PageRank

1. Mục tiêu:

Sinh viên tìm hiểu, cài đặt, và thử nghiệm thuật toán PageRank. Qua bài học này sinh viên sẽ nắm được:

- Cách cài đặt thuật toán PageRank một cách đơn giản nhất.
- Các vấn đề gặp phải của thuật toán PageRank: Dead end, Spider trap, Spam farm.
- Cách cải tiến thuật toán PageRank để giải quyết các vấn đề phía trên.
- Tìm hiểu về bộ thư viện NetworkX. So sánh kết quả của cách cài đặt PageRank phía trên với thuật toán PageRank trong bộ thư viện này.

2. Nội dung:

Sinh viên viết chương trình đọc đồ thị Web từ file txt. Tiến hành thuật toán PageRank để lấy ra chỉ số PageRank của các đỉnh. Sau đó xuất ra file output.

- Đầu vào: File txt theo định dạng đính kèm
- Xử lý: tính giá trị PageRank của tất cả các đỉnh trong đồ thị cung cấp bởi file Input.

Yêu cầu nộp bài:

Sinh viên nộp file nén MSSV.rar/zip trong đó bao gồm:

+ File source code: MSSV.py
+ File Readme: Ghi chú các hướng dẫn để chạy chương trình.

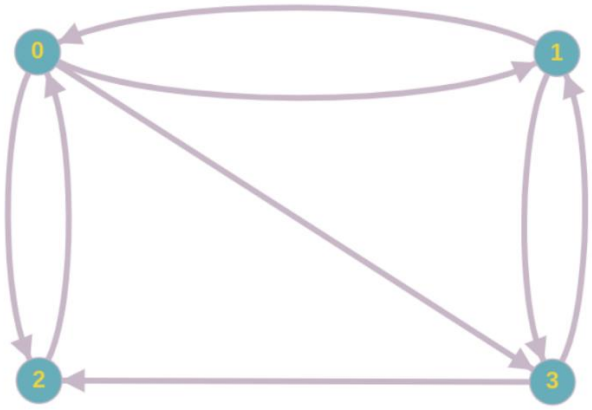
Sinh viên cần chuẩn bị một thư mục các file test và sẵn sàng chạy chương trình khi giáo viên yêu cầu.

3. Hướng dẫn thực hiện

Phần 1 – Cài đặt và thử nghiệm với PageRank

1. **Biểu diễn đồ thị:**

   Có nhiều cách để biểu diễn đồ thị trong Python. Trong ví dụ này, chúng ta dùng dictionary.



Hình 1 - Toy Graph (Đồ thị ví dụ)

Đồ thị trên được biểu diễn trong Python như sau:

```
def toy_graph():
    G = dict()
    G[0] = [1, 2, 3]
    G[1] = [0, 3]
    G[2] = [0]
    G[3] = [1, 2]
    return G
```

2. **Thuật toán PageRank giản lược**

   Thuật toán PageRank sử dụng mô hình Markov (Sinh viên tham khảo lý thuyết)

```
def pagerank(G, iteration_count=100):

    N = len(G.keys())
    next_rank_lst = [1/N for _ in range(N)]
    current_rank_lst = next_rank_lst[:]

    for i in range(iteration_count):
        current_rank_lst, next_rank_lst = next_rank_lst, current_rank_lst
        for j in range(N):
            next_rank_lst[j] = 0
        for node in G:
            if G[node]: # Để tránh vấn đề chia cho 0 trong dòng kế
                contribution = current_rank_lst[node] / len(G[node])
                for edge in G[node]:
                    next_rank_lst[edge] += contribution

    return next_rank_lst
```

Thử nghiệm với Toy Graph

```
G = toy_graph()
rank_lst = pagerank(G)
print(rank_lst)
print(sum(rank_lst)) # Tổng rank bằng 1
```

3. Dead End và Spider Trap

Dead End: Là những node không có cạnh ra, và không có nơi nào đến node này gửi đi độ quan trọng của nó. Cuối cùng, toàn bộ độ quan trong sẽ thoát ra khỏi Web.

Bằng cách xóa đi các cạnh ra của node 2 trong Toy Graph, node này sẽ trở thành Dead End

```
G = toy_graph()
G[2].clear()

rank_lst = pagerank(G)
print(rank_lst)
print(sum(rank_lst))
```

PageRank của các node sẽ là một con số rất nhỏ (gần bằng 0)

Spider Trap: một nhóm các trang web không có cạnh ra khỏi nhóm này. Nhóm này sẽ dần dần hút hết độ quan trong của Web

Trong ví dụ sau, node 2 sẽ hút hết độ quan trọng

```
G = toy_graph()
G[2] = [2]

rank_lst = pagerank(G)
print(rank_lst)
print(sum(rank_lst))
```

Trong ví dụ sau, node 2 và 4, sẽ hút hết độ quan trọng

```
G = toy_graph()
G[2] = [4]
G[4] = [2]

rank_lst = pagerank(G)
print(rank_lst)
print(sum(rank_lst))
```

