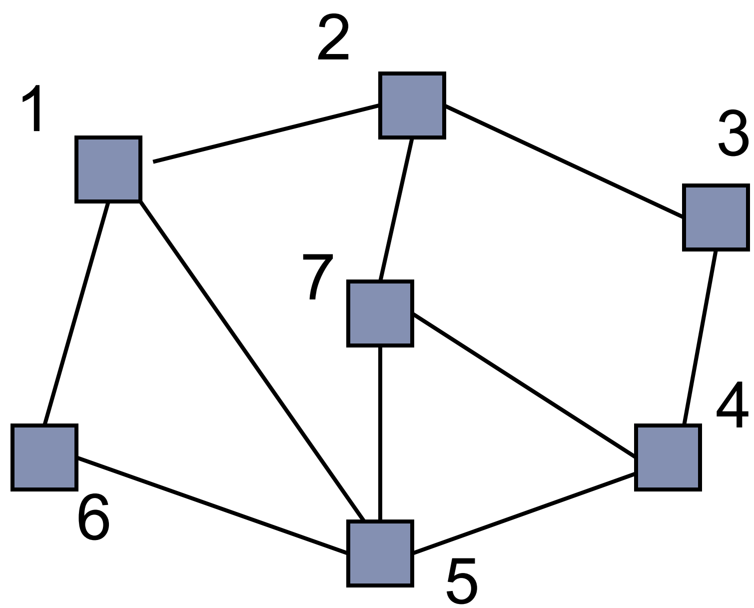
CSE 373
Data Structures and Algorithms

Lecture 20: Graphs II

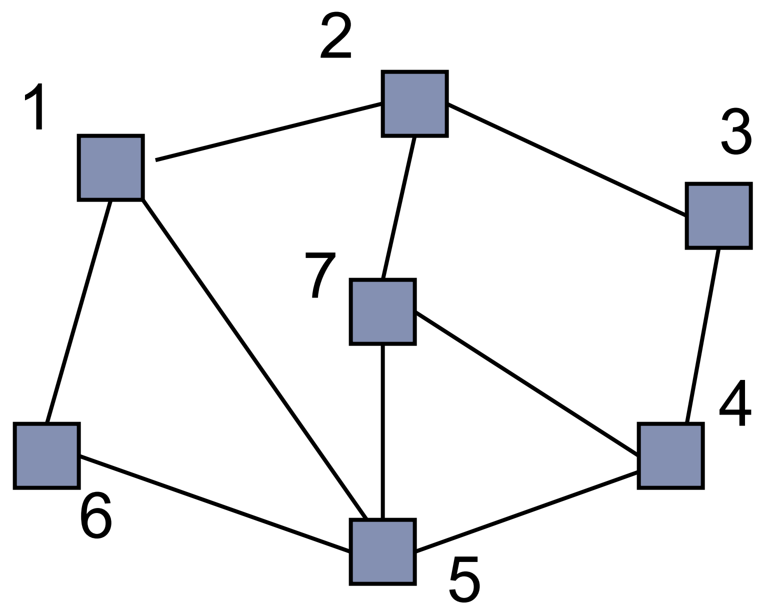
Implementing a Graph

- To program a graph data structure, what information would we need to store?
  - For each vertex?
  - For each edge?



Implementing a Graph

- What kinds of questions would we want to be able to answer (quickly?) about a graph *G*?
  - Where is vertex *v*?
  - Which vertices are adjacent to vertex *v*?
  - What edges touch vertex *v*?
  - What are the edges of *G*?
  - What are the vertices of *G*?
  - What is the degree of vertex *v*?



