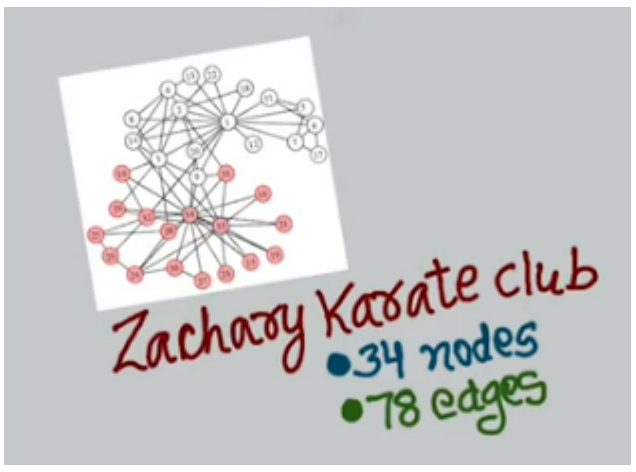
Week-2: Real World Network Datasets

In the earlier days, there wasn't much data to crunch... but now, totally contrast.. today we have data, such that *We don't know how to make sense out of it.*

For an example:

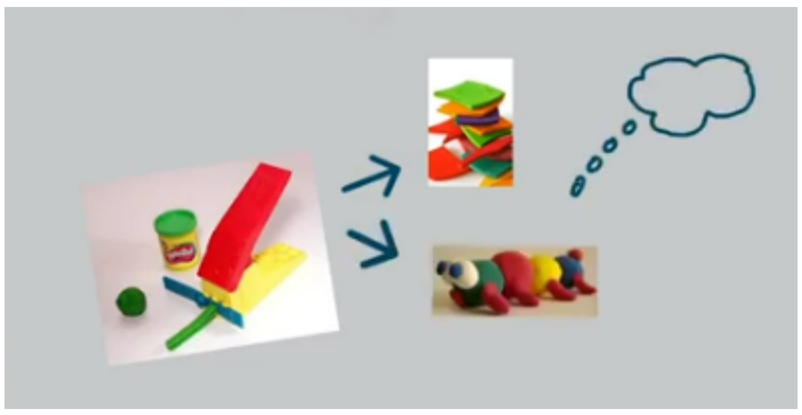
- There was a time, people looked at networks of 30-35 nodes.. popularly called as **Zachary Karate Network** It had 34 edges and 78 edges. Its a widely studied and a popular network.. 

But today, due to the advent of internet and storage facilites.. we can crunch the data -- We can crunch or not, but we have a whole lot of data..

This week..

- What are social network datasets..?
- How to make sense out of those datasets?

An analogy...

> Have you seen **Play-Dough**. The options are multitude, you can go on doing whatever you can..the limit is just is in your imagination.. 

Similarly....

When people saw a large datasets... they realized that, one can ask many questions on this data..like..

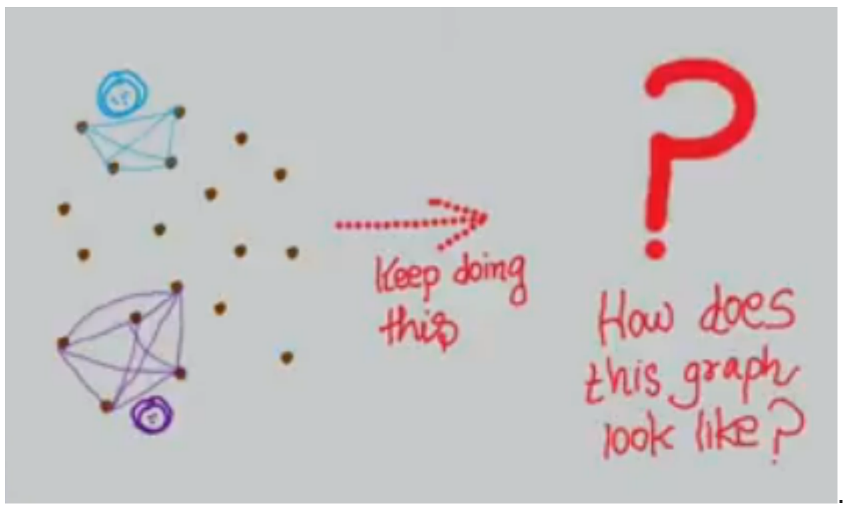
(Say) A classroom of 100 students, you have data available with you, who is friend with whom.. now..

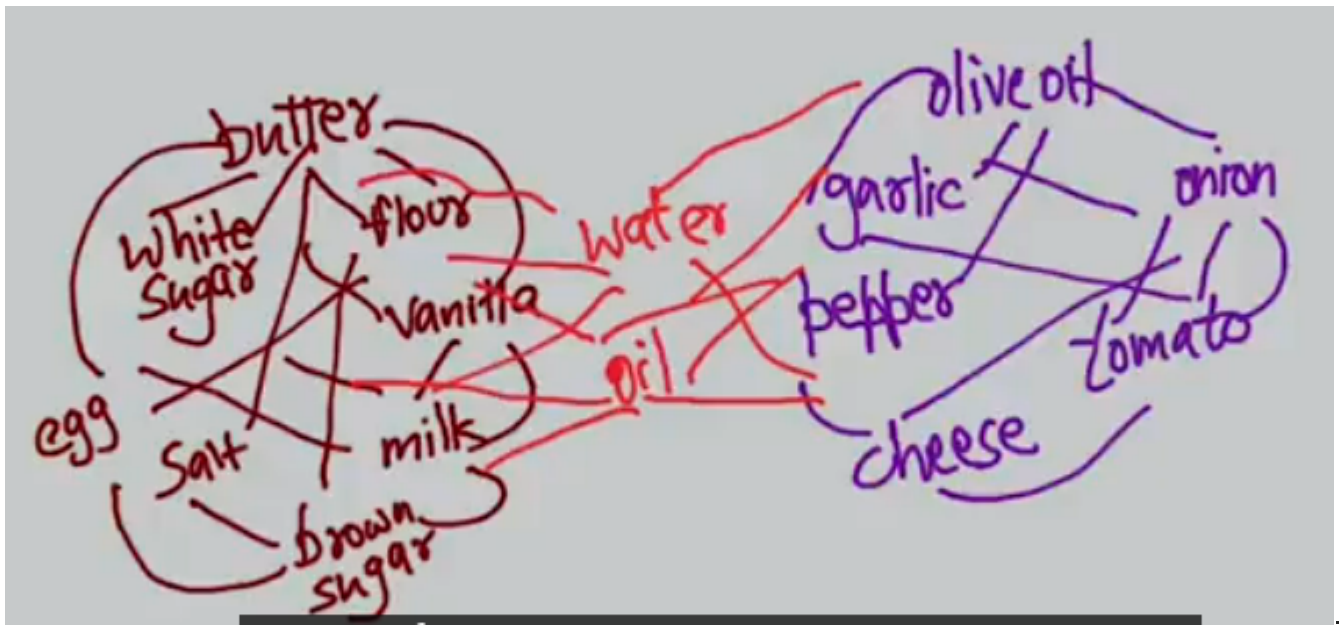
- Is the network **connected**..??
  - Most of the networks are connected..
- What is the **degree** of a person?
  - degree: How many friends, does one had..
  - Make a list of all those.. make the average, standard deviation --- ie., tells Is *Is is widely accepted or sparsely..*
- Datasets, gave this ability to ask **Random questions** -- like the Play dough..play-around and arrive at *Fantastic Results*.

Here, in the course, some 8-10 datasets are picked up and carefully studied..

