Saloni Shah
NUID: 001617242

CS6240 : Assignment 4

Design Discussion

To **execute the source code**, Spark uses the below commands:
Standalone Execution:
jar clean-local-output ${spark.root}/bin/spark-submit --class ${job.name} --master local[*] ${jar.path} ${local.input} ${local.output}
where job.name is PageRank.scala
spark.root is the location of root directory of Spark
jar.path is location of jar files
local.input and local.output are locations for input and output
AWS Execution:
*--applications Name=Spark \
--steps '[{"Name":"Spark Program", "Args":["--class", "${job.name}", "--master", "yarn", "--deploy-mode", "cluster", "s3://${aws.bucket.name}/${jar.name}", "s3://${aws.bucket.name}/${aws.input}","s3://${aws.bucket.name}/${aws.output}"],"Type":"Spark","Jar":"s3://${aws.bucket.name}/${jar.name}","ActionOnFailure":"TERMINATE_CLUSTER"}]' \
--configurations '[{"Classification":"spark", "Properties":{"maximizeResourceAllocation": "true"}}]' *
where job.name is PageRank.scala
aws.bucket.name is the name of bucket
jar.name is name of jar file
aws.input and aws.output are locations for input and output

These configurations give Spark all the required details to execute the source code.

Following are the methods are used in my program and their high-level description of how Spark processes the data:

- map: it returns a new data set by iterating over each element in source and implementing a function described on that element
- filter: it returns a new data set by returning only those elements in source that satisfy the condition specified in function
- persist: when called on a RDD, each node will save the portion of RDD which was computed on it and store it in memory

- flatMap: it returns a new data set. It is similar to map, but each element in source could be mapped to 1 or more items. It is like flattening a map
- reduceByKey: for a data set of type (key, value), reduceByKey will return a new data set of type (key, value), where the values in the source will be grouped by key, and a reduce function of type (value, value) => value will be implemented on it
- union: it returns a new data which will contain the union of elements from source and data set specified inside the method
- count: returns number of elements in data set
- join: when called on data set of types (key, value1) and (key, value2), it will perform a join operation based on keys and return a data set of type (key, (value1, value2))
- reduce: it is an action, and it reduces the data set by performing the function mentioned which will reduce 2 arguments into 1
- top: it will return top N elements in RDD based on it's the default ordering

Comparison between Hadoop MapReduce and Spark implementations:

Pre-Processing:

```
/* PairRDD: (PageName: String, LinkedPageNames: List[String])
  RDD represents pages and their linked page names
  Sequence of operations:
    - reads input
    - map & filter: for each line in the input removes null objects (pages having invalid urls)
    - map: converts each row to tuple of form (String, List[String])
    - persist: stores the RDD in memory
*/
val pageWithLinks = sc.textFile(args(0))
 .map(line => Bz2WikiParser.ParseLine(line))
 .filter(row => row != null)
 .map(row => (row.PageName, row.LinkedPageNames.toList))
 .persist()

/* PairRDD: (PageName: String, AdjList: List[String])
  RDD represents pages and their adjacency list. Nodes without any linked pages
  will have empty adjacency list
  Sequence of operations:
    - flatMap: for each node in adjacency list, emits an empty list
    - reduceByKey: reduces the adjacency lists for each page
    - union: unions the reduced data set with pageWithLinks RDD
    - reduceByKey: reduces the adjacency lists for each page
*/
val nodesAndAdjList = pageWithLinks.flatMap
```

```
{case(pageName, links) => links.map(node => (node, List[String]()))}
 .reduceByKey((adjList1, adjList2) => adjList1 ::: adjList2)
 .union(pageWithLinks)
 .reduceByKey((adjList1, adjList2) => adjList1 ::: adjList2)

val TOTAL_NODES = nodesAndAdjList.count()

/* PairRDD: (PageName: String, PageRank: Double)
  RDD represents pages and their initial page ranks
*/
var nodesAndPageRanks = nodesAndAdjList.map(node => (node._1, 1.0/TOTAL_NODES))
```

