Graph

Chapter 9 of textbook 1

Outline

A graph is an abstract data type for storing adjacency relations

- We start with definitions:
  - Vertices, edges, degree and sub-graphs
- We will describe paths in graphs
  - Simple paths and cycles
- Definition of connectedness
- Weighted graphs
- We will then reinterpret these in terms of directed graphs
- Directed acyclic graphs

Outline

We will define an Undirected Graph ADT as a collection of *vertices*

$$V = \{v_1, v_2, ..., v_n\}$$

– The number of vertices is denoted by

$$|V| = n$$

– Associated with this is a collection E of unordered pairs $\{v_i, v_j\}$ termed *edges* which connect the vertices

There are a number of data structures that can be used to implement abstract undirected graphs

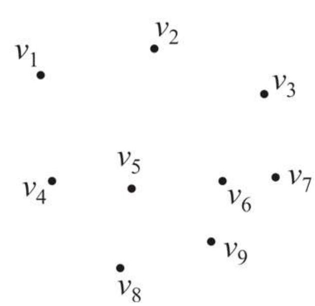
– Adjacency matrices
– Adjacency lists

Graphs

Consider this collection of vertices

$$V = \{v_1, v_2, ..., v_9\}$$

where $|V| = 9$

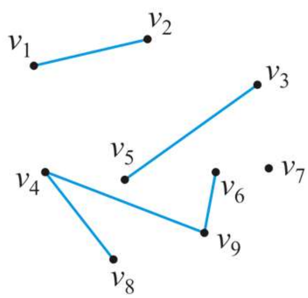


Undirected graphs

Associated with these vertices are $|E| = 5$ edges

$$E = \{\{v_1, v_2\}, \{v_3, v_5\}, \{v_4, v_8\}, \{v_4, v_9\}, \{v_6, v_9\}\}$$

– The pair $\{v_j , v_k\}$ indicates that both vertex v_j is adjacent to vertex v_k and vertex v_k is adjacent to vertex v_j



Undirected graphs

We will assume in this course that a vertex is never adjacent to itself

– For example, $\{v_1, v_1\}$ will not define an edge

The maximum number of edges in an undirected graph is

$$|E| \le \binom{|V|}{2} = \frac{|V|(|V|-1)}{2} = \mathrm{O}\left(|V|^2\right)$$

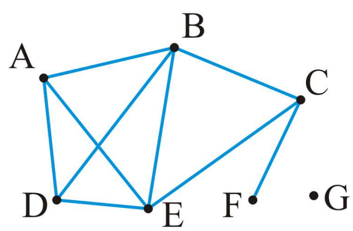
An undirected graph

Example: given the $|V| = 7$ vertices

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

and the $|E| = 9$ edges

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



Degree

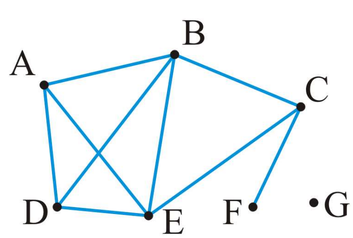
The degree of a vertex is defined as the number of adjacent vertices

degree(A) = degree(D) = degree(C) = 3

degree(B) = degree(E) = 4

degree(F) = 1

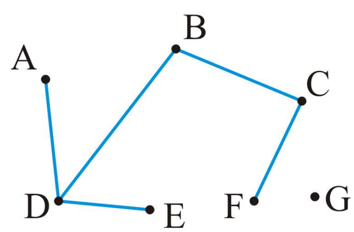
degree(G) = 0

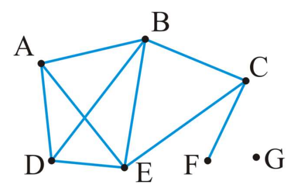


Those vertices adjacent to a given vertex are its *neighbors*

Sub-graphs

A *sub-graph* of a graph is a subset of the vertices and a subset of the edges that connected the subset of vertices in the original graph





