CS 225

Data Structures

March 28 – Graphs

G Carl Evans

Disjoint Sets Analysis

The **iterated log** function:

The number of times you can take a log of a number.

$$\log^*(n) = \begin{cases} 0 & , n \le 1 \\ 1 + \log^*(\log(n)) & , n > 1 \end{cases}$$

What is $\mathbf{lg^*(2^{65536})}$?

Disjoint Sets Analysis

In an Disjoint Sets implemented with smart **unions** and path compression on **find**:

Any sequence of **m union** and **find** operations result in the worse case running time of O(____________),

where **n** is the number of items in the Disjoint Sets.

In Review: Data Structures

Array
- Sorted Array
- Unsorted Array
  - Stacks
  - Queues
  - Hashing
  - Heaps
    - Priority Queues
  - UpTrees
    - Disjoint Sets

Linked
- Doubly Linked List
- Trees
  - BTree
  - Binary Tree
    - Huffman Encoding
    - kd-Tree
    - AVL Tree

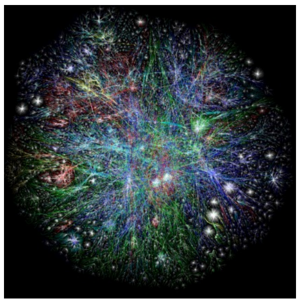
In Review: Data Structures

Array
- Sorted Array
- Unsorted Array
  - Stacks
  - Queues
  - Hashing
  - Heaps
    - Priority Queues
  - UpTrees
    - Disjoint Sets

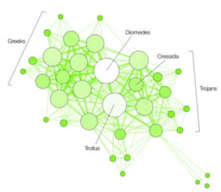
Graphs

Linked
- Doubly Linked List
- Skip List
- Trees
  - BTree
  - Binary Tree
    - Huffman Encoding
    - kd-Tree
    - AVL Tree

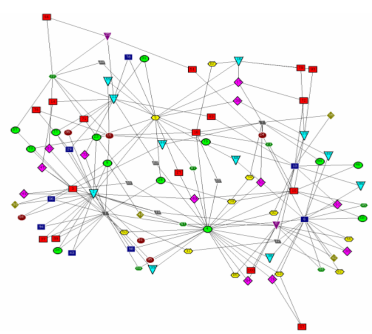
Graphs



HAMLET

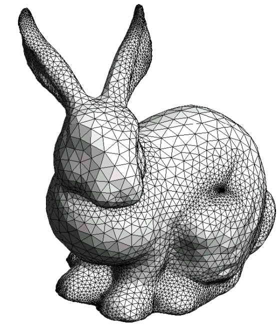


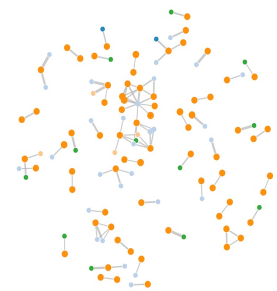
TROILUS AND CRESSIDA

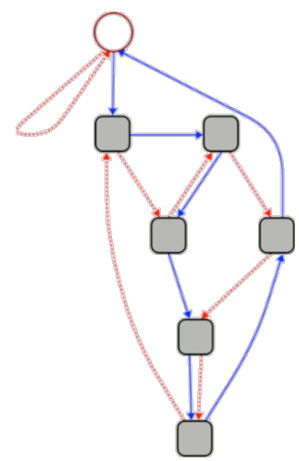


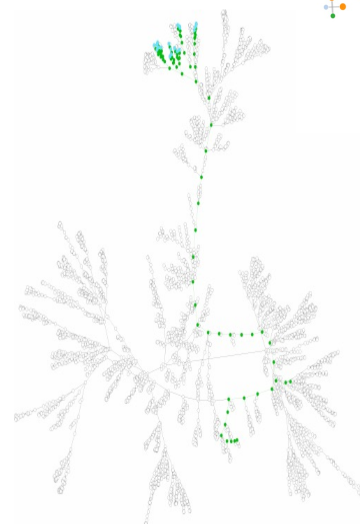
To study all of these structures:

