GRAPHS

Lecture 17

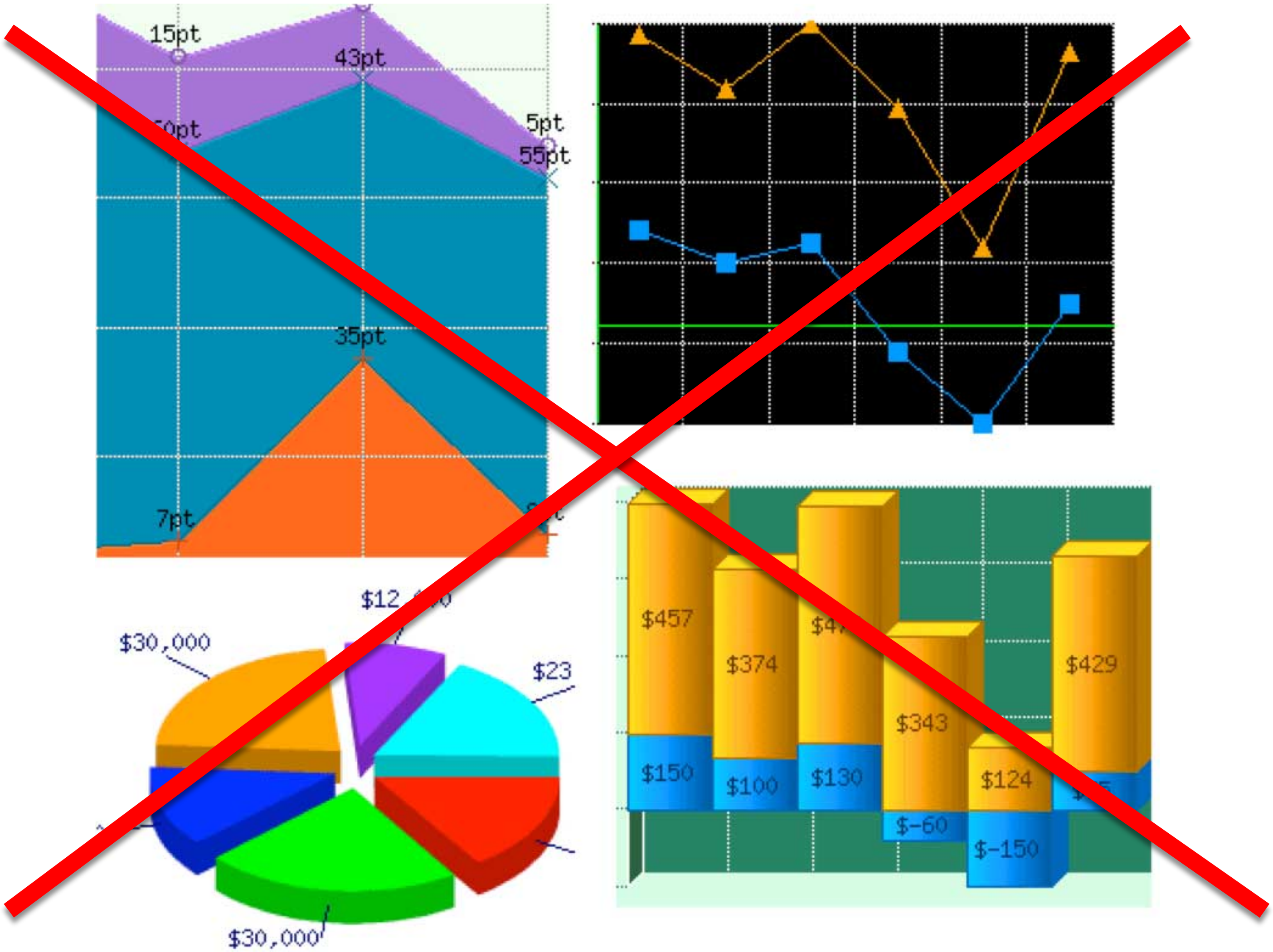
CS 2110 — Spring 2019

JavaHyperText Topics

"Graphs", topics 1-3

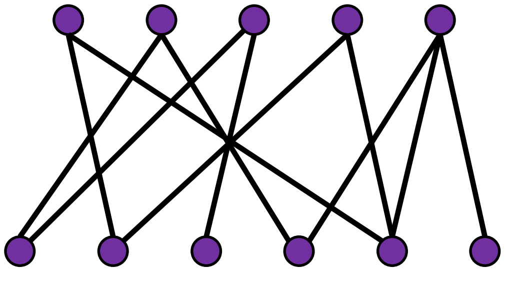
- 1: Graph definitions
- 2: Graph terminology
- 3: Graph representations

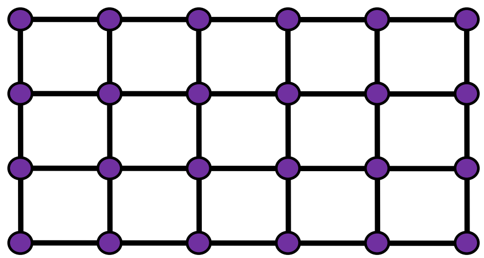
Charts (aka graphs)

