Topological Sort; Reductions

CSE 373 Winter 2020

Instructor: Hannah C. Tang

Teaching Assistants:

Aaron Johnston	Ethan Knutson	Nathan Lipiarski
Amanda Park	Farrell Fileas	Sam Long
Anish Velagapudi	Howard Xiao	Yifan Bai
Brian Chan	Jade Watkins	Yuma Tou
Elena Spasova	Lea Quan	

Poll Everywhere

pollev.com/uwcse373

- Approximately how long did Homework 6 (A* Search) take?

A. 0-3 hours
B. 4-7 hours
C. 8-11 hours
D. 12-15 hours
E. 16-19 hours
F. 19+ hours
G. I don't want to say …

Announcements

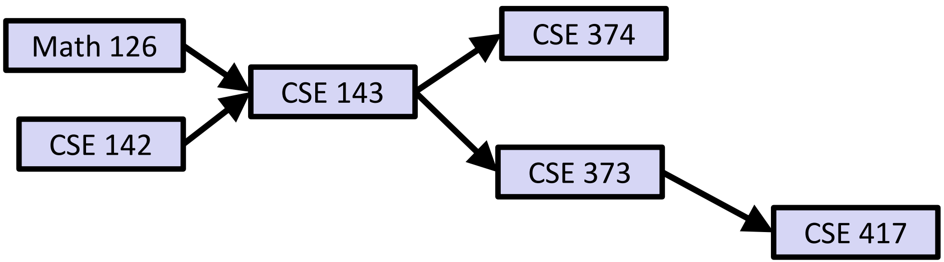
- We are converting Friday's Workshop (12:30-1:20 @ CSE 203) into DIT
- My DIT on Friday is in CSE 303 (not my office)
- Extra time for today's reading quiz due to a broken link

Lecture Outline

- **Topological Sorting**
- General Technique: Reductions
  - Reducing Shortest Path in a negative DAG to TopoSort
  - Reducing Seam Carving to A* Search

Sorting Dependencies

- Given a set of courses and their prerequisites, find an order to take the courses in, assuming you can only take one course per quarter



Topological Sort (aka Topological Ordering)

- Example: dependency graphs
  - An edge (u, v) means u must happen before v
  - A topological sort of a dependency graph gives an ordering that **respects dependencies**

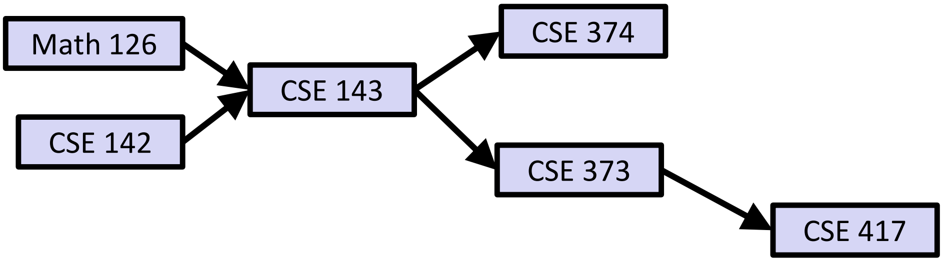
- Applications:
  - Graduating
  - Compiling multiple Java files
  - Multi-job Workflows

Topological Sort

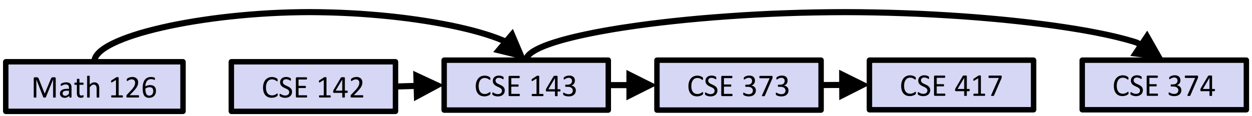
- Given a directed graph G
- Find an ordering of the vertices so all edges go from "left to right"

Sorting Dependencies

- Our course prerequisite chart:

