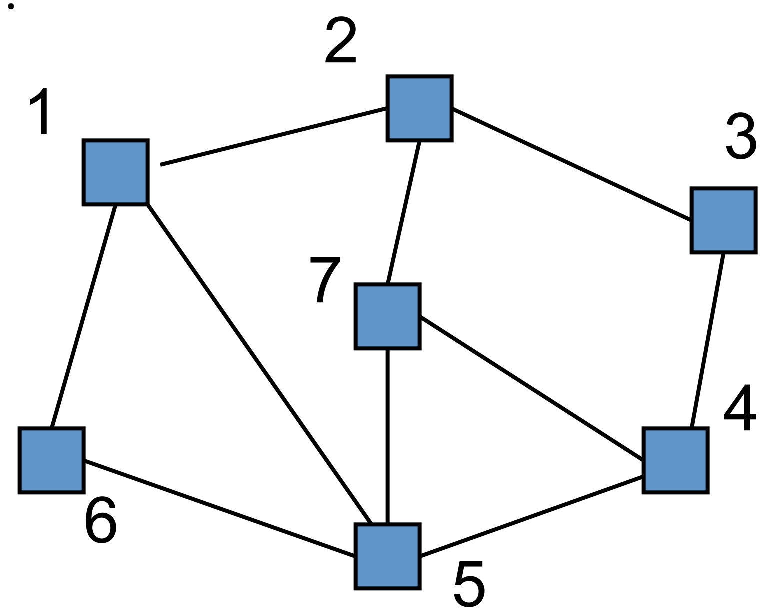
CSE 373: Data Structures and Algorithms

Lecture 20: Graphs II

Implementing a graph

- If we wanted to program an actual data structure to represent a graph, what information would we need to store?
  - for each vertex?
  - for each edge?



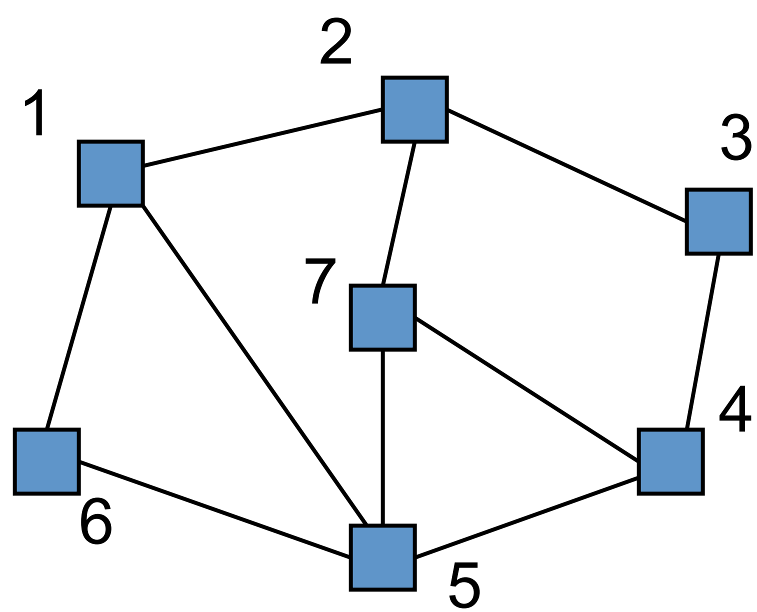
- What kinds of questions would we want to be able to answer quickly:
  - about a vertex?
  - about its edges / neighbors?
  - about paths?
  - about what edges exist in the graph?