4. Thuật toán PageRank với Random Teleportation

Thuật toán PageRank cải tiến sau sẽ giải quyết được những vấn đề xảy ra trong trường hợp đồ thị có Dead End hoặc Spider Trap

```
def pagerank(G, beta=0.85, iteration_count=100):

    N = len(G.keys())
    next_rank_lst = [1/N for _ in range(N)]
    current_rank_lst = next_rank_lst[:]

    for i in range(iteration_count):
        current_rank_lst, next_rank_lst = next_rank_lst, current_rank_lst
        for j in range(N):
            next_rank_lst[j] = (1 - beta) / N
        for node in G:
            if G[node]:
                contribution = beta * (current_rank_lst[node] / len(G[node]))
                for edge in G[node]:
                    next_rank_lst[edge] += contribution

        leakage_contribution = (1 - sum(next_rank_lst)) / N
        for j in range(N):
            next_rank_lst[j] += leakage_contribution

    return next_rank_lst
```

Sinh viên thử nghiệm áp dụng thuật toán này cho các ví dụ gây ra lỗi ở phần 3 (Dead End và Spider Trap)

5. Spam Farm

Đây là vấn đề xảy ra khi một nhóm các spammer tạo ra rất nhiều trang web rác và liên kết với nhau để boost PageRank một cách bất thường.

```
G = toy_graph()
G[2].clear()
for i in range(len(G.keys()), 100):
    G[i] = [2]
    G[2].append(i)

rank_lst = pagerank(G, beta=0.8)
print(rank_lst[:4])
```

6. Trust Rank

Đây là kỹ thuật phân tính link nhằm chống lại vấn đề Spam Farm.

```
def pagerank(G, beta=0.85, iteration_count=100, teleport_lst=None):

    if not teleport_lst:
        teleport_lst = G.keys()

    N = len(G.keys())
    next_rank_lst = [1/N for _ in range(N)]
    current_rank_lst = next_rank_lst[:]

    teleport_lst_count = len(teleport_lst)

    for i in range(iteration_count):
        current_rank_lst, next_rank_lst = next_rank_lst, current_rank_lst
        for j in range(N):
            next_rank_lst[j] = 0
        for node in teleport_lst:
            next_rank_lst[node] = (1 - beta) / teleport_lst_count
        for node in G:
            if G[node]:
                contribution = beta * (current_rank_lst[node] / len(G[node]))
                for edge in G[node]:
                    next_rank_lst[edge] += contribution

        leakage_contribution = (1 - sum(next_rank_lst)) / N
        for j in range(N):
            next_rank_lst[j] += leakage_contribution

    return next_rank_lst
```

Thử nghiệm chống Spam Farm:

```
G = toy_graph()
G[2].clear()
for i in range(len(G.keys()), 100):
    G[i] = [2]
    G[2].append(i)

trust_lst = [1]
rank_lst = pagerank(G, beta=0.8, teleport_lst=trust_lst)
print(rank_lst[:4])
```

7. PageRank với điều kiện dừng.

Trong các phần trước, thuật toán PageRank luôn lặp theo một số lượng nhất định trước khi dừng (iteration_count). Trên thực tế, thuật toán PageRank có thể được dừng khi đạt trạng thái hội tụ: PageRank thay đổi giữa 2 lần lặp gần nhất nhỏ hơn một giá trị $\boldsymbol{\varepsilon}$ cho trước.

```
def pagerank(G, beta=0.85, iteration_count=100, teleport_lst=None, eps=1e-8):

    if not teleport_lst:
        teleport_lst = G.keys()

    N = len(G.keys())
    next_rank_lst = [1/N for _ in range(N)]
    current_rank_lst = next_rank_lst[:]

    teleport_lst_count = len(teleport_lst)

    for i in range(iteration_count):
        current_rank_lst, next_rank_lst = next_rank_lst, current_rank_lst
        for j in range(N):
            next_rank_lst[j] = 0
        for node in teleport_lst:
            next_rank_lst[node] = (1 - beta) / teleport_lst_count
        for node in G:
            if G[node]:
                contribution = beta * (current_rank_lst[node] / len(G[node]))
                for edge in G[node]:
                    next_rank_lst[edge] += contribution

        leakage_contribution = (1 - sum(next_rank_lst)) / N
```

```
        for j in range(N):
            next_rank_lst[j] += leakage_contribution

        total_diff = 0
        for c, n in zip(current_rank_lst, next_rank_lst):
            total_diff += abs(c - n)

        if total_diff < eps:
            return next_rank_lst

    return next_rank_lst
```

Phần 2: PageRank với bộ thư viện NetworkX

NetworkX là một bộ thư viện Python hỗ trợ rất nhiều chức năng và thuật toán liên quan đến xử lý đồ thị.

Sinh viên có thể tham khảo cách sử dụng bộ thư viện này ở đây.

NetworkX cũng hỗ trợ hàm để tính PageRank. Sinh viên có thể xem chi tiết về các tham số và các vấn đề khác của thuật toán PageRank hỗ trợ bởi NetworkX ở đây.

Sinh viên thử nghiệm chạy thuật toán HITS của bộ thư viện NetworkX và so sánh với kết quả của phần 1.

Ví dụ sau sẽ minh họa cách sử dụng trong trường hợp đơn giản.

```
import networkx as nx
# Khởi tạo đồ thị có hướng
G = nx.DiGraph()

# Thêm cạnh
G.add_edge(1,6)
G.add_edge(2,6)
G.add_edge(3,6)
G.add_edge(4,6)
G.add_edge(5,6)
G.add_edge(4,5)
G.add_edge(6,7)

# Tính PageRank
pr = nx.pagerank(G)
print(pr)
```