# mad-prac-8-page-rank

June 28, 2024

### 0.0.1 PRACTICAL 8 PAGE RANK Define a Web graph for the given web page table and perform the following: (i) assign them the outgoing-link number = (1 / number of hyperlinks.) to each page. (ii) Take the appropriate damping factor d, recursively compute the new page ranks. (iii) Take the threshold say 0.001, 0.005 to find the converged iteratively solution. (iv) Compute the final page ranks and display the high ranked pages on the top.

To implement the PageRank algorithm in Python, we need a way to represent the web graph, which is the network of web pages and links between them. One common way to do this is to use a dictionary, where the keys are the page names, and the values are lists of pages that link to them. For example, if we have four pages A, B, C, and D, and the links are: A -> B, C B -> A, D C -> A D -> B, C

[1]:
```
web_graph = {'A': ['B', 'C'],'B': ['A', 'D'], 'C': ['A'],'D': ['B', 'C']}
```

To start the PageRank algorithm, we need to assign some initial values to each page. A simple way to do this is to assign a uniform value of 1 / n, where n is the number of pages in the web graph. For example, if we have four pages, we can initialize the PageRank values by 1/4 = 0.25.

[2]:
```
page_rank = {'A': 0.25,'B': 0.25,'C': 0.25,'D': 0.25}
```

PageRank is a way of assigning a numerical value to each web page, based on how many other pages link to it, and how important those pages are. The idea is that a page is more important if it is linked by many other pages, and especially by other important pages. The PageRank algorithm uses a recursive formula to calculate the PageRank of each page, based on the PageRank of the pages that link to it. The formula is: PR(A) = (1 - d) + d * (PR(B) / L(B) + PR(C) / L(C) + ... + PR(N) / L(N)) where PR(A) is the PageRank of page A, d is a damping factor (usually set to 0.85), L(B) is the number of outgoing-links from page B, and PR(B) / L(B) is the contribution of page B to the PageRank of page A. The formula is applied iteratively until the PageRank values converge to a stable state.

[3]:
```
d = 0.85 # damping factor
new_page_rank = {} # variable to store the new values
for page in web_graph: # loop over the web graph
  sum = 0
  # sum of contributions from other pages
  for other_page in web_graph:
    # loop over the other pages
    if page in web_graph[other_page]:
```

```
      # check if the other page links to the current page
      sum += page_rank[other_page] / len(web_graph[other_page])
      # add the contribution of the other page
  new_page_rank[page] = (1 - d) + d * sum
  # calculate the new PageRank value using the formula
page_rank = new_page_rank
# update the PageRank values
```

To update the PageRank values, we need to apply the formula for each page, using the current values of the other pages. We can use a loop to iterate over the web graph, and a variable to store the new values. We also need to use the damping factor, which is usually set to 0.85, to account for the possibility of random jumps from one page to another.

To check if the PageRank values have converged to a stable state, we need to compare the old and new values, and see if they are close enough. We can use a threshold, which is a small number that indicates how much difference we can tolerate. For example, we can use a threshold of 0.001, and check for convergence.

[4]:
```
threshold = 0.001
# threshold for convergence
converged = True
# flag to indicate if the PageRank values have converged
for page in page_rank:
  # loop over the PageRank values
  if abs(page_rank[page] - new_page_rank[page]) > threshold:
    # check if the difference is larger than the threshold
    converged = False
    # set the flag to False
    break
    # exit the loop
if converged:
  # if the flag is True
  print("The PageRank values are converged. Hence Iteration Process stopped")
  # print a message
else:
  # if the variable is False
  print("The PageRank values are not converged yet")
  # print a message
```

```
The PageRank values are converged. Hence Iteration Process stopped
```

To rank the web pages, we need to sort them by their PageRank values, from highest to lowest. We can use a built-in function in Python, called sorted, which takes a list or a dictionary, and returns a sorted version of it. It returns a new sorted list from the items in iteratively in “iterable” It has two optional arguments which must be specified as keyword arguments(key, reverse). key specifies a function of one argument that is used to extract a comparison key from each element in iterable (for example, key=str.lower). The default value is None (compare the elements directly). reverse is a boolean value. If set to True, then the list elements are sorted in descending order. We can also use a lambda function, which is a short and anonymous function, to specify the sorting key,

which is the PageRank value.

[5]:
```
ranked_pages = sorted(page_rank.items(), key=lambda x: x[1], reverse=True)
# sort the PageRank values by descending order
print('Ranks for the Pages A,B,C,D are:')
for page, rank in ranked_pages:
  # loop over the sorted list
  print(page, rank)
  # print the page name and the PageRank value
```

```
Ranks for the Pages A,B,C,D are:
A 0.46875
B 0.36250000000000004
C 0.36250000000000004
D 0.25625000000000003
```

[ ]:
```

```