Assignment 5

> Sahand Sabour - 山姆 - 2022380024

WordCount

To implement WordCount using Spark, we first read the input file and leverage the pyspark API to flatten the lines in the input and split them into words using RegEx.

```
# Split lines to words
words = lines.flatMap(lambda x: re.split(r'[^\w]+', x))
```

Then, we use this API to implement a trivial map-reduce logic. First, for each word in words, we create a new tuple (word, 1) to represent an occurrence of this word in the input dataset. Accordingly, we add up the 1s for the tuples that share the same key to obtain the wordcount for that key.

```
# Calculate the wordcount using mapreduce
word_counts = words.map(lambda x: (x, 1)).reduceByKey(lambda v1,v2: v1 + v2)
```

Lastly, as we are tasked to print the top-10 most frequent words in the dataset, we sort the wordcounts in the descending order and print the first 10 elements of this list.

```
# Sort in descending order and print top-10
for item in word_counts.sortBy(lambda x: -x[1]).collect()[:10]:
    print(item)
```

By running this program, we obtained the following results for */data/hw6_data/pg100.txt*:

```
('', 198753)
('the', 23288)
('I', 22225)
('and', 18653)
('to', 16373)
('of', 15725)
('a', 12796)
('you', 12186)
('my', 10839)
('in', 10016)
Total program time: 3.08 seconds
```

PageRank

To implement PageRank using Spark, we first read and process the input file. Each line in the provided input file is in the form **node edge**. Hence, we split each line by the tab space to extract each node and their edge. Then, we create tuples (node, edge) to track existing links in the dataset and remove duplicate tuples using the distinct() function as duplicate mentions of the same link should be neglected. Accordingly, we group the tuples by their keys and create a list of corresponding values to obtain all the edges for a given node. We cache this value in memory to improve the processing speed.

```
# Read the file and make a list of (node, list of edges) tuples
node_edge_list = (
  lines.map(lambda x: x.split("\t"))
  .map(lambda item: (item[0], item[1]))
  .distinct()
  .groupByKey()
  .map(lambda x: (x[0], (list(x[1]))))
  .cache()
)
```

Based on PageRank, We assign equal probabilities to all nodes (i.e., 1 / number of nodes).

```
# Count the number of nodes
num_nodes = node_edge_list.count()
# Set equal probability for each node --> list of (node, rank) tuples
node_rank_list = node_edge_list.map(lambda x: (x[0], 1.0 / num_nodes))
```

In this algorithm, the new rank for a node p_i with set of edges $S(p_i)$ would be

$$rank_{new}(p_i) = \frac{(1-d)}{n_{nodes}} + d \sum_{p_j \in S(p_i)} \frac{rank_{old}(p_j)}{L_{p_i}},$$

where L_{p_i} is the number of edges in $S(p_i)$ and d is the damping factor. We set d = 0.8 as required. Accordingly, the process within each iteration consists of the three steps. First, we join the list of edges for each node with its rank so that it could be used to calculate new ranks within the mapreduce function. Second, we calculate the contribution of each edge of the node, the rank of each would be the original rank divided by the number of edges for the node (map function). Then, we sum up the new ranks collected from the edges of the node (reduce function). Third, we calculate the new ranks using the mentioned formula. As we observed overflow errors during several runs, we create checkpoints of the ranks every 10 iterations.

```
for i in range(num_iter):
  # Create the input for the PageRank algorithm ---> list of (node, (edges,
rank))
  data = node_edge_list.join(node_rank_list)
  # Calculate the contribution of each edge in the node (the sum in the
formula)
  contributions = data.flatMap(
    lambda x: [(e, x[1][1] / len(x[1][0])) for e in x[1][0]]
  ).reduceByKey(lambda v1, v2: v1 + v2)
  # Update ranks
  node_rank_list = contributions.mapValues(lambda r: d * r + (1 - d) /
num_nodes)
  # Create checkpoints
  if i % 10 == 0:
    node_rank_list.checkpoint()
```

Lastly, as we are tasked to print the top-5 node IDs with the highest PageRank score, we sort the ranks in the descending order and print the first 5 elements of this list.

```
# Sort in descending order and print top-5
highest = node_rank_list.sortBy(lambda x: -x[1]).collect()
print("5 highest:", highest[:5])
```

By running this program, we obtained the following results for */data/hw6_data/full.txt*:

```
('263', 0.0020202911815182l8)
('537', 0.0019433415714531501)
('965', 0.0019254478071662634)
('243', 0.001852634016241731)
('285', 0.0018273721700645144)
Total program time: 345.21 seconds
```

Guide

For running the program, simply follow the commands as provided the assignment description:

```
spart-submit [python_file].py [path_to_data]
```