

CSE 332 Winter 2026

Lecture 14: Graphs

Nathan Brunelle

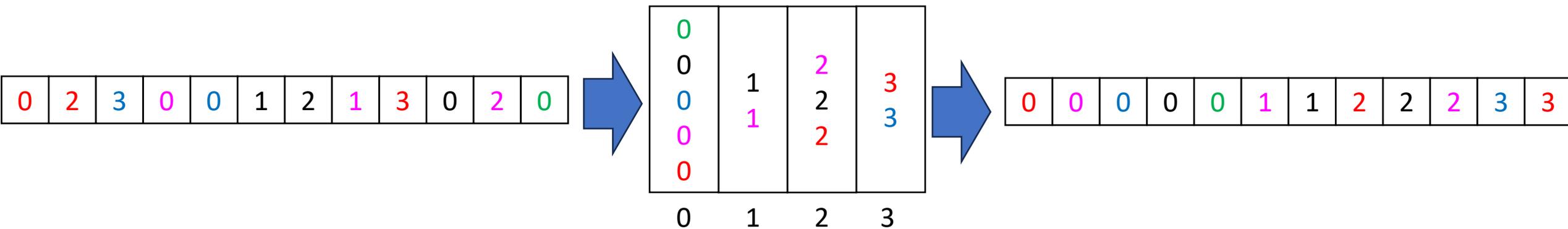
<http://www.cs.uw.edu/332>

“Linear Time” Sorting Algorithms

- Useable when you are able to make additional assumptions about the contents of your list (beyond the ability to compare)
 - Examples:
 - The list contains only positive integers less than k
 - The number of distinct values in the list is much smaller than the length of the list
- The running time expression will always have a term other than the list's length to account for this assumption
 - Examples:
 - Running time might be $\Theta(k \cdot n)$ where k is the range/count of values

BucketSort

- Assumes the array contains integers between 0 and $k - 1$ (or some other small range)
- Idea:
 - Use each value as an index into an array of size k
 - Add the item into the “bucket” at that index (e.g. linked list)
 - Get sorted array by “appending” all the buckets



BucketSort Running Time

- Create array of k buckets
 - Either $\Theta(k)$ or $\Theta(1)$ depending on some things...
- Insert all n things into buckets
 - $\Theta(n)$
- Empty buckets into an array
 - $\Theta(n + k)$
- Overall:
 - $\Theta(n + k)$
- When is this better than mergesort?

Properties of BucketSort

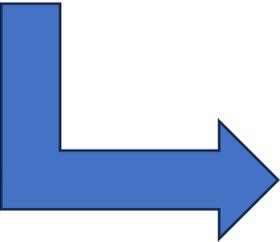
- In-Place?
 - No
- Adaptive?
 - No
- Stable?
 - Yes!

RadixSort

- Radix: The base of a number system
 - We'll use base 10, most implementations will use larger bases
- Idea:
 - BucketSort by each digit, one at a time, from least significant to most significant

103	801	401	323	255	823	999	101	113	901	555	512	245	800	018	121
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Place each element into a "bucket" according to its 1's place



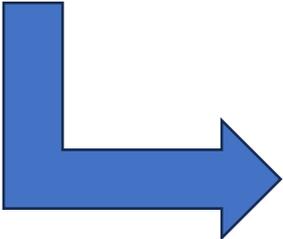
800	801 401 101 901 121	512	103 323 823 113		255 555 245			018	999
0	1	2	3	4	5	6	7	8	9

RadixSort

- Radix: The base of a number system
 - We'll use base 10, most implementations will use larger bases
- Idea:
 - BucketSort by each digit, one at a time, from least significant to most significant

800	801 401 101 901 121	512	103 323 823 113		255 555 245			018	999
0	1	2	3	4	5	6	7	8	9

Place each element into a "bucket" according to its 10's place



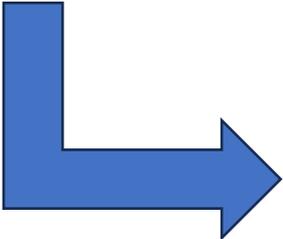
800									
801	512	121							
401	113	323		245	255				999
101	018	823			555				
901									
103									
0	1	2	3	4	5	6	7	8	9

RadixSort

- Radix: The base of a number system
 - We'll use base 10, most implementations will use larger bases
- Idea:
 - BucketSort by each digit, one at a time, from least significant to most significant

800									
801									
401	512	121			255				999
101	113	323		245	555				
901	018	823							
103									
0	1	2	3	4	5	6	7	8	9

Place each element into a "bucket" according to its 100's place

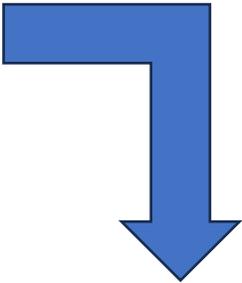


	101							800	901
018	103	245	323	401	512			801	999
	113	255			555			823	
	121								
0	1	2	3	4	5	6	7	8	9

RadixSort

- Radix: The base of a number system
 - We'll use base 10, most implementations will use larger bases
- Idea:
 - BucketSort by each digit, one at a time, from least significant to most significant

018	101 103 113 121	245 255	323	401	512 555			800 801 823	901 999
0	1	2	3	4	5	6	7	8	9



