# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

## 15. Page Rank Algorithm by Base Line Method:

**1- Dummy Internet Page Rank**

PageRank revolutionized web search by generating a ranked list of web pages based on the underlying connectivity of the web. The PageRank algorithm is based on an ideal random web surfer who, when reaching a page, goes to the next page by clicking on a link. The surfer has equal probability of clicking any link on the page and, when reaching a page with no links, has equal probability of moving to any other page by typing in its URL. In addition, the surfer may occasionally choose to type in a random URL instead of following the links on a page. The PageRank is the ranked order of the pages from the most to the least probable page the surfer will be viewing.

**Code:**

```
%pylab notebook
import numpy as np
import numpy.linalg as la
np.set_printoptions(suppress=True)
```

### PageRank as a linear algebra problem

Let's imagine a micro-internet, with just 6 websites (**A**vocado, **B**ullseye, **C**atBabel, **D**romeda, **e**Tings, and **F**aceSpace). Each website links to some of the others, and this forms a network as shown,

The design principle of PageRank is that important websites will be linked to by important websites. This somewhat recursive principle will form the basis of our thinking.

Imagine we have 100 *Procrastinating Pat*s on our micro-internet, each viewing a single website at a time. Each minute the Pats follow a link on their website to another site on the micro-internet. After a while, the websites that are most linked to will have more Pats visiting them, and in the long run, each minute for every Pat that leaves a website, another will enter keeping the total numbers of Pats on each website constant. The PageRank is simply the ranking of websites by how many Pats they have on them at the end of this process. We represent the number of Pats on each website with the vector,

$$\mathbf{r} = \begin{bmatrix} r_A \\ r_B \\ r_C \\ r_D \\ r_E \\ r_F \end{bmatrix}$$

And say that the number of Pats on each website in minute $i$+1i+1 is related to those at minute $i$i by the matrix transformation

$$\mathbf{r}_{(i+1)} = L\mathbf{r}_{(i)} \text{r(i+1)=Lr(i)}$$

with the matrix $L$L taking the form,

$$L = \begin{bmatrix} L_{A\to A} & L_{B\to A} & L_{C\to A} & L_{D\to A} & L_{E\to A} & L_{F\to A} \\ L_{A\to B} & L_{B\to B} & L_{C\to B} & L_{D\to B} & L_{E\to B} & L_{F\to B} \\ L_{A\to C} & L_{B\to C} & L_{C\to C} & L_{D\to C} & L_{E\to C} & L_{F\to C} \\ L_{A\to D} & L_{B\to D} & L_{C\to D} & L_{D\to D} & L_{E\to D} & L_{F\to D} \\ L_{A\to E} & L_{B\to E} & L_{C\to E} & L_{D\to E} & L_{E\to E} & L_{F\to E} \\ L_{A\to F} & L_{B\to F} & L_{C\to F} & L_{D\to F} & L_{E\to F} & L_{F\to F} \end{bmatrix}$$

where the columns represent the probability of leaving a website for any other website and sum to one. The rows determine how likely you are to enter a website from any other, though these need not add to one. The long time behaviour of this system is when $\mathbf{r}_{(i+1)}=\mathbf{r}_{(i)}$r(i+1)=r(i), so we'll drop the superscripts here, and that allows us to write,

$$L\mathbf{r}=\mathbf{r} \text{Lr=r}$$

which is an eigenvalue equation for the matrix $L$L, with eigenvalue 1 (this is guaranteed by the probabalistic structure of the matrix $L$L).

```
L = np.array([[0,   1/2, 1/3, 0, 0,   0 ],
              [1/3, 0,   0,   0, 1/2, 0 ],
              [1/3, 1/2, 0,   1, 0,   1/2 ],
              [1/3, 0,   1/3, 0, 1/2, 1/2 ],
              [0,   0,   0,   0, 0,   0 ],
              [0,   0,   1/3, 0, 0,   0 ]])
```

In principle, we could use a linear algebra library, as below, to calculate the eigenvalues and vectors. And this would work for a small system. But this gets unmanagable for large systems. And since we only care about the principal eigenvector (the one with the largest eigenvalue, which will be 1 in this case), we can use the power iteration method which will scale better, and is faster for large systems.

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

**Use the code below to peek at the PageRank for this micro-internet.**

```
eVals, eVecs = la.eig(L) # Gets the eigenvalues and vectors
 order = np.absolute(eVals).argsort()[::-1]
# Orders them by their eigenvalues
 eVals = eVals[order] eVecs = eVecs[:,order]


r = eVecs[:, 0] # Sets r to be the principal eigenvector  100 * np.real(r / np.sum(r))
 # Make this eigenvector sum to one, then multiply by 100 Procrastinating Pats

 array([16.,  5.33333333, 40., 25.33333333,  0,13.33333333])
```

Let's now try to get the same result using the Power-Iteration method that was covered in the video.

This method will be much better at dealing with large systems.

First let's set up our initial vector, $\mathbf{r}^ {(0)}$, so that we have our 100 Procrastinating Pats equally distributed on each of our 6 websites.

```
r = 100 * np.ones(6) / 6 # Sets up this vector (6 entries of 1/6 × 100 each)
 r # Shows it's value
array([16.66666667, 16.66666667, 16.66666667, 16.66666667, 16.66666667, 16.66666667])
```

Next, let's update the vector to the next minute, with the matrix $L$L. Run it in loop to stabilise it.

```
for i in np.arange(100) : # Repeat 100 times
            r = L @ r
r
array([16. ,  5.33333333, 40., 25.33333333,  0,13.33333333])
```

Or even better, we can keep running until we get to the required tolerance.

```
r = 100 * np.ones(6) / 6 # Sets up this vector (6 entries of 1/6 × 100 each) lastR = r r = L @ r i = 0
while la.norm(lastR - r) > 0.01 :
      lastR = r     r = L @ r     i += 1 print(str(i) + " iterations to convergence.")
 r
array([16.00149917,5.33252025, 39.99916911, 25.3324738 ,  0. ,  13.33433767])
```

**Damping Parameter:**

The system we just studied converged fairly quickly to the correct answer. Let's consider an extension to our micro-intern et where things start to go wrong.

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

Say a new website is added to the micro-internet: Geoff's Website. This website is linked to by Face Space and only links to itself.

Intuitively, only Face Space, which is in the bottom half of the page rank, links to this website amongst the two others it links to, so we might expect Geoff's site to have a correspondingly low PageRank score.

Build the new $L$ matrix for the expanded micro-internet, and use Power-Iteration on the Procrastinating Pat vector. See what happens…

```
# We'll call this one L2, to distinguish it from the previous L.
L2 = np.array([[0,   1/2, 1/3, 0, 0,   0, 0 ],
               [1/3, 0,   0,   0, 1/2, 0, 0 ],
               [1/3, 1/2, 0,   1, 0,   0, 0 ],
               [1/3, 0,   1/3, 0, 1/2, 0, 0 ],
               [0,   0,   0,   0, 0,   0, 0 ],
               [0,   0,   1/3, 0, 0,   1, 0 ],
               [0,   0,   0,   0, 0,   0, 1 ]])
r = 100 * np.ones(7) / 7 # Sets up this vector (7 entries of 1/7 × 100 each)
lastR = r
r = L2 @ r
 i = 0
 while la.norm(lastR - r) > 0.01 :
   lastR = r     r = L2 @ r
   i += 1
print(str(i) + " iterations to convergence.")
 r
46 iterations to convergence.
array([ 0.01077429,  0.00420324,  0.02131321,  0.01251789,  0.        ,
       85.66547709, 14.28571429])
```

That's no good! Geoff seems to be taking all the traffic on the micro-internet, and somehow coming at the top of the Page Rank. This behaviour can be understood, because once a Pat get's to Geoff's Website, they can't leave, as all links head back to Geoff.

To combat this, we can add a small probability that the Procrastinating Pats don't follow any link on a webpage, but instead visit a website on the micro-internet at random. We'll say the probability of them following a link is $d$ and the probability of choosing a random website is therefore $1-d$. We can use a new matrix to work out where the Pat's visit each minute.

$$M = dL +$$

$(1-d/n)J$ where $J$ is an $n \times n$ matrix where every element is one.

If $d$ is one, we have the case we had previously, whereas if $d$ is zero, we will always visit a random webpage and therefor e all webpages will be equally likely and equally ranked. For this extension to work best, $1-d$ should be somewhat small - though we won't go into a discussion about exactly how small.