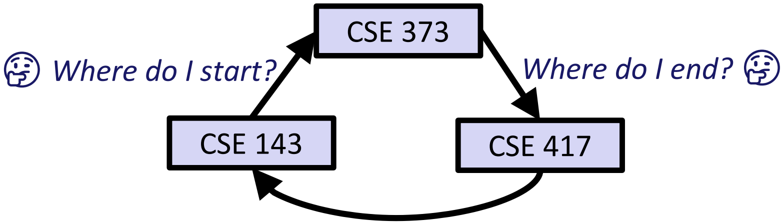


- Possible ordering:



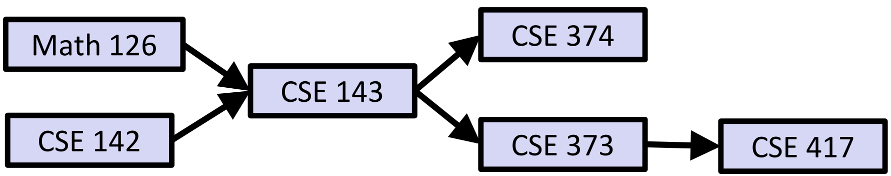
Can We Always TopoSort a Graph?

- Can you topologically sort this graph?



No

- What's the difference between this graph and our first graph?



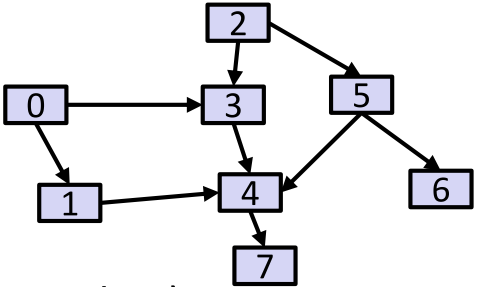
- A graph has a topological ordering iff it is a DAG
  - But a DAG can have multiple orderings

Directed Acyclic Graph

- A directed graph without any cycles
- Edges may or may not be weighted

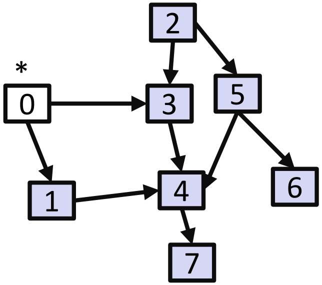
Sorting a DAG

- Does this graph have a topological ordering? If so find one

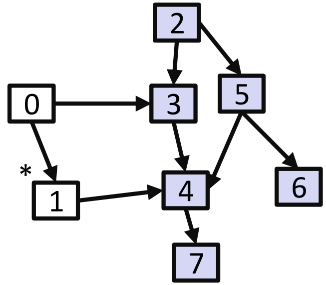


- Algorithm (from reading):
  - Perform DFS post-order traversal(s) from every vertex with in-degree 0, *remembering marked vertices between traversals*

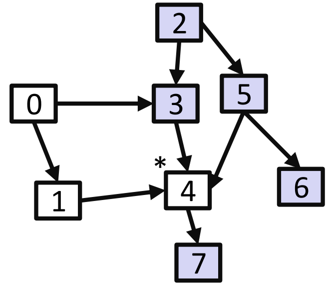
Sorting a DAG



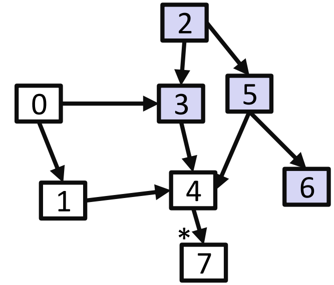
Post-order: []



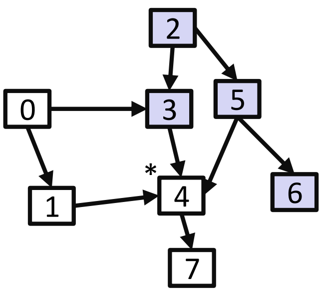
Post-order: []



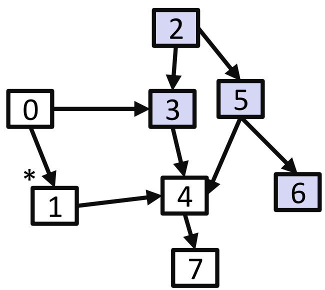
Post-order: []



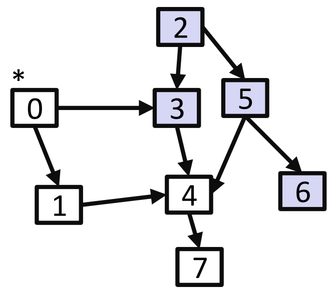
Post-order: [7]