Convert back into an array

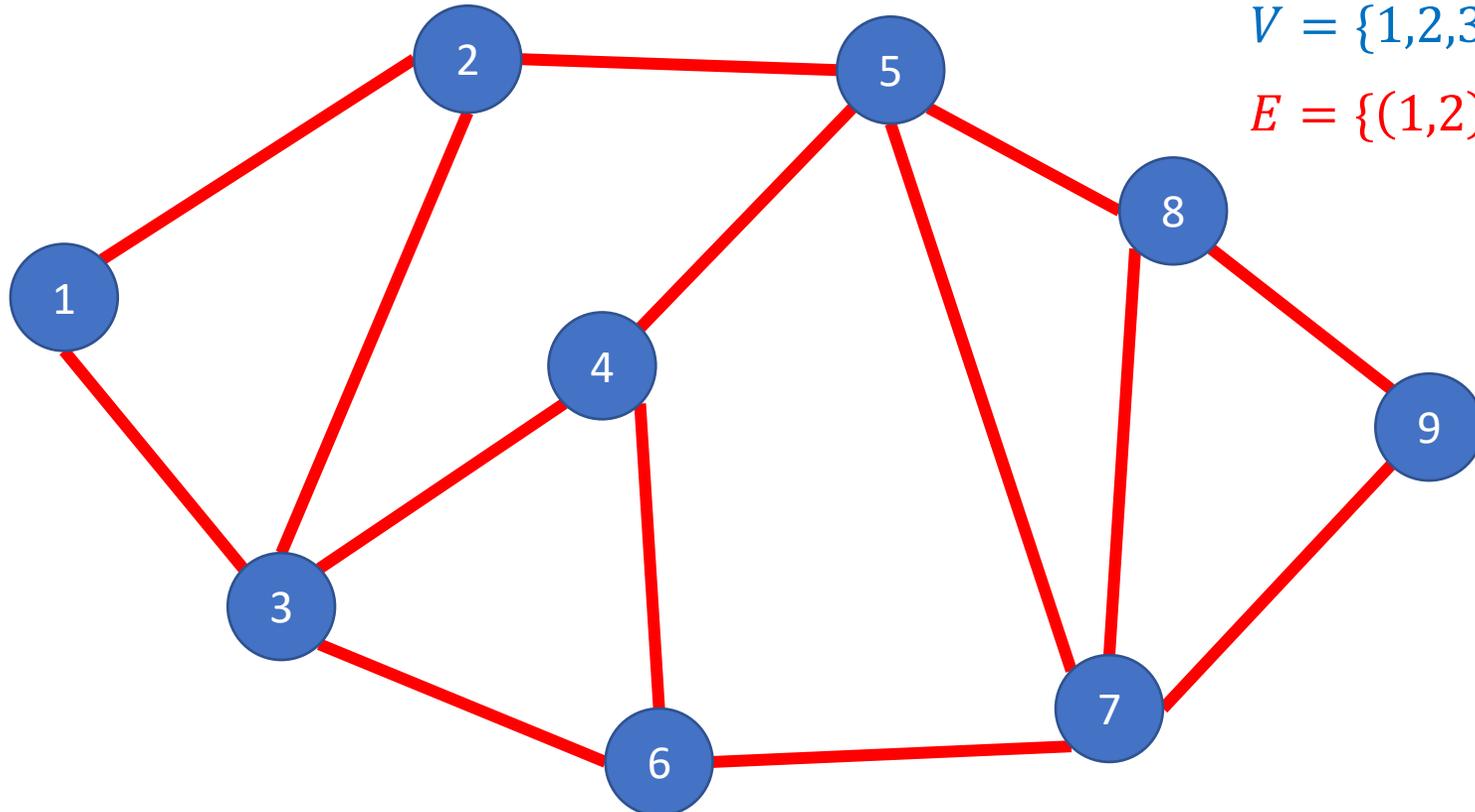
018	101	103	113	121	245	255	323	401	512	555	800	801	823	901	999
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

RadixSort Running Time

- Suppose largest value is m
- Choose a radix (base of representation) b
- BucketSort all n things using b buckets
 - $\Theta(n + k)$
- Repeat once per each digit
 - $\log_b m$ iterations
- Overall:
 - $\Theta(n \log_b m + b \log_b m)$
- In practice, you can select the value of b to optimize running time
- When is this better than mergesort?