Let's retry this PageRank with this extension.

```
d = 0.5 # Feel free to play with this parameter after running the code once.
M = d * L2 + (1-d)/7 * np.ones([7, 7]) # np.ones() is the J matrix, with ones for each entry.
r = 100 * np.ones(7) / 7 # Sets up this vector (6 entries of 1/6 × 100 each)
lastR = r r = M @ r
i = 0
while la.norm(lastR - r) > 0.01 :
    lastR = r
    r = M @ r
   i += 1
 print(str(i) + " iterations to convergence.")
r
```

```
array([13.13619674, 11.11812027, 19.27885503, 14.33173875,  7.14285714, 20.70651779,
14.28571429])
```

**2. Sub Internet Page Ranking**

In this assessment, you will be asked to produce a function that can calculate the PageRank for an arbitrarily large probability matrix.

```
# Complete this function to provide the PageRank for an arbitrarily sized internet.

# I.e. the principal eigenvector of the damped system, using the power iteration method.

# (Normalisation doesn't matter here)

# The functions inputs are the linkMatrix, and d the damping parameter - as defined in this

worksheet. def pageRank(linkMatrix, d) :     n = linkMatrix.shape[0]

  M = d * linkMatrix + (1-d)/n * np.ones([n, n])
   r = 100 *  np.ones(n) / n # Sets up this vector (n entries of 1/n × 100 each)
   last = r

    r = M @ r
   while la.norm(last - r) >0.01 :
         last = r

r = M @ r

return r

def generate_internet(n) :
  c = np.full([n,n], np.arange(n))
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

```
    c = (abs(np.random.standard_cauchy([n,n])/2) > (np.abs(c - c.T) + 1)) + 0
    c = (c+1e-10) / np.sum((c+1e-10), axis=0)
    return c
# Use the following function to generate internets of different sizes.
generate_internet(5)
array([[0. , 0. , 0. , 0.2, 0. ],       [0. , 0. , 0. , 0.2, 0. ],
       [1. , 1. , 0. , 0.2, 1. ],
       [0. , 0. , 0. , 0.2, 0. ],
       [0. , 0. , 1. , 0.2, 0. ]])
      # Test your PageRank method against the built in "eig" method.

      # You should see yours is a lot faster for large internets L =

      generate_internet(100) pageRank(L, 1) array([ 0.0021761 ,

      0.00123391, 0.00068895, 0.00166299, 0.00398759,

            0.00733913, 7.86065101, 0.01062761, 0.00104149,
      10.83302715,

            0.00050824, 0.00050824, 0.00068895, 0.00224713,
      0.00305282,

      # Do note, this is calculating the eigenvalues of the link matrix,
      L,

      # without any damping. It may give different results that your pageRank function.

      # If you wish, you could modify this cell to include damping.

      eVals, eVecs = la.eig(L) # Gets the eigenvalues and vectors order

      = np.absolute(eVals).argsort()[::-1] # Orders them by their

      eigenvalues eVals = eVals[order] eVecs = eVecs[:,order]


      r = eVecs[:, 0]

      100 * np.real(r / np.sum(r)) array([ 0.00000014,  0.00000009,

      0.00000005,  0.00000011,  0.00000023,            0.00000043,

      7.88515126, 0.00000061, 0.00000008, 10.86710098,

            0.00000004, 0.00000004, 0.00000005, 0.00000015,
      0.00000018,
```

# VNRVJIET

**Name of the Laboratory:**

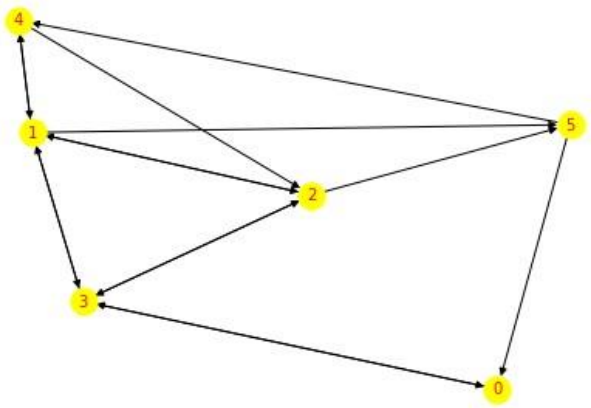
**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

## Page Rank Algorithm using Libraries:

**Code:**

```
import networkx as
nx import numpy as
np import pandas as
pd import
matplotlib.pyplot as
plt import operator
import random as rd


# created a directed graph
graph=nx.gnp_random_graph(6,0.6,directed=True)
#draw a graph
nx.draw(graph,with_labels=True,font_color='red',font_size=10,node_color='yellow')
```



# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

```
#number of nodes for graph
count=graph.number_of_nodes()
print(count)
#graph neighbours of a node 1
print(list(graph.neighbors(1)))
6
[2, 3, 4, 5]
#Page rank by networkx library pagerank=nx.pagerank(graph) #sorting
both dictionaries based on items
pagerank_sorted=sorted(pagerank.items(),key=lambda
v:(v[1],v[0]),reverse=True)
print("\n\nThe order generated by networkx library is\n")
for i in pagerank_sorted:
    print(i[0],end=" ")
The order generated by networkx library is
3 1 2 0 5 4
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

## 16. Web Scrapping with Wikipedia

It is basically a technique or a process in which large amounts of data from a huge number of websites is passed through a web scraping software coded in a programming language and as a result, structured data is extracted which can be saved locally in our devices preferably in Excel sheets, JSON or spreadsheets. Now, we don't have to manually copy and paste data from websites but a scraper can perform that task for us in a couple of seconds.

The first thing we'll need to do to scrape a web page is to download the page. We can download pages using the Python requests library. The *requests* library will make a GET request to a web server, which will download the HTML contents of a given web page for us. There are several types of requests we can make using requests, of which GET is just one. The URL of our sample website is https://en.wikipedia.org/wiki/Main_Page. The task is to download it using *requests.get()* method. After running our request, we get a Response object. This object has a *status_code* property, which indicates if the page was downloaded successfully. And a content property that gives the HTML content of the webpage as output.

```
!pip install virtualenv
!python -m pip install selenium
!python -m pip install requests
!python -m pip install urllib3
```

```
import requests
page = requests.get("https://en.wikipedia.org/wiki/Main_Page")
print(page.status_code)
 print(page.content)
```

```
200 b'<!DOCTYPE html>\n<html class="client-nojs" lang="en" dir="ltr">\n<head>\n<meta charset="UTF-
8"/>\n<title>Wikipedia, the free
encyclopedia</title>\n<script>document.documentElement.className="clientjs";RLCONF={"wgBreakFra
mes":false,"wgSeparatorTransformTable":["",""],"wgDigitTr
ansformTable":["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":["","January","
February","March","April","May","June","July","August","September","October","No
vember","December"],"wgRequestId":"3e2257bb-c492-4fd0-9894-
35ad6b7197f3","wgCSPNonce":false,"wgCanonicalNamespace":"","wgCanonicalSpecialPa
geName":false,"wgNamespaceNumber":0,"wgPageName":"Main_Page","wgTitle":"Main
Page","wgCurRevisionId":1114291180,"wgRevisionId":1114291180,"wgArticleId":15580
374,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUserName":null,
"wgUserGroups":["*"],"wgCategories":[],"wgPageContentLanguage":"en"
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

We can use the *BeautifulSoup* library to parse this document and extract the text from the *p* tag. We first have to import the library and create an instance of the *BeautifulSoup* class to parse our document. We can now print out the HTML content of the page, formatted nicely, using the prettify method on the *BeautifulSoup* object. As all the tags are nested, we can move through the structure one level at a time. We can first select all the elements at the top level of the page using the children's property of soup. Note that children return a list generator, so we need to call the list function on it.

```
from bs4 import BeautifulSoup
import requests
page = requests.get("https://en.wikipedia.org/wiki/Main_Page") soup =
BeautifulSoup(page.content, 'html.parser') print(soup.prettify())
<!DOCTYPE html>
<html class="client-nojs" dir="ltr" lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   Wikipedia, the free encyclopedia
  </title>
<script>
   document.documentElement.className="client-
js";RLCONF={"wgBreakFrames":false,"wgSeparatorTransformTable":["",""],"wgDigitTr
ansformTable":["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":["","January","
February","March","April","May","June","July","August","September","October","No
vember","December"],"wgRequestId":"3e2257bb-c492-4fd0-9894-
35ad6b7197f3","wgCSPNonce":false,"wgCanonicalNamespace":"","wgCanonicalSpecialPa
geName":false,"wgNamespaceNumber":0,"wgPageName":"Main_Page","wgTitle":"Main
Page","wgCurRevisionId":1114291180,"wgRevisionId":1114291180,"wgArticleId":15580
374,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUserName":null,
"wgUserGroups":["*"],"wgCategories":[],"wgPageContentLanguage":"en","wgPageConte
ntModel":"wikitext","wgRelevantPageName":"Main_Page","wgRelevantArticleId":15580
374,"wgIsProbablyEditable":false,"wgRelevantPageIsProbablyEditable":false,"wgRes
trictionEdit":["sysop"],"wgRestrictionMove":["sysop"],"wgIsMainPage":true,"wgFla ggedRevsParams":{
"tags":{"status":{"levels":1}}},"wgVisualEditor":{"pageLanguageCode":"en","pageL
anguageDir":"ltr","pageVariantFallbacks":"en"},"wgMFDisplayWikibaseDescriptions"
:{"search":true,"watchlist":true,"tagline":false,"nearby":true},"wgWMESchemaEdit
AttemptStepOversample":false,"wgWMEPageLength":3000,"wgNoticeProject":"wikipedia
","wgVector2022PreviewPages":[],"wgMediaViewerOnClick":true,"wgMediaViewerEnable
dByDefault":true,"wgPopupsFlags":10,"wgULSCurrentAutonym":"English","wgEditSubmi
tButtonLabelPublish":true,"wgCentralAuthMobileDomain":false,"wgULSPosition":"int
erlanguage","wgULSisCompactLinksEnabled":true,"wgWikibaseItemId":"Q5296","GEHome
pageSuggestedEditsEnableTopics":true,"wgGETopicsMatchModeEnabled":false,"wgGEStr
ucturedTaskRejectionReasonTextInputEnabled":false};RLSTATE={"ext.globalCssJs.use
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

r.styles":"ready","site.styles":"ready","user.styles":"ready","ext.globalCssJs.u
ser":"ready","user":"ready","user.options":"loading","skins.vector.styles.legacy ":"ready",
"ext.visualEditor.desktopArticleTarget.noscript":"ready","ext.wikimediaBadges":"
ready","ext.uls.interlanguage":"ready"};RLPAGEMODULES=["site","mediawiki.page.re
ady","skins.vector.legacy.js","mmv.head","mmv.bootstrap.autostart","ext.visualEd
itor.desktopArticleTarget.init","ext.visualEditor.targetLoader","ext.eventLoggin
g","ext.wikimediaEvents","ext.navigationTiming","ext.cx.eventlogging.campaigns",
"ext.centralNotice.geoIP","ext.centralNotice.startUp","ext.gadget.ReferenceToolt
ips","ext.gadget.charinsert","ext.gadget.extra-toolbar-
buttons","ext.gadget.switcher","ext.centralauth.centralautologin","ext.popups","
ext.echo.centralauth","ext.uls.interface","ext.growthExperiments.SuggestedEditSe ssion"];
</script>

## 17. Data Extraction from YouTube

Web scraping is a way to extract the data from websites. It helps us to gather or copy the specific data and we can store the date into the database or spread sheet for the later analysis or retrieval.

Python comes with the Beautiful Soup library which is widely used to scrap data from other websites. we will discuss how to extract data from YouTube and get the useful insights from it. We will use the Beautiful Soup library and HTML parser.

Since YouTube is a largest video-sharing platform has billions users across the world. Many creators are making million. It would be good idea to fetch the information of popular channels. We can keep track some popular channels, subscribers, views on videos, likes and dislikes.

```
# Install dependencies   from requests_html
import HTMLSession
from bs4 import BeautifulSoup as bs # importing BeautifulSoup


# copy youtube url from YouTube website   video_url =
"https://youtu.be/u4N45v8f7cY"
# init an HTML Session   session =
HTMLSession()   # get the html
content   response =
session.get(video_url)
# execute Javascript, timeout is necessary pass otherwise it will through error
response.html.render(timeout=20)   # create bs object to parse HTML
soup = bs(response.html.html, "html.parser")
 print("Title of the Video: ", soup.find("meta", itemprop="name")["cont ent"])
 print("Views on the Video: ", soup.find("meta", itemprop="interactionC ount")["content"])
```

```
Title of the Video:  What If You Lived on Uranus?
Views on the Video:  153544
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

## 18. Web Scrapping from Google

Web scraping is an automatic method to obtain large amounts of data from websites. Most of this data is unstructured data in an HTML format which is then converted into structured data in a spread sheet or a database so that it can be used in various applications.

There are many different ways to perform web scraping to obtain data from websites. These include using online services, particular API, or even creating your code for web scraping from scratch. Many large websites, like Google, Twitter, Facebook, Stack Overflow, etc. have APIs that allow you to access their data in a structured format.

This is the best option, but there are other sites that don't allow users to access large amounts of data in a structured form or they are simply not that technologically advanced. In that situation, it's best to use Web Scraping to scrape the website for data.

Web scraping requires two parts, namely the crawler and the scraper.

The crawler is an artificial intelligence algorithm that browses the web to search for the particular data required by following the links across the internet.

The scraper, on the other hand, is a specific tool created to extract data from the website. The design of the scraper can vary greatly according to the complexity and scope of the project so that it can quickly and accurately extract the data.

```
!pip install bs4
!pip install requests
 import requests
 import bs4
 text= "geeksforgeeks"
url = 'https://google.com/search?q=' + text
 request_result=requests.get( url )
 soup = bs4.BeautifulSoup(request_result.text,  "html.parser")
print(soup)
```

```
<!DOCTYPE doctype html>
<html lang="zh-TW">
<head><meta charset="utf-8"/>
<meta
content="/images/branding/googleg/1x/googleg_standard_color_128dp.png"
itemprop="image"/>
<title>geeksforgeeks - Google 搜尋</title>
<script nonce="UgdjTwsXNpu6VXHrspInHA">(function(){ document.
```

```
heading_object=soup.find_all( 'h3' )
 for info in heading_object:
    print(info.getText())
    print("------")
```

```
GeeksforGeeks
------
GeeksforGeeks - Learn To Code - Google Play 應用程式
------
GeeksforGeeks - YouTube ------
GeeksforGeeks - Twitter
------
GeeksforGeeks - Home | Facebook
------
GeeksforGeeks 的貼文 - LinkedIn
------
GeeksforGeeks's (@geeks_for_geeks) Instagram profile • 1,655 ...
------
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

## 19. Topic Modelling using LDA.

Latent Dirichlet Allocation (LDA) is a popular topic modeling technique to extract topics from a given corpus. The term latent conveys something that exists but is not yet developed. In other words, latent means hidden or concealed.

Now, the topics that we want to extract from the data are also “hidden topics”. It is yet to be discovered. Hence, the term “latent” in LDA. The Dirichlet allocation is after the Dirichlet distribution and process.

A tool and technique for Topic Modeling, Latent Dirichlet Allocation (LDA) classifies or categorizes the text into a document and the words per topic, these are modeled based on the Dirichlet distributions and processes.

The LDA makes two key assumptions:

- Documents are a mixture of topics, and
- Topics are a mixture of tokens (or words)

And, these topics using the probability distribution generate the words. In statistical language, the documents are known as the probability density (or distribution) of topics and the topics are the probability density (or distribution) of words.

```
import pandas as pd dataset =
pd.read_csv('drive/MyDrive/Classroom/Reviews.csv') data = dataset[0:1000]
 import nltk nltk.download('stopwords') nltk.download('punkt')
nltk.download('wordnet') nltk.download('omw-1.4') from nltk.tokenize
import word_tokenize from nltk.corpus import stopwords  #stopwords from
nltk.stem import WordNetLemmatizer   from sklearn.feature_extraction.text
import TfidfVectorizer
stop_words=set(nltk.corpus.stopwords.words('english'))
 def clean_text(headline):
le=WordNetLemmatizer()
    word_tokens=word_tokenize(headline)
    tokens=[le.lemmatize(w) for w in word_tokens if w not in stop_words and le n(w)>3]
    cleaned_text=" ".join(tokens)       return cleaned_text
data['cleaned_text']=data['Text'].apply(clean_text)


vect =TfidfVectorizer(stop_words=stop_words,max_features=1000)
vect_text=vect.fit_transform(data['cleaned_text'])
 from sklearn.decomposition import LatentDirichletAllocation
lda_model=LatentDirichletAllocation(n_components=10, learning_method='online',ra
ndom_state=42,max_iter=1)
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