1. Ingredients Dataset

(Say) In preparation of dishes, needs some ingredients..(Say for a simple dish 5-6 ingredients).

- Now, with taking all possible ingredients in the world, you put an edge between two ingredients, if they are part of a dish.
- Say, with 200 ingredients.. 

. This way, might get like any two ingredients are part of a dish..
- So, lets take only the popular ingredients. Like Salt and Sugar (Mostly used). so one can put an edge between these two. But Chilli and Sugar (not much, with an exceptions like *Chilli Chocolate*).. one or two never used together. -- So there is no edge between these two.
- Now if we take the entire big network. We'll observe **Strong communities**.. 

. i.e., some bunch of ingredients are moslty connected, but no connections across.

Why one would study this??

> - Plain intersting *(This may not be, but other..)*
>   - Community structure (Such community structures contain, a lot of hints on overall behaviour of the system)

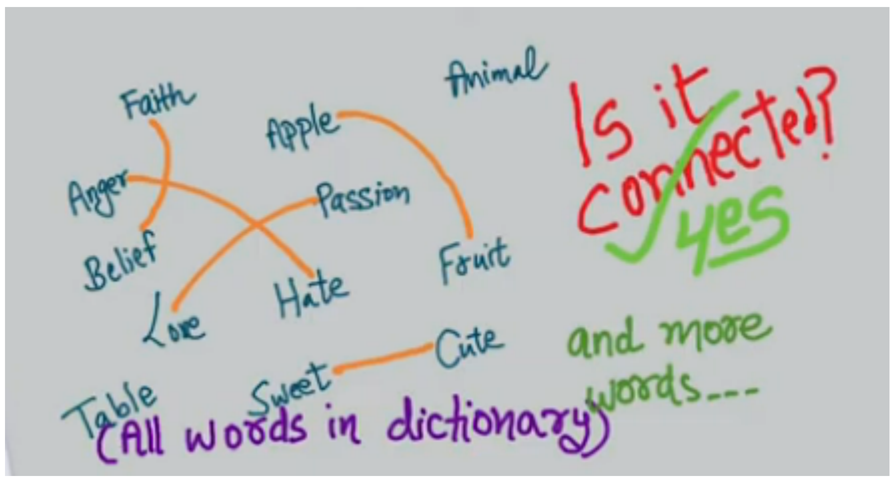
So, we'll study more such properties of networks..

2. Synonymy Dataset

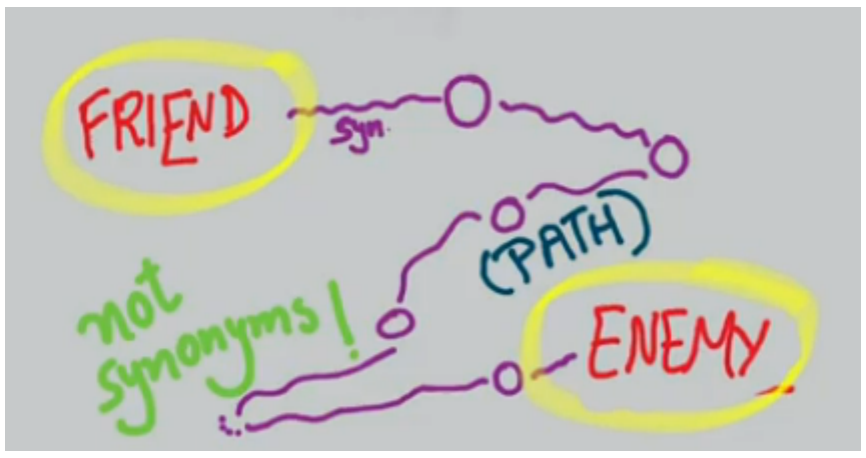
One more weird example.. let's take all the English words, and put an edge between those - if they are synonyms.

- like *Love* and *Passion*, *Bad* and *Hate*...would have an edge..

Now, the obvious question.. **Is the graph connected?**, Yes -- will get a large component..

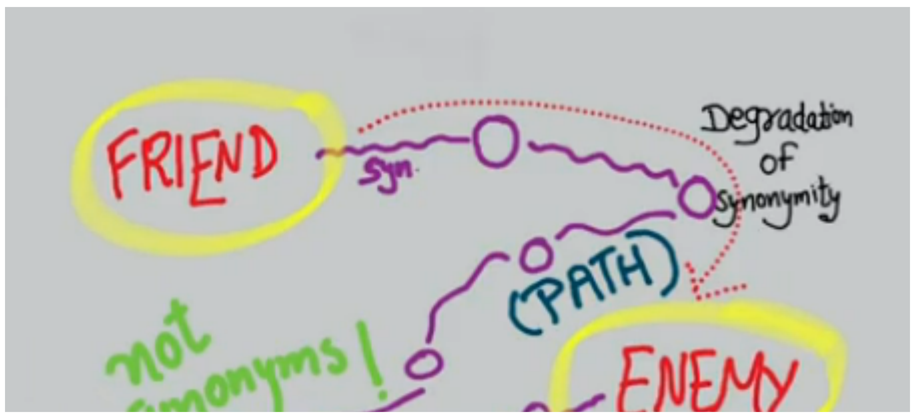


A very interesting happens here.. as the graph is connected, words like **Love** and **Hate**, **Friend** and **Enemy** *(Opposite words)* would have some path between these.. -- i.e., You move synonymously.



i.e., If A is a synonym of B, and B is a synonym of C, then C should be synonym of A -- A kind of like a *Transitive property*.. Now, Friend is synonymous to Enemy, Love is synonymous to hate..-- how wierd it is,.. A graph theoretic analysis tells such wierdity.

But, the synonym network is not actually symmetric. i.e., If A is synonymous to B, and B is synonymous to C, then one can't say as C is also synonym to A. ie., some Degradation happens..

