BDA Individual Assignment

Yash Srivastava

PG ID: 12010060

Q1. PageRank

Steps:

1. Read the text file and create an RDD.
2. Pass this RDD into a parsing function and the edges node wise
3. Count the number of linkages
4. Create ranks
5. Iterating until the distance get conversed, assuming that the comparing distance(delta) is 0.1

Outputs:

```
# Passing the text through the parser function and grouping it as per the nodes as key and values being the connections tha
txt_lnk = text.map(lambda x: passed(x)).distinct().groupByKey().mapValues(lambda x: list(x)).cache()
txt_lnk.collect()
```

```
[('1', ['1', '2']), ('2', ['1', '3']), ('3', ['2'])]
```

```

```

```
# Counting the number of nodes.
X = txt_lnk.count()
print(X)
```

```
3
```

```
# Generating ranks for each node
rank = txt_lnk.map(lambda node: (node[0],1.0/N))
print(rank.collect())
```

```
[('1', 0.3333333333333333), ('2', 0.3333333333333333), ('3', 0.3333333333333333)]
```

In [28]:

```
old_rank = rank.collect()
while True:
    print("Old Rank - {0}".format(old_rank))
    ranks = txt_lnk.join(rank).flatMap(lambda x : [(i, float(x[1][1])/len(x[1][0])) for i in x[1][0]])
    ranks = ranks.reduceByKey(lambda x,y: x+y)
    new_rank = ranks.sortByKey().collect()
    print("New_rank - {0}".format(new_rank))
    if check_conv(old_rank, new_rank):
        print("Page ranking is completed.")
        break
    old_rank = new_rank
```

```
Old Rank - [('1', 0.3333333333333333), ('2', 0.3333333333333333), ('3', 0.3333333333333333)]
New_rank - [('1', 0.3333333333333333), ('2', 0.5), ('3', 0.16666666666666666)]
Old Rank - [('1', 0.3333333333333333), ('2', 0.5), ('3', 0.16666666666666666)]
New_rank - [('1', 0.3333333333333333), ('2', 0.5), ('3', 0.16666666666666666)]
Page ranking is completed.
```

Q2. Inverted Index

Steps:

1. Read the text file and create RDD.
2. Parsing the content to get key value pairs (tuples within a tuple). This contains word and the corresponding document.
3. Generating frequency of each using mapper
4. Formatting the RDD to get desired result
5. Grouping the result by Key
6. Sorting the output by keys
7. Formatting the result to get the desired format

