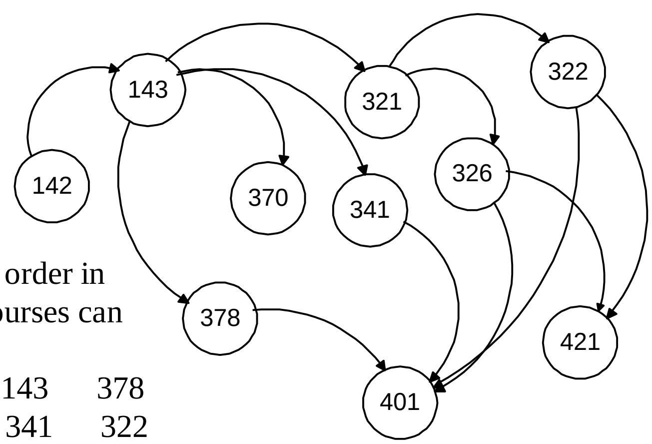
Lecture 20: Topo-Sort and Dijkstra's Greedy Idea

- Items on Today's Lunch Menu:
  - Topological Sort (ver. 1 & 2): Gunning for linear time…
  - Finding Shortest Paths
    - Breadth-First Search
    - Dijkstra's Method: Greed is good!

- Covered in Chapter 9 in the textbook

Graph Algorithm #1: Topological Sort

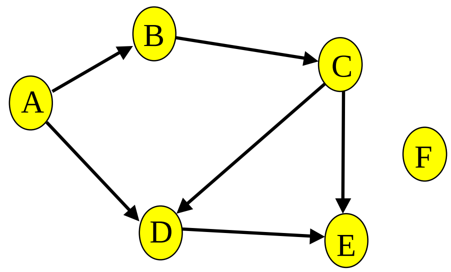


Problem: Find an order in which all these courses can be taken.

Example: 142 143 378 370 321 341 322 326 421 401

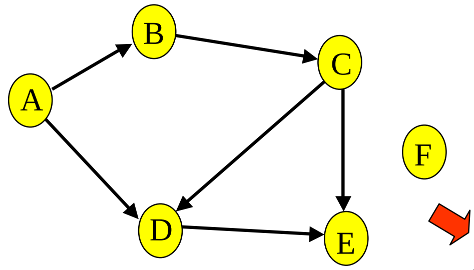
Topological Sort Definition

Topological sorting problem: given digraph $G = (V, E)$, find a linear ordering of vertices such that:
for all edges (v, w) in E, v precedes w in the ordering

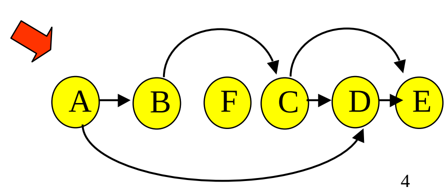


Topological Sort

Topological sorting problem: given digraph $G = (V, E)$, find a linear ordering of vertices such that:
for any edge (v, w) in E, v precedes w in the ordering

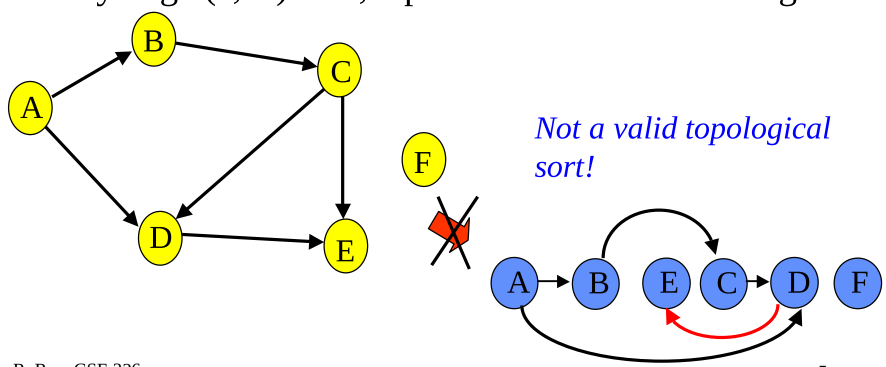


Any linear ordering in which all the arrows go to the right is a valid solution



Topological Sort

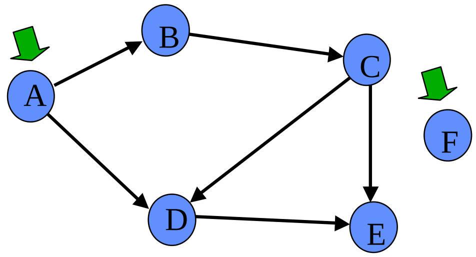
Topological sorting problem: given digraph $G = (V, E)$,
find a linear ordering of vertices such that:
for any edge (v, w) in E, v precedes w in the ordering



Topological Sort Algorithm

Step 1: Identify vertices that have no incoming edge

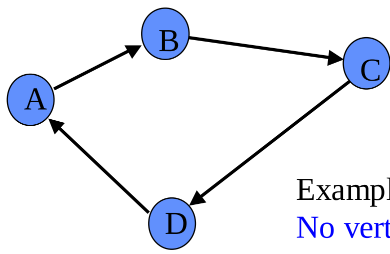
- The "**in-degree**" of these vertices is zero



Topological Sort Algorithm

Step 1: Identify vertices that have no incoming edge

- If *no such edges*, graph has cycles (cyclic graph)

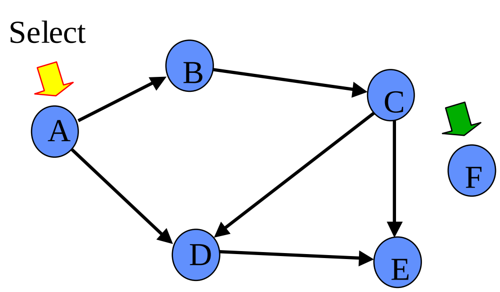


Example of a cyclic graph:
No vertex of in-degree 0

Topological Sort Algorithm

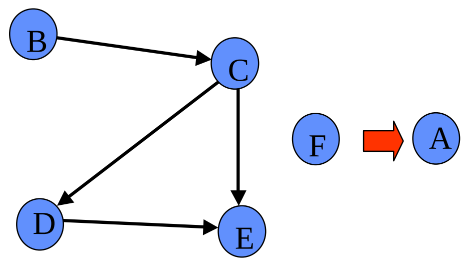
Step 1: Identify vertices that have no incoming edges

- Select one such vertex



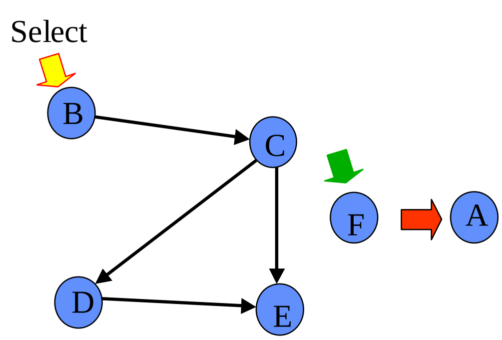
Topological Sort Algorithm

Step 2: Delete this vertex of in-degree 0 and all its outgoing edges from the graph. Place it in the output.



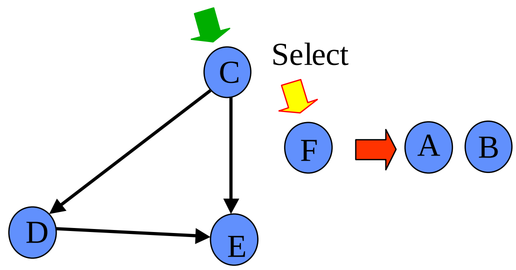
Topological Sort Algorithm

Repeat Steps 1 and Step 2 until graph is empty



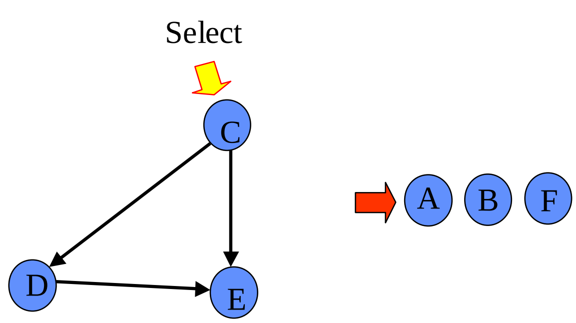
Topological Sort Algorithm

Repeat Steps 1 and Step 2 until graph is empty



Topological Sort Algorithm

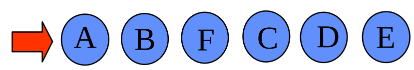
Repeat Steps 1 and Step 2 until graph is empty