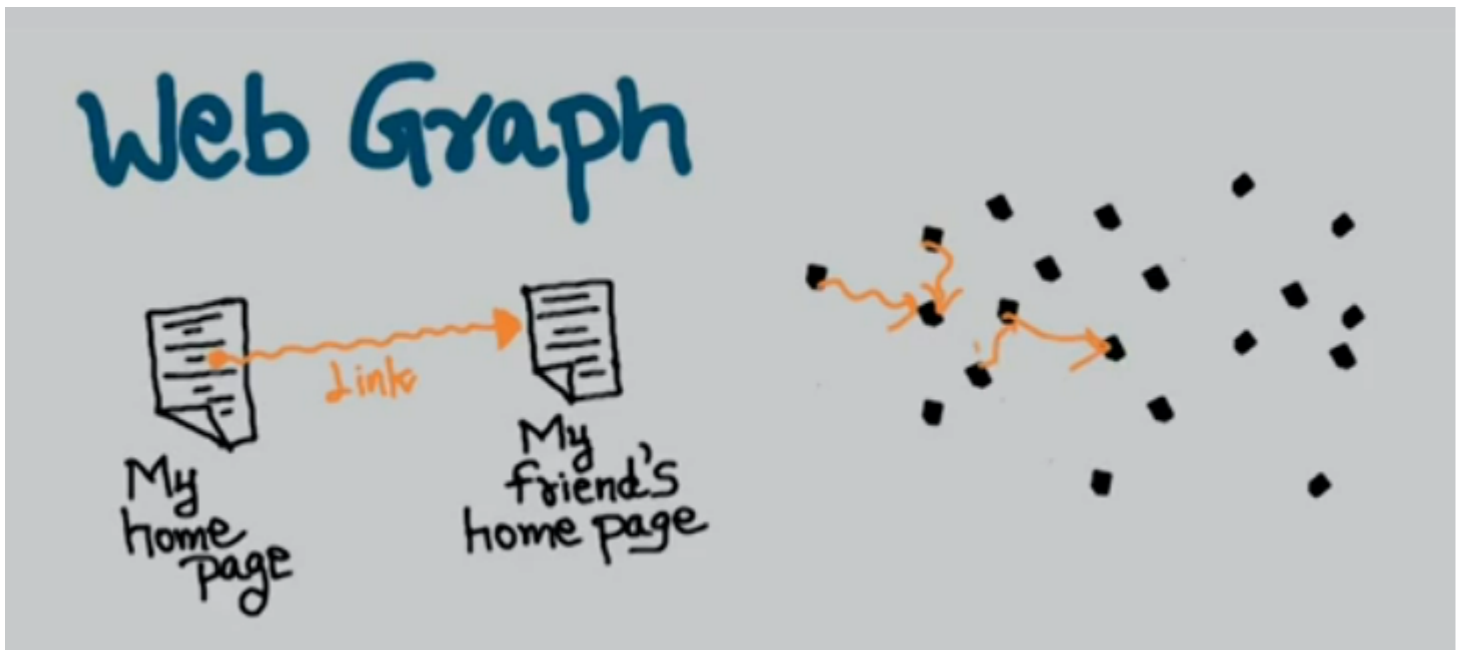


3. Web Graph - WWW

Now, yet another seeemingly useless example, but extremely useful one.

(Say), you had a website, in your homepage, you link to your friend's home page (like, for more info, refer here..). Now you put an edge between your friend's home page and your's.

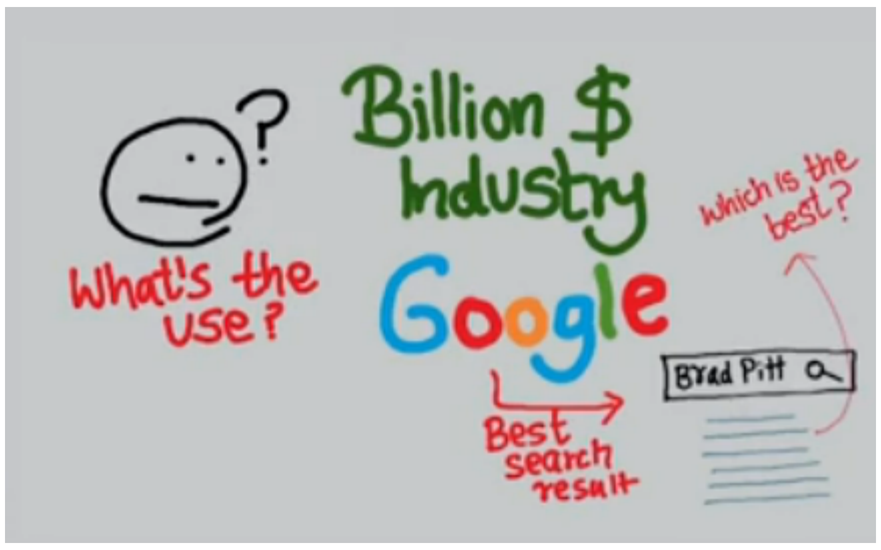
Doing it for all the pages..



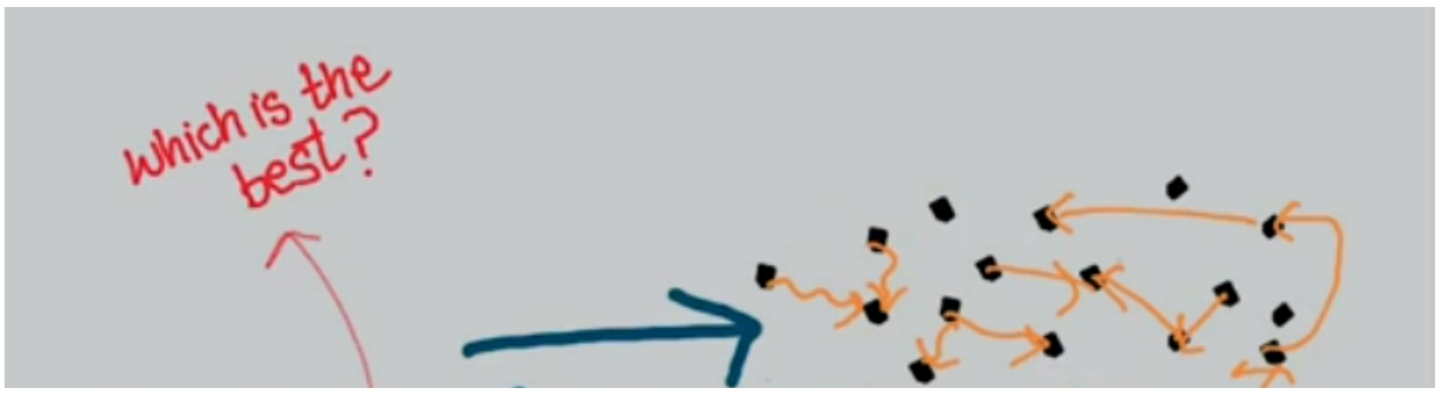
(Each page considering as node).

What's the use of such collection..?? it seems like a no use..

> But, this resulted in a Multi billion dollar network - **GOOGLE**. Google harness the fact that, deep within these connections is the clue to answer to this question: **How can one get best search results?** when typed a query..



the answer is in this web graph..

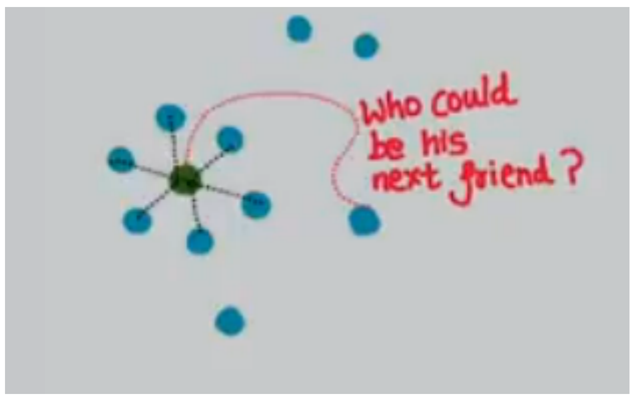


4. Social networks dataset

Now, let's look at the social network dataset.. like a Google+, FaceBook, Twitter, LinkedIn.. When one is friend with other, the other is also a fried to him -- a mutualness -- you put an edge between those.

Do this of entire friends.

Again, seemingly useless dataset.. Facebook made a good use of it, by asking a question. *(Here is a bunch of friends list)* Now, who can be his next friend

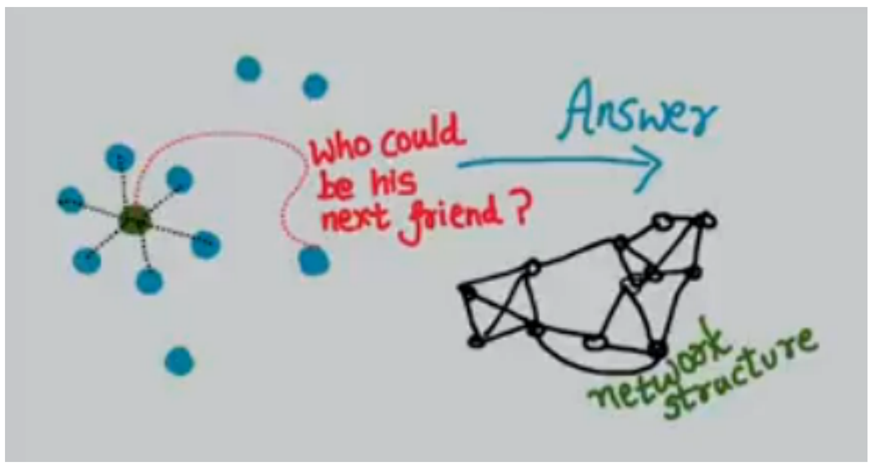


Need a pause to go further..