Spark: The above code performs pre-processing. It reads the input parses (using Bz2WikiParser parser) and creates adjacency list.
Hadoop: Pre-processing is handled in Hadoop jobs inside PreProcessing.java file in Assignment 3. In Mapper class, each line from input is passed, parsed using same parse as above, and it emits the page name and its linked page names. Along with this, to handling dangling nodes, it iterates over each node in linked pages and emits empty list for each node. In Reducers, we get list of list of strings for each node, and we combine it to get (pageName, adjList).

PageRank Calculations:

```
for(i <- 1 to ITERATIONS){
 var DELTA = 0.0

 /* Delta (sum of dangling nodes contributions)
   Sequence of operations:
     - filter: retrieves dangling nodes
     - join: joins filtered data set with nodesAndPageRanks RDD to obtain page ranks
          of dangling nodes
     - map: converts to RDD of type (Double) (only page ranks)
     - reduce: sums up the dangling nodes contributions
  */
 DELTA = nodesAndAdjList.filter(node => node._2.isEmpty)
   .join(nodesAndPageRanks)
   .map(node => node._2._2)
   .reduce((acc, n) => acc + n)

 /* PairRDD: (PageName: String, PageRank: Double)
  RDD represents pages and their page ranks after iteration i
  Sequence of operations:
    - join: joins page ranks and adjacency list
    - flatMap: for each node in adjacency list of a particular page, it emits
```

```
            it's outlinks contribution (it also emits (page, 0.0) to handle
            pages having no inlinks)
    - reduceByKey: for each page, it sums it's inlinks contribution
    - map: calculates the page rank of each node
 */
 nodesAndPageRanks = nodesAndAdjList.join(nodesAndPageRanks)
   .flatMap{
     case (pageName, (adjList, pageRank)) => {
       List(List((pageName, 0.0)), adjList.map(node => (node, pageRank / adjList.size))).flatten
     }
   }
   .reduceByKey((acc, n) => acc + n)
   .map(node => (node._1, ALPHA/TOTAL_NODES + (1 - ALPHA) * ((DELTA/TOTAL_NODES) + node._2)))
}
```

Spark: The above code computes page ranks for 10 iterations. It computes Delta in each iteration, calculates outlinks and inlinks contributions and in the end calculates the page rank for each node at ith iteration.

Hadoop: In Driver program, inside a for loop running for 10 iterations, a MapReduce job is called to calculate page ranks for ith iteration. In Mapper, for each node, it calculates its outlinks contributions i.e for each node in its adjacency list, it emits (node, pageRank/adjList.size). To handle dangling nodes, if the adjacency list is empty, it emits a (dummy key, pageRank). For each iteration i, δ is calculated in Reduce phase, and used by Mapper in iteration (i+1) to correct the page rank obtained in ith iteration. To achieve this, a global counter is used that is updated by all Reduce calls for dangling nodes. The only additional step taken in the approach I have taken, is to include an extra Map job to correct the pagerank of each node after the 10th iteration. So, the final correct pagerank values are generated in 11th iteration (where Reduce task is set to 0).

TOPK Job:

```
/* Array[(String, Double)]
  Represnts the top-100 pages along with their ranks
  Sequence of operations:
    - map: swaps position of page ranks and pages => Array[(Double, String)]
    - top: takes top 100 elements from array
    - map: swaps position of page and page ranks => Array[(String, Double)]
*/
val top100PageRanks = nodesAndPageRanks.map(node => node.swap).top(TOP_K).map(node =>
node.swap)

/* Converts top100PageRanks to RDD and writes to file */
```

```
sc.parallelize(top100PageRanks, 1).saveAsTextFile(args(1))
```

Spark: The above code computes top 100 pages with highest page ranks.
Hadoop: A MapReduce job is executed to calculate top k pages. For this, Reducer is set to 1. Each Mapper emits a local top 100 winners having the highest page ranks. Reducer receives (local 100)*#machine pages. There, it calculates a global top 100. Both Mappers and Reducer keep top 100 pages in PriorityQueue, and as soon as 101th record is pushed, it removes the 101th from the queue maintaining top 100 at any point.

