Graph Algorithms #1 - Introduction & Theory

CptS 223 - Fall 2017 - Aaron Crandall

Today’s Agenda

- Announcements
- Thing of the day
- Introducing graph algorithms

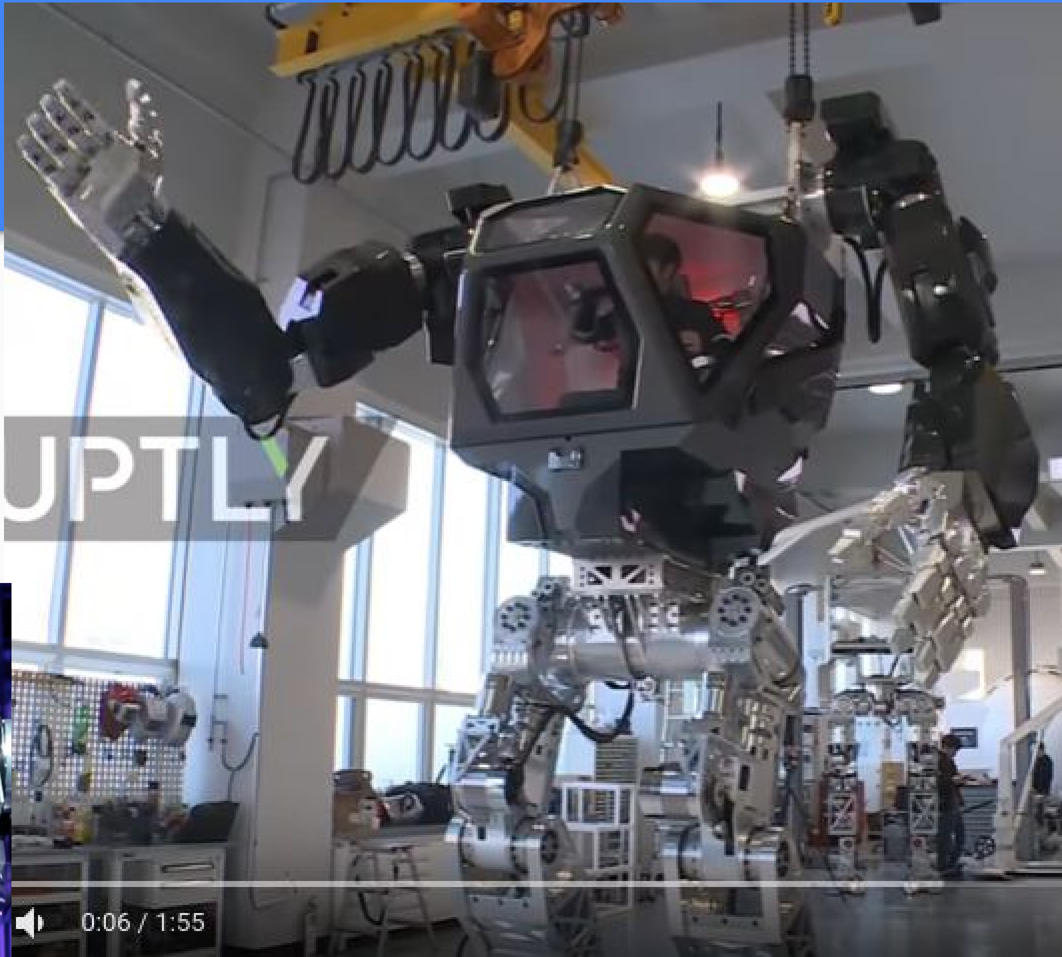
Announcements

- Days left:
    - 8 more real days
    - 3 days in dead week
    - Final
- Targeting return of finals on Friday
- Will go over exam on Wednesday for logistical reasons
- Yes, we’re so far behind on grading it’s nail biting for me
    - It’s got to be caught up ASAP

Thing of the Day

South Korea's first giant robot take its first steps: https://www.youtube.com/watch?v=7rgFtkMiXms

Jeff Bezos →



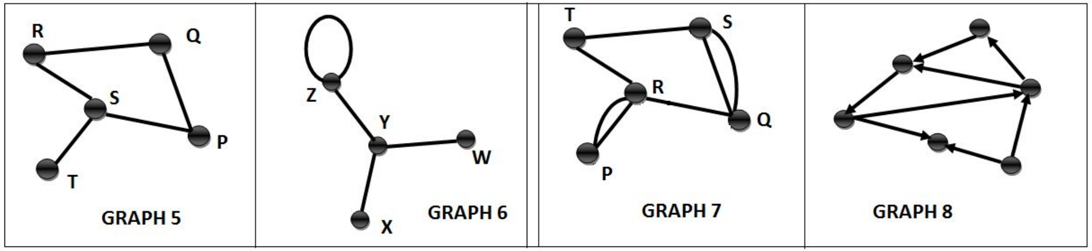
Graphs and Graph Algorithms

Wrong kind of graph

- What is a graph?

