LEC 17

CSE 373

Topo Sort & Reductions

BEFORE WE START

Instructor: Hunter Schafer

TAs:
Ken Aragon
Khushi Chaudhari
Joyce Elauria
Santino Iannone
Leona Kazi
Nathan Lipiarski
Sam Long
Amanda Park
Paul Pham
Mitchell Szeto
Batina Shikhalieva
Ryan Siu
Elena Spasova
Alex Teng
Blarry Wang
Aileen Zeng

Learning Objectives

After this lecture, you should be able to...

1. Describe the runtime for Dijkstra's algorithm and explain where it comes from
2. Define a topological sort and determine whether a given problem could be solved with a topological sort
3. Write code to produce a topological sort and identify valid and invalid topological sorts for a given graph
4. Explain the makeup of a reduction, identify whether algorithms are considered reductions, and solve a problem using a reduction to a known problem

Lecture Outline

- Dijkstra's Algorithm
  - *Review* Definition & Examples
  - Implementing Dijkstra's
- Topological Sort
- Reductions
  - Definitions
  - Examples

Review Our Graph Problem Collection

WED

s-t Connectivity Problem

Given source vertex **s** and a target vertex **t**, does there exist a path from **s** to **t**?

SOLUTION
Base Traversal: BFS or DFS
Modification: Check if each vertex == t

WED

Unweighted Shortest Path Problem

Given source vertex **s** and target vertex **t**, what path from s to t **minimizes the number of edges**? How long is that path, and what edges make it up?

SOLUTION
Base Traversal: BFS
Modification: Generate shortest path tree as we go

FRI

Weighted Shortest Path Problem

Given source vertex **s** and target vertex **t**, what path from s to t **minimizes the total weight of its edges**? How long is that path, and what edges make it up?

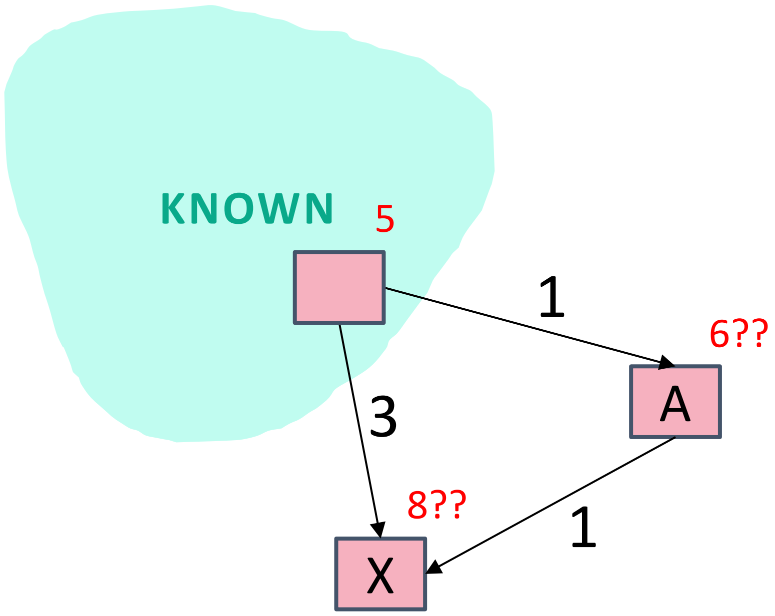
SOLUTION
Base Traversal: Dijkstra's Algorithm
Modification: Generate shortest path tree as we go

Review Dijkstra's Algorithm: Key Properties

- Once a vertex is marked known, its shortest path is known
  - Can reconstruct path by following back-pointers (in edgeTo map)
- While a vertex is not known, another shorter path might be found
  - We call this update **relaxing** the distance because it only ever shortens the current best path
- Going through closest vertices first lets us confidently say no shorter path will be found once known
  - Because not possible to find a shorter path that uses a farther vertex we'll consider later

```
dijkstraShortestPath(G graph, V start)
  Set known; Map edgeTo, distTo;
  initialize distTo with all nodes mapped to ∞, except start to 0

  while (there are unknown vertices):
    let u be the closest unknown vertex
    known.add(u)
    for each edge (u,v) to unknown v with weight w:
      oldDist = distTo.get(v)      // previous best path to v
      newDist = distTo.get(u) + w  // what if we went through u?
      if (newDist < oldDist):
        distTo.put(v, newDist)
        edgeTo.put(v, u)
```

Review Why Does Dijkstra's Work?



Example:

- We're about to add X to the known set
- But how can we be sure we won't later find a path through some node A that is shorter to X?

INVARIANT

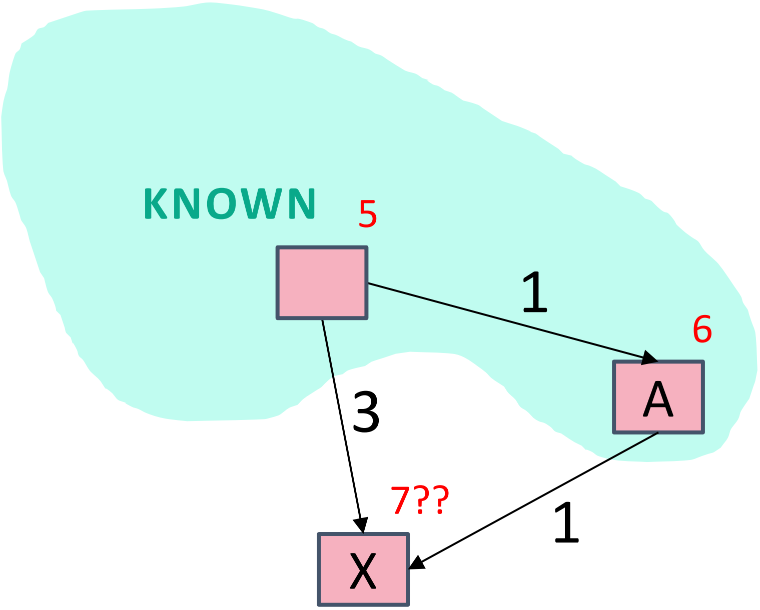
Dijkstra's Algorithm Invariant

All vertices in the "known" set have the correct shortest path

- Similar "First Try Phenomenon" to BFS

- How can we be sure we won't find a shorter path to X later?