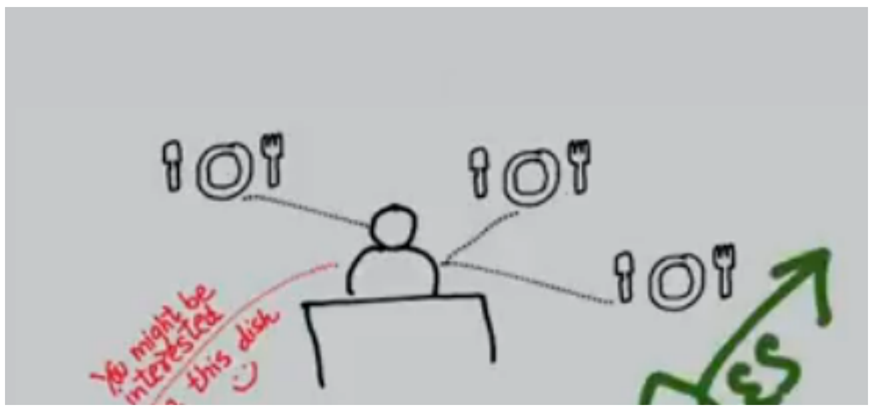
Say, one has 1000 friends in facebook, now who do you choose one *prospective friend* out of (say) 7Billion nodes.

- All those work on **numbers**. The more the number you have, the more the people glued to the screens being on their site.
- So, they want to increase the count. The can only increase the count, only if they show the prospective friend (like your childhood friend or in someone you found something interesting or whatsover reason..) .... How come facebook know about these..?

> The answer is hidden in the network structure



Like in a hotel, a food recommending system.. with a good recommendations..



/signin## Real world Network Datasets

Some examples:

- Undirected (or Bidirectional) network *(as they are mutual..)*
  - Friendship network
  - Road network
- Directed network *(as, its link from another side, it doesn't mean from other end too..)*
  - Email network: *One sending email to another.*
  - Citation network: *Research papers citing other papers.*
  - Collaboration / Co-authorship network: *Like who authored which papers, who collaborated with whom in a research..* Ofcourse, there are wide range of networks... these are some of the examples..

Network datasets formats

(Most commonly used formats)..

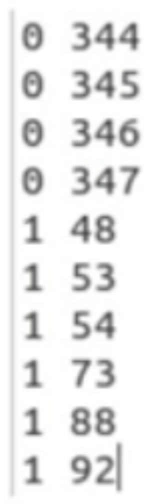
- CSV: *Comma Separated values*
- GML: *Graph Modelling Language*
- Pajek Net
- GraphML
- GEXF

CSV: Comma Separated Values

- Extension: `*.txt` or `*.csv`
- Can be in
  - Edge list format
  - Adjacency list format

Edge list format

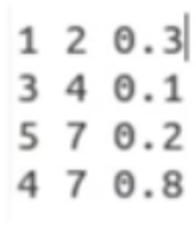
- Each row represents an edge
- First value: *Source*, next successive *Target*



```
0 344
0 345
0 346
0 347
1 48
1 53
1 54
1 73
1 88
1 92
```

Each

Can even have weights.. (the third column,,)



```
1 2 0.3
3 4 0.1
5 7 0.2
4 7 0.8
```

Adjacency List Format

- Each row tells, all the nodes adjacent to it.

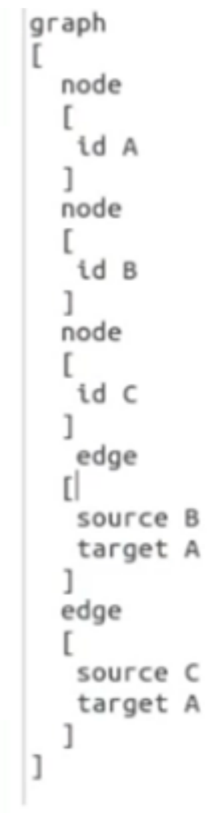
- First node represents the source node, successive ones denote the nodes, having an edge with this

Limitation

- Cannot hold **non-numeric** values.

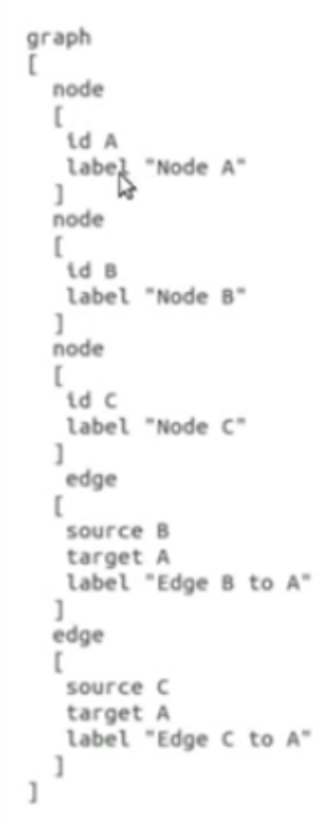
GML Format: Graph Modelling Language

- Most commonly used format.
- Starts with `graph`.
- Then list the `node`, inside it can have id, label... -- list all such nodes..
- Now, edges as `edge`



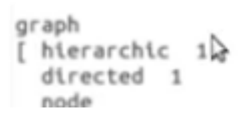
```
graph
[
  node
  [
   id A
  ]
  node
  [
   id B
  ]
  node
  [
   id C
  ]
   edge
  [
   source B
   target A
  ]
  edge
  [
   source C
   target A
  ]
]
```

GML format with labels



```
graph
[
  node
  [
   id A
   label "Node A"
  ]
  node
  [
   id B
   label "Node B"
  ]
  node
  [
   id C
   label "Node C"
  ]
   edge
  [
   source B
   target A
   label "Edge B to A"
  ]
  edge
  [
   source C
   target A
   label "Edge C to A"
  ]
]
```

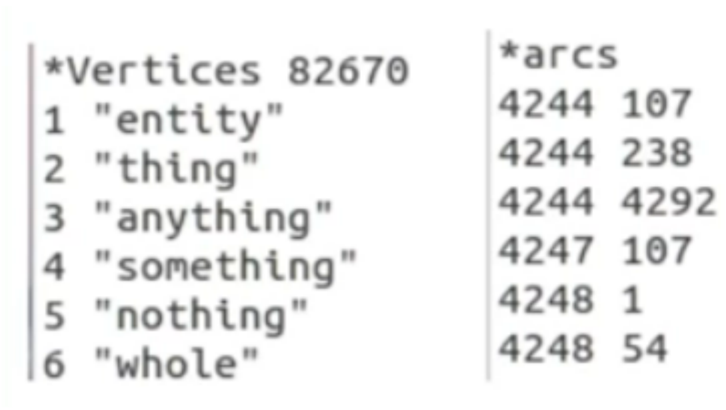
GML format with attributes Along with the node and edge properties, we can even have the graph properties.



```
graph
[ hierarchic  1
  directed  1
  node
```

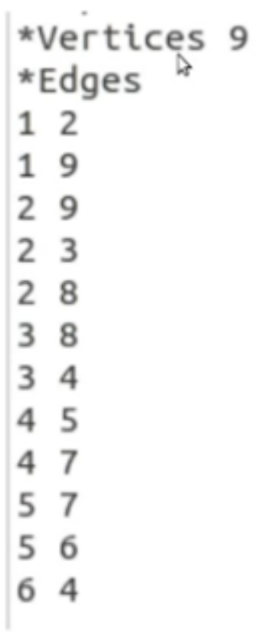
Pajek Net format: Uses `*.net` / `*.paj` extension

- Will be started with keyword `*Vertices`, followed by `count_of_vertices`
- Then in the subsequent rows, `node_no node_label` ---*list out all the nodes*-.
- Once done about the nodes, next the edges as `*arcs` or `*edges`, in the subsequent rows `source_node target_node`



```
*Vertices 82670      *arcs
1 "entity"           4244 107
2 "thing"            4244 238
3 "anything"         4244 4292
4 "something"        4247 107
5 "nothing"          4248 1
6 "whole"            4248 54
```

If had no labels to the nodes, then need not list those all.. simply can go as below..



```
*Vertices 9
*Edges
1 2
1 9
2 9
2 3
2 8
3 8
3 4
4 5
4 7
5 7
5 6
6 4
```

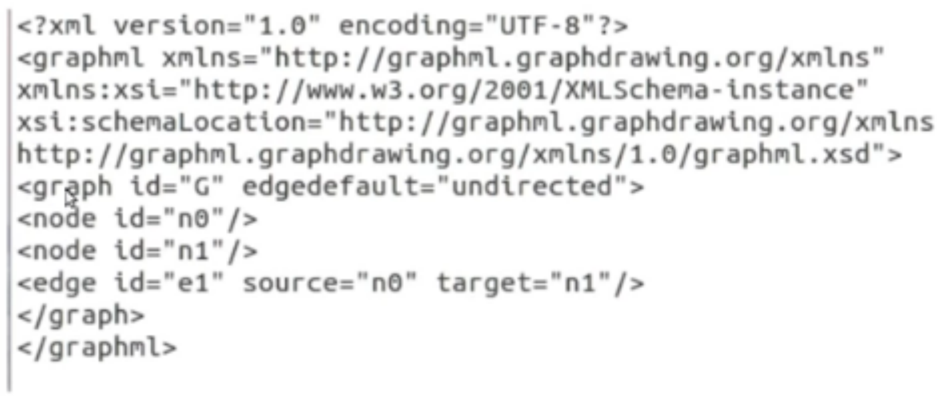
Can also give the attributes to the edges (i.e., weights) as..

.png)

GraphML format: `*.graphml` extension

Her ML used of XML format.

- First an `<?xml..` tag, next `<graphml ..`. Inside it we have `<graph ...` tag, inside which contains `<node...` tags, then after

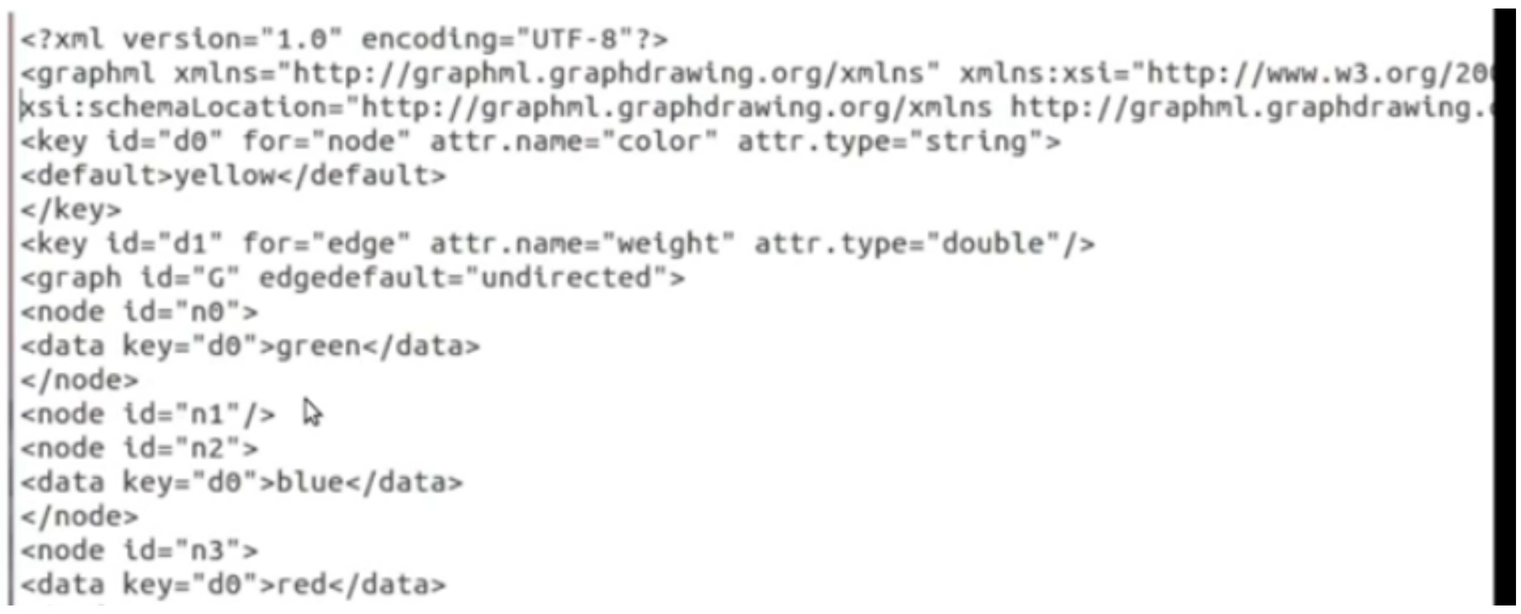


```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
<graph id="G" edgedefault="undirected">
<node id="n0"/>
<node id="n1"/>
<edge id="e1" source="n0" target="n1"/>
</graph>
</graphml>
```

  `<edge..` tags.

Structure seems complex, but the flexibility it provides in terms of attributes is better than previous.

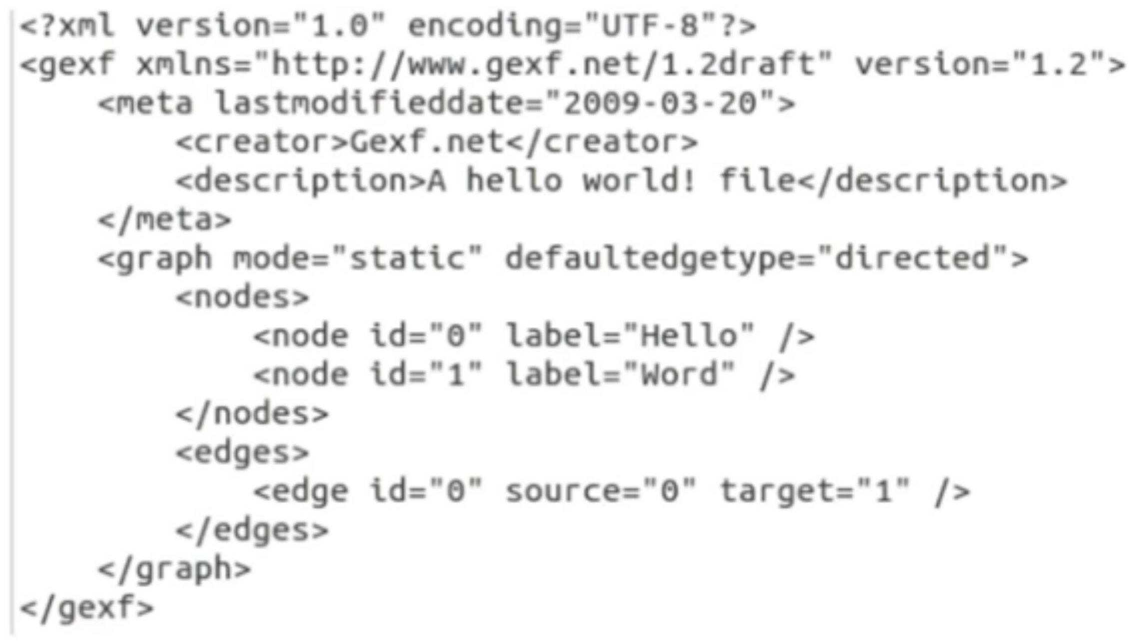
Another example: *Assigning the attributes to the nodes and edges.*



```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/20
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns http://graphml.graphdrawing.
<key id="d0" for="node" attr.name="color" attr.type="string">
<default>yellow</default>
</key>
<key id="d1" for="edge" attr.name="weight" attr.type="double"/>
<graph id="G" edgedefault="undirected">
<node id="n0">
<data key="d0">green</data>
</node>
<node id="n1"/>
<node id="n2">
<data key="d0">blue</data>
</node>
<node id="n3">
<data key="d0">red</data>
```