Undirected Graphs

Definition: $G = (V, E)$
Vertices/Nodes
Edges

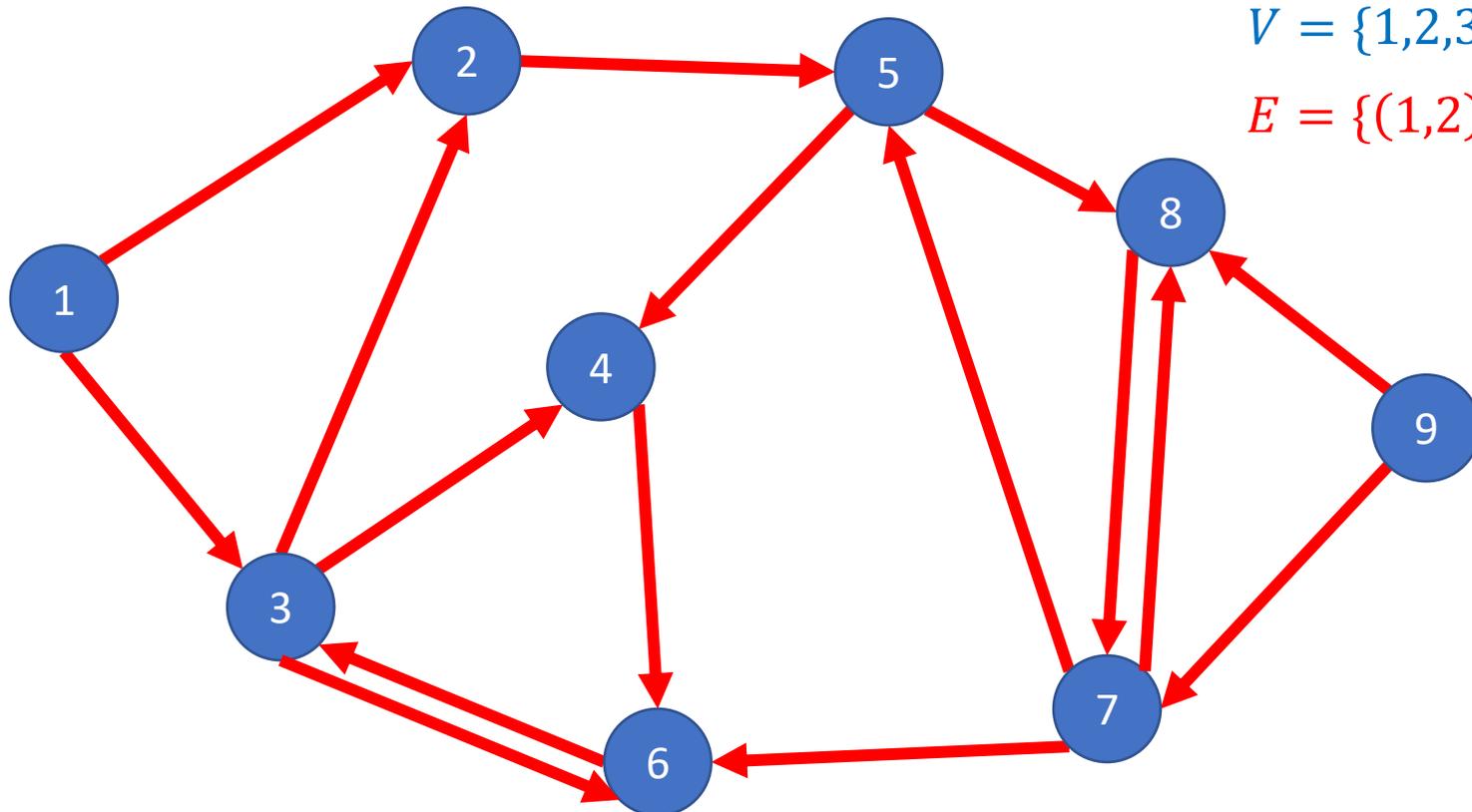


$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$E = \{(1, 2), (2, 3), (1, 3), \dots\}$

