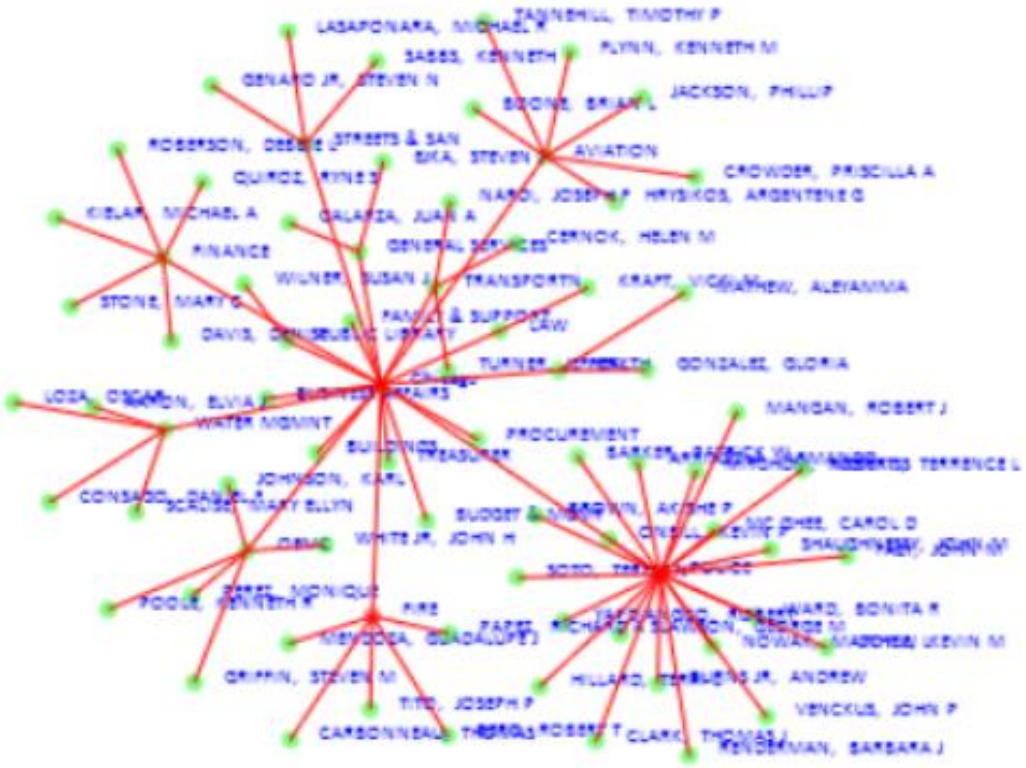
Lecture 28 - Networks, Graphs and Trees (part 1)

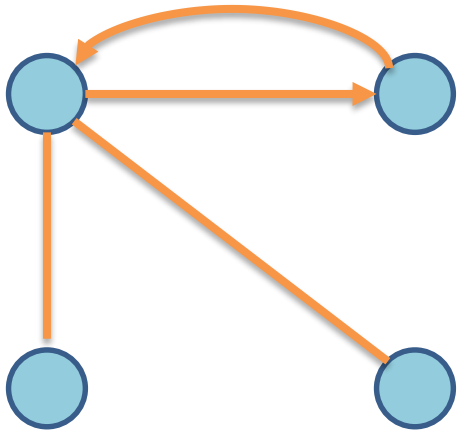
John R. Williams and Abel Sanchez, MIT



Networks and Graphs

Network = Graph

A graph is a set of nodes joined by a set of links either directed or not directed.

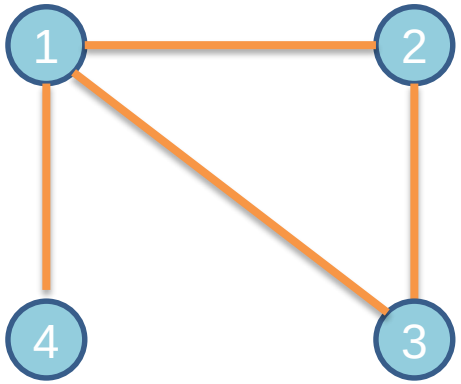


The links may be weighted or un-weighted. For weighted links think of multi-lane highways where the weight corresponds to the number of lanes.

Degree of a node

The goal of network theory is to give us insight into the properties of networks.

