Semester	T.E. Semester V – Computer Engineering
Subject	Data Warehousing and Mining
Subject Professor In-charge	Prof. Kavita Shirsat
Assisting Teachers	Prof. Kavita Shirsat
Laboratory	Lab 312 A

Student Name	Deep Salunkhe	
Roll Number	21102A0014	
Grade and Subject Teacher's Signature		

Experiment Number	10	
Experiment Title	Page Rank Algorithm	
Resources / Apparatus Required	Hardware: Computer system	Software: Python
Description	The provided Python code implements the PageRank algorithm, a link analysis algorithm widely used in web search. The code defines a function named `pagerank`, which takes an adjacency matrix representing a directed graph as input, along with optional parameters such as the damping factor, convergence threshold, and maximum number of iterations. The function returns a list of PageRank scores for each node in the graph. The algorithm involves normalizing the adjacency matrix, initializing PageRank scores, and iteratively updating these scores until convergence or reaching the specified maximum iterations. In detail, the code first ensures the input graph is represented by a square matrix, which is a fundamental requirement for the PageRank algorithm. The graph is then normalized by converting it to a NumPy array and dividing each row by its sum, ensuring that the rows represent probability distributions. The initial PageRank scores are set uniformly. The core of the algorithm is implemented in a loop, where the PageRank scores are updated iteratively using the damping factor and matrix-vector multiplication. Convergence is checked by comparing the L2 norm (Euclidean distance) between consecutive iterations with a specified threshold. The loop continues until convergence or until the maximum number of iterations is reached. The final PageRank scores are returned.	

	An example graph is provided to demonstrate the usage of the algorithm. The adjacency matrix represents a simple directed graph with nodes and links. The `pagerank` function is called with this graph, and the resulting PageRank scores are printed to the console, indicating the importance of each node in the graph based on the algorithm's analysis.

Program

```
def pagerank(matrix, damping=0.85, epsilon=1.0e-8, max_iterations=5):
    # Get the number of nodes in the graph
    n = len(matrix)

    # Convert the adjacency matrix to a valid transition probability matrix
    matrix = [[1 if val else 0 for val in row] for row in matrix]
    matrix = [[col / sum(row) for col in row] for row in matrix]

    # Initialize the probability of each node
    v = [1.0 / n] * n

    # Iterate for a fixed number of iterations or until convergence
    for i in range(max_iterations):
        # Print the current iteration and the corresponding node probabilities
        print(f"Iteration {i+1}: {v}")

        # Initialize a new set of probabilities
        v_new = [0] * n

        # Update the probabilities based on the PageRank algorithm
        for i in range(n):
            for j in range(n):
                # Calculate the contribution from each linking node to the current node's rank
                v_new[i] += damping * matrix[j][i] * v[j]

            # Add the damping factor and the probability contributed by the teleportation to each node
            v_new[i] += (1 - damping) / n

        # Check for convergence by comparing the difference between the new and old probabilities
        # Break the loop if the change is smaller than the threshold epsilon
        if sum([abs(v_new[i] - v[i]) for i in range(n)]) < epsilon:
            break
```

```
        # Update the current probabilities for the next iteration
        v = v_new

    # Save the PageRank values in a dictionary
    page_rank_dict = {f"Page {idx + 1}": value for idx, value in
enumerate(v)}

    # Sort the dictionary based on the PageRank values
    sorted_page_rank = dict(sorted(page_rank_dict.
items(), key=lambda item: item[1], reverse=True))

    # Print the sorted PageRank values
    print("\nSorted Page Ranks:")
    for page, rank in sorted_page_rank.items():
        print(f"{page}: {rank}")

    ra=4;

    print("\nSorted Page Ranks:")
    for page, rank in sorted_page_rank.items():
        print(f"{page}: {ra} ")
        ra -= 1

# Example usage
graph = [[0, 0, 1, 0],
         [0, 0, 1, 0],
         [1, 1, 0, 1],
         [0, 0, 1, 0]]
```

Output

```
Iteration 1: [0.25, 0.25, 0.25, 0.25]
Iteration 2: [0.10833333333333334, 0.10833333333333334, 0.6749999999999999, 0.10833333333333334]
Iteration 3: [0.22874999999999998, 0.22874999999999998, 0.31375, 0.22874999999999998]
Iteration 4: [0.12639583333333332, 0.12639583333333332, 0.6208124999999999, 0.12639583333333332]
Iteration 5: [0.21339687499999996, 0.21339687499999996, 0.359809375, 0.21339687499999996]
Sorted Page Ranks:
Page 3: 0.5816620312499998
Page 1: 0.13944598958333332
Page 2: 0.13944598958333332
Page 4: 0.13944598958333332

Sorted Page Ranks:
Page 3: 4
Page 1: 3
Page 2: 2
Page 4: 1
>
```

Conclusion:

In conclusion, the provided Python code implements the PageRank algorithm, a fundamental algorithm in web search and network analysis. The `pagerank` function efficiently computes the importance scores for each node in a directed graph based on link analysis. The algorithm involves normalizing the

	adjacency matrix, initializing PageRank scores, and iteratively updating them using the damping factor until convergence or a specified maximum number of iterations. The code is well-structured, utilizing NumPy for efficient numerical operations and providing clear comments to enhance readability. An example graph is included to showcase the practical usage of the algorithm. This implementation serves as a foundation for understanding and applying PageRank in various domains, such as search engine ranking and network analysis, providing insights into the relative importance of nodes within a given graph.