- We'll explore three common graph implementation strategies:
  - edge list, adjacency list, adjacency matrix

Edge list

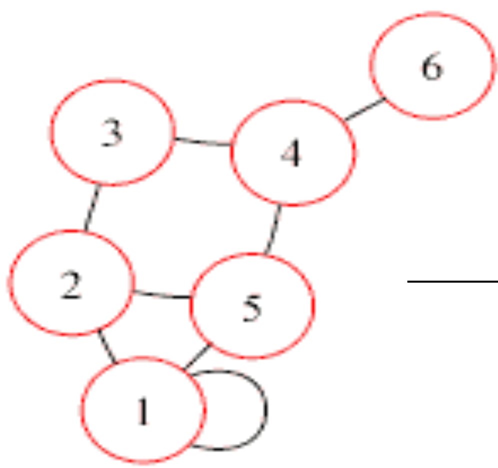
- **edge list**: an unordered list of all edges in the graph
- *advantages*
  - easy to loop/iterate over all edges
- *disadvantages*
  - hard to tell if an edge exists from A to B
  - hard to tell how many edges a vertex touches (its degree)

1	1	1	2	2	3	5	5	5	7
2	5	6	7	3	4	6	7	4	4



Adjacency matrix

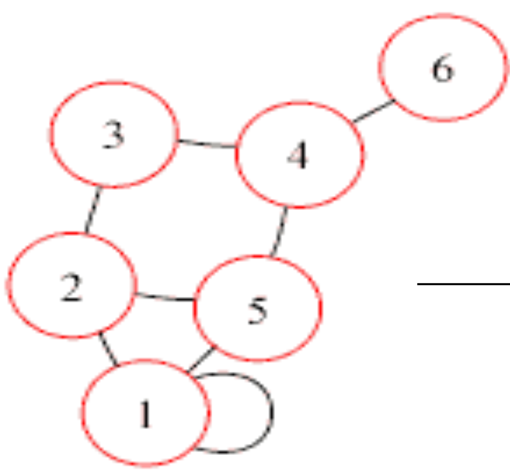
- **adjacency matrix**: an $n \times n$ matrix where:
  - the nondiagonal entry a_{ij} is the number of edges joining vertex *i* and vertex *j* (or the weight of the edge joining vertex *i* and vertex *j*)
  - the diagonal entry a_{ii} corresponds to the number of loops (self-connecting edges) at vertex *i*



$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Pros/cons of Adj. matrix

- *advantage*: fast to tell whether edge exists between any two vertices *i* and *j* (and to get its weight)
- *disadvantage*: consumes a lot of memory on sparse graphs (ones with few edges)

