

EC504 ALGORITHMS AND DATA STRUCTURES
FALL 2020 MONDAY & WEDNESDAY
2:30 PM - 4:15 PM

Prof: David Castañón, dac@bu.edu

GTF: Mert Toslali, toslali@bu.edu

Haoyang Wang: haoyangw@bu.edu

Christopher Liao: cliao25@bu.edu

- Sw 2 → Due Friday
- Hw 3 → Due Mon. 3/1
- Exam 1 → Due Mon. 3/8

Exam 1: 3/8: 2:30-4:15
(+ upload time for gradescope)

Open-book/notes

Closed WWW

New Data Structure: Disjoint Sets

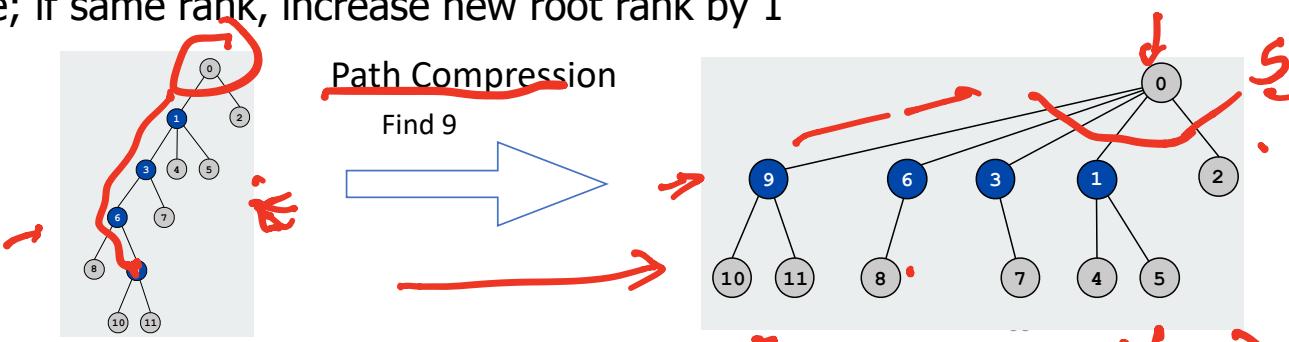
- Problem of Interest: keep track of dynamic relations
 - e.g. who is connected to whom
 - Who belongs to same club
 - Which computers are in same network
 - Which web pages link on the Internet
- Abstraction
 - Have a set of objects
 - Have relations between objects:
 - symmetric, $a \sim b$, transitive $a \sim b, b \sim c \rightarrow a \sim c$
 - Want to keep track of subsets of objects that are related as relations are added

Disjoint Sets

- Operations
 - Find(a): Find the subset that contains element a
 - Union(a,b): Add a relation between two elements a, b which merges the subsets containing a, b
 - Often known as Union-Find problem
- Would like to do Unions and Find in O(1) ✓
 - Seems really hard...
 - We'll come close with a disjoint set data structure

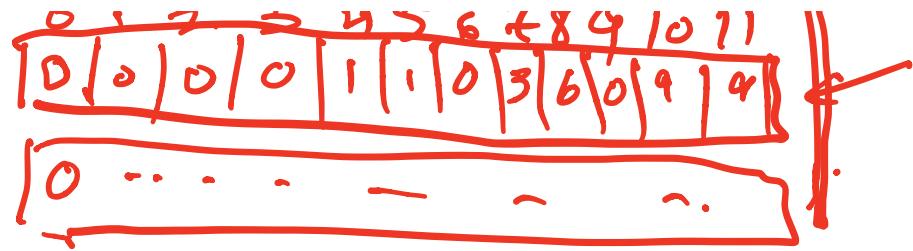
Disjoint Sets Algorithm

- Best Solution
 - Find: get root label of tree, moving up the tree
 - Restructure tree by making all ancestors have root as parent
 - Union: if new relation $a \sim b$, Find(a) and Find(b). If a, b are in different trees, merge trees
 - Merge: roots have ranks. Merge root so smaller rank is child of larger rank one; if same rank, increase new root rank by 1



array.

$O(n)$



Disjoint Sets

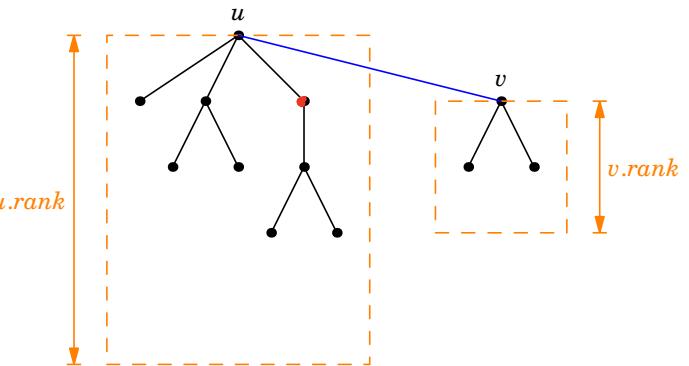
- **Theorem.** Starting from an empty data structure, any sequence of M union and find operations on N objects takes $O(N + M \log^*(N))$ time
- $\log^*(N)$: Ackerman function (*inverse*)
 - $A(i) = 2^{A(i-1)}$; $A(0) = 1$
 - $A(1) = 2, A(2) = 4, A(3) = 16, A(4) = 216 = 65536, A(5) = 265636,$
 - $A(6) = \text{VERY VERY VERY BIG!}$
- Inverse Ackerman: $i = \log^*(N)$
 - $\log^*(N) = \min$ number times you take $\log_2(\log_2(\dots))$ to get smaller than or equal to 1
 - For all practical purposes, it is $O(1)$ (Ch. 21, CLRS)

$$\log^*(2^{64}) \approx 4$$

$$\log^*(\underline{\underline{2^{2^{64}}}}) \approx 5$$

More on Rank and Path Compression

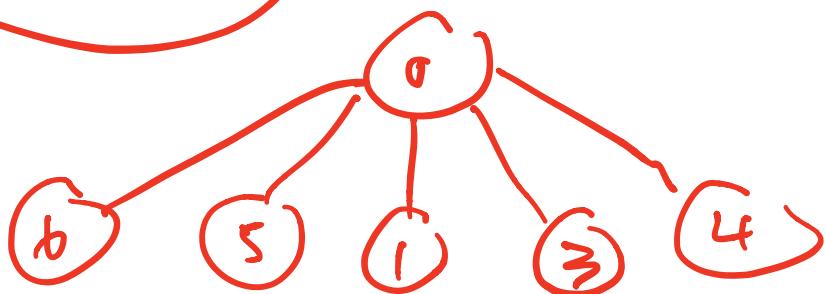
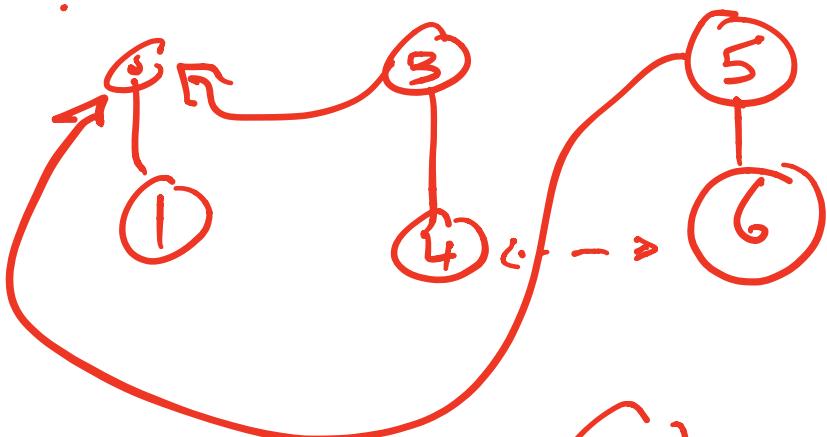
- Rank is an upper bound on height
 - Height is hard to keep track of exactly because of path compression
- How is rank assigned?
 - Rank of single nodes = 0
 - If 2 trees of equal ranks are merged, $\text{rank}(\text{new root}) = \text{rank}(\text{trees}) + 1$ ✓
 - Without path compression, this makes $\text{rank} = \text{height}$ ✓
 - After path compression, rank is upper bound to height ✓



```
Algorithm: UNION( $\tilde{u}, \tilde{v}$ )
1    $u \leftarrow \text{FIND-SET}(\tilde{u})$ 
2    $v \leftarrow \text{FIND-SET}(\tilde{v})$ 
3   if  $u.\text{rank} = v.\text{rank}$  then
4      $u.\text{rank} \leftarrow u.\text{rank} + 1$ 
5      $v.\text{parent} \leftarrow u$ 
6   else if  $u.\text{rank} > v.\text{rank}$  then
7      $v.\text{parent} \leftarrow u$ 
8   else
9      $u.\text{parent} \leftarrow v$ 
```