Graph Implementation Strategies

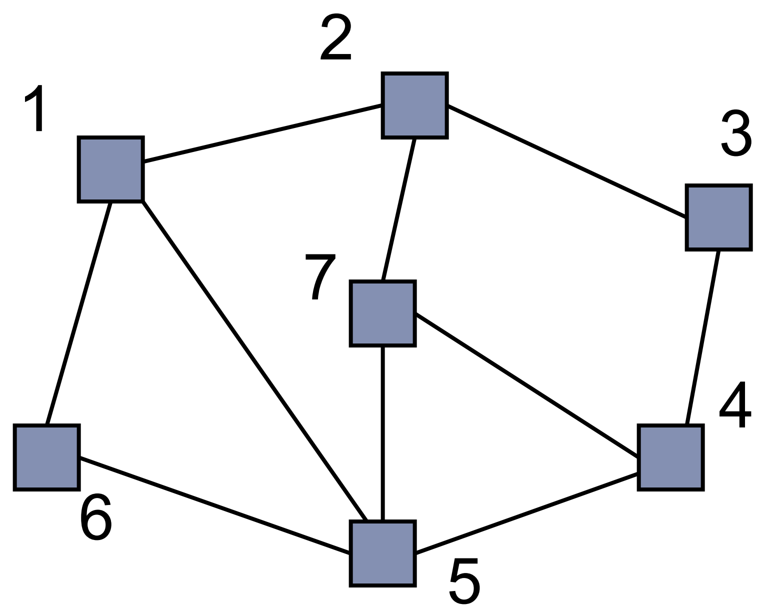
- Edge List
- Adjacency Matrix
- Adjacency List

Edge List

- **edge list**: an unordered list of all edges in the graph

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

* This is NOT an array



Edge List: Pros and Cons

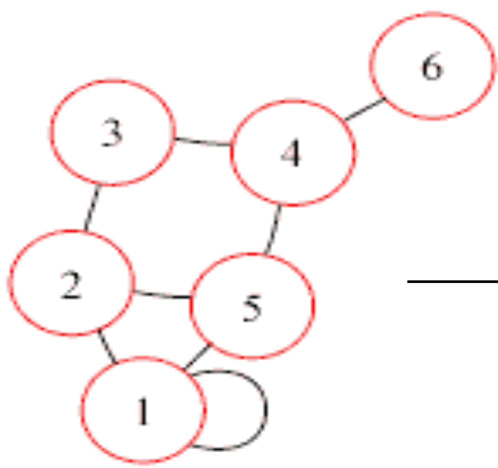
- *advantages*
  - easy to loop/iterate over all edges

- *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 × 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*



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

Adjacency Matrix: Pros and Cons

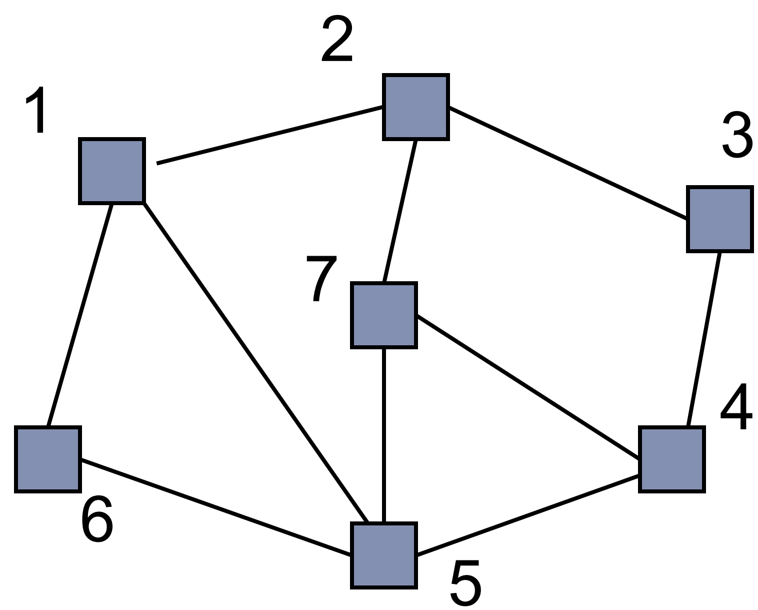
- *advantages*
  - fast to tell whether edge exists between any two vertices *i* and *j* (and to get its weight)

- *disadvantages*
  - consumes a lot of memory on sparse graphs (ones with few edges)
  - redundant information for undirected graphs