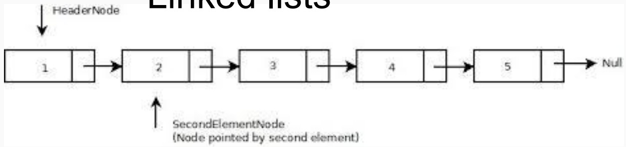
  A graph G = (V, E) consists of a set of vertices, V, and a set of edges, E.

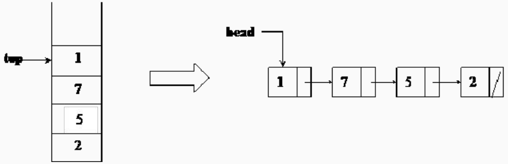
- Each edge is a pair (v, w), where v, w ∈ V.
- Graph algorithm?
  - An algorithm designed to work with data kept in a graph structure



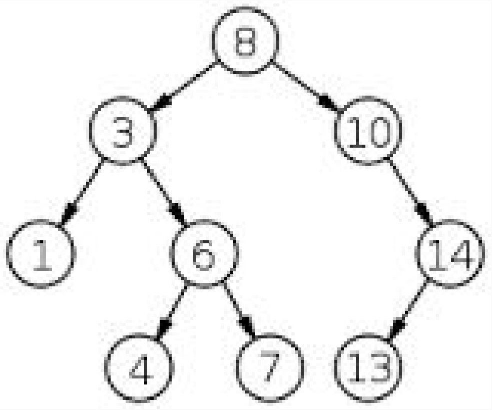
Where have we seen graphs already?

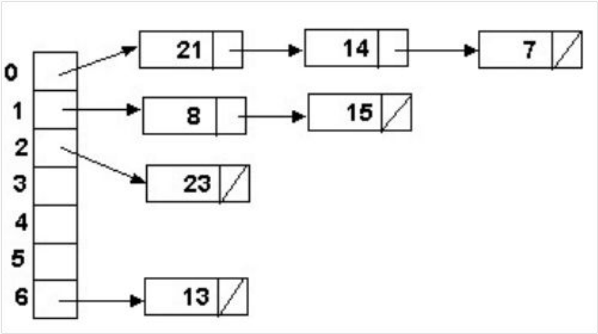
Linked lists





Even stacks





Hash tables (if you zoom out)

Trees (Directed Acyclic Graph / DAG)

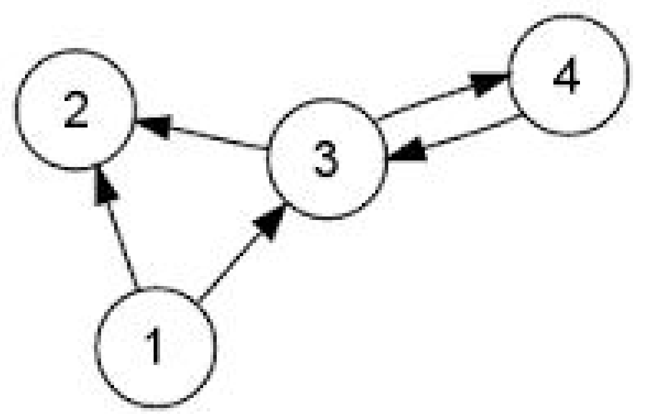
What isn't a graph?
Well… not much
(if anything)

How do we calculate big-O for graphs?