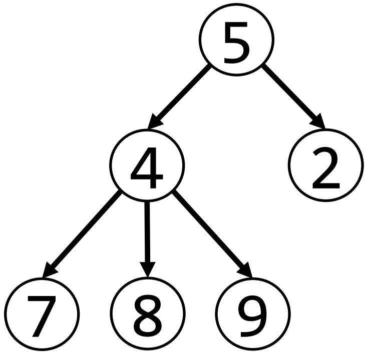


Graphs

- **Graph:**
  - [charts] **Points** connected by **curves**
  - [in CS] **Vertices** connected by **edges**
- Graphs generalize trees
- Graphs are relevant far beyond CS…examples…





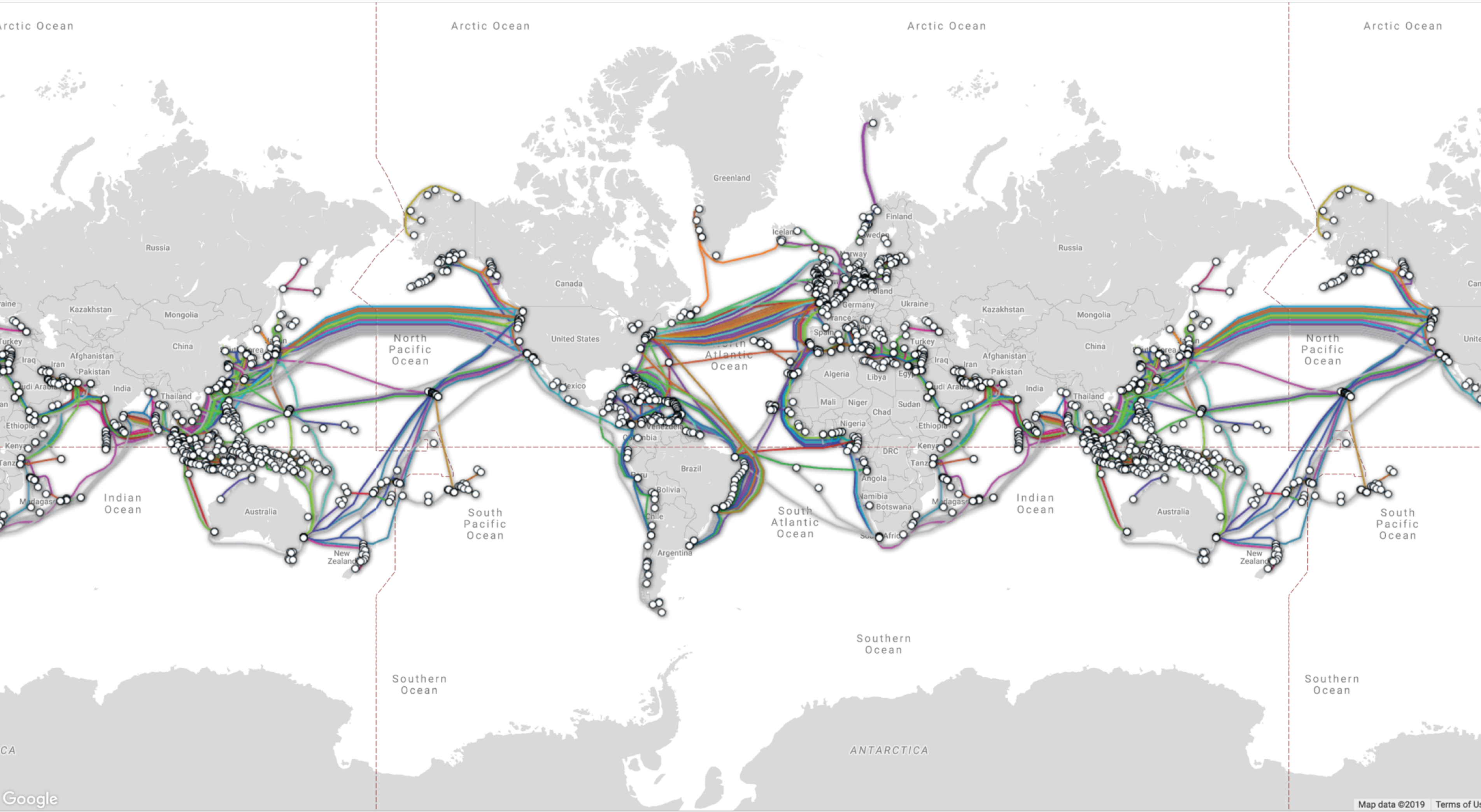


https://medium.com/@johnrobb/facebook-the-complete-social-graph-b58157ee6594

Vertices: people "from"
Edges: friendships

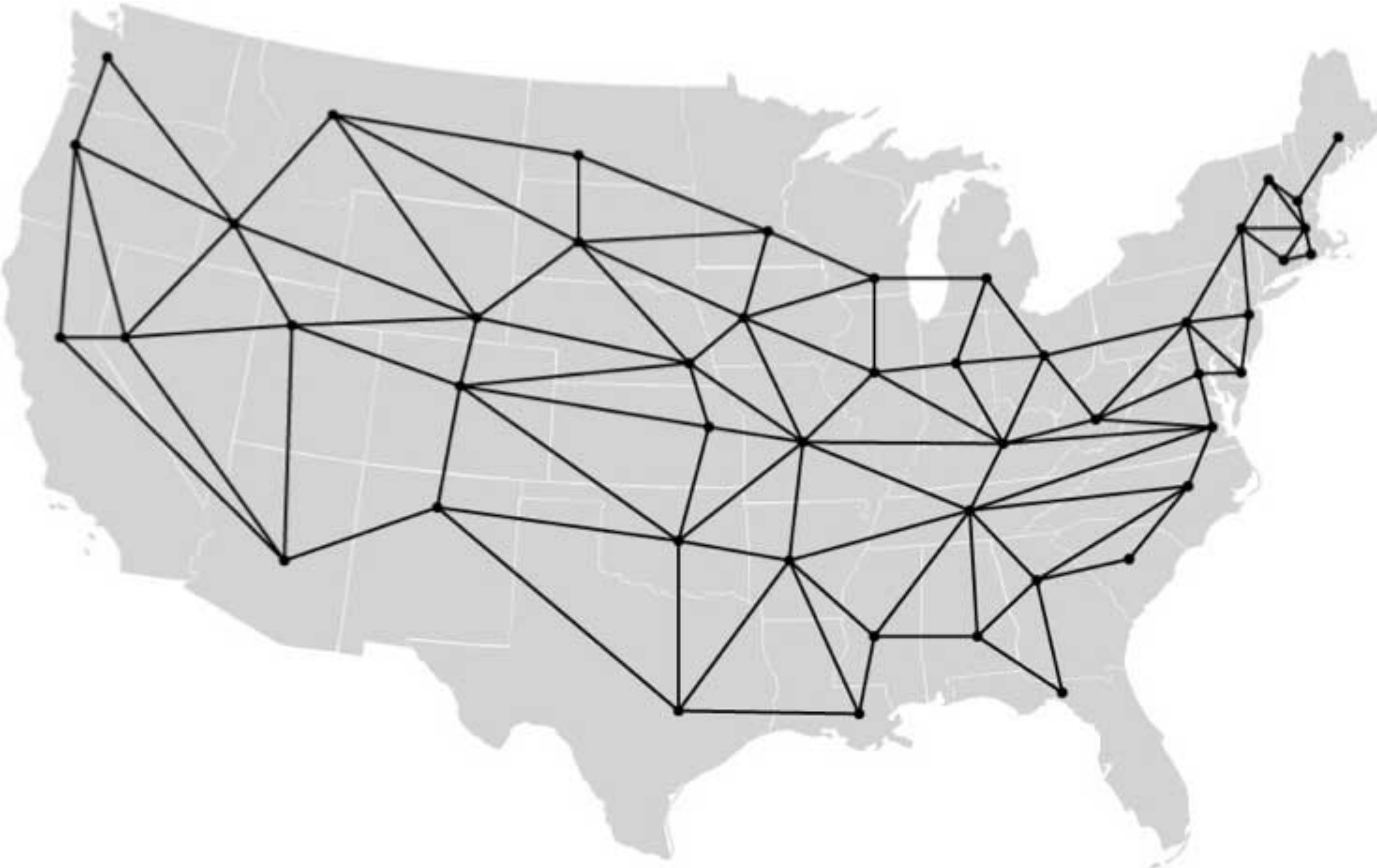