Paths

A path in an undirected graph is an ordered sequence of vertices

$(v_0, v_1, v_2, ..., v_k)$

where $\{v_{j-1}, v_j\}$ is an edge for $j = 1, ..., k$

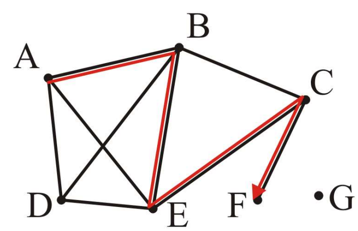
– Termed *a path from* v_0 to v_k

– The length of this path is k

Paths

A path of length 4:

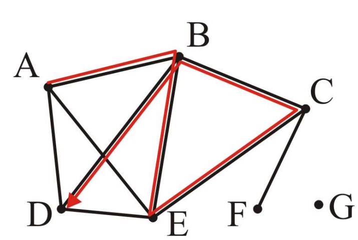
(A, B, E, C, F)



Paths

A path of length 5:

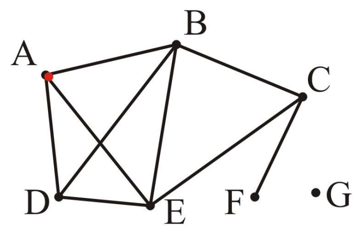
(A, B, E, C, B, D)



Paths

A *trivial* path of length 0:

(A)



Simple paths

A *simple path* has no repetitions other than perhaps the first and last vertices

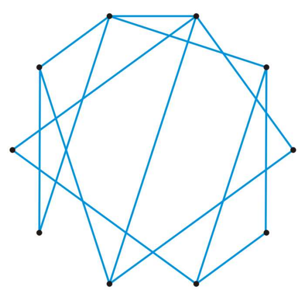
A *simple cycle* is a simple path of at least two vertices with the first and last vertices equal

– Note: these definitions are not universal

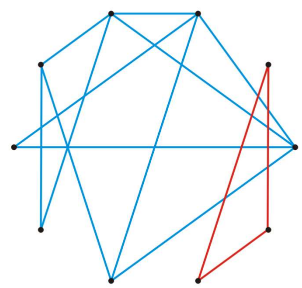
Connectedness

Two vertices v_i, v_j are said to be *connected* if there exists a path from v_i to v_j

A graph is connected if there exists a path between any two vertices



A connected graph



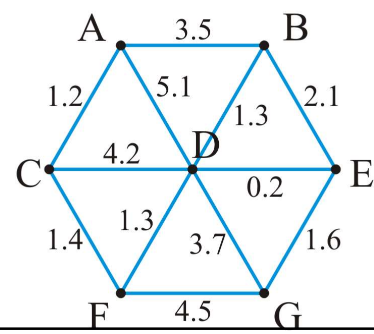
An unconnected graph

Weighted graphs

A weight may be associated with each edge in a graph

- This could represent distance, energy consumption, cost, etc.
- Such a graph is called a *weighted graph*

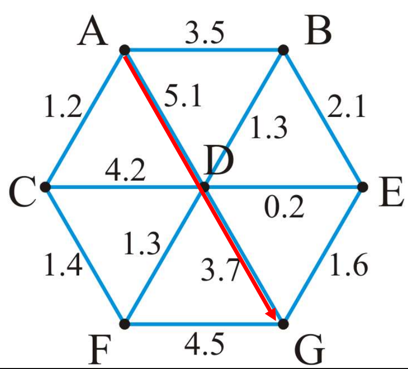
Pictorially, we will represent weights by numbers next to the edges



Weighted graphs

The *length* of a path within a weighted graph is the sum of all of the edges which make up the path

