Lecture 11

- Graphs
- Vertices, edges, paths, cycles
- Sparse and dense graphs
- Representations: adjacency matrices and adjacency lists

Reading: Weiss, Chapter 9

Kinds of data structures

- You are familiar with these kinds of data structures:
  - unstructured structures: sets
  - linear, sequential structures: arrays, linked lists
  - hierarchical structures: trees
- Now we will look at *graphs*
- Graphs consist of
  - a collection of elements, called “nodes” or “vertices”
  - a set of connections, called “edges” or “links” or “arcs”, between pairs of nodes
- Graphs are in general not hierarchical or sequential: there is no requirement for a distinguished root node or first node, no requirement that nodes have a unique parent or a unique successor, etc.

Why graphs?

- Trees are a generalization of lists (a list is just a special case of a tree)...
- Graphs are a generalization of of trees (a tree is just a special case of a graph)...

- So, graphs are very general structures and are very useful in many applications

  - the set of machines on the internet, and network lines between them, form a graph
  - the set of statements in a program, and flow of control between them, form a graph
  - the set of web pages in the world, and HREF links between them, form a graph
  - the set of transistors on a chip, and wires between them, form a graph
  - the set of possible base sequences in a DNA gene, and mutations between them, form a graph
  - the set of possible situations that can arise in solving a problem or playing a game, and moves that get you from one situation to another, form a graph
  - et cetera...

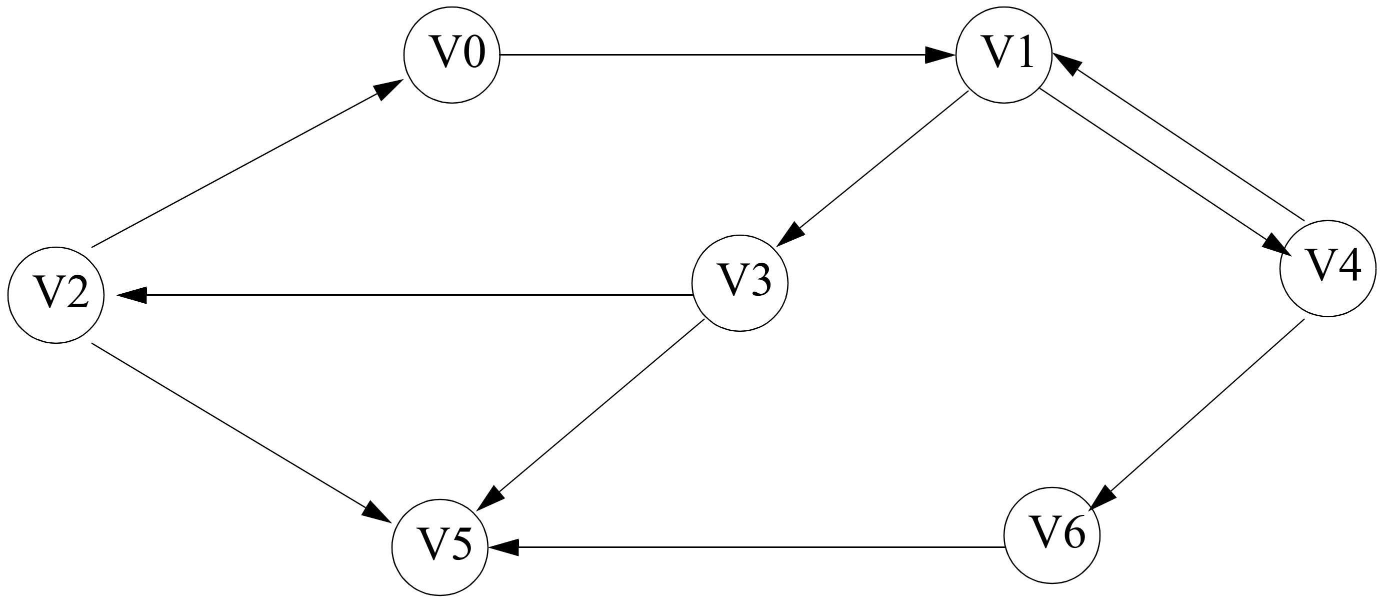
- We will look at a formal definition of a graph, some ways of representing graphs, and some important algorithms on graphs