Vertices: subway stops

Edges: railways



https://www.submarinecablemap.com/

Vertices: stations
Edges: cables



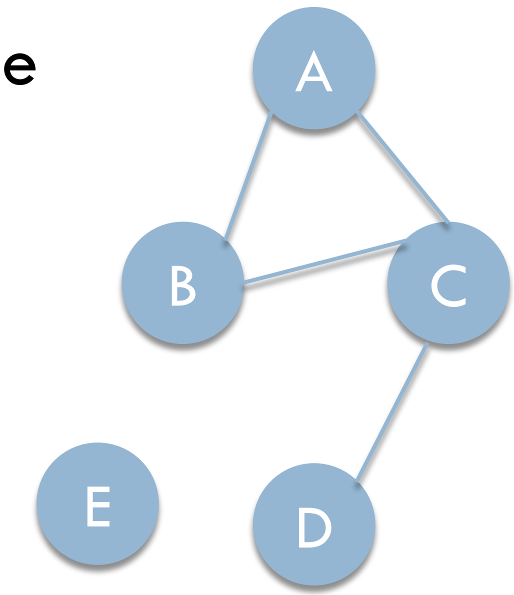
Vertices: State capitals
Edges: "States have shared border"

http://www.cs.cmu.edu/~bryant/boolean/maps.html

Graphs as mathematical structures

Undirected graphs

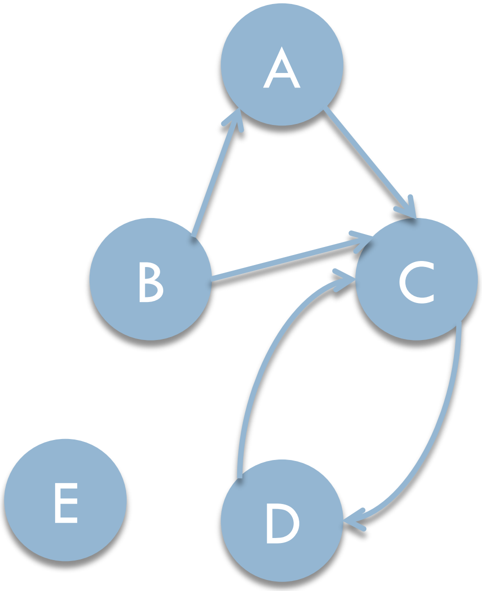
An **undirected** graph is a pair (V, E) where