Advantages and Shortcomings:
Both Spark and Hadoop can be used to implement PageRank algorithms, with Spark implementation being faster than Hadoop implementation. Spark has a very flexible API and can be used to implement lots of iterative algorithms on huge data sets. RDD's and storing the data sets in memory gives Spark the advantage of faster access and saves lot of time spent by Hadoop and MapReduce to read and write data from disk. Due to this, Spark has a less disk data footprint. It will write to the disk only when memory is full, else it will store everything in memory. While in Hadoop, everything is read and written to disk. Mappers read from disk, writes intermediate input to disk, which is read by Reducers and the output from Reducers is again written back to disk, increasing lot of disk data footprint and consumes lot of time for heavy I/O operations. Due to this, memory footprint is more in Spark, and less in Hadoop. The only memory usage in Hadoop will be storing object, variable and counters in memory. Because of transformations and actions operations in Spark, the source code verbosity is very less, since many operations are combined in map or filter in compact form.

Performance Comparison

Running Times:

	6 m4.large machines	**11 m4.large machines**
Spark	1 hour 30 minutes 58 seconds	46 minutes 6 seconds
MapReduce Hadoop	1 hour 11 minutes 7 seconds	37 minutes 59 seconds

Spark is faster than Hadoop, even though from above comparison, it seems Hadoop is faster than Spark. The time taken for Pre-Processing job in Spark is way more than Hadoop. This is because in Hadoop, pre-processing was done parallely on different machines. The input was split across different Mappers, each would pre-process the data, handle dangling nodes, and Reducers will combine them and generate the adjacency lists. While here in Spark, pre-processing is done sequentially. Each line is scanned and sent to the parser to be pre-processed sequentially. Due to this, here, Spark becomes slower than Hadoop. When compared only the time to compute PageRank and TopK, in Spark is very fast compared to Hadoop.

Top-100 Pages:
The results from both Spark and MapReduce execution (for both datasets) are same. The only difference is that their page ranks differ by last few digits. This is because essentially logic of pagerank for both Spark and Mapreduce execution is the same, and hence the results are bound to be same. The pageranks differ by last few digits because it depends how precision works on both systems. The following top-100 pages and their ranks (for both systems) back-up the observations.

Simple Dataset- MapReduce Hadoop:
United_States_09d4   0.0051890090002740434
Wikimedia_Commons_7b57   0.00480676647470988
Country   0.003940284687713574
England   0.0027524814361112155
Water   0.0026878096234471574
Animal   0.0025540875651497643
City   0.0025108240807830287
United_Kingdom_5ad7   0.002358647093612773
Germany   0.002350401697711995
Earth   0.0023247348599551684
France   0.0023236079471426027
Europe   0.002038097037168201
Wiktionary   0.0017538842142764614
English_language   0.0017496771217548222
Government   0.0017323446521037042
Computer   0.001716840484713746
India   0.0017131709183853
Money   0.0016673836980231798
Japan   0.0015516905685357793
Plant   0.0015235595093602682
Italy   0.001507433090498333
Canada   0.0014814073434532187
Spain   0.0014711236922238576
Food   0.0014246868489679767
Human   0.0014120970062699617