Directed Graphs

Definition: $G = (V, E)$
Vertices/Nodes
Edges

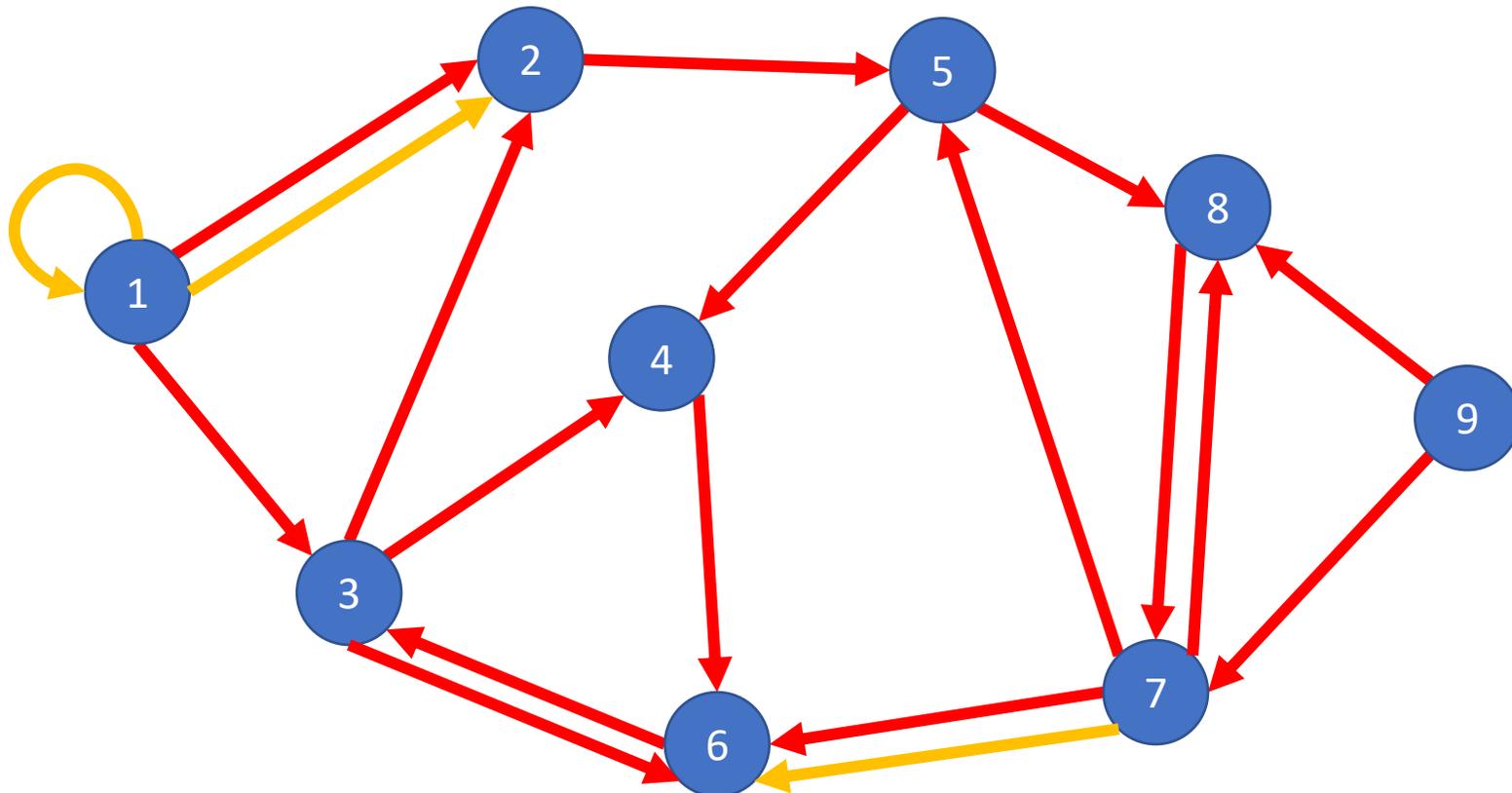


$V = \{1,2,3,4,5,6,7,8,9\}$

$E = \{(1,2), (3,2), (1,3), \dots\}$

Self-Edges and Duplicate Edges

Some graphs may have duplicate edges (e.g. here we have the edge (1,2) twice). Some may also have self-edges/loops (e.g. here there is an edge from 1 to 1). Graph with neither self-edges nor duplicate edges are called **simple graphs**