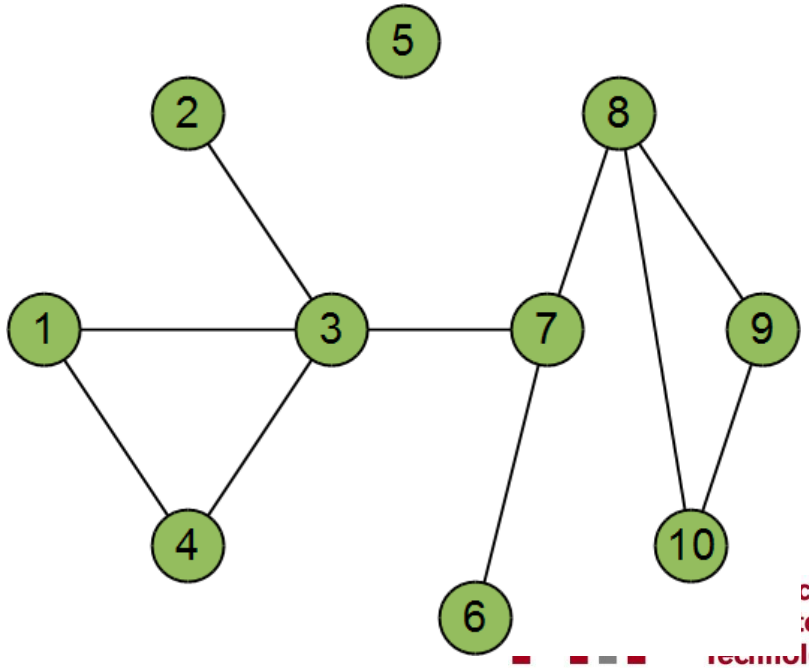
An example of a property is the degree of a node. The degree is the number of links attached to the node. In the network below node 1 has degree 3, node 2 degree 2, node 3 degree 2 and node 4 degree 1. Often k is used to represent the degree.



Concepts: Degree Distribution

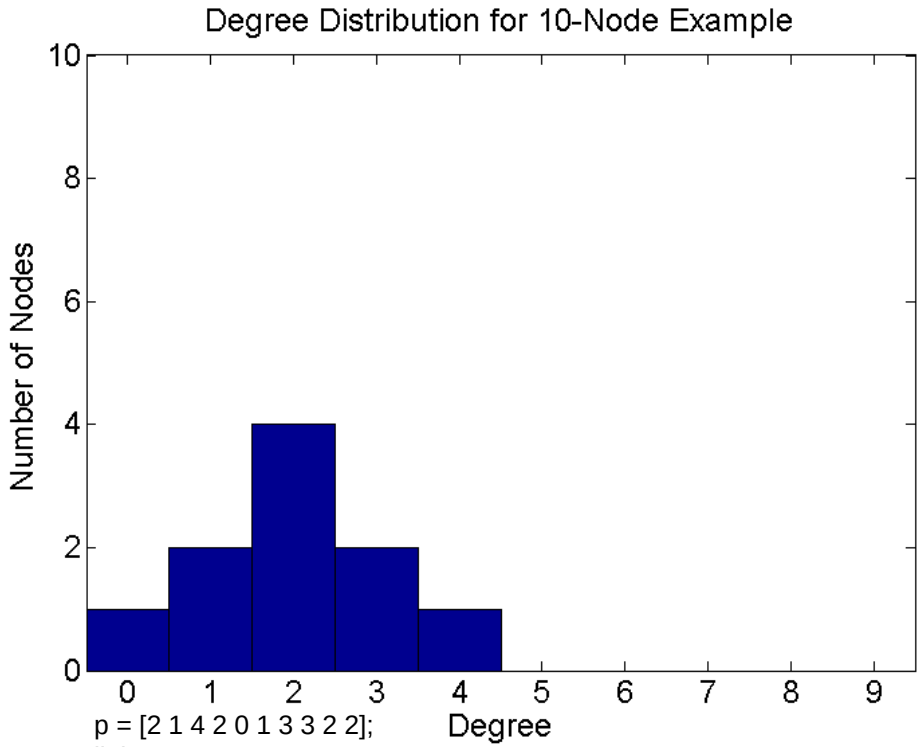
- A defining characteristic of network structure
- Define p_k to be fraction of nodes in a network that have degree, k
- In this example: $p_0 = 1/10$, $p_1 = 2/10$, $p_2 = 4/10$, $p_3 = 2/10$, $p_4 = 1/10$
- **Can you see why?**

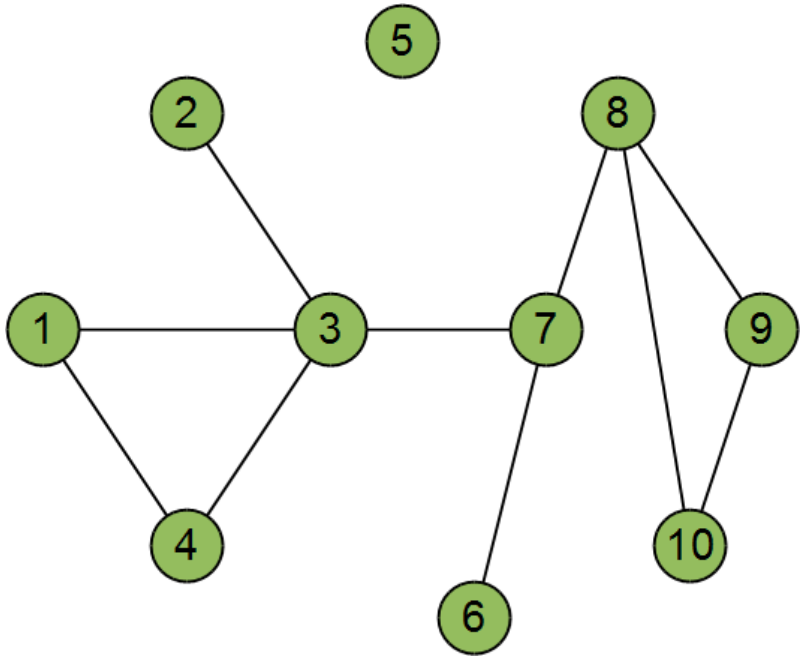
- Can also be thought of as the probability that a chosen node has degree, k