Pos \rightarrow	0	1	2	3	4	5	6
arr	0.	0	2	0	0	0	0
rand	2	0	0	1	0	1	0

?



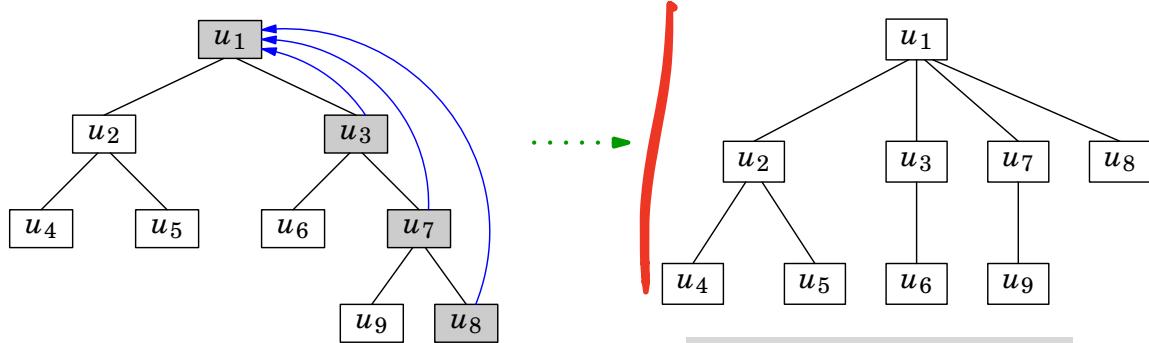
Find 6:

$A = \text{nodes to update}$
 $\Rightarrow \{6, 5\}.$

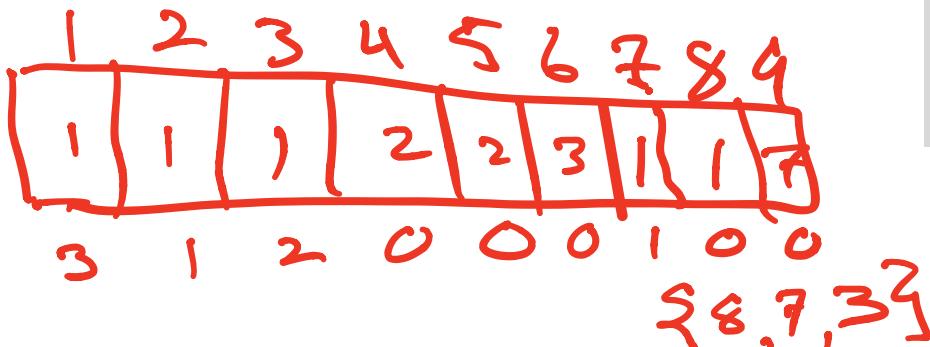
Find 4: $A = \{4, 3\}$

More on Rank and Path Compression

- Path compression illustration: Find u_8



- Rank does not change!
 - But height dropped by 1



Algorithm: FIND-SET(u)

```
1  $A \leftarrow \emptyset$ 
2  $n \leftarrow u$ 
3 while  $n$  is not the root do
4      $A \leftarrow A \cup \{n\}$ 
5      $n \leftarrow n.parent$ 
6 for each  $x \in A$  do
7      $x.parent \leftarrow n$ 
8 return  $n$ 
```

Outline of Proof of Complexity

$O(?)$?

- Properties of rank (without path compression): Let $p(x)$ be $\text{parent}(x)$
 - x is not a root node, then $\text{rank}[x] < \text{rank}[p[x]]$
 - Rank k root created by linking two roots of rank $k - 1$
 - x is not a root node, then $\text{rank}[x]$ never changes
 - $p[x]$ changes, then $\text{rank}[\text{parent}[x]]$ strictly increases
 - $p[x]$ only changes for root
 - When $p[x]$ changes, $\text{rank}[x] < \text{rank}[p[x]]$
 - $p[x]$ only changes for roots, when one root goes under other

```
Algorithm: UNION( $\tilde{u}, \tilde{v}$ )
1  $u \leftarrow \text{FIND-SET}(\tilde{u})$ 
2  $v \leftarrow \text{FIND-SET}(\tilde{v})$ 
3 if  $u.\text{rank} = v.\text{rank}$  then
4    $u.\text{rank} \leftarrow u.\text{rank} + 1$ 
5    $v.\text{parent} \leftarrow u$ 
6 else if  $u.\text{rank} > v.\text{rank}$  then
7    $v.\text{parent} \leftarrow u$ 
8 else
9    $u.\text{parent} \leftarrow v$ 
```

Outline of Proof of Complexity - 2

- Properties of rank (without path compression): Let $p(x)$ be parent(x)

- Root of rank k has $\geq 2^k$ elements in its tree

- Induction: true for $k = 0$

- Assume true for $k-1$. Root gets rank k by merging two roots of rank $k-1$. Thus, no. elements $\geq 2^{k-1} + 2^{k-1} \geq 2^k$

- Highest rank of node $\leq \lfloor \log_2(n) \rfloor$

- For any integer k , there are $\leq n/2^k$ nodes with rank k

- Non-root notes with rank k were roots with rank k at some point

- Every rank k node has at least 2^k elements,

$$\frac{n}{2^k}$$



Algorithm: UNION(\tilde{u}, \tilde{v})

```

1  $u \leftarrow \text{FIND-SET}(\tilde{u})$ 
2  $v \leftarrow \text{FIND-SET}(\tilde{v})$ 
3 if  $u.rank = v.rank$  then
4    $u.rank \leftarrow u.rank + 1$ 
5    $v.parent \leftarrow u$ 
6 else if  $u.rank > v.rank$  then
7    $v.parent \leftarrow u$ 
8 else
9    $u.parent \leftarrow v$ 

```

Outline of Proof of Complexity - 3

- Path compression: Let $p(x)$ be $\text{parent}(x)$
 - Key property: roots, node ranks, tree elements don't change!
All the properties of union by rank stay true
parent pointers updated to root means $\text{rank}(p(x)) > \text{rank}(x)$ still

- Iterated logarithm function

$$\log^*(n) = \begin{cases} 0 & n \leq 1 \\ 1 + \log^*(\log_2(n)) & \text{otherwise} \end{cases}$$

✓

n	$\lg^* n$
1	0
2	1
[3, 4]	2
[5, 16]	3
[17, 65536]	4
[65537, 2^{65536}]	5

Outline of Proof of Complexity - 4

- Analysis: Buckets of elements
 - Divide elements by non-zero ranks into the following groups

- $\{1\}, \{2, 3\}, \{4, \dots, 15\}, \dots, \{B, 2^B - 1\}, \{2^B, 2^{2^B} - 1\}$

