Today's announcements:

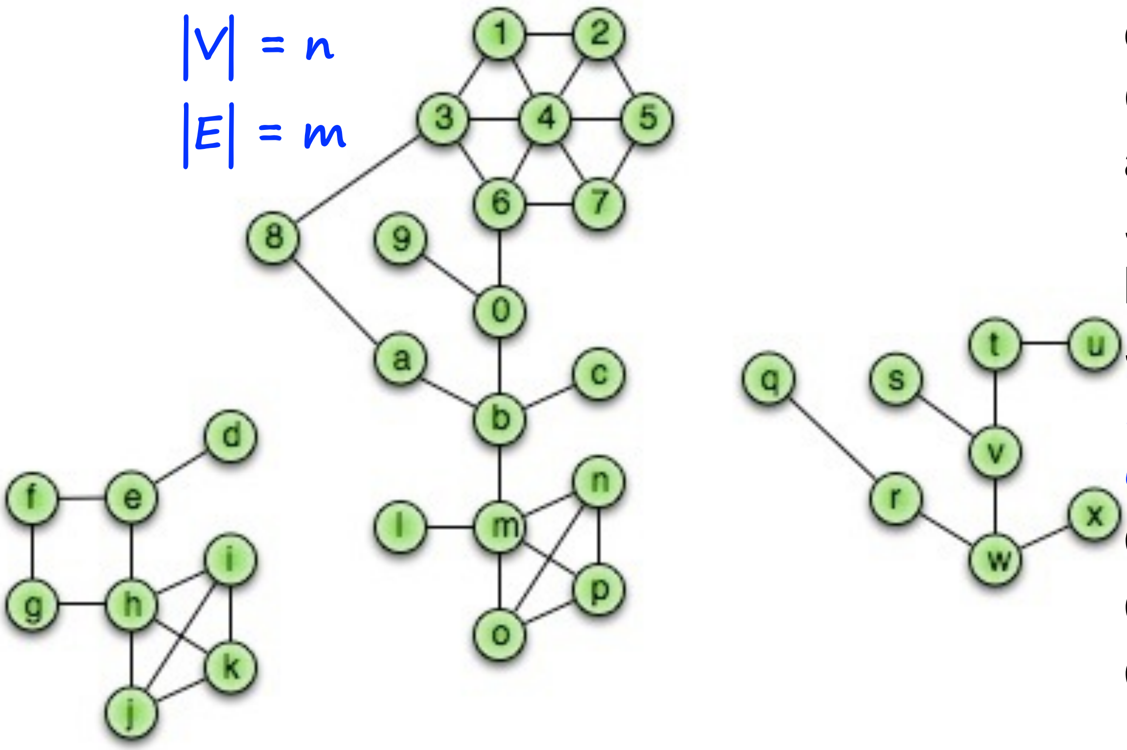
MP7 available. Due 12/8, 11:59p. EC due 12/1, 11:59p.

Graph Vocabulary:

$G = (V,E)$

$|V| = n$

$|E| = m$



G_1 G_2 G_3

Incident edges(v): I = {(x,v) in E}

Degree(v): |I|

Adjacent vertices(v): A = {x: (x,v) in E}

Path(G_2) - sequence of vertices connected by edges.

Cycle(G_1) - path with common begin and end vertex.

Simple graph(G) - graph with no self-loops and no multi-edges.

Subgraph(G) – $G' = (V', E')$, V' __ V, E' __ E, and (u,v) __ E' implies u __ V' and v __ V'.

Complete subgraph(G_2) –

Connected subgraph(G) –

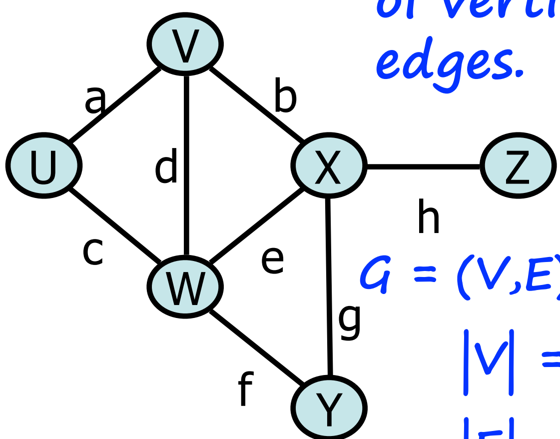
Connected component(G) –

Acyclic subgraph(G_2) –

Spanning tree(G_1) –

Graphs: theory that will help us in analysis

Running times often reported in terms of n, the number of vertices, but they often depend on m, the number of edges.