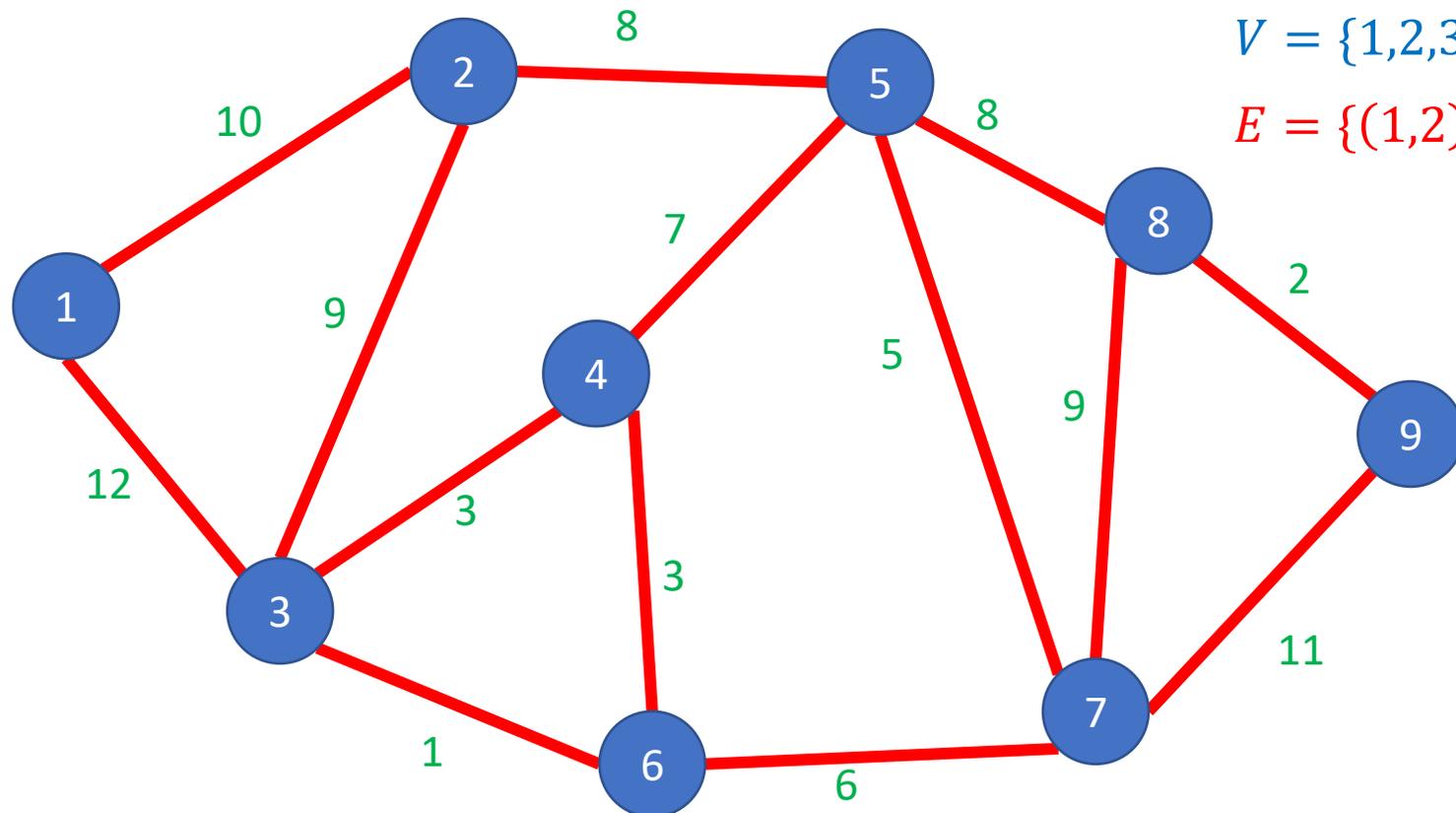
Weighted Graphs

Vertices/Nodes

Definition: $G = (V, E)$

Edges

$w(e)$ = weight of edge e



$V = \{1,2,3,4,5,6,7,8,9\}$

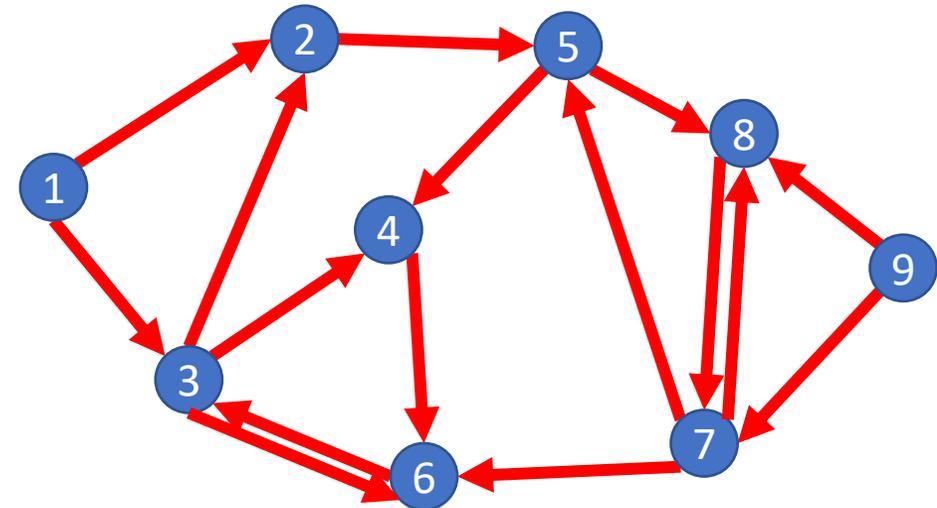
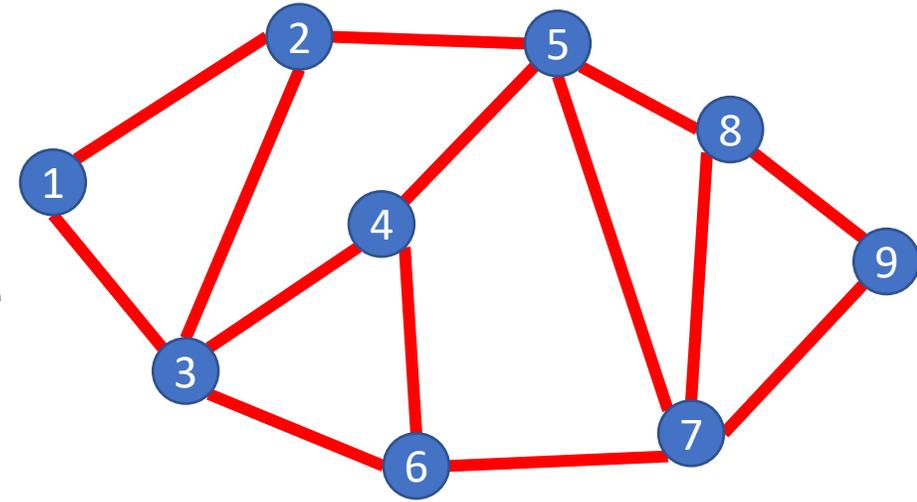
$E = \{(1,2), (2,3), (1,3), \dots\}$

Graph Applications

- For each application below, consider:
 - What are the nodes, what are the edges?
 - Is the graph directed?
 - Is the graph simple?
 - Is the graph weighted?
- LinkedIn Connections
- Twitter Followers
- Java Inheritance
- Airline Routes
- Course Prerequisites

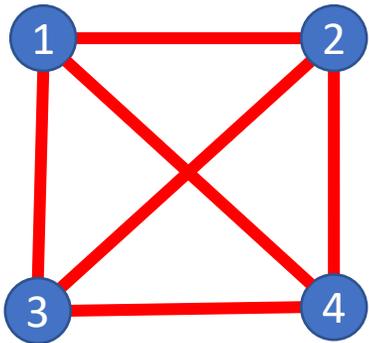
Some Graph Terms

- Adjacent/Neighbors
 - Nodes are adjacent/neighbors if they share an edge
- Degree
 - Number of edges “touching” a vertex
- Indegree
 - Number of incoming edges
- Outdegree
 - Number of outgoing edges

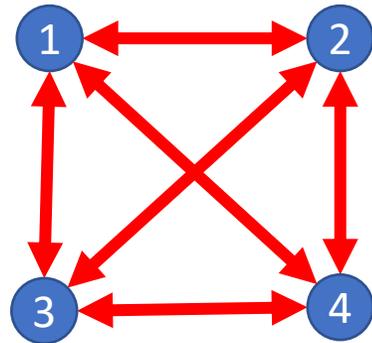


Definition: Complete Graph

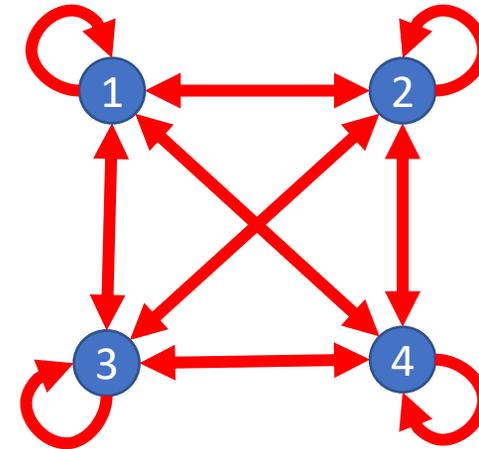
A Graph $G = (V, E)$ s.t. for any pair of nodes $v_1, v_2 \in V$ there is an edge from v_1 to v_2