- Total number of buckets $\leq \log^*(n)$

- Max elements in bucket $\{B, 2^B - 1\}$: $\leq n/2^B$

- Every node has rank in the first $\log^*(n)$ groups

- Creative accounting

- Every node receives credits the moment it ceases to be a root:

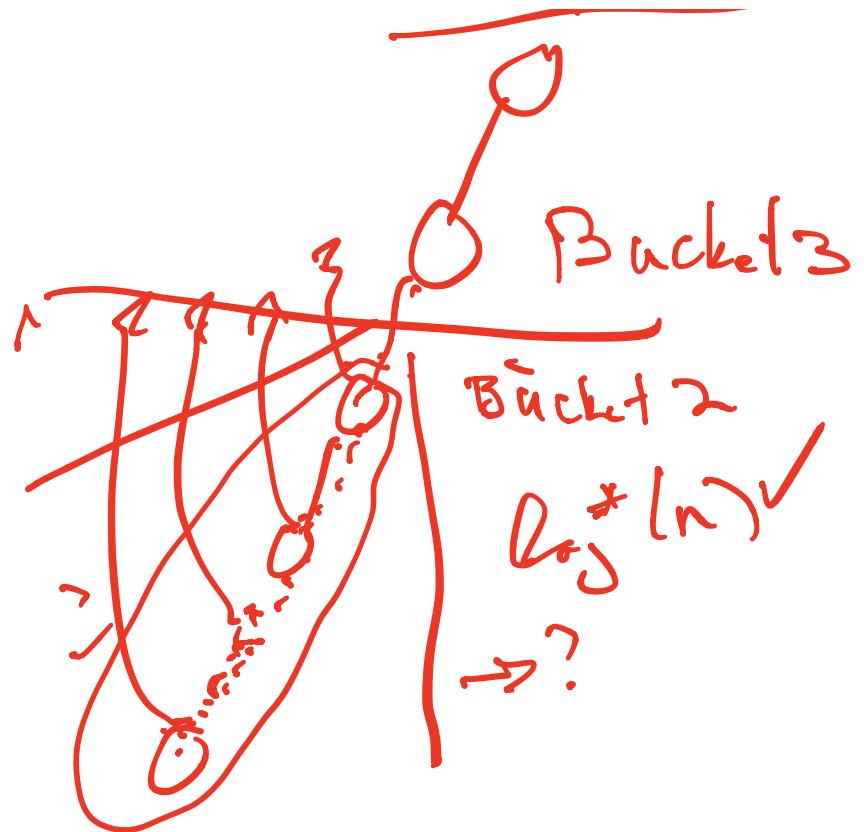
- If node rank is in $\{B, 2^B - 1\}$, get 2^B credits

- Number of credits disbursed to all nodes $\leq n \log_2(n)$

~~5 groups~~

$$\frac{2^B}{2} - B \approx 2^B$$

Σ Bucket 4



Outline of Proof of Complexity - 5

- Creative accounting
 - Every node receives credits the moment it ceases to be a root:
 - If node rank is in $\{B, 2^B - 1\}$, get 2^B credits
 - Number of credits disbursed to all nodes $\leq n \log^*(n)$
 - Max nodes in bucket $\{B, 2^B - 1\}$: $\leq n/2^B$, so max of n credits in bucket
 - Run time of Find(x): bounded by number of parent pointers to root
 - Rank increases strictly as you search up the tree
 - 3 Cases:
 - Case 0: $\text{parent}[x]$ is a root \rightarrow happens for 1 link per find
 - Case 1: $\text{rank}[\text{parent}[x]]$ is in higher bucket than $\text{rank}[x]$
 - Case 2: $\text{rank}[\text{parent}[x]]$ is in same bucket as $\text{rank}[x]$

Outline of Proof of Complexity - 6

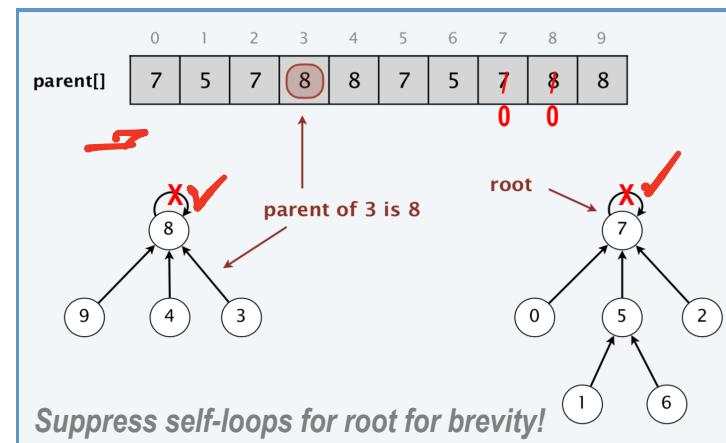
- Case 1: $\text{rank}[\text{parent}[x]]$ is in higher bucket than $\text{rank}[x]$
 - At most $\log^*(n)$ links have this property
- Case 2: charge 1 credit to follow parent pointer
 - If $\text{rank}[x]$ is in bucket $\{B, 2^B - 1\}$, x will move to Case 1 before paying all its credits (and will still have some left over)
- So, since you started with $\leq n \log^*(n)$ credits across all nodes, and you spend less than that across all find operations, total amortized cost of m operations, for $m \geq n$, is $O(m \log^*(n))$

1

.

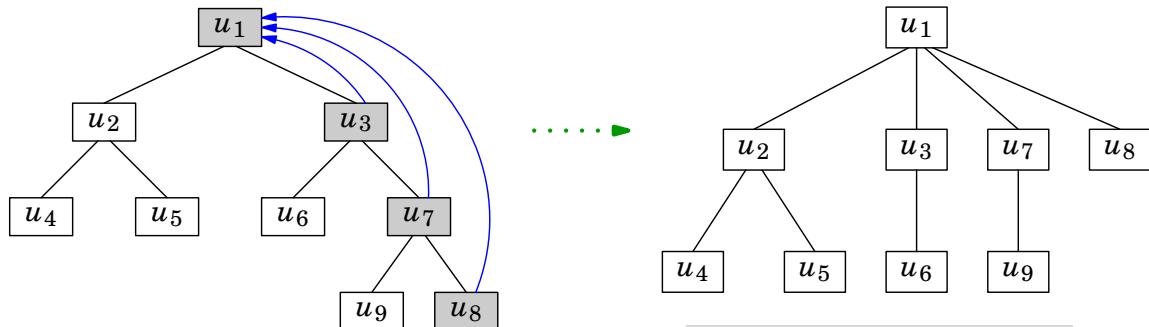
Representation of Disjoint Set Operations

- **Array:** Objects indexed in the array, with value pointing to parent location
 - Roots point to self-location
 - Keep track of separate array for ranks
- How is rank assigned?
 - Rank of single nodes = 0
 - If 2 trees of equal ranks are merged, $\text{rank}(\text{new root}) = \text{rank}(\text{trees}) + 1$
 - Without path compression, this makes $\text{rank} = \text{height}$
 - After path compression, rank is upper bound to height