Concepts: Degree Distribution

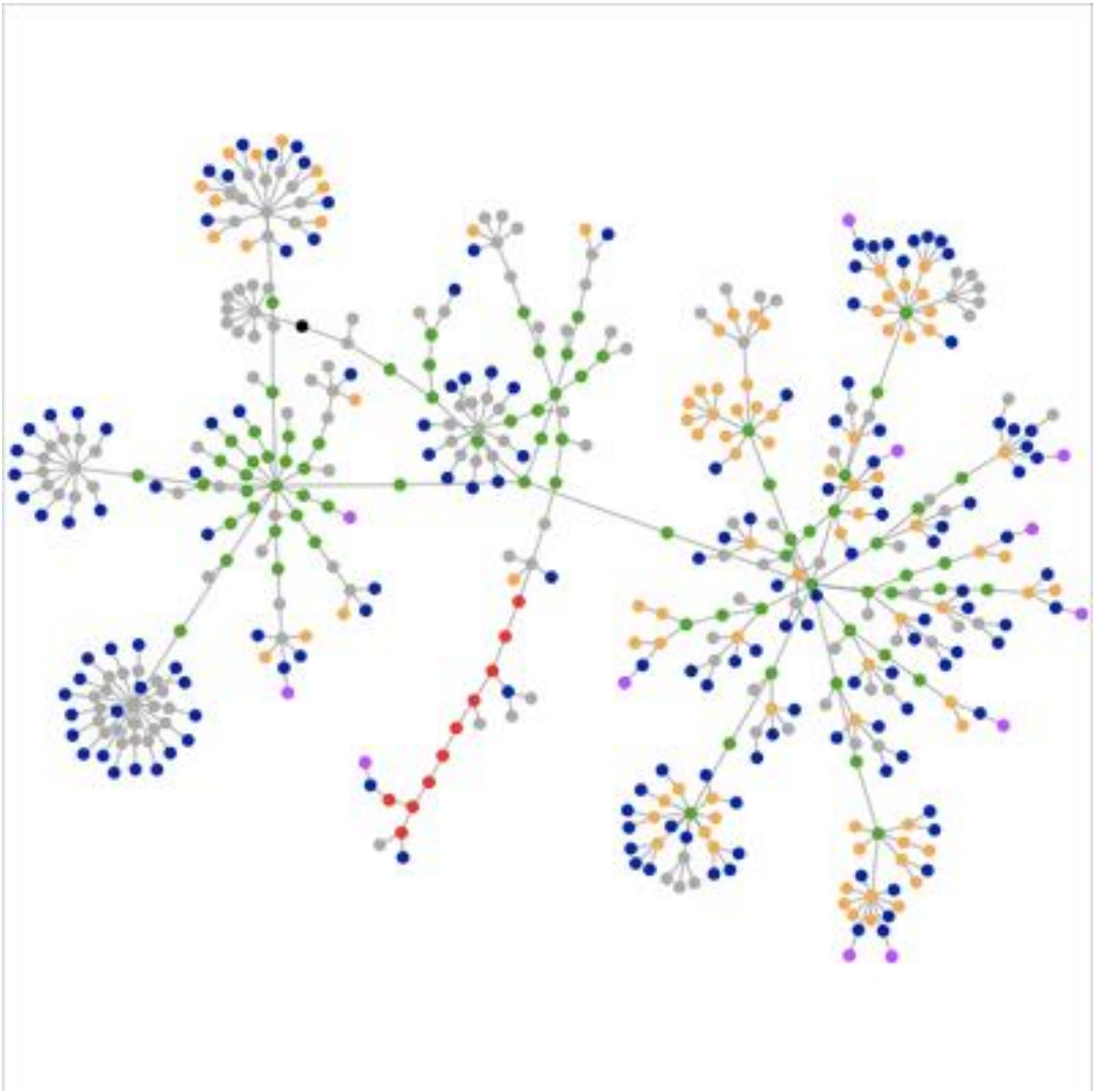
- Extending the probability concept of degree distribution, it can be plotted as a histogram






```
p = [2 1 4 2 0 1 3 3 2 2];
links = 0:9;
hist(p,links)
set(gca, 'FontSize', 18, 'XLim',[-0.5 9.5], 'YLim', [0 10]);
xlabel('Degree', 'FontSize', 18)
ylabel('Number of Nodes', 'FontSize', 18)
title('Degree Distribution for 10-Node Example', 'FontSize', 18)
```

Hubs are often most important nodes in a network.