1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms



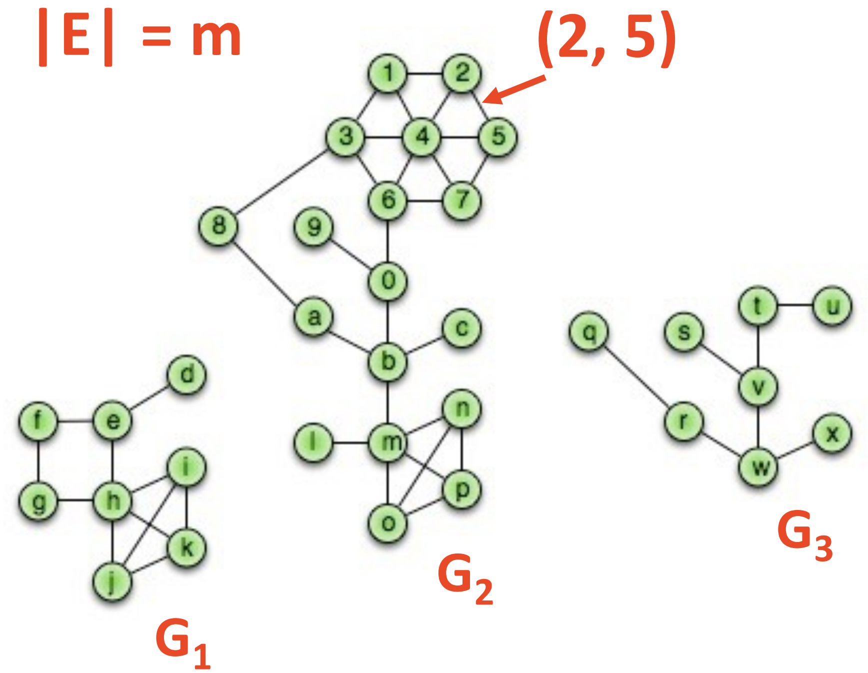






Graph Vocabulary

G = (V, E)
|V| = n
|E| = m



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

Degree(v): |I|

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

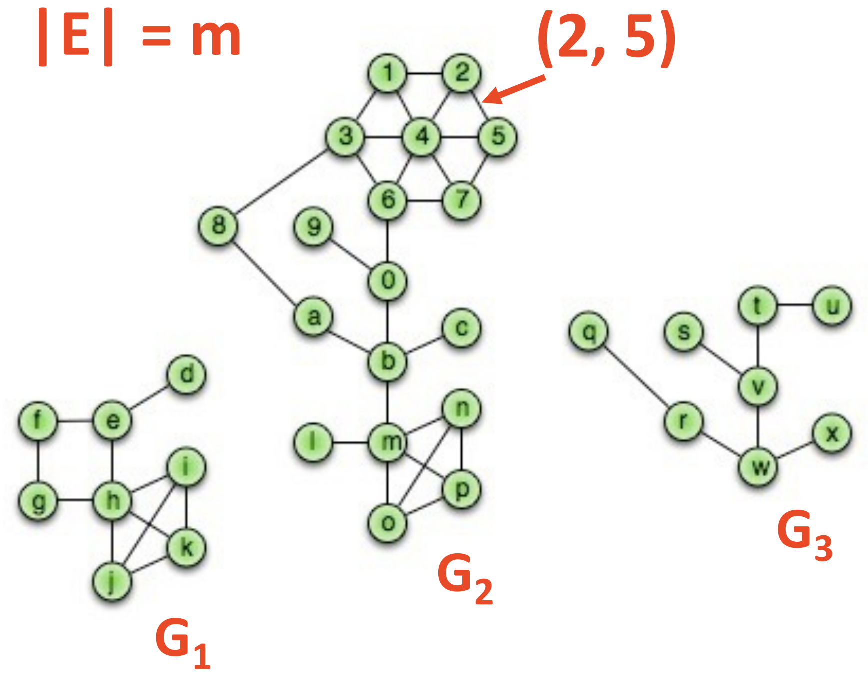
Path(G_2): Sequence of vertices connected by edges

Cycle(G_1): Path with a common begin and end vertex with at least 3 vertices.