- In trees, heaps, hashing, stacks
  - Number of updates or operations to complete
  - Push == add node to head of list
  - Insert (tree) == traversal to bottom of tree, then node create & update
- In sorting:
  - Number of swaps/moves and comparisons done
- For graphs, it’s based around two things:
  - Edges traversed, nodes (vertices) inspected
  - |E| == number of edges || |V| == number of vertices

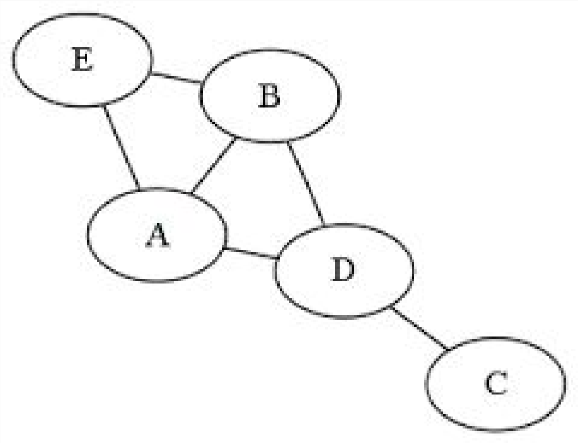
Directed graphs - digraphs

- Directed graphs are when the edges are directed.
- This means they have a front and a back, normally shown as an arrow
- Where have we seen these already?
- $(v,w) \in E$ -- Edge from v to w



Undirected Graphs

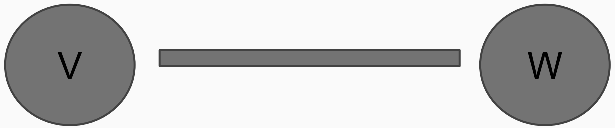
- Edges do not have a front and back, normally shown with a line (no arrow)
    - Edges can be traversed in either direction
    - Think of it being the difference between one way streets and normal streets
    - Both nodes are adjacent to each other
- This below is (v, w), which is also (w, v)



Adjacency

- Vertex w is adjacent to v if and only if (v, w) ∈ E
    - This means you can traverse the edge from v to w
- When using undirected edges, (v, w) means (w, v) so w and v are both adjacent to each other

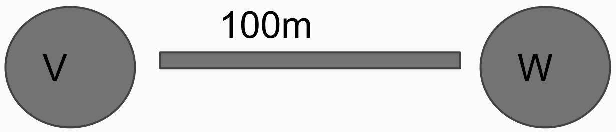
W is *adjacent* to V

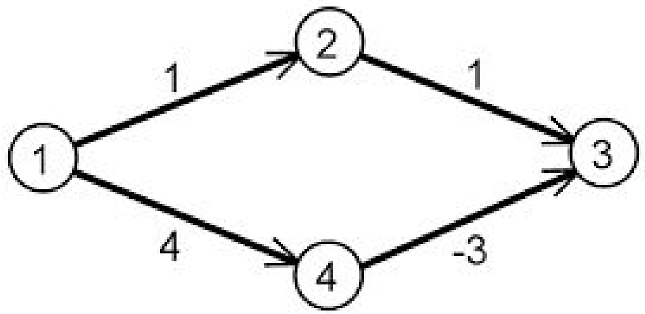


W is *adjacent* to V
V is *adjacent* to W

Weight or Cost of an edge

- Edges can carry a cost to traverse them
  - For example, two intersections are connected and the cost is how many meters long the connecting road is





Degree of a vertex

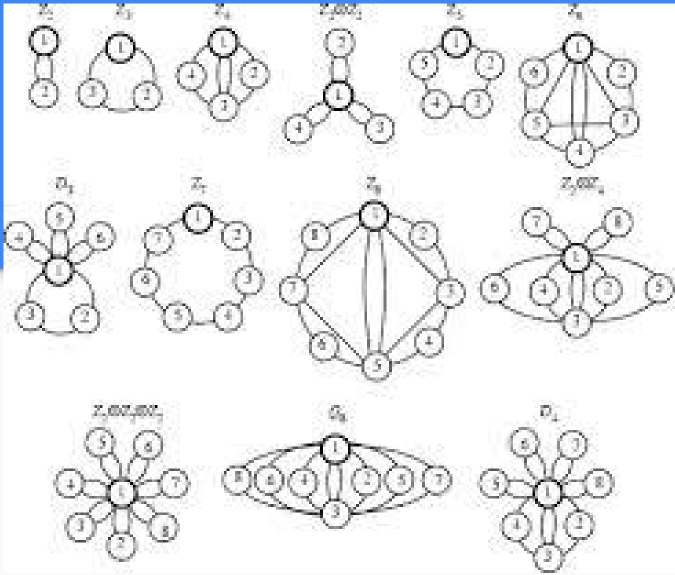
- The number of adjacent vertices to the vertex
- *Indegree* is the number of incoming edges
- *Outdegree* is the number of outgoing edges

