202: Computer Science II

Northern Virginia Community College

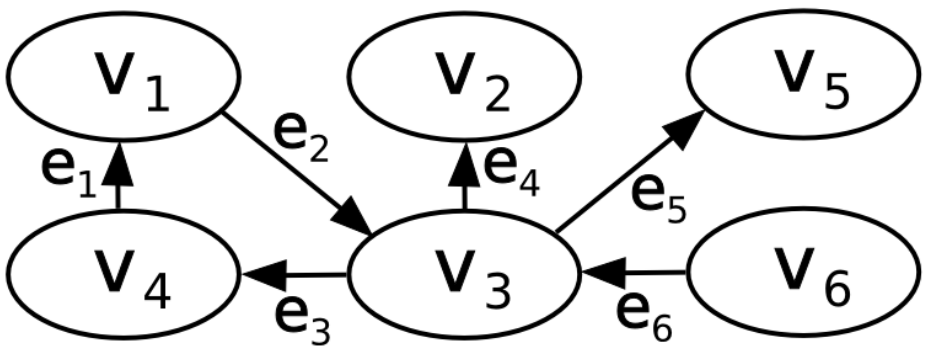
Graphs: Implementations

Cody Narber, M.S.
November 18, 2017

Data Structures - Graph

A **Graph** is a data structure that consists of several vertices and several edges that connect vertices. Vertices are reminiscent of the nodes we have used in the past and the edges like that of references between nodes. Typically graphs are denoted as $G = (V, E)$ where V is the set of n vertices $\{v_1, v_2, ...v_{n-1}, v_n\}$ and E is the set of m edges $\{e_1, e_2, .., e_{m-1}, e_m\}$. Each e_x is defined as a pair of vertices (v_i, v_j).

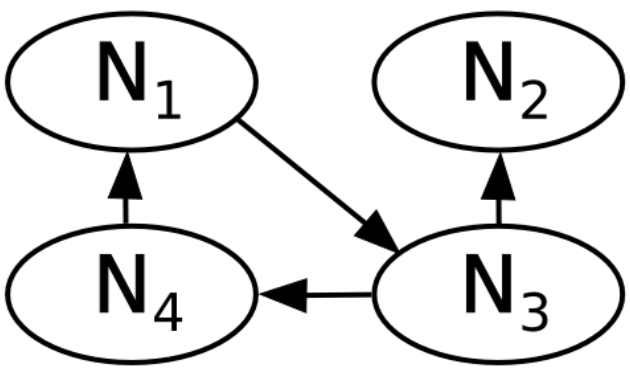
- **Vertex** - can contain data and may have edges coming in or going out.
- **Edge** - can be weighted (i.e. have a value) and connects two vertices (which are considered **adjacent**)



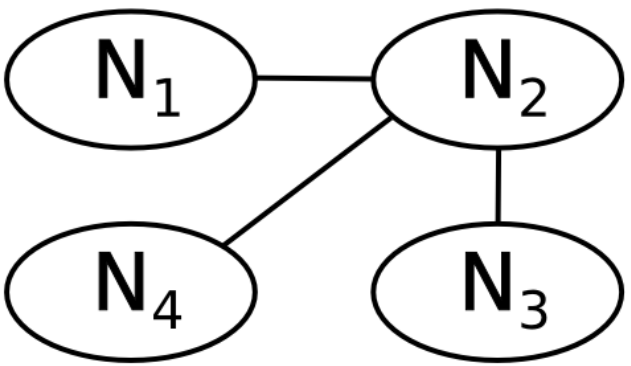
Graph Terminology

There are several ways to describe types/properties of graphs (we have already seen what qualifies as a tree).

- **Directed** - The edge pair (v_i, v_j) denotes a single direction from v_i to v_j, but not the other way around.
- **Undirected** - The edge pair (v_i, v_j) denotes that v_i and v_j can go back and forth. Equivalent to two directed edges.
- **Cyclic** - There exists a **cycle** in the graph. *A cycle is defined as a path from a node back to itself, where no edges are used twice.*
- **Acyclic** - There are no cycles in the graph.