Adjacency Matrix Example

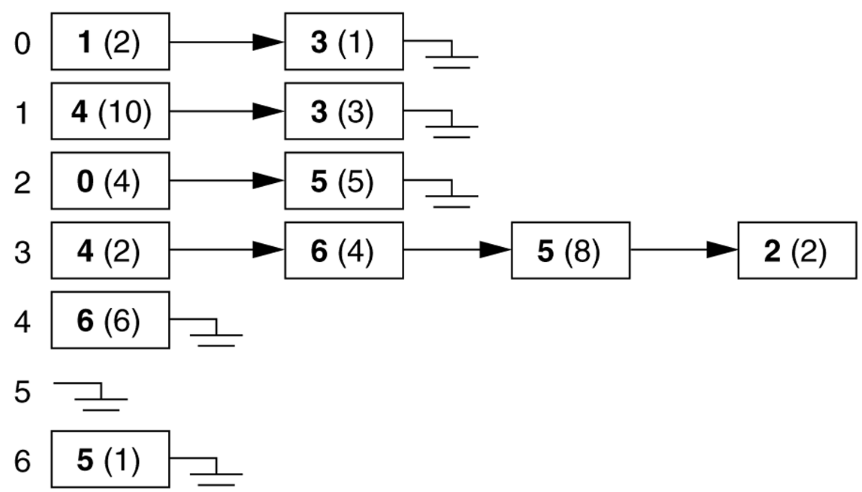
- How do we figure out the degree of a given vertex?
- How do we find out whether an edge exists from A to B?
- How could we look for loops in the graph?

	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



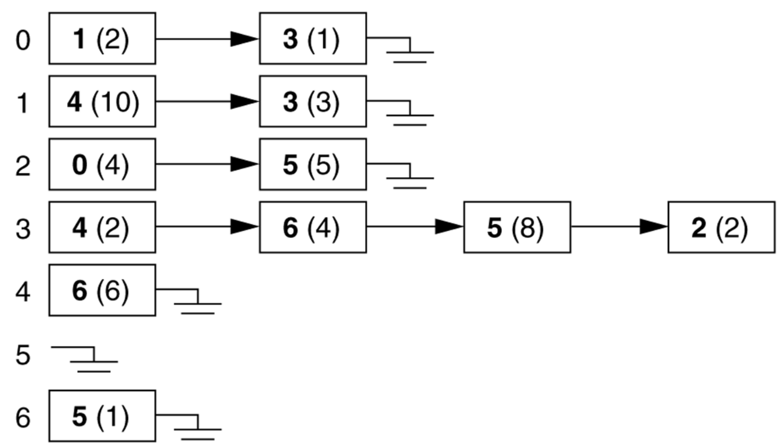
Adjacency Lists

- **adjacency list**: stores edges as individual linked lists of references to each vertex's neighbors