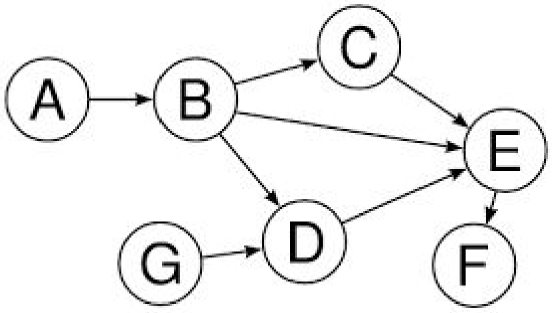
Paths

- A path is a sequence of vertices
  - $w_1, w_2, w_3, \ldots, w_N$ such that $(w_i, w_{\{i+1\}}) \in E$ for $1 \leq i < N$
- The length of the path is the number of edges on the path (not vertices!)
  - So the length = count(V) - 1
- The path can go from a vertex to itself
  - If that path has no edges, it has a length of zero. This is a special case
- The path can do (v, v), which is a loop
  - Normally loops don't happen in most algorithm traversals, but can happen
- Simple paths are where all vertices are distinct (no repeated vertices)
  - Exception: First and last can be the same if it's a path *and* a loop.

Cycles

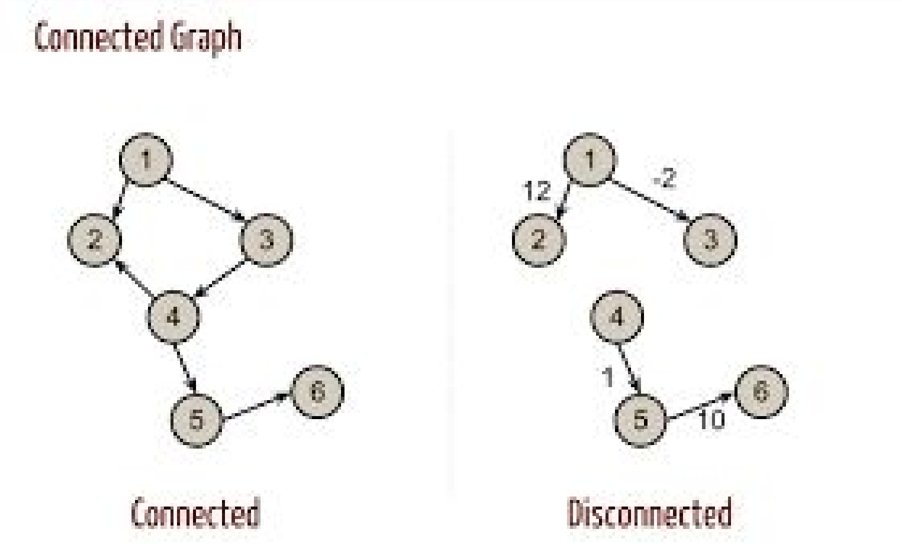
- Directed graph of at least length 1
  - such that $w_1 == w_N$ (you get back to the start w)
- It's a simple cycle if the path is **simple**
  - No repeated nodes in the path, except the start & end
- In undirected graphs the edges must be **distinct**
  - You can only use an edge once in a cycle
- It's a **Directed Acyclic Graph** if it has no cycles
  - A special case called a DAG
  - Where have we seen these?