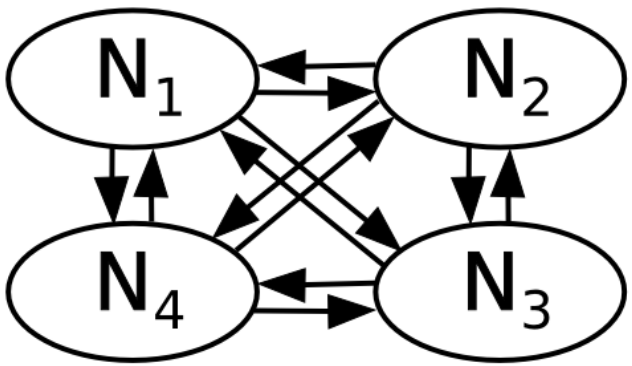
Directed - Cyclic



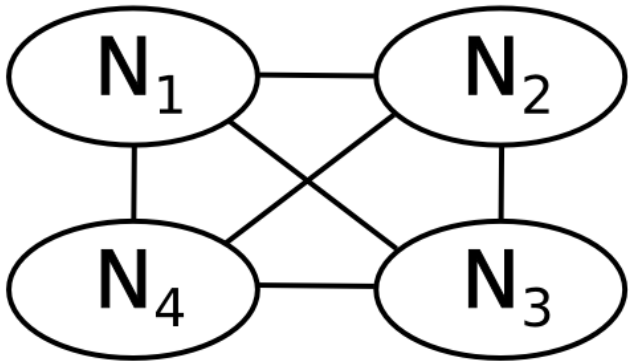
Undirected - Acyclic

Graph Terminology

A complete graph is one such that every vertext is adjacent to every other vertex. **Adjacent** vertices are those that are directly connected via an edge (i.e. have a path of length 1 to the other node).



Directed - Complete

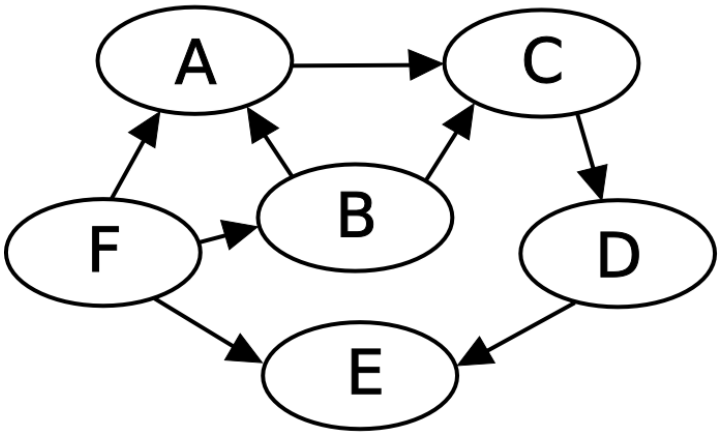


Undirected - Complete

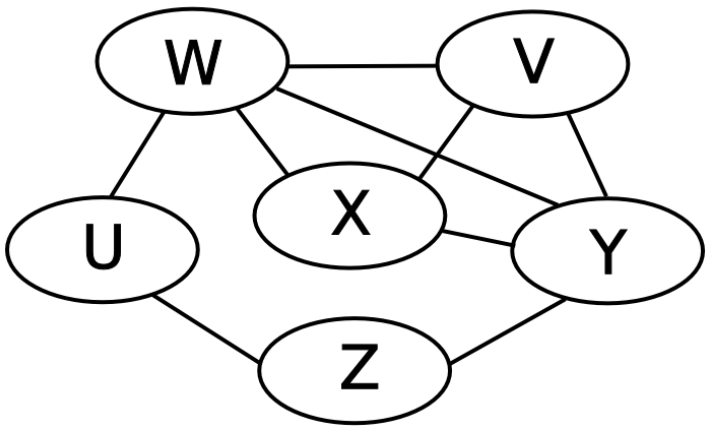
We usually denote the sizes of the vertex and edge sets with V and E respectively, and their sizes as $|V|$ and $|E|$ which is known as set **cardinality**. When discussing runtime of graph algorithms they could be thought of as $O(|V|log|E|)$, but typically cardinality is assumed when discussing runtime, therefore we leave off the cardinality notation and would say something like: $O(V * log(E))$