Review Why Does Dijkstra's Work?



Example:
- We're about to add X to the known set
- But how can we be sure we won't later find a path through some node A that is shorter to X?
- Because *if we could, Dijkstra's would explore A first*

INVARIANT

Dijkstra's Algorithm Invariant

All vertices in the "known" set have the correct shortest path

- Similar "First Try Phenomenon" to BFS
- How can we be sure we won't find a shorter path to X later?
  - **Key Intuition**: Dijkstra's works because:
  - IF we always add the closest vertices to "known" first,
  - THEN by the time a vertex is added, any possible relaxing has happened and the path we know is *always the shortest*!

Lecture Outline

- Dijkstra's Algorithm
  - *Review* Definition & Examples
  - Implementing Dijkstra's
- Topological Sort
- Reductions
  - Definitions
  - Examples

Implementing Dijkstra's

- How do we implement "let u be the closest unknown vertex"?

- Would sure be convenient to store vertices in a structure that…
  - Gives them each a distance "priority" value
  - Makes it fast to grab the one with the smallest distance
  - Lets us update that distance as we discover new, better paths

MIN PRIORITY QUEUE ADT

```
dijkstraShortestPath(G graph, V start)
  Set known; Map edgeTo, distTo;
  initialize distTo with all nodes mapped to ∞, except start to 0

  while (there are unknown vertices):
    let u be the closest unknown vertex
    known.add(u)
    for each edge (u,v) to unknown v with weight w:
      oldDist = distTo.get(v)      // previous best path to v
      newDist = distTo.get(u) + w  // what if we went through u?
      if (newDist < oldDist):
        distTo.put(u, newDist)
        edgeTo.put(u, v)
```

Implementing Dijkstra's: Pseudocode

- Use a MinPriorityQueue to keep track of the perimeter
  - Don't need to track entire graph
  - Don't need separate "known" set – implicit in PQ (we'll never try to update a "known" vertex)
- This pseudocode is much closer to what you'll implement in P4
  - However, still some details for you to figure out!
  - e.g. how to initialize distTo with all nodes mapped to ∞
  - Spec will describe some optimizations for you to make ☺

```
dijkstraShortestPath(G graph, V start)
  Map edgeTo, distTo;
  initialize distTo with all nodes mapped to ∞, except start to 0

  PriorityQueue<V> perimeter; perimeter.add(start);

  while (!perimeter.isEmpty()):
    u = perimeter.removeMin()

    for each edge (u,v) to v with weight w:
      oldDist = distTo.get(v)      // previous best path to v
      newDist = distTo.get(u) + w  // what if we went through u?
      if (newDist < oldDist):
        distTo.put(v, newDist)
        edgeTo.put(v, u)
        if (perimeter.contains(v)):
          perimeter.changePriority(v, newDist)
        else:
          perimeter.add(v, newDist)
```

Dijkstra's Runtime

$\Theta(|V|)$ → initialize distTo

$\Theta(|V|\log|V|)$ = $\Theta(|V|)$ iterations (while loop), $\Theta(\log|V|)$ (removeMin)

total $\Theta(|E|)$ iterations (for each edge loop)

$\Theta(1)$ (oldDist through edgeTo.put)

$\Theta(\log|V|)$ (changePriority)

$\Theta(\log|V|)$ (add)

$\Theta(|E|\log|V|)$

```
dijkstraShortestPath(G graph, V start)
  Map edgeTo, distTo;
  initialize distTo with all nodes mapped to ∞, except start to 0

  PriorityQueue<V> perimeter; perimeter.add(start);

  while (!perimeter.isEmpty()):
    u = perimeter.removeMin()

    for each edge (u,v) to v with weight w:
      oldDist = distTo.get(v)       // previous best path to v
      newDist = distTo.get(u) + w   // what if we went through u?
      if (newDist < oldDist):
        distTo.put(v, newDist)
        edgeTo.put(v, u)
        if (perimeter.contains(v)):
          perimeter.changePriority(v, newDist)
        else:
          perimeter.add(v, newDist)
```

Dijkstra's Runtime

Final result:

$\Theta(|V|\log|V| + |E|\log|V|)$

Why can't we simplify further?

- We don't know if |V| or |E| is going to be larger, so we don't know which term will dominate.
- *Sometimes we assume |E| is larger than |V|, so |E|log|V| dominates. But not always true!*

$\Theta(|V|\log|V|)$: $\Theta(|V|)$ iterations, $\Theta(\log|V|)$

$\Theta(|E|\log|V|)$: total $\Theta(|E|)$ iterations, $\Theta(1)$, $\Theta(\log|V|)$, $\Theta(\log|V|)$