Simple Graph(G): A graph with no self loops or multi-edges.

Graph Vocabulary

$G = (V, E)$
$|V| = n$
$|E| = m$



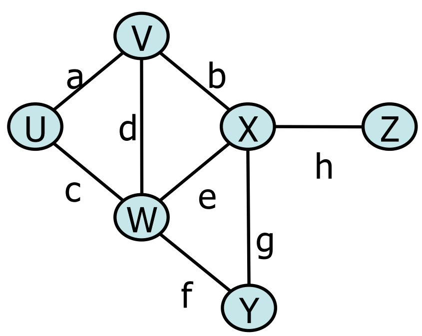
Subgraph(G):
$G' = (V', E')$:
$V' \in V$, $E' \in E$, and
$(u, v) \in E' \rightarrow u \in V', v \in V'$

Complete subgraph(G)
Connected subgraph(G)
Connected component(G)
Acyclic subgraph(G)
Spanning tree(G)

Running times are often reported by **n**, the number of vertices, but often depend on **m**, the number of edges.

How many edges? **Minimum edges:**

Not Connected:



Connected*:

Maximum edges:

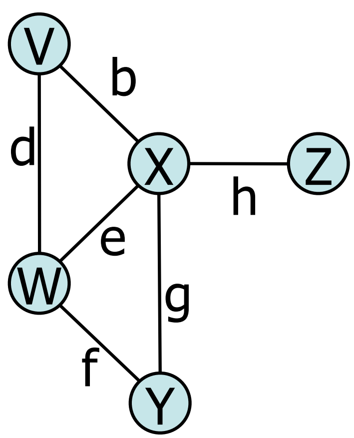
Simple:

Not simple:

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

Graph ADT

Data:
- Vertices
- Edges
- Some data structure maintaining the structure between vertices and edges.



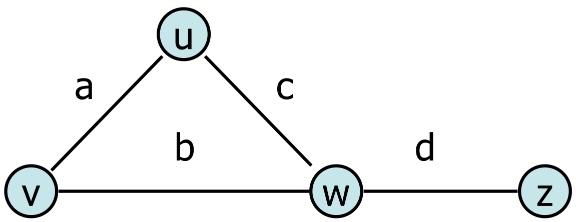
Functions:
- insertVertex(K key);
- insertEdge(Vertex v1, Vertex v2, K key);

- removeVertex(Vertex v);
- removeEdge(Vertex v1, Vertex v2);

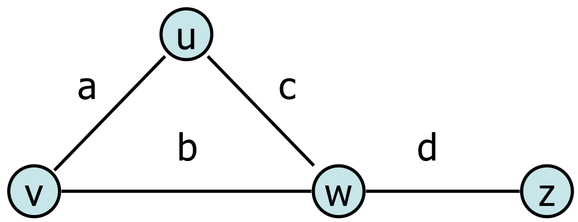
- incidentEdges(Vertex v);
- areAdjacent(Vertex v1, Vertex v2);

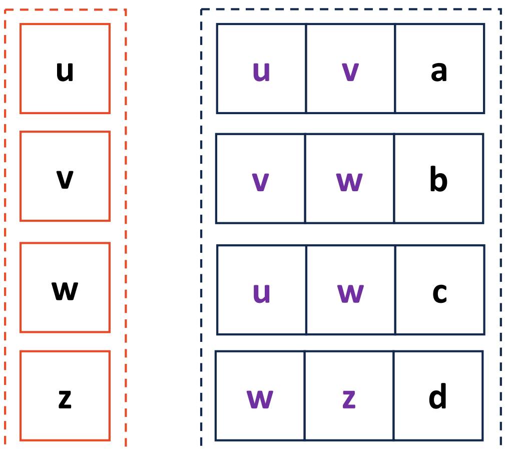
- origin(Edge e);
- destination(Edge e);

