CSCI803 Assignment

Yao Xiao
SID 2019180015

December 8, 2020

1 Problem Description

In fact, we are inseparable from the web page every day. How to efficiently and accurately give the ranking of the web page or give the relevant page list is what I want to discuss and solve.

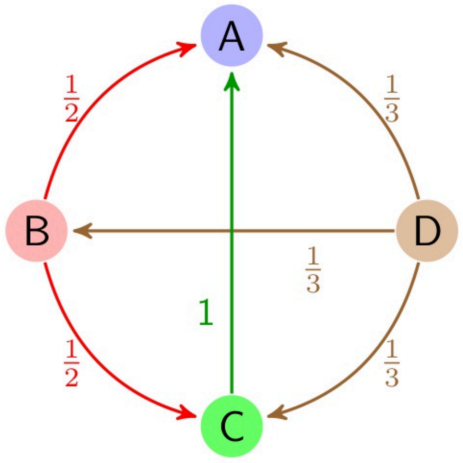
2 Solution

2.1 Strategy 1

First of all, give the core idea.

1. If a page is linked by many other pages, it means that the page is more important, that is, the PageRank value will be relatively high.
2. If a page with a high PageRank value links to another page, the PageRank value of the linked page will increase accordingly.

The Internet can be regarded as a directed graph, each web page is a node of the digraph, and each hyperlink between the web pages is an edge of the digraph.



The R value of a will be the sum of the R values of B, C and D pages, but other nodes still have links

to different nodes. So we can get the formula:

$$\begin{aligned} R(A) &= \frac{R(B)}{2} + \frac{R(C)}{2} + \frac{R(D)}{3} \\ R(u) &= \sum_{v \in B_u} \frac{R(v)}{L(v)} \end{aligned} \tag{1}$$

B_u is a collection of all web pages linked to web page u, web page v is a web page belonging to the collection B_u, and $L(v)$ is the number of external links to the web page v (that is, out-degree).

In the Internet, a web page only has its own chain, or a few pages out of the chain to form a circle. Then, in the process of continuous iteration, the R value of one or several pages will only increase but not decrease. So we can further define it as:

$$R(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{R(p_j)}{L_n(p_j)} \tag{2}$$

d is called damping factor, which means the probability that a user reaches a page and continues to browse backward at any time. The matrix form of the above formula is:

$$\mathbf{R} = \begin{bmatrix} \frac{1-d}{N} \\ \vdots \\ \frac{1-d}{N} \end{bmatrix} + d \begin{bmatrix} l_{1,1} & \cdots & l_{1,n} \\ \vdots & \ddots & \vdots \\ l_{n,1} & \cdots & l_{n,n} \end{bmatrix} \mathbf{R}$$

Then through $R_{n+1} = AR_n$ iterate continuously the PR value, when the following inequality is satisfied, the iteration ends and the PR values of all pages are obtained:

$$|R_{n+1} - R_n| < \epsilon \tag{3}$$

In summary, a random walk model is defined on a directed graph, that is, a first-order Markov chain, which describes the behavior of random walkers randomly visiting each node along the directed graph. Under certain conditions, the probability of visiting each node in the extreme case converges to a stable distribution. At this time, the stable probability value of each node is its PageRank value, which indicates the importance of the node.

2.2 Strategy 2

At the very beginning, we need to understand two properties. "Authority webpage": For a specific search, this webpage provides the best relevant information; "Hub webpage": This webpage provides many hyperlinks to other authoritative webpages.

The core idea is:

1. A good Hub web page points to a good Authority web page.
2. A good Authority webpage is directed by a good Hub-type webpage.

First, the query q is submitted to the retrieval system based on keyword query, and the first n web pages are always taken from the set of returned result pages, as the root set, denoted as S. By adding web pages referenced by S and web pages referencing to S, expand S into a larger set T. Take the Hub web page in T as the node set $V1$ and the authoritative web page as the node set $V2$. The hyperlink from the web page in $V1$ to the web page in $V2$ is the edge set E, forming a bipartite directed graph. For any node v in $V1$, use $h(v)$ to denote the Hub value of web page v, and $h(v)$ Convergence; for the vertex u in $V2$, use $a(u)$ to represent the authority value of the web page. At the beginning, $h(u) = a(u) = 1$, and then we start the iterative process:

$$\forall u, a(u) = \sum_{i=1}^{n} h(u) \tag{4}$$

$$\forall u, h(u) = \sum_{i=1}^{n} a(u) \tag{5}$$

At the end of each iteration, standardization is required:

$$\sum_{i=1}^{n} h(i)^2 = \sum_{i=1}^{n} a(i)^2 = 1 \tag{6}$$

We can set an upper limit of the number of iterations k to control, or set a threshold, when the change is less than the threshold, the iteration ends. Then just return to the top dozen or so pages of the user's authority value.

From the perspective of the two types of weight propagation, the first strategy is based on the random walk model, which directly transmits the weight of the webpage from the authority webpage to the authority webpage; and the second strategy is to spread the weight of the authority webpage through the hub webpage.

From the point of view of algorithm, although both are link analysis algorithms, there are still some differences between the two. The principle of strategy two is as mentioned above, its authority value is only relative to the weight of a certain retrieval subject, while strategy one is independent retrieval subject.

3 Experimental Results

3.1 Strategy 1

```
from pygraph.classes.digraph import digraph

class PRIterator:
    __doc__ = "calculate the PR of a graph"
    def __init__(self, dg):
        self.damping_factor = 0.85
        self.max_iterations = 100
        self.min_delta = 0.00001
        self.graph = dg

    def page_rank(self):
        for node in self.graph.nodes():
```

```
13              if len(self.graph.neighbors(node)) == 0:
14                  for node2 in self.graph.nodes():
15                      digraph.add_edge(self.graph, (node, node2))
16
17          nodes = self.graph.nodes()
18          graph_size = len(nodes)
19
20          if graph_size == 0:
21              return {}
22          page_rank = dict.fromkeys(nodes, 1.0 / graph_size)
23          # (1 - \alpha) / N
24          damping_value = (1.0 - self.damping_factor) / graph_size
25
26          flag = False
27          for i in range(self.max_iterations):
28              change = 0
29              for node in nodes:
30                  rank = 0
31                  for incident_page in self.graph.incidents(node):
32                      rank += self.damping_factor * (page_rank[
                            incident_page] / len(self.graph.neighbors(
                            incident_page)))
33                  rank += damping_value
34                  change += abs(page_rank[node] - rank)
35                  page_rank[node] = rank
36
37              print("This is NO.%s iteration" % (i + 1))
38              print(page_rank)
39
40              if change < self.min_delta:
41                  flag = True
42                  break
43          if flag:
44              print("finished in %s iterations!" % node)
45          else:
46              print("finished out of 100 iterations!")
47          return page_rank
48
49
50  if __name__ == '__main__':
51      dg = digraph()
52
53      dg.add_nodes(["A", "B", "C", "D", "E"])
54
55      dg.add_edge(("A", "B"))
56      dg.add_edge(("A", "C"))
57      dg.add_edge(("A", "D"))
58      dg.add_edge(("B", "D"))
59      dg.add_edge(("C", "E"))
60      dg.add_edge(("D", "E"))
61      dg.add_edge(("B", "E"))
```

```
    dg.add_edge(("E", "A"))

    pr = PRIterator(dg)
    page_ranks = pr.page_rank()

    print("The final page rank is\n", page_ranks)
```

```
▲ ~/Desktop/ASS3 py3 solution1.py
===This is NO.1 iteration===
{'A': 0.2, 'B': 0.08666666666666667, 'C': 0.08666666666666667, 'D': 0.1235, 'E': 0.245475}
===This is NO.2 iteration===
{'A': 0.23865375, 'B': 0.0976185625, 'C': 0.0976185625, 'D': 0.1391064515625, 'E': 0.272704151015625}
===This is NO.3 iteration===
{'A': 0.26179852836328127, 'B': 0.10417624970292971, 'C': 0.10417624970292971, 'D': 0.14845115582667484, 'E': 0.289008200823909}
===This is NO.4 iteration===
{'A': 0.27565697070032263, 'B': 0.10810280836509142, 'C': 0.10810280836509142, 'D': 0.15404650192025526, 'E': 0.29877060729770855}
===This is NO.5 iteration===
{'A': 0.2839550162030523, 'B': 0.11045392125753148, 'C': 0.11045392125753148, 'D': 0.15739683779198235, 'E': 0.30461606172653766}
===This is NO.6 iteration===
{'A': 0.288923652467557, 'B': 0.11186170153247449, 'C': 0.11186170153247449, 'D': 0.15940292468377615, 'E': 0.3081161554351147}
===This is NO.7 iteration===
{'A': 0.2918987321198475, 'B': 0.11270464076729012, 'C': 0.11270464076729012, 'D': 0.16060411309338843, 'E': 0.3102119131076751}
===This is NO.8 iteration===
{'A': 0.29368012614152383, 'B': 0.11320936907343176, 'C': 0.11320936907343176, 'D': 0.16132335092964026, 'E': 0.31146679385881976}
===This is NO.9 iteration===
{'A': 0.29474677477999683, 'B': 0.11351158618766577, 'C': 0.11351158618766577, 'D': 0.1617540103174237, 'E': 0.312218181159084}
===This is NO.10 iteration===
{'A': 0.2953854539852214, 'B': 0.11369254529581274, 'C': 0.11369254529581274, 'D': 0.16201187704653314, 'E': 0.3126680907417144}
===This is NO.11 iteration===
{'A': 0.29576787713045727, 'B': 0.11380089852029623, 'C': 0.11380089852029623, 'D': 0.1621662803914221, 'E': 0.3129374839460865}
===This is NO.12 iteration===
{'A': 0.2959968613541735, 'B': 0.1138657773836825, 'C': 0.1138657773836825, 'D': 0.16225873277174754, 'E': 0.3130987890201806}
===This is NO.13 iteration===
{'A': 0.2961339706671535, 'B': 0.11390462502236015, 'C': 0.11390462502236015, 'D': 0.16231409065686322, 'E': 0.31319537396184294}
===This is NO.14 iteration===
{'A': 0.2962160678675665, 'B': 0.11392788589581052, 'C': 0.11392788589581052, 'D': 0.16234723740152998, 'E': 0.3132532063084589}
===This is NO.15 iteration===
{'A': 0.2962652253621901, 'B': 0.11394181385262053, 'C': 0.11394181385262053, 'D': 0.16236708473998424, 'E': 0.3132878346910778}
===This is NO.16 iteration===
{'A': 0.2962946594874161, 'B': 0.11395015352143459, 'C': 0.11395015352143459, 'D': 0.16237896876804428, 'E': 0.31330856919266675}
===This is NO.17 iteration===
{'A': 0.2963122838137668, 'B': 0.11395514708056725, 'C': 0.11395514708056725, 'D': 0.16238608458980833, 'E': 0.31332098442906037}
===This is NO.18 iteration===
{'A': 0.29632283676470134, 'B': 0.11395813708333205, 'C': 0.11395813708333205, 'D': 0.16239034534374816, 'E': 0.31332841832343433}
===This is NO.19 iteration===
{'A': 0.2963291555749192, 'B': 0.11395992741289376, 'C': 0.11395992741289376, 'D': 0.1623928965633736, 'E': 0.31333286953030715}
===This is NO.20 iteration===
{'A': 0.2963329391007611, 'B': 0.11396099941188231, 'C': 0.11396099941188231, 'D': 0.1623944241619323, 'E': 0.3133355347877924}
===This is NO.21 iteration===
{'A': 0.2963352045696236, 'B': 0.11396164129472669, 'C': 0.11396164129472669, 'D': 0.16239533884498553, 'E': 0.31333713066901425}
***finished in Node E iterations!***
The final page rank is
 {'A': 0.2963352045696236, 'B': 0.11396164129472669, 'C': 0.11396164129472669, 'D': 0.16239533884498553, 'E': 0.31333713066901425}
```

3.2 Strategy 2

```
from pygraph.classes.digraph import digraph
from math import sqrt

class HITSIterator:
    __doc__ = "calculate the Hup and Authority value of a graph"

    def __init__(self, bg):
        self.max_iterations = 100
        self.min_delta = 0.0001
```

```
10          self.graph = bg
11
12          self.hub = {}
13          self.authority = {}
14          for node in self.graph.nodes():
15              self.hub[node] = 1
16              self.authority[node] = 1
17
18      def hits(self):
19          if not self.graph:
20              return
21
22          flag = False
23          for i in range(self.max_iterations):
24              change = 0.0
25              norm = 0
26              tmp = {}
27              tmp = self.authority.copy()
28              for node in self.graph.nodes():
29                  self.authority[node] = 0
30                  for incident_page in self.graph.incidents(node):
31                      self.authority[node] += self.hub[incident_page]
32                  norm += pow(self.authority[node], 2)
33              norm = sqrt(norm)
34              for node in self.graph.nodes():
35                  self.authority[node] /= norm
36                  change += abs(tmp[node] - self.authority[node])
37
38              # calculate the hup value of each page
39              norm = 0
40              tmp = self.hub.copy()
41              for node in self.graph.nodes():
42                  self.hub[node] = 0
43                  for neighbor_page in self.graph.neighbors(node):
44                      self.hub[node] += self.authority[neighbor_page]
45                  norm += pow(self.hub[node], 2)
46              # normalization
47              norm = sqrt(norm)
48              for node in self.graph.nodes():
49                  self.hub[node] /= norm
50                  change += abs(tmp[node] - self.hub[node])
51
52              print("====This is NO.%s iteration====" % (i + 1))
53              print("@authority", self.authority)
54              print("@hub", self.hub)
55
56              if change < self.min_delta:
57                  flag = True
58                  break
59          if flag:
60              print("***finished in %s iterations!***" % (i + 1))
```

```
            else:
                print("***finished out of 100 iterations!***")

            print("The best authority page: ", max(self.authority.items(), key=lambda x: x[1]))
            print("The best hub page: ", max(self.hub.items(), key=lambda x: x[1]))


if __name__ == '__main__':
    bg = digraph()

    bg.add_nodes(["A", "B", "C", "D", "E"])

    bg.add_edge(("A", "C"))
    bg.add_edge(("A", "D"))
    bg.add_edge(("B", "D"))
    bg.add_edge(("C", "E"))
    bg.add_edge(("D", "E"))
    bg.add_edge(("B", "E"))
    bg.add_edge(("E", "A"))

    hits = HITSIterator(bg)
    hits.hits()
```

```
△ ~/Desktop/ASS3 py3 solution2.py
====This is NO.1 iteration===
@authority {'A': 0.2581988897471611, 'B': 0.0, 'C': 0.2581988897471611, 'D': 0.5163977794943222, 'E': 0.7745966692414834}
@hub {'A': 0.412081691846067, 'B': 0.6868028197434451, 'C': 0.4120816918460671, 'D': 0.4120816918460671, 'E': 0.137360563948689}
====This is NO.2 iteration===
@authority {'A': 0.07161148740394328, 'B': 0.0, 'C': 0.21483446221182984, 'D': 0.5728918992315463, 'E': 0.7877263614433762}
@hub {'A': 0.4085297439895141, 'B': 0.705642285072797, 'C': 0.4085297439895141, 'D': 0.4085297439895141, 'E': 0.03713906763541037}
====This is NO.3 iteration===
@authority {'A': 0.019234326202820506, 'B': 0.0, 'C': 0.21157758823102554, 'D': 0.5770297860846152, 'E': 0.7886073743156409}
@hub {'A': 0.40826853033350996, 'B': 0.7070016013092492, 'C': 0.40826853033351007, 'D': 0.40826853033351007, 'E': 0.009957769032524634}
====This is NO.4 iteration===
@authority {'A': 0.005154707656790011, 'B': 0.0, 'C': 0.21134301392839044, 'D': 0.5773272575604813, 'E': 0.7886702714888718}
@hub {'A': 0.4082497437883339, 'B': 0.7070992294373103, 'C': 0.4082497437883339, 'D': 0.4082497437883339, 'E': 0.0026682989790087178}
====This is NO.5 iteration===
@authority {'A': 0.0013812167871263142, 'B': 0.0, 'C': 0.2113261684303261, 'D': 0.5773486170187994, 'E': 0.7886747854491255}
@hub {'A': 0.4082483948087273, 'B': 0.7071062389944508, 'C': 0.4082483948087274, 'D': 0.4082483948087274, 'E': 0.0007149709191046012}
====This is NO.6 iteration===
@authority {'A': 0.0003700962503647723, 'B': 0.0, 'C': 0.211324958958285, 'D': 0.5773501505690449, 'E': 0.7886751095273299}
@hub {'A': 0.40824829795549167, 'B': 0.7071067422589018, 'C': 0.40824829795549167, 'D': 0.40824829795549167, 'E': 0.00019157592583551931}
====This is NO.7 iteration===
@authority {'A': 9.916699771093284e-05, 'B': 0.0, 'C': 0.21132487212199788, 'D': 0.577350260673051, 'E': 0.7886751327950488}
@hub {'A': 0.4082482910017378, 'B': 0.7071067783916682, 'C': 0.40824829100173776, 'D': 0.40824829100173776, 'E': 5.133261549122819e-05}
====This is NO.8 iteration===
@authority {'A': 2.6571717073737266e-05, 'B': 0.0, 'C': 0.2113248658874325, 'D': 0.5773502685781633, 'E': 0.7886751344655957}
@hub {'A': 0.4082482905024808, 'B': 0.7071067809858843, 'C': 0.40824829050248074, 'D': 0.40824829050248074, 'E': 1.37545328830727e-05}
====This is NO.9 iteration===
@authority {'A': 7.119870133749227e-06, 'B': 0.0, 'C': 0.2113248654398108, 'D': 0.5773502691457247, 'E': 0.7886751345855355}
@hub {'A': 0.40824829046663563, 'B': 0.7071067811721405, 'C': 0.40824829046663563, 'D': 0.40824829046663563, 'E': 3.6855159786102473e-06}
***finished in 9 iterations!***
The best authority page:  ('E', 0.7886751345855355)
The best hub page:  ('B', 0.7071067811721405)
```