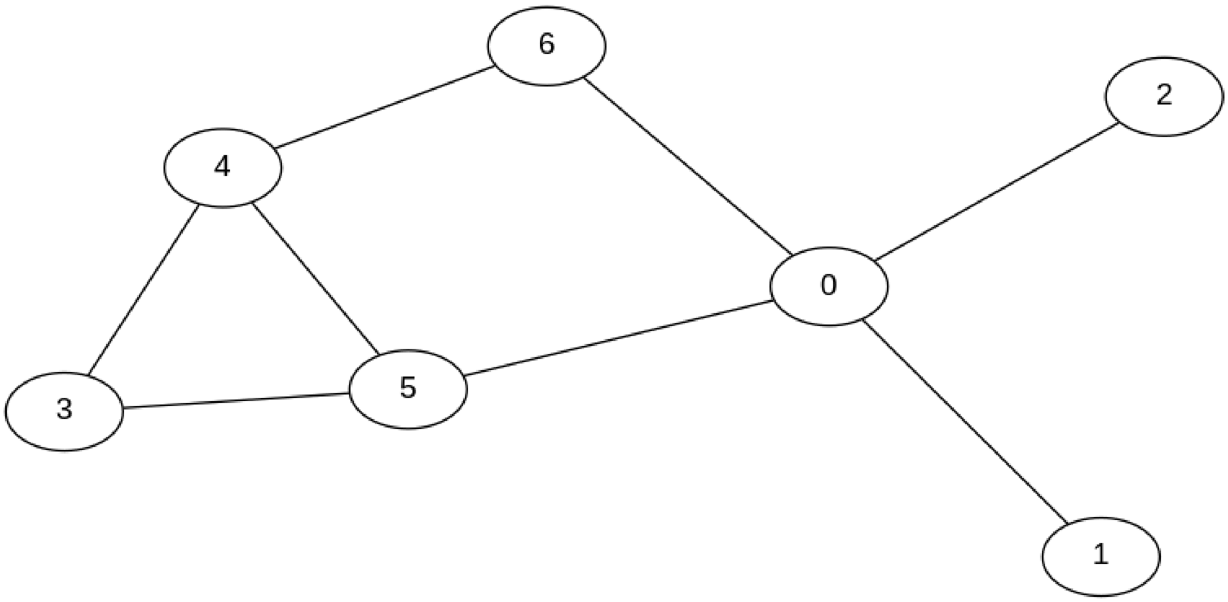
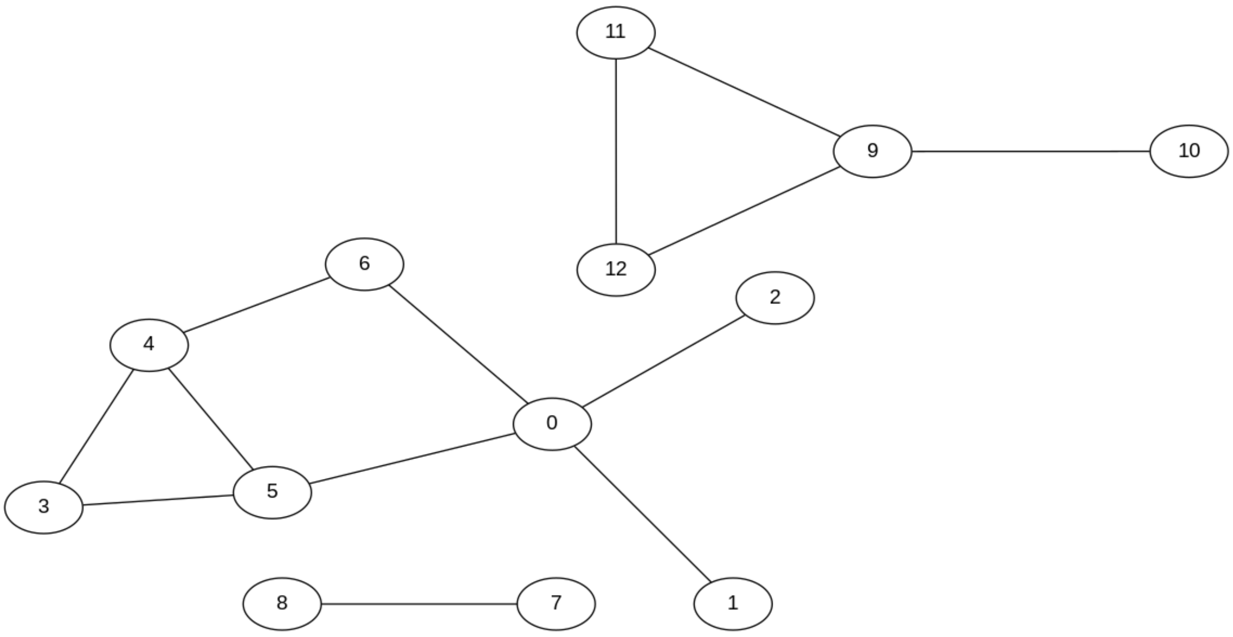
Lecture 11.1 : Graphs