$\Theta(|V|)$

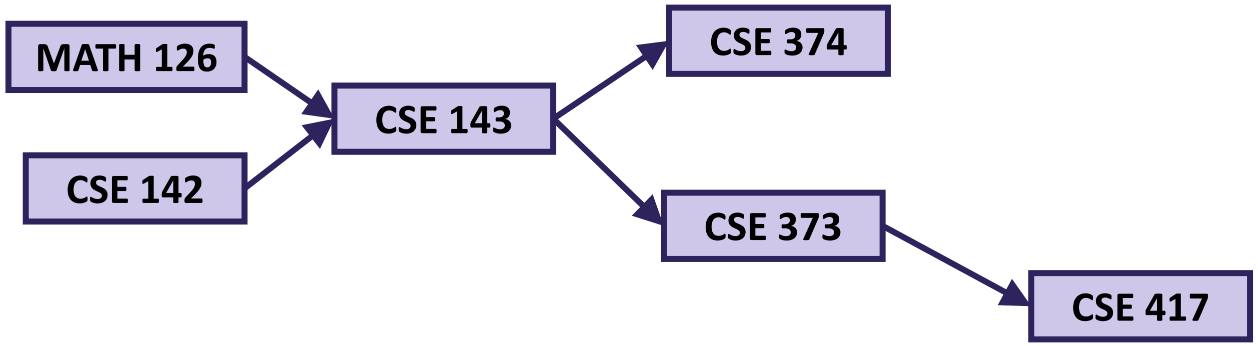
```
dijkstraShortestPath(G graph, V start)
  Map edgeTo, distTo;
  initialize distTo with all nodes mappe

  PriorityQueue<V> perimeter; perimeter.

  while (!perimeter.isEmpty()):
    u = perimeter.removeMin()

    for each edge (u,v) to v with weight
      oldDist = distTo.get(v)      // pr
      newDist = distTo.get(u) + w  // wh
      if (newDist < oldDist):
        distTo.put(v, newDist)
        edgeTo.put(v, u)
        if (perimeter.contains(v)):
          perimeter.changePriority(v, ne
        else:
          perimeter.add(v, newDist)
```

Lecture Outline

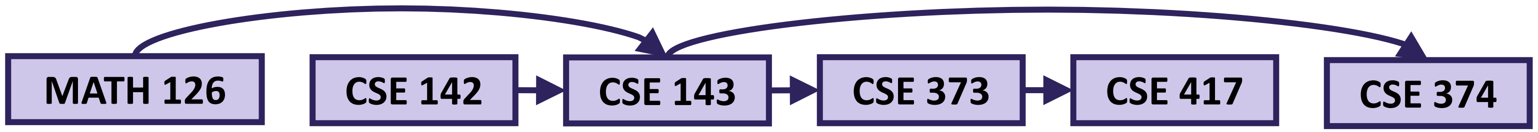
- Dijkstra's Algorithm
  - *Review* Definition & Examples
  - Implementing Dijkstra's
- **Topological Sort**
- Reductions
  - Definitions
  - Examples

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)



- Possible ordering:



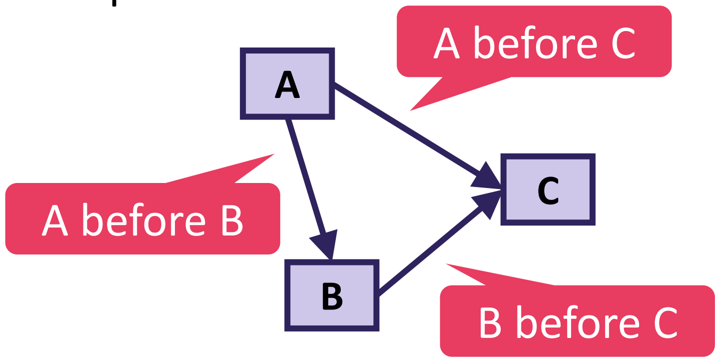
Topological Sort

- A **topological sort** of a directed graph G is an ordering of the nodes, where for every edge in the graph, the origin appears before the destination in the ordering

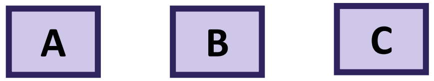
- Intuition: a “dependency graph”
  - An edge (u, v) means u must happen before v
  - A topological sort of a dependency graph gives an ordering that **respects dependencies**

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

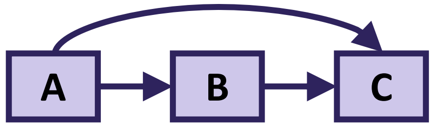
Input:



Topological Sort:

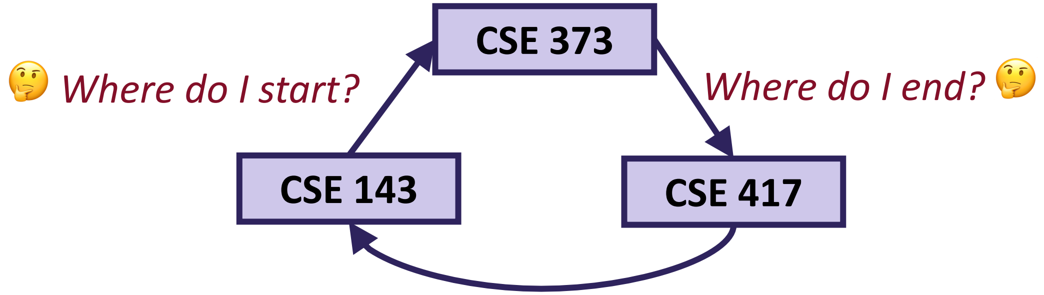


With original edges for reference:



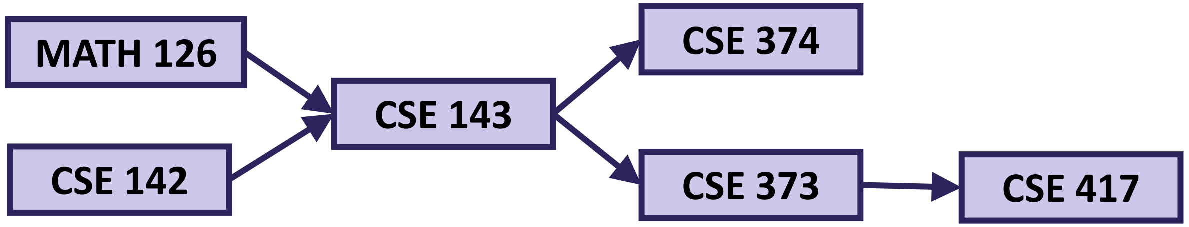
Can We Always Topo Sort a Graph?

- 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

How To Perform Topo Sort?

- Topo sort is an ordering problem. Could we use… BFS?

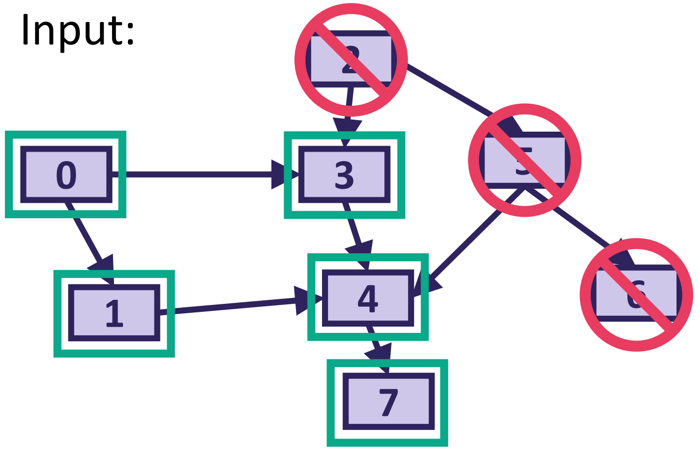
IDEA 1

Performing Topo Sort

Use BFS, starting from a vertex with no incoming edges

Doesn't reach all vertices ☹

Input:



BFS starting at 0:

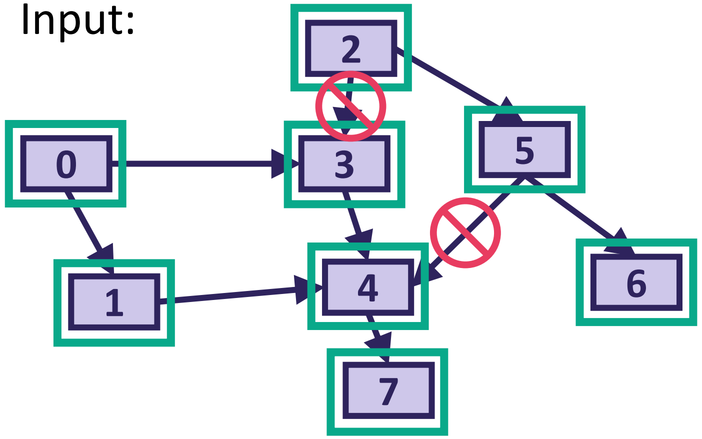
How To Perform Topo Sort?

- Okay, there may be multiple "roots". What if we use BFS multiple times?

IDEA 2 **Performing Topo Sort**

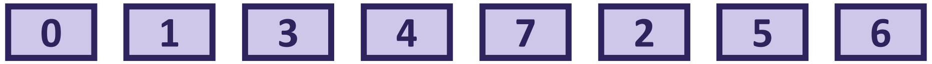
Use BFS, starting from ALL vertices with no incoming edges

Input:



BFS starting at 0:

+ BFS starting at 2:



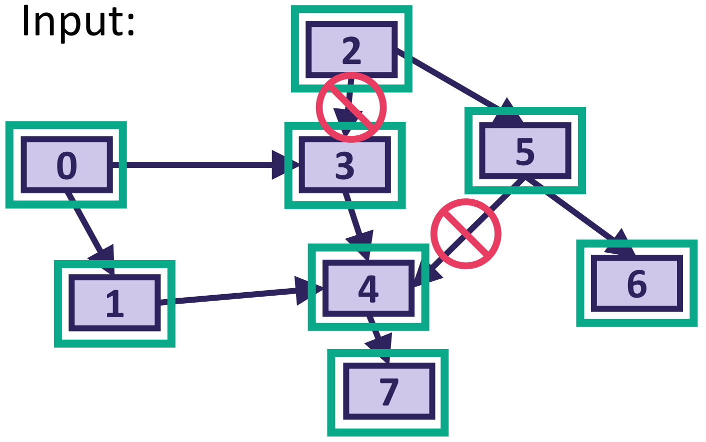
Does this idea work? Why or why not?

- Okay, there may be multiple "roots". What if we use BFS multiple times?

IDEA 2 Performing Topo Sort

Use BFS, starting from ALL vertices with no incoming edges

Input:



BFS starting at 0:

+ BFS starting at 2:

0	1	3	4	7	2	5	6

How To Perform Topo Sort?

- Okay, there may be multiple “roots”. What if we use BFS multiple times?

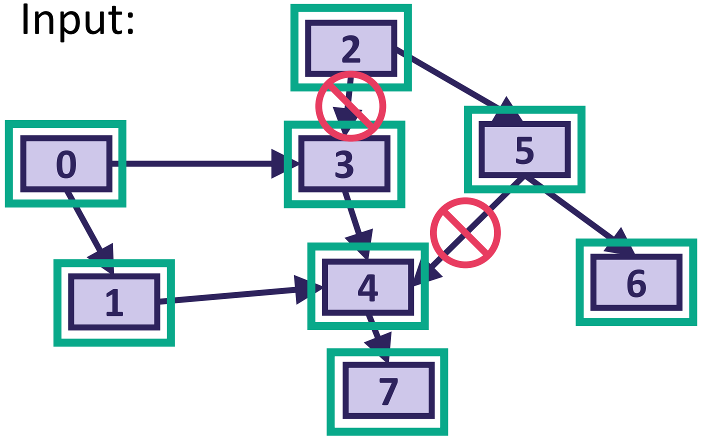
IDEA 2

Performing Topo Sort

Use BFS, starting from ALL vertices with no incoming edges

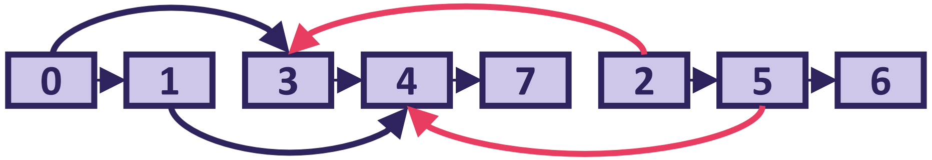
Doesn’t respect all edges ☹

Input:



BFS starting at 0:

+ BFS starting at 2:



CSE 373

- Home
- Projects
- Exercises
- Exams
- Office Hours
- Staff
- Syllabus

Course Tools

- Zoom
- Ed
- Gradescope
- GitLab
- Anonymous Feedback

Fri 11/06 LEC 16 Dijkstra's Algorithm

Lesson: videos › pdf pptx

Class Session: zoom handout solution

P3 Heaps

RELEASED

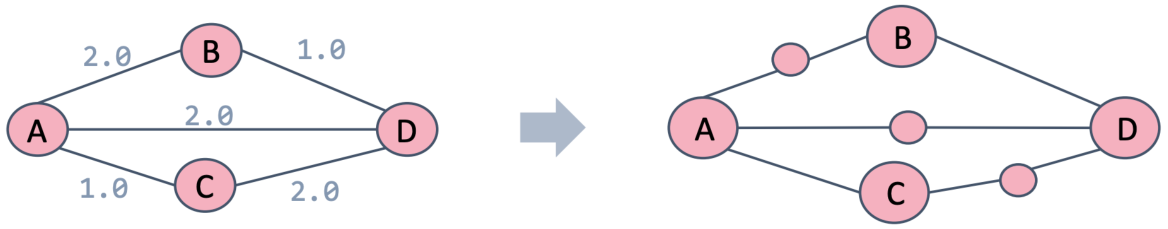
Week 7

UNIVERSITY of WASHINGTON LEC 16: Dijkstra's Algorithm CSE 373 Autumn 2020

Idea 1: Change into an unweighted graph

- We know BFS works on unweighted graphs
  - If we can transform a weighted graph to unweighted, we can solve it!

- This idea is known as a **reduction**
  - "Reduce" a problem you can't solve to one you can
  - Here, we're trying to reduce BFS on weighted graphs to BFS on unweighted graphs
  - We'll revisit this concept later in the course!



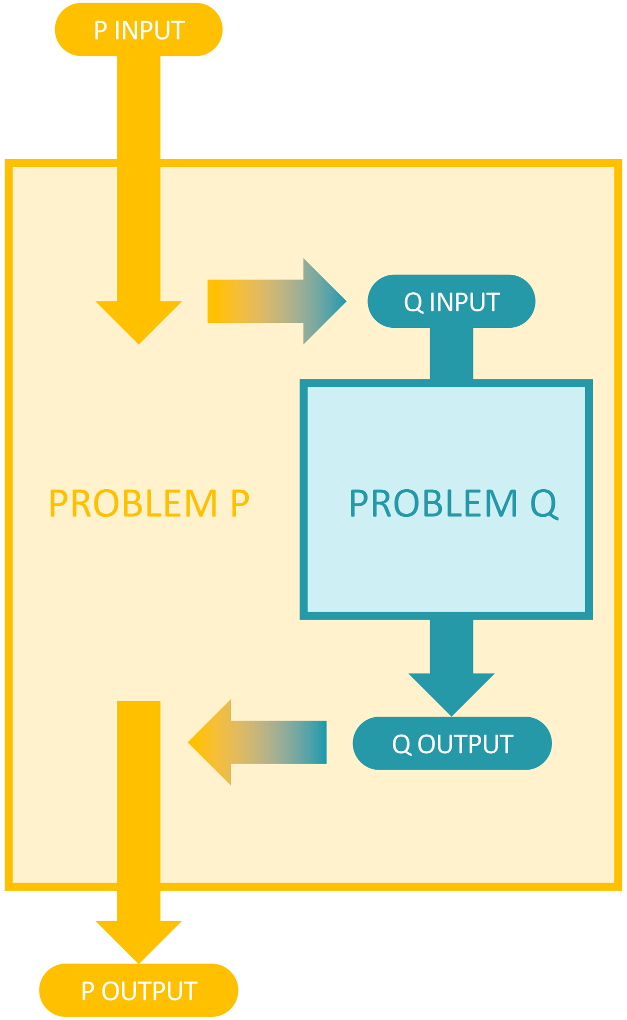
Lecture Outline

- Dijkstra's Algorithm
  - *Review* Definition & Examples
  - Implementing Dijkstra's
- Topological Sort
- **Reductions**
  - **Definitions**
  - Examples

Reductions

- A **reduction** is a problem-solving strategy that involves using an algorithm for problem Q to solve a different problem P
  - Rather than modifying the algorithm for Q, we **modify the inputs/outputs** to make them compatible with Q!
  - "P reduces to Q"

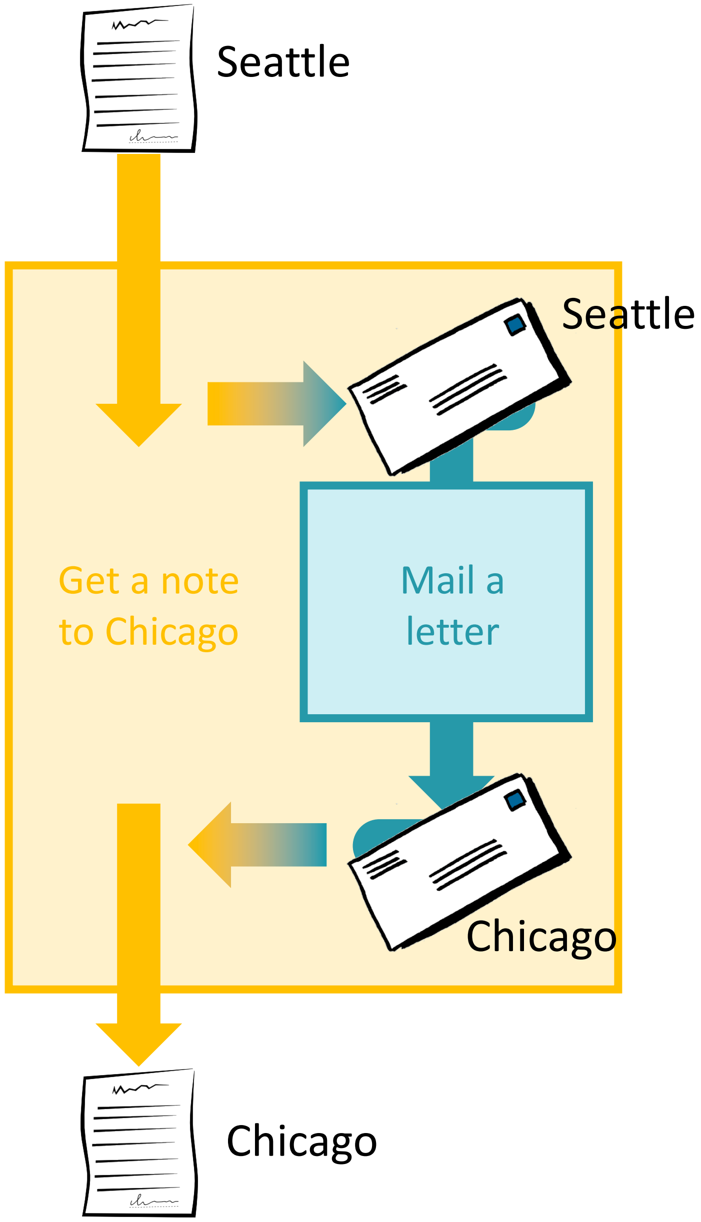
1. Convert input for P into input for Q
2. Solve using algorithm for Q
3. Convert output from Q into output from P