Graph Examples

- Which Graph is undirected (1,2,both)?
- Which Graph is cyclic (1,2,both)?
- Does there exist a path from A to F (Y or N)?
- Is there a subgraph($|V| > 2$) that is complete (Y or N)?
- What is the cardinality of E in Graph 1?
- Does there exist a path between each pair of vertices in Graph 1 (how about Graph 2)?



Graph 1



Graph 2

Graph Notation

A graph consists of a set of vertices. We can store these in a variety of ways (though we can think about it as an array-list). The graph also needs a way to reference the edges and link them to the vertices. There are two ways this is typically done:

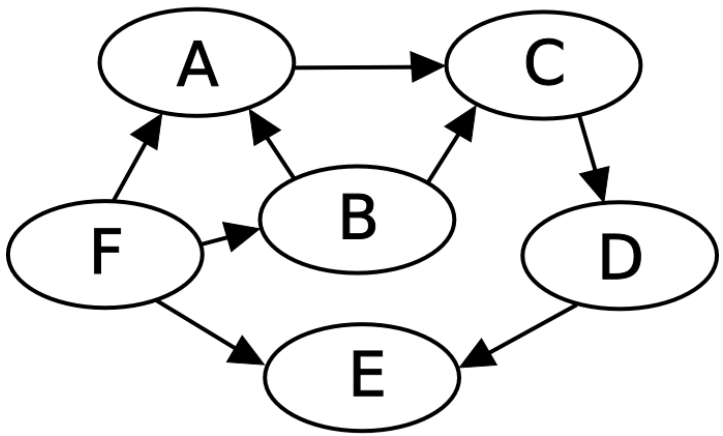
- **Adjacency List** - where each Vertex object houses a list of edges
- **Adjacency Matrix** - where the graph data structure contains a 2D Array with vertices represented on the rows and columns

Graph sets can also be written out by hand using set notation:

$G = (V, E)$

$V = \{A, B, C, D, E, F\}$

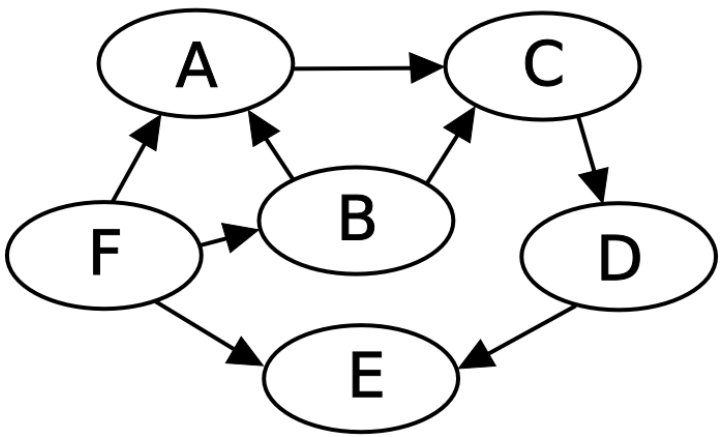
$E = \{(A, C), (B, A), (B, C), (C, D), (D, E), (F, A), (F, B), (F, E)\}$



Graph Implementation 1

Adjacency List:

We can implement the graph data structure as a array-based list of vertices. Each Vertex will be another data structure that contains a list of edges, with indices to the connect vertex locations in the vertex list.