China 0.0013967150612732362
People 0.0013822485250560876
Australia 0.0013298542407507953
Asia 0.0012844361711364049
Capital_(city) 0.0012742684212522326
Television 0.0012649972257606518
Sun 0.0012602100811783014
Number 0.0012432362289291035
State 0.0012403756814549144
Sound 0.0012352116672222275
Science 0.0012325431753597168
Mathematics 0.0012310566392958523
Metal 0.001192304623749709
Year 0.0011770925835108761
2004 0.001173357313768757
Language 0.001150165884858011
Russia 0.0011461817792128453
Wikipedia 0.001123330280988467
Religion 0.0010985666999662946
19th_century 0.0010965391417803436
Music 0.0010874313232146736
Scotland 0.0010548007350065563
20th_century 0.0010537049832591268
Greece 0.0010492227329348632
Latin 0.0010298606131876865
London 0.00102735544285155
Greek_language 0.001004357256650529
Energy 9.990118103796386E-4
World 9.863508479979037E-4
Centuries 9.759058651368076E-4
Culture 9.452039652115251E-4
History 9.364696034256512E-4
Liquid 9.145230968002311E-4
Netherlands 9.057245076491723E-4
Planet 9.049322622392159E-4
Light 9.016763526865974E-4
Society 9.014920621454229E-4
Atom 8.900226406531608E-4
Wikimedia_Foundation_83d9 8.88440070776325E-4

Scientist   8.883836105737015E-4
Image   8.876884860222222E-4
Law   8.862908055986277E-4
Geography   8.788451614551093E-4
List_of_decades   8.785742942839124E-4
Uniform_Resource_Locator_1b4e   8.618845063634374E-4
Africa   8.605699671526503E-4
Turkey   8.448863678892099E-4
Inhabitant   8.30479488232508E-4
Capital_city   8.230488140439364E-4
Plural   8.215155955104328E-4
Electricity   8.137230016666818E-4
Poland   7.972379043155155E-4
Building   7.971238925722246E-4
Car   7.946540606240864E-4
Sweden   7.917125562342923E-4
Book   7.914884705321319E-4
Biology   7.869328964315926E-4
War   7.708172945482264E-4
Chemical_element   7.681607959198563E-4
God   7.609357218915576E-4
North_America_e7c4   7.562868644168624E-4
September_7   7.547781812642647E-4
Website   7.462973500605942E-4
Nation   7.426671526407832E-4
Politics   7.397103787590738E-4
2006   7.332900172260957E-4
Fish   7.322371112911346E-4
Species   7.308711176294948E-4
Mammal   7.216744135950795E-4
Island   7.178090203037469E-4
Portugal   7.171070596607501E-4
Gas   7.155515366540768E-4
River   7.115777513010706E-4
Switzerland   7.061075074386641E-4
World_War_II_d045   7.020304931583214E-4

Simple Dataset- Spark:
(United_States_09d4,0.005189009000274016)

(Wikimedia_Commons_7b57,0.004806766474709868)
(Country,0.003940284687713537)
(England,0.002752481436111213)
(Water,0.002687809623447136)
(Animal,0.0025540875651497473)
(City,0.0025108240807830188)
(United_Kingdom_5ad7,0.0023586470936127588)
(Germany,0.0023504016977119986)
(Earth,0.002324734859955148)
(France,0.0023236079471425967)
(Europe,0.002038097037168188)
(Wiktionary,0.0017538842142764558)
(English_language,0.001749677121754813)
(Government,0.0017323446521036894)
(Computer,0.0017168404847137408)
(India,0.0017131709183852923)
(Money,0.0016673836980231685)
(Japan,0.0015516905685357763)
(Plant,0.0015235595093602604)
(Italy,0.0015074330904983266)
(Canada,0.001481407343453213)
(Spain,0.0014711236922238542)
(Food,0.0014246868489679694)
(Human,0.001412097006269954)
(China,0.001396715061273228)
(People,0.001382248525056078)
(Australia,0.0013298542407507935)
(Asia,0.0012844361711363953)
(Capital_(city),0.0012742684212522257)
(Television,0.0012649972257606488)
(Sun,0.0012602100811782918)
(Number,0.0012432362289290957)
(State,0.001240375681454904)
(Sound,0.0012352116672222162)
(Science,0.001232543175359708)
(Mathematics,0.001231056639295848)
(Metal,0.0011923046237497033)
(Year,0.001177092583510871)
(2004,0.0011733573137687491)