Topological Sort Algorithm

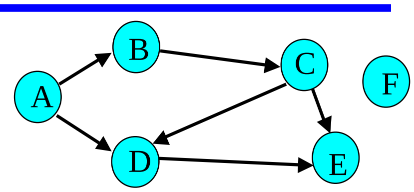
Repeat Steps 1 and Step 2 until graph is empty

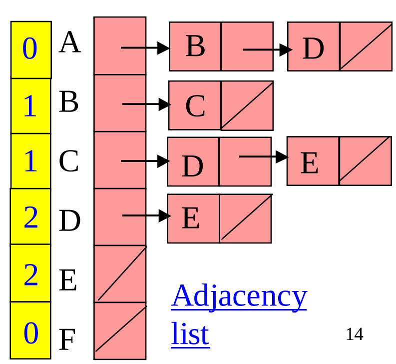
Final Result:



Summary of Topo-Sort Algorithm #1

1. Store each vertex's **In-Degree** (# of incoming edges) in an array
2. While there are vertices remaining:
   - Find a vertex with In-Degree zero and output it
   - Reduce In-Degree of all vertices adjacent to it by 1
   - Mark this vertex (In-Degree = -1)





In-Degree array

Adjacency list

Topological Sort Algorithm #1: Analysis

For input graph G = (V,E), Run Time = ?

Break down into total time required to:

- Initialize In-Degree array:
  O(|E|)
- Find vertex with in-degree 0:
  |V| vertices, each takes O(|V|) to search In-Degree array.
  Total time = O(|V|^2)
- Reduce In-Degree of all vertices adjacent to a vertex:
  O(|E|)
- Output and mark vertex:
  O(|V|)

Total time = **O(|V|^2 + |E|)** **Quadratic time!**

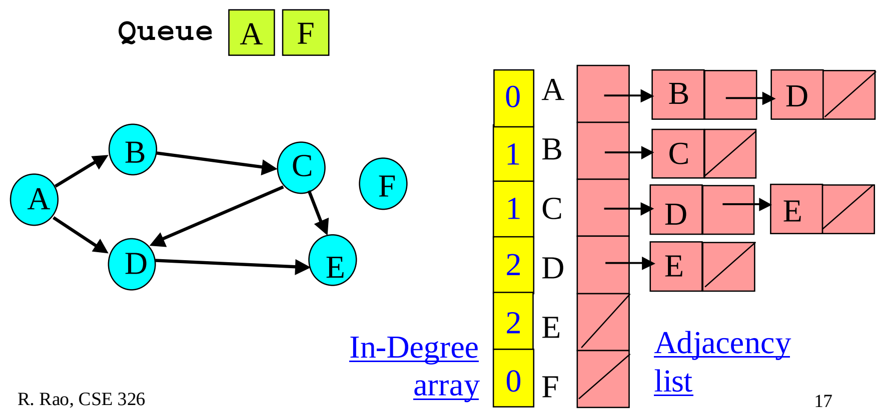
Can we do better than quadratic time?

Problem:

Need a faster way to find vertices with in-degree 0 instead of searching through entire in-degree array

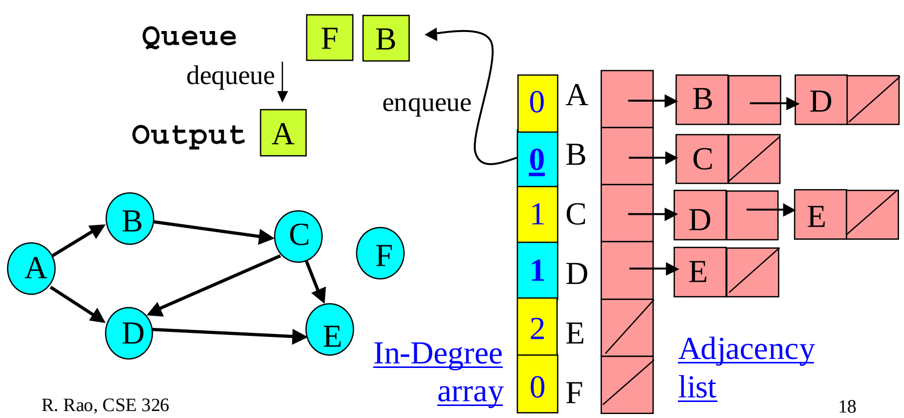
Topological Sort (Take 2)

Key idea: Initialize and maintain a *queue (or stack)* of vertices with In-Degree 0



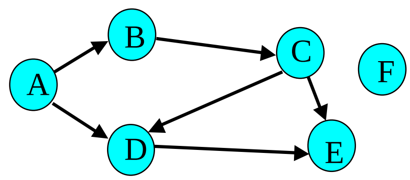
Topological Sort (Take 2)

After each vertex is output, when updating In-Degree array, *enqueue any vertex whose In-Degree has become zero*



Topological Sort Algorithm #2

1. Store each vertex's **In-Degree** in an array
2. Initialize a queue with all in-degree zero vertices
3. While there are vertices remaining in the queue:
   - Dequeue and output a vertex
   - Reduce In-Degree of all vertices adjacent to it by 1
   - Enqueue any of these vertices whose In-Degree became zero



Sort this digraph!

Topological Sort Algorithm #2: Analysis

For input graph G = (V,E), Run Time = ?

Break down into total time to:

Initialize In-Degree array:
O(|E|)

Initialize Queue with In-Degree 0 vertices:
O(|V|)

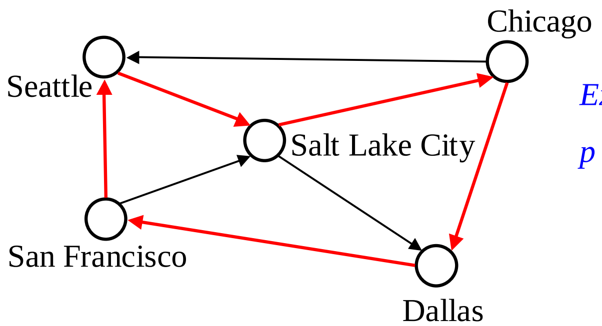
Dequeue and output vertex:
|V| vertices, each takes only O(1) to dequeue and output.
Total time = O(|V|)

Reduce In-Degree of all vertices adjacent to a vertex and Enqueue any In-Degree 0 vertices:
O(|E|)

Total time = **O(|V| + |E|)** **Linear running time!**

Paths

- Recall definition of a path in a tree – same for graphs
- A *path* is a list of vertices $\{v_1, v_2, \ldots, v_n\}$ such that (v_i, v_{i+1}) is in E for all $0 \leq i < n$.



Example of a path:

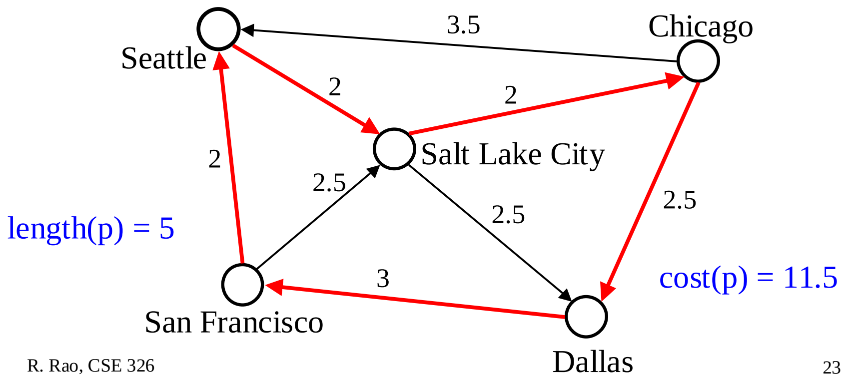
p = {Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle}

Simple Paths and Cycles

- A *simple path* repeats no vertices (except the 1st can be the last):
  - p = {Seattle, Salt Lake City, San Francisco, Dallas}
  - p = {Seattle, Salt Lake City, Dallas, San Francisco, Seattle}
- A *cycle* is a path that starts and ends at the same node:
  - p = {Seattle, Salt Lake City, Dallas, San Francisco, Seattle}
- A *simple cycle* is a cycle that repeats no vertices except that the first vertex is also the last
- A directed graph with no cycles is called a DAG (directed acyclic graph) E.g. All trees are DAGs
  - A graph with cycles is often a drag…

Path Length and Cost

- *Path length*: the number of edges in the path
- *Path cost*: the sum of the costs of each edge
  - Note: Path length = unweighted path cost (edge weight = 1)



Single Source, Shortest Path Problems

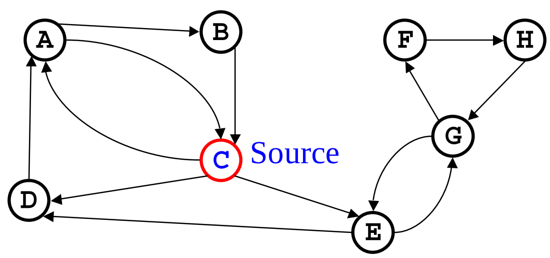
- Given a graph G = (V, E) and a “source” vertex s in V, find the minimum cost paths from s to every vertex in V
- Many variations:
  - unweighted vs. weighted
  - cyclic vs. acyclic
  - positive weights only vs. negative weights allowed
  - multiple weight types to optimize
  - Etc.
- We will look at only a couple of these…
  - See text for the others

Why study shortest path problems?

- Plenty of applications
- **Traveling on a "starving student" budget**: What is the cheapest multi-stop airline schedule from Seattle to city X?
- **Optimizing routing of packets on the internet**:
  - Vertices = routers, edges = network links with different delays
  - *What is the routing path with smallest total delay?*
- **Hassle-free commuting**: Finding what highways and roads to take to minimize total delay due to traffic
- **Finding the fastest way to get to coffee vendors on campus from your classrooms**

Unweighted Shortest Paths Problem

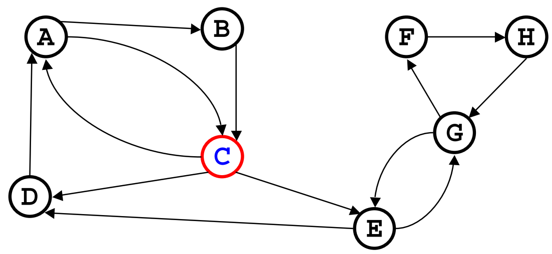
Problem: Given a "source" vertex s in an unweighted graph G = (V,E), find the shortest path from s to all vertices in G



Find the shortest path from `C` to: `A B C D E F G H`

Solution based on Breadth-First Search

- Basic Idea: Starting at node s, find vertices that can be reached using 0, 1, 2, 3, …, N-1 edges (works even for cyclic graphs!)



On-board example:

Find the shortest path from **C** to: **A B C D E F G H**

Breadth-First Search (BFS) Algorithm

- Uses a **queue** to store vertices that need to be expanded
- Pseudocode (source vertex is `s`):

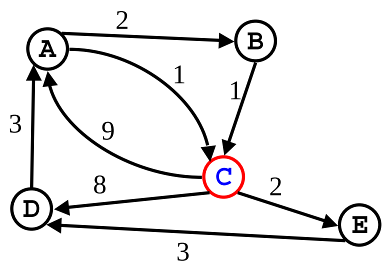
```
1. Dist[s] = 0
2. Enqueue(s)
3. While queue is not empty
   1. X = dequeue
   2. For each vertex Y adjacent to X and not
      previously visited
      ● Dist[Y] = Dist[X] + 1
      ● Prev[Y] = X
      ● Enqueue Y
```

(**Prev** allows paths to be reconstructed)

- Running time (same as topological sort) = **O(|*V*| + |*E*|)** (why?)

That was easy but what if edges have weights?

Does BFS still work for finding minimum cost paths?

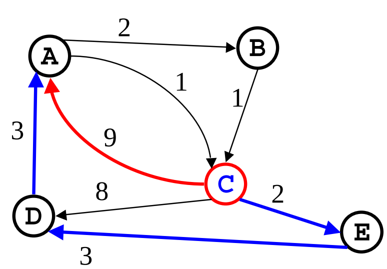


Can you find a counterexample (a path) for this graph to show BFS won't work?

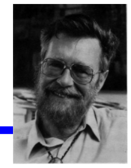
What if edges have weights?

- BFS does not work anymore – minimum cost path may have additional hops

Shortest path from C to A:
BFS: C A
(cost = 9)
Minimum Cost Path = C E D A
(cost = 8)



Dijkstra to the rescue…



E. W. Dijkstra
(1930-2002)

- Legendary figure in computer science
- Some rumors collected from previous classes…
- Rumor #1: Supported teaching introductory computer courses without computers (pencil and paper programming)
- Rumor #2: Supposedly wouldn't read his e-mail; so, his staff had to print out his e-mails and put them in his mailbox

An Aside: Dijsktra on GOTOs

"For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce."

Opening sentence of: "*Go To Statement Considered Harmful*" by Edsger W. Dijkstra, Letter to the Editor, Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148.

Dijkstra's Algorithm for Weighted Shortest Path

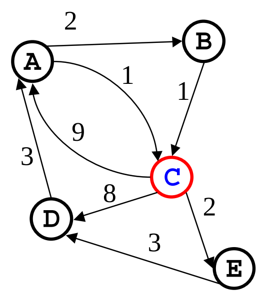
- Classic algorithm for solving shortest path in weighted graphs (without negative weights)
- Example of a *greedy* algorithm
  - Irrevocably makes decisions without considering future consequences
  - Sound familiar? Not necessarily the best life strategy… but works in some cases (e.g. Huffman encoding)

Dijkstra's Algorithm for Weighted Shortest Path

- Basic Idea:
  - Similar to BFS
    - Each vertex stores a cost for path from source
    - Vertex to be expanded is the one with least path cost seen so far
      - Greedy choice – always select current best vertex
      - Update costs of all neighbors of selected vertex
  - But unlike BFS, a vertex already visited may be updated if a better path to it is found

Pseudocode for Dijkstra's Algorithm

1. Initialize the cost of each node to ∞
2. Initialize the cost of the source to 0
3. While there are unknown nodes left in the graph
   1. Select the unknown node *N* with the *lowest cost* (greedy choice)
   2. Mark *N* as known
   3. For each node *X* adjacent to *N*
      If (*N*'s cost + cost of (*N*, *X*)) < *X*'s cost
         X's cost = *N*'s cost + cost of (*N*, *X*)
         Prev[*X*] = *N* *//store preceding node*



(Prev allows paths to be reconstructed)

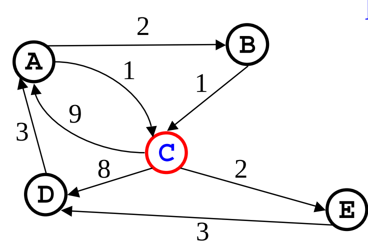
Dijkstra's Algorithm (greed in action)

vertex	known	cost	Prev
A	No	∞	
B	No	∞	
C	Yes	0	
D	No	∞	
E	No	∞	

Initial

vertex	known	cost	Prev
A			
B			
C			
D			
E			

Final



Dijkstra's Algorithm (greed in action)

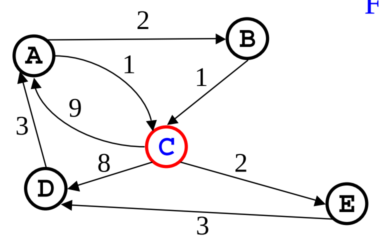
vertex	known	cost	Prev
A	No	∞	-
B	No	∞	-
C	Yes	0	-
D	No	∞	-
E	No	∞	-

Initial

→

vertex	known	cost	Prev
A	Yes	8	D
B	Yes	10	A
C	Yes	0	-
D	Yes	5	E
E	Yes	2	C

Final



Questions for Next Time:

Does Dijkstra's method always work?

How fast does it run?

Where else in life can I be greedy?

To Do:

Start Homework Assignment #4

(Don't wait until the last few days!!!)

Continue reading and enjoying chapter 9