$G = (V,E)$

$|V| = n$

$|E| = m$

How many edges?

At least:

connected –

not connected -

At most:

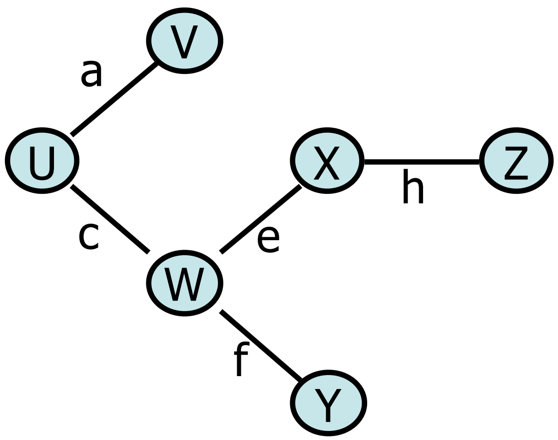
simple -

not simple -

Relationship to degree sum:

$$\sum_{v \in V} \deg(v) =$$

Thm: Every minimally connected graph $G=(V,E)$ has $|V|-1$ edges.



Proof: Consider an arbitrary minimally connected graph $G=(V,E)$.

Lemma: Every connected subgraph of G is minimally connected. (easy proof by contradiction)

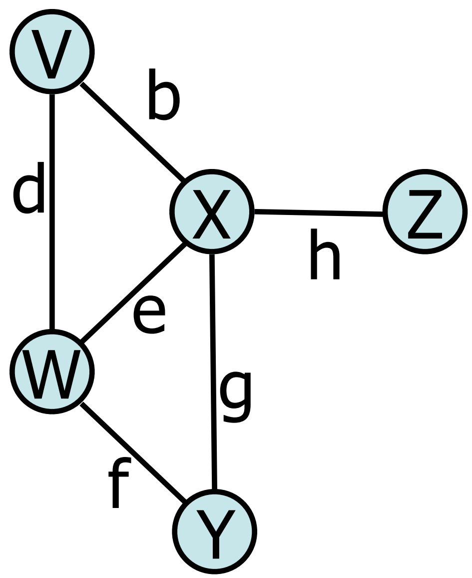
IH: For any $j < |V|$, any minimally connected graph of j vertices has j-1 edges.

Suppose $|V| = 1$: A minimally connected graph of 1 vertex has no edges, and $0 = 1-1$.

Suppose $|V| > 1$: Choose any vertex and let d denote its degree. Remove its incident edges, partitioning the graph into ________ components, C_0=(_____,_____), ... C_d=(_____, _____), each of which is a minimally connected subgraph of G. This means that $|E_k|$ = _________ by ______.

Now we'll just add up edges in the original graph:

Graphs: Toward implementation…(ADT)



Data:

Vertices

Edges

+ some structure that reflects the connectivity of the graph

Functions: (merely a smattering…)

insertVertex(pair keyData)

insertEdge(vertex v1, vertex v2, pair keyData)

removeEdge(edge e);

removeVertex(vertex v);

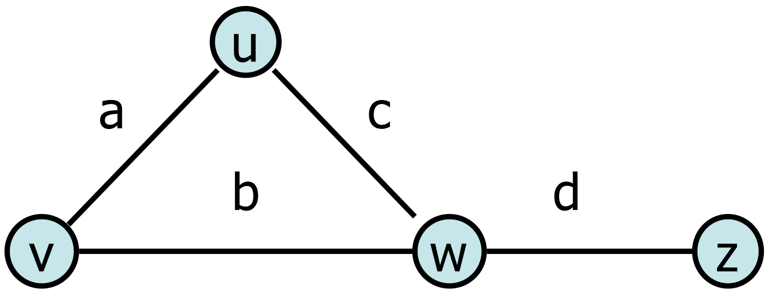
incidentEdges(vertex v);

areAdjacent(vertex v1, vertex v2);

origin(edge e);

destination(edge e);

Graphs: Edge List (a first implementation)



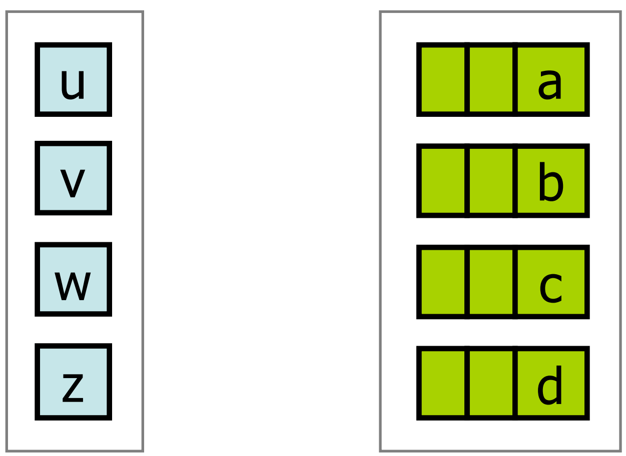
Some functions we'll compare:

insertVertex(vertex v)