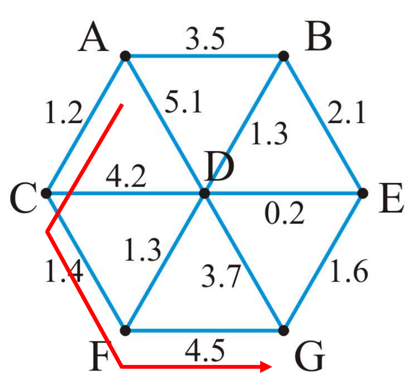
– The length of the path (A, D, G) in the following graph is 5.1 + 3.7 = 8.8



Weighted graphs

Different paths may have different weights

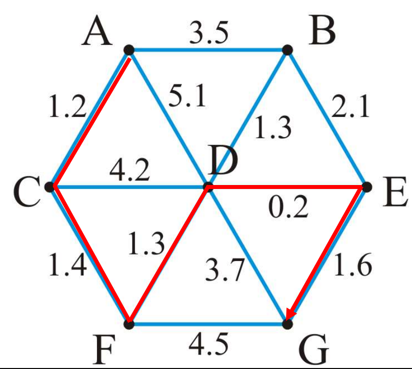
– Another path is (A, C, F, G) with length 1.2 + 1.4 + 4.5 = 7.1



Weighted graphs

Problem: find the shortest path between two vertices

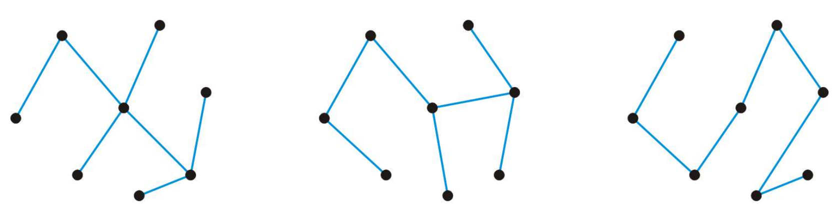
– Here, the shortest path from A to G is (A, C, F, D, E, G) with length 5.7



Trees

A graph is a tree if it is connected and there is a unique path between any two vertices

– Three trees on the same eight vertices



Consequences:

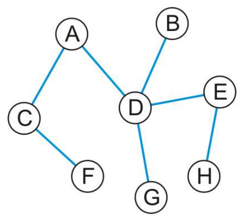
– The number of edges is $|E| = |V| - 1$
– The graph is *acyclic*, that is, it does not contain any cycles
– Adding one more edge must create a cycle
– Removing any one edge creates two disjoint non-empty sub-graphs

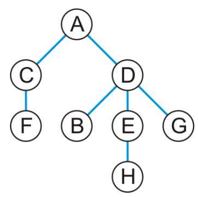
Trees

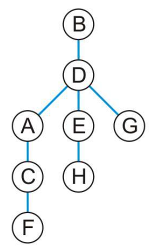
Any tree can be converted into a rooted tree by:
- Choosing any vertex to be the root
- Defining its neighboring vertices as its children

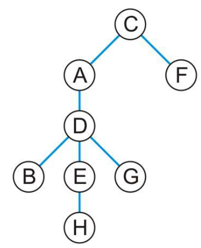
and then recursively defining:
- All neighboring vertices other than that one designated its parent are now defined to be that vertices children

Given this tree, here are three rooted trees associated with it







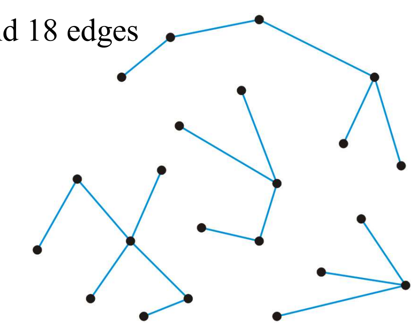


Forests

A forest is any graph that has no cycles

Consequences:
- The number of edges is $|E| < |V|$
- The number of trees is $|V| - |E|$
- Removing any one edge adds one more tree to the forest

Here is a forest with 22 vertices and 18 edges
- There are four trees



Directed graphs

In a *directed graph*, the edges on a graph are be associated with a direction

- Edges are ordered pairs (v_j, v_k) denoting a connection from v_j to v_k
- The edge (v_j, v_k) is different from the edge (v_k, v_j)

Streets are directed graphs:

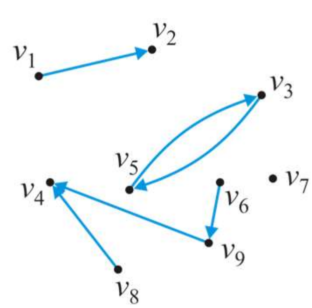
- In most cases, you can go two ways unless it is a one-way street

Directed graphs

Given our graph of nine vertices $V = \{v_1, v_2, \ldots v_9\}$

- These six pairs (v_j , v_k) are *directed edges*

$E = \{(v_1, v_2), (v_3, v_5), (v_5, v_3), (v_6, v_9), (v_8, v_4), (v_9, v_4)\}$



Directed graphs

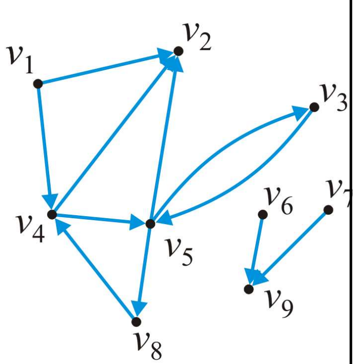
The maximum number of directed edges in a directed graph is

$$|E| \le 2\binom{|V|}{2} = 2\frac{|V|\left(|V|-1\right)}{2} = |V|\left(|V|-1\right) = \mathrm{O}\left(|V|^2\right)$$

In and out degrees

The degree of a vertex must be modified to consider both cases:

- The *out-degree* of a vertex is the number of vertices which are adjacent to the given vertex
- The *in-degree* of a vertex is the number of vertices which this vertex is adjacent to



In this graph:

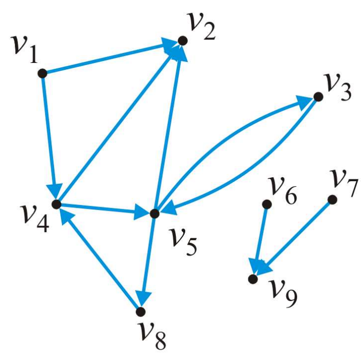
in_degree(v_1) = 0 out_degree(v_1) = 2

in_degree(v_5) = 2 out_degree(v_5) = 3

Sources and sinks

Some definitions:
- Vertices with an in-degree of zero are described as *sources*
- Vertices with an out-degree of zero are described as *sinks*

In this graph:
- Sources: v_1, v_6, v_7
- Sinks: v_2, v_9



Paths

A path in a directed graph is an ordered sequence of vertices

$(v_0, v_1, v_2, ..., v_k)$

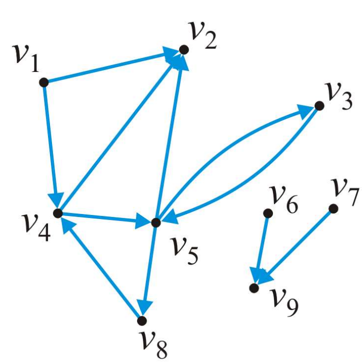
where (v_{j-1}, v_j) is an edge for $j = 1, ..., k$

A path of length 5 in this graph is

$(v_1, v_4, v_5, v_3, v_5, v_2)$

A simple cycle of length 3 is

(v_8, v_4, v_5, v_8)



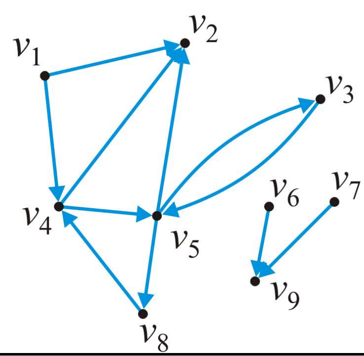
Connectedness

Two vertices v_j, v_k are said to be *connected* if there exists a path from v_j to v_k

- A graph is *strongly connected* if there exists a directed path between any two vertices
- A graph is *weakly connected* if there exists a path between any two vertices that ignores the direction

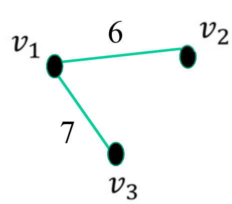
In this graph:

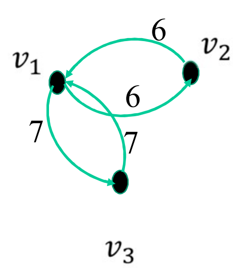
- The sub-graph $\{v_3, v_4, v_5, v_8\}$ is strongly connected
- The sub-graph $\{v_1, v_2, v_3, v_4, v_5, v_8\}$ is weakly connected



Weighted undirected graphs

- Given a weighted undirected graph. In some cases, we can consider the graph as weighted directed graph

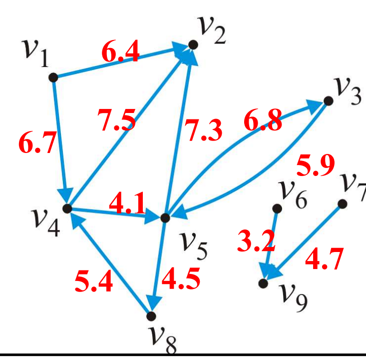




Weighted directed graphs

In a weighted directed graphs, each edge is associated with a value

Unlike weighted undirected graphs, if both (v_j, v_k) and (v_j, v_k) are edges, it is not required that they have the same weight

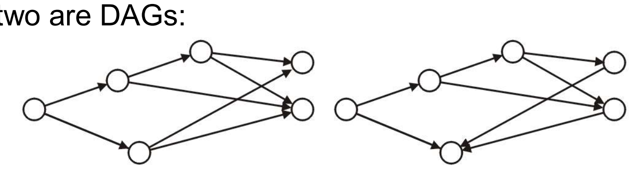


Directed acyclic graphs

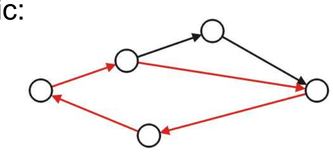
A *directed acyclic graph* is a directed graph which has no cycles

- These are commonly referred to as DAGs
- They are graphical representations of partial orders on a finite number of elements

These two are DAGs:



This directed graph is not acyclic:



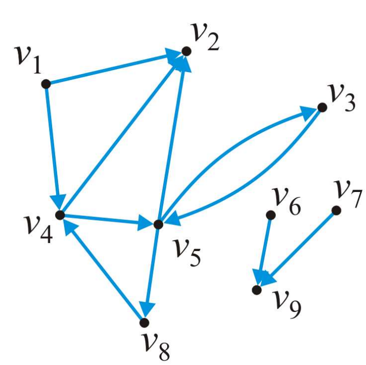
Directed acyclic graphs

Applications of directed acyclic graphs include:
- The parse tree constructed by a compiler
- A reference graph that can be garbage collected using simple reference counting
- Dependency graphs such as those used in instruction scheduling and **`makefiles`**
- Dependency graphs between classes formed by inheritance relationships in object-oriented programming languages
- Information categorization systems, such as folders in a computer
- Directed acyclic word graph data structure to memory-efficiently store a set of strings (words)

Reference: http://en.wikipedia.org/wiki/Directed_acyclic_graph

Representations

How do we store the adjacency relations?
- Binary-relation list
- Adjacency matrix
- Adjacency list



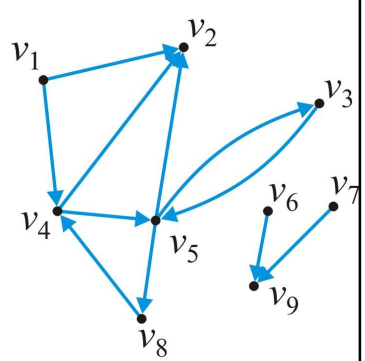
Binary-relation list

The most inefficient is a relation list:

– A container storing the edges

{(1, 2), (1, 4), (3, 5), (4, 2), (4, 5), (5, 2), (5, 3), (5, 8), (6, 9), (7, 9), (8, 4)}

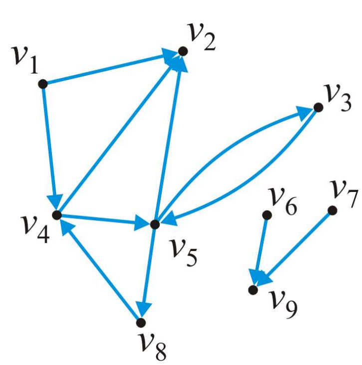
– Requires $\Theta(|E|)$ memory
– Determining if v_j is adjacent to v_k is $O(|E|)$
– Finding all neighbors of v_j is $\Theta(|E|)$



Adjacency matrix

– The matrix entry (j, k) is set to true if there is an edge (v_j, v_k)

	1	2	3	4	5	6	7	8	9
1		T		T					
2									
3					T				
4		T			T				
5		T	T					T	
6									T
7									T
8				T					
9									



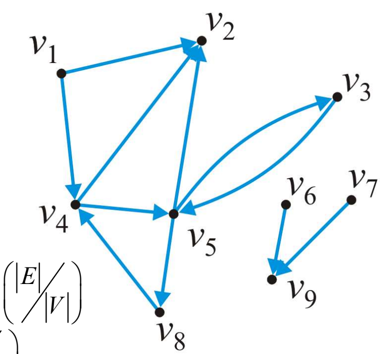
– Requires $\Theta(|V|^2)$ memory
– Determining if v_j is adjacent to v_k is $O(1)$
– Finding all neighbors of v_j is $\Theta(|V|)$

Adjacency list

Most efficient for algorithms is an adjacency list

– Each vertex is associated with a list of its neighbors

1 • → 2 → 4
2 •
3 • → 5
4 • → 2 → 5
5 • → 2 → 3 → 8
6 • → 9
7 • → 9
8 • → 4
9 •



– Requires $\Theta(|V| + |E|)$ memory

– On average:

- Determining if v_j is adjacent to v_k is $\mathrm{O}\left(\frac{|E|}{|V|}\right)$
- Finding all neighbors of v_j is $\Theta\left(\frac{|E|}{|V|}\right)$

The Graph ADT

The Graph ADT describes a container storing an adjacency relation

– Queries include:

- The number of vertices
- The number of edges
- List the vertices adjacent to a given vertex
- Are two vertices adjacent?
- Are two vertices connected?

– Modifications include:

- Inserting or removing an edge
- Inserting or removing a vertex (and all edges containing that vertex)

The run-time of these operations will depend on the representation

Some Famous Graph Problems

- **s-t Path**. Is there a path between vertices s and t?
- **Shortest s-t Path.** What is the shortest path between vertices s and t? (lowest total cost)
- **Minimum spanning tree**: finding the tree formed from graph edges that connects all the vertices of graph at the lowest total cost
- **Cycle Detection.** Does the graph contain any cycles?
- **Euler Tour.** Is there a cycle that uses every edge exactly once?
- **Hamilton Tour.** Is there a cycle that uses every vertex exactly once?

Summary

In this topic, we have covered:

- Basic graph definitions
  - Vertex, edge, degree, adjacency
- Paths, simple paths, and cycles
- Connectedness
- Weighted graphs
- Directed graphs
- Directed acyclic graphs

Next week: finding a route, shortest path problem