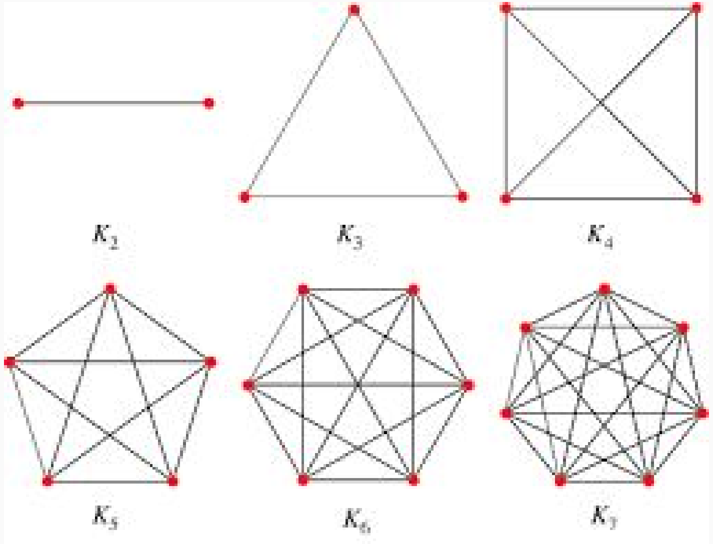
Connected vs. Disconnected

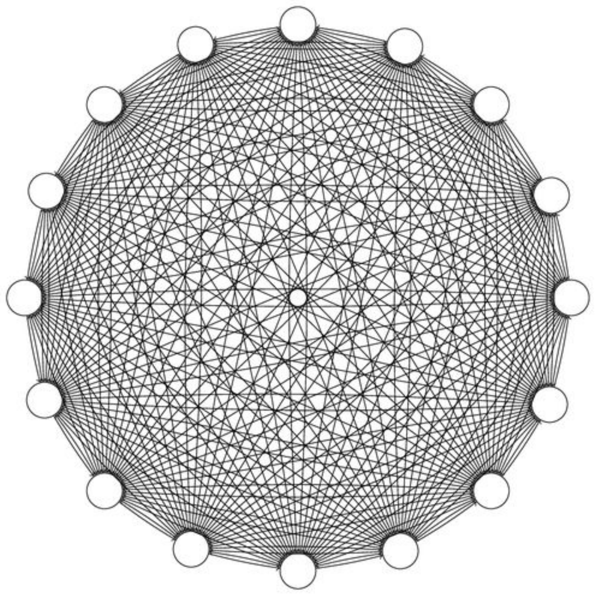
- An undirected graph is connected if there’s a path from every vertex to every other vertex
- A directed graph like that is **strongly connected**
- If it’s all included, but not strong then it’s **weakly connected**
    - This means you have dead ends in the paths you can make
    - The graphs → are weakly connected



Complete graph

- When there's an edge between every pair of vertices
  - There's a path of length 1 between every pair of vertices



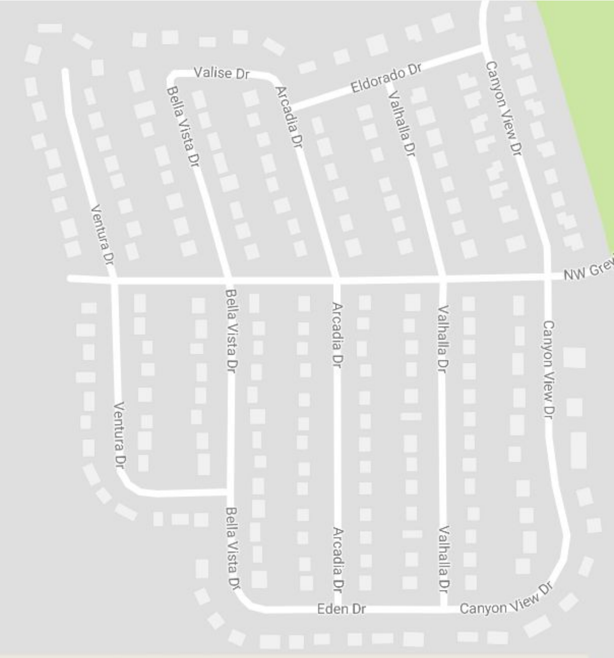


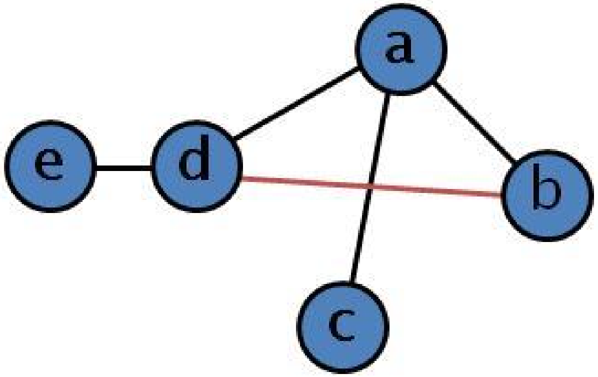
Examples of situations that suit graphs

- Airport connections
- Road trip route planning
- Traffic flow
- Networking
- LinkedIn
- Course prerequisites
- What else?
  - Bacon!