- V is a set
  - Element of V is called a **vertex** or **node**
  - We'll consider only finite graphs
  - *Ex:* $\mathbf{V} = \{A, B, C, D, E\}; |V| = 5$
- E is a set
  - Element of E is called an **edge** or **arc**
  - An edge is itself a two-element set {u, v} where $\{u, v\} \subseteq V$
  - Often require $u \neq v$ (i.e., no **self-loops**)
  - *Ex:* $\mathbf{E} = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}\}, |E| = 4$

Directed graphs

A **directed** graph is similar except the edges are **pairs** (u, v), hence order matters



$\boldsymbol{V} = \{A, B, C, D, E\}$

$\boldsymbol{E} = \{(A, C), (B, A), (B, C), (C, D), (D, C)\}$

$|\boldsymbol{V}| = 5$

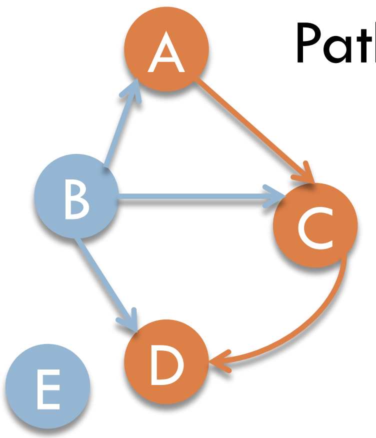
$|\boldsymbol{E}| = 5$

Convert undirected ←→ directed?

- Right question is: convert and maintain which properties of graph?
- Convert undirected to directed and maintain **paths?**

Paths

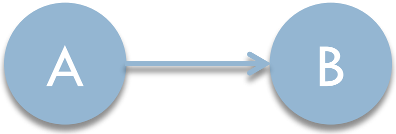
- A **path** is a sequence $v_0, v_1, v_2, \ldots, v_p$ of vertices such that for $0 \leq i < p$,
  - Directed: $(v_i, v_{i+1}) \in E$
  - Undirected: $\{v_i, v_{i+1}\} \in E$
- The **length** of a path is its number of edges



Path: A,C,D

Convert undirected ←→ directed?

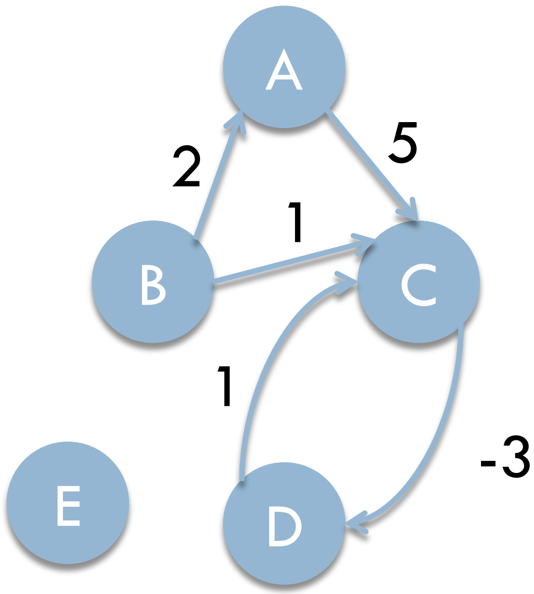
- Right question is: convert and maintain which properties of graph?
- Convert **undirected to directed** and maintain **paths**:
  - Nodes unchanged
  - Replace each edge {u,v} with two edges {(u,v), (v,u)}
- Convert **directed to undirected** and maintain paths: Can't:



Labels

Whether directed or undirected, edges and vertices can be **labeled** with additional data