Bigger Example

- Path compression illustration: Find u_8



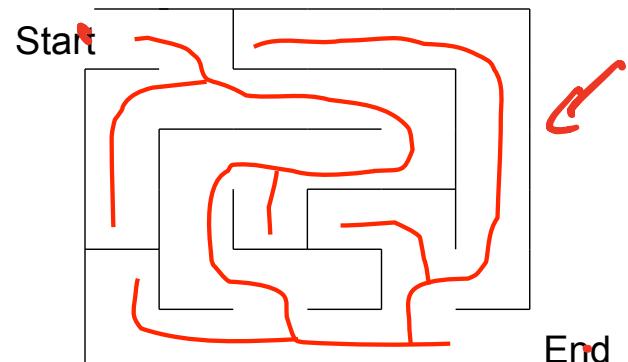
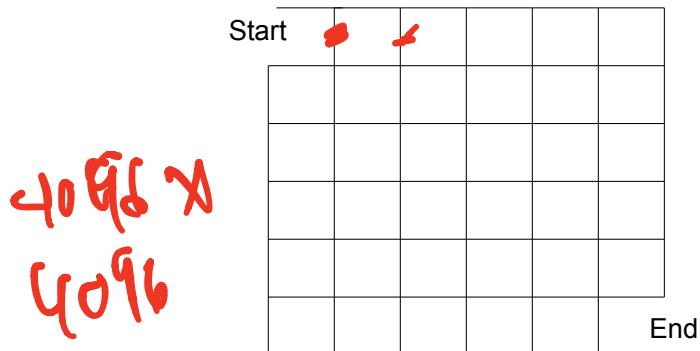
- Rank does not change!
 - But height dropped by 1

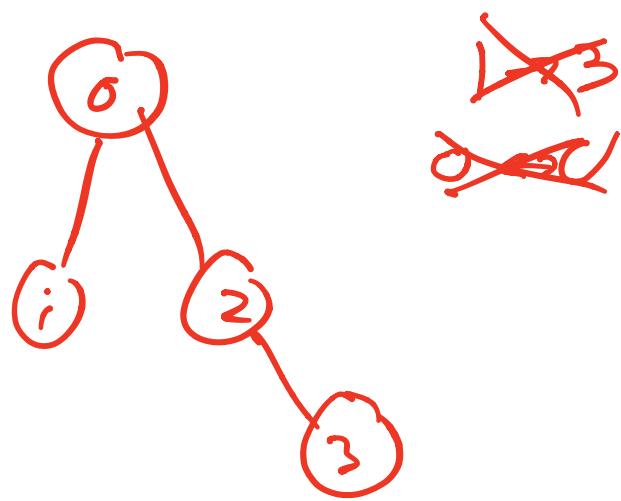
Algorithm: FIND-SET(u)

```
1  $A \leftarrow \emptyset$ 
2  $n \leftarrow u$ 
3 while  $n$  is not the root do
4      $A \leftarrow A \cup \{n\}$ 
5      $n \leftarrow n.parent$ 
6 for each  $x \in A$  do
7      $x.parent \leftarrow n$ 
8 return  $n$ 
```

Application

- Maze Building: take out walls to create a maze
 - Consider edges in random order (i.e. pick an edge)
 - Only delete an edge if it introduces no cycles (how? TBD)
 - When done, we will have a way to get from any place to any other place (including from start to end points)

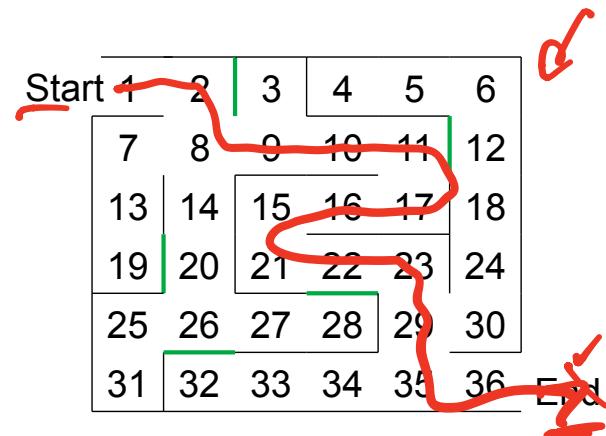




Application 2

- Maze Building: take out walls to create a maze
 - Number the cells; every edge allows a relation between cells
 - Keep track of disjoint sets: don't remove edge if it is between two nodes in same subset

Start	1	2	3	4	5	6
	7	8	9	10	11	12
	13	14	15	16	17	18
	19	20	21	22	23	24
	25	26	27	28	29	30
	31	32	33	34	35	36
End						



End of Exam 1:

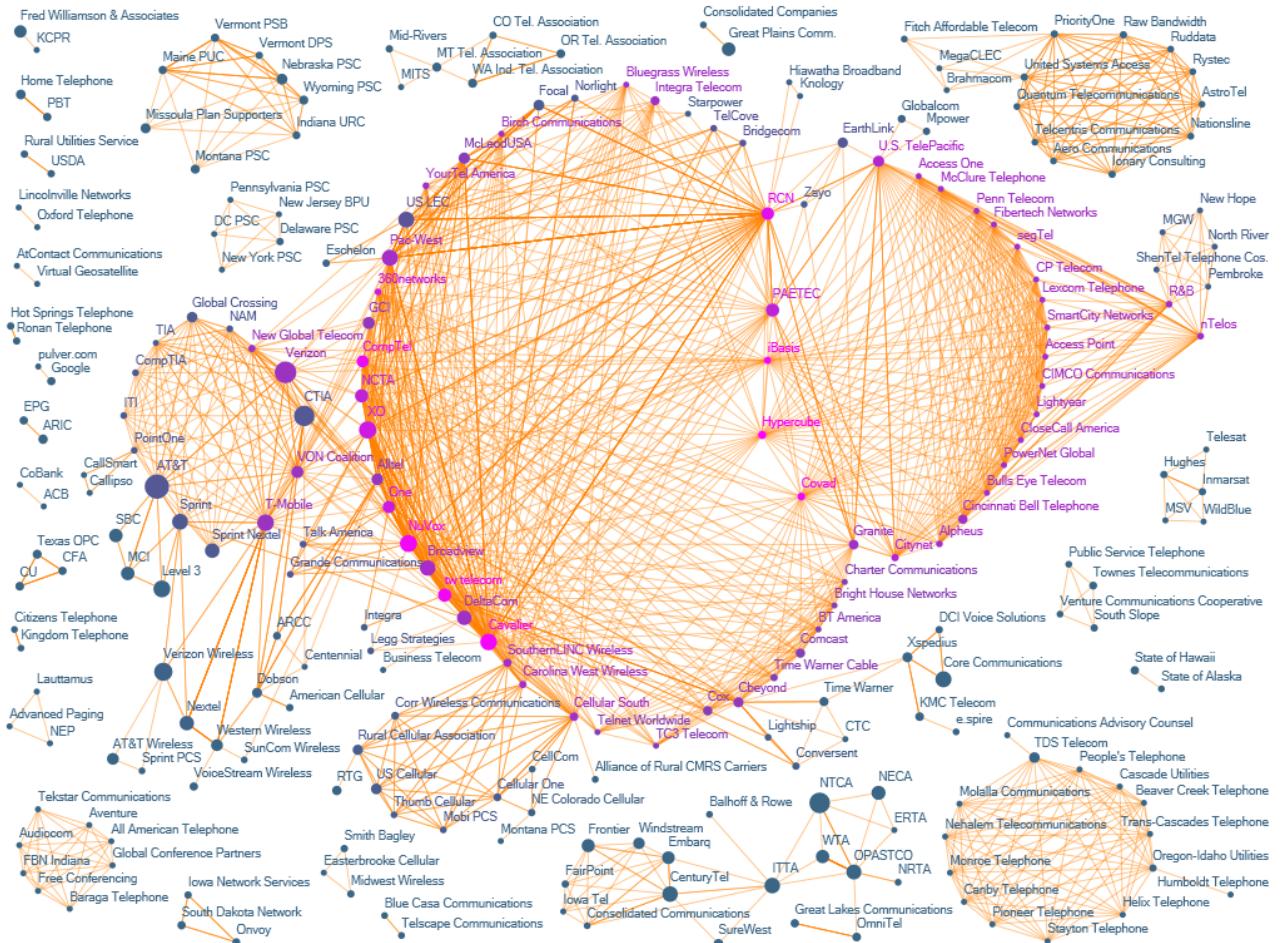
Graphs

in the "real world"



- Networks are graphs
 - Information: WWW, citation, ...
 - Social: co-actor, dating, messenger, communities, ...
 - Technological: Internet, power grids, airline routes, ...
 - Biological: Neural networks, food web, blood vessels, ...
- Object hierarchies are graphs
- Circuit layouts are graphs
- Computer programs are graphs

The evolution of FCC lobbying coalitions



"The Evolution of FCC Lobbying Coalitions" by Pierre de Vries in JoSS Visualization Symposium 2010

Undirected Graph $G=(V,E)$ (or $G=(N,A)$)

V : Set of vertices (nodes...)

