Aim: To implement Page Rank Algorithm

Objective: Develop a program to implement a page rank algorithm.

Theory:

PageRank (PR) is an algorithm used by Google Search to rank web pages in their search engine results. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. Page Rank Algorithm is designed to increase the effectiveness of search engines and improve their efficiency. It is a way of measuring the importance of website pages. Page rank is used to prioritize the pages returned from a traditional search engine using keyword searching. Page rank is calculated based on the number of pages that point to it. The value of the page rank is the probability will be between 0 and 1. A web page is a directed graph having two important components: nodes and connections. The pages are nodes and hyperlinks are the connections, the connection between two nodes. Page rank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important website are likely to receive more links from other websites. The page rank value of individual node in a graph depends on the page rank value of all the nodes which connect to it and those nodes are cyclically connected to the nodes whose ranking we want; we use converging iterative method for assigning values to page rank. In short page rank is a vote, by all the other pages on the web, about how important a page is. A link to a page count as a vote of support. If there is no link, there is no support.

We assume that page A has pages B.....N which point to it. Page rank of a page A is given as follows:

$PR(A)=(1-\beta) +\beta ((PR(B)/cout(B))+ (PR(C)/cout(C))+-----+(PR(N)/cout(N)))$

Parameter β is a teleportation factor which can be set between 0 and 1. Cout(A) is defined as the number of links going out of page A.

CODE:

```
class PageRank:

    def __init__(self, num_nodes):

        self.num_nodes = num_nodes

        self.path = [[0] * (num_nodes + 1) for _ in range(num_nodes + 1)]

        self.pagerank = [0.0] * (num_nodes + 1)



    def calc(self):
```

```
        initial_pagerank = 1 / self.num_nodes
        damping_factor = 0.85
        temp_pagerank = [0.0] * (self.num_nodes + 1)
        for k in range(1, self.num_nodes + 1):
            self.pagerank[k] = initial_pagerank

        print(f"Total Number of Nodes: {self.num_nodes}, Initial PageRank of All Nodes: {initial_pagerank}\n")
        print("Initial PageRank Values (0th Step):")
        for k in range(1, self.num_nodes + 1):
            print(f"Page Rank of {k} is: {self.pagerank[k]}")

        for iteration in range(1, 11):
            for k in range(1, self.num_nodes + 1):
                temp_pagerank[k] = self.pagerank[k]
                self.pagerank[k] = 0.0

            for internal_node in range(1, self.num_nodes + 1):
                for external_node in range(1, self.num_nodes + 1):
                    if self.path[external_node][internal_node] == 1:
                        outgoing_links = sum(self.path[external_node][j] for j in range(1, self.num_nodes + 1))
                        if outgoing_links > 0:
                            self.pagerank[internal_node] += temp_pagerank[external_node] * (1 / outgoing_links)

            print(f"\nAfter {iteration}th Step:")
```

```
        for k in range(1, self.num_nodes + 1):
            print(f"Page Rank of {k} is: {self.pagerank[k]}")

    for k in range(1, self.num_nodes + 1):
        self.pagerank[k] = (1 - damping_factor) + damping_factor * self.pagerank[k]

    print("\nFinal Page Rank:")
    for k in range(1, self.num_nodes + 1):
        print(f"Page Rank of {k} is: {self.pagerank[k]}")

if __name__ == "__main__":
    nodes = int(input("Enter the Number of WebPages: "))
    page_rank = PageRank(nodes)
    print("Enter the Adjacency Matrix with 1->PATH & 0->NO PATH Between two WebPages:")

    for i in range(1, nodes + 1):
        for j in range(1, nodes + 1):
            value = int(input(f"Path from {i} to {j} (0 or 1): "))
            if i == j:
                page_rank.path[i][j] = 0
            else:
                page_rank.path[i][j] = value

    print("\nAdjacency Matrix:")
    for i in range(1, nodes + 1):
        print(page_rank.path[i][1:nodes + 1])
```

```
page_rank.calc()
```

OUTPUT:

```
Enter the Number of WebPages: 2
Enter the Adjacency Matrix with 1->PATH & 0->NO PATH Between two WebPages:
Path from 1 to 1 (0 or 1): 0
Path from 1 to 2 (0 or 1): 1
Path from 2 to 1 (0 or 1): 1
Path from 2 to 2 (0 or 1): 0

Adjacency Matrix:
[0, 1]
[1, 0]
Total Number of Nodes: 2, Initial PageRank of All Nodes: 0.5

Initial PageRank Values (0th Step):
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

After 1th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5


After 2th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

After 3th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

After 4th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

After 5th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

After 6th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5
```

```
After 7th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

After 8th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

After 9th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

After 10th Step:
Page Rank of 1 is: 0.5
Page Rank of 2 is: 0.5

Final Page Rank:
Page Rank of 1 is: 0.575
Page Rank of 2 is: 0.575
```

Conclusion:

In this practical, we implemented the PageRank algorithm to evaluate the importance of web pages based on their interconnections. By using an adjacency matrix to represent the links between pages, we observed how PageRank values are influenced by both the number and quality of incoming links. The iterative approach allowed us to refine these values, demonstrating the algorithm's effectiveness in ranking web pages.

What are the key parameters of the PageRank algorithm, and how do they affect the algorithm's performance?

Damping Factor (β): he damping factor, typically set between 0 and 1 (commonly around 0.85), represents the probability that a user continues navigating from one page to another. A value of 1 would mean the user always follows links, while a value of 0 means they randomly jump to any page. A higher damping factor emphasizes the influence of incoming links, making the algorithm more sensitive to the number of links pointing to a page. A lower value can result in more uniform PageRank distributions, as it reduces the impact of linking structures.

Number of Iterations: **Description**: This parameter defines how many times the algorithm will update the PageRank values of the nodes (web pages). **Effect**: More iterations typically lead to more accurate PageRank values as the algorithm converges towards a stable solution. However, too many iterations can lead to diminishing returns in terms of performance improvement and increased computational time.

Initial PageRank Values: This refers to the initial value assigned to each page before the iterative calculation begins. It is often set uniformly (e.g., 1/total number of nodes1 / \text{total number of nodes}1/total number of nodes). While the choice of initial values can affect the convergence speed of the algorithm, the final PageRank values are generally not sensitive to these initial conditions after several iterations.

Adjacency Matrix (Graph Structure): The structure of the graph, represented as an adjacency matrix, defines how pages are linked to each other. The configuration of links (both in quantity and quality) directly affects PageRank calculations. Well-connected nodes tend to accumulate higher PageRank scores, while poorly connected nodes may receive lower scores.

Link Quality: This refers to the importance or relevance of the pages linking to a particular page. If the algorithm is modified to consider the quality of links (e.g., assigning different weights to links based on the authority of the linking page), it can produce more meaningful rankings. High-quality links will contribute more to the PageRank of the target page.

Convergence Threshold: A predefined limit that determines when the PageRank values have stabilized and further iterations are unnecessary. Setting a proper convergence threshold can optimize performance, as it avoids unnecessary calculations once the values have sufficiently stabilized.