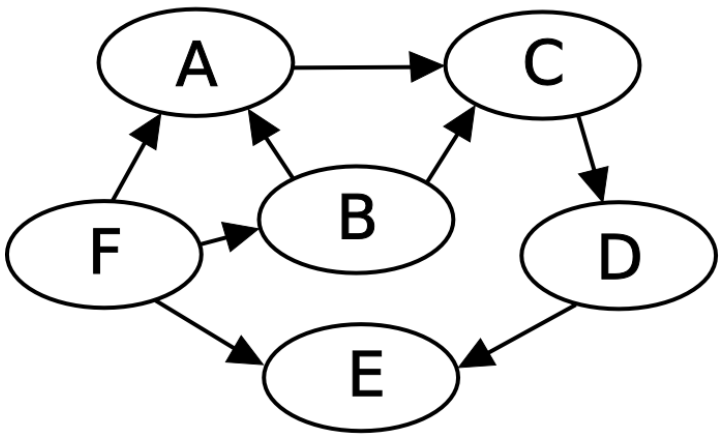


Index	Vertex	
0	A	2
1	B	0 → 2
2	C	3
3	D	4
4	E	
5	F	0 → 1 → 4
...	...	

Graph Implementation 2

Adjacency Matrix:

We can implement the graph data structure as a array-based list of vertices as before, but the edges are also stored on the graph, this time there will be a 2D array created of size $|V|x|V|$, where it is initialized to all zeros (no edges) and everytime there is an edge between vertices a 1 will be store to indicate there is an edge from the row vertex to the column vertex.



Index	Vertex
0	A
1	B
2	C
3	D
4	E
5	F
...	...

0	0	1	0	0	0
1	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	0
1	1	0	0	1	0

Graph Implementation

What is the benefit of two different implementation?

Adjacency List

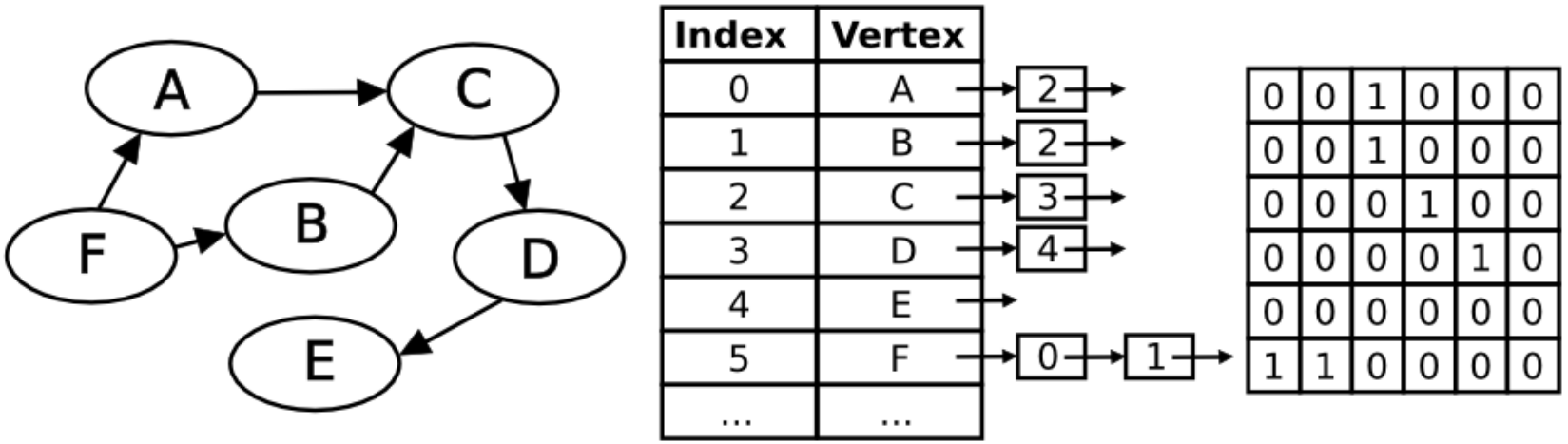
- Memory Efficient
- Easy/Cheap to add edges
- Could take a long time to iterate through the graph
- Typically used when $|E| <<< |V|^2$, a.k.a. **sparse** graph

Adjacency Matrix

- Very fast look up and iteration
- Costly to add edges (grow matrix)
- Takes up a lot of memory especially for graphs containing a lot of vertices.
- Typically used when $|E| \approx |V|^2$, a.k.a. **dense** graph

Graph Implementation

Sparse Graph:



Dense Graph: