**Practical No.1**

**Aim:** Write a program to demonstrate bitwise operation

**Source Code:**

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

corpus=[

'this is the first document.',

'this document is second document.',

'and this is the third one.',

'is this the first document?',

]

vectorizer= CountVectorizer()

X=vectorizer.fit\_transform(corpus)

print("fit transform is ")

print(X.toarray())

df=pd.DataFrame(X.toarray(),columns=vectorizer.get\_feature\_names())

print("the generated data frame is")

print(df)

alldata= df[(df['this']==1)&(df['first']==1)]

print("indices where 'this'and 'first'terms are present are ",alldata.index.tolist())

ordata= df[(df['this']==1)|(df['first']==1)]

print("indices where either of 'this'and 'first'terms are present are ",ordata.index.tolist())

notdata=df[(df['and']!=1)]

print("indices where 'and' term is not present ",notdata.index.tolist())

**Output:**

runfile('C:/Users/gauri/untitled12.py', wdir='C:/Users/gauri')

fit transform is

[[0 1 1 1 0 0 1 0 1]

[0 2 0 1 0 1 0 0 1]

[1 0 0 1 1 0 1 1 1]

[0 1 1 1 0 0 1 0 1]]

the generated data frame is

and document first is one second the third this

0 0 1 1 1 0 0 1 0 1

1 0 2 0 1 0 1 0 0 1

2 1 0 0 1 1 0 1 1 1

3 0 1 1 1 0 0 1 0 1

indices where 'this'and 'first'terms are present are [0, 3]

indices where either of 'this'and 'first'terms are present are [0, 1, 2, 3]

indices where 'and' term is not present [0, 1, 3]

**Practical No.2**

**Aim:** Implement PageRank Algorithm.

**Source Code:**

import numpy as np

from scipy.sparse import csc\_matrix

from fractions import Fraction

def float\_format(vector,decimal):

return np.round((vector).astype(np.float),decimals=decimal)

G=np.matrix([[1,1,0],

[1,0,1],

[0,1,0]])

n=len(G)

print(n)

M=csc\_matrix(G,dtype=np.float)

rsums=np.array(M.sum(1))[:,0]

ri,ci=M.nonzero()

M.data/rsums[ri]

dp=Fraction(1,n)

E=np.zeros((3,3))

E[:]=dp

beta=0.85

A=beta\*M+((1-beta)\*E)

r=np.matrix([dp,dp,dp])

r=np.transpose(r)

previous\_r=r

for it in range(1,30):

r=A\*r

if(previous\_r==r).all():

break

previous\_r=r

print("Final:\n",float\_format(r,3))

**Output:**

runfile('C:/Users/ckt/prac2ir.py', wdir='C:/Users/ckt')

3

('Final:\n', array([[0.617],

[0.617],

[0.333]]))

('Final:\n', array([[1.127],

[0.886],

[0.603]]))

('Final:\n', array([[1.841],

[1.601],

[0.884]]))

('Final:\n', array([[3.142],

[2.533],

[1.577]]))

('Final:\n', array([[5.186],

[4.373],

[2.515]]))

('Final:\n', array([[8.729],

[7.15 ],

[4.321]]))

('Final:\n', array([[14.507],

[12.103],

[ 7.087]]))

('Final:\n', array([[24.303],

[20.04 ],

[11.972]]))

('Final:\n', array([[40.507],

[33.65 ],

[19.85 ]]))

('Final:\n', array([[67.734],

[56.004],

[33.303]]))

('Final:\n', array([[113.029],

[ 93.733],

[ 55.455]]))

('Final:\n', array([[188.859],

[156.323],

[ 92.784]]))

('Final:\n', array([[315.303],

[261.295],

[154.773]]))

('Final:\n', array([[526.677],

[436.133],

[258.669]]))

('Final:\n', array([[879.463],

[728.619],

[431.787]]))

('Final:\n', array([[1468.863],

[1216.556],

[ 721.319]]))

('Final:\n', array([[2452.943],

[2031.992],

[1204.41 ]]))

('Final:\n', array([[4096.662],

[3393.217],

[2011.66 ]]))

('Final:\n', array([[6841.474],

[5667.15 ],

[3359.311]]))

**Practical No.3**

**Aim:** Implement Dynamic programming algorithm for computing the edit distance between strings s1 and s2.

**Source Code:**

import numpy as np

def Levenshtein(s1, s2):

if s1 == "":

return len(s2)

if s2 == "":

return len(s1)

if s1[-1] == s2[-1]:

cost = 0

else:

cost = 1

res = min([Levenshtein(s1[:-1], s2)+1,

Levenshtein(s1, s2[:-1])+1,

Levenshtein(s1[:-1], s2[:-1]) + cost])

return res

print(Levenshtein("execution", "intention"))

**Output:**

**runfile('C:/Users/gauri/editdistancepy.py', wdir='C:/Users/gauri')**

**5**

**Practical No 4**

**Aim:** Write a program to Compute Similarity between two text documents.

**Source Code:**

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

def cosine\_similarity(x, y):

# Ensure length of x and y are the same

if len(x) != len(y) :

return None

# Compute the dot product between x and y

dot\_product = np.dot(x, y)

# Compute the L2 norms (magnitudes) of x and y

magnitude\_x = np.sqrt(np.sum(x\*\*2))

magnitude\_y = np.sqrt(np.sum(y\*\*2))

# Compute the cosine similarity

cosine\_similarity = dot\_product / (magnitude\_x \* magnitude\_y)

return cosine\_similarity

corpus = ['data science is one of the most important fields of science',

'this is one of the best data science courses',

'data scientists analyze data']

# Create a matrix to represent the corpus

X = CountVectorizer().fit\_transform(corpus).toarray()

print(X)

cos\_sim\_1\_2 = cosine\_similarity(X[0, :], X[1, :])

cos\_sim\_1\_3 = cosine\_similarity(X[0, :], X[2, :])

cos\_sim\_2\_3 = cosine\_similarity(X[1, :], X[2, :])

print('Cosine Similarity between: ')

print('\tDocument 1 and Document 2: ', cos\_sim\_1\_2)

print('\tDocument 1 and Document 3: ', cos\_sim\_1\_3)

print('\tDocument 2 and Document 3: ', cos\_sim\_2\_3)

**Output:**

runfile('C:/Users/gauri/cosinepython.py', wdir='C:/Users/gauri')

[[0 0 0 1 1 1 1 1 2 1 2 0 1 0]

[0 1 1 1 0 0 1 0 1 1 1 0 1 1]

[1 0 0 2 0 0 0 0 0 0 0 1 0 0]]

Cosine Similarity between:

Document 1 and Document 2: 0.6885303726590962

Document 1 and Document 3: 0.21081851067789195

Document 2 and Document 3: 0.2721655269759087

**Practical No. 7**

**Aim:** Write program for pre-processing of Text document: stop word removal

**Source Code:**

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

example\_sent= "We are students of CKT college"

stop\_words=set(stopwords.words('english'))

word\_tokens=word\_tokenize(example\_sent)

filtered\_sentence=[w for w in word\_tokens if not w in stop\_words]

filtered\_sentence=[]

for w in word\_tokens:

if w not in stop\_words:

filtered\_sentence.append(w)

print(word\_tokens)

print(filtered\_sentence)

**Output:**

runfile('C:/Users/CKT/prac7.py', wdir='C:/Users/CKT') ['We', 'are', 'students', 'of', 'CKT', 'college'] ['We', 'students', 'CKT', 'college']

Practical 8

**Aim:** Write a program for mining Twitter to identify tweets for a specific period and identify trends

and named entities.

**Source Code:**

#Import the necessary methods from tweepy library

from tweepy.streaming import StreamListener

from tweepy import OAuthHandler

from tweepy import Stream

#Variables that contains the user credentials to access Twitter API

consumer\_key = "3yMYKK5Ben0iUaaJ0KGLqrlzk"

consumer\_secret = "gIS4fQrYjpREWxi9RrtgiS4vxzPjlNTIuQmnBCizoL06nrhmNu"

access\_token = "1101412887430479872-7YPZMaFXJrR3dRj4BkBHezad9wmJEI"

access\_token\_secret = "1a3sXc1OE892iwiEN9GXLLCB2paKkzR1VDBkyvPSbjjDn"

#This is a basic listener that just prints received tweets to stdout.

class StdOutListener(StreamListener):

def on\_data(self, data):

print data

return True

def on\_error(self, status):

print status

if \_\_name\_\_ == '\_\_main\_\_':

#This handles Twitter authetification and the connection to Twitter Streaming API

l = StdOutListener()

auth = OAuthHandler(consumer\_key, consumer\_secret)

auth.set\_access\_token(access\_token, access\_token\_secret)

stream = Stream(auth, l)

#This line filter Twitter Streams to capture data by the keywords: 'python', 'javascript', 'ruby'

stream.filter(track=['python', 'javascript', 'ruby'])

**Output:**

{"created\_at":"Tue Mar 12 09:23:42 +0000 2019","id":1105398953091035136,"id\_str":"1105398953091035136","text":"RT @karen73984451: #k9hour #gorgeous girlie ready and waiting on a super duper home #Ruby #Itsallaboutthedogs #TeamZay @epsomcanine plz RT\u2026","source":"\u003ca href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for iPhone\u003c\/a\u003e","truncated":false,"in\_reply\_to\_status\_id":null,"in\_reply\_to\_status\_id\_str ":null,"in\_reply\_to\_user\_id":null,"in\_reply\_to\_user\_id\_str":null,"in\_reply\_to\_screen\_name":null ,"user":{"id":2615014568,"id\_str":"2615014568","name":"Titachot","screen\_name":"Titachot"," location":" Chonburi, Thailand","url":null,"description":"Adopt Don't Shop ! Do not support Puppy Mills

**Practical 9**

**Aim:** Write a program to implement simple crawler.

**Source Code:**

import logging

from urllib.parse import urljoin

import requests

from bs4 import BeautifulSoup

logging.basicConfig(

format='%(asctime)s %(levelname)s:%(message)s',

level=logging.INFO)

class Crawler:

def \_\_init\_\_(self, urls=[]):

self.visited\_urls = []

self.urls\_to\_visit = urls

def download\_url(self, url):

return requests.get(url).text

def get\_linked\_urls(self, url, html):

soup = BeautifulSoup(html, 'html.parser')

for link in soup.find\_all('a'):

path = link.get('href')

if path and path.startswith('/'):

path = urljoin(url, path)

yield path

def add\_url\_to\_visit(self, url):

if url not in self.visited\_urls and url not in self.urls\_to\_visit:

self.urls\_to\_visit.append(url)

def crawl(self, url):

html = self.download\_url(url)

for url in self.get\_linked\_urls(url, html):

self.add\_url\_to\_visit(url)

def run(self):

while self.urls\_to\_visit:

url = self.urls\_to\_visit.pop(0)

logging.info(f'Crawling: {url}')

try:

self.crawl(url)

except Exception:

logging.exception(f'Failed to crawl: {url}')

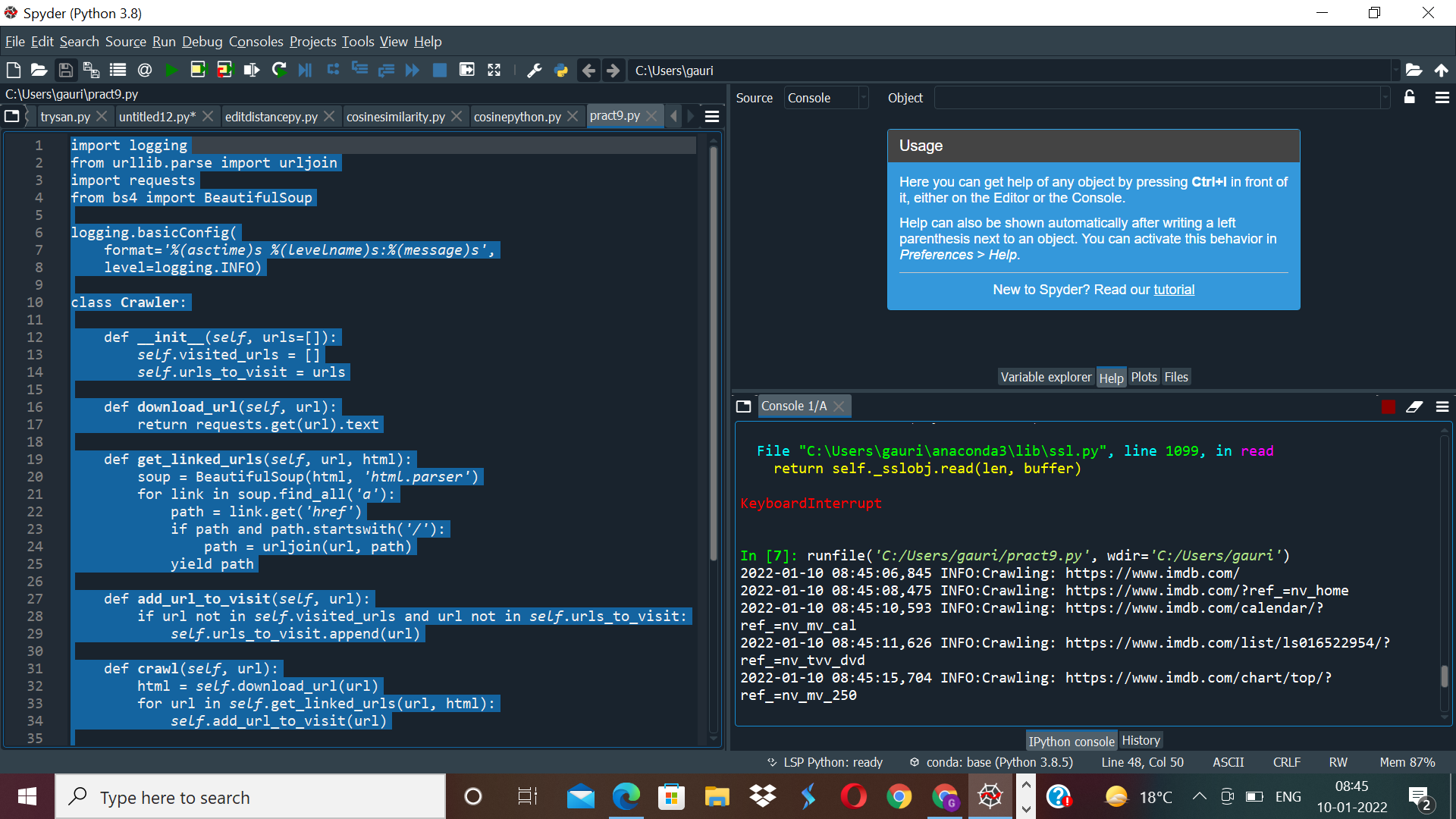
finally:

self.visited\_urls.append(url)

if \_\_name\_\_ == '\_\_main\_\_':

Crawler(urls=['https://www.imdb.com/']).run()

**Output:**



**Practical 10**

**Aim:** Write a program to parse XML text, generate Web graph and compute topic specific page rank.

**movies.xml:**

<collection shelf="New Arrivals">

<movie title="Enemy Behind">

<type>War, Thriller</type>

<format>DVD</format>

<year>2003</year>

<rating>PG</rating>

<stars>10</stars>

<description>Talk about a US-Japan war</description>

</movie>

<movie title="Transformers">

<type>Anime, Science Fiction</type>

<format>DVD</format>

<year>1989</year>

<rating>R</rating>

<stars>8</stars>

<description>A schientific fiction</description>

</movie>

<movie title="Trigun">

<type>Anime, Action</type>

<format>DVD</format>

<episodes>4</episodes>

<rating>PG</rating>

<stars>10</stars>

<description>Vash the Stampede!</description>

</movie>

<movie title="Ishtar">

<type>Comedy</type>

<format>VHS</format>

<rating>PG</rating>

<stars>2</stars>

<description>Viewable boredom</description>

</movie>

</collection>

**Source Code:**

import networkx as nx

import matplotlib.pyplot as plt

from xml.dom.minidom import parse

import xml.dom.minidom

# Open xml document using minidom parser

DOMTree=xml.dom.minidom.parse("movies.xml")

collection=DOMTree.documentElement

if collection.hasAttribute("shelf"):

print "Root element: %s" % collection.getAttribute("shelf")

# get all the movies in the collection

movies = collection.getElementsByTagName("movie")

#print detail of each movie.

for movie in movies:

print"\*\*\*\*\*Movie\*\*\*\*\*"

if movie.hasAttribute("title"):

print"Title: %s" %movie.getAttribute("title")

type = movie.getElementsByTagName('type')[0]

print "Type: %s" % type.childNodes[0].data

format= movie.getElementsByTagName('format')[0]

print "format: %s" % format.childNodes[0].data

rating= movie.getElementsByTagName('rating')[0]

print "Rating: %s" % rating.childNodes[0].data

description=movie.getElementsByTagName('description')[0]

print"description: %s" % description.childNodes[0].data

def GenerateGraph():

G=nx.Graph()

# adding just one node:

G.add\_node("a")

# adding a list of edges:

G.add\_edges\_from([("a","b"),("b","c"), ("c","d"), ("d","a"),("a","c")])

nx.draw(G)

plt.savefig("simple\_path.png") # save as png

plt.show() # display

print("Nodes of graph: ")

print(G.nodes())

print("Edges of graph: ")

print(G.edges())

GenerateGraph()

**Output**

runfile('C:/Users/ckt/.spyder/prct 10 .py', wdir='C:/Users/ckt/.spyder')

Root element: New Arrivals

\*\*\*\*\*Movie\*\*\*\*\*

Title: Enemy Behind

Type: War, Thriller

format: DVD

Rating: PG

description: Talk about a US-Japan war

\*\*\*\*\*Movie\*\*\*\*\*

Title: Transformers

Type: Anime, Science Fiction

format: DVD

Rating: R

description: A schientific fiction

\*\*\*\*\*Movie\*\*\*\*\*

Title: Trigun

Type: Anime, Action

format: DVD

Rating: PG

description: Vash the Stampede!