Edge $\{a,b\}$ is an unordered pair of nodes

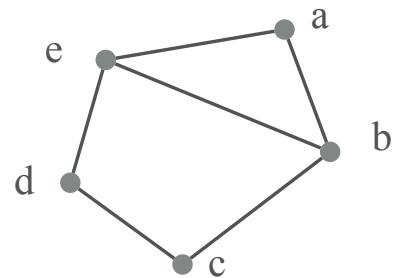
$E \subset V \times V$: Set of vertices (arcs...)

Undirected graph: $G=(V,E)$

- Typically, don't allow $\{a,a\}$ in E

Vertex Set: $V = (a,b,c,d,e)$

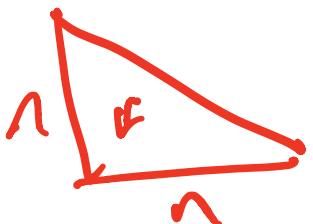
Edge Set: $E = (\{a,b\}, \{b,c\}, \{c,d\}, \{d,e\}, \{e,a\}, \{e,b\})$



Undirected Graph $G(V,E)$

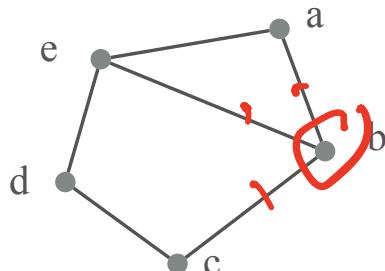
Maximum number of edges:

$$|E| \leq \binom{|V|}{2} = \frac{|V|(|V|-1)}{2} = O(|V|^2)$$

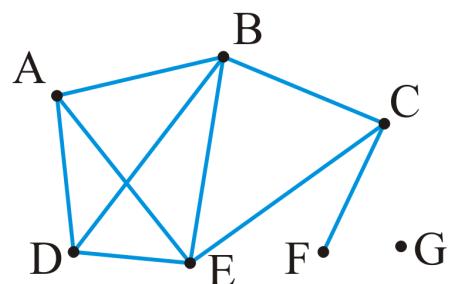


Degree of vertex: number of adjacent vertices

Vertex a **adjacent** to vertex b $\iff \{a,b\} \in E$



$$\begin{aligned}\text{degree}(b) &= 3 \\ \text{degree}(c) &= 2\end{aligned}$$

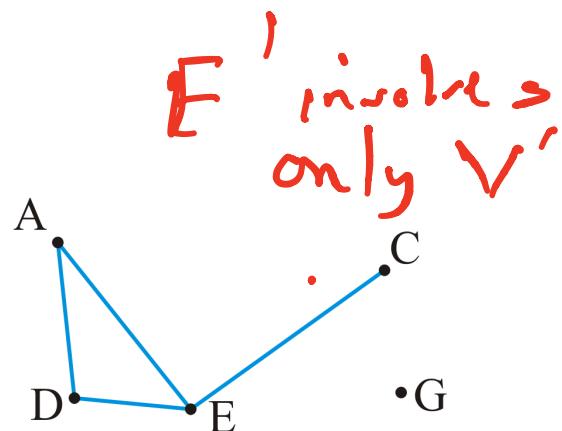
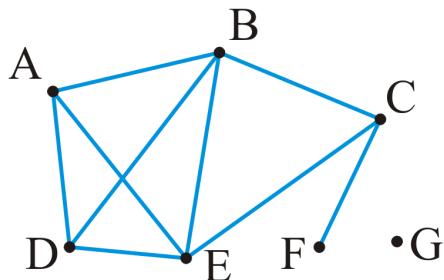


$$\begin{aligned}\text{degree}(E) &= 4 \\ \text{degree}(G) &= 0\end{aligned}$$

Subgraphs (V', E') of (V, E)

(V', E') is a subgraph of (V, E) if and only if

- $V' \subset V$, and $E' \subset E$
- If $\{a, b\} \in E' \Rightarrow a, b \in V'$

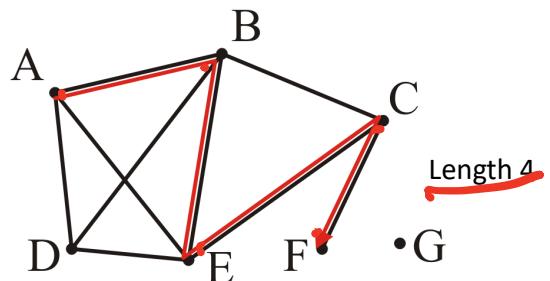
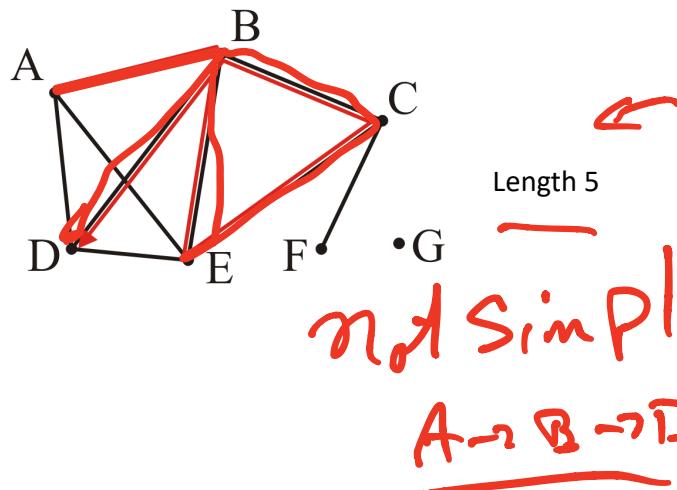


Paths in (V, E)

A path in an undirected graph is an ordered sequence of vertices $(v_0, v_1, v_2, \dots, v_k)$



- $\{v_{j-1}, v_j\}$ is an edge in E for $j = 1, \dots, k$
- Termed a path from v_0 to v_k
- The length of this path is k



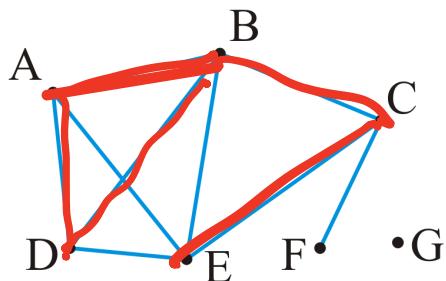
A -> B -> E -> C -> B -> D

Simple paths in (V,E)