Complete
Undirected Graph



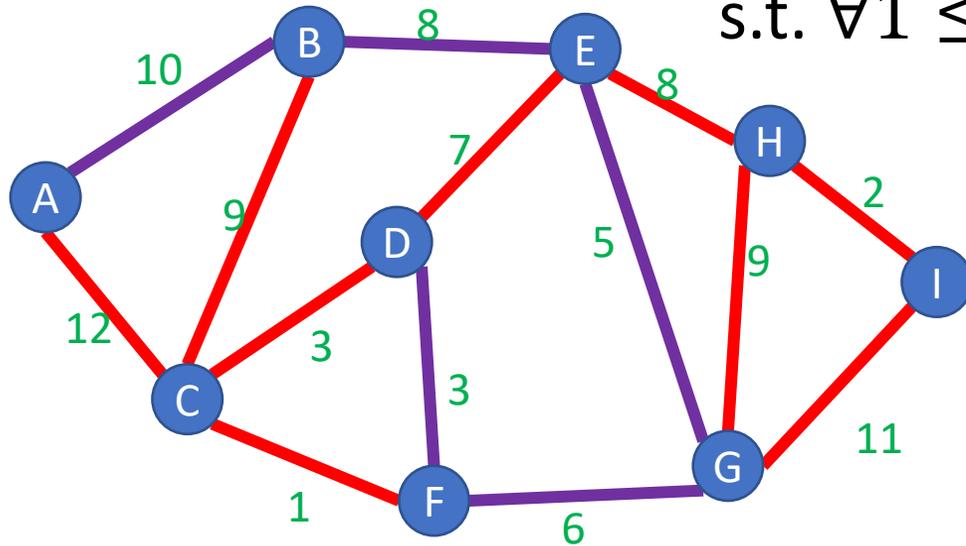
Complete
Directed Graph



Complete Directed
Non-simple Graph

Definition: Path

A sequence of nodes (v_1, v_2, \dots, v_k)
s.t. $\forall 1 \leq i \leq k - 1, (v_i, v_{i+1}) \in E$



Simple Path:

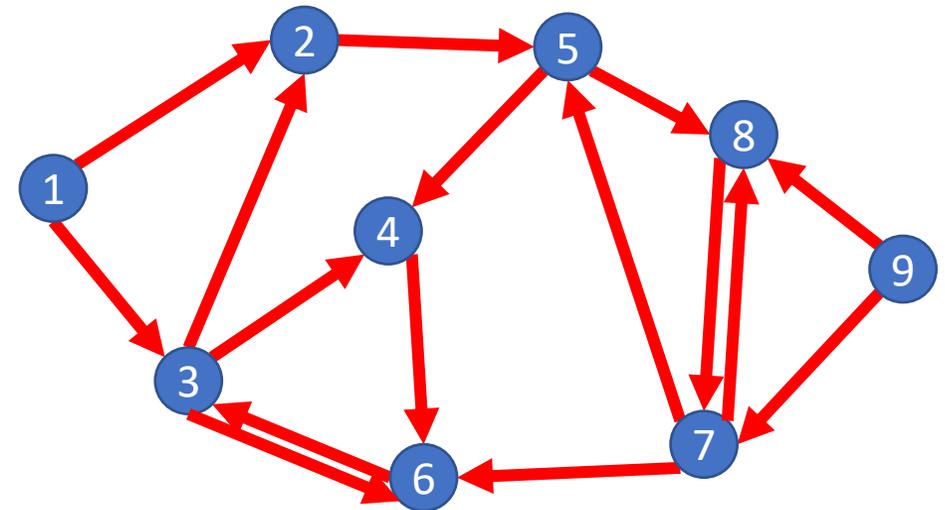
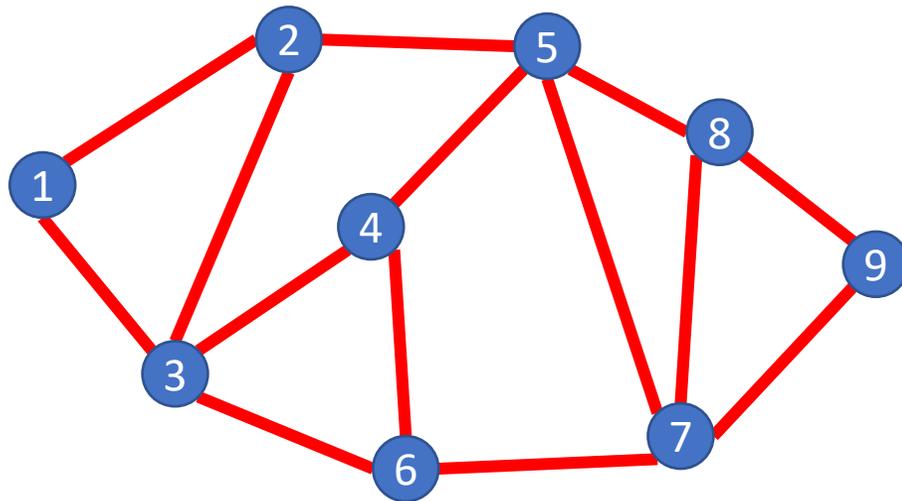
A path in which each node appears at most once