\*\*\*\*\*Movie\*\*\*\*\*

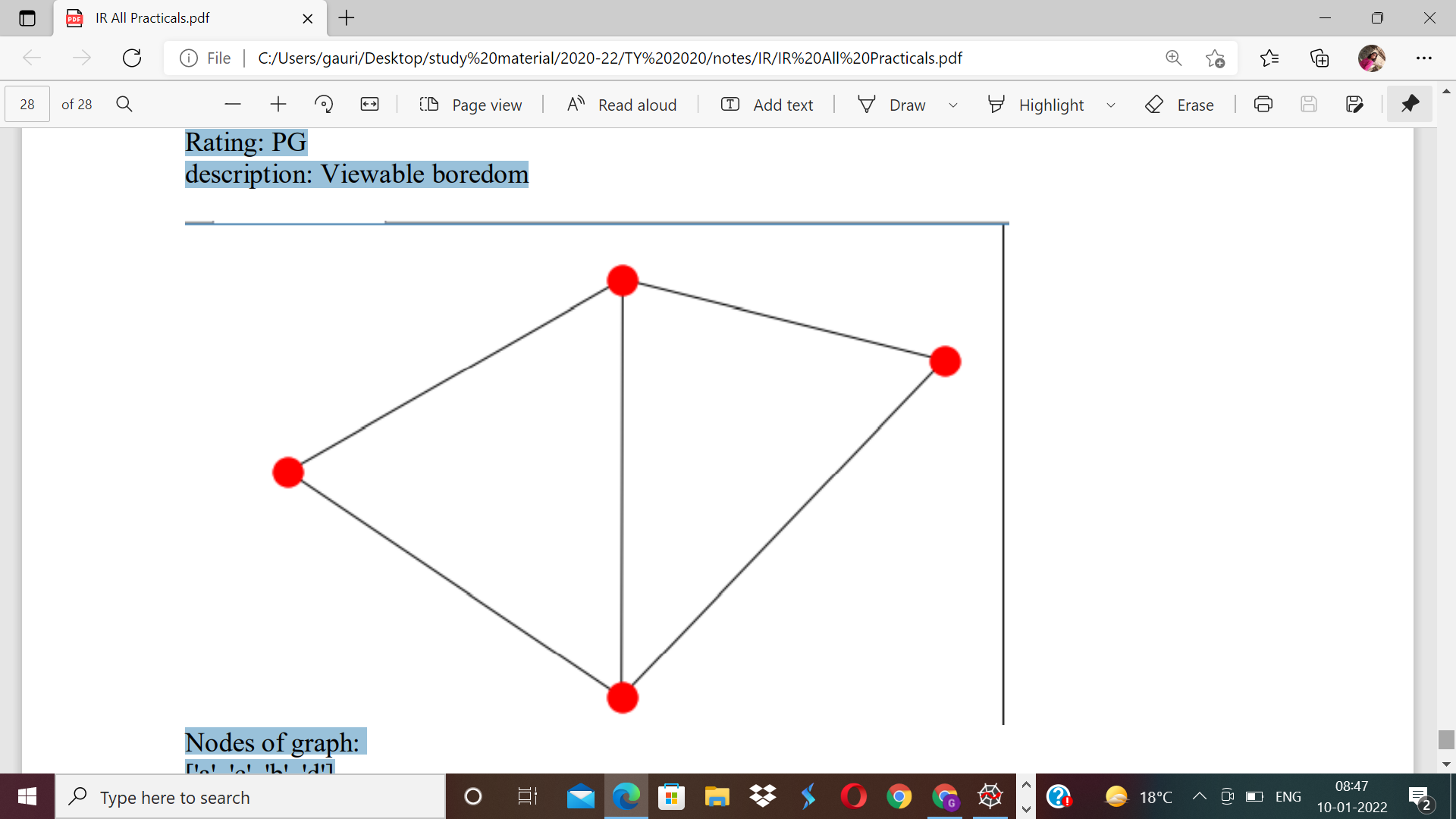
Title: Ishtar

Type: Comedy

format: VHS

Rating: PG

description: Viewable boredom



Nodes of graph:

['a', 'c', 'b', 'd']

Edges of graph:

[('a', 'c'), ('a', 'b'), ('a', 'd'), ('c', 'b'), ('c', 'd')