GEXF format: Graph Exchange XML Format

- Its created by **Gephi**, an open-source software which is used to analyze and visualize the social networks.
- Its inspired from XML.



```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.2draft" version="1.2">
    <meta lastmodifieddate="2009-03-20">
        <creator>Gexf.net</creator>
        <description>A hello world! file</description>
    </meta>
    <graph mode="static" defaultedgetype="directed">
        <nodes>
            <node id="0" label="Hello" />
            <node id="1" label="Word" />
        </nodes>
        <edges>
            <edge id="0" source="0" target="1" />
        </edges>
    </graph>
</gexf>
```

This is quite similar to the XML format, but this is much cleaner.

NetworkX provides functions, through which one can read in one format and can write in another format. Like can read in GML format and can write in GEXF format.

Downloading datasets

Some resources.

- SNAP - Stanford Large Network Dataset Collection
- The UCI Network Data Repository - maintained by University of California.
- Konect](http://konect.cc/

Datasets analyzation using NetworkX

In [2]:

```python
import networkx as nx
import matplotlib.pyplot as plt
```

Reading edgelist

In [58]:

```python
#G = nx.read_edgelist('resources/datasets/facebook_combined.txt')  # Returns the graph object..
```

Reading Pajek files

In [49]:

```python
#G = nx.read_pajek('resources/datasets/football.net')
G = nx.read_pajek('resources/datasets/ZacharyKarateClub.paj')
```

Reading GraphML files

In [57]:

```python
#G = nx.read_graphml('resources/datasets/airlines.graphml')
```

Reading GEXF files

In [56]:

```python
#G = nx.read_gexf(r'resources/datasets/eurosis/EuroSiS_Generale_Pays.gexf')
```

In [37]:
```
print (nx.info(G))
nx.is_directed(G)
```

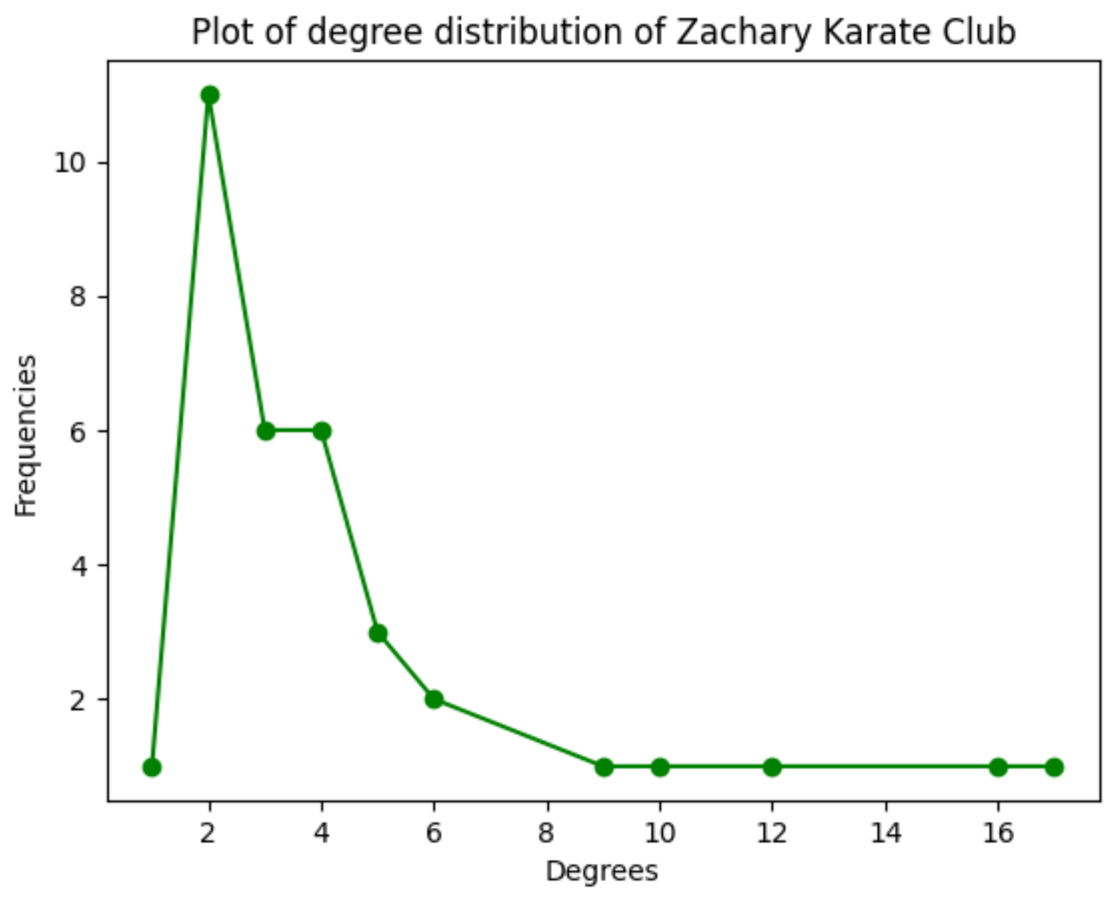
```
Graph with 4039 nodes and 88234 edges
```
Out[37]:
```
False
```

In [39]:
```
#list(set(dict(nx.degree(G)).values()))
```

In [73]:
```
def plot_degrees_distribution(G):
    degrees_contained = list(dict(nx.degree(G)).values())
    unique_degrees = list(set(degrees_contained))  #as  Converting into "Set" eliminates the duplicates right..??
    frequencies = []
    # Get the frequency of each degree.,.
    for degree in unique_degrees:
        temp = degrees_contained.count(degree)
        frequencies.append(temp)

    # Plot on a graph..
    plt.plot(unique_degrees, frequencies, 'g-o')
    # plt.loglog(unique_degrees, frequencies)  # If the graph was of Power-log distributio, we should get a straight
    plt.title("Plot of degree distribution of Zachary Karate Club")
    plt.xlabel("Degrees")
    plt.ylabel("Frequencies")
    plt.show()
```

In [ ]:
```
plt.average_clister
```

In [74]:
```
plot_degrees_distribution(G)
```



Most of the real world datasets, exhibit (**Power log distribution** *(Name might be incorrect)*) - i.e., There will be few nodes having a high degree, and a lot nodes having less degree.