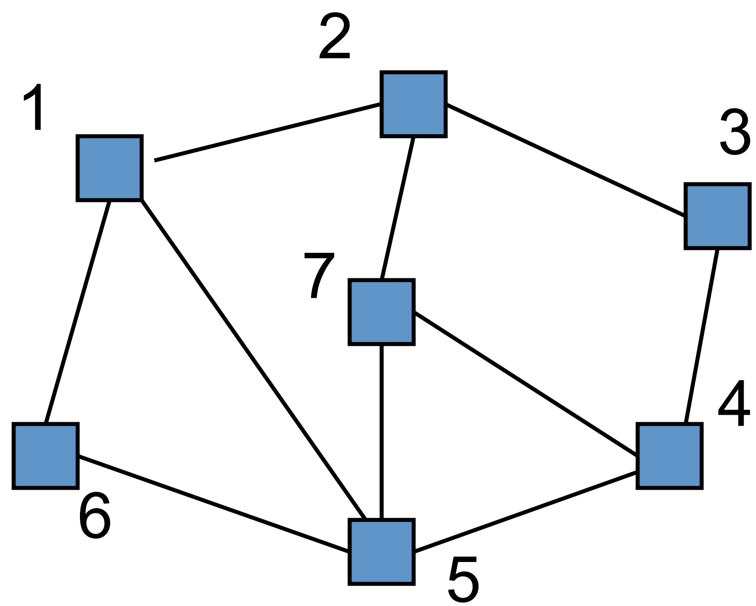


$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Adjacency matrix example

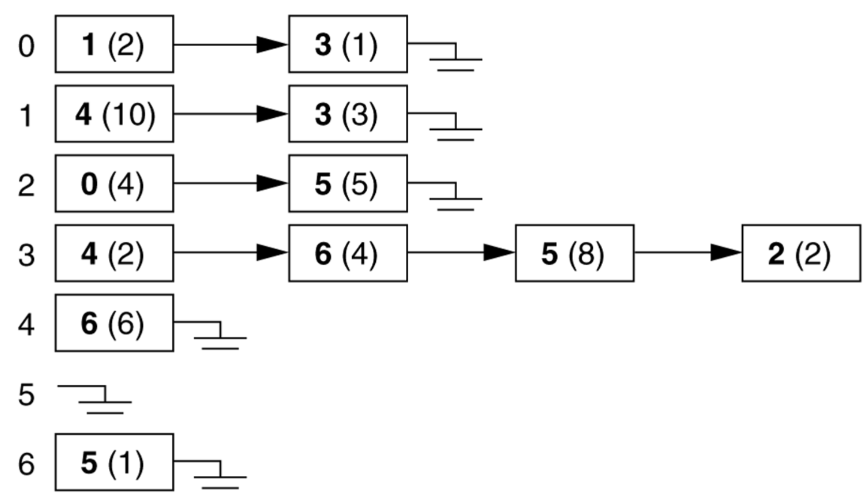
- The graph at right has the following adjacency matrix:
  - How do we figure out the degree of a given vertex?
  - How do we find out whether an edge exists from A to B?
  - How could we look for loops in the graph?

	1	2	3	4	5	6	7
1	0	1	0	0	1	1	0
2	1	0	1	0	0	0	1
3	0	1	0	1	0	0	0
4	0	0	1	0	1	0	1
5	1	0	0	1	0	1	1
6	1	0	0	0	1	0	0
7	0	1	0	1	1	0	0



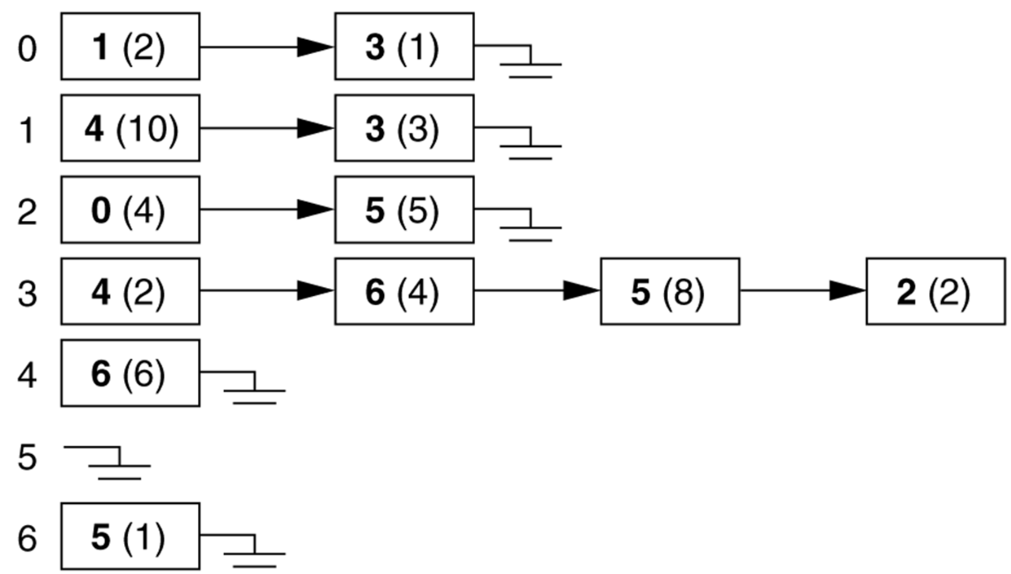
Adjacency lists

- **adjacency list**: stores edges as individual linked lists of references to each vertex's neighbors
  - generally, no information needs to be stored in the edges, only in nodes, these arrays can simply be pointers to other nodes and thus represent edges with little memory requirement