Reductions

- **Example**: I want to get a note to my friend in Chicago, but walking all the way there is a difficult problem to solve ☹
  - Instead, **reduce** the "get a note to Chicago" problem to the "mail a letter" problem!

1. Place note inside of envelope
2. Mail using US Postal Service
3. Take note out of envelope



How To Perform Topo Sort?

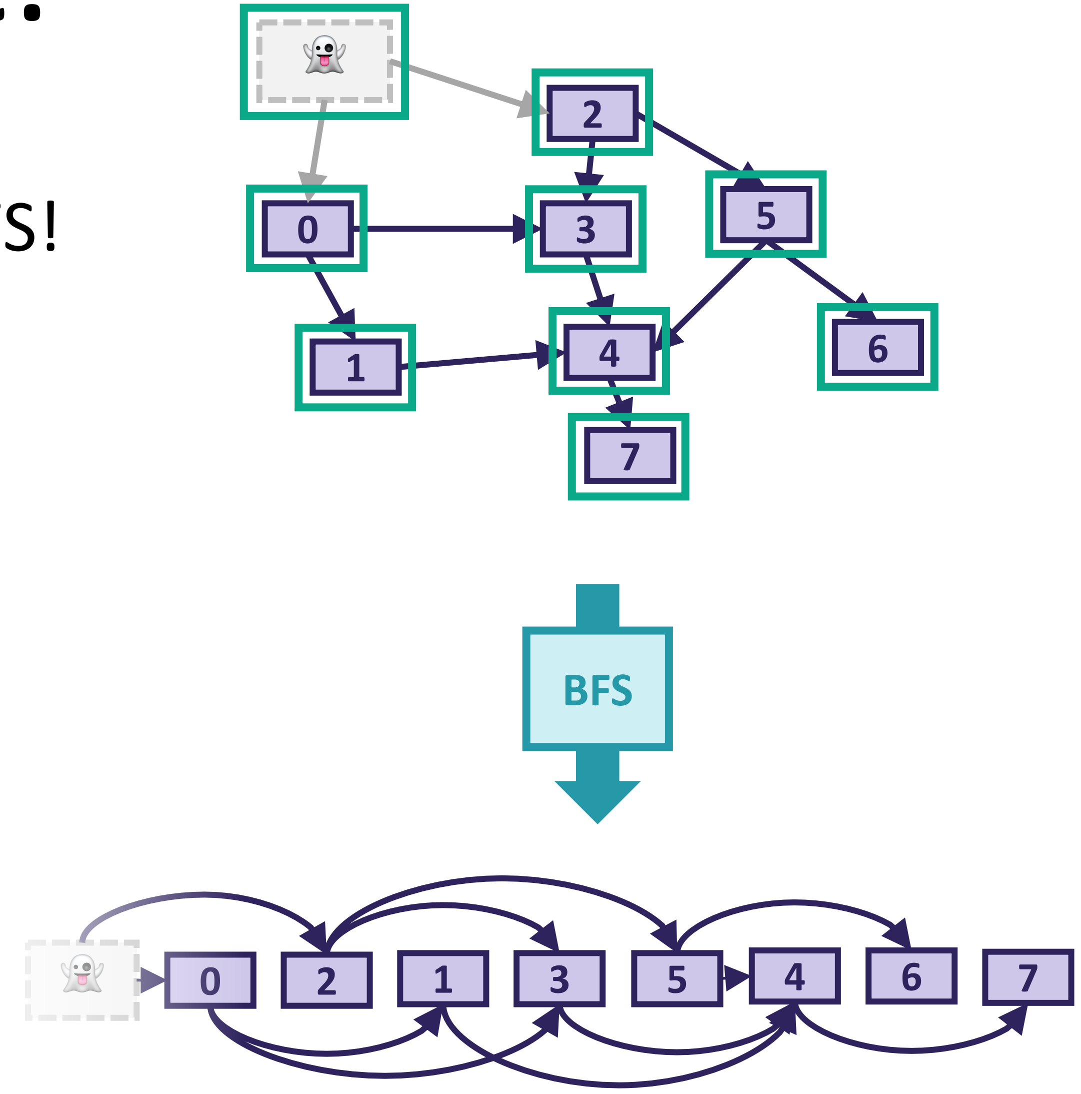
- If we add a phantom "start" vertex pointing to other starts, we could use BFS!

