**Link Analysis**

**Two Distinct Methods of Link Analysis:-**

1. PageRank
2. HITS

PageRank

The PageRank of a node depends on the link structure of the web graph. The PageRank values of pages (and the implicit ordering among them) are independent of any query a user might pose; PageRank is thus a query independent measure of the static quality of each web page

HITS

For a given query, every web page is assigned *two* scores. One is called its *hub score* and the other its *authority* *score.* For any query, we compute two ranked lists of results rather than one.

The ranking of one list is induced by the hub scores and that of the other by the authority scores.

Project Files/Directories:-

1. *HITS.py*
2. *page\_rank.py*
3. *web\_graph.gpickle*

HITS.py

It contains Two Library:

Networkx

Numpy

Import Base\_Li and set\_base from base\_setter file

Import root\_setter file functions as set\_root and list as root\_list

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

page\_rank.py

Uses 2 Library:-

1)Networkx

2)Numpy

File contains 3 functions:-

node\_sorted:-

Sort the nodes of the graph.

**def** **nodes\_sorted**(g, points):

t = np.array(points)

t = np.argsort(-t)

**return** t

page\_rank:-

implement page\_rank algorithm by power method.

**def** **pagerank**(G, alpha=0.90, personalization=None,

max\_iter=100, tol=1.0e-6, nstart=None, weight='weight',

dangling=None):

**if** len(G) == 0:

**return** {}

**if** **not** G.is\_directed():

D = G.to\_directed()

**else**:

D = G

*# Create a copy in (right) stochastic form*

W = nx.stochastic\_graph(D, weight=weight)

N = W.number\_of\_nodes()

*# Choose fixed starting vector if not given*

**if** nstart **is** None:

x = dict.fromkeys(W, 1.0 / N)

**else**:

*# Normalized nstart vector*

s = float(sum(nstart.values()))

x = dict((k, v / s) **for** k, v **in** nstart.items())

**if** personalization **is** None:

Take graph,damping factor, list as personalization, dangling,weight and max\_iter and tol as parameter.

Random Walk:-

Implement Random walk method.

Take Graph g as input and return list of nodes having higher pageRank.

**def** **random\_Walk**(g):

rwp = [0 **for** i **in** range(g.number\_of\_nodes())]

nodes = list(g.nodes())

r = random.choice(nodes)

rwp[r] += 1

neigh = list(g.out\_edges(r))

z = 0

**while** (z != 10000):

**if** (len(neigh) == 0):

focus = random.choice(nodes)

**else**:

r1 = random.choice(neigh)

focus = r1[1]

rwp[focus] += 1

neigh = list(g.out\_edges(focus))

z += 1

**return** rwp

**How to Start Program**

1. **Install Library such as numpy,random,network**
2. **Run pagerank.py file for pagerank algorithm**
3. **It will show the page ranks**
4. **Run hits.py file for hits algorithm**
5. **It will shows hub and authority score**