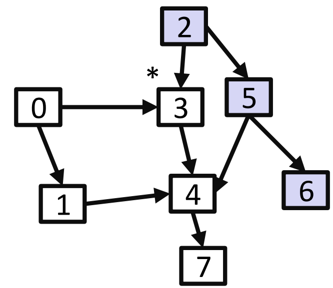
Post-order: [7, 4]



Post-order: [7, 4, 1]



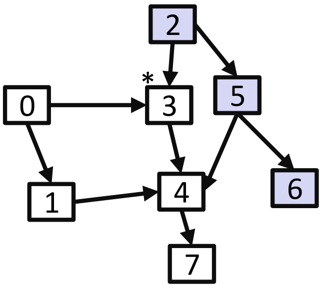
Post-order: [7, 4, 1]



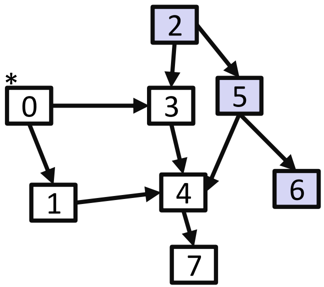
Post-order: [7, 4, 1, 3]

Sorting a DAG

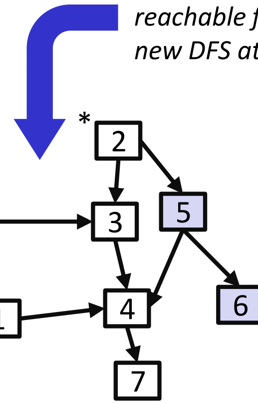
At this step, we've visited everything reachable from node 0 and are starting a new DFS at a node with an in-degree of 0



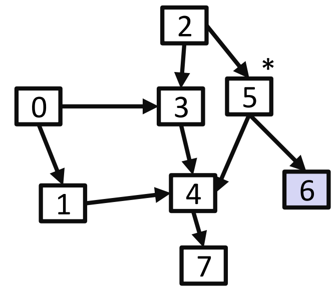
Post-order: [7, 4, 1, 3]



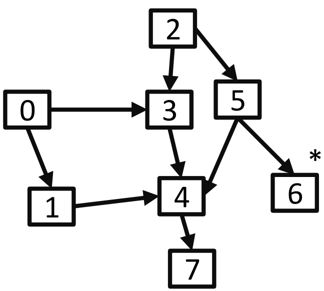
Post-order: [7, 4, 1, 3, 0]



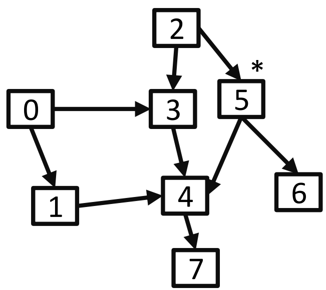
Post-order: [7, 4, 1, 3, 0]



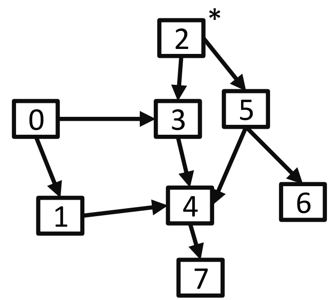
Post-order: [7, 4, 1, 3, 0]



Post-order: [7, 4, 1, 3, 0, 6]



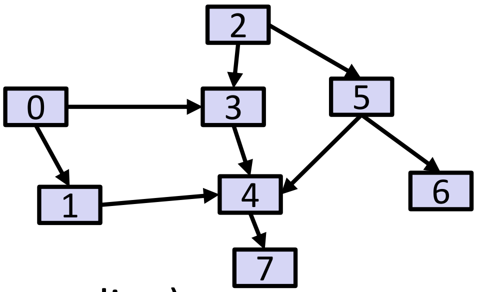
Post-order: [7, 4, 1, 3, 0, 6, 5]



Post-order: [7, 4, 1, 3, 0, 6, 5, 2]

How Did We Find a Topological Sort?

