#26: Graph Vocabulary + Implementation

April 12, 2021 · *G Carl Evans*

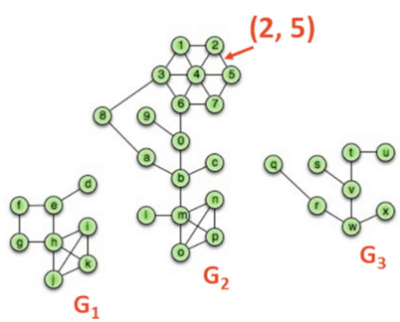
A Review of Major Data Structures So Far

Array-based	List/Pointer-based
- Sorted Array	- Singly Linked List
- Unsorted Array	- Doubly Linked List
- Stacks	- Skip Lists
- Queues	- Trees
- Hashing	- BTree
- Heaps	- Binary Tree
- Priority Queues	- Huffman Encoding
- UpTrees	- kd-Tree
- Disjoint Sets	- AVL Tree

Motivation:
Graphs are awesome data structures that allow us to represent an enormous range of problems. To study these problems, we need:

1. A common vocabulary to talk about graphs
2. Implementation(s) of a graph
3. Traversals on graphs
4. Algorithms on graphs

Graph Vocabulary
Consider a graph **G** with vertices **V** and edges **E**, **G=(V,E)**.



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.

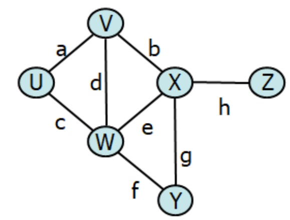
Simple Graph(G): A graph with no self loops or multi-edges.

Subgraph(G): **G' = (V', E')**:
V' ∈ V, E' ∈ E, and (u, v) ∈ E → u ∈ V', v ∈ V'

Graphs that we will study this semester include:
Complete subgraph(G)
Connected subgraph(G)
Connected component(G)
Acyclic subgraph(G)
Spanning tree(G)

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

For arbitrary graphs, the **minimum** number of edges given a graph that is:



Not Connected:

Minimally Connected:*

The **maximum** number of edges given a graph that is:

Simple:

Not Simple:

The relationship between the degree of the graph and the edges:

Proving the Size of a Minimally Connected Graph

Theorem: Every connected graph **G=(V, E)** has at least **|V|-1** edges.

Proof of Theorem
Consider an arbitrary, connected graph **G=(V, E)**.

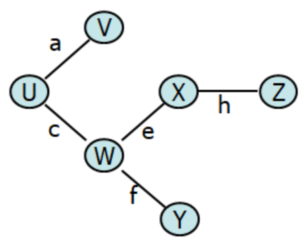
Suppose |V| = 1:
Definition:

Theorem:

Inductive Hypothesis: For any **j** < **|V|**, any connected graph of **j** vertices has at lest **j-1** edges.

Suppose |V| > 1:
1. Choose any vertex:
 -

 -



2. Partitions:
 -

 -

 - **C_0 :=**

 - **C_k, k=[1…d] :=**

3. Count the edges:

$|E_G|$ =

*…by application of our IH and Lemma #1, every component **C_k** is a minimally connected subgraph of **G**…*

$|E_G|$ =

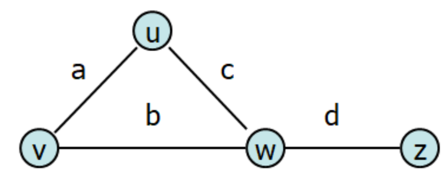
Graph ADT

Data	Functions

1. Vertices
2. Edges
3. Some data structure maintaining the structure between vertices and edges.
	`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 #1: Edge List

Vert.	Edges		
u			**a**
v			**b**
w			**c**
z			**d**



Operations:
insertVertex(K key):

removeVertex(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

incidentEdges(Vertex v):

CS 225 – Things To Be Doing:
1. mp_traversal due today. **2.** Daily POTDs are ongoing!