Outputs:

```
In [2]:
# Reading the given textfiles. As seen from the output we have got a list which has tuples.
textFile = sc.wholeTextFiles("C:/Users/ysrivastava/BDA/Assignment/InvertedIndex/*.txt")
textFile.collect()
```

```
Out[2]: [('file:/C:/Users/ysrivastava/BDA/Assignment/InvertedIndex/d1.txt',
  'China officially the People Republic of China is a sovereign state located in East Asia\nIt is the world most populous count
ry\nThe People Republic of China is a single party state governed by the Communist Party with its seat of government in the cap
ital city of Beijing exercises jurisdiction many provinces five autonomous regions four direct controlled municipalities Beijin
g Tianjin Shanghai and Chongqing and two mostly self governing special administrative regions Hong Kong and Macau\nChina also c
laims Taiwan which is controlled by the Republic of China a separate political entity as its extra province a claim which is co
ntroversial due to the complex political status of Taiwan'),
 ('file:/C:/Users/ysrivastava/BDA/Assignment/InvertedIndex/d2.txt',
  'India officially the Republic of India is a country in South Asia\nIt is the seventh largest country by area the second most
populous country\nThe most populous democracy in the world\nBounded by the Indian Ocean on the south the Arabian Sea on the sou
th west and the Bay of Bengal on the south east\nIt shares land borders with Pakistan to the west China Nepal and Bhutan to the
north east and Burma and Bangladesh to the east\nIn the Indian Ocean India is in the vicinity of Sri Lanka and the Maldives\nIn
addition India Andaman and Nicobar Islands share a maritime border with Thailand and Indonesia'),
 ('file:/C:/Users/ysrivastava/BDA/Assignment/InvertedIndex/d3.txt',
  'Relations between India and Pakistan have been strained by a number of historical and political issues and are defined by th
e violent partition of British India \nThe Kashmir dispute and the numerous military conflicts fought between the two nations\n
Consequently even though the two South Asian nations share historic cultural ethnic geographic and economic links their relatio
nship has been plagued by hostility and suspicion\nAfter the dissolution of the British Raj two new sovereign nations were form
ed the Union of India and the Dominion of Pakistan\nThe subsequent partition of the former British India displaced millions of
people with estimates of loss of life varying from several hundred thousand to a million\nIndia emerged as a secular nation wit
h a Hindu majority population and a large Muslim minority while Pakistan was established as an Islamic republic with an overwhe
lming Muslim majority population\nSoon after their independence India and Pakistan established diplomatic relations but the vio
lent partition and numerous territorial disputes would overshadow their relationship\nSince their independence the two countrie
s have fought three major wars one undeclared war and have been involved in numerous armed skirmishes and military standoffs\nT
he Kashmir dispute is the main centre point of all of these conflicts with the exception of the India Pakistan War and Banglade
sh Liberation War which resulted in the secession of East Pakistan now Bangladesh'),
 ('file:/C:/Users/ysrivastava/BDA/Assignment/InvertedIndex/d4.txt',
  'The India US China Pakistan strategic quadrilateral\nAlthough the disputed border between China and India is often highlight
ed as the major sticking point in Sino Indian relations in reality it has remained relatively peaceful since the end of a war t
he potential for overt military conflict in the region remains minimal\nOf much greater concern is the strategic quadrilateral
relationship in South Asia involving China India the United States and Pakistan\nIt has both regional and wider implications\nA
```

```
In [32]:
Kep_pair1 = textFile.flatMap(lambda x: map_out(x)).cache()
Kep_pair1.collect()
# From the output below, we are getting key value pairs(tuples within a tuple). This contains w
```

```
Out[32]: [(('china', 'd1'), 1),
 (('officially', 'd1'), 1),
 (('the', 'd1'), 1),
 (('people', 'd1'), 1),
 (('republic', 'd1'), 1),
 (('of', 'd1'), 1),
 (('china', 'd1'), 1),
 (('is', 'd1'), 1),
 (('a', 'd1'), 1),
 (('sovereign', 'd1'), 1),
 (('state', 'd1'), 1),
 (('located', 'd1'), 1),
 (('in', 'd1'), 1),
 (('east', 'd1'), 1),
 (('asia', 'd1'), 1),
 (('it', 'd1'), 1),
 (('is', 'd1'), 1),
 (('the', 'd1'), 1),
 (('world', 'd1'), 1),
 (('most', 'd1'), 1),
```

```
(('most', 'd1'), 1),
```

In [33]:

```
# Setting up our reducer - This will give us the frequecy of each word with respect to th
from operator import add
red = Kep_pair1.reduceByKey(add)
red.collect()
```

Out[33]:

```
[(('china', 'd1'), 5),
 (('officially', 'd1'), 1),
 (('the', 'd1'), 7),
 (('people', 'd1'), 2),
 (('republic', 'd1'), 3),
 (('of', 'd1'), 6),
 (('is', 'd1'), 5),
 (('a', 'd1'), 4),
 (('sovereign', 'd1'), 1),
 (('state', 'd1'), 2),
 (('located', 'd1'), 1),
 (('in', 'd1'), 2),
 (('east', 'd1'), 1),
 (('asia', 'd1'), 1),
 (('it', 'd1'), 1),
 (('world', 'd1'), 1),
 (('most', 'd1'), 1),
 (('populous', 'd1'), 1),
 (('country', 'd1'), 1),
```

