CSC4008 Assignment 3

Start: Nov 10 at 00:00

End: Nov 26 at 23:59

Correspondence TA/USTF: Mengzhen Zhang (120090384@link.cuhk.edu.cn)

Important Notes

1. The assignment is an individual project, to be finished on one's own effort.
2. The work must be submitted before the deadline. Late submissions within 2 days will apply 20% penalty. Late submissions after 2 days will not be accepted.
3. Plagiarism is strictly forbidden, regardless of the role in the process. Notably, ten consecutive lines of identical codes are treated as plagiarism. Depending on the seriousness of the plagiarism, 30%-100% marks will be deducted.
4. Please read the document carefully. No argument will be accepted on issues that have been specified clearly in the documents.

1 Implementing PageRank and HITS [90pts]

In this problem, you will learn how to implement the PageRank and HITS algorithms in Spark. The general computation should be done in Spark, and you may also include numpy operations whenever needed. You will be experimenting with a small randomly generated graph (assume graph has no dead-ends) provided at `graph-full.txt`.

There are 100 nodes ($n = 100$) in the small graph and 1000 nodes ($n = 1000$) in the full graph, and $m = 8192$ edges, 1000 of which form a directed cycle (through all the nodes) which ensures that the graph is connected. It is easy to see that the existence of such a cycle ensures that there are no dead ends in the graph. There may be multiple directed edges between a pair of nodes (e.g., two edges between A and E in Figure 1), and your solution should treat them as one edge. The first column in `graph-full.txt` refers to the source node, and the second column refers to the destination node.

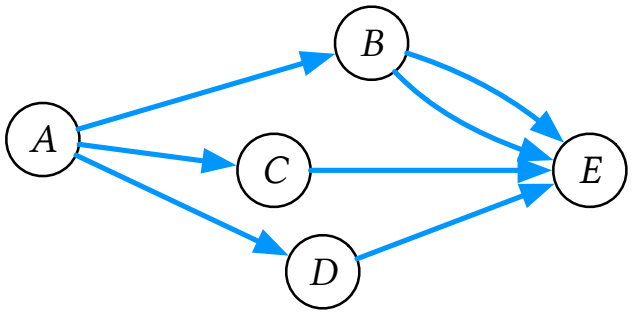

Figure 1: A toy graph.

(a) PageRank Implementation [45pts (5pts for code)]

Assume the directed graph $G = (V, E)$ has n nodes (numbered $1, 2, ..., n$) and m edges, all nodes have positive out-degree, and $M = [M_{ji}]_{n \times n}$ is an $n \times n$ matrix as defined in lecture such that for any $i, j \in [1, n]$:

$$M_{ji} = \begin{cases} \frac{1}{\deg(i)} & \text{if } (i \to j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Here, $\deg(i)$ is the number of outgoing edges of node i in G. If there are multiple edges in the same direction between two nodes, treat them as a single edge. By the definition of PageRank, assuming $1–\beta$ to be the teleport probability, and denoting the PageRank vector by the column vector $\mathbf{r}$, we have the following equation:

$$\mathbf{r} = \frac{1-\beta}{n}\mathbf{1} + \beta M\mathbf{r}. \tag{2}$$

Based on this equation, the iterative procedure to compute PageRank works as follows:

1. Initialize: $\mathbf{r}^{(0)} = \frac{1}{n}\mathbf{1}$
2. For i from 1 to k, iterate: $\mathbf{r}^{(i)} = \frac{1-\beta}{n}\mathbf{1} + \beta M\mathbf{r}^{i-1}$

Run the aforementioned iterative process in Spark for 40 iterations (assuming β = 0.8) and obtain the PageRank vector $\mathbf{r}$. In particular, you don't have to implement the blocking algorithm from lecture. The matrix M can be large and should be processed as an RDD in your solution.

Compute the PageRank scores and report the node id for the following using `graph-full.txt`:

- List the top 5 node ids with the highest PageRank scores. [20 = 5 × 4 points for each]
- List the bottom 5 node ids with the lowest PageRank scores. [20 = 5 × 4 points for each]

For a sanity check, we have provided a smaller dataset (`graph-small.txt`). In that dataset, the top node has id 53 with value 0.036. Note that the `graph-small.txt` dataset is only provided for sanity check purpose. Your write-up **should include** results obtained by using `graph-full.txt` as input (for both top and bottom ids). Some key spark functions that may be relevant to your implementation are the following: `map()`, `distinct()`, `groupByKey()`, `cache()`, `count()`, `flatMap()`, `reduceByKey()`.

(b) HITS Implementation [45pts (5pts for code)]

Assume the directed graph $G = (V, E)$ has n nodes (numbered 1, 2, ..., n) and m edges, all nodes have non-negative out-degree, and $L = [L_{ij}]_{n \times n}$ is a an $n \times n$ matrix referred to as the link matrix such that for any $i, j \in [1, n]$:

$$L_{ij} = \begin{cases} 1 & \text{if } (i \to j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Given the link matrix L and some scaling factors λ, μ, the hubbiness vector h and the authority vector a can be expressed using the equations:

$$h = \lambda La, a = \mu L^T h, \tag{4}$$

where **1** is the $n \times 1$ vector with all entried equal to 1.

Based on this equation, the iterative method to compute h and a is as follows:

1. Initialize h with a column vector (of size $n \times 1$) of all 1's.
2. Compute $a = L^T h$, and then scale a so that the largest value in the vector a has value 1. To scale vector a, normalize a by dividing each entry by the maximum value in a.
3. Compute $h = La$, and then scale h so that the largest value in the vector h has value 1. To scale vector h, normalize h by dividing each entry by the maximum value in h.
4. Go to step 2.

Repeat the iterative process for 40 iterations, and then obtain the hubbiness and authority scores of all the nodes (pages). The link matrix L can be large and should be processed as an RDD. Compute the following using `graph-full.txt`:

- List the 5 node ids with the highest hubbiness score. [10 = 5 × 2 points for each]
- List the 5 node ids with the lowest hubbiness score. [10 = 5 × 2 points for each]
- List the 5 node ids with the highest authority score. [10 = 5 × 2 points for each]
- List the 5 node ids with the lowest authority score. [10 = 5 × 2 points for each]

For a sanity check, you should confirm that `graph-small.txt` has highest hubbiness node id 59 with value 1 and highest authority node id 66 with value 1.

Some key spark functions that may be relevant to your implementation are the following: `map()`, `mapValues()`, `distinct()`, `groupByKey()`, `cache()`.

2 Bloom Filter [10pts]

Consider a Bloom filter of size $m = 7$ and we use 2 hash functions that both take a string (lowercase) as input:

1. $h_1(str) = \sum_{c \in str}(c - \text{'a'}) \bmod 7$,
2. $h_2(str) = (str.length * 3) \bmod 7$.

Here, $c - \text{'a'}$ is used to compute the position of the letter c in the 26 alphabetical letters, e.g., $h_1(\texttt{"bd"}) = (1 + 3) \bmod 7 = 4$.

Given a set of string `S = {"hi", "big", "data", "spark"}`, show the empty bloom filter and the intermediate bloom filter when the bits are updated after inserting each string [10 = 5 × 2 points for each].

What to submit

You need to submit the following two files to BlackBoard. Please format your files as "student_id.pdf" and "student_id.py" (or "student_id.ipynb"). For example, if your student id is 123456789, then you should submit "123456789.pdf" and "123456789.py" (or "123456789.ipynb"). You can submit several files in one submission. Don't submit them in different submission.

1. A writeup contains
    - List 5 node ids with the highest and least PageRank scores using `graph-full.txt` for 1(a). [40pts]
    - List 5 node ids with the highest and least hubbiness and authority scores, respectively, using `graph-full.txt` for 1(b). [40pts]
    - The update and final result for 2. [10pts]
2. Your code for 1. *Please note if you do not use RDD for implementing PageRank or HITS, you cannot get full marks for code.* [10pts]