(Language,0.0011501658848580036)
(Russia,0.0011461817792128388)
(Wikipedia,0.0011233302809884565)
(Religion,0.0010985666999662885)
(19th_century,0.0010965391417803376)
(Music,0.0010874313232146714)
(Scotland,0.0010548007350065524)
(20th_century,0.0010537049832591194)
(Greece,0.001049222732934861)
(Latin,0.0010298606131876804)
(London,0.0010273554428515473)
(Greek_language,0.0010043572566505211)
(Energy,9.990118103796295E-4)
(World,9.86350847997895E-4)
(Centuries,9.759058651368015E-4)
(Culture,9.452039652115159E-4)
(History,9.364696034256441E-4)
(Liquid,9.14523096800225E-4)
(Netherlands,9.0572450764917E-4)
(Planet,9.049322622392072E-4)
(Light,9.016763526865905E-4)
(Society,9.014920621454138E-4)
(Atom,8.900226406531535E-4)
(Wikimedia_Foundation_83d9,8.884400707763155E-4)
(Scientist,8.883836105736957E-4)
(Image,8.876884860222128E-4)
(Law,8.86290805598622E-4)
(Geography,8.788451614551008E-4)
(List_of_decades,8.785742942839061E-4)
(Uniform_Resource_Locator_1b4e,8.61884506363428E-4)
(Africa,8.60569967152646E-4)
(Turkey,8.448863678892028E-4)
(Inhabitant,8.30479488232518E-4)
(Capital_city,8.230488140439359E-4)
(Plural,8.215155955104245E-4)
(Electricity,8.137230016666766E-4)
(Poland,7.972379043155139E-4)
(Building,7.971238925722195E-4)
(Car,7.946540606240832E-4)

(Sweden,7.917125562342899E-4)
(Book,7.914884705321306E-4)
(Biology,7.869328964315866E-4)
(War,7.708172945482221E-4)
(Chemical_element,7.681607959198536E-4)
(God,7.609357218915552E-4)
(North_America_e7c4,7.562868644168596E-4)
(September_7,7.547781812642573E-4)
(Website,7.462973500605874E-4)
(Nation,7.426671526407754E-4)
(Politics,7.397103787590689E-4)
(2006,7.332900172260948E-4)
(Fish,7.322371112911313E-4)
(Species,7.308711176294924E-4)
(Mammal,7.216744135950762E-4)
(Island,7.178090203037438E-4)
(Portugal,7.171070596607465E-4)
(Gas,7.155515366540709E-4)
(River,7.115777513010679E-4)
(Switzerland,7.061075074386606E-4)
(World_War_II_d045,7.020304931583193E-4)

Full Dataset- MapReduce Hadoop:

United_States_09d4 0.002622883307725724
2006 0.0012284974115401603
United_Kingdom_5ad7 0.0012031345232478765
Biography 9.820750030583663E-4
2005 9.170453114331424E-4
England 8.802045052385164E-4
Canada 8.559019243189323E-4
Geographic_coordinate_system 7.716537557510497E-4
France 7.250155425564715E-4
2004 7.198917516046923E-4
Australia 6.804752357198294E-4
Germany 6.543395104727504E-4
2003 5.873910170218375E-4
India 5.834188603062393E-4
Japan 5.828499867966542E-4