- Does this graph have a topological ordering? If so find one

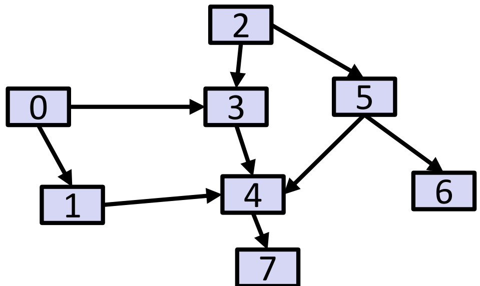


- Algorithm (from reading):
  - Perform DFS post-order traversal(s) from every vertex with in-degree 0, *remembering marked vertices between traversals*

- Topological Sort is given by reverse of the DFS post-order list
  - Post-order list: [7, 4, 1, 3, 0, 6, 5, 2]
  - Reversed: [2, 5, 6, 0, 3, 1, 4, 7]

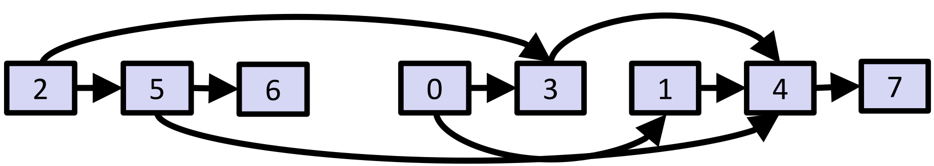
How Did We Find a Topological Sort?

- Does this graph have a topological ordering? If so find one



- Topological Sort is given by reverse of the DFS post-order list
  - Post-order list: [7, 4, 1, 3, 0, 6, 5, 2]
  - Reversed: [2, 5, 6, 0, 3, 1, 4, 7]

- Our ”topological sort”:



Lecture Outline

- Topological Sorting
- **General Technique: Reductions**
  - Reducing Shortest Path in a negative DAG to TopoSort
  - Reducing Seam Carving to A* Search

Decomposition vs Reduction

- **Decomposition**: Taking a complex task and breaking it into smaller parts
  - This is the heart of computer science and programming
  - We might use "canned solutions" for solving these smaller parts

- **Reduction**: Using an algorithm for Problem Q to solve a different Problem P
  - We do not modify the algorithm for Problem Q; we only modify the inputs/outputs
  - "Problem P reduces to Q"
  - Hopefully problem Q is easier than problem P!

Decomposition

- **Decomposition**: Taking a complex task and breaking it into smaller parts

- Examples:
  - Decomposing HuskyMaps to:
    - Autocomplete (search bar prefix queries)
    - Heap (priority queue for route-finding)
    - A* search (route-finding)
    - k-d tree (nearest neighbors to find start and goal vertices)
    - Rasterer (map tile display)
  - Recognizing that “finding the next trie node” meant using a Map ADT

Reduction

- **Reduction**: Using an algorithm for Problem Q to solve a different Problem P

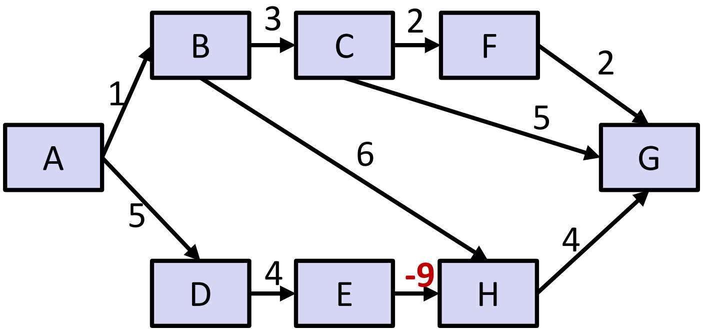
- Examples and Non-examples:
  - A* does not reduce to Dijkstra's
    - We *modified* Dijkstra's by adding a heuristic
  - "Climbing a hill" reduces to:
    - "Riding a ski lift"
    - "Being shot out of a cannon"
    - "Riding a bike"
  - Shortest path in a negative DAG reduces to Topological Sort
  - … does Topological Sort reduce to DFS? Yes!

Lecture Outline

- Topological Sorting
- General Technique: Reductions
  - **Reducing Shortest Path in a negative DAG to TopoSort**
  - Reducing Seam Carving to A* Search

Shortest Path in a Negative DAG

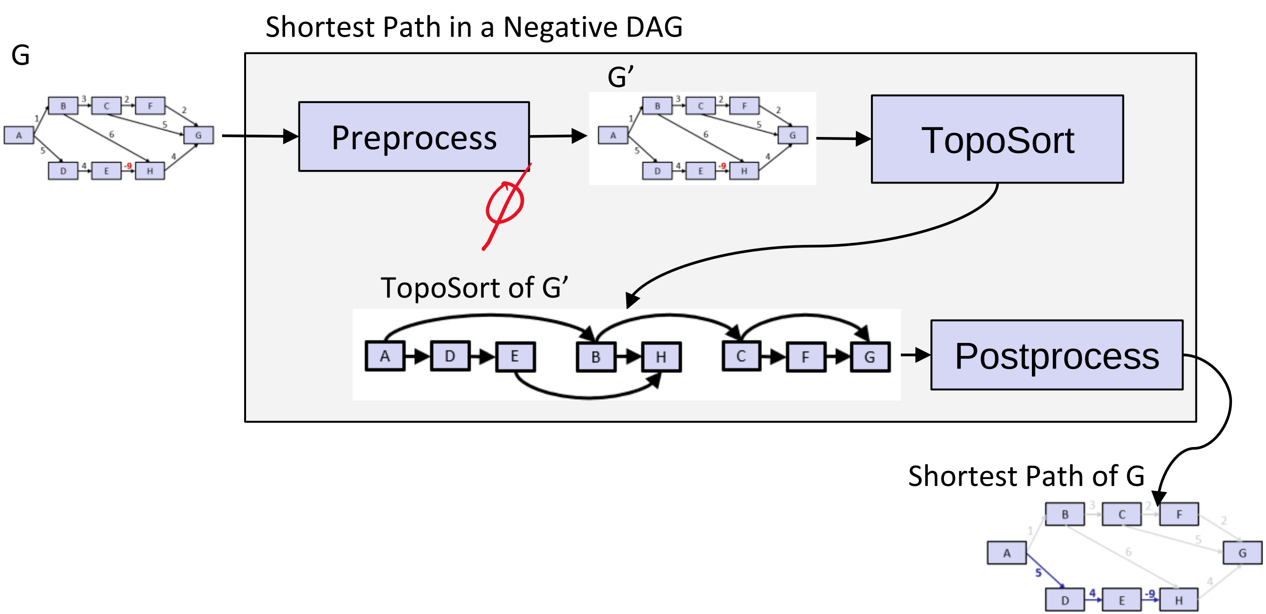
- Recall this graph from our reading:



- We can reduce this problem to TopoSort!

Reductions in 3 Easy Steps!

1. Transform the input so that it can be solved by the standard algorithm
2. Run the standard algorithm as-is on the transformed input
3. Transform the output of the algorithm to solve the original problem



Lecture Outline

- Topological Sorting

- General Technique: Reductions
  - Reducing Shortest Path in a negative DAG to TopoSort
  - **Reducing Seam Carving to A* Search**

Content-Aware Image Resizing

Seam carving: A distortion-free technique for resizing an image by removing "unimportant seams"

Original Photo

Horizontally-Scaled
(castle and person are distorted)

Seam-Carved
(castle and person are undistorted; "unimportant" sky removed instead)

Seam carving for content-aware image resizing (Avidan, Shamir/ACM); Broadway Tower (Newton2, Yummifruitbat/Wikimedia)

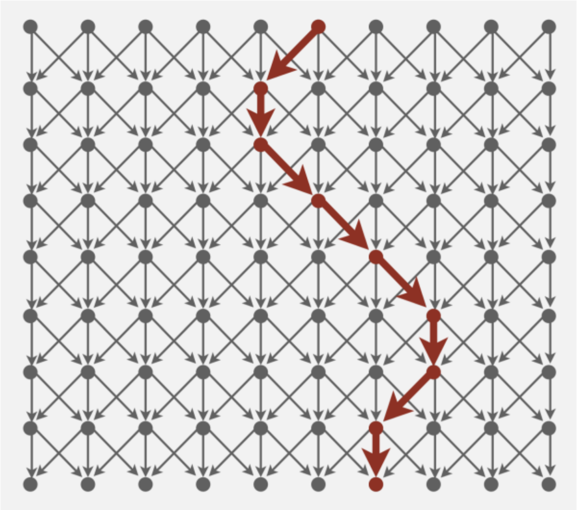
Demo: https://www.youtube.com/watch?v=vIFCV2spKtg