Cycle:

A path which starts and ends in the same place

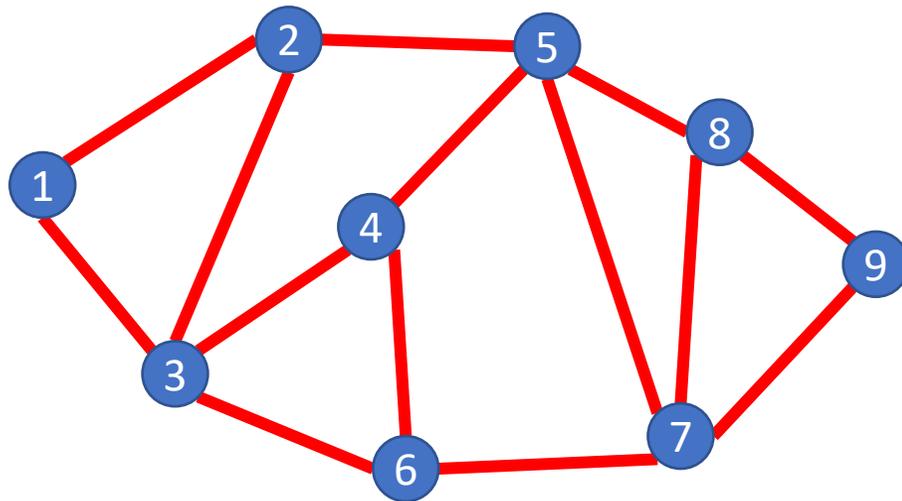
Definition: (Strongly) Connected Graph

A Graph $G = (V, E)$ s.t. for any pair of nodes $v_1, v_2 \in V$ there is a path from v_1 to v_2

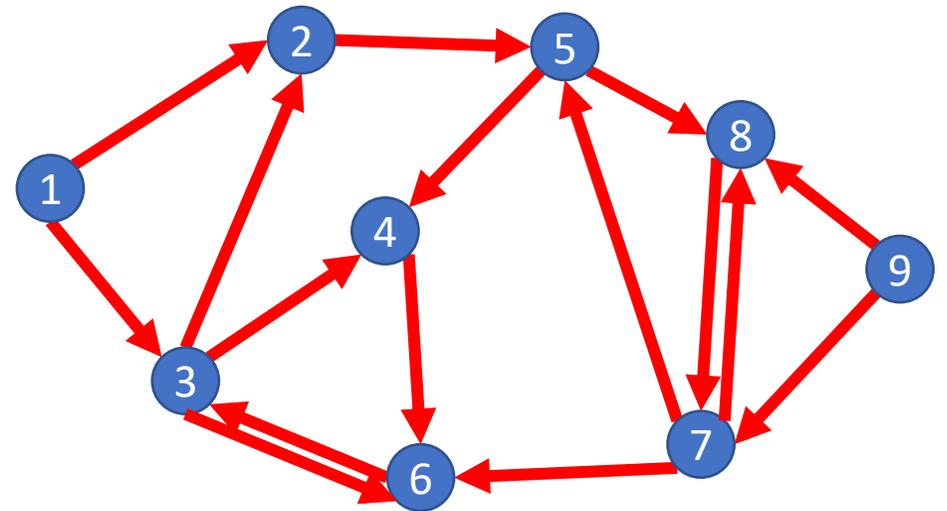


Definition: (Strongly) Connected Graph

A Graph $G = (V, E)$ s.t. for any pair of nodes $v_1, v_2 \in V$ there is a path from v_1 to v_2



