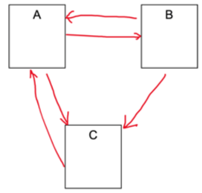
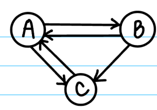
1. Given the following three web pages A, B, and C, with the corresponding links.



1) How many backlinks and forward links does each web page (A, B, C) have?

	A	B	C
Backlinks	2 (from B and C)	1 (from A)	2 (from A and C)
Forward links	2 (to B and C)	2 (to A and C)	1 (to A)

2) Using the simplified version of the PageRank algorithm, assume C=1, calculate the page ranks of each web page (A, B, C). How many iterations does it take for the page ranks to converge? Show your work. (Note: Show your work manually for the 1st and 2nd iterations. Then write a program to do all iterations and print out the converged value of each web page).



$$R_0 = \begin{vmatrix} 1 \\ 1 \\ 1 \end{vmatrix} \qquad M = \begin{array}{c|ccc|} & A & B & C \\ A & 0 & 1/2 & 1 \\ B & 1/2 & 0 & 0 \\ C & 1/2 & 1/2 & 0 \end{array}$$

$*$ Iteration 1: $R_1 = CMR_0 = MR_0 = \begin{vmatrix} 0 & 1/2 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 \end{vmatrix} \begin{vmatrix} 1 \\ 1 \\ 1 \end{vmatrix} = \begin{vmatrix} 3/2 \\ 1/2 \\ 1 \end{vmatrix}$

$*$ Iteration 2: $R_2 = CMR_1 = MR_1 = \begin{vmatrix} 0 & 1/2 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 \end{vmatrix} \begin{vmatrix} 3/2 \\ 1/2 \\ 1 \end{vmatrix} = \begin{vmatrix} 5/4 \\ 3/4 \\ 1 \end{vmatrix}$

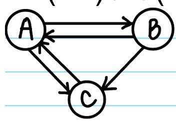
Users > nhl > test.py > ...

```python
import numpy as np

def calculate_pagerank(M, num_iterations=20, tolerance=1e-6):
    n = M.shape[0]
    r = np.ones(n) # Initialize r0 with equal probabilities
    results = [r.copy()]

    for i in range(num_iterations):
        r_next = M @ r
        results.append(r_next.copy())
        if np.allclose(r, r_next, rtol=tolerance): # Check for convergence
            break
        r = r_next
    return results

M = np.array([
    [0, 1/2, 1],
    [1/2, 0, 0],
    [1/2, 1/2, 0]
])

iterations = calculate_pagerank(M)

print("PageRank Iterations:")
print("-" * 40)
for i, r in enumerate(iterations):
    print(f"Iteration {i}:")
    for j, value in enumerate(r):
        page = chr(65 + j)  # Convert 0,1,2 to A,B,C
        print(f"Page {page}: {value:.6f}")
    print()

print("Final converged values:")
print("-" * 40)
final_values = iterations[-1]
for i, value in enumerate(final_values):
    page = chr(65 + i)
    print(f"Page {page}: {value:.6f}")
```

TERMINAL

```
Page A: 1.333008
Page B: 0.666992
Page C: 1.000000

Iteration 11:
Page A: 1.333496
Page B: 0.666504
Page C: 1.000000

Iteration 12:
Page A: 1.333252
Page B: 0.666748
Page C: 1.000000

Iteration 13:
Page A: 1.333374
Page B: 0.666626
Page C: 1.000000

Iteration 14:
Page A: 1.333313
Page B: 0.666687
Page C: 1.000000

Iteration 15:
Page A: 1.333344
Page B: 0.666656
Page C: 1.000000

Iteration 16:
Page A: 1.333328
Page B: 0.666672
Page C: 1.000000

Iteration 17:
Page A: 1.333336
Page B: 0.666664
Page C: 1.000000

Iteration 18:
Page A: 1.333332
Page B: 0.666668
Page C: 1.000000

Iteration 19:
Page A: 1.333334
Page B: 0.666666
Page C: 1.000000

Iteration 20:
Page A: 1.333333
Page B: 0.666667
Page C: 1.000000

Final converged values:
----------------------------------------
Page A: 1.333333
Page B: 0.666667
Page C: 1.000000
```

3) Using the modified version of the PageRank algorithm, manually calculate the page rank of each webpage for the 1st iteration (assuming d=0.8, initially the PageRank rating for each page is 1). (Note: you may use the following equation.) PR(A) = (1-d) + d (PR(T1) / C(T1) + ... + PR(Tn) / C(Tn))



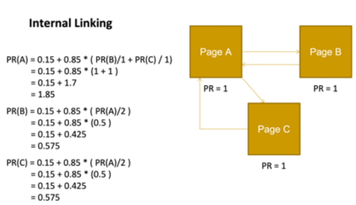
$R_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ $\qquad d = 0.8$

$PR(A) = (1-d) + d\left(\frac{PR(B)}{N_B} + \frac{PR(C)}{N_C}\right) = 1 - 0.8 + 0.8\left(\frac{1}{2} + \frac{1}{1}\right) = 1.4$

$PR(B) = (1-d) + d\left(\frac{PR(A)}{N_A}\right) = 1 - 0.8 + 0.8\left(\frac{1}{2}\right) = 0.6$

$PR(C) = (1-d) + d\left(\frac{PR(A)}{N_A} + \frac{PR(B)}{N_B}\right) = 1 - 0.8 + 0.8\left(\frac{1}{2} + \frac{1}{2}\right) = 1$

2. The following figure is an example we explained in class. The page rank values of each web page after the 1st iteration are given in the figure (PR(A), PR(B), and PR(C). Please continue to calculate the page rank values of these three web pages for the 2nd iteration manually. Write a program to do all iterations and print out the converged value of each web page.



* Iteration 1: PR(A) = 1.85 PR(B) = 0.575 PR(C) = 0.575

* Iteration 2:

$PR(A) = 1 - d + d\left(\frac{PR(B)}{N_B} + \frac{PR(C)}{N_C}\right) = 0.15 + 0.85\left(\frac{0.575}{1} + \frac{0.575}{1}\right) = 1.1275$

$PR(B) = 1 - d + d\left(\frac{PR(A)}{N_A}\right) = 0.15 + 0.85\left(\frac{1.85}{2}\right) = 0.93625$

$PR(C) = 1 - d + d\left(\frac{PR(A)}{N_A}\right) = 0.15 + 0.85\left(\frac{1.85}{2}\right) = 0.93625$

```
Users > nhl > test.py > ...
# 2 - Modified PageRank
def calculate_pagerank_with_damping(links, damping=0.85, num_iterations=50, tolerance=1e-6):
    n = len(links)
    # Initialize PageRank scores with 1/n
    scores = {page: 1.0 for page in links}
    iterations = [{page: score for page, score in scores.items()}]

    for iteration in range(num_iterations):
        new_scores = {}
        max_change = 0

        for page in links:
            # Calculate sum of contributions from incoming links
            incoming_pr = sum(scores[src_page] / len(links[src_page])
                            for src_page, targets in links.items()
                            if page in targets)

            # Apply damping factor formula: (1-d) + d * (sum of contributions)
            new_score = (1 - damping) + damping * incoming_pr
            max_change = max(max_change, abs(new_score - scores[page]))
            new_scores[page] = new_score

        # Store this iteration's results
        iterations.append({page: score for page, score in new_scores.items()})
        scores = new_scores

        # Check for convergence
        if max_change < tolerance:
            break

    return iterations

links = {
    'A': ['B', 'C'],  # Page A links to B and C
    'B': ['A'],       # Page B links to A
    'C': ['A']        # Page C links to A
}

# Calculate PageRank
iterations = calculate_pagerank_with_damping(links)

# Print results for each iteration
print("\nModified PageRank Iterations:")
print("-" * 50)
for i, scores in enumerate(iterations):
    print(f"\nIteration {i}:")
    for page, score in sorted(scores.items()):
        print(f"PR({page}) = {score:.6f}")

print("\nFinal converged values:")
print("-" * 50)
final_scores = iterations[-1]
for page, score in sorted(final_scores.items()):
    print(f"PR({page}) = {score:.6f}")

# Print the number of iterations needed for convergence
print(f"\nConverged after {len(iterations)-1} iterations")
```

```
TERMINAL
PR(B) = 0.769708
PR(C) = 0.769708

Iteration 38:
PR(A) = 1.458504
PR(B) = 0.770748
PR(C) = 0.770748

Iteration 39:
PR(A) = 1.460272
PR(B) = 0.769864
PR(C) = 0.769864

Iteration 40:
PR(A) = 1.458769
PR(B) = 0.770615
PR(C) = 0.770615

Iteration 41:
PR(A) = 1.460046
PR(B) = 0.769977
PR(C) = 0.769977

Iteration 42:
PR(A) = 1.458961
PR(B) = 0.770520
PR(C) = 0.770520

Iteration 43:
PR(A) = 1.459883
PR(B) = 0.770058
PR(C) = 0.770058

Iteration 44:
PR(A) = 1.459099
PR(B) = 0.770450
PR(C) = 0.770450

Iteration 45:
PR(A) = 1.459766
PR(B) = 0.770117
PR(C) = 0.770117

Iteration 46:
PR(A) = 1.459199
PR(B) = 0.770400
PR(C) = 0.770400

Iteration 47:
PR(A) = 1.459681
PR(B) = 0.770160
PR(C) = 0.770160

Iteration 48:
PR(A) = 1.459271
PR(B) = 0.770364
PR(C) = 0.770364

Iteration 49:
PR(A) = 1.459619
PR(B) = 0.770190
PR(C) = 0.770190

Iteration 50:
PR(A) = 1.459324
PR(B) = 0.770338
PR(C) = 0.770338

Final converged values:
--------------------------------------------------
PR(A) = 1.459324
PR(B) = 0.770338
PR(C) = 0.770338
```

3. Explain dangling links and how to prevent the negative effect of them.
- Definition: Dangling links (or dangling nodes) are links that point to any page with no outgoing links. They act like "rank sinks" - when the random surfer reaches these pages, the PageRank score gets trapped and isn't distributed further, which can distort the overall PageRank calculations.
- Resolution:
    i. Add virtual links from dangling nodes to all other pages with equal probability
    ii. Add a link from the dangling node back to itself
    iii. Remove pages without outbound links until the PageRank values are computed