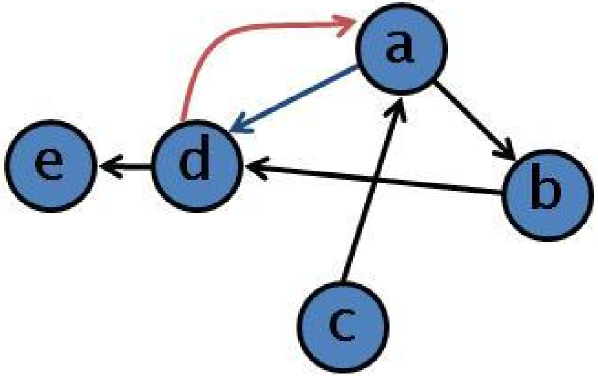
Storing and representing graphs

- Adjacency matrix
  - For each edge (u, v), we set A[u][v] to true ; otherwise the entry in the array is false
  - If edges have weights, set A[u][v] equal to the weight and use either a very large or a very small weight as a sentinel to indicate nonexistent edges (INF, -INF)
  - Requires $\Theta(|V|^2)$ space, only appropriate if $|E| \sim= \Theta(|V|^2)$
    - Example of badness with street maps

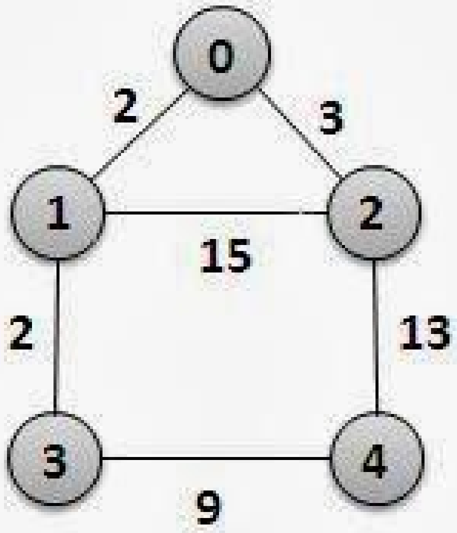




	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0



	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

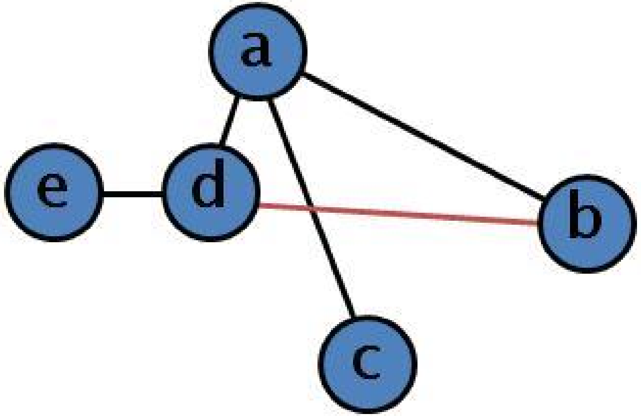


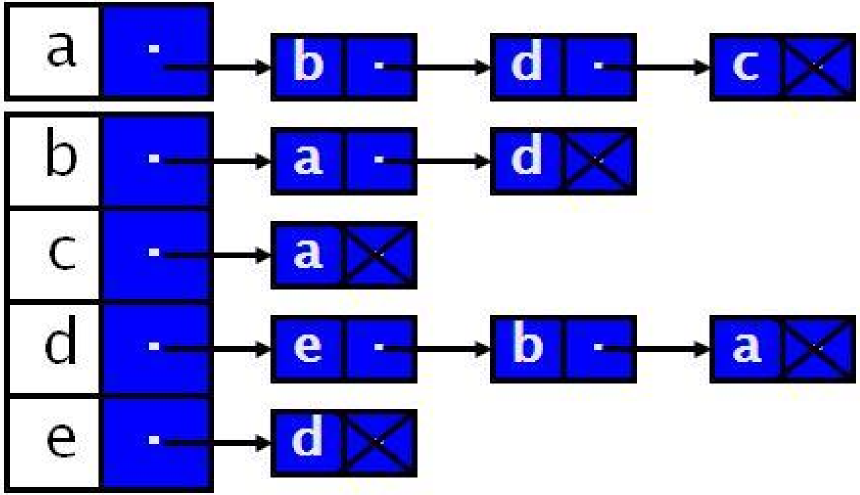
	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

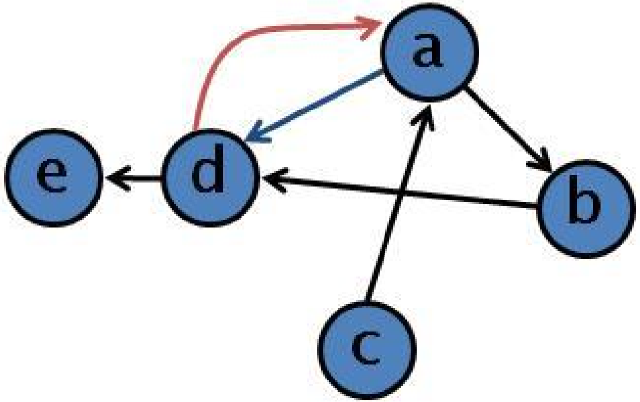
Adjacency Matrix Representation of Weighted Graph

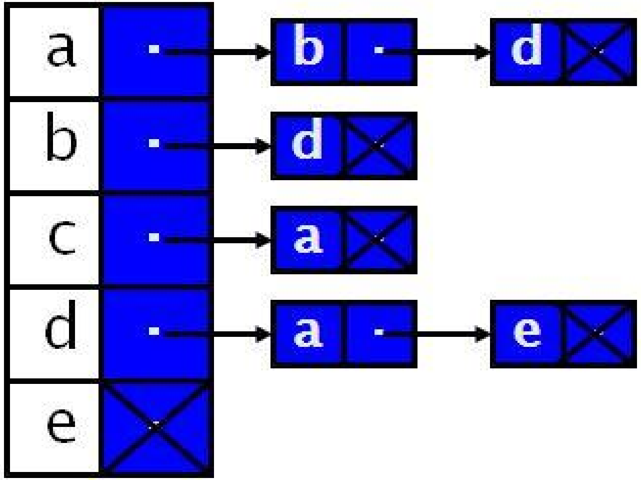
Dense vs. Sparse

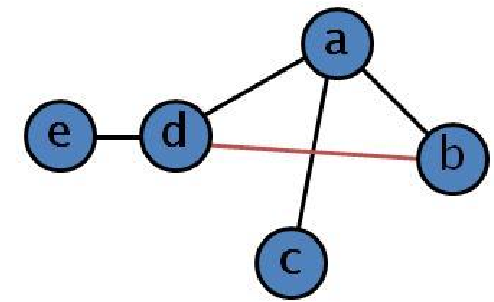
- Most graphs are ***sparse***
  - This means $|E| << |V|$
- Better solution for these is an adjacency list representation
  - Keep a list of all adjacent vertices for each vertex
  - Space requirement becomes $O(|E| + |V|)$
    - Instead of $\Theta(|V|^2)$ with the matrix
  - Weights can be kept with edges in adjacency list
  - Standard way to represent graphs