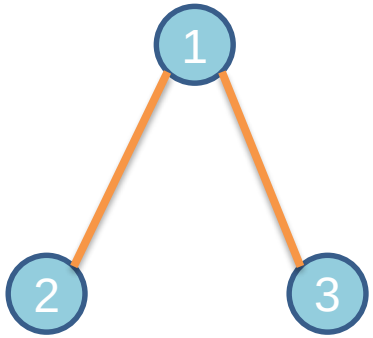


MITei

Massachusetts Institute of Technology

Back to Computing – Starting with Trees

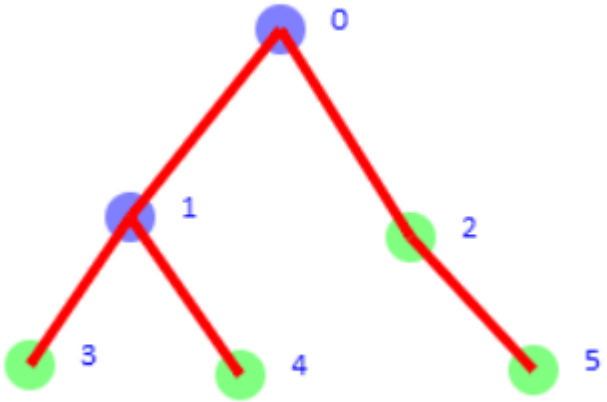
Lets look now at how we represent trees, which are a sub-set of graphs, in the computer. Lets consider a node such as node 1. The data structure needs to capture that its connect to nodes 2 and 3.




```
node ={
id: id,
children: [{node2}, {node3} ]
}
```

In Class Exercise 1

The present code generates two nodes with a link between them. It also writes out JSON for the network assuming node 1 is the “root” and all others are children of this node ie we are dealing with a ‘tree’. Create the following nodes as shown below. Hand in the screen shot and the JSON data structure written to the console.



City of Chicago Data

One of the data sets is the salary data for all city employees.

We want to plot out the data to understand it better.

The data looks like this.

var data = [[1, "26B361A2-F674-467D-9374-1208DBAFFBE0", 1, 1389641641, "700397", 1389641641, "700397", "{\n}", "AARON, ELVIA J", "WATER RATE TAKER", "WATER MGMNT", "87228.00"]

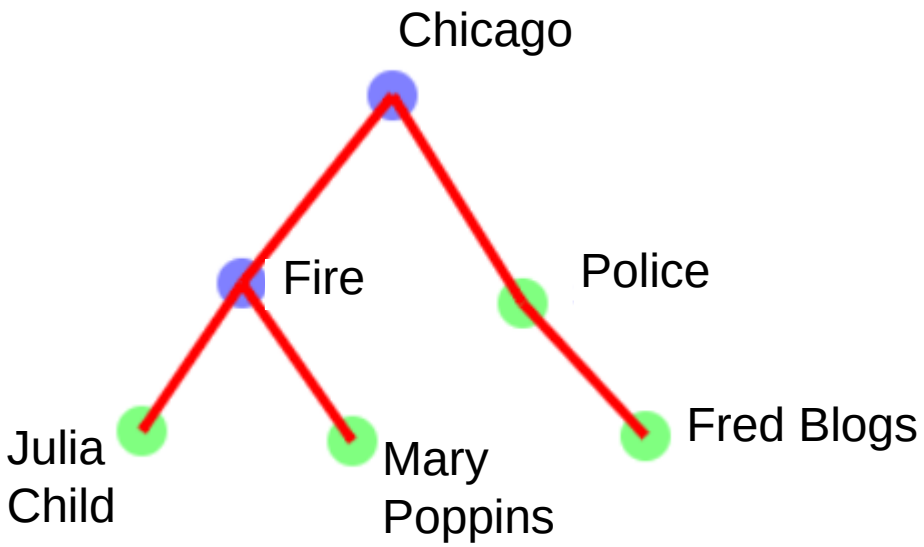
, [2, "2FDAC83E-C605-4DAF-8673-36916BD1C5D3", 2, 1389641641, "700397", 1389641641, "700397", "{\n}", "AARON, JEFFERY M", "POLICE OFFICER", "POLICE", "85372.00"]

Data structure

[[…… name, x, department, salary], […. name, x, department, salary]….]

Its basically an array of arrays and within each array we need to pull out locations 8,10,11.

Since the data is not in a tree structure we'll need to first decide on the structure we want to create in terms of node hierarchy.