Seam Carving Reduces to A* Search

1. *Transform the input so that it can be solved by the standard algorithm*
   - Formulate the image as a graph
     - **Vertices**: pixel in the image
     - **Edges**: connects a pixel to its 3 downward neighbors
     - **Edge Weights**: the "energy" (visual difference) between adjacent pixels

2. *Run the standard algorithm as-is on the transformed input*
   - Run A* Search to find the shortest path (sum of weights) from top row to bottom row

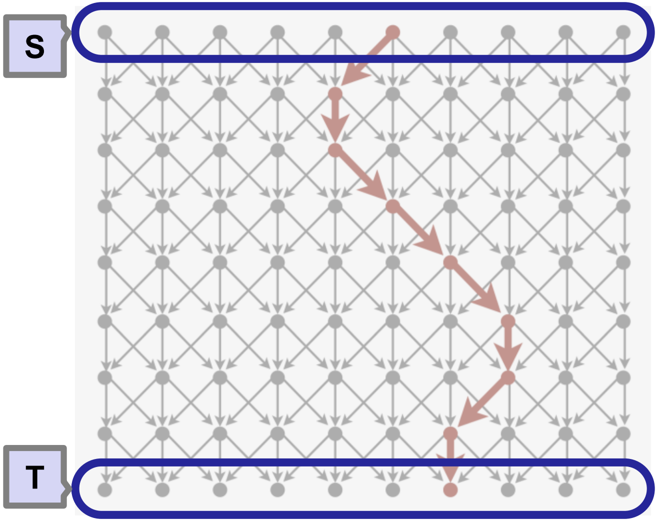
3. *Transform the output of the algorithm to solve the original problem*
   - Interpret the path as a removable "seam" of unimportant pixels



Shortest Paths (Robert Sedgewick, Kevin Wayne/Princeton)

Formal Problem Statement

- Using AStarSolver, find the seam from any top vertex to any bottom vertex
- Given a digraph with positive edge weights and two distinguished subsets of vertices S and T, find a shortest path from any vertex in S to any vertex in T
- Your algorithm should run in time proportional to E log V in the worst case

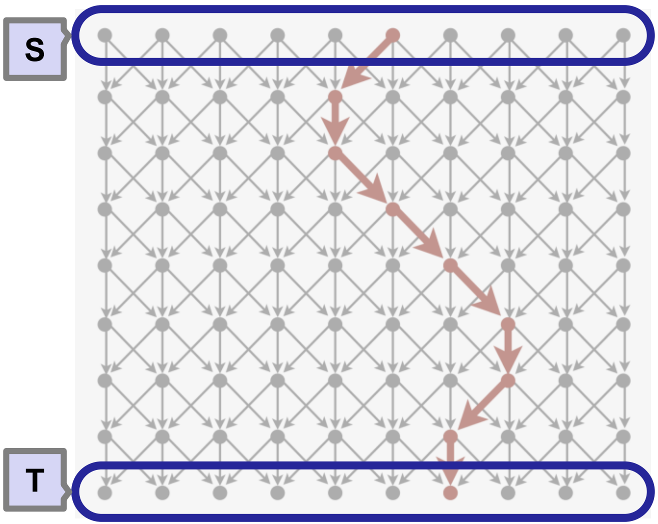


Shortest Paths (Robert Sedgewick, Kevin Wayne/Princeton)

An Incomplete Reduction

- A* Search starts with a single vertex S and ends with a single vertex T
  - This problem specifies *sets of vertices* for the start and end

- **Your turn**: brainstorm how to transform this graph into something A* Search knows how to operate on



Shortest Paths (Robert Sedgewick, Kevin Wayne/Princeton)

tl;dr

- TopoSort is widely applicable; many problems can be *decomposed* into a dependency graph

- But problems can also be *reduced* to TopoSort!

- Reductions are a powerful tool in theoretical computer science
  - You won't have to do a reduction on the final, but you should be aware of the general principles