a
e
d
b
c

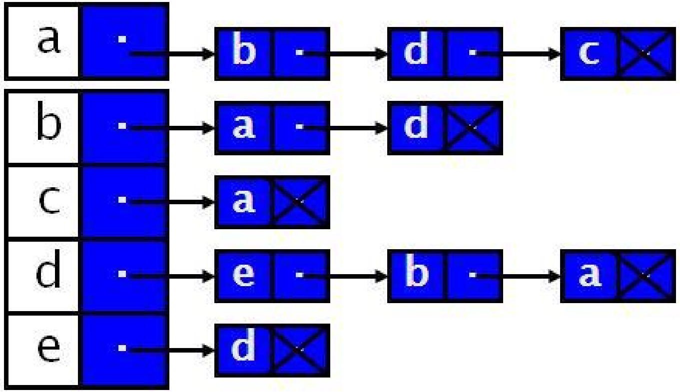

a b d c
b a d
c a
d e b a
e d

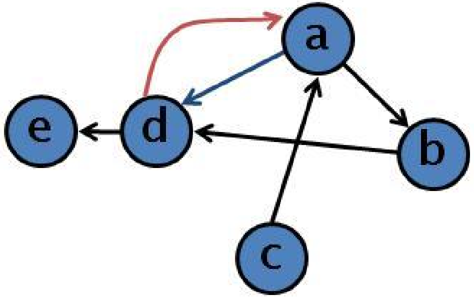

a
e
d
b
c


a b d
b d
c a
d a e
e

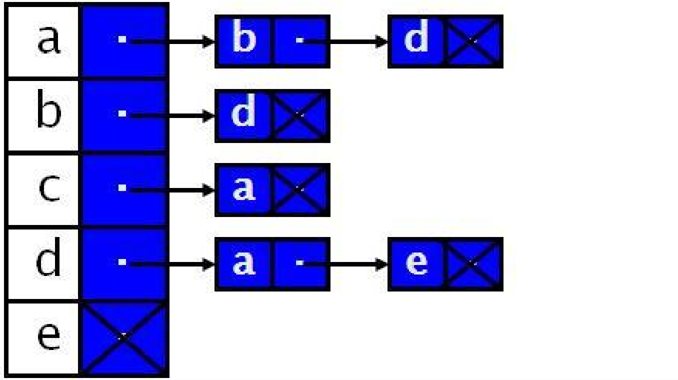


	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0





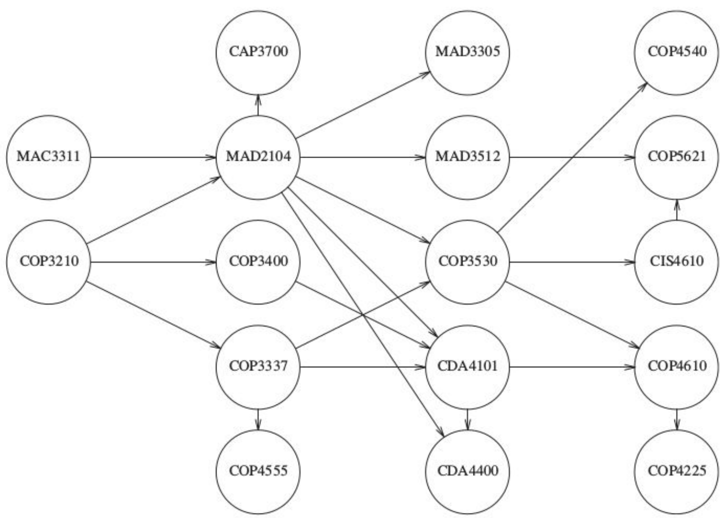
	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0



Direct comparison of adjacency matrix vs. adjacency list

If time, then do topo sort!

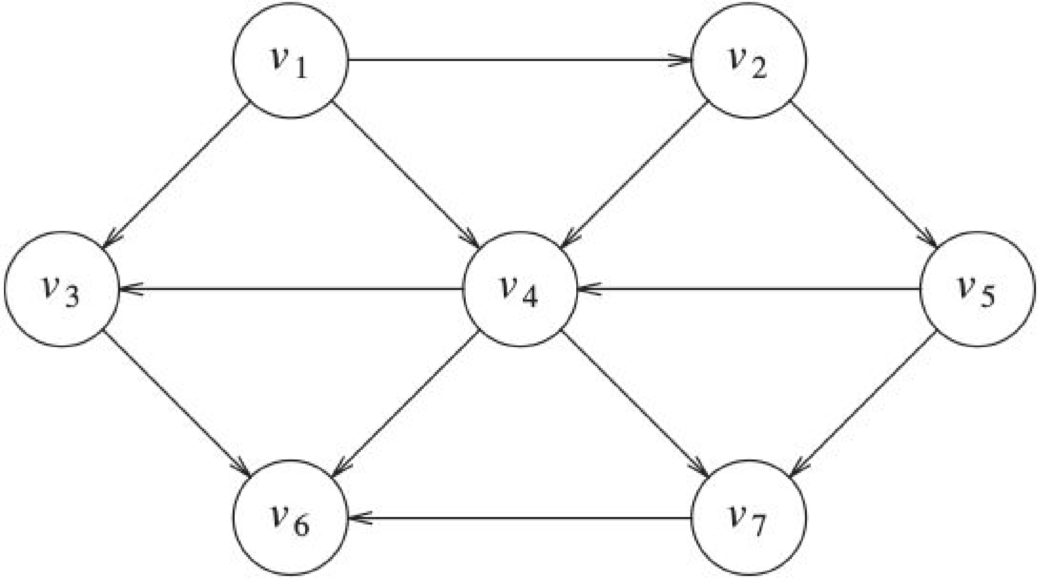
- Topological sort is an ordering of vertices in a directed acyclic graph, such that if there is a path from v_i to v_j , then v_j appears after v_i in the ordering
- Can't work if there's a cycle in the graph
- Does not guarantee a unique ordering
- Used for deciding scheduling of work units
    - Edges represent the dependency of work units
    - Only those with an indegree of 0 can be done next



Topographical Sorting Example

$\{v_1, v_2, v_5, v_4, v_3, v_7, v_6\}$ and $\{v_1, v_2, v_5, v_4, v_7, v_3, v_6\}$ are both valid topological orderings

1) Find node with no in edges
   a) Indegree of zero
   b) Called a "source node"
2) Print out node && remove from graph
3) Repeat



For Wednesday:

Reviewing exam

More on graphs, starting into Dijkstra’s algorithm to stage for next MA