In Class Exercise 2

In the 2nd starter code I've added the capability to rip through the Chicago salary data and create 3 levels of nodes. However, I haven't quite got it right. The 2nd level seems to repeat WATER MGMNT twice. Also the layout is not very good.

First correct the problem with the Water Management.

Next think about how we might lay out the nodes in a better way.

Searching the Tree Structure

Now we have a tree structure we want to “walk” the tree starting at the “root”.

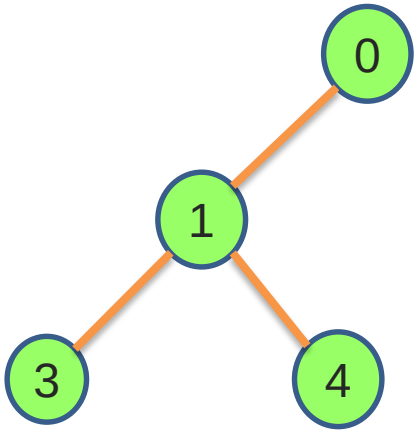
We’ll apply what is called Breadth First Search which starts at the root node then goes layer by layer downwards, visiting all the nodes in layer 2 before proceeding to those at layer 3.

Lets see how it works.

Basic Data Structure – Leaving a Trail

node = { children : children [ ] }

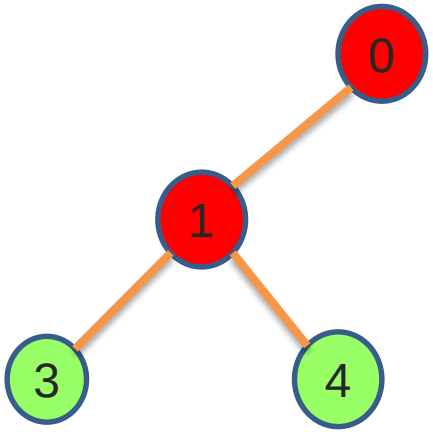
If we want to keep track of a node we store a reference to it in an array. Eg [0, 1] might indicate we have moved from node 0 to 1. Sometimes these are called ‘bread crumbs’ because they allow us to retrace our steps.



Array

Basic Data Structure – Flagging nodes visited

node = {children : children [ ], color:'Red' }
A mechanism we use is to color nodes we have 'visited' so that we don't repeat our steps. Here I'm going to color nodes I've visited 'Red'



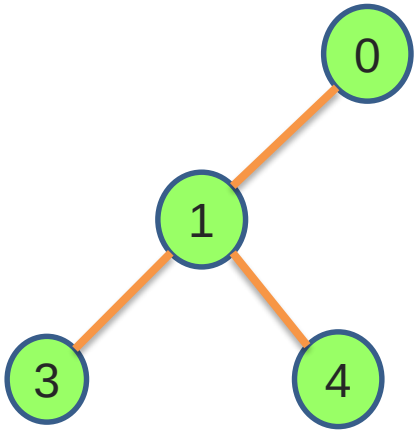
Stack

Now we are ready to systematically explore the tree using the array to retrace our steps and the color to flag that we've already explored part of the tree. Depending on the way we explore the tree we use the array as a Stack or a Queue.

Trees- Basic Data Structure

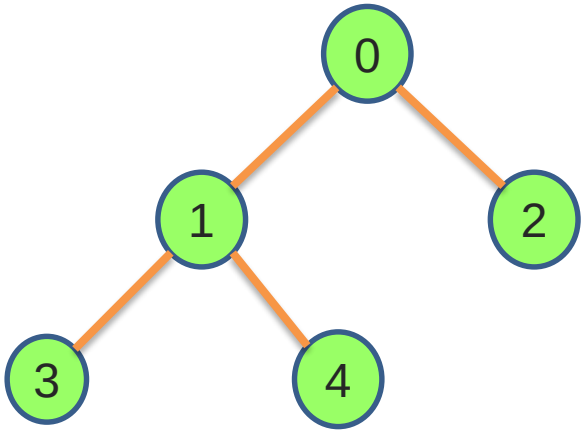
A node may have at most one parent but many children. Eg node 1 has parent 0 and children [3, 4]. Thus, the minimum data structure for any node is as below