In [34]:

```
new = red.map(lambda x: (x[0][0],(x[0][1], x[1])))
new.collect()
# Formating the output, which should keep the word as key and its corresponding information as value
```

Out[34]:

```
[('china', ('d1', 5)),
 ('officially', ('d1', 1)),
 ('the', ('d1', 7)),
 ('people', ('d1', 2)),
 ('republic', ('d1', 3)),
 ('of', ('d1', 6)),
 ('is', ('d1', 5)),
 ('a', ('d1', 4)),
 ('sovereign', ('d1', 1)),
 ('state', ('d1', 2)),
 ('located', ('d1', 1)),
 ('in', ('d1', 2)),
 ('east', ('d1', 1)),
 ('asia', ('d1', 1)),
 ('it', ('d1', 1)),
 ('world', ('d1', 1)),
 ('most', ('d1', 1)),
 ('populous', ('d1', 1)),
 ('country', ('d1', 1)),
 ('single', ('d1', 1)),
```

In [35]:

```
new2 = new.groupByKey().mapValues(lambda x: list(x))
new2.collect()
# just basic grouping using the words as key
```

Out[35]:

```
[('china', [('d1', 5), ('d2', 1), ('d4', 11), ('d5', 1)]),
 ('officially', [('d1', 1), ('d2', 1), ('d5', 1)]),
 ('the',
  [('d1', 7),
   ('d2', 17),
   ('d3', 18),
   ('d4', 26),
   ('d5', 9),
   ('d6', 16),
   ('d7', 15)]),
 ('people', [('d1', 2), ('d3', 1)]),
 ('republic', [('d1', 3), ('d2', 1), ('d3', 1), ('d5', 1), ('d6', 1)]),
 ('of',
  [('d1', 6),
```

```
In [36]:
1 new3 = new2.sortByKey()
2 new3.collect()
```

```
Out[36]: [('a',
  [('d1', 4),
   ('d2', 2),
   ('d3', 5),
   ('d4', 5),
   ('d5', 3),
   ('d6', 3),
   ('d7', 6)]),
 ('about', [('d4', 1)]),
 ('across', [('d5', 2)]),
 ('actors', [('d4', 1)]),
 ('addition', [('d2', 1)]),
 ('administrative', [('d1', 1)]),
 ('affect', [('d4', 1)]),
 ('affects', [('d4', 1)]),
 ('afghanistan', [('d4', 1)]),
 ('after', [('d3', 2), ('d7', 1)]),
 ('agenda', [('d7', 1)]),
 ('agree', [('d7', 1)]),
```

```
In [37]:
1 new4 = new3.map(lambda x: new_data(x)).cache()
2 new4.collect()
```

```
Out[37]: ['a#d1:4;d2:2;d3:5;d4:5;d5:3;d6:3;d7:6;',
 'about#d4:1;',
 'across#d5:2;',
 'actors#d4:1;',
 'addition#d2:1;',
 'administrative#d1:1;',
 'affect#d4:1;',
 'affects#d4:1;',
 'afghanistan#d4:1;',
 'after#d3:2;d7:1;',
 'agenda#d7:1;',
 'agree#d7:1;',
 'air#d7:1;',
 'aircraft#d7:1;',
 'airlines#d7:1;',
 'alaska#d5:1;d6:1;',
 'all#d3:1;',
```

Q3. Kmeans

Steps:

1. First step to read the info files.
2. Then parse the info.
3. Initialize the K clusters.
4. Now perform iteration until converse.
5. Get nearest points mapper.
6. Combined the mapper results using reducer.
7. Update the centroids.
8. Check the space between points and centroid.
9. After converse results as follows.