Introduction

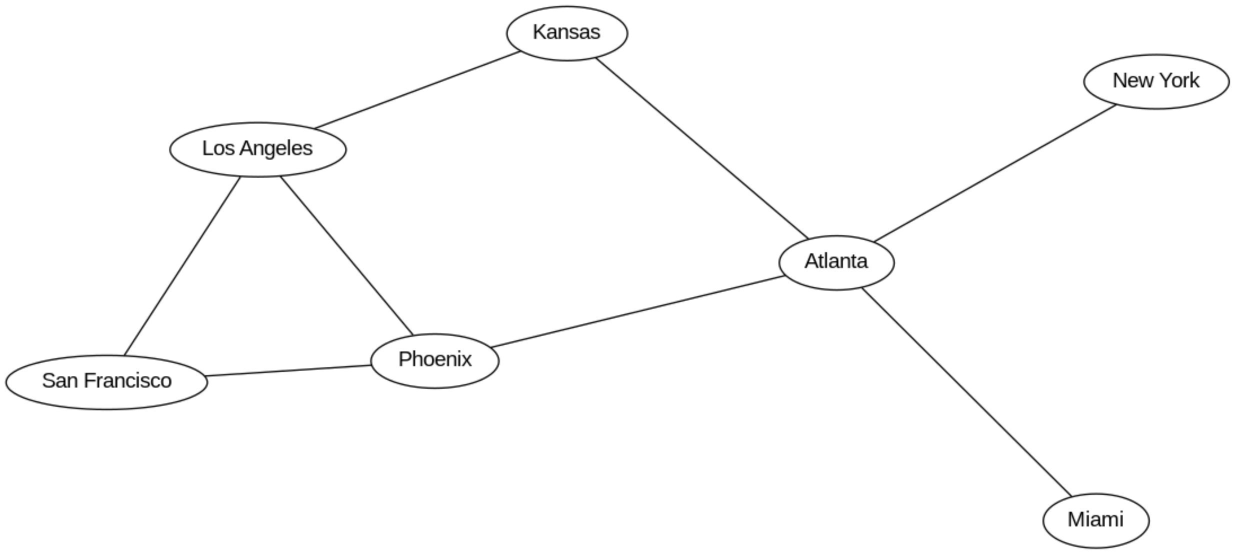
- A graph is a set of vertices connected by pairwise edges.
- Here is an example graph:



- Here is another example (this time with three connected components):



- Graphs have thousands of practical applications i.e. we can apply the graph abstraction to model thousands of real-world problems.
- For example, in a computer network, computers are vertices and fibre optic cables are edges.
- In a social network, people are vertices and friendships are edges.
- In a transport network, airports are vertices and flights are edges.
- Here is a transport network modelled with a graph:



- A *path* is a sequence of vertices connected by edges.
- A *cycle* is a path whose first and last vertices are the same.
- An understanding of and an ability to apply graph algorithms will make you a stand-out programmer.
- We first consider how to represent a graph in Python and then examine a small number of graph algorithms (there are hundreds).

