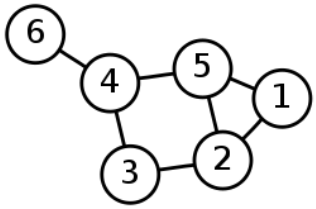
In this class…

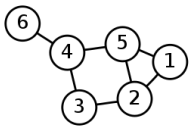
We're going to ask you to work with others on the same problem. So if you'd like to work on the earlier problems from Set 1, sit in the first 2 rows. If you'd like to work on the later problems from Set 1 (slightly more difficult), sit behind the first 2 rows.

CS149s: Graphs 1



Graphs

- A set of vertexes V
- A set of edges E
- Directed/undirected
- Weighted/unweighted



Adjacency Matrix

```
//v is number of vertexes
//adj matrix version
int[][] g1 = new int[V][V];

//add a directed edge between 12 and 13
g1[12][13] = 1;
//iterate over all the neighbors of 7
for(int i = 0; i < V; i++)
    if(g1[7][i] != 0)
         //do whatever
```

- Basically a 2 dimensional array M[x,y] is 1 if there is an edge between X and Y, 0 otherwise
- Can also be used for weighted graphs (but be careful if "0" is a valid weight)
- Might be too large if there are a lot of vertexes

Hybrid Adjacency List

```
//hybrid adjacency list version
ArrayList<List<Integer>> g2 = new ArrayList<List<Integer>>();
//initialize each list
for(int i = 0; i < V; i++)
    g2.add(new LinkedList<Integer>());

//add a directed edge between 12 and 13
g2.get(12).add(13);
//iterate over all the neighbors of 7
for(int adj : g2.get(7))
    //do whatever
```

- Array/List of vertexes, list of adjacent vertexes
- Can be used if you don't know V up front
- Can be used on weighted graphs too (you'd use a Map e.g. ArrayList<Map<Integer,Integer>>)

Depth First Search

```
// you have to be careful to initialize this at the right time
   HashSet<Integer> discovered = new HashSet<Integer>();

   public void doDFS(int[][] g, int cur) {
       discovered.add(cur);

       // in real code, you'd replace this print with doing
       // whatever processing your problem needs
       System.out.println("Visiting " + cur);

       // for each adjacent node
       for (int adj = 0; adj < g[cur].length; adj++) {
           if (g[cur][adj] == 0)
               // this means the edge does not exist
               continue;
           if (discovered.contains(adj))
               // already in the queue
               continue;
           doDFS(g, adj);
       }

   }
```

Breadth First Search

```
public void doBFS(int[][] g, int startingNode) {
        List<Integer> q = new LinkedList<Integer>();
        HashSet<Integer> discovered = new HashSet<Integer>();
        q.add(startingNode);
        discovered.add(startingNode);
        while (!q.isEmpty()) {
            int cur = q.remove(0);

            // in real code, you'd replace this print with doing
            // whatever processing your problem needs

            System.out.println("Visiting " + cur);

            // for each adjacent node
            for (int adj = 0; adj < g[cur].length; adj++) {
                if (g[cur][adj] == 0)
                    // this means the edge does not exist
                    continue;
                if (discovered.contains(adj))
                    // already in the queue
                    continue;
                q.add(adj);
                discovered.add(adj);
            }
        }
    }
```

For Next Week

- Do as many problems as you can from Problem Set 1
- Do 1 Problem from Problem Set 2 (for beginners, we recommend the 1st one which uses Prim's minimum spanning tree algorithm)