```
lda_top=lda_model.fit_transform(vect_text)

print("Document 0: ") for i,topic in
enumerate(lda_top[0]):   print("Topic ",i,":
",topic*100,"%")
Document 0:
Topic  0 :  2.2792673247031985 %
Topic  1 :  2.2795217163079737 % Topic  2 :
2.279382913448876 %
Topic  3 :  2.2792309609635315 %
Topic  4 :  2.2792629883677114 %
Topic  5 :  2.2792792543486486 %
Topic  6 :  2.2792262663390828 % Topic  7 :
2.279598381596126 %
Topic  8 :  2.2792525792649743 %
Topic  9 :  79.48597761465989 %
 vocab = vect.get_feature_names() for i, comp in
enumerate(lda_model.components_):
    vocab_comp = zip(vocab, comp)      sorted_words = sorted(vocab_comp, key= lambda x:x[1],
reverse=True)[:10]      print("\nTopic "+str(i)+": ")      for t in sorted_words:
          print(t[0],end=" ")
Topic 0:
good blend taste sugar order like item love stop ordered
Topic 1:
chip kettle flavor brand like better potato great spicy best
Topic 2:
coffee chocolate flavor chip salt like good love perfect taste
Topic 3:
taffy chewy watermelon grain food flavor complaint quality bear would
Topic 4:
help shipping little roasted candy 00 like friend also especially
Topic 5:
coffee good drink ingredient help food minute since like quality
Topic 6:
sauce service marinade love salt year fact product seller rice
Topic 7:
product brings tasting great pack quality salsa juice amazon pineapple
Topic 8:
price great half good taffy gram stuff selling single product
Topic 9:
chip like great good taste love flavor product food really
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

## 20. Social Network Analysis using Facebook

Social network analysis (SNA) is a process of quantitative and qualitative analysis of a social network. SNA measures and maps the flow of relationships and relationship changes between knowledgepossessing entities. Simple and complex entities include websites, computers, animals, humans, groups, organizations and nations.

The SNA structure is made up of node entities, such as humans, and ties, such as relationships. The advent of modern thought and computing facilitated a gradual evolution of the social networking concept in the form of highly complex, graph-based networks with many types of nodes and ties. These networks are the key to procedures and initiatives involving problem-solving, administration, and operations.

SNA usually refers to varied information and knowledge entities, but most actual studies focus on human (node) and relational (tie) analysis. The tie value is social capital.

SNA is often diagrammed with points (nodes) and lines (ties) to present the intricacies related to social networking. Professional researchers perform analysis using software and unique theories and methodologies.

SNA research is conducted in either of the following ways:

- Studying the complete social network, including all ties in a defined population.
- Studying egocentric components, including all ties and personal communities, which involves studying the relationship between the focal points in the network and the social ties they make in their communities.

A snowball network forms when alters become egos and can create, or nominate, additional alters. Conducting snowball studies is difficult, due to logistical limitations. The abstract SNA concept is complicated further by studying hybrid networks, in which complete networks may create unlisted alters available for ego observation. Hybrid networks are analogous to employees affected by outside consultants, where data collection is not thoroughly defined.

Three analytical tendencies make SNA distinctive, as follows:

- Groups are not assumed to be societal building blocks.
- Studies focus on how ties affect individuals and other relationships, versus discrete individuals, organizations or states.
- Studies focus on structure, the composition of ties and how they affect societal norms, versus assuming that socialized norms determine behaviour.

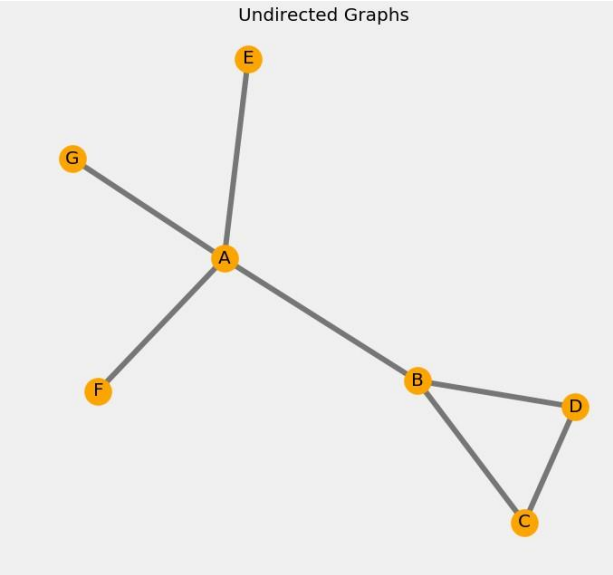
# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

```
import numpy as np  import
pandas as pd  import
matplotlib.pyplot as plt import
seaborn as sns import networkx as
nx


# Undirected
Graphs g =
nx.Graph()
g.add_edge('A', 'B')
g.add_edge('B', 'C')
g.add_edge('C', 'D')
g.add_edge('B', 'D')
g.add_edge('A', 'E')
g.add_edge('A', 'F')
g.add_edge('A', 'G')

import warnings warnings.filterwarnings('ignore') plt.rcParams['figure.figsize'] = (10, 10)
plt.style.use('fivethirtyeight') pos = nx.spring_layout(g) nx.draw_networkx_nodes(g, pos, node_size =
900, node_color = 'orange') nx.draw_networkx_edges(g, pos, width = 6, alpha = 0.5, edge_color =
'black') nx.draw_networkx_labels(g, pos, font_size = 20, font_family = 'sans-serif')
plt.title('Undirected Graphs', fontsize = 20) plt.axis('off') plt.show()
```



# VNRVJIET

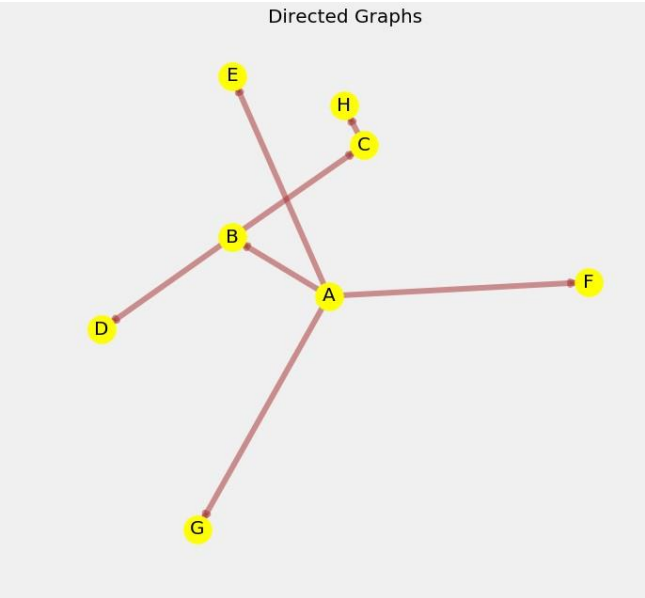
**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

- An undirected graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are bidirectional. An undirected graph is sometimes called an undirected network. In contrast, a graph where the edges point in a direction is called a directed graph.

```
# Directed
Graphs g =
nx.DiGraph()
g.add_edge('A', 'B')
g.add_edge('B', 'C')
g.add_edge('C', 'H')
g.add_edge('B', 'D')
g.add_edge('A', 'E')
g.add_edge('A', 'F')
g.add_edge('A', 'G')
 import warnings warnings.filterwarnings('ignore')
plt.rcParams['figure.figsize'] = (10, 10)
plt.style.use('fivethirtyeight')
pos = nx.spring_layout(g)
nx.draw_networkx_nodes(g, pos, node_size = 900, node_color = 'yellow')
nx.draw_networkx_edges(g, pos, edge_color = 'brown', width = 6, alpha = 0.5)
nx.draw_networkx_labels(g, pos, font_size=20, font_family='sans-serif')
plt.title('Directed Graphs', fontsize = 20)
plt.axis('off')
 plt.show()
```



# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

- A directed graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are directed from one vertex to another. A directed graph is sometimes called a digraph or a directed network.