Density of a graph $\frac{1}{4}$

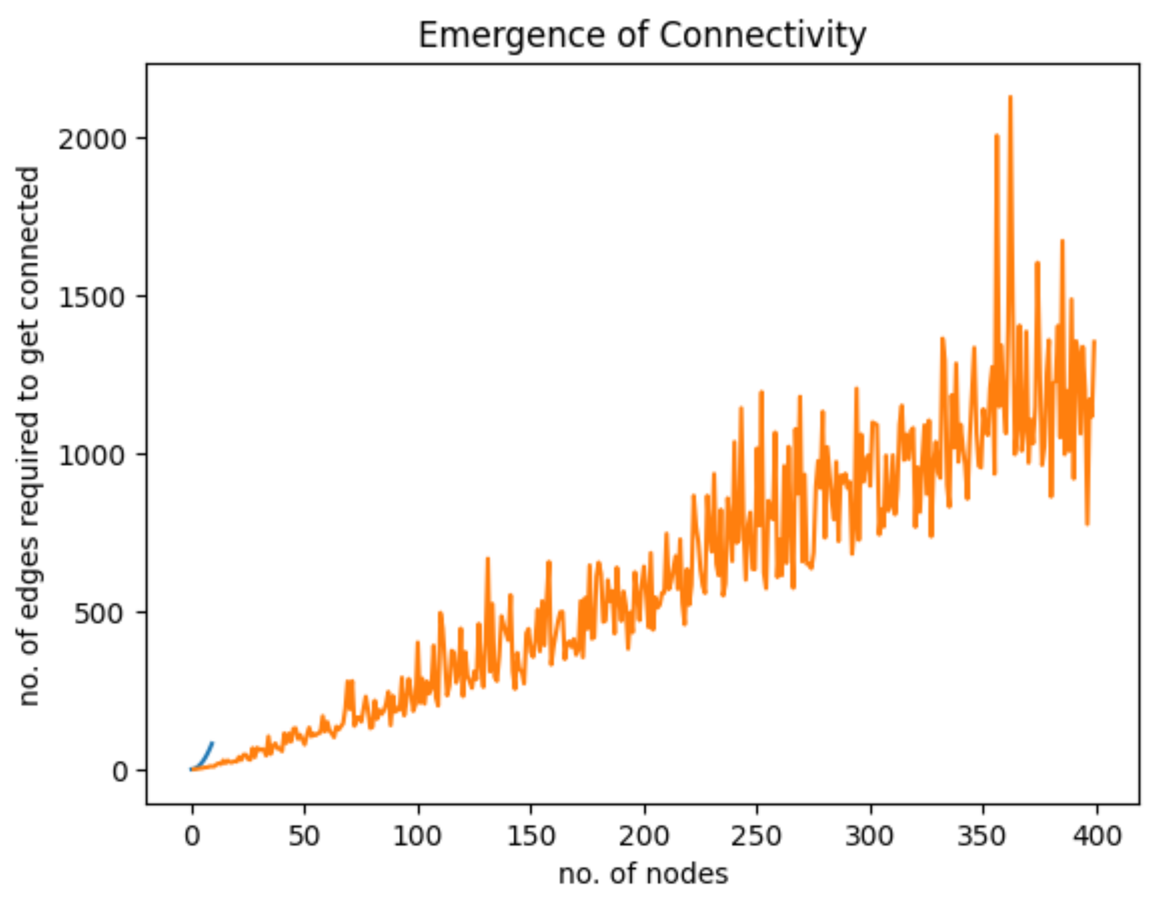
$Density = \frac{Number\ of\ edges\ present}{Total\ possessible\ edges..}$, where $Total\ possible\ edges = 2^{\binom{n}{k}}$

on September 3rd, 2021 ~ Friday

In [2]:
```
import networkx as nx

# Adding the nodes to the gfraph
def add_nodes(n):
    G = nx.Graph()
    G.add_nodes_from(range(n))
    return G
```

In [3]:
```
G = add_nodes(10)
```

In [40]:
```
G.number_of_nodes()
```
Out[40]:
```
10
```
In [10]:
```
G.number_of_edges()
```
Out[10]:
```
0
```
In [6]:
```
nx.is_connected(G)
```
Out[6]:
```
False
```
In [4]:
```
# Let's try addding the random edge in the graph..
import random
def add_random_edge(G):
    choice1 = random.choice(list(G.nodes()))
    choice2 = random.choice(list(G.nodes()))
    if choice1 != choice2:              # To avoid self-loops..
        G.add_edge(choice1, choice2)

    return G
```
In [6]:
```
G = add_random_edge(G)
nx.is_connected(G), G.number_of_edges()
```
Out[6]:
```
(False, 2)
```
In [17]:
```
# Now the activity is... keep adding the edges, until it becomes connected...
def add_till_gets_connected(G):
    while (nx.is_connected(G) == False):  # keep running as long as it is not-connected..
        G = add_random_edge(G)

    return G   # returned when got connected..
```
In [18]:
```
G = add_till_gets_connected(G)
nx.is_connected(G), G.number_of_nodes()
```
Out[18]:
```
(True, 10)
```
In [19]:
```
# Let's group together... -- it takes a input-"n", and returns the no. of edges taken for it to get connected..
def create_instance(n):
    G = add_nodes(n)
    return add_till_gets_connected(G).number_of_edges()
```
In [20]:
```
create_instance(int(input("Enter the value of n: ")))
```
Out[20]:
```
6
```
In [57]:
```
%matplotlib inline
```
In [21]:
```
# Make the plot of it..
import matplotlib.pyplot as plt
def plot_connectivity():
    x = []
    y = []
    for num_nodes in range(1, 400):
        x.append(num_nodes)
        y.append(create_instance(num_nodes))  # Getting its total required no. of edges required to get connected fo
    plt.title("Emergence of Connectivity")
    plt.xlabel("no. of nodes")
    plt.ylabel("no. of edges required to get connected")
    plt.plot(x, y)
    plt.show()
```
In [22]:
```
plot_connectivity()
```

