Implementing PageRank and HITS

이 문제에서는 Spark에서 PageRank 및 HITS 알고리즘을 구현하는 방법을 배웁니다. 일반적인 계산은 스파크에서 수행해야 하며 필요할 때마다 numpy 연산을 포함할 수도 있습니다.

graph-full.txt에 제공된 무작위로 생성된 작은 그래프(그래프에 막다른 곳이 없다고 가정)로 실험합니다.
작은 그래프에는 100개의 노드(n = 100)와 1000개의 노드(n = 1000)가 있고, m = 8192개의 edge가 있으며, 그 중 1000개는 (모든 노드를 통해) directed cycle을 형성하여 그래프가 연결되도록 합니다. 이러한 cycle이 존재함에 따라 그래프에 막다른 곳이 없음을 보장합니다.
한 쌍의 노드 사이에 여러 방향 edge가 있을 수 있으나 해답에서는 이들을 동일한 edge로 취급해야 합니다. **graph-full.txt**에서 첫번째 열은 source node를 나타내고, 두 번째 열은 destination node를 나타냅니다.

Hint: PageRank vector ***r***을 메모리에 저장하거나 RDD로 저장하도록 선택할 수 있습니다. Link의 행렬 M만 너무 커서 메모리에 저장할 수 없으며, 행렬 M을 RDD에 저장할 수 있습니다 (예: data = sc.textFile("graph-full.txt")). 실제 클러스터에서 RDD는 클러스터의 노드에 걸쳐 분할됩니다. 그러나 M = data.collect는 전체 RDD를 드라이버 노드의 single machine에 가져와 로컬 array로 저장하는 기능을 수행할 수 없습니다.

(a) PangRank Implementation

Directed Graph $G = (V, E)$는 n개의 node (numbered 1, 2, 3, …, n)와 m개의 edge를 가지며 모든 노드는 positive out-degree를 갖는다. 그리고 $M = [M_{ij}]_{n\times n}$ 은 $i, j \in [1, n]$에 대해 다음과 같이 정의된 $n \times n$ matrix입니다.:

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

여기서 $deg(i)$는 G의 노드 i에서 나가는 (outgoing) edge의 개수이다. 만약 두 노드 사이에 같은 방향으로 여러 개의 edge가 있다면 이들을 하나의 edge로 취급합니다.

PageRank의 정의에 따라, 1 − β를 teleport 확률로 가정하고 열 벡터 r로 PageRank 벡터를 나타내는 정의에 따르면, 우리는 다음과 같은 방정식을 갖는다.:

$$\mathbf{r} = \frac{1-\beta}{n}\mathbf{1} + \beta M\mathbf{r},$$

이 방정식을 기반으로 페이지랭크를 계산하는 반복 절차는 다음과 같이 작동합니다.

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

앞서 언급한 반복 과정을 스파크에서 40회 동안 실행하고 (108 β = 0.8) PageRank 벡터 r을 구하세요. blocking 알고리즘을 구현할 필요는 없습니다. 행렬 M은 클 수 있으며 솔루션에서 RDD로 처리되어야 합니다.

PageRank score를 계산하고 graph-full.txt 를 사용해 다음의 node ID를 확인하세요.

1. PageRank가 가장 높은 상위 5개 노드의 ID를 나열하세요.
2. PageRank가 가장 낮은 하위 5개 노드의 ID를 나열하세요.

Sanity check을 위해서 더 작은 dataset (graph-small.txt)이 제공됩니다. 해당 dataset에서 top node의 ID는 53이고 값은 0.036입니다.
최종 결과물에는 graph-full.txt를 통해 얻은 결과를 포함해야 합니다. 문제와 관련하여 주요 spark function에는 map(), distincent(), groupByKey(), cache(), count(), flatMap() 및 reduceByKey()가 있습니다.

(b) HITS Implementation

Directed graph $G = (V, E)$가 n개의 node와 m개의 edge를 갖고 모든 노드가 non-negative out-degree이며 $L = [L_{ij}]_{n\times n}$ 은 다음의 식을 만족하는 link matrix로 지칭되는 $n \times n$ matrix라고 가정합니다:

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

link matrix L과 scaling factors λ, μ가 주어질 때, hubbiness 벡터 h와 authority 벡터 a는 다음과 같이 표현됩니다.:

$$h = \lambda La, a = \mu L^T h$$

여기서 $\mathbf{1}$ 은 모든 항목이 1인 $n \times 1$ 벡터입니다.

위 식에 기초하여 h와 a를 계산하는 반복 방법은 다음과 같습니다.:

1. 모든 1의 열 벡터(크기 n × 1)로 h를 초기화한다.
2. $a = L^T h$를 계산하고 가장 큰 값이 1이 되도록 스케일링한다.
3. $h = La$를 계산하고 벡터 h에서 가장 큰 값이 1이 되도록 스케일링한다.
4. 2단계로 돌아간다.

위의 과정을 40번 반복하고 λ = 1, μ = 1이라 가정한 후 모든 노드(page)의 hubbiness, authority scores를 계산합니다. Link matrix L은 클 수 있으니 RDD로 처리되어야 합니다. graph-full.txt를 사용해 다음을 구하세요.:

1. hubbiness score를 계산한 뒤 상위 / 하위 5개 node의 ID를 구하세요.
2. authority score를 계산한 뒤, 상위/하위 5개 node의 ID를 구하세요.

Sanity check을 위한 작은 데이터, graph-small.txt에는 값이 1인 가장 높은 hubbiness node의 ID는 59이고, 값이 1인 가장 높은 authority node의 ID는 66입니다.

이 문제에서 사용될 주요 spark function은 map(), mapValues(), distinct(), groupByKey(), cache() 등 입니다.