Graphs

Contents

In mathematics, a graph is a way of representing relational data. A graph $G(V, E)$ consists of a **vertex set** V, and an **edge set** $E \subseteq V \times V$.

Often vertices are referred to as **nodes**.

In a **directed graph**, there can be an edge (v_0, v_1) as well as an edge (v_1, v_0). This is good for representing asymmetric relations like v_1 is better than v_0 at some task.

In an **undirected graph**, the edge (v_0, v_1) is the same as the edge (v_1, v_0). This is good for representing relations like friendship.

In a **multigraph**, there can be multiple edges between the same vertices. Multigraphs can be directed or undirected.

We'll primarily consider undirected graphs today.

The **neighbors** of a vertex v are all vertices that participate in an edge with v $N(v) = \{w \in V \mid (v,w) \in E\}$

The **degree** of a vertex is the size of its neighborhood set.

Applications of Graphs

Graphs come up in a variety of situations studied in scientific computing

1. Studying social networks (e.g. Facebook)
2. Studying food webs
3. Control processes
4. PDE meshes

and more!

Representing Graphs

There are a variety of ways we might represent graphs on a computer

Here are some common representations:

1. Edge list
2. Adjacency list
3. Ajacency matrix
4. ...

We'll typically treat the vertex set as $[0, 1, \dots, N-1]$ where N is the number of vertices in the graph

Edge Lists

An edge list is exactly what you might think - a list of edges.

For example:

```
elist = [(0,1), (1,2), (2,3)]
elist
```

```
[(0, 1), (1, 2), (2, 3)]
```

Adjacency List

An adjacency list is a list of lists - every vertex has a list of **neighbors**

For our example above, we would have

```
adjlist = [
           [1],
           [0,2],
           [1,3],
           [2]
          ]
adjlist
```

```
[[1], [0, 2], [1, 3], [2]]
```

Adjacency Matrix

An adjacency matrix A is a $|V| \times |V|$ matrix, where $A[i,j] = \begin{cases} 1 & (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$ Continuing our example, we would have

```
import numpy as np
A = np.zeros((4,4))
for e in elist:
    A[e[0], e[1]] = 1
    A[e[1], e[0]] = 1

A
```

```
array([[0., 1., 0., 0.],
       [1., 0., 1., 0.],
       [0., 1., 0., 1.],
       [0., 0., 1., 0.]])
```

the adjacency matrix of an undirected graph is always symmetric (why?)

Often adjacency matrices are sparse, so it makes sense to use a sparse matrix format.

Exercises

Assume we are working with undirected graphs

1. Write a function to return an adjacency list from an edge list
2. Write a function to return an adjacency matrix from an adjacency list
3. Write a function to return an edge list from an adjacency matrix

```
# Your code here
```

A Graph Class

Usually, there is data associated with vertices and/or edges in a graph.

Let's define a graph class that allows us to store data.

We'll represent the graph as an edge list.

```
class Graph(object):
    def __init__(self):
        self.elist = []
        self.edata = dict() # edge data
        self.ndata = dict() # node data

    def add_node(self, i, **kwargs):
        """
        add a node to the graph, with data indexed by keywords
        """
        self.ndata[i] = dict(**kwargs)

    def add_edge(self, i, j, **kwargs):
        """
        add an edge to the graph, with data indexed by keywords
        """
        self.elist.append((i,j))
        self.edata[(i,j)] = dict(**kwargs)

    def node_data(self, i):
        if i in self.ndata.keys():
            return self.ndata[i]
        else:
            raise ValueError("No such node!")

    def edge_data(self, i, j):
        if (i,j) in self.edata.keys():
            return self.edata[(i,j)]
        elif (j,i) in self.edata.keys():
            return self.edata[(j,i)]
        else:
            raise ValueError("No such edge!")

    def edge_list(self):
        return self.elist

    def adjacency_list(self):
        pass

    def adjacency_matrix(self):
        pass
```

```
G = Graph()
G.add_node(0, name="dog")
G.add_node(1, name="cat")
G.add_node(2)
G.add_edge(0,1, weight=0.5)
```

```
print(G.node_data(2))
print(G.edge_data(0,1))
```

```
{}
{'weight': 0.5}
```

```
G.edge_list()
```

```
[(0, 1)]
```

Exercise

1. Implement the `adjacency_list` method
2. Implement the `adjacency_matrix` method

By Brad Nelson