Simple path: no repeated nodes other than perhaps, the first and last vertices

Simple cycle: simple path with first and last nodes the same

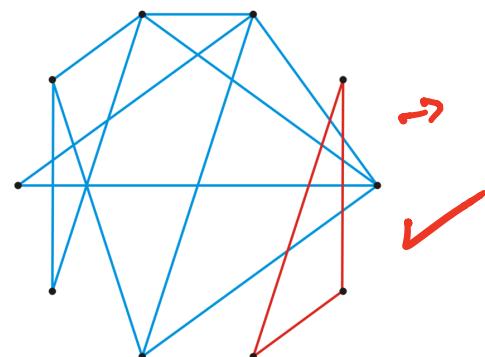
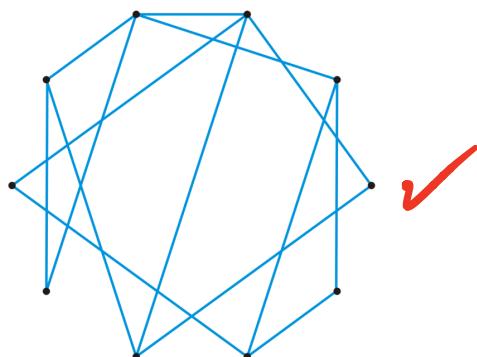
- Example: (A,B,C,E) simple path
- (A,B,E,C,B,D) Not simple path ✓
- (A,B,D,A) simple cycle



Connectedness

Vertices a, b in V are connected if there exists a path from a to b

Graph(V, E) is connected if, for every a, b in V , a and b are connected



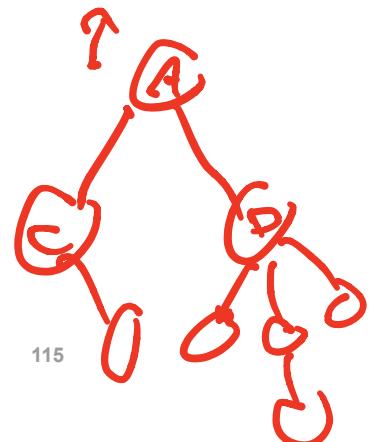
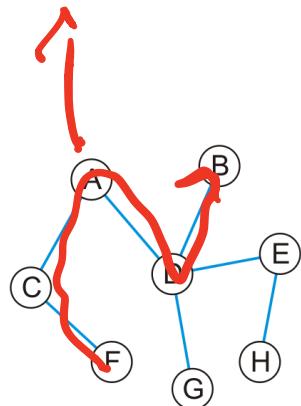
Trees

Graph(V,E) is a tree if it is connected, and

- there is only one path between any two vertices a, b
- Equivalently, there are no cycles in the graph

Trees: $|E| = |V| - 1$

- Adding one more edge to a tree creates a cycle
- Removing an edge from a tree disconnects the graph
- Can convert to rooted tree picking any vertex as root



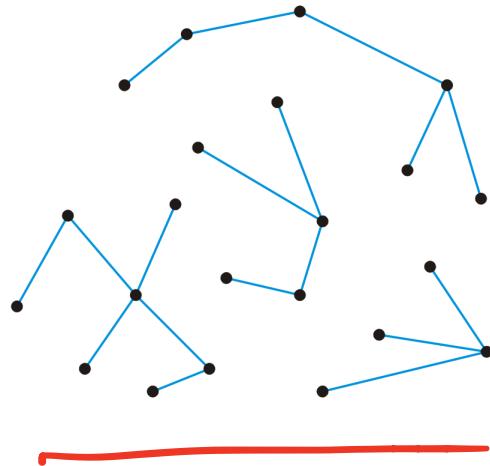
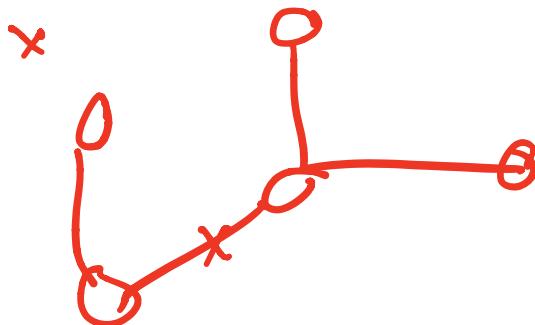
Forests

Forest is a graph (V, E) with no cycles

- Forest can be multiple disjoint trees

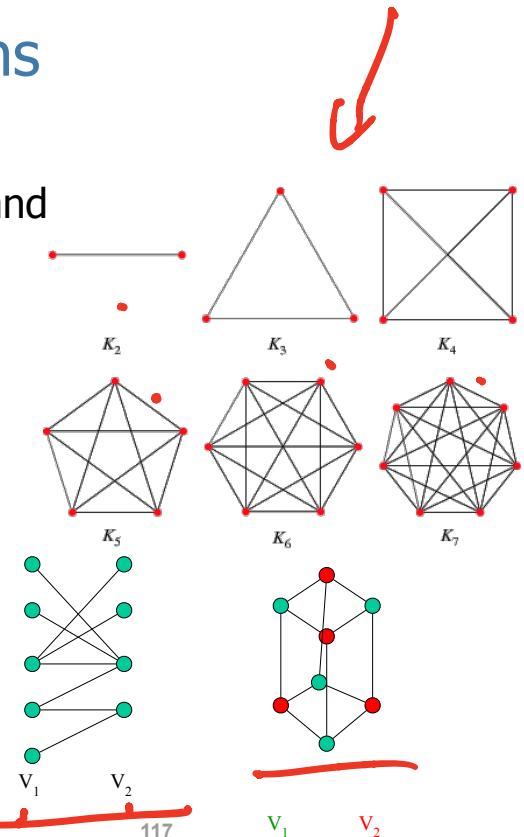
Properties:

- $|E| < |V|$
- Number of trees = $|V| - |E|$
- Removing edge: adds one more tree



Some Specific Graphs

Complete graph K_n : undirected graph with n vertices, and an edge between every pair of vertices



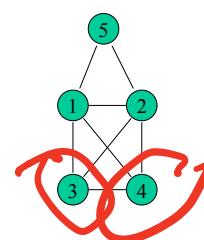
Bipartite graph $K_{m,n}$: undirected graph where V can be partitioned into vertex set V can be partitioned into disjoint, nonempty sets V_1 and V_2 such that

every edge connects a vertex in V_1 to a vertex in V_2

Equivalent: every cycle has even length!

Results in Undirected Graphs

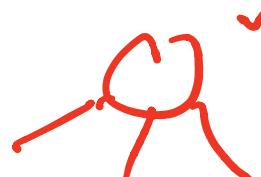
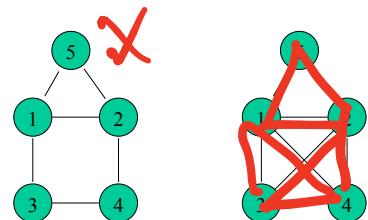
Let (V, E) be undirected. Then, $\sum_{k \in V} \text{degree}(k) = 2|E|$



- Every edge connects 2 vertices...
—> must have even number of odd-degree vertices

A path is Eulerian if it covers every edge in the graph exactly once

- Draw it without lifting pencil
- Theorem: a connected graph has an Eulerian cycle
if and only if every node has even degree



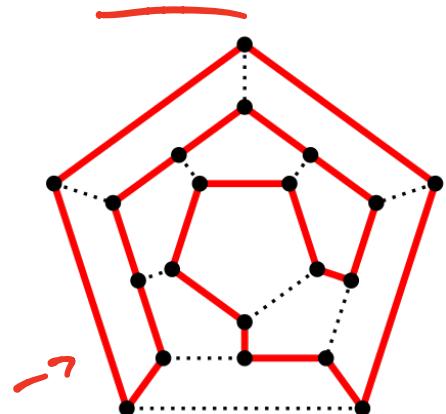
Results in Undirected Graphs

A path is Hamiltonian if it covers every vertex in the graph exactly once; it is a cycle if it covers every vertex but the start and end vertices exactly once

- Theorem: A graph with n vertices has Hamiltonian cycle if degree of every vertex is at least $n/2$

- Theorem: For a graph G with $n \geq 3$ vertices, and $\text{degree}(a) + \text{degree}(b) \geq n$ whenever a, b are not adjacent, then G has a Hamiltonian cycle

- Finding a Hamiltonian cycle is a hard problem!

