#33: Graph Vocabulary + Implementation
November 3, 2021 · *G Carl Evans*

A Review of Major Data Structures so Far

Array-based	List/Pointer-based

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

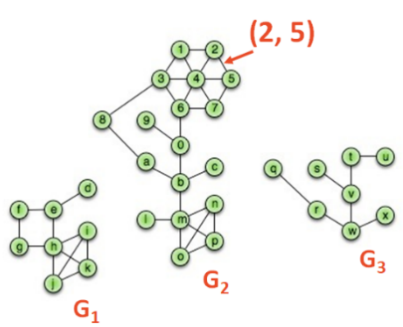
Motivation:

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 containing at least three vertices.

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

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

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

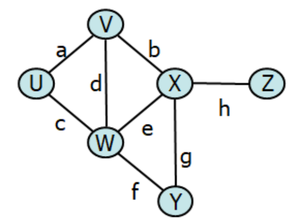
Size and Running Times

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:

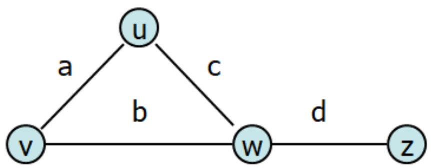
Graph ADT

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**



Data Structures:

Vertex Collection:

Edge Collection:

Operations on an Edge List implementation:

insertVertex(K key):
- What needs to be done?

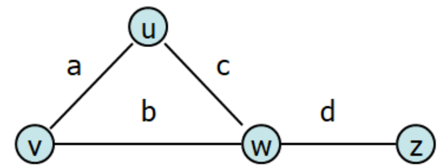
removeVertex(Vertex v):
- What needs to be done?

incidentEdges(Vertex v):
- What needs to be done?

areAdjacent(Vertex v1, Vertex v2):
- Can this be faster than `G.incidentEdges(v1).contains(v2)`?

insertEdge(Vertex v1, Vertex v2, K key):
- What needs to be done?

Graph Implementation #2: Adjacency Matrix



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

CS 225 – Things To Be Doing:

1. mp_mazes out now EC due Monday Nov. 8
2. Final Project teams being contacted now.
3. Daily POTDs are ongoing!