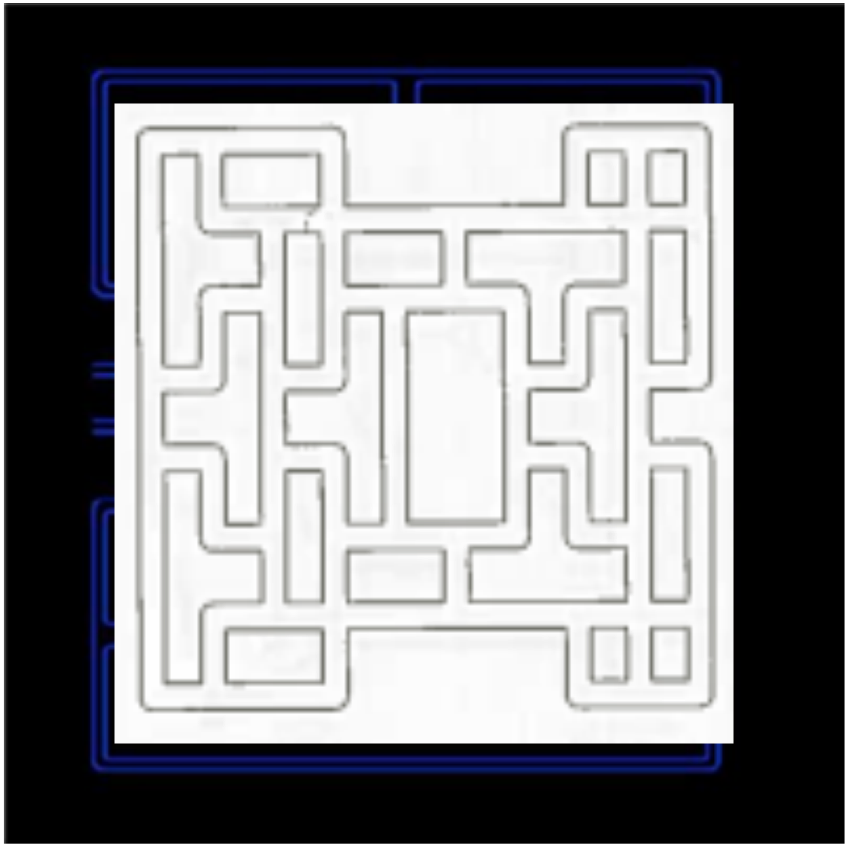
Nodes already labeled with characters

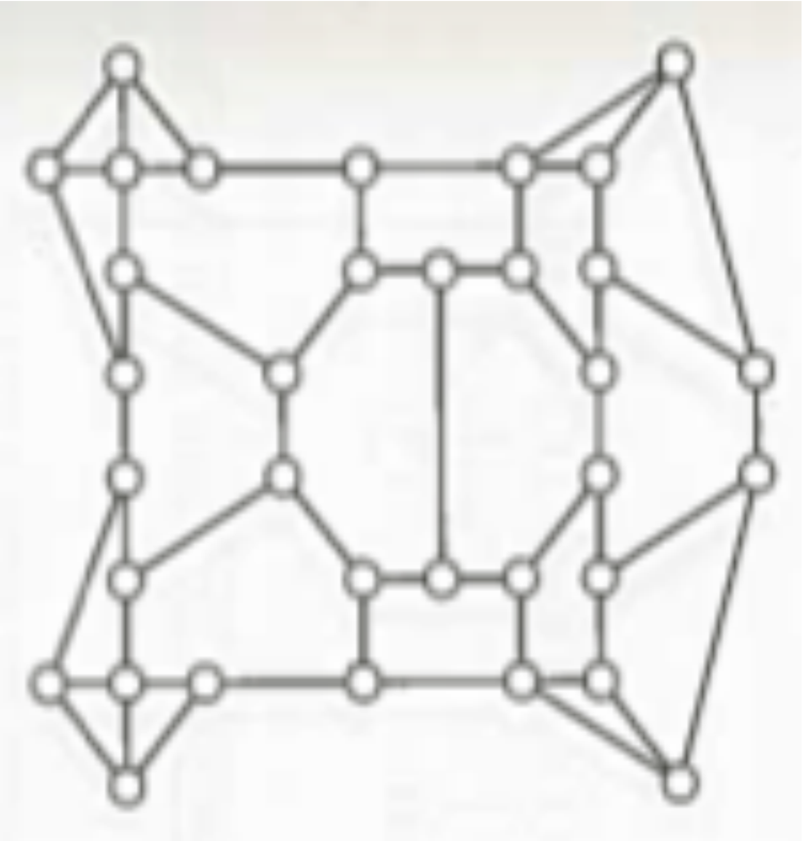


Edges now labeled with integers

Discuss

How could you represent a maze as a graph?





Algorithms, 2nd ed., Sedgewick, 1988

Announcement

A4: See time distribution and comments @735

- Spending >16 hours is a problem; talk to us or a TA about why that might be happening
- Comments on the GUI:
  - "GUI was pretty awesome."
  - "I didn't see the relevance of the GUI."
- Hints:
  - "Hints were extremely useful and I would've been lost without them."
  - "Hints are too helpful. You should leave more for people to figure out on their own."
- Adjectives:
  - "Fun" (x30), "Cool" (x19)
  - "Whack", "Stressful", "Tedious", "Rough"

Graphs as data structures

Graph ADT

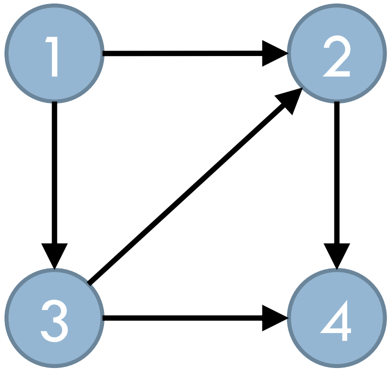
Operations could include:

- Add a vertex
- Remove a vertex
- Search for a vertex
- Number of vertices
- Add an edge
- Remove an edge
- Search for an edge
- Number of edges

Demo

Graph representations

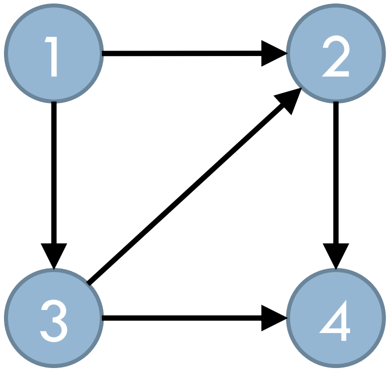
- Two vertices are **adjacent** if they are connected by an edge
- Common graph representations:
  - **Adjacency list**
  - **Adjacency matrix**



running example
(directed, no edge labels)