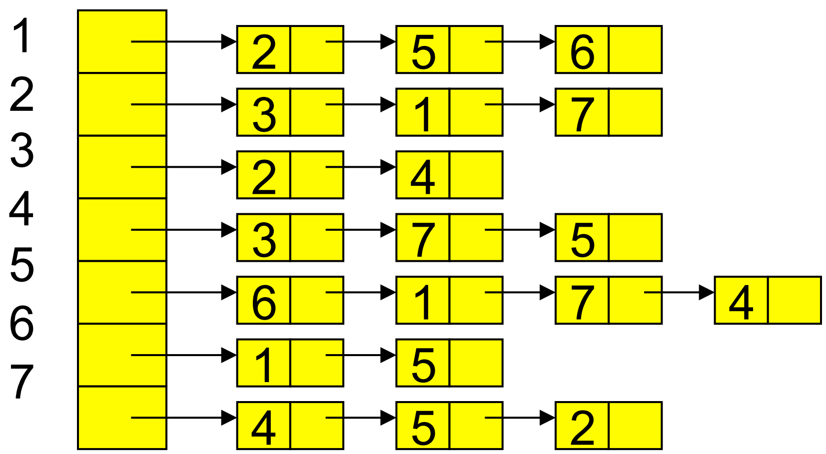
Pros/cons of adjacency list

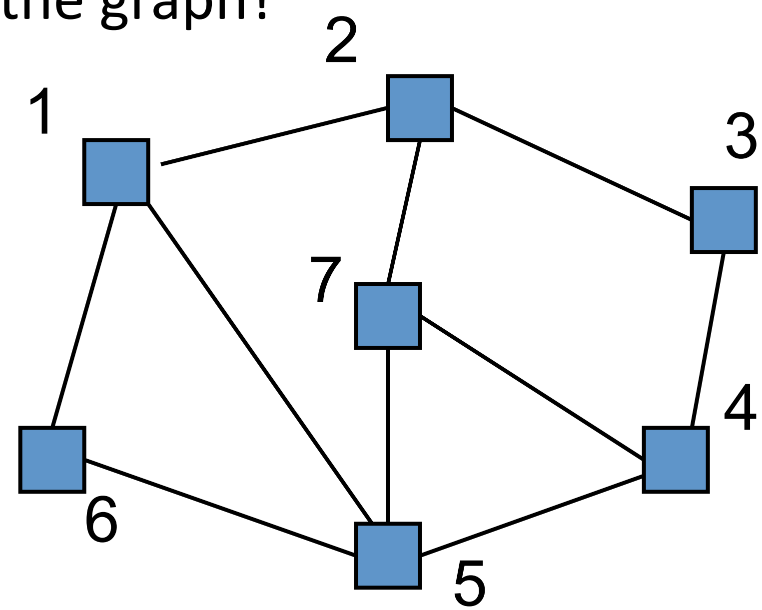
- *advantage*: new nodes can be added to the graph easily, and they can be connected with existing nodes simply by adding elements to the appropriate arrays; "who are my neighbors" easily answered
- *disadvantage*: determining whether an edge exists between two nodes requires O(*n*) time, where *n* is the average number of incident edges per node



Adjacency list example

- The graph at right has the following adjacency list:
  - How do we figure out the degree of a given vertex?
  - How do we find out whether an edge exists from A to B?
  - How could we look for loops in the graph?