Graph Implementation Idea



Graph Implementation: Edge List

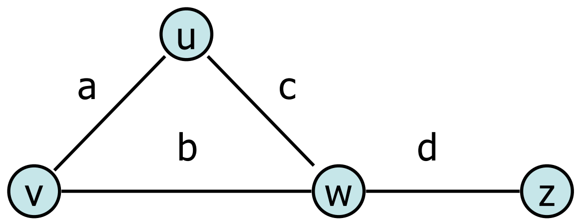




Vertex Collection:

Edge Collection:

Graph Implementation: Edge List



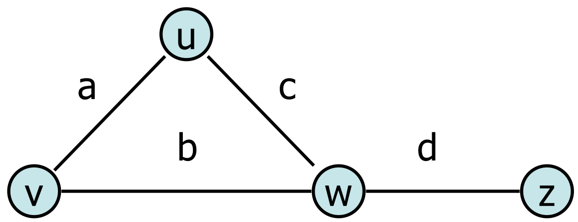
u
v
w
z

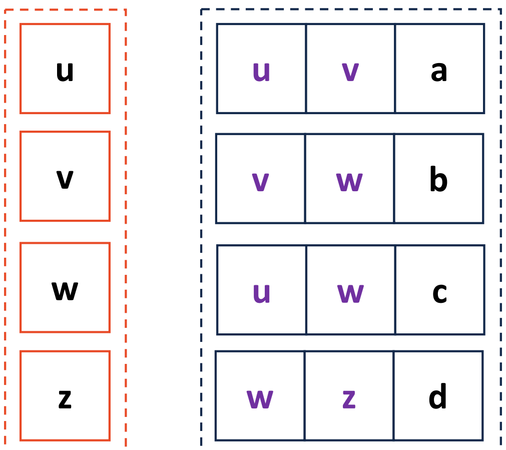
u	v	a
v	w	b
u	w	c
w	z	d

insertVertex(K key):

removeVertex(Vertex v):

Graph Implementation: Edge List



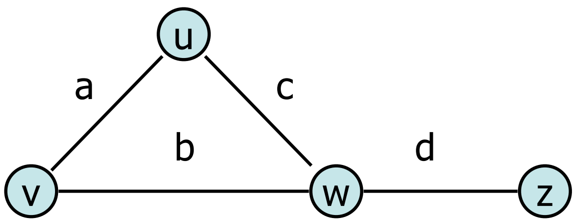


incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

```
G.incidentEdges(v1).contains(v2)
```

Graph Implementation: Edge List

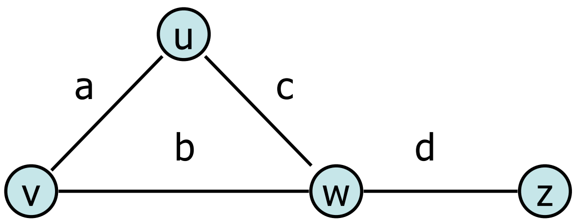


u
v
w
z

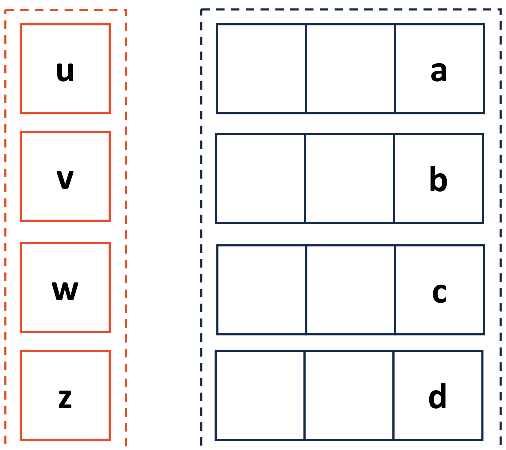
u	v	a
v	w	b
u	w	c
w	z	d

insertEdge(Vertex v1, Vertex v2, K key):

Graph Implementation: Adjacency Matrix



insertVertex(K key);
removeVertex(Vertex v);
areAdjacent(Vertex v1, Vertex v2);
incidentEdges(Vertex v);



	u	v	w	z
u				
v				
w				
z				