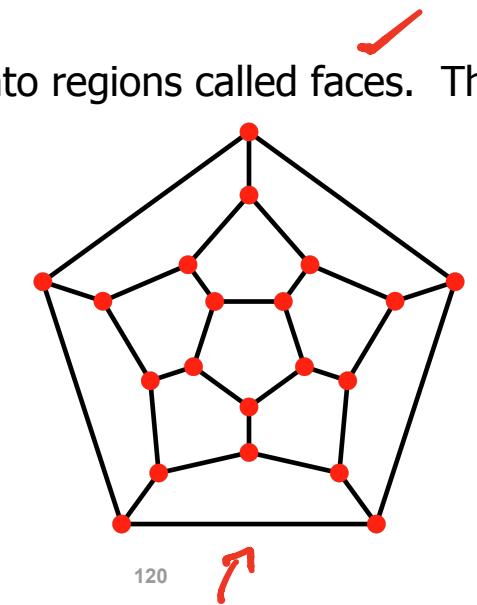
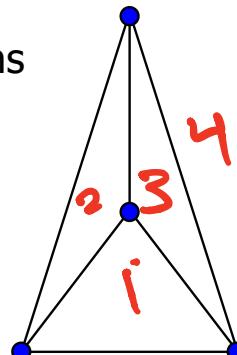


Planar Graphs

A graph G is planar if it can be drawn in the plane so that no edges cross

- Useful for VLSI, circuit design
- Euler's Formula: a planar graph divides the plane into regions called faces. The number of regions $r = m-n+2$
- Sufficient conditions for planar graphs

$$\begin{aligned} & \bullet m \leq 3n - 6 \\ & \bullet r \leq 2n - 4 \end{aligned}$$



Directed Graph G=(V,E)

V is the set of vertices.

An edge $e = \underline{(a,b)}$ is an ordered pair from a to b, where a, b in V

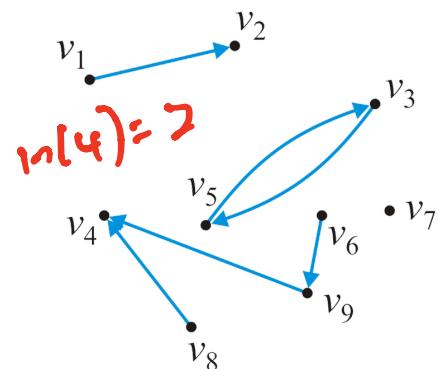
Note: edge $\underline{(b,a)}$ is different from edge $\underline{(a,b)}$

Example: streets, ...

Max. no. edges = $|V|^2$ (allow self loops)

Out-degree of a vertex: no. edges leaving vertex

In-degree of a vertex: no. edges entering vertex



$$E = \{(v_1, v_2), (v_3, v_5), (v_5, v_3), (v_5, v_6), (v_6, v_9), (v_8, v_4), (v_9, v_4)\}$$

Sources, Sinks, Paths

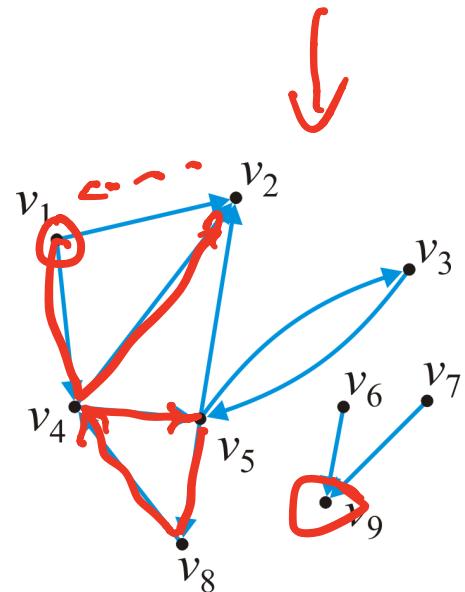
Vertex with in-degree 0: **Source** (V_1, V_6, V_7)

Vertex with out_degree 0: **Sink** (V_9)

Path in directed graph (V, E) is an ordered sequence of vertices $(v_0, v_1, v_2, \dots, v_k)$ such that $(v_{j-1}, v_j) \in E$ for $j = 1, \dots, k$

Simple path: no repeated nodes in path

Cycle: path with $v_0 = v_k$. **Simple cycle**: no repeated nodes except start-end



$v_4 \rightarrow v_5 \rightarrow v_8 \rightarrow v_4$

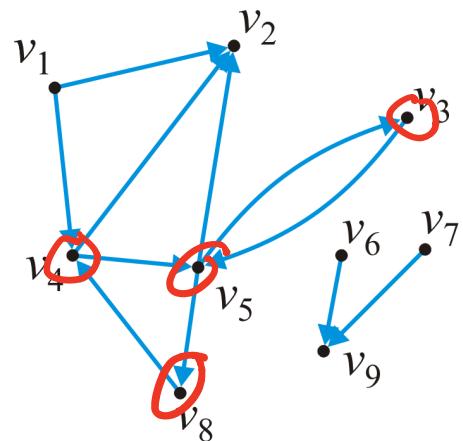
$v_3 \rightarrow v_5 \rightarrow v_3$

$v_1 \rightarrow v_4 \rightarrow v_5 \rightarrow v_1$

Connectedness in Directed Graphs

Vertex a is connected to vertex b if there exists a path from a to b (directed path)

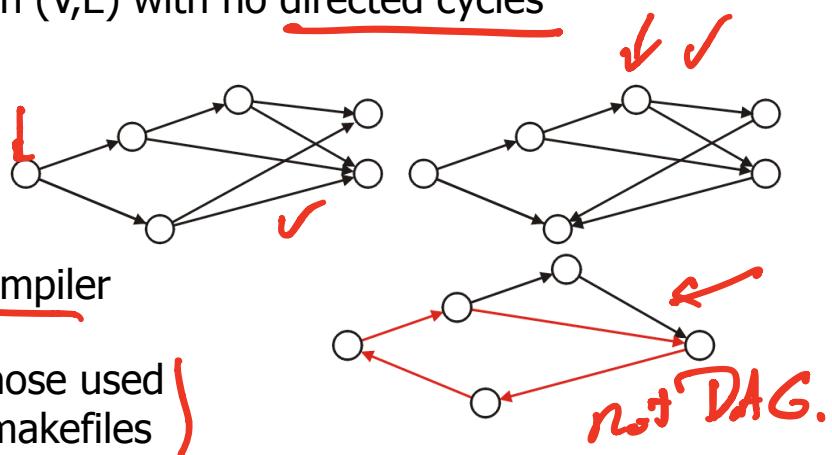
- Graph (V, E) is **strongly connected** if there exists directed path between any two vertices
- Graph (V, E) is **weakly connected** if there exists undirected path between any two vertices
 - The sub-graph $\{v_3, v_4, v_5, v_8\}$ is strongly connected
 - The sub-graph $\{v_1, v_2, v_3, v_4, v_5, v_8\}$ is weakly connected



Directed Acyclic Graphs

Directed acyclic graph (DAG): Graph (V, E) with no directed cycles

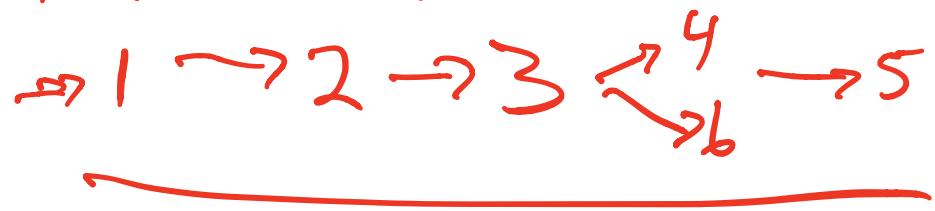
- Represents partial order
- DAGs appear often
- Parse tree constructed by a compiler
- Dependency graphs such as those used in instruction scheduling and makefiles
- Dependency formed by inheritance relationships in object-oriented programming languages



Topological Sort.