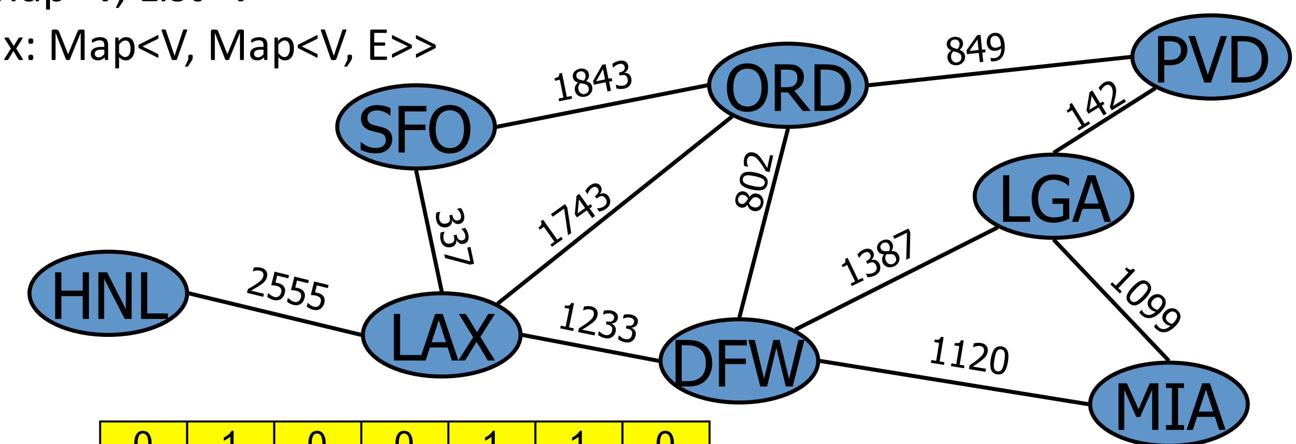
Runtime table

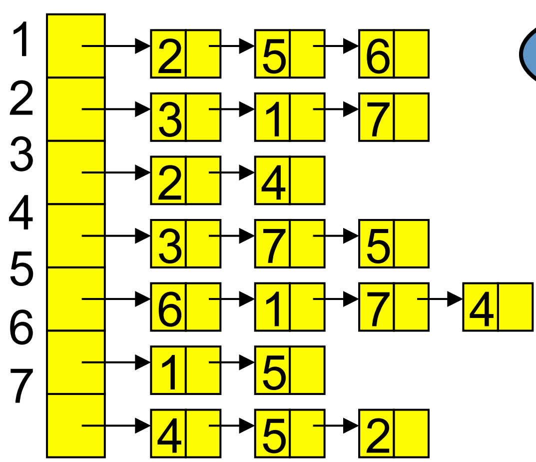
- ***n*** vertices, ***m*** edges - no parallel edges - no self-loops	Edge List	Adjacency List	Adjacency Matrix
Space	***n*** + ***m***	***n*** + ***m***	n^2
Finding all adjacent vertices to ***v***	***m***	deg(***v***)	***n***
Determining if ***v*** is adjacent to ***w***	***m***	deg(***v***)	1
inserting a vertex	1	1	n^2
inserting an edge	1	1	1
removing vertex ***v***	***m***	1	n^2
removing an edge	***m***	deg(***v***)	1

Practical implementation

- Not all graphs have vertices/edges that are easily "numbered"
  - how do we actually represent 'lists' or 'matrices' of vertex/edge relationships? How do we quickly look up the edges and/or vertices adjacent to a given vertex?

  - Adjacency list: Map<V, List<V>>
  - Adjacency matrix: Map<V, Map<V, E>>





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

Maps and sets within graphs

since not all vertices can be numbered, we can use:

1. adjacency list
   - each Vertex maps to a List of edges
   - Vertex --> List of Edges
   - to get all edges adjacent to V_1, look up
     List<Edge> neighbors = map.get(V_1)

2. adjacency map (adjacency matrix for objects)
   - each Vertex maps to a hashtable of adjacent vertices
   - Vertex --> (Vertex --> Edge)
   - to find out whether there's an edge from V1 to V2, call
     map.get(V1).containsKey(V2)
   - to get the edge from V1 to V2, call *map.get(V1).get(V2)*

Implementing Graph with Adjacency List

```
public interface Graph<V> {
    public void addVertex(V v);

    public void addEdge(V v1, V v2, int weight);

    public boolean hasEdge(V v1, V v2);

    public Edge<V> getEdge(V v1, V v2);

    public boolean hasPath(V v1, V v2);

    public List<V> getDFSPath(V v1, V v2);

    public String toString();
}
```

`Edge` class

```
public class Edge<V> {
    public V from, to;
    public int weight;

    public Edge(V from, V to, int weight) {
        if (from == null || to == null) {
            throw new IllegalArgumentException("null");
        }
        this.from = from;
        this.to = to;
        this.weight = weight;
    }

    public String toString() {
        return "<" + from + ", " + to + ", " + weight + ">";
    }
}
```