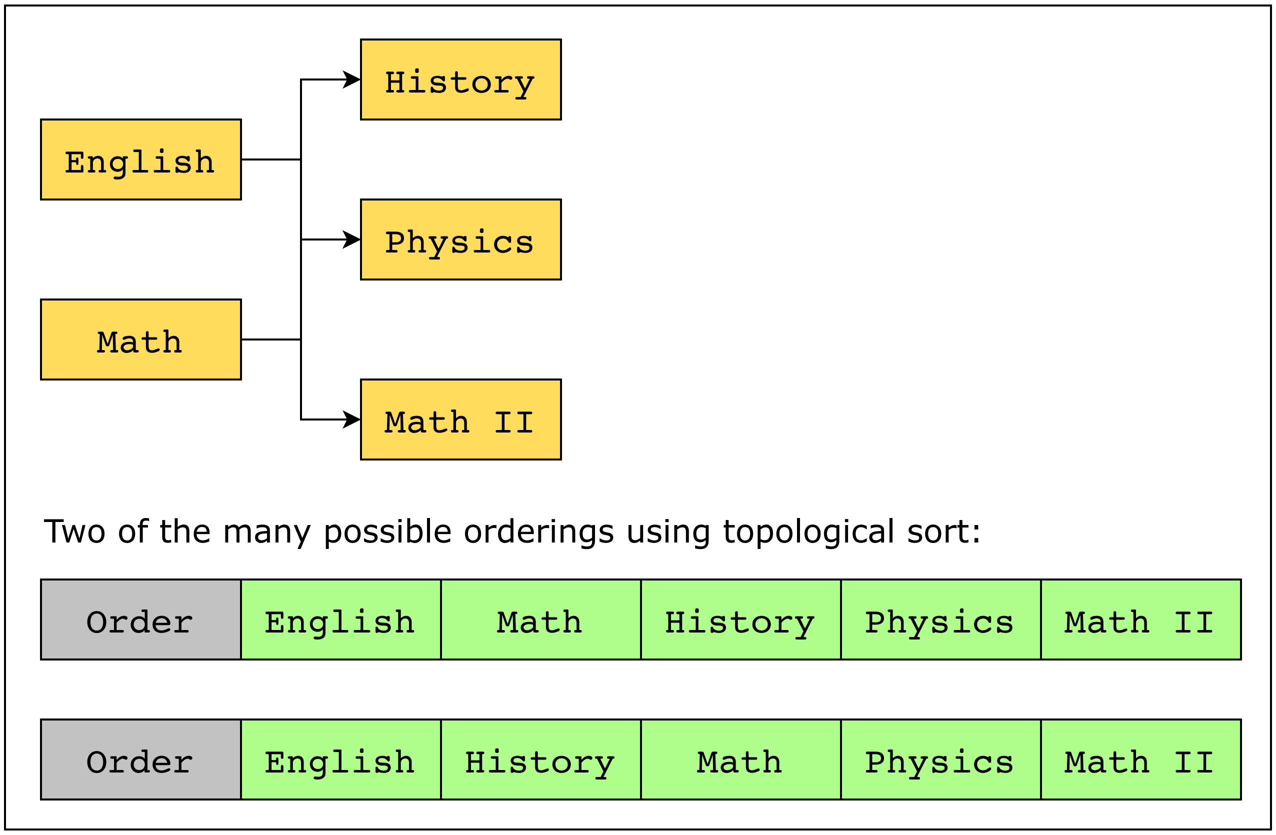
7 of 7

Because topological sort can be applied to DAGs, it's important to convert a non-DAG input to a DAG before solving the problem.

Examples

The following examples illustrate some problems that can be solved with this approach:

1. **Course prerequisites:** Given a list of courses, each with a prerequisite course, determine if it's possible to complete all the courses and, if so, provide a valid order in which the courses can b taken.

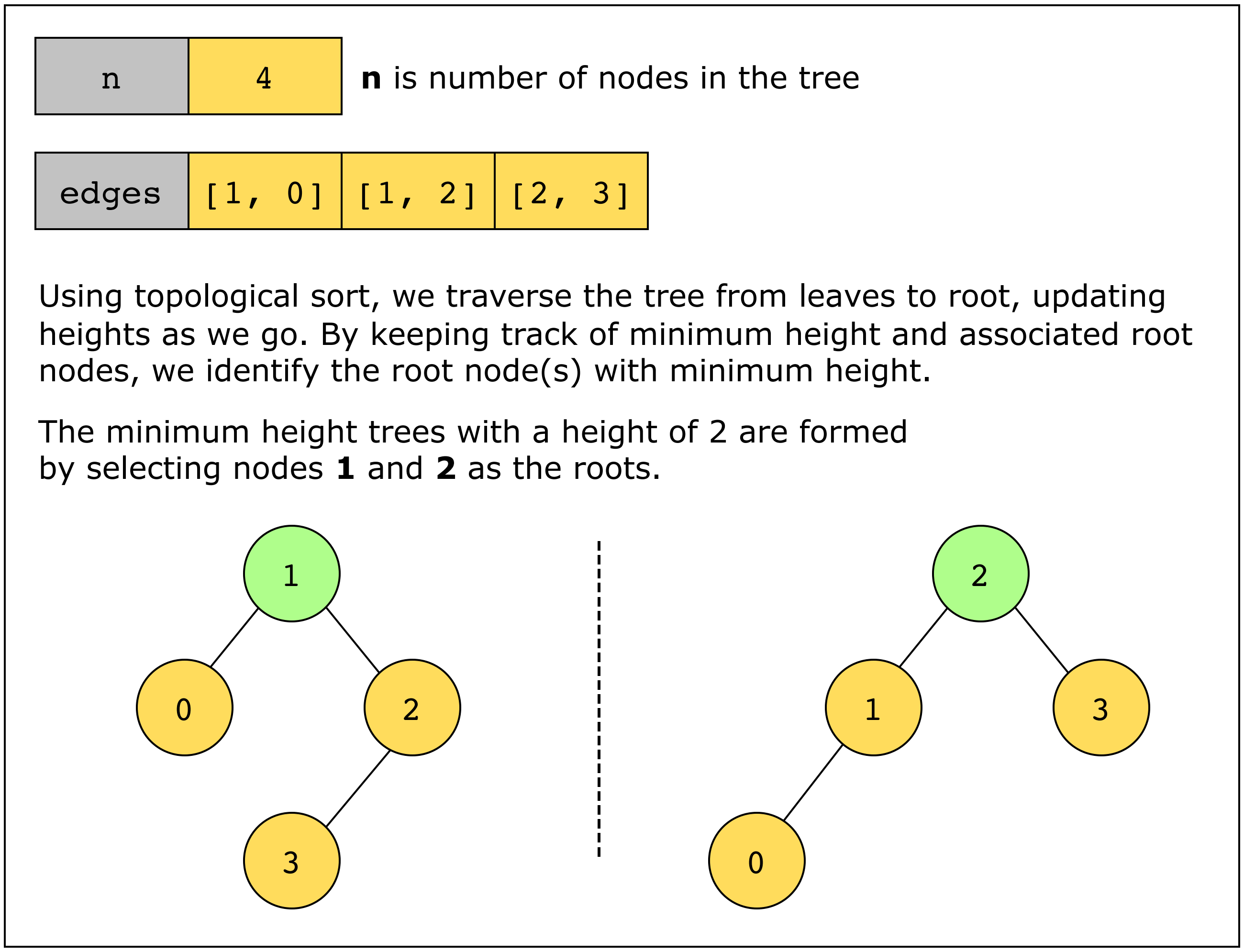


Two of the many possible orderings using topological sort:

Order	English	Math	History	Physics	Math II

Order	English	History	Math	Physics	Math II

2. **Minimum height trees:** Given an input tree, we can redraw it with different nodes as the root. Of all these trees, we want to find the root node of the tree with the minimum height. Return multiple nodes if multiple trees with different root nodes have the same minimum height.



n	4

n is number of nodes in the tree

edges	[1, 0]	[1, 2]	[2, 3]

Using topological sort, we traverse the tree from leaves to root, updating heights as we go. By keeping track of minimum height and associated root nodes, we identify the root node(s) with minimum height.

The minimum height trees with a height of 2 are formed by selecting nodes **1** and **2** as the roots.

Does your problem match this pattern?

- Yes, if either of these conditions is fulfilled:
    - **Dependency relationships:** The problem involves tasks, jobs, courses, or elements with dependencies between them. These dependencies create a partial order, and topological sorting can be used to establish a total order based on these dependencies.
    - **Ordering or sequencing:** The problem requires determining a valid order or sequence to perform tasks, jobs, or activities, considering their dependencies or prerequisites.
- No, if either of these conditions is fulfilled:
    - **Presence of cycles:** If the problem involves a graph with cycles, topological sorting cannot be applied because there is no valid linear ordering of vertices that respects the cyclic dependencies.

- ◦ **Dynamic dependencies:** If the dependencies between elements change dynamically during the execution of the algorithm, topological sorting may not be suitable. Topological sorting assumes static dependencies that are known beforehand.

Real-world problems

Many problems in the real world share the topological sort pattern. Let’s look at some examples.

- **Course scheduling**: In academic institutions, students need to enroll in courses based on prerequisites. Topological sorting helps determine the order in which courses should be taken to ensure that students meet the prerequisites for each course.
- **Recipe planning in cooking**: Recipes often have steps that must be performed in a specific order. For example, a cake can’t be baked until we’ve mixed the batter. Topological sorting can help plan the steps of a recipe to ensure that they are performed in the correct sequence.
- **Process scheduling in computer systems:** During system boot-up, the operating system needs to initiate various processes, some of which depend on others. These dependencies are represented as ordered pairs. Circular dependencies are not allowed. The operating system selects an order for executing processes, ensuring that each process’s dependencies are met before execution.

Strategy time!

Match the problems that can be solved using the topological sort pattern.

> **Note**: Select a problem in the left-hand column by clicking it, and

> then click one of the two options in the right-hand column.

Match The Answer

Select an option from the left-hand side

Given a sample of words in an alien language, deduce the order of the letters in their alphabet.	Topological Sort
Find the number of	Some other pattern

?

Tт