>(5)

Execution order:



Representations of graphs

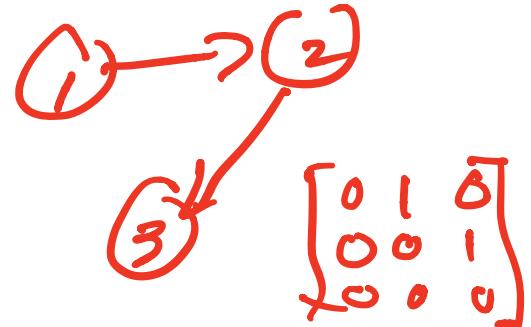
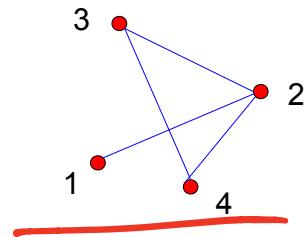
- Adjacency matrix: ✓

- $A = [a_{ij}]$, where

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{if } (i, j) \notin E \end{cases}$$

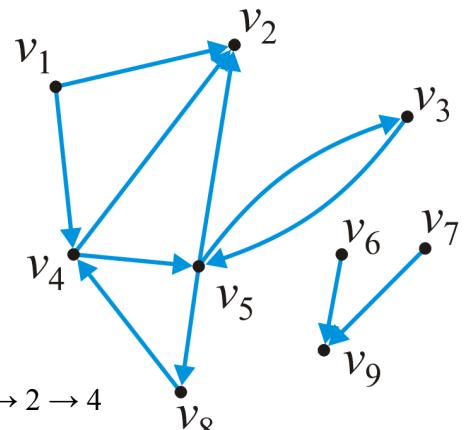
- Symmetric for undirected graphs

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2 & 3 & 4 \end{bmatrix} \cdots \begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \end{array}$$



Adjacency List

- Each vertex has list of neighbor nodes
 - For directed graphs, out-list ✓
 - Used in MATLAB for sparse matrices
 - Can be implemented as arrays
 - Forward-Star
 - or linked lists
 - $|V| + |E|$ for directed graphs
 - Must store 2 edges for undirected



1	• → 2 → 4
2	• -
3	• → 5 •
4	• → 2 → 5 •
5	• → 2 → 3 → 8
6	• → 9
7	• → 9
8	• → 4
9	•

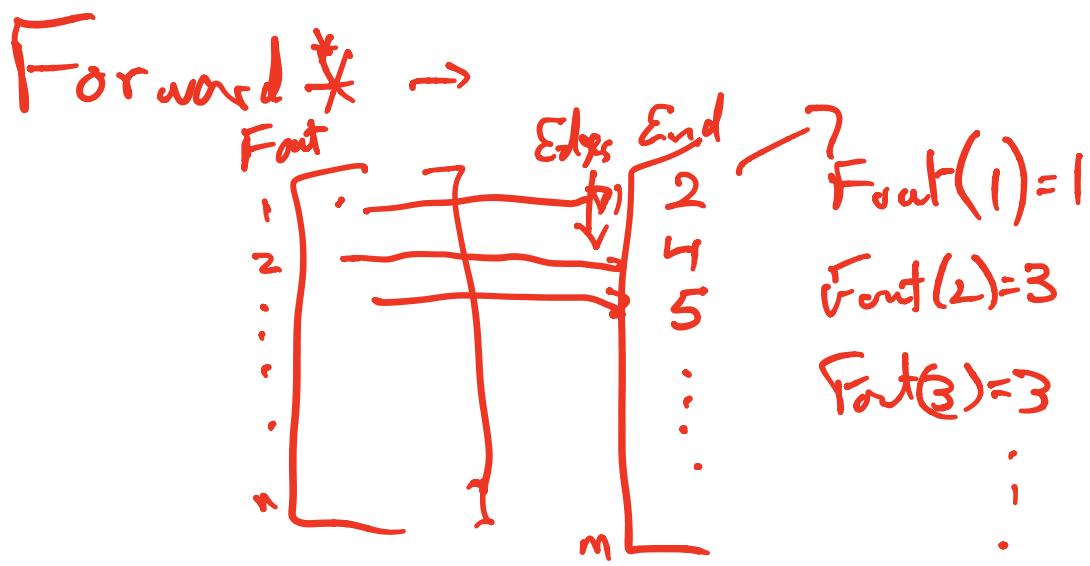
$1 \rightarrow (2, 19, 4, 25)$;

$2 \rightarrow 1, 3, 20$;

;

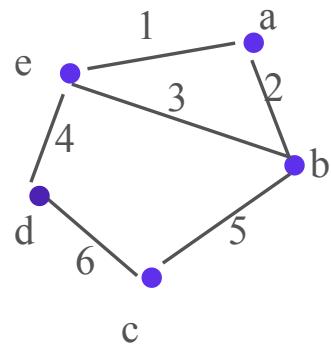
;

| Fortran \rightarrow Matlab ...
arrays ...



V vertices
E edges

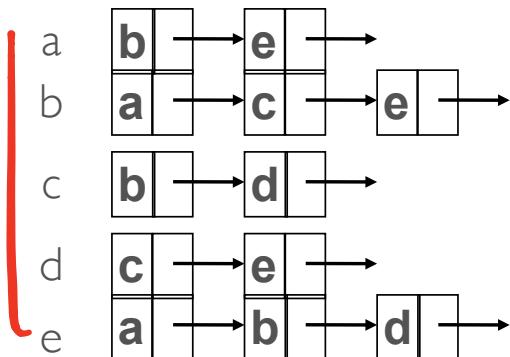
1. Edge list: {a,b}, {b,c}, {c,d}, {d,e}, {e,a}, {e,b}

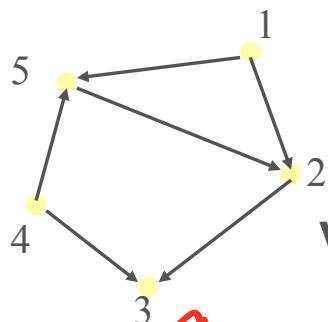


2. Adjacency list

Vertex	Adjacencies
a	b, e
b	a, c, e
c	b, d
d	c, e
e	a, b, d

Linked List Form





Forward Star Representation

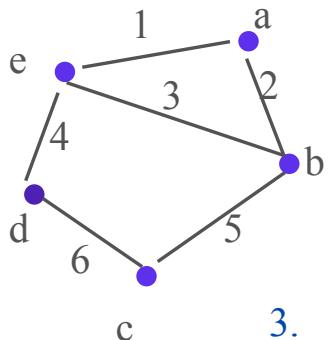
Vertex Array First[i]

i = 1	1	→	2	j = 1
i = 2	3	→	5	j = 2
i = 3	4	→	3	j = 3
i = 4	4	→	3	j = 4
i = 5	6	→	5	j = 5
i = 6	7	→	2	j = 6
		→	-1	j = 7

Edges[j]

2	j = 1
5	j = 2
3	j = 3
3	j = 4
5	j = 5
2	j = 6
-1	j = 7

Just the adjacency list put end to end in the arc array!



$A =$

3. Adjacency matrix:

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

4. Incidence matrix:

Matrix of vertex rows, edges columns

$M =$

Directed: -1 in start of edge, 1 in end

Undirected: 1 in both

	1	2	3	4	5	6
a	1	1	0	0	0	0
b	0	1	1	0	1	0
c	0	0	0	0	1	1
d	0	0	0	1	0	1
e	1	0	1	1	0	0