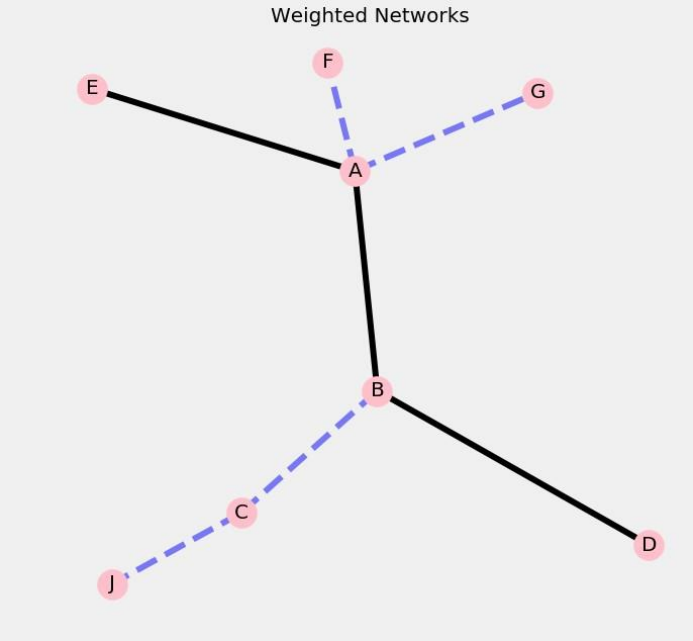
```
# weighted
networks #
Undirected Graphs
g = nx.Graph()
g.add_edge('A', 'B', weight = 8)
g.add_edge('B', 'C', weight = 12)
g.add_edge('C', 'J', weight = 15)
g.add_edge('B', 'D', weight = 3)
g.add_edge('A', 'E', weight = 5)
g.add_edge('A', 'F', weight = 18)
g.add_edge('A', 'G', weight = 10)
import warnings warnings.filterwarnings('ignore')
 plt.rcParams['figure.figsize'] = (10, 10)
 plt.style.use('fivethirtyeight')
 plt.title('Weighted Networks', fontsize = 20)
 elarge = [(u, v) for (u, v, d) in g.edges(data=True) if d['weight'] <  10] esmall = [(u, v)
for (u, v, d) in g.edges(data=True)  if d['weight'] >= 10]
pos = nx.spring_layout(g)
# nodes
nx.draw_networkx_nodes(g, pos, node_size = 900, node_color = 'pink')
# edges
nx.draw_networkx_edges(g, pos, edgelist = elarge, width = 6)
 nx.draw_networkx_edges(g, pos, edgelist = esmall, width = 6, alpha = 0.5, edge_color = 'b', style = 'dashed')
 # labels
nx.draw_networkx_labels(g, pos, font_size = 20, font_family = 'sans-serif') plt.axis('off') plt.show()
```

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____




- A weighted graph is a graph in which each branch is given a numerical weight. A weighted graph is therefore a special type of labeled graph in which the labels are numbers (which are usually taken to be positive). SEE ALSO: Labeled Graph, Taylor's Condition, Weighted Tree.

```
# signed
networks #
Undirected
Graphs
g=nx.Graph()
g.add_edge('A', 'B', sign = '+')
g.add_edge('B', 'C', sign = '-')
g.add_edge('C', 'J', sign = '+')
g.add_edge('B', 'D', sign = '-')
g.add_edge('A', 'E', sign = '+')
g.add_edge('A', 'F', sign = '+')
g.add_edge('A', 'G', sign = '-')
 import warnings warnings.filterwarnings('ignore')
plt.rcParams['figure.figsize'] = (10, 10)
 plt.style.use('fivethirtyeight')
plt.title('Signed Networks', fontsize = 20)
elarge = [(u, v) for (u, v, d) in g.edges(data=True)
 if d['sign'] == '+']
 esmall = [(u, v) for (u, v, d) in g.edges(data=True) if d['sign'] == '-']
```

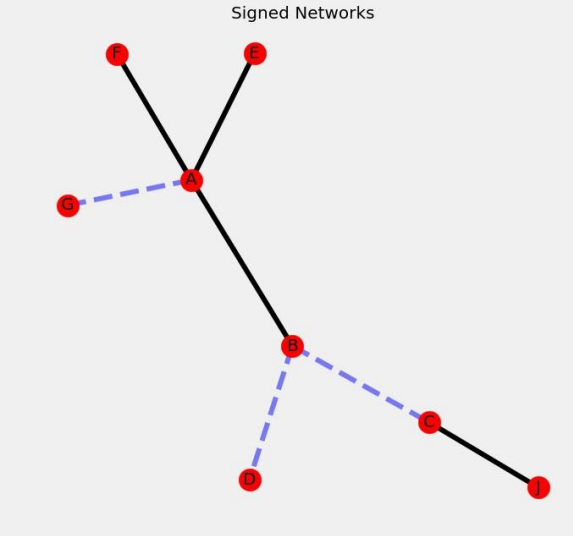
# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

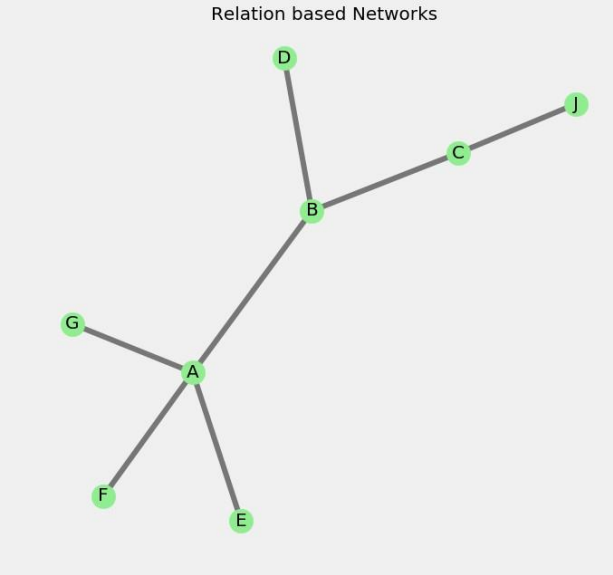
pos = nx.spring_layout(g)  nx.draw_networkx_nodes(g, pos, node_size=700)
nx.draw_networkx_edges(g, pos, edgelist=elarge,width=6) nx.draw_networkx_edges(g, pos, edgelist=esmall,
width=6, alpha=0.5, edge_color='b', style='dashed') nx.draw_networkx_labels(g, pos, font_size=20, font_family='sans-serif') plt.axis('off') plt.show()




- In the area of graph theory in mathematics, a signed graph is a graph in which each edge has a positive or negative sign. A signed graph is balanced if the product of edge signs around every cycle is positive. Three fundamental questions about a signed graph are: Is it balanced?

```
# relation networks
#UndirectedGraph
s g=nx.Graph()
g.add_edge('A', 'B', relation = 'family')
g.add_edge('B', 'C', relation = 'friend')
g.add_edge('C', 'J', relation = 'coworker')
g.add_edge('B', 'D', relation = 'family')
g.add_edge('A', 'E', relation = 'friend')
g.add_edge('A', 'F', relation = 'coworker')
g.add_edge('A', 'G', relation = 'friend')
 import warnings
warnings.filterwarnings('ignore')
 plt.rcParams['figure.figsize'] = (10, 10)
plt.style.use('fivethirtyeight')
plt.title('Relation based Networks', fontsize = 20)
 pos = nx.spring_layout(g)
```

```
nx.draw_networkx_nodes(g, pos, node_size = 700, node_color = 'lightgreen')
nx.draw_networkx_edges(g, pos, width = 6, alpha = 0.5, edge_color = 'black')
nx.draw_networkx_labels(g, pos, font_size = 20, font_family = 'sans-serif')
plt.axis('off')
plt.show()
```




## Checking the different types of Layouts available in Networkx

```
[x for x in nx.__dir__() if x.endswith('_layout')]
```

```
['bipartite_layout',
 'circular_layout',
 'kamada_kawai_layout',
 'random_layout',
 'rescale_layout',
 'shell_layout',
 'spring_layout',
 'spectral_layout',
 'fruchterman_reingold_layout']
```

```
# bipartite graphs from networkx.algorithms
import bipartite
B = nx.Graph()
B.add_nodes_from(['A','B','C','D','E'], bipartite = 0)
B.add_nodes_from([1, 2, 3, 4], bipartite = 1)
B.add_edges_from([('A', 1),('B', 1),('C', 1),('C', 3),('D', 2),('E',3),('E',4)]) import warnings
```