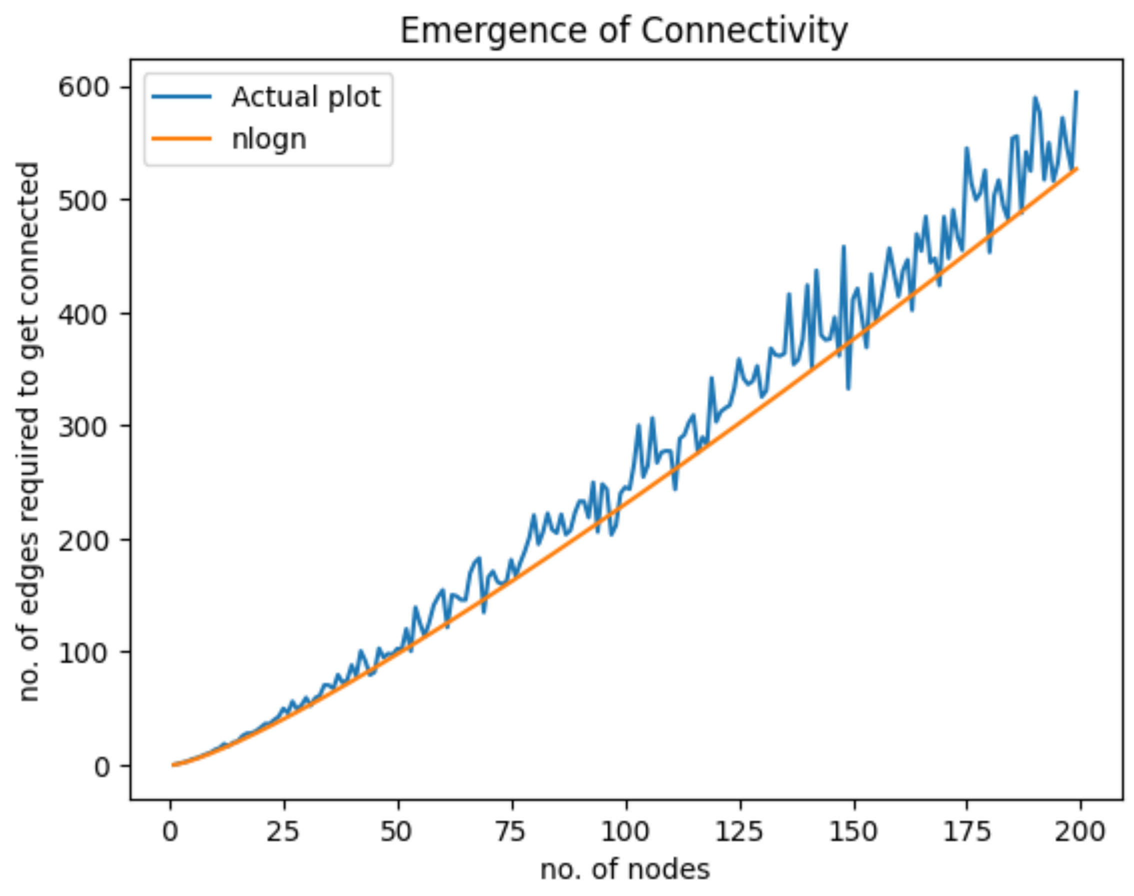


There seems much spikes in the plot, due to the randomness..

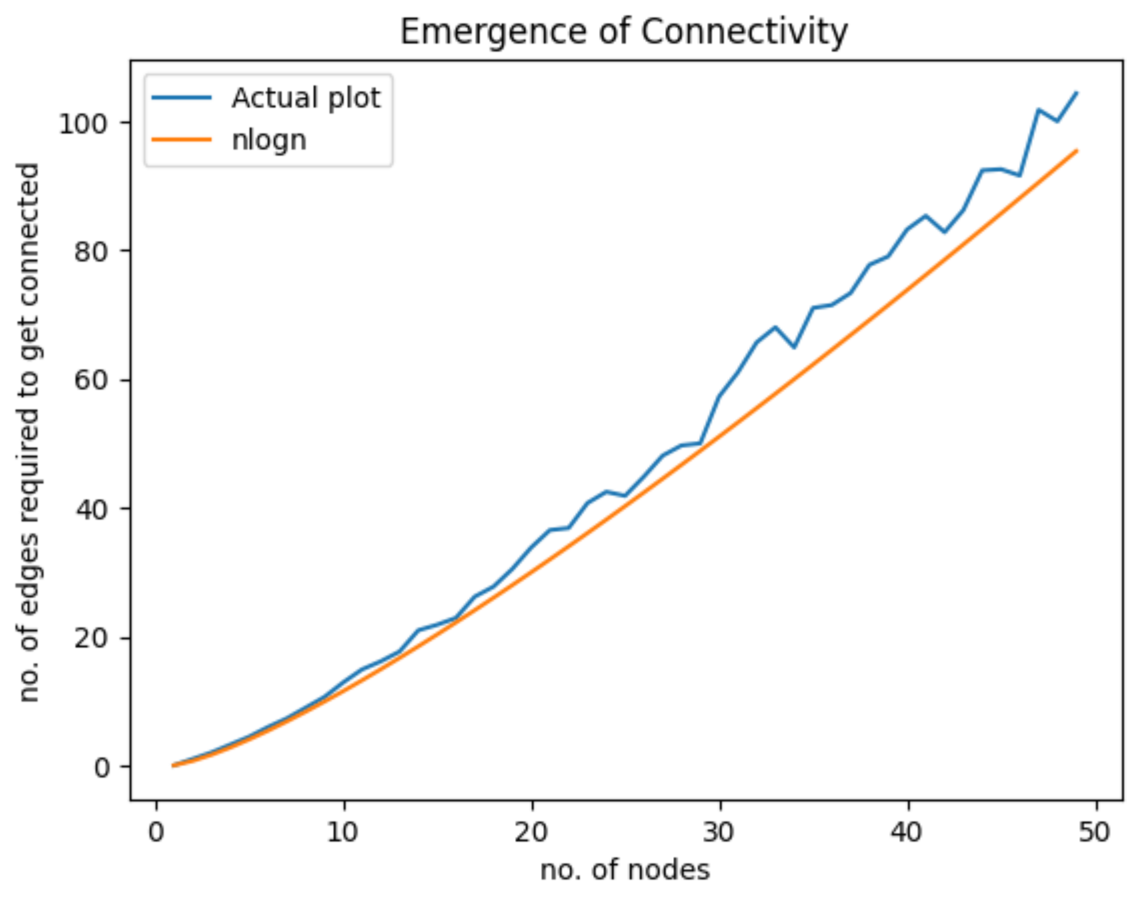
Let's try averaging for a set of no_of_nodes..

In [58]:
```
# Make the plot of it..
import numpy as np
import matplotlib.pyplot as plt
def plot_connectivity_updated(test_size=100, batch_size=10):
    x, x1 = [], []
    y, y1 = [], []
    for num_nodes in range(1, test_size):
        x.append(num_nodes)
        temp = []
        for i in range(batch_size):
            temp.append(create_instance(num_nodes))
            #num_nodes += 1
        y.append(sum(temp)/batch_size)  # Getting its total required no. of edges required to get connected for the

        x1.append(num_nodes)
        y1.append(num_nodes*(np.log(num_nodes)/2))# As the ORDER of nlogn doesn't affect with the constants, as the

    plt.title("Emergence of Connectivity")
    plt.xlabel("no. of nodes")
    plt.ylabel("no. of edges required to get connected")
    plt.plot(x, y)
    # For the comparison with the nlogn..
    plt.plot(x1, y1)

    plt.legend(["Actual plot", "nlogn"])
    plt.show()
```

In [57]:
```
plot_connectivity_updated(test_size=200, batch_size=10)
```



```
In [56]: plot_connectivity_updated(test_size=50, batch_size=50)
```



See the proximity between the plots.... almost nearer

```
In [14]: a = [a for a in range(10)]
         plt.plot(a, [b*b for b in range(10)])

Out[14]: [<matplotlib.lines.Line2D at 0x7febad8e2790>]
```

```
In [15]: sum(a)

Out[15]: 45
```

```
In [16]: plt.plot(x, num_nodes*np.log(num_nodes))
         plt.show()

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-c898de1027d7> in <module>
----> 1 plt.plot(x, num_nodes*np.log(num_nodes))
      2 plt.show()

NameError: name 'x' is not defined
```

```
In [ ]:
```

```
In [ ]:
```

Summary to the week-2

We've seen:

- Introduction to the datasets
- Why datasets?
- Making inferences from datasets, via python, networkx..
- Graph visualization -- like Gephi
- Emergence of Connectedness.
  - You start with a graph with no edges, you start putting the edge each time.. at one point, the graph becomes connected --- that happens with $nlog(n)$ no. of edges.
  - Proven mathematically and via program.
- Power of Synthetic datasets
  - Synthetic means.... Say if had 2000 nodes,.. (some big dataset), and you want to know the pattern in that. -- here you would like to create your own network *(as we have done for the Emergence of connectedness)*.

In a nutshell.. seen about..

- Learnt about datasets
- Emergence of connectedness
- Power of synthetic datasets.

Coming up next....

- Google page rank..
- and some more as discussed in the motivational pushs in these 2 weeks..

```
In [ ]:
```