node = {children : children [ ] }

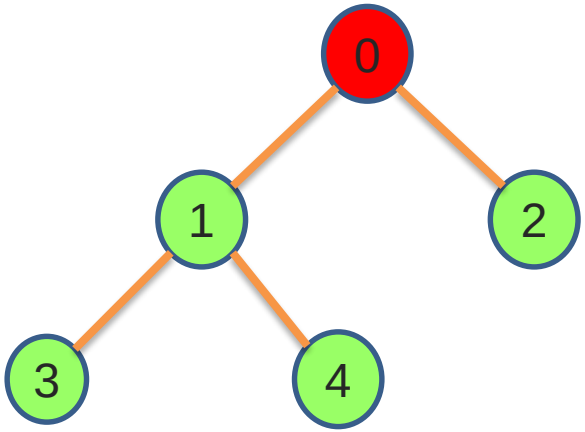


Breadth First Search

Queue

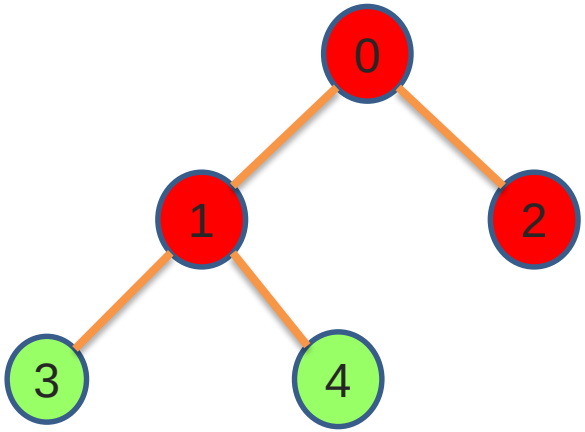


Breadth First Search



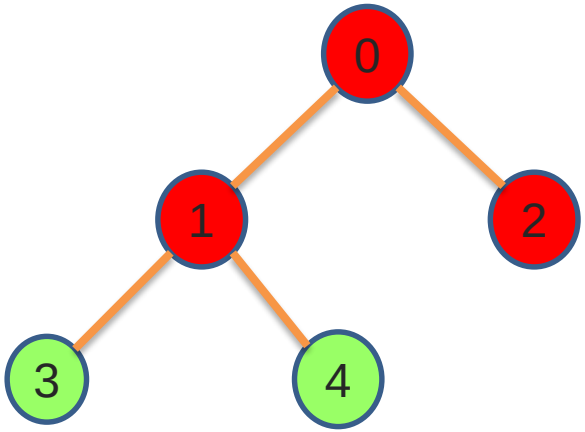
Queue

Breadth First Search



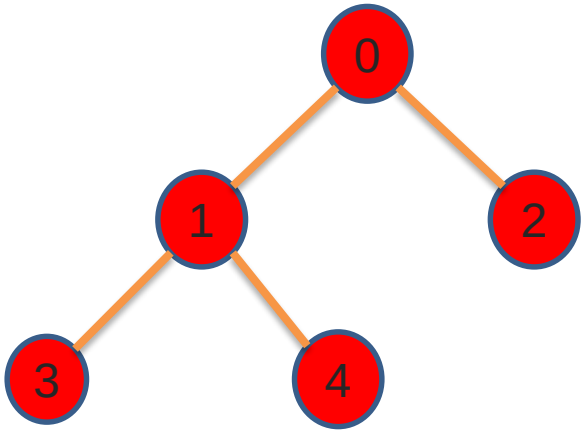
Queue

Breadth First Search



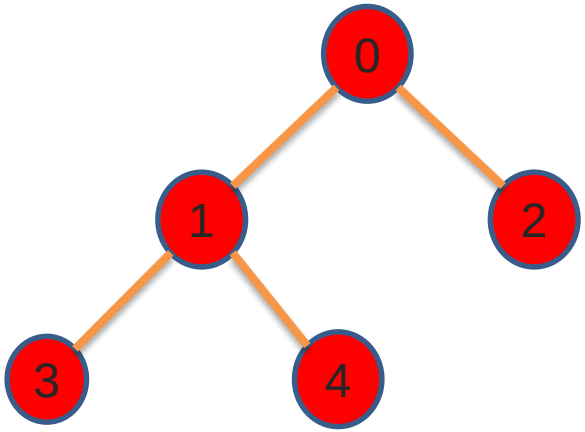
Queue

Breadth First Search



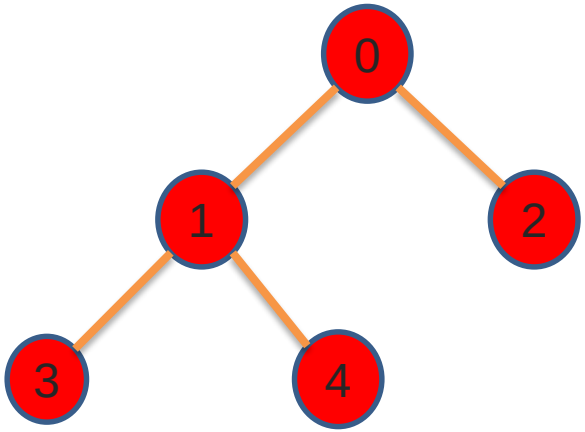
Queue

Breadth First Search



Queue

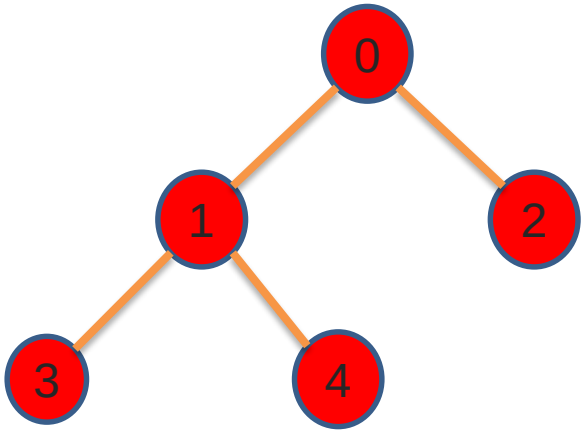
Breadth First Search



Queue

Breadth First Search

Queue



In Class Exercise 3

The starter code implements Breadth First Search. I want you now to understand of Breadth First Search works.

