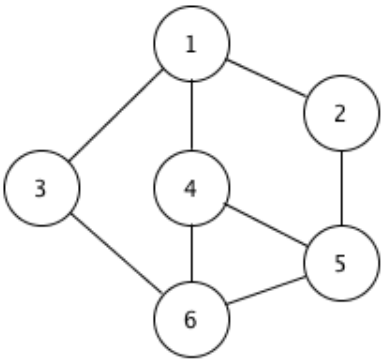
CECS 277 – Lecture 6 – Graphs and Trees Review

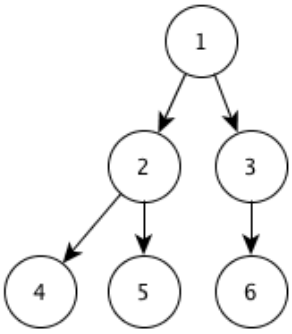
Graph – A graph is a set of ordered pairs that represent connections (edges) between nodes. Each node has a value and often keeps track of neighboring nodes. Graphs can often be cyclical, which means that there might be a path of nodes that leads back to the starting node.



The ordered pairs that represent the edges in this graph would be: (1,3), (3,1), (1,4), (4,1), (1,2), (2,1), (3,6), (6,3), (4,6), (6,4), (4,5), (5,4), (2,5), (5,2), (6,5), (5,6).

Each edge gets two connections since each edge is bi-directional, if they were singly directional, then they would only get one.

Tree – A tree is a set of nodes where each has a value and a list of child nodes. Any node may have zero or more children, but each child only has one parent. A tree may not be cyclical.



The values and children for this tree are:

1: (2, 3)
2: (4, 5)
3: (6, null)
4: (null, null)
5: (null, null)
6: (null, null)

Representing a Node – For either graphs or trees, a node is usually similar in nature. It is made up of a value, and a list of neighboring nodes (or children). A Node class may also require functions to retrieve information about a node or to modify the node.

```
public class Node{
     private int data;
     private ArrayList<Node> list = new ArrayList<Node>();
     public Node(int v){
          data = v;
     }
     public int getData(){
          return data;
     }
     public ArrayList<Node> getNeighbors(){
          return list;
     }
}
```

Building a Graph – A graph is a set of nodes and a list of their connections. To create the example graph from above, we would need:

```
public class Graph{
     private int [][] edges = {{1,3},{3,1},{1,4},{4,1},
                               {1,2},{2,1},{3,6},{6,3},
                               {4,6},{6,4},{4,5},{5,4},
                               {2,5},{5,2},{6,5},{5,6}};

     public ArrayList<Node> makeGraph(){
          ArrayList<Node> list = new ArrayList<Node>();
          for(int i=1;i<=6;i++){
               list.add(new Node(i));
          }
          for(int[]e: edges){
               list.get(e[0]-1).getNeighbors().add(
                                        list.get(e[1]-1));
          }
          return list;
     }
}
```

Building a Tree – A tree is a set of nodes and their children. To create the sample tree from above, we would need:

```
public class Tree{
     private int [][] children = {{1,2},{1,3},{2,4},
                                  {2,5},{3,6}};
     public ArrayList<Node> makeTree(){
          ArrayList<Node> list = new ArrayList<Node>();
          for(int i=1; i<=6; i++){
               list.add(new Node(i));
          }
          for(int[]c: children){
               list.get(c[0]-1).getNeighbors().add(
                                        list.get(c[1]-1));
          }
          return list;
     }
}
```

Traversing a Graph or a Tree – After you have built a graph or a tree, you will want to traverse through it in some fashion in order to do something with it, such as: search it, determine a path through it, or modify it in some way. To do this, you need to define the starting location, the ending location, and keep a list of where you have been. To begin, add the starting node to the list, then explore the neighboring nodes of that node, continue on until you have found the ending node.

```
public class TraverseTree{
     private Node start;
     private Node end;
     private Stack<Node> travels;

     public TraverseTree (Node s, Node e, Stack<Node> t){
          start = s;
          end = e;
          travels = t;
     }
     public boolean same(Node a, Node b){
          return a.getData() == b.getData();
     }
     public void traverse(){
          Node n;
          travels.push(start);
          boolean found = false;
          do{
               n = travels.pop();
               System.out.println("Node "+n.getData());
               if(same(n, end)){
                    found = true;
               }else{
                    for(Node c: n.getNeighbors()){
                         travels.push(c);
                         System.out.print(c.getData()+" ");
                    }
               }
          }while(!found && !travels.empty());
     }
}
public class TestTree{
     public static void main{String [] args){
          Tree t = new Tree();
          ArrayList<Node> list = t.makeGraph();
          Stack<Node> path = new Stack<Node>();
          Node end = new Node(6);
          TraverseTree t = new TraverseTree(list.get(0),
                                            end, path);
          t.traverse();
     }
}
/*Output
Node 1
2 3 Node 3
6 Node 6
*/
```