Graphs: some definitions

- A graph G = (V,E) consists of a set of vertices V and a set of edges E
- Each edge in E is a pair (v,w) such that v and w are in V.
  - If G is an *undirected* graph, (v,w) in E means vertices v and w are connected by an edge in G. This (v,w) is an unordered pair
  - If G is a *directed* graph, (v,w) in E means there is an edge going from vertex v to vertex w in G. This (v,w) is an ordered pair; there may or may not also be an edge (w,v) in E
- In a *weighted* graph, each edge also has a “weight” or “cost” c, and an edge in E is a triple (v,w,c)
- When talking about the size of a problem involving a graph, the number of vertices |V| and the number of edges |E| will be relevant

Graphs: an example

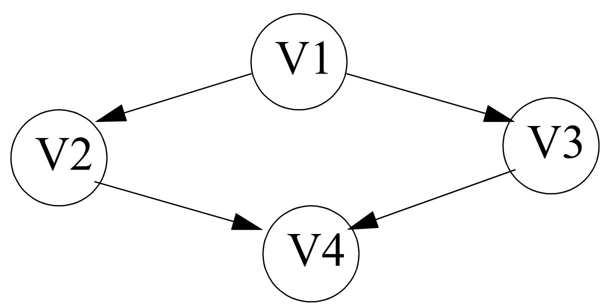
- Here is an unweighted directed graph:



- V = {   }
- |V| =
- E = {   }
- |E| =

Graphs: more definitions

- A *path* in a graph G=(V,E) is a sequence of vertices $v_1, v_2, ..., v_N$ in V such that (v_i, v_{i+1}) is in E for all i = 1,...,N-1.
- The *length* of a path is the number of edges in the path (might be zero)
- The *weighted length* of a path is the sum of the weights of the edges in the path
- A *simple path* is a path in which all the vertices are different (except the first and last can be the same)
- A *cycle* in a directed graph is a path of length >= 1 in which the first and last vertices are the same (in an undirected graph, the edges in a cycle must be distinct)
- A *simple cycle* is a cycle that is a simple path
- If a directed graph has no cycles, it is called a *directed acyclic graph* (DAG)
  - Is the example graph on the previous page a DAG?
  - Note: Every tree is a DAG, but not every DAG is a tree. Example:



Dense and sparse graphs

- If a directed graph has |V| vertices, how many edges can it have?
  - The first vertex can have an edge to every vertex (including itself): |V| edges
  - The second vertex can have an edge to every vertex (including itself): |V| edges
  - ... and so on for each of the |V| vertices; and all these edges are distinct
- So, the maximum total number of edges possible is $|E| = |V| x |V| = |V|^2$
- A graph with “close to” $|V|^2$ edges is considered *dense*
- A graph with “closer to” |V| edges is considered *sparse*

Representing graphs

- There are two major techniques for representing graphs:
  - Adjacency matrix
  - Adjacency list

- Each of these has advantages and we will look at each

Adjacency matrices

- An adjacency matrix is a 2D array
- The [i][j] entry in the matrix encodes connectivity information between vertices i and j
  - For an unweighted graph, the entry is “1” or “true” if there is an edge, “0” or “false” if there is no edge
  - For a weighted graph, the entry is the weight of the edge, or “infinity” if there is no edge
  - For an undirected graph, the matrix will be symmetric (or you could just use an upper-triangular matrix)
- There are |V| rows and |V| columns in an adjacency matrix, and so the matrix has $|V|^2$ entries
- This is space inefficient for sparse graphs

Adjacency matrix, an example

- Fill in this adjacency matrix for the example graph:

	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

Adjacency lists

- An adjacency list representation uses, well, lists
- Each vertex in the graph has associated with it a list of the vertices adjacent to it
- That is, if (v_j, v_k) is an edge in the graph, then v_j's adjacency list contains (a reference to) v_k
  - For a weighted graph, the list entry would also contain the weight of the edge
  - For an undirected graph, if v_j's adjacency list contains v_k , then v_k's adjacency list should contain v_j
- Using an adjacency list representation, each edge in a directed graph is represented by one item in one list; and there are as many lists as there are vertices
- Therefore the storage required is proportional to $|V| + |E|$, which is much better than $|V|^2$ for sparse graphs, and comparable to $|V|^2$ for dense graphs

Adjacency lists, an example

- Write down the adjacency lists to represent the example graph:

V0:

V1:

V2:

V3:

V4:

V5:

V6: