

빅데이터 최신기술

Spark로 삼각형 구하기

목차

1. 과제 수행 방법

- 1.1. Data 수정
- 1.2. Load Data

2. 결과 화면

- 1.1 Problem 1
- 1.2 Problem 2
- 1.3 Problem 3
- 1.4 Problem 4

소프트웨어학부

20163162

차운성

"take" action을 사용하여 RDD "data" 중 5개의 항목을 확인하여 데이터가 정상적으로 load되었음을 확인한다.

2. 결과 화면

2.1 Problem1: 전체 삼각형의 수를 구해보세요 (정답이 잘 나오는지 확인, 정답: 667,129)

```
1 from pyspark import SparkConf, SparkContext
2
3 conf = SparkConf().setAppName("p1")
4 sc = SparkContext(conf=conf)
5
6 # Load Dataset
7 #data = sc.textFile("graph_sample.txt")
8 data = sc.textFile("com-amazon.ungraph.txt")
```

[그림 5] - Problem1.1

먼저 spark-submit을 위해 SparkContext를 정의한 후 data를 load한다.

```
10 # Create Edges
11 # [((0, 1), -1), ((0, 2), -1), ((0, 3), -1), ((1, 2), -1), ((1, 3), -1), ((2, 3), -1), ((3, 4), -1)]
12 edges = data.map(lambda line: (tuple(map(int, line.split("\t"))), -1)) # [((0, 1), -1), ((0, 2), -1), ...]
```

[그림 6] - Problem1.2

주어진 Dataset에서 Edges를 산출한다. 산출된 Edge들은 ((v1, v2) , -1)의 format을 가지며, 이때 -1은 해당 데이터가 Edge임을 의미한다.

```
14 # Create Wedges
15 # [((1, 2), 0), ((1, 3), 0), ((2, 3), 0), ((2, 3), 1)]
16 wedges = data.map(lambda line: tuple(map(int, line.split("\t")))).groupByKey()
17 .map(lambda x: [(v1,v2), x[0]] for v1 in x[1] for v2 in x[1] if v1 < v2]).flatMap(lambda x : x)
```

[그림 7] - Problem1.3

주어진 Dataset에서 Wedges를 산출한다. 산출된 Wedge들은 ((v1, v2), key)의 format을 가지며, 이때 key는 Wedge의 중심 node(key)를 의미한다.

```
18 # Get Triangles
19 # [((1, 2), (0, -1)), ((2, 3), (0, -1)), ((2, 3), (1, -1)), ((1, 3), (0, -1))]
20 triangle = wedges.join(edges)
```

[그림 8] - Problem1.4

산출된 Edge와 Wedge를 join하여 전체 삼각형을 얻는다. 이때 전체 삼각형의 개수는 triangle.count()를 통해 확인할 수 있다.

```
22 # Print Triangle's Count && Save Triangles As Text File
23 print(triangle.count())
24 triangle.saveAsTextFile("p1")
```

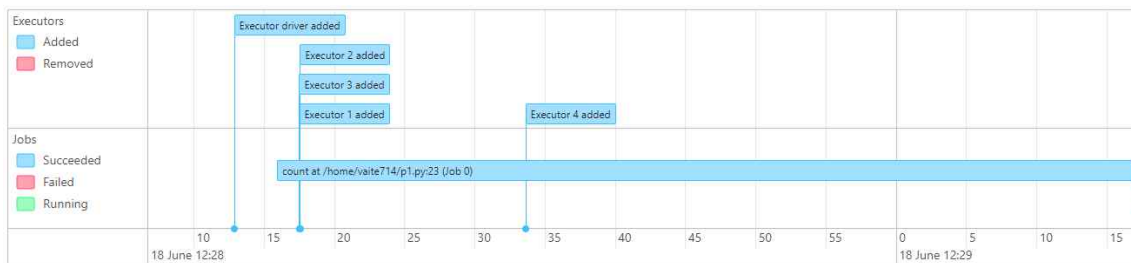
[그림 9] - Problem1.5

전체 삼각형의 개수를 print하고 각 삼각형들을 p1으로 저장하였다.

```
21/06/18 12:15:40 INFO org.apache.hadoop:
667129
21/06/18 12:18:06 INFO org.sparkproject:
{0,0:0}
```

[그림 10] - Problem1.6

프로그램 p1.py를 spark-submit으로 실행한 결과창 중 일부이다. triangle을 count한 결과 667,129가 정상적으로 출력됨을 확인할 수 있다.



[그림 11] - Problem1.7

YarnResourceManager의 Application Master이다. Executor가 추가되고, 정상적으로 모든 작업이 완료되었음을 확인할 수 있다.

```
vaite714@kmu-cluster-5e6d-m:~$ hdfs dfs -ls p1
Found 5 items
-rw-r--r--  2 vaite714 hadoop          0 2021-06-18 12:29 p1/_SUCCESS
-rw-r--r--  2 yarn      hadoop    5358892 2021-06-18 12:29 p1/part-00000
-rw-r--r--  2 yarn      hadoop    5414581 2021-06-18 12:29 p1/part-00001
-rw-r--r--  2 yarn      hadoop    5421843 2021-06-18 12:29 p1/part-00002
-rw-r--r--  2 yarn      hadoop    5410456 2021-06-18 12:29 p1/part-00003
```

[그림 12] - Problem1.8

삼각형을 저장하고 있는 데이터 또한 분산 파일 시스템에 정상적으로 저장되어있다.

```
vaite714@kmu-cluster-5e6d-m:~$ cat p1
((548099, 548406), (82633, -1))
((548099, 548406), (547095, -1))
((548099, 548406), (548089, -1))
((548190, 548445), (545730, -1))
((548190, 548445), (548188, -1))
((548191, 548442), (109594, -1))
((548191, 548442), (208249, -1))
((548268, 548271), (527644, -1))
((548268, 548271), (537088, -1))
((548268, 548271), (547818, -1))
((548268, 548271), (98005, -1))
((548268, 548271), (99171, -1))
((548268, 548271), (169235, -1))
((548268, 548271), (184743, -1))
((548268, 548271), (436783, -1))
((548304, 548383), (30254, -1))
((548304, 548383), (238599, -1))
((548304, 548383), (391985, -1))
((548304, 548383), (544679, -1))
((548304, 548383), (546335, -1))
((548304, 548383), (547779, -1))
((548411, 548458), (174840, -1))
((548411, 548458), (236762, -1))
((548411, 548458), (545624, -1))
((548411, 548458), (54635, -1))
((548411, 548458), (55261, -1))
((548411, 548458), (139287, -1))
((548411, 548458), (164067, -1))
((548411, 548458), (354455, -1))
```

[그림 13] - Problem1.9

분할된 파일들을 cat한 결과이다. 각 삼각형들이 정상적으로 저장되어 있음을 확인할 수 있다.

2.2 Problem2: 각 정점마다 그 정점이 속한 삼각형의 개수를 구하세요.

```
22 # Get Node's Triangle Count
23 node_triangle = triangle.flatMap(lambda x: x).flatMap(lambda x:x).filter(lambda x: x!= -1)
24 .map(lambda x: (x, 1)).reduceByKey(lambda x,y: x+y).sortByKey()
```

[그림 14] - Problem2.1

triangle을 추출하는 과정까지는 앞선 Problem1과 동일하다. 이후 triangle 중 -1을 제거한 1차원 리스트 형태의 RDD로 transformation 과정을 거치고 WordCount와 동일한 방식으로 (Node, 1)의 형태로 RDD를 변형한다. 해당 RDD에 대해 key값을 기준으로 reduce를 진행하면 각 노드가 구성하고 있는 삼각형의 총 개수가 산출된다.

```
25 # Save Node's Triangle count As Text File
26 node_triangle.saveAsTextFile("p2")
```

[그림 15] - Problem2.2

Problem1과 같은 방식으로 도출한 RDD를 TextFile로 저장한다.

```
vaite714@kmu-cluster-5e6d-m:~$ hdfs dfs -cat p2/* | more
(1, 13)
(2, 21)
(4, 9)
(5, 6)
(6, 3)
(7, 1)
(8, 6)
(10, 9)
(11, 1)
(15, 14)
(16, 1)
(17, 1)
(20, 9)
(21, 28)
(22, 1)
(25, 1)
(26, 9)
(29, 6)
```

[그림 16] - Problem2.3

이 또한 결과 TextFile이 분산되어 저장되므로 hdfs dfs -cat p2/* 명령을 통해 결과를 확인한다. 1번 노드부터 삼각형의 개수 결과를 확인하기 위해 파이프라인과 more를 활용하였다.

결과로 1번 노드의 경우 13개의 삼각형이 구성되고, 2번 노드를 통해서는 21개의 삼각형이 구성됨을 확인할 수 있었다.

2.3 Problem3: 각 정점의 이웃의 수 (즉, degree)를 구하세요.

```
10 # Get Node's Degree
11 degree = data.map(lambda line: tuple(map(int, line.split("\t")))).flatMap(lambda x: x)
12 .map(lambda x: (x, 1)).reduceByKey(lambda x,y: x+y).sortByKey()
```

[그림 17] - Problem3.1

먼저 앞서와 동일하게 textFile을 이용해 dataset을 load하여 data에 기록한다. Node별 degree는 dataset에서 해당 Node가 출현하는 횟수와 동일하기 때문에 Edge 산출과 동일하게 file에서 읽은 한 줄을 tuple로 만들어준 후, flatMap을 적용하고, word count와 동일한 방식으로 (Node, 1)의 형태로 만들어 준다. 이후 reduceByKey를 사용해 각 key값들의 합을 구한 후 sorting한다.

```
13 # Save Node's Degree As Text File
14 degree.saveAsTextFile("p3")
```

[그림 18] - Problem3.2

각 Node들의 Degree를 tuple 형태로 담고 있는 RDD를 text file로 저장한다.

```
vaite714@kmu-cluster-5e6d-m:~$ hdfs dfs -cat p3/* | more
(1, 8)
(2, 9)
(4, 6)
(5, 4)
(6, 4)
(7, 3)
(8, 8)
(10, 5)
(11, 4)
(12, 1)
(15, 6)
(16, 3)
```

[그림 19] - Problem3.3

Problem2와 같은 방식으로 결과를 확인해보면 1번 노드의 Degree는 8, 2번 노드의 Degree는 9임을 알 수 있다.

2.4 Problem4: (문제 2)와 (문제 3)에서 구한 값을 이용하여 가장 clustering coefficient가 가장 높은 정점 10개를 구하세요.

$$\text{Clustering Coefficient} = \text{Count of Triangle} / \text{Degree} * (\text{Degree} - 1) / 2$$

Clustering Coefficient는 위와 같은 공식으로 얻어낸다. Problem2와 Problem3의 결과를 이용해 1번 노드의 C.C를 계산하면 $13 / ((8 * 7) / 2) = 0.4642$ 가 된다.

```
28 # Calculate Each Node's Clustering Coefficient
29 # x[0]: Node Number   x[1][0]: Count of Node's Triangle   x[1][1]: Degree of Node
30 # result format: (C.C, Node's Number)
31 clustering_coefficient = node_triangle.join(degree).map(lambda x: (x[1][0] / ((x[1][1] * (x[1][1] - 1)) / 2), x[0])).sortByKey(False)
```

[그림 20] - Problem4.1

Problem2와 Problem3에서 구한 각 노드들의 삼각형 수(node_triangle)와 각 노드들의 이웃 수(degree)를 join한다. 그러면 RDD의 형태가 (Node, (Triangle, Degree))가 되는데, 이때 map과 Clustering Coefficient 공식을 활용해 RDD의 형태를 (Node, C.C)의 형태로 Transformation한다.

```
>>> clustering_coefficient.take(5)
[(1, 0.4642857142857143), (2, 0.5833333333333334), (4, 0.6), (5, 1.0), (6, 0.5)]
```

[그림 21] - Problem4.2

[그림 21]의 Clustering_Coefficient는 노드의 번호를 키 값으로 반환된 결과이며, 1번 노드의 경우 앞서 직접 계산한 것처럼 C.C가 0.4642임을 확인할 수 있다.

```
33 # Get Top 10 Node With Biggest CC
34 rank10 = sc.parallelize(clustering_coefficient.take(10))
35
36 # Save Node's Clustering Coefficient As Text File
37 rank10.saveAsTextFile("p4")
```

[그림 22] - Problem4.3

Problem4의 의도에 맞게 10개의 가장 큰 Clustering Coefficient를 가지는 노드를 반환하기 위해 c.c를 10개 take한 것을 RDD로 변형하여 textfile로 저장한다.

```
vaite714@kmu-cluster-5e6d-m:~$ hdfs dfs -cat p4/* | more
(1.0, 418813)
(1.0, 419203)
(1.0, 419215)
(1.0, 419383)
(1.0, 419395)
(1.0, 419401)
(1.0, 419539)
(1.0, 419575)
(1.0, 419881)
(1.0, 419959)
```

[그림 23] - Problem4.4

C.C가 1.0인 Node가 많아 실행할 때 마다 결과값은 다르게 나온다. 해당 실행 결과에서는 [418813, 419203, 419215, 419383, 419395, 419401, 419539, 419575, 419881, 419959]번 노드들이 10개의 결과로 채택되었음을 확인할 수 있다.