Adjacency “list”

- Maintain a collection of the vertices
- For each vertex, also maintain a collection of its adjacent vertices



- Vertices: 1, 2, 3, 4
- Adjacencies:
  - 1: 2, 3
  - 2: 4
  - 3: 2, 4
  - 4: none

Could implement these “lists” in many ways…

Adjacency list implementation #1

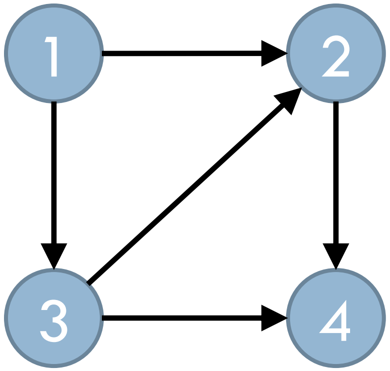
Map from vertex label to sets of vertex labels

$1 \mapsto \{2, 3\}$

$2 \mapsto \{4\}$

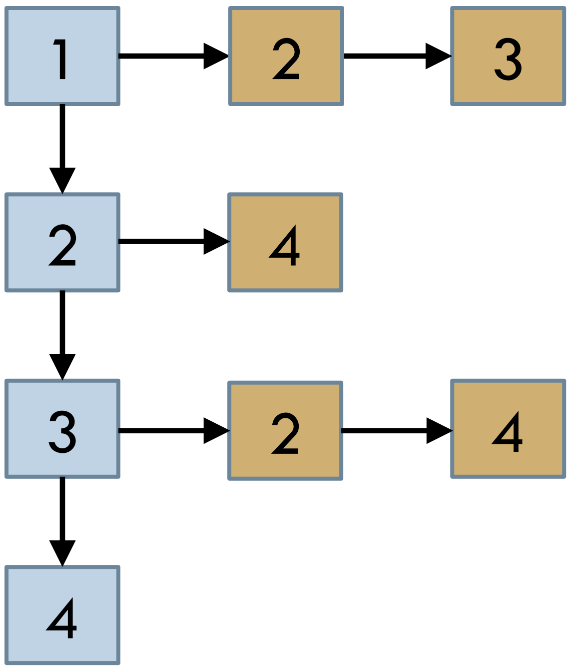
$3 \mapsto \{2, 4\}$

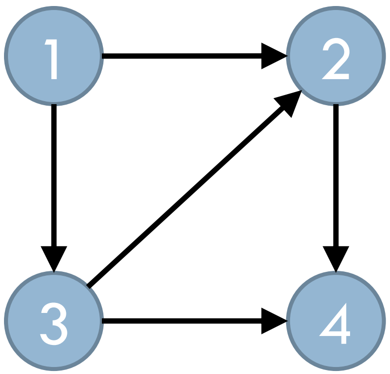
$4 \mapsto \{\text{none}\}$



Adjacency list implementation #2

Linked list, where each node contains vertex label and linked list of adjacent vertex labels