Graph description

- We will describe a graph with a simple text file where the first line defines the number of vertices in the graph and the following lines define the edges (i.e. which vertices are connected to which).
- The first graph given above is then described by the following text file.

```
$ cat graph01.txt
7
0 1
0 2
0 5
0 6
3 4
3 5
4 5
4 6
```

Graph representation

- How are we going to represent a graph in Python?
- We will do so by developing a Graph class.
- The vertices can be represented simply by integers.
- If there are *V* vertices in the graph we will number them *0* to *V-1*.
- What about the edges? How are we going to capture which vertices are connected to which?
- To capture information about edges we will use an *adjacency-list* representation.
- This will involve a dictionary called *adj* which maps from vertex number (dictionary keys) to a list of vertices connected to that vertex (dictionary values).

Python implementation

```
class Graph(object):

   def __init__(self, V):
      self.V = V
      self.adj = {}
      for v in range(V):
         self.adj[v] = list()

   def addEdge(self, v, w):
      self.adj[v].append(w)
      self.adj[w].append(v)
```

Initialising our graph

- Below we initialise a graph based on its text description.

```
with open('graph01.txt') as f:
   V = int(f.readline())
   g = Graph(V)

   for line in f:
      v, w = [int(t) for t in line.strip().split()]
      g.addEdge(v, w)
```

Computing the degree of a vertex

- The degree of a vertex is the number of edges connecting it.
- Let’s add a method to our Graph class to return the degree of any vertex v.

```
class Graph(object):

   def __init__(self, V):
      self.V = V
      self.adj = {}
      for v in range(V):
         self.adj[v] = list()

   def addEdge(self, v, w):
      self.adj[v].append(w)
      self.adj[w].append(v)

   def degree(self, v):
      return len(self.adj[v])
```

```
with open('graph01.txt') as f:
   V = int(f.readline())
   g = Graph(V)

   for line in f:
      v, w = [int(t) for t in line.strip().split()]
      g.addEdge(v, w)

print(g.degree(0))
print(g.degree(1))
print(g.degree(4))
```

```
4
1
3
```

Computing the maximum degree

- Let's add a method to our Graph class to return the maximum degree of all vertices.

```
class Graph(object):

   def __init__(self, V):
      self.V = V
      self.adj = {}
      for v in range(V):
         self.adj[v] = list()

   def addEdge(self, v, w):
      self.adj[v].append(w)
      self.adj[w].append(v)

   def degree(self, v):
      return len(self.adj[v])

   def maxDegree(self):
      return max([self.degree(v) for v in range(self.V)])
```

```
with open('graph01.txt') as f:
   V = int(f.readline())
   g = Graph(V)

   for line in f:
      v, w = [int(t) for t in line.strip().split()]
      g.addEdge(v, w)

print(g.maxDegree())
```

```
4
```

Computing the average degree

- Let's add a method to our Graph class to return the average degree of all vertices.

```
class Graph(object):

   def __init__(self, V):
      self.V = V
      self.adj = {}
      for v in range(V):
         self.adj[v] = list()

   def addEdge(self, v, w):
      self.adj[v].append(w)
      self.adj[w].append(v)
   def degree(self, v):
      return len(self.adj[v])

   def maxDegree(self):
      return max([self.degree(v) for v in range(self.V)])

   def avgDegree(self):
      return sum([self.degree(v) for v in range(self.V)]) / self.V
```

```
with open('graph01.txt') as f:
   V = int(f.readline())
   g = Graph(V)

   for line in f:
      v, w = [int(t) for t in line.strip().split()]
      g.addEdge(v, w)

print(g.avgDegree())
```

```
2.2857142857142856
```