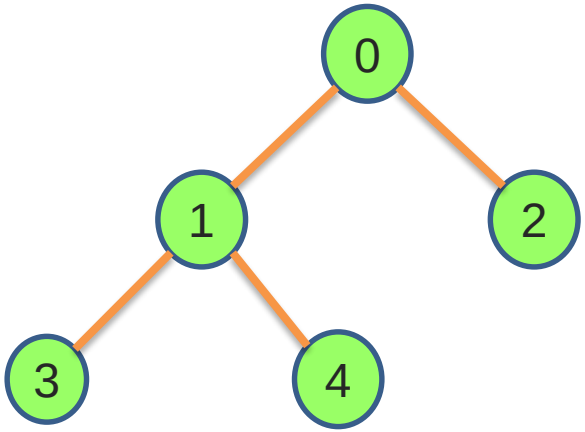
Part 1- Print out the state of the Queue after any change to the length of the Queue. ie when something is added or something taken out. Use ‘printArray(xxx)’ to do this.

Hand in a screen shot of the history of the Queue.

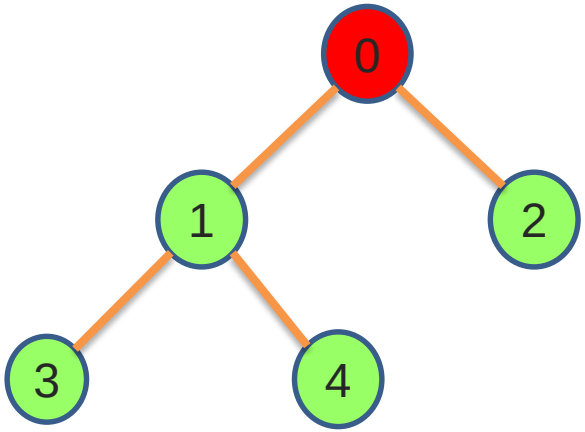
Part 2- I want to calculate the “degree k” of every node. Modify the code to print this to the console.