Outputs:

In [2]:
```
# Reading the text file
text = sc.textFile("C:/Users/ysrivastava/BDA/Assignment/KMeans/points.txt")
text.collect()
```

Out[2]:
```
['3.023\t5.138',
 '3.075\t4.989',
 '2.321\t5.350',
 '3.328\t4.944',
 '3.195\t5.186',
 '3.034\t4.656',
 '3.484\t4.877',
 '3.023\t4.951',
 '2.707\t4.871',
 '2.894\t5.031',
 '2.880\t5.207',
 '2.780\t5.196',
 '2.941\t4.951',
 '3.213\t5.123',
 '2.972\t4.941',
 '3.138\t5.460',
 '3.247\t5.185',
 '3.602\t5.405',
 '2.857\t4.686',
```

In [6]:
```
# Passing the data points to remove the tabs
def parse(points):
    var = points.split('\t')
    result = []
    for x in var:
        value = float(x)
        result.append(value)
    return np.array(result)
```

In [7]:
```
#Calling the parsing function
arr = text.map(lambda m: parse(m)).cache()
arr.collect()
```

Out[7]:
```
[array([3.023, 5.138]),
 array([3.075, 4.989]),
 array([2.321, 5.35 ]),
 array([3.328, 4.944]),
 array([3.195, 5.186]),
 array([3.034, 4.656]),
 array([3.484, 4.877]),
 array([3.023, 4.951]),
 array([2.707, 4.871]),
```

In [18]:
```
Dist = float(0.1)
kp = [np.array([float(2) , float(5)]), np.array([float(6), float(2)]), np.array([float(1), float(1)])]
dist2 = 1.0
type(kp)
print(kp)
```

```
[array([2., 5.]), array([6., 2.]), array([1., 1.])]
```

In [21]:
```
nearest = arr.map(lambda p: process_map(p, kp))
print(nearest.take(5))
```

```
[(0, (array([3.023, 5.138]), 1)), (0, (array([3.075, 4.989]), 1)), (0, (array([2.321, 5.35 ]), 1)), (0, (array([3.328, 4.944]),
1)), (0, (array([3.195, 5.186]), 1))]
```

In [23]:
```
Status = nearest.reduceByKey(lambda pointer1 , pointer2: pairs(pointer1 , pointer2))
print(Status.take(4))
```

```
[(0, (array([1496.991, 2502.065]), 500)), (2, (array([1016.494,  501.689]), 500)), (1, (array([2496.329,  995.104]), 500))]
```

In [24]:
```
new_points = Status.map(lambda map_object: mapper(map_object)).collect()
print(new_points)
```

```
[(0, array([2.993982, 5.00413 ])), (2, array([2.032988, 1.003378])), (1, array([4.992658, 1.990208]))]
```

In [25]:
```
updated_distance = all_points_dist(new_points)
print(updated_distance)
```

```
51.92141868048006
```

In [27]:
```
def change_points(new_points, kp):
    for (cluster_number, point) in new_points:
        kp[cluster_number] = point
    return kp

change_points(new_points, kp)
print(kp)
```

```
[array([2.993982, 5.00413 ]), array([4.992658, 1.990208]), array([2.032988, 1.003378])]
```

In [30]:

```
while dist2 > Dist:
    print(dist2)
    near = arr.map(lambda p: process_map(p, kp))
    point_Stats = nearest.reduceByKey(lambda pointer1 , pointer2: pairs(pointer1 , pointer2))
    new_Points = point_Stats.map(lambda map_object: mapper(map_object)).collect()
    print(new_Points)
    dist2 = sum(np.sum((kp[iK] - p) ** 2) for (iK, p) in new_Points)

    for (iK, p) in new_Points:
        kp[iK] = p

results(kp)
```

```
1.0
[(0, array([2.993982, 5.00413 ])), (2, array([2.032988, 1.003378])), (1, array([4.992658, 1.990208]))]
2.99398#5.00413
4.99266#1.99021
2.03299#1.00338
```