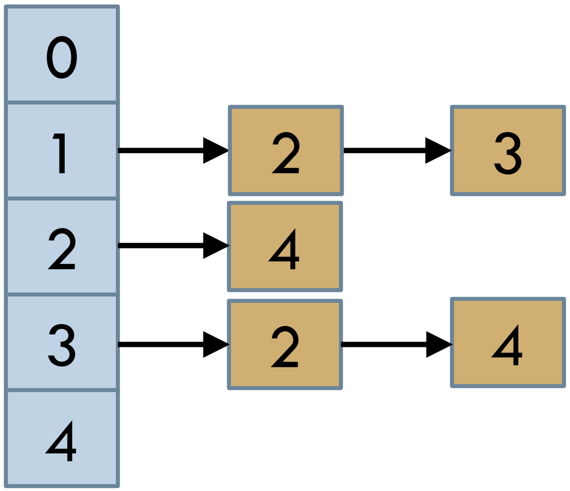


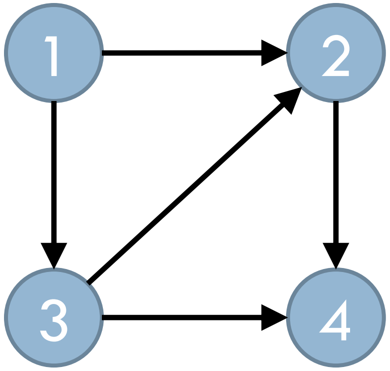


Demo

Adjacency list implementation #3

Array, where each element contains linked list of adjacent vertex labels



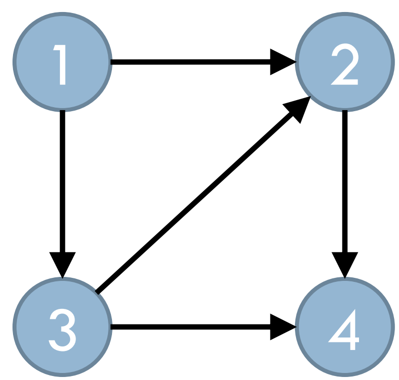


Requires: labels are integers; dealing with bounded number of vertices

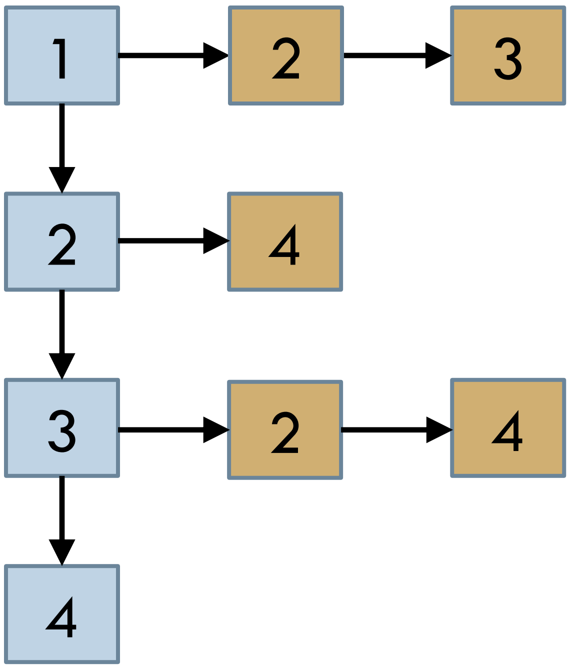
Adjacency "matrix"

- Given integer labels and bounded # of vertices…
- Maintain a 2D Boolean array **`b`**
- Invariant: element **`b[i][j]`** is **`true`** iff there is an edge from vertex **`i`** to vertex **`j`**

	0	1	2	3	4
0	F	F	F	F	F
1	F	F	**T**	**T**	F
2	F	F	F	F	**T**
3	F	F	**T**	F	**T**
4	F	F	F	F	F



Adjacency list vs. Adjacency matrix



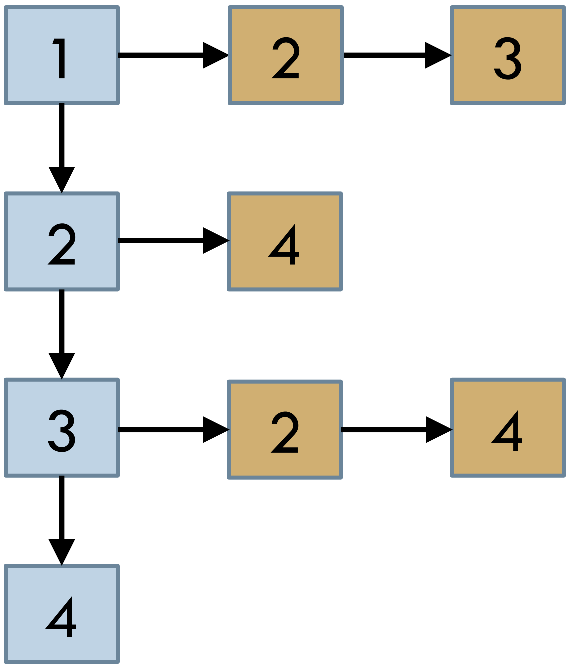
	0	1	2	3	4
0	F	F	F	F	F
1	F	F	**T**	**T**	F
2	F	F	F	F	**T**
3	F	F	**T**	F	**T**
4	F	F	F	F	F

Efficiency: Space to store?

$O(|V| + |E|)$

$O(|V|^2)$

Adjacency list vs. Adjacency matrix



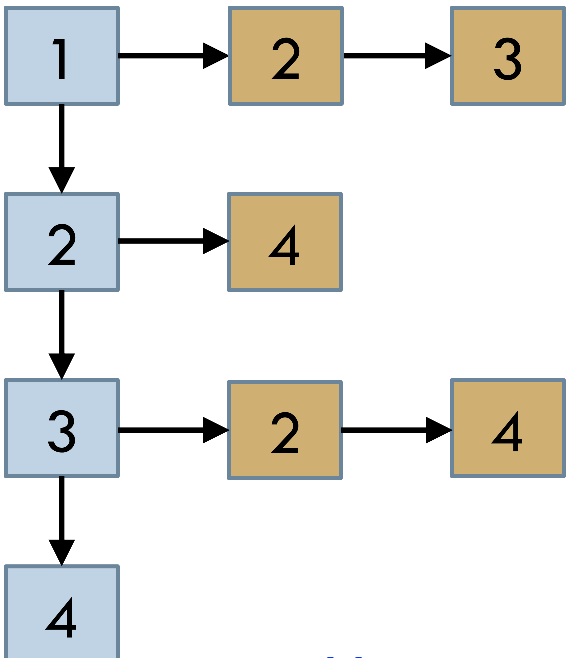
	0	1	2	3	4
0	F	F	F	F	F
1	F	F	**T**	**T**	F
2	F	F	F	F	**T**
3	F	F	**T**	F	**T**
4	F	F	F	F	F

Efficiency: Time to visit all edges?