Depth First Search

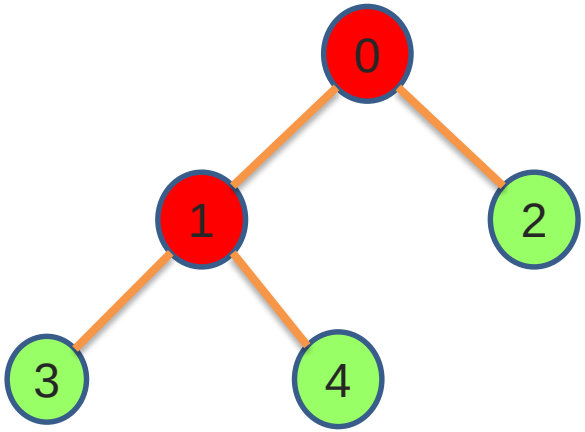
Stack



Depth First Search

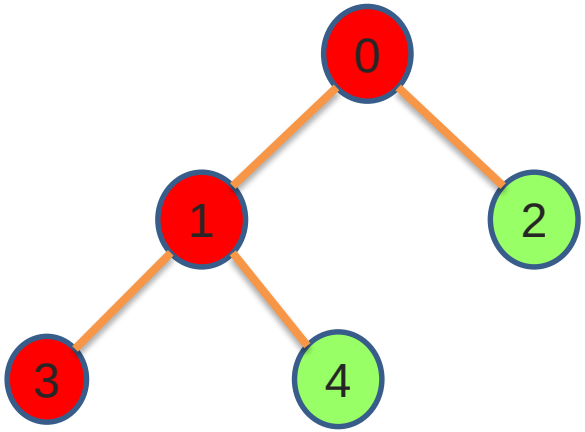


Stack



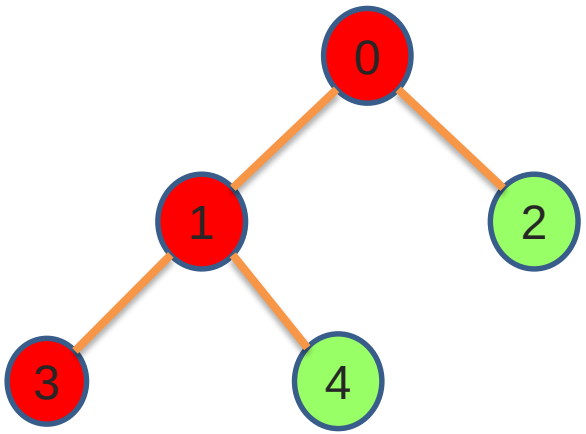
Stack

Depth First Search



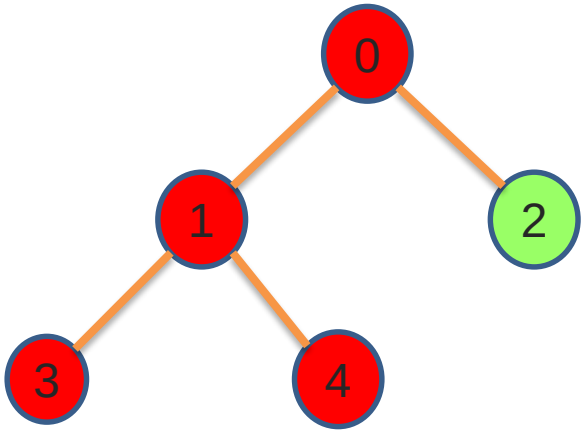
Stack

Depth First Search



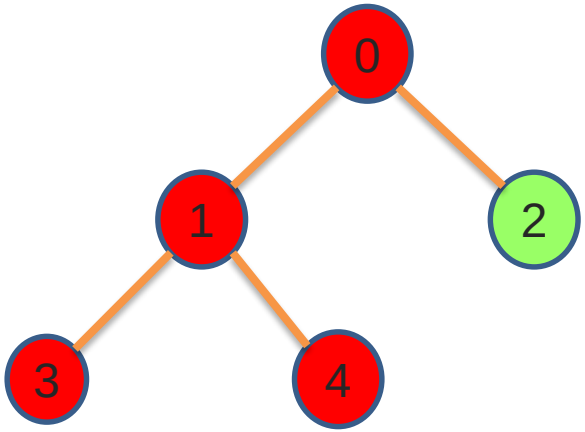
Stack

Depth First Search



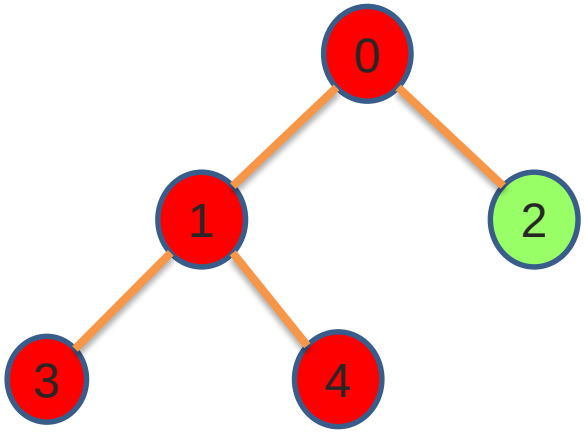
Stack

Depth First Search



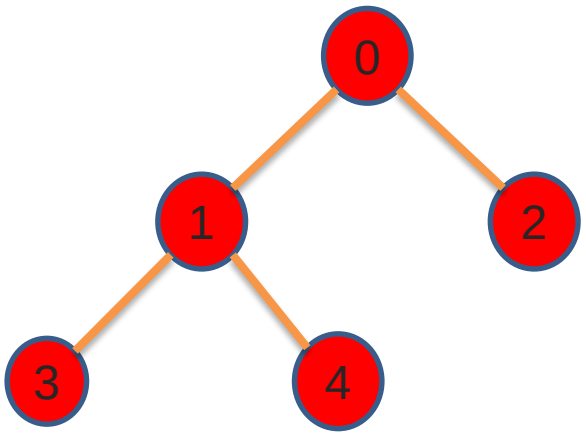
Stack

Depth First Search



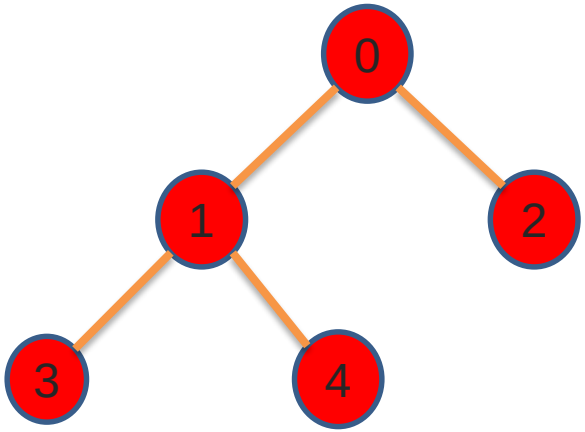
Stack

Depth First Search



Stack

Depth First Search



Stack

Depth First Search

Stack