IDEA 3

Performing Topo Sort

Reduce topo sort to BFS by modifying graph, running BFS, then modifying output back

Sweet sweet victory 😎



Reductions

- A **reduction** is a problem-solving strategy that involves using an algorithm for problem Q to solve a different problem P
  - Rather than modifying the algorithm for Q, we **modify the inputs/outputs** to make them compatible with Q!
  - "P reduces to Q"

1. Convert input for P into input for Q
2. Solve using algorithm for Q
3. Convert output from Q into output from P

Did we reduce Unweighted Shortest Paths (USP) to BFS?

a) Yes.
USP reduces to BFS.

b) Yes.
BFS reduces to USP.

c) No.
This is not a reduction.

In a reduction, we modify inputs/outputs, not the algorithm itself!

Lecture Outline

- Comparison Sorts
  - *Review* Sorting Overview
  - In-Place Quick Sort
- Topological Sort
- **Reductions**
  - Definitions
  - **Examples**

Checking for Duplicates

- Problem: We want to determine whether an array contains duplicate elements.

- Initial idea: Compare every element to every other element!
  - Runtime: $\theta(n^2)$

0	1	2	3	4
2	4	8	3	8

```
containsDuplicates(array) {
  for (int i = 0; i < array.length; i++):
    for (int j = i; j < array.length; j++):
      if (array[i] == array[j]):
        return true
  return false
}
```

- Could we do better?

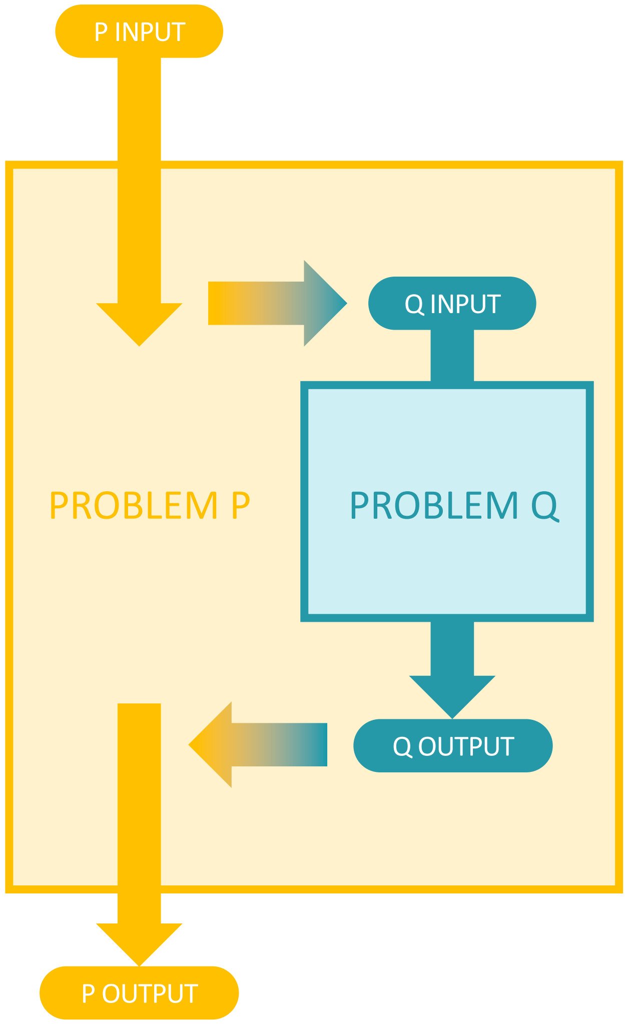
Goal of a Reduction

Goal: Reduce the problem of “Contains Duplicates?” to another problem we have an algorithm for.

Try to identify each of the following:

1. How will you convert the “Contains Duplicates?” input?
2. What algorithm will you apply?
3. How will you convert the algorithm’s output?

0	1	2	3	4
2	4	8	3	8



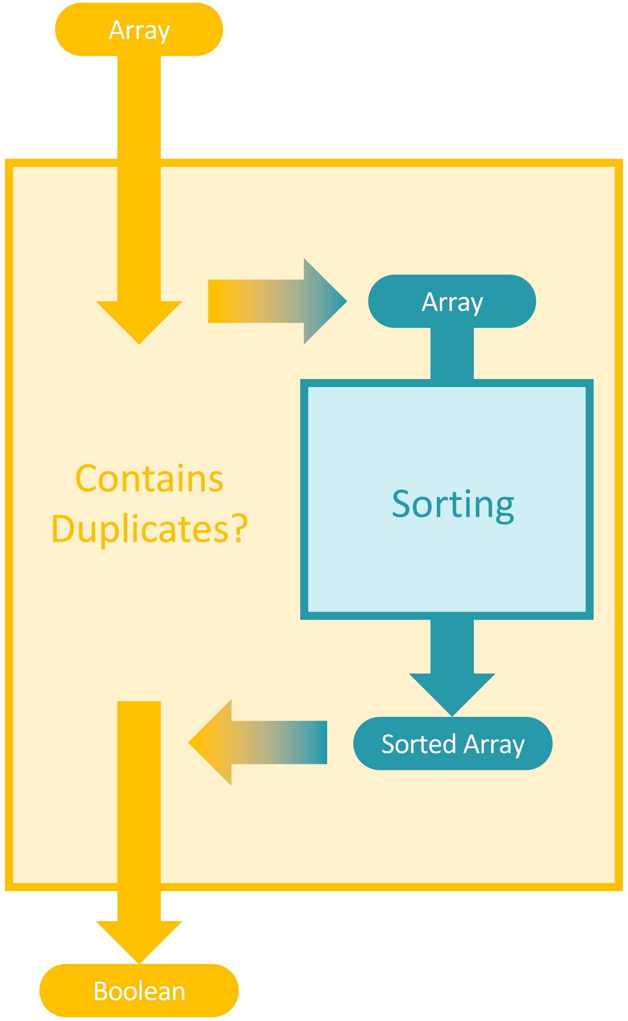
One Solution: Sorting!

One Solution: Reduce "Contains Duplicates?" to the problem of *sorting an array*

- We know several algorithms that solve this problem quickly!

1. Simply pass array input to "Sorting"
2. Use Heap Sort, Merge Sort, or Quick Sort to sort
3. Scan through sorted array: check for duplicates now *next to each other*, a $\theta(n)$ operation!

- Totally okay to do work in input/output conversion! Even with this pass, runtime is $\theta(n \log n + n)$, so just $\theta(n \log n)$. Reduction helped us avoid quadratic runtime!



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

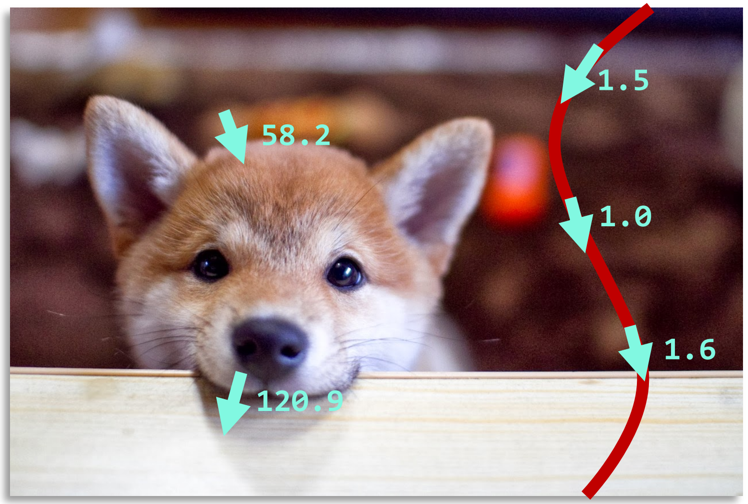
Seam Carving Reduces to Dijkstra's!

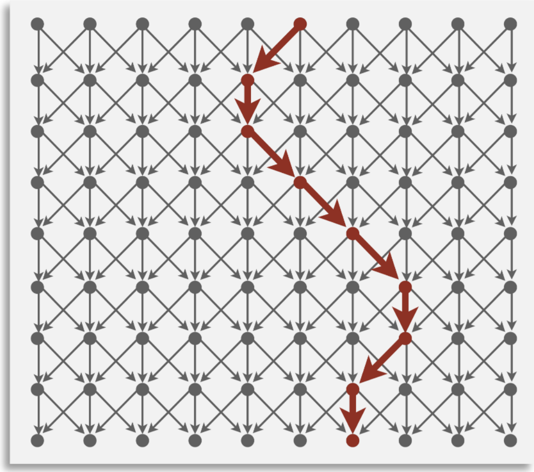
1. *Transform the input so that it can be solved by the standard algorithm*
   - Formulate the image as a graph
     - Vertices: pixel in the image
     - Edges: connects a pixel to its 3 downward neighbors
     - Edge Weights: the "energy" (**visual difference**) between adjacent pixels

2. *Run the standard algorithm as-is on the transformed input*
   - Run Dijkstra's 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

greater pixel difference = higher weight!



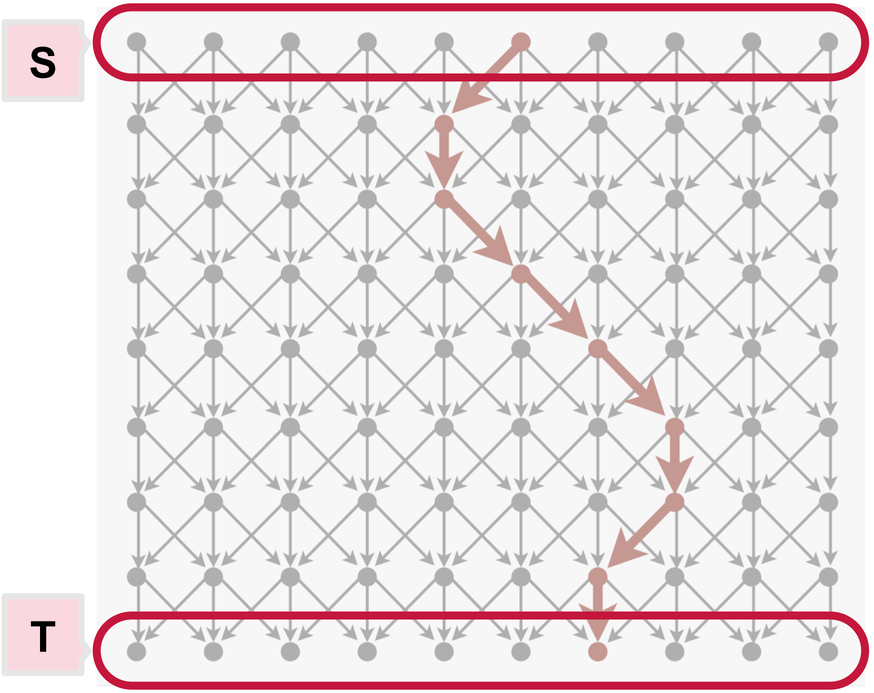


Shortest Paths (Robert Sedgewick, Kevin Wayne/Princeton)

An Incomplete Reduction

- Complication:
  - Dijkstra's starts with a single vertex S and ends with a single vertex T
  - This problem specifies *sets of vertices* for the start and end

- **Question to think about**: how would you transform this graph into something Dijkstra's knows how to operate on?



Shortest Paths (Robert Sedgewick, Kevin Wayne/Princeton)

In Conclusion

- Topo Sort is a widely applicable “sorting” algorithm
- Reductions are an essential tool in your CS toolbox -- you’re probably already doing them without putting a name to it
- Many more reductions than we can cover!
  - Shortest Path in DAG with Negative Edges *reduces to* Topological Sort! (Link)
  - 2-Color Graph Coloring *reduces to* 2-SAT (Link)
  - …
  - Staying on top of the end of the quarter in this course *reduces to* starting early on P4 and EX4/5