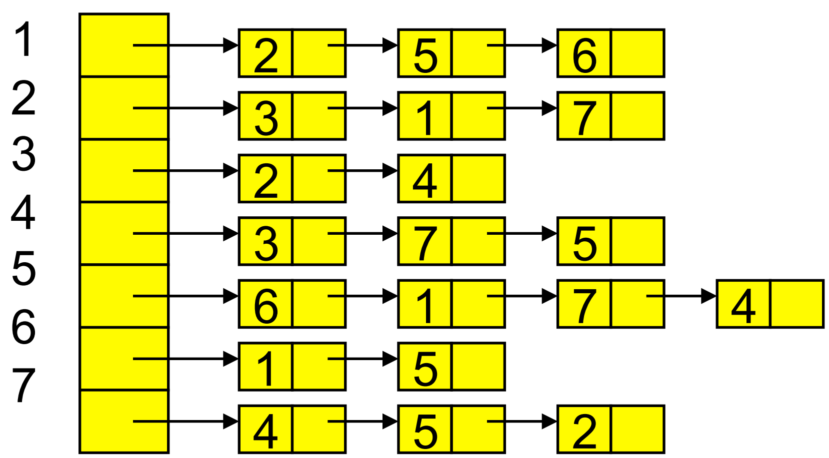
Adjacency List: Pros and Cons

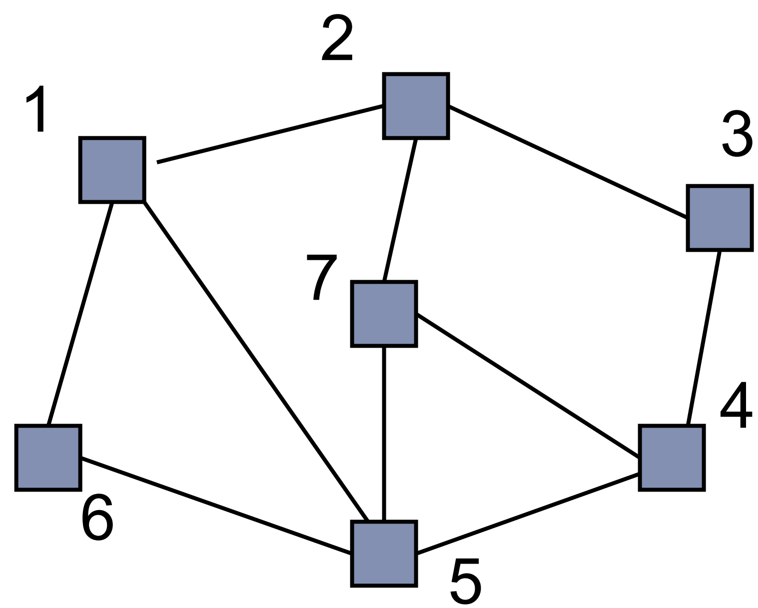
- *advantages*:
  - new nodes can be added easily
  - new nodes can be connected with existing nodes easily
  - "who are my neighbors" easily answered
- *disadvantages*:
  - determining whether an edge exists between two nodes: O(average degree)



Adjacency List Example

- 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?



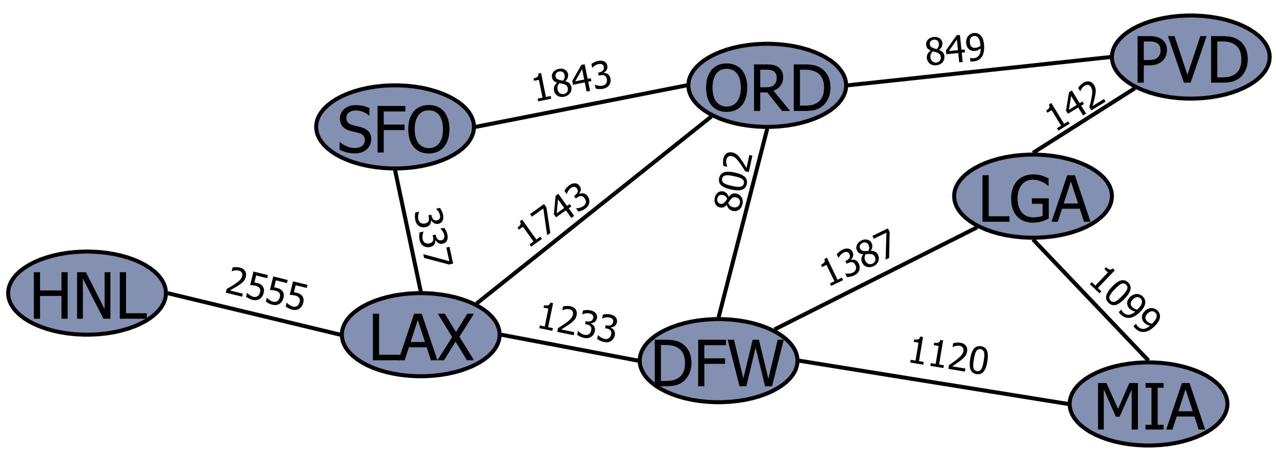


Runtime table

▪ 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
adding a vertex	1	1	n^2
adding an edge to v	1	1	1
removing vertex v	m	n**?**	n^2
removing an edge from v	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?



Practical Implementation

- Adjacency list
  - Each Vertex maps to a List of edges
  - Vertex → List<Edge>
  - To get all edges adjacent to v_1, look up List<Edge> neighbors = map.get(v_1)
- 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 v_1 to v_2, call map.get(v_1).containsKey(v_2)
  - To get the edge from v_1 to v_2, call map.get(v_1).get(v_2)

Implementing Graph with Adjacency List

```
public interface IGraph<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 + ">";
    }
}
```