Internet_Movie_Database_7ea7 5.335068278947029E-4
Europe 5.092684279282765E-4
Record_label 4.914575092040242E-4
2001 4.8700951198761414E-4
2002 4.8287569488536823E-4
World_War_II_d045 4.7805172711679826E-4
Population_density 4.703435073017509E-4
Music_genre 4.6719637178231063E-4
2000 4.646639470823794E-4
Italy 4.458079830035117E-4
Wiktionary 4.362093187146297E-4
Wikimedia_Commons_7b57 4.352977195224375E-4
London 4.3479475608461675E-4
English_language 4.184924190124008E-4
1999 4.0593676886523377E-4
Spain 3.6292229527105577E-4
1998 3.563095348985902E-4
Russia 3.438958027851477E-4
1997 3.3728506998715403E-4
Television 3.3629707612170177E-4
New_York_City_1428 3.3462856024990344E-4
Football_(soccer) 3.26148648392111E-4
1996 3.236267727634881E-4
Census 3.235551257749954E-4
Scotland 3.221898058120045E-4
1995 3.1015498593562127E-4
China 3.086407053476629E-4
Population 3.043214375168833E-4
Square_mile 3.04056159848861E-4
Scientific_classification 3.0401129926075406E-4
California 3.0166613242840735E-4
1994 2.9069059165481116E-4
Sweden 2.876209953787776E-4
Public_domain 2.8741664930924404E-4
Film 2.8626953981236556E-4
Record_producer 2.8411279243647825E-4
New_Zealand_2311 2.8310101842408004E-4
New_York_3da4 2.7888558279744717E-4
Netherlands 2.76671181070038E-4

Marriage 2.758133039378725E-4
1993 2.748027246452099E-4
United_States_Census_Bureau_2c85 2.7466711649185965E-4
1991 2.718970189676913E-4
1990 2.683246782500269E-4
1992 2.663656156472363E-4
Politician 2.6489459038802444E-4
Album 2.605577884155138E-4
Latin 2.6045696116246966E-4
Actor 2.583393632505134E-4
Ireland 2.5810098404018743E-4
Per_capita_income 2.5564270352658393E-4
Studio_album 2.5185786280951093E-4
Poverty_line 2.511650008893579E-4
Km² 2.4950708971558256E-4
1989 2.4688974587744404E-4
Norway 2.4086685269665328E-4
Website 2.3901474110413337E-4
1980 2.3532256907970485E-4
Animal 2.2937819007781048E-4
Area 2.292130433722194E-4
1986 2.2703360707975189E-4
Personal_name 2.2624086525437702E-4
Poland 2.261199647608192E-4
Brazil 2.256619988669503E-4
1985 2.2402853548642287E-4
1987 2.233052142740763E-4
1983 2.2175551866755638E-4
1982 2.210976597675 72E-4
French_language 2.193810555473214E-4
1981 2.1934770408862716E-4
1979 2.193298954042148E-4
1984 2.1878974281640544E-4
World_War_I_9429 2.1869361511968075E-4
1988 2.185763275043908E-4
Paris 2.180114096060794E-4
1974 2.179757176312975E-4
Mexico 2.156691801773946E-4
19th_century 2.118571806277182E-4

1970   2.1132376508534002E-4
January_1   2.1086786200188968E-4
USA_f75d   2.1070856929063453E-4
1975   2.0860252359153428E-4
1976   2.084679274023311E-4
Africa   2.0779879925956986E-4
South_Africa_1287   2.0736014983858958E-4

Full DataSet- Spark:
(United_States_09d4,0.0026228833077249552)
(2006,0.0012284974115398144)
(United_Kingdom_5ad7,0.001203134523247543)
(Biography,9.820750030580467E-4)
(2005,9.170453114328799E-4)
(England,8.802045052382592E-4)
(Canada,8.559019243186874E-4)
(Geographic_coordinate_system,7.716537557508232E-4)
(France,7.250155425562569E-4)
(2004,7.198917516044857E-4)
(Australia,6.804752357196347E-4)
(Germany,6.5433951047256E-4)
(2003,5.873910170216661E-4)
(India,5.83418860306065E-4)
(Japan,5.82849986796488E-4)
(Internet_Movie_Database_7ea7,5.335068278945448E-4)
(Europe,5.092684279281295E-4)
(Record_label,4.914575092038814E-4)
(2001,4.8700951198747466E-4)
(2002,4.8287569488523E-4)
(World_War_II_d045,4.780517271166618E-4)
(Population_density,4.7034350730161816E-4)
(Music_genre,4.6719637178217413E-4)
(2000,4.64663947082247E-4)
(Italy,4.4580798300338127E-4)
(Wiktionary,4.362093187145028E-4)
(Wikimedia_Commons_7b57,4.352977195223203E-4)
(London,4.3479475608449136E-4)
(English_language,4.184924190122787E-4)
(1999,4.0593676886511667E-4)