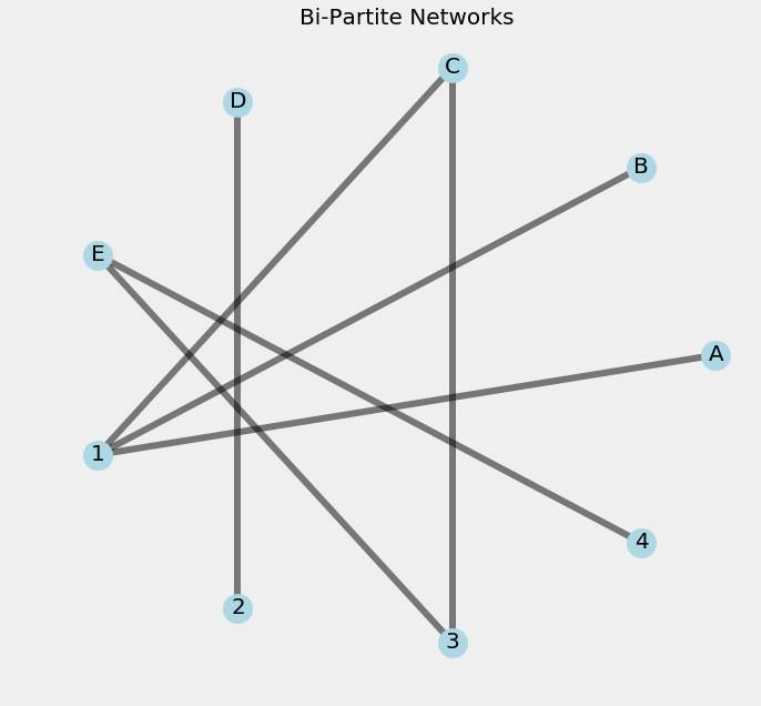
# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

```
warnings.filterwarnings('ignore') plt.rcParams['figure.figsize'] = (10, 10) plt.style.use('fivethirtyeight')
plt.title('Bi-Partite Networks', fontsize = 20) pos = nx.shell_layout(B)   nx.draw_networkx_nodes(B,
pos, node_size = 700, node_color = 'lightblue') nx.draw_networkx_edges(B, pos, width = 6, alpha =
0.5, edge_color = 'black') nx.draw_networkx_labels(B, pos, font_size = 20, font_family = 'sans-serif')
plt.axis('off') plt.show()
```




- In the mathematical field of graph theory, a bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets and such that every edge connects a vertex in to one in. Vertex sets and are usually called the parts of the grap

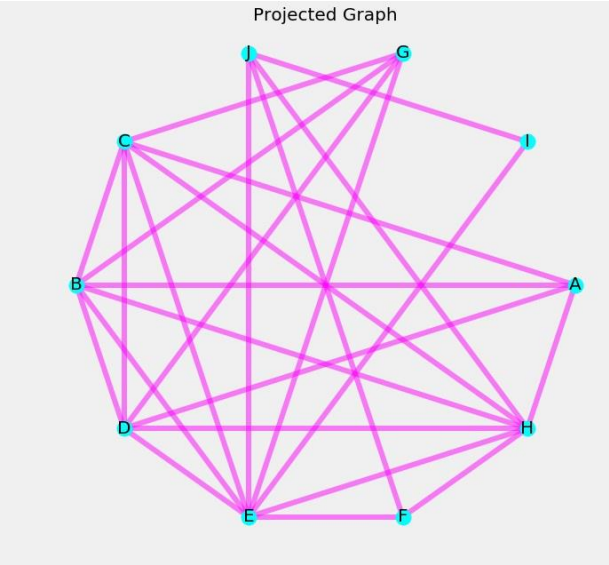
```
# we can also check whether a graph is bipartite or not
bipartite.is_bipartite(B)
```

True

```
# checking if a set of nodes is a bipartition of a graph X = set([1, 2, 3, 4])
bipartite.is_bipartite_node_set(B, X)
```

True

```
# projected graphs
B = nx.Graph()
B.add_edges_from([('A', 1),('B', 1),('C', 1),('D', 1),('H', 1),('B', 2),('C', 2),('D', 1),
        ('H', 1),('B', 2),('C', 2),('D', 2),('E', 2),('G', 2),('E', 3),('F', 3),
        ('H', 3), ('J', 3), ('E', 4), ('I',4), ('J', 4)])
X = set(['A','B','C','D','E','F','G','H','I','J']) P = bipartite.projected_graph(B, X) pos = nx.circular_layout(P)
nx.draw_networkx_nodes(P, pos, node_color = 'cyan') nx.draw_networkx_edges(P, pos, edge_color =
'magenta', width = 6, alpha = 0.5) nx.draw_networkx_labels(P, pos, font_size = 20, font_family = 'sans-
serif') plt.title('Projected Graph', fontsize = 20) plt.axis('off') plt.show()
```




- Breadth-first search is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root, and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level

```
a = nx.Graph()
a.add_edge('A', 'B')
a.add_edge('A', 'K')
```

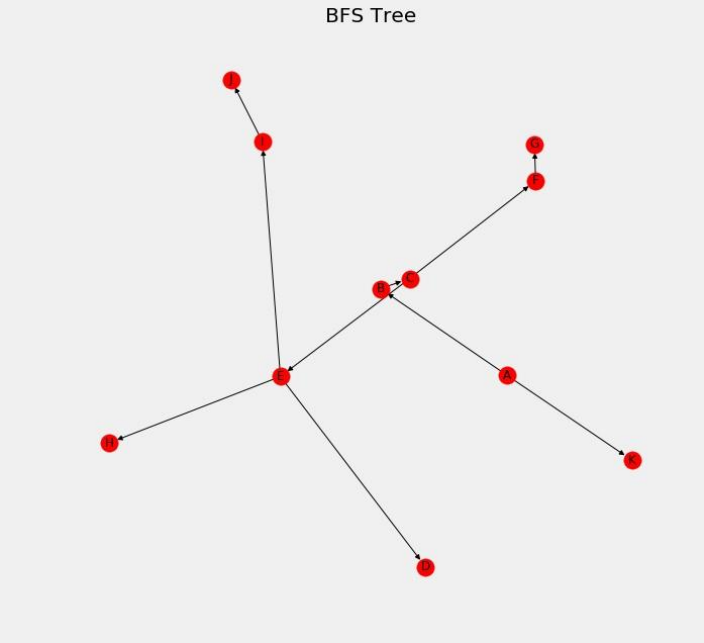
# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

```
a.add_edge('B', 'C')
a.add_edge('C', 'F')
a.add_edge('F', 'E')
a.add_edge('F', 'G')
a.add_edge('C', 'E')
a.add_edge('E', 'D')
a.add_edge('E', 'H')
a.add_edge('K', 'B')
a.add_edge('E', 'I')
a.add_edge('I', 'J') a =
nx.bfs_tree(a, 'A') pos =
nx.kamada_kawai_layout(a)
nx.draw_networkx(a, size = 900)
plt.axis('off') plt.title('BFS Tree')
plt.show()
```




*# let's check the edges of the tree*

```
a.edges()
```

```
OutEdgeView([('A', 'B'), ('A', 'K'), ('B', 'C'), ('C', 'F'), ('C', 'E'), ('F', 'G'), ('E', 'D'), ('E', 'H'), ('E', 'I'), ('I', 'J')])
```

*# let's check the shortest path from A* nx.shortest_path_length(a, 'A')

```
{'A': 0,
 'B': 1,
 'K': 1,
```

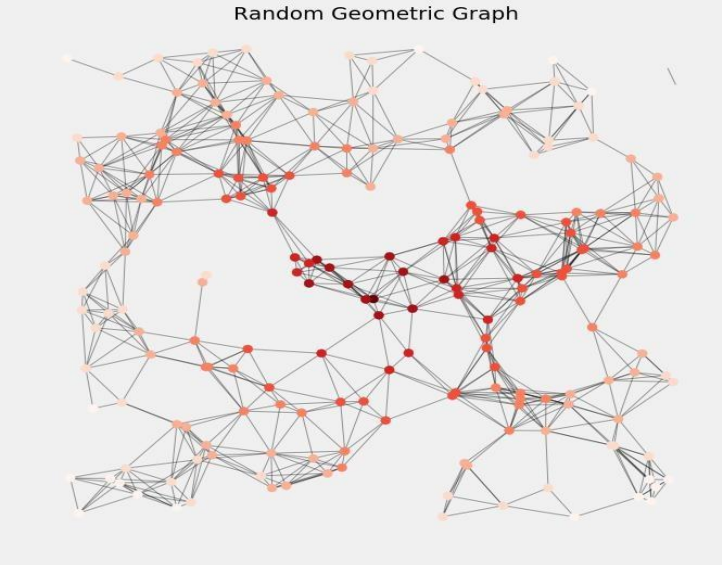
```
 'C': 2,
 'F': 3,
 'E': 3,
 'G': 4,
 'D': 4,
 'H': 4,
 'I': 4,
 'J': 5}

# checking the average shortest path length
nx.average_shortest_path_length(a)
0.6545454545454545
import matplotlib.pyplot as plt
import networkx as nx
G = nx.random_geometric_graph(200, 0.125)
# position is stored as node attribute data for random_geometric_graph pos =
nx.get_node_attributes(G, 'pos') # find node near center (0.5,0.5) dmin = 1 ncenter = 0
for n in pos:
    x, y = pos[n]     d = (x - 0.5)**2 + (y -
0.5)**2
 if d < dmin ncenter = n   dmin = d

# color by path length from node near center
p = dict(nx.single_source_shortest_path_length(G, ncenter))
plt.rcParams['figure.figsize'] = (10, 10)
nx.draw_networkx_edges(G, pos, nodelist=[ncenter], alpha=0.4)
 nx.draw_networkx_nodes(G, pos, nodelist=list(p.keys()),node_size=80,
node_color=list(p.values()),cmap=plt.cm.Reds_r)
plt.title('Random Geometric Graph', fontsize = 20)
plt.xlim(-0.05, 1.05)
plt.ylim(-0.05, 1.05)
 plt.axis('off')
plt.show()
```

# VNRVJIET

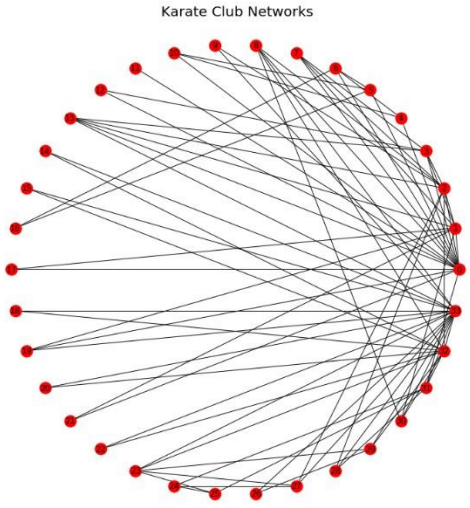
**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____




```
import matplotlib.pyplot as plt import
networkx as nx G =
nx.karate_club_graph()
plt.style.use('fivethirtyeight')
nx.draw_circular(G, with_labels=True)
plt.title(' Karate Club Networks') plt.show()
```



# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

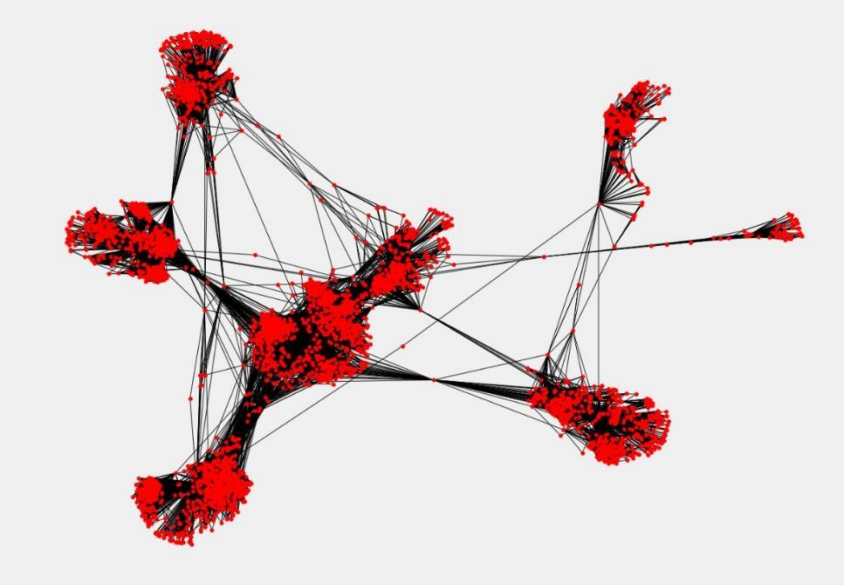
**Experiment No:** _____ **Date:** _____

## Facebook Network Analysis

**import** os print(os**.**listdir('../input/'))

['facebook-combined.txt'] *# reading the dataset* fb = nx**.**read_edgelist('../input/facebook-combined.txt', create_using = nx**.**Graph(), nodetype = int) print(nx**.**info(fb))

Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88234
Average degree: 43.6910
pos = nx**.**spring_layout(fb)
**import** warnings warnings**.**filterwarnings('ignore') plt**.**style**.**use('fivethirtyeight') plt**.**rcParams['figure.figsize'] = (20, 15) plt**.**axis('off') nx**.**draw_networkx(fb, pos, with_labels = **False**, node_size = 35) plt**.**show()



*# checking the betweenness centrality* bc = nx**.**betweenness_centrality(fb) bc

*# checking the degree of each node in the network* degree = nx**.**degree_histogram(fb) degree print(fb**.**order()) print(fb**.**size())

4039
88234

# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

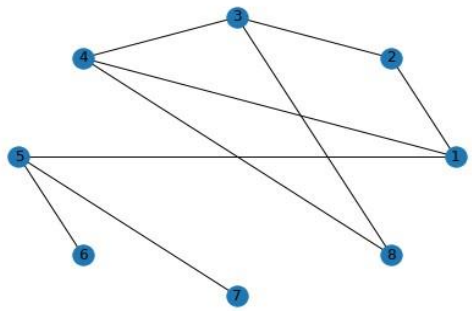
## 21. Explore any 7 methods in networkx library

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and function of complex networks. It is used to study large complex networks represented in form of graphs with nodes and edges. Using networkx we can load and store complex networks. We can generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms and draw networks.

Highlights of NetworkX

- Classes for diagrams and digraphs.
- Transformation of diagrams to and from a few configurations. • Capacity to steadily develop irregular diagrams or build them.
- Capacity to find subgraphs, inner circles, and k-centres.
- Investigate nearness, Degree, distance across, range, focus, betweenness, and so forth.
- Attract networks 2D and 3D.

```
import networkx as nx import
matplotlib.pyplot as plt g =
nx.Graph()
g.add_edge(1, 2)
g.add_edge(2, 3)
g.add_edge(3, 4)
g.add_edge(1, 4)
g.add_edge(1, 5)
g.add_edge(5, 6)
g.add_edge(5, 7)
g.add_edge(4, 8)
g.add_edge(3, 8)

# drawing in circular layout nx.draw_circular(g, with_labels
= True) plt.savefig("filename1.png")
```



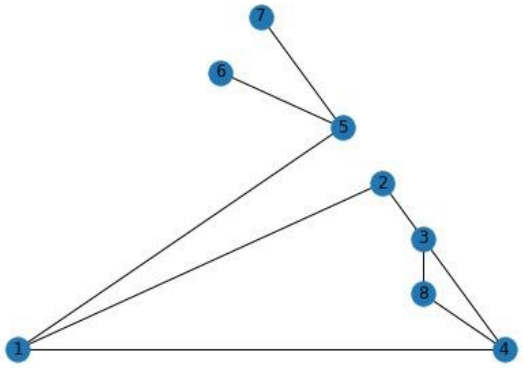
# VNRVJIET

**Name of the Laboratory:**

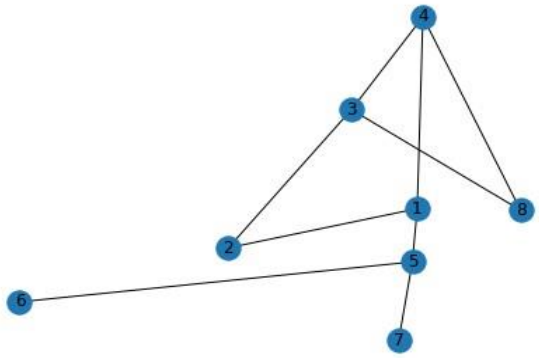
**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

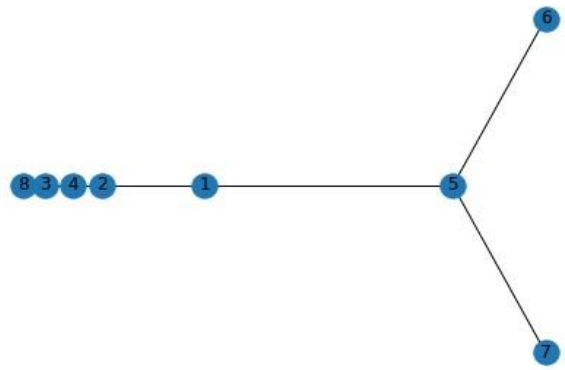
```
# drawing in planar layout nx.draw_planar(g,
with_labels = True) plt.savefig("filename2.png")
```



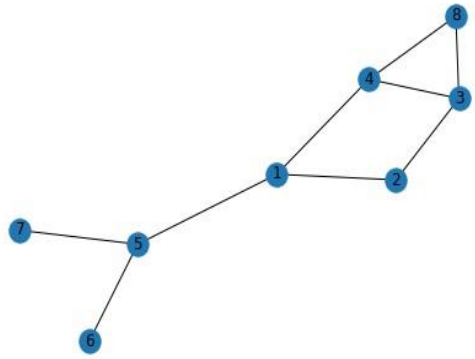

```
#
drawing in random layout nx.draw_random(g,
with_labels = True) plt.savefig("filename3.png")
```



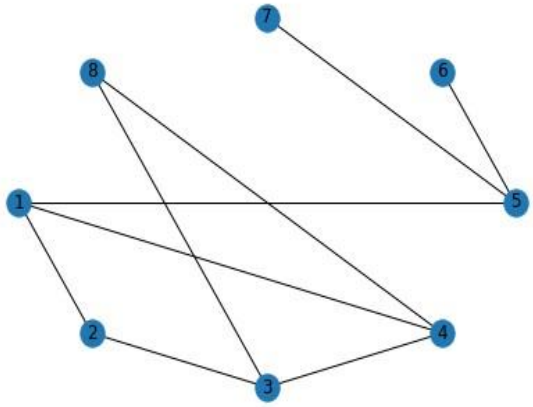

```
 # drawing in spectral layout nx.draw_spectral(g,
with_labels = True) plt.savefig("filename4.png")
```



```
# drawing in spring layout nx.draw_spring(g,
with_labels = True) plt.savefig("filename5.png")
```




```
# drawing in shell layout nx.draw_shell(g,
with_labels = True) plt.savefig("filename6.png")
```



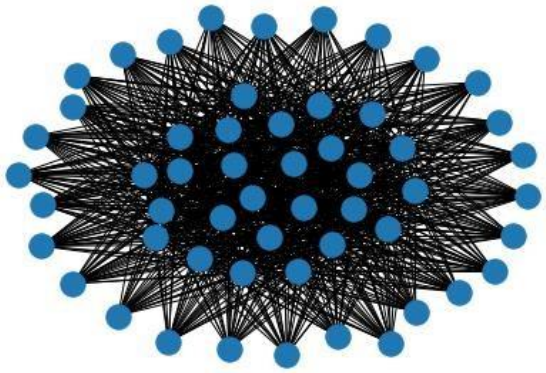

```
G =
nx.complete_multipartite_graph(28, 16, 10) pos =
nx.multipartite_layout(G) print('Multipartite Graph')
nx.draw(G)
```

# VNRVJIET

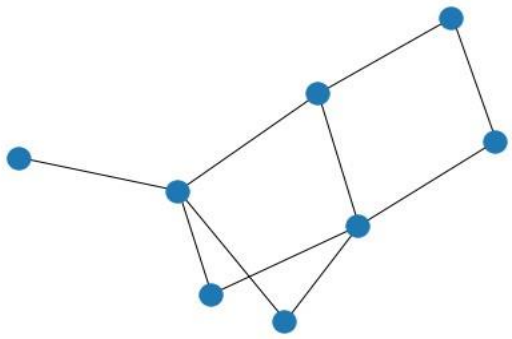
**Name of the Laboratory:**

**Name of the Experiment:** _____

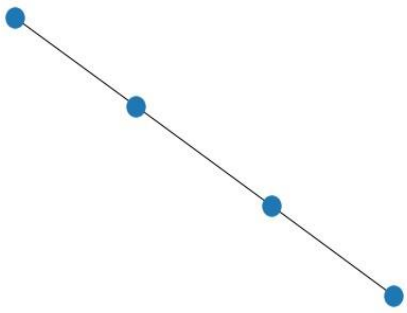
**Experiment No:** _____ **Date:** _____



```
G = nx.bipartite.gnmk_random_graph(3, 5, 10, seed=123) top = nx.bipartite.sets(G)[0] pos = nx.bipartite_layout(G, top)
print('Bipartite Graph')
nx.draw(G)
```



```
G = nx.path_graph(4) pos = nx.kamada_kawai_layout(G)
print('kamada kawai Graph')
nx.draw(G)
```



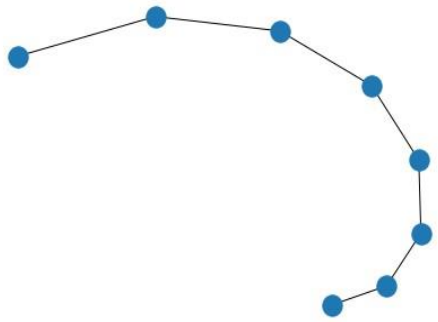
# VNRVJIET

**Name of the Laboratory:**

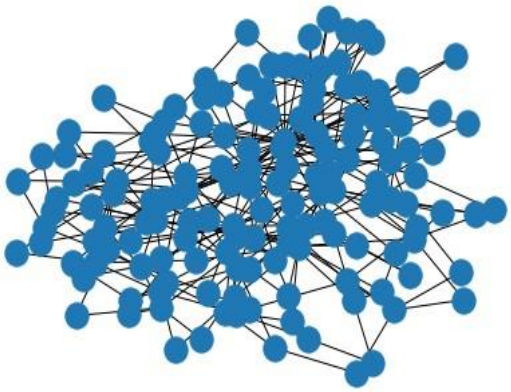
**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

```
G = nx.path_graph(8) pos
= nx.spiral_layout(G)
print('Spiral Graph')
nx.draw(G, pos=pos)
```



```
H=nx.barabasi_albert_graph(150, m=2, seed=1)
print('barabasi albert Graph') nx.draw(H)
```



# VNRVJIET

**Name of the Laboratory:**

**Name of the Experiment:** _____

**Experiment No:** _____ **Date:** _____

## 22. Web crawler using python

A web crawler, spider, or search engine bot downloads and indexes content from all over the Internet. The goal of such a bot is to learn what (almost) every webpage on the web is about, so that the information can be retrieved when it's needed. They're called "web crawlers" because crawling is the technical term for automatically accessing a website and obtaining data via a software program.

These bots are almost always operated by search engines. By applying a search algorithm to the data collected by web crawlers, search engines can provide relevant links in response to user search queries, generating the list of webpages that show up after a user types a search into Google or Bing (or another search engine).

A web crawler bot is like someone who goes through all the books in a disorganized library and puts together a card catalog so that anyone who visits the library can quickly and easily find the information they need. To help categorize and sort the library's books by topic, the organizer will read the title, summary, and some of the internal text of each book to figure out what it's about.

Web crawlers start their crawling process by downloading the website's robot.txt file. The file includes sitemaps that list the URLs that the search engine can crawl. Once web crawlers start crawling a page, they discover new pages via links. These crawlers add newly discovered URLs to the crawl queue so that they can be crawled later. Thanks to these techniques, web crawlers can index every single page that is connected to others.

Since pages change regularly, it is also important to identify how frequently search engines should crawl them. Search engine crawlers use several algorithms to decide factors such as how often an existing page should be re-crawled and how many pages on a site should be indexed.

```
 from bs4 import * from urllib.request import urlopen from html.parser
import HTMLParser import re choice=input("Enter choice \n1) indeed.com
2) dice.com \n") skill=input("Enter skill ") pincode=input("Enter pincode ")

if(choice=
=1):
        html_page = urlopen("https://www.indeed.com/jobs?q=" +skill+"&l=" +pincode)     soup =
BeautifulSoup(html_page,"html.parser")     f= open("links_indeed.txt","w+")     for link in soup.findAll('a',
attrs={'href': re.compile("vjs")}):
       f.write("https://www.indeed.com" + link.get('href') + "\n")



f.close()
```

```
else:
    html_page = urlopen("https://www.dice.com/jobs?q=" +skill+"&l=" +pincode)
    soup = BeautifulSoup(html_page,"html.parser")
    f= open("links_dice.txt","w+")     for link in soup.findAll('a', attrs={'href': re.compile("detail/")}):
    f.write("https://www.dice.com" + link.get('href') + "\n")
     f.close()
```

```
Enter choice
1) indeed.com
2) dice.com
1
Enter skill
fresher
Enter pincode
500072
```