$O(|V| + |E|)$

$O(|V|^2)$

Adjacency list vs. Adjacency matrix



	0	1	2	3	4
0	F	F	F	F	F
1	F	F	**T**	**T**	F
2	F	F	F	F	**T**
3	F	F	**T**	F	**T**
4	F	F	F	F	F

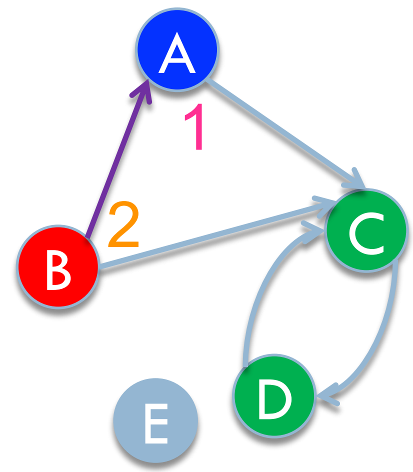
Efficiency: Time to determine whether edge from v_1 to v_2 exists?

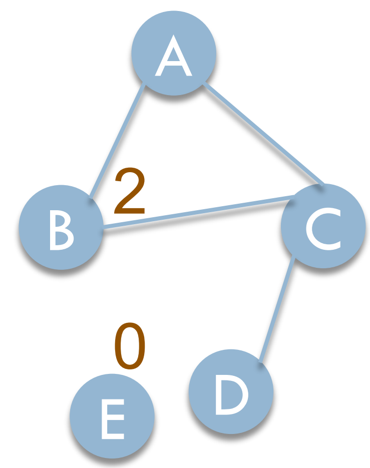
$O(|V| + |E|)$ $O(1)$

Tighter: $O(|V|$ + # edges leaving $v_1)$

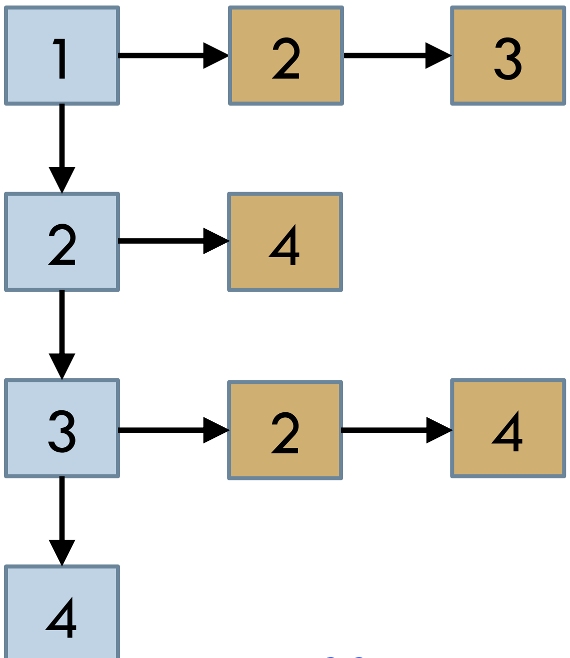
More graph terminology

- Vertices u and v are called
  - the source and sink of the directed edge (u, v), respectively
  - the **endpoints** of (u, v) or $\{u, v\}$
- The outdegree of a vertex u in a directed graph is the number of edges for which u is the source
- The indegree of a vertex v in a directed graph is the number of edges for which v is the sink
- The degree of a vertex u in an undirected graph is the number of edges of which u is an endpoint





Adjacency list vs. Adjacency matrix



	0	1	2	3	4
0	F	F	F	F	F
1	F	F	**T**	**T**	F
2	F	F	F	F	**T**
3	F	F	**T**	F	**T**
4	F	F	F	F	F

Efficiency: Time to determine whether edge from v_1 to v_2 exists?

$O(|V| + |E|)$

$O(1)$

Tighter: $O(|V| + \text{outdegree}(v_1))$

Adjacency list vs. Adjacency matrix

List	Property	Matrix
$O(\|V\| + \|E\|)$	Space	$O(\|V\|^2)$
$O(\|V\| + \|E\|)$	Time to visit all edges	$O(\|V\|^2)$
$O(\|V\| + od(v_1))$	Time to find edge (v_1,v_2)	$O(1)$

Adjacency list vs. Adjacency matrix

List	Property	Matrix
$O(\|V\| + \|E\|)$	Space	$O(\|V\|^2)$
$O(\|V\| + \|E\|)$	Time to visit all edges	$O(\|V\|^2)$
$O(\|V\| + od(v_1))$	Time to find edge (v_1, v_2)	$O(1)$

Adjacency list vs. Adjacency matrix

List	Property	Matrix
$O(\|V\| + \|E\|)$	Space	$O(\|V\|^2)$
$O(\|V\| + \|E\|)$	Time to visit all edges	$O(\|V\|^2)$
$O(\|V\| + od(v_1))$	Time to find edge (v_1, v_2)	$O(1)$

Max # edges $= |V|^2$

Sparse: $|E| \ll |V|^2$

Dense: $|E| \approx |V|^2$

Adjacency list vs. Adjacency matrix

List	Property	Matrix
$O(\|V\| + \|E\|)$	Space	$O(\|V\|^2)$
$O(\|V\| + \|E\|)$	Time to visit all edges	$O(\|V\|^2)$
$O(\|V\| + od(v_1))$	Time to find edge (v_1,v_2)	$O(1)$
Sparse graphs	**Better for**	**Dense graphs**

Max # edges $= |V|^2$

Sparse: $|E| \ll |V|^2$

Dense: $|E| \approx |V|^2$