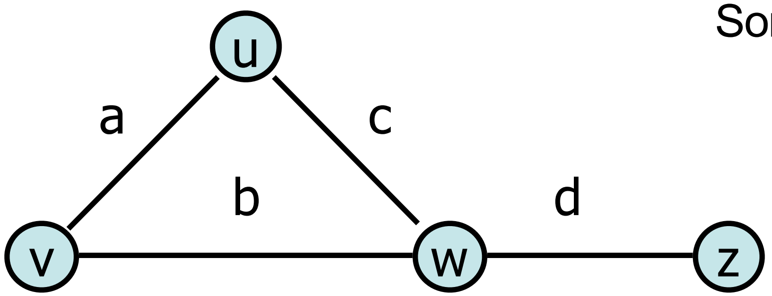
removeVertex(vertex v)

areAdjacent(vertex v, vertex u)

incidentEdges(vertex v)



Graphs: Adjacency Matrix



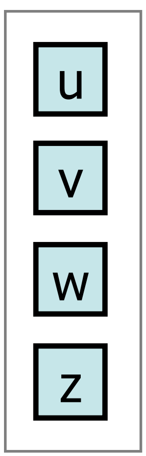
Some functions we'll compare:

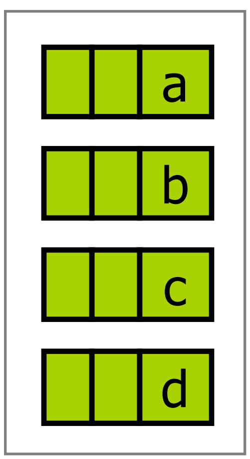
insertVertex(vertex v)

removeVertex(vertex v)

areAdjacent(vertex v, vertex u)

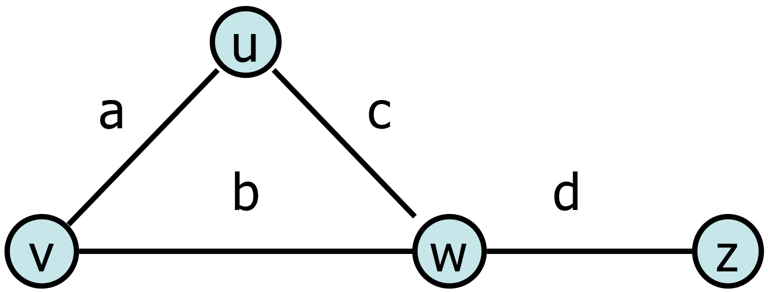
incidentEdges(vertex v)





	u	v	w	z
u				
v				
w				
z				

Graphs: Adjacency List



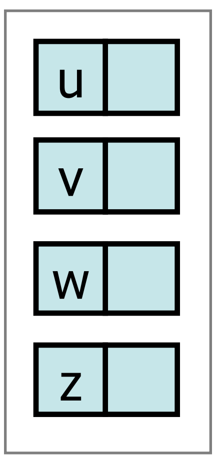
Some functions we'll compare:

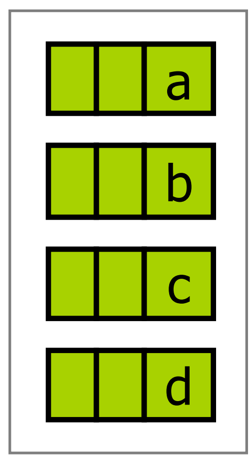
insertVertex(vertex v)

removeVertex(vertex v)

areAdjacent(vertex v, vertex u)

incidentEdges(vertex v)





Graphs: Asymptotic Performance

• n vertices, m edges • no parallel edges • no self-loops • Bounds are big-O	Edge List	Adjacency List	Adjacency Matrix
Space	$n + m$	$n + m$	n^2
incidentEdges(v)	m	$\deg(v)$	n
areAdjacent (v, w)	m	$\min(\deg(v), \deg(w))$	1
insertVertex(o)	1	1	n^2
insertEdge(v, w, o)	1	1	1
removeVertex(v)	m	$\deg(v)$	n^2
removeEdge(e)	1	1	1