(Spain,3.6292229527094903E-4)
(1998,3.5630953489848745E-4)
(Russia,3.438958027850476E-4)
(1997,3.372850699870574E-4)
(Television,3.3629707612160505E-4)
(New_York_City_1428,3.346285602498086E-4)
(Football_(soccer),3.2614864839201614E-4)
(1996,3.2362677276339525E-4)
(Census,3.235551257749014E-4)
(Scotland,3.2218980581195163E-4)
(1995,3.1015498593553356E-4)
(China,3.0864070534757256E-4)
(Population,3.0432143751680005E-4)
(Square_mile,3.040561598487761E-4)
(Scientific_classification,3.0401129926066223E-4)
(California,3.016661324283208E-4)
(1994,2.906905916547277E-4)
(Sweden,2.8762099537869585E-4)
(Public_domain,2.8741664930915503E-4)
(Film,2.862695398122874E-4)
(Record_producer,2.8411279243639606E-4)
(New_Zealand_2311,2.83101018424E-4)
(New_York_3da4,2.788855827973679E-4)
(Netherlands,2.766711810699591E-4)
(Marriage,2.758133039377933E-4)
(1993,2.748027246451303E-4)
(United_States_Census_Bureau_2c85,2.7466711649178077E-4)
(1991,2.718970189676136E-4)
(1990,2.683246782499497E-4)
(1992,2.663656156471592E-4)
(Politician,2.6489459038793884E-4)
(Album,2.6055778841544065E-4)
(Latin,2.6045696116239707E-4)
(Actor,2.58339363250437E-4)
(Ireland,2.5810098404011446E-4)
(Per_capita_income,2.556427035265114E-4)
(Studio_album,2.518578628094355E-4)
(Poverty_line,2.511650008892861E-4)
(Km²,2.495070897155112E-4)

(1989,2.468897458773726E-4)
(Norway,2.408668526965835E-4)
(Website,2.390147411040667E-4)
(1980,2.353225690796357E-4)
(Animal,2.2937819007774315E-4)
(Area,2.2921304337215773E-4)
(1986,2.2703360707968542E-4)
(Personal_name,2.2624086525430522E-4)
(Poland,2.261199647607528E-4)
(Brazil,2.256619988668846E-4)
(1985,2.2402853548635762E-4)
(1987,2.2330521427401162E-4)
(1983,2.2175551866749114E-4)
(1982,2.2109765976750676E-4)
(French_language,2.1938105554725865E-4)
(1981,2.1934770408856206E-4)
(1979,2.1932989540414982E-4)
(1984,2.1878974281634153E-4)
(World_War_I_9429,2.1869361511961768E-4)
(1988,2.1857632750432713E-4)
(Paris,2.1801140960601523E-4)
(1974,2.1797571763123335E-4)
(Mexico,2.1566918017733362E-4)
(19th_century,2.1185718062765997E-4)
(1970,2.113237650852778E-4)
(January_1,2.1086786200182859E-4)
(USA_f75d,2.1070856929057352E-4)
(1975,2.0860252359147246E-4)
(1976,2.084679274022694E-4)
(Africa,2.0779879925950925E-4)
(South_Africa_1287,2.0736014983852984E-4)