

Authors: Xihao Jiang

Algorithm Design

```
1: class MAPPER
2:   method MAP1(string s)
3:     word list = s.split()
4:     for all i = 2 → word list.length do // word list[1] is the document id of each line
5:       EMIT(pair (word list[1], word list[i]), integer 1)
6:   method MAP2(pair(docId, token), integer tf)
7:     EMIT(string token, integer 1)
8:   method MAP3(string token, integer sum)
9:     EMIT(string token, float log10( T sum ))
10:  method MAP4(string token, pair (pair (string docId, float tf), float idf))
11:    EMIT(string docId, pair (string token, integer tf * idf))
12:  method MAP5(string docId, pair (string token, integer tf * idf))
13:    a ← BUILD ARRAY (token)
14:    EMIT(string docId, array a)
15:  method MAP6(string docId, words [w1, w2, ...])
16:    count ← words.length
17:    EMIT(string docId, integer count)
18:  method MAP7(string docId, pair (pair (string token, integer tc), integer count))
19:    tf ← tc count
20:    EMIT(string token, pair (string docId, float tf))
21: class REDUCER
22:   method REDUCE1(pairs (docId, token), integers [r1, r2, ..])
23:     sum ← 0
```

```

24:         initialize vocabulary count T
25:         for all integer r ∈ integers [r1, r2, ...] do
26:             sum ← sum + r
27:             tf ← sum/T
28:             EMIT(pair(docId, token), integer tf)
29:     method REDUCE2(token,integers [r1, r2, ...])
30:         sum ← 0
31:         for all integer r ∈ integers [r1, r2, ...] do
32:             sum ← sum + r
33:             EMIT(string token, integer sum)
34: class JOINER
35:     method JOIN1(pair(string token, pair (string docId, float tf)), pair(string token, float idf))
36:         EMIT(string token, pair (pair (string docId, float tf), float idf))
37:     method JOIN2(pair(string docId, pair (string token, integer tc)), pair(string docId, integer
count))
38:         EMIT(string docId, pair (pair (string token, integer tc), integer count))

```

Results

```

[('10022814', 113), ('10023672', 197), ('10029092', 151), ('10029453', 188), ('10037190', 121), ('10071987', 164), ('10075741', 194), ('10077162', 153), ('10091218', 138), ('10096554', 249), ('10208432', 151), ('10226546', 152), ('10233167', 164), ('10233260', 193), ('10235542', 201), ('10318765', 175), ('10323082', 85), ('10330000', 141), ('10346884', 166), ('10360641', 192), ('10362109', 176), ('10381396', 147), ('10383771', 212), ('10402241', 124), ('10404636', 106), ('10406262', 199), ('10411339', 113), ('10427128', 155), ('10445062', 134), ('10454197', 130), ('10459848', 130), ('10462375', 173), ('10470109', 240), ('10471533', 105), ('10473862', 169), ('10489163', 129), ('10505544', 265), ('10506934', 99), ('10523301', 107), ('10525095', 177), ('10541432', 166), ('10542134', 97), ('10554021', 84), ('10556937', 241), ('10559135', 180), ('10576656', 157), ('10579343', 213), ('10583160', 97), ('10597250', 190), ('10598712', 258), ('10599309', 84), ('10600765', 163), ('10601618', 139), ('10618391', 167), ('10622253', 104), ('10632369', 174), ('10638988', 91), ('10656450', 223), ('10666371', 167), ('10680538', 109), ('10690627', 146), ('10690712', 221), ('10693240', 189), ('10698499', 137), ('10702262', 222), ('10703508', 45), ('10706093', 137), ('10706134', 254), ('10707088', 89), ('10711495', 99), ('10725459', 122), ('10738188', 187), ('10738256', 180), ('10739654', 147), ('10749673', 188), ('10754268', 216), ('10764044', 150), ('10770635', 158), ('10773880', 161), ('10778984', 178), ('10797285', 153), ('10809207', 257), ('10815804', 148), ('10815932', 251), ('10816576', 92), ('10820155', 123), ('10828317', 174), ('10845563', 202), ('10850332', 163), ('10869779', 156), ('10873065', 232), ('10873086', 228), ('10877829', 198), ('10880237', 141), ('10882298', 214), ('10888627', 144), ('10900013', 167), ('10903426', 151), ('10942601', 126), ('10962556', 185)]

```

```
[0.00764713 0.00199598 0.00169936 ... 0. 0. 0.]
[0.00198793 0.00291865 0.00795174 ... 0. 0. 0.]
[0.00250858 0.00147322 0.00501717 ... 0. 0. 0.]
...
[0.00381741 0.00089674 0.00305393 ... 0. 0. 0.]
[0.00108619 0.00446524 0.00434476 ... 0. 0. 0.]
[0.00060552 0. 0.0060552 ... 0. 0. 0.]
```

```
[('expression', 'cell'), ('expression', 'nuclear'), ('expression', 'much'), ('expression', 'antisense-cdna'), ('expression', 'tpa'), ('expression', 'resection'), ('expression', '18'), ('expression', 'gradual'), ('expression', 'patient'), ('expression', 'concept'), ('expression', 'dis_adenocarcinoma_dis'), ('expression', 'peptide'), ('expression', 'catalyz'), ('expression', 'electron-microscopic'), ('expression', 'egf-pdgf'), ('expression', 'stromelysins'), ('expression', '8%'), ('expression', 'thereby'), ('expression', 'wide'), ('expression', 'bladder'), ('expression', 'correspond'), ('expression', 'gamma'), ('expression', 'subclone'), ('expression', 'doxorubicin'), ('expression', 'bt474'), ('expression', 'interferon-gamma'), ('expression', 'phosphopeptide'), ('expression', 'defective'), ('expression', 'dis_pkc_dis'), ('expression', 'apoptotical'), ('expression', 'impairment'), ('expression', 'asthmatic'), ('expression', 'n=10'), ('expression', 'dis_carcinogenesis_tumor_dis'), ('expression', '8-fold'), ('expression', 'subcutaneous'), ('expression', '11lin-dtpa-hegf'), ('expression', 'atp'), ('expression', 'x'), ('expression', 'dis_astrocytoma_dis'), ('expression', 'tgf-alpha)-induced'), ('expression', 'jnk2'), ('expression', 'stability'), ('expression', 'keratin'), ('expression', 'require'), ('expression', 'il-1beta'), ('expression', 'growth-stimulating'), ('expression', 'dimerization'), ('expression', 'monolayers'), ('expression', 'megabase-sized'), ('expression', 'dis_submucosal_layer_dis'), ('expression', 'start'), ('expression', 'predicted'), ('expression', 'alpha-cyano-beta-hydroxy-beta-methyl-n-[4-(trifluoromethoxy)phenoxy]phenyl'), ('expression', 'energy'), ('expression', '30'), ('expression', 'chelation'), ('expression', 'deduce'), ('expression', 'insulin-like'), ('expression', 'tk'), ('expression', 'heterologous'), ('expression', 'protein-coupled'), ('expression', 'reversible'), ('expression', 'jak3'), ('expression', 'orthotopic'), ('expression', 'c-dependent'), ('expression', 'connect'), ('expression', '16'), ('expression', 'meningitidis'), ('expression', 'gene_egfr_gene'), ('expression', 'vitro'), ('expression', 'staining'), ('expression', 'loss'), ('expression', 'mcf-tr5-egfr-cd533'), ('expression', 'fine-granulated'), ('expression', 'altogether'), ('expression', 'gene_egfrviii_gene'), ('expression', 'enhance'), ('expression', 'il-6'), ('expression', 'whey'), ('expression', '1478'), ('expression', 'imp'), ('expression', 'thyroid'), ('expression', 'diploid'), ('expression', 'mt1-mmp'), ('expression', 'alpha'), ('expression', 'four'), ('expression', 'comparable'), ('expression', 'see'), ('expression', 'ductal'), ('expression', 'plasmid'), ('expression', 'cis-diammedichloroplatinum'), ('expression', 'chemoresistance'), ('expression', 'agonist'), ('expression', 'add'), ('expression', 'grow'), ('expression', 'extract'), ('expression', 'resident'), ('expression', '3-28'), ('expression', 'physiology'), ('expression', 'endometrium'), ('expression', 'gbms'), ('expression', 'remodel'), ('expression', '40%'), ('expression', '6-fold'), ('expression', 'compound'), ('expression', 'implant'), ('expression', '11lin-dtpa-mab'), ('expression', 'nanomolar'), ('expression', 'upon'), ('expression', 'dis_restenosis_dis'), ('expression', 'unrelated'), ('expression', 'deltaegfr-containing'), ('expression', 'non-clear'), ('expression', 'tyr845'), ('expression', 'k18'), ('expression', 'brain'), ('expression', 'gastric'), ('expression', 'exert'), ('expression', 'outline'), ('expression', 'keratinocyte'), ('expression', 'dis_pkd_dis'), ('expression', 'fractionate')]
```

Potential Improvements

1. Try to reduce operations like `groupByKey()`, `reduceByKey()`, `join()`.
 - a. The function `groupByKey` must hold all the key-value pair in memory and if a key has too many values, it can cause an out of memory error.
2. Reduce shuffling
 - a. Spark uses shuffling to redistribute data.
 - b. Shuffling is an expensive operation.
3. Caching
 - a. Spark will store the dataset in memory which allows for faster access and retrieval.
4. Dynamic allocation
 - a. Scaling up or down based number of executors based on workload.
5. Data Skewing
 - a. There might be uneven distribution of data which reduces utilization.

6. Optimize the amount of Spark partitions
 - a. Too much or too little spark partitions could mean some executors are idle or scheduling overhead.
7. Use mapPartitions() over map()
 - a. Using mapPartitions provides initialization for many RDD elements rather once per RDD element.
8. Check for memory leaks
 - a. Unchecked memory leaks can cause a host of memory issues and slow data processing.
9. Check for bottlenecks
 - a. Bottlenecks can occur in any stage of our algorithm which can often slow data processing.
10. Improve queries
 - a. Instead of returning every row or column we should only return the ones we are looking for.