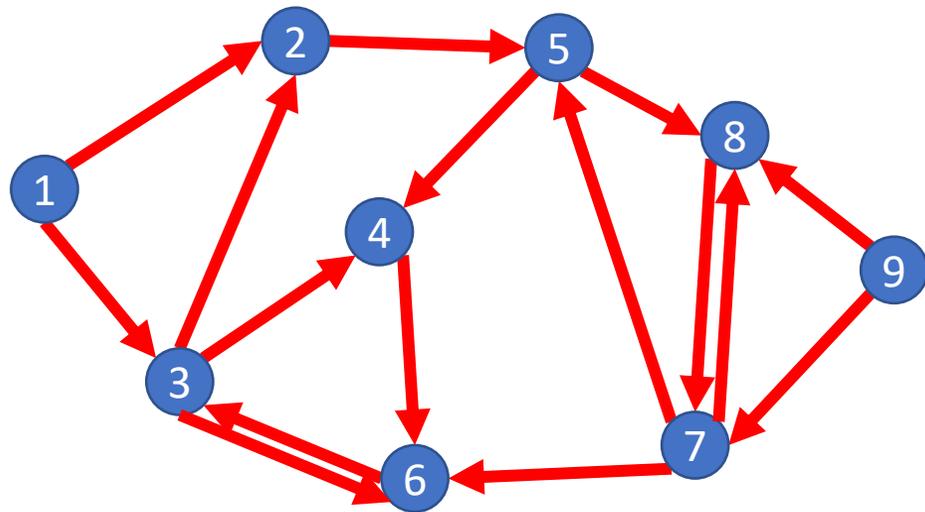
Connected



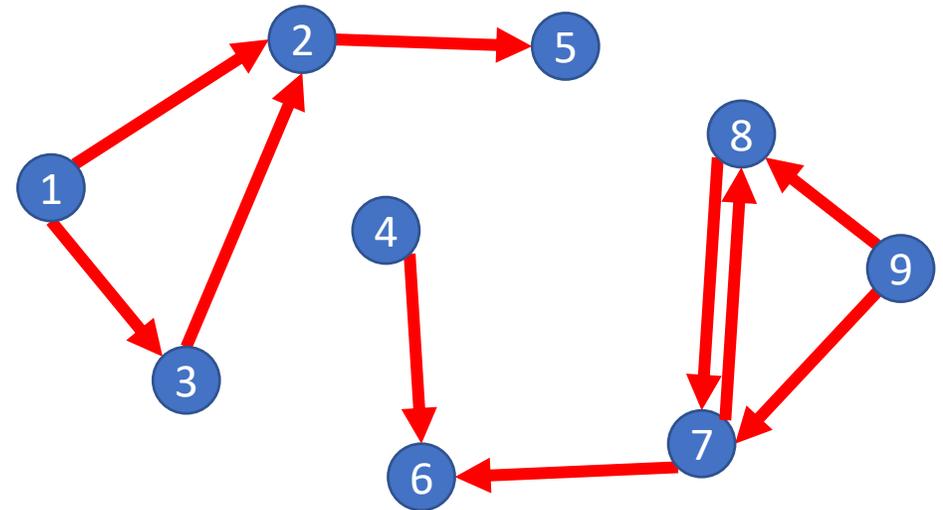
Not (strongly) Connected

Definition: Weakly Connected Graph

A Graph $G = (V, E)$ s.t. for any pair of nodes $v_1, v_2 \in V$ there is a path from v_1 to v_2 ignoring direction of edges



Weakly Connected



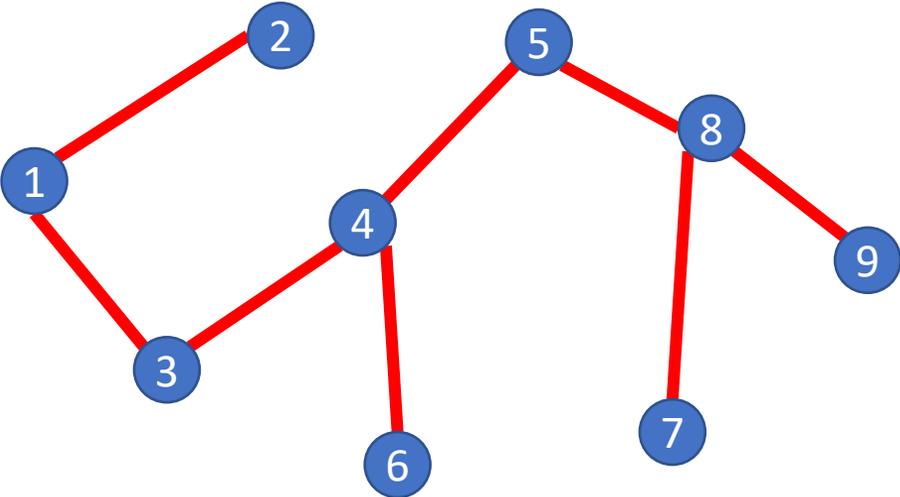
Not Weakly Connected

Graph Density, Data Structures, Efficiency

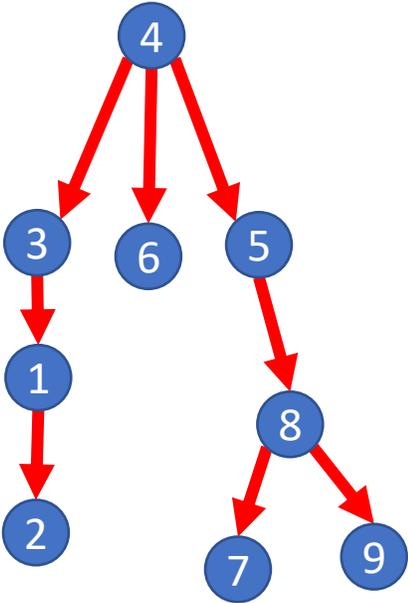
- The maximum number of edges in a graph is $\Theta(|V|^2)$:
 - Undirected and simple: $\frac{|V|(|V|-1)}{2}$
 - Directed and simple: $|V|(|V| - 1)$
 - Direct and non-simple (but no duplicates): $|V|^2$
- If the graph is connected, the minimum number of edges is $|V| - 1$
- If $|E| \in \Theta(|V|^2)$ we say the graph is **dense**
- If $|E| \in \Theta(|V|)$ we say the graph is **sparse**
- Because $|E|$ is not always near to $|V|^2$ we do not typically substitute $|V|^2$ for $|E|$ in running times, but leave it as a separate variable
 - However, $\log(|E|) \in \Theta(\log(|V|))$

Definition: Tree

A Graph $G = (V, E)$ is a tree if it is undirect, connected, and has no cycles (i.e. is acyclic). Often one node is identified as the “root”



A Tree



A Rooted Tree