Assignment 5 - CS 6240

- Sriharsha Srinivasa Karthik Kaipa, Sec 01

Running the code:

The source code for this assignment is available in the folder named “assignment5”. The “src” folder contains the final code. Make sure the inputs are in the folder “`input`” alongside the “src” folder. Run instructions for the code are as follows:

- For cleaning the project, building the jar and then running it from the command line, please use the following rules:
  For row by column
  `:~$ make pagerankrc`
  For column by row
  `:~$ make pagerankcr`
- For cleaning the project, use the following rule
  `make clean`
- For building the jar, use the following rule
  `make jar`

Pseudo Code:

```
--------------- Preprocessing ---------------
Adjacency list creation: Same as Assignment 3
LabelMapper:
map (line) {
     emit(null, node);
}
LabelReducer:
reduce (null, list = [node1, node2…]) {
     for node in list {
          emit(node, index);
     }
}
DanglingMapper:
setup () {
     map = map(node, label);
}
map (line) {
     if adjacencyList is empty {
          emit(node);
     }
}
```

MatrixMapper:

```
setup () {
     map = map(node, label);
}
map (line) {
     for link in adjacencyList {
          if link is real { // not ghost node
               emit(link, {node, realAdjacencyCount});
          }
     }
     emit(node, {node, 0});
}
```

RankMapper:

```
map (line) {
     emit(label, 1/COUNT);
}
```

--- Row by column ---

PageRankMapper:

```
map (line) {
     emit(row, {column, ajdCount});
}
```

PageRankReducer:

```
setup () {
     map = map(label, rank);
     set = set(dangling);
     dangC += map.get(dangling);
}
reduce (row, list = [{column1, contrib1}...]) {
     for col, con in list {
          contrib += map.get(col) / con;
     }
     rank = 0.15/COUNT + 0.85 * (contrib + dangC/Count);
     emit(row, rank);
}
```

--- Column by row ---

PageRankColMapper:

```
map (line) {
     emit(column, {row, ajdCount});
}
```

PageRankColReducer:

```
setup () {
     map = map(label, rank);
}
```

```
reduce (column, list = [{row1, contrib1}...]) {
      for row, con in list {
            emit(row, map.get(column)/con);
      }
}
```
PageRankAggrMapper:
```
map (line) {
      emit(node, partialRank);
}
```
PageRankAggrReducer:
```
setup () {
      map = map(label, rank);
      set = set(dangling);
      dangC += map.get(dangling);
}
reduce (node, list = [con1, con2…]) {
      for con in list {
            contrib += con;
      }
      rank = 0.15/COUNT + 0.85 * (contrib + dangC/Count);
      emit(node, rank);
}
--------------- Top K ---------------
```
TopKMapper:
```
setup () {
      HashMap hs = new HashMap();
}
map (line) {
      hs.put(node, rank);
}
cleanup () {
      hs = sortByValue(hs);
      count = 0;
      for each {node, rank} in hs {
            emit(null, {node, rank});
            count++;
            if count >= k
                  break;
      }
}
```
TopKReducer:
```
setup () {
      map = map(label, node);
}
```

```
reduce (null, list = [{n1, r1}, {n2, r2}...]) {
     HashMap hs = new HashMap();
     for each {node, rank} in list {
          hs.put(node, rank);
     }
     hs = sortByValue(hs);
     count = 0;
     for each {node, rank} in hs {
          emit(null, {map.get(node), rank});
          count++;
          if count >= k
               break;
     }
}
```

Explanation:

Assume, for the purpose of this section, that the following is the adjacency list that is formed from the data:

```
A : [B, C, D]
B : [C]
C : []
```

D is a ghost node and C is a dangling node.

Serialization:

The matrix formed from the above graph is stored as follows:

```
Labels:
(A, 0); (B, 1); (C, 2)
Dangling;
2
Matrix:
(0, 0, 0); (1, 0, 2); (2, 0; 2)
(1, 1, 0); (1, 2, 1)
(2, 2, 0)
Vector:
(0, ⅓); (1, ⅓); (2, ⅓)
```

Dangling nodes and ghost nodes:

As can be seen from the labels, matrix and vector created, ghost nodes are considered and removed without harming the pagerank calculation at all.

By keeping store the list of all dangling nodes in the graph and also storing ranks vector in cache, we can always calculate the dangling node contribution whenever needed. Since the list and vector are small in size, there is not much overhead. The calculation for and addition of dangling node contribution is handled in the same reduce job as the one for creating new rank for each node.

Performance Comparison:

Matrix based:

1 Master 10 Core:

Row by column:

Step 1 + 2: 4:25 + 0:38 = 5:03 (m:s)
Step 3: 4:37 (m:s)
Step 4: 0:55 (m:s)

Column by row:

Step 1 + 2: 4:19 + 0:35 = 4:54 (m:s)
Step 3: 7:17 (m:s)
Step 4: 0:46 (m:s)

1 Master 5 Core:

Row by column:

Step 1 + 2: 8:10 + 0:37 = 8:47 (m:s)
Step 3: 6:14 (m:s)
Step 4: 0:52 (m:s)

Column by row:

Step 1 + 2: 10:01 + 0:36 = 10:37 (m:s)
Step 3: 11:26 (m:s)
Step 4: 1:14 (m:s)

Adjacency list based:

1 Master 10 Core:

1 iteration: 4:21 (m:s)

1 Master 5 Core:

1 iteration: 9:22 (m:s)

Comparison:

As can be seen from the above run times, it is clear that for 10 slave configuration, there is not much difference between row by column and adjacency list but there is a significant difference between the former two and column by row. This is not surprising as column by row partition takes an additional job to aggregate the partial ranks for each node and this causes it to take longer.

The same can't be said for 5 slave configuration, as row by column partitioning takes a significantly lesser amount of time than the adjacency list approach. There is no change in the comparison between the former two and column by row partitioning.

The run time mismatch between row by column and adjacency list for 10 and 5 slaves may be due to the amount of data that is being replicated into cache in both cases. There is a large amount of cache replication in 10 slaves as compared to 5 slaves so the speedup provided by the matrix method is dwarfed a little in higher machine count. Hence we see a significant improvement in 5 slave configuration.

Sample Page Rank:

United_States_09d4 0.006296650189808671
Wikimedia_Commons_7b57 0.00480196269427849
Country 0.0039273765228700795
England 0.0026852951799648285
Europe 0.0026253569146424387
United_Kingdom_5ad7 0.0026194465503446324
Germany 0.002607788833316004
Water 0.0025871803210273598
France 0.002539347902483243
Animal 0.0024565486341748405
Earth 0.002434167763211277
City 0.002409192135172867
Week 0.0020150591598035767
Asia 0.0019346926025019795
Sunday 0.0018744919669770436
Wiktionary 0.0018635870791926601
Monday 0.0018472416026991277
Money 0.0018430346538627648
Wednesday 0.0018128940265568747
Plant 0.0018105648771827177
Friday 0.0017846279081085077
Saturday 0.0017645371857353657
Computer 0.0017605398801726615
English_language 0.001754213628263421
Thursday 0.0017418603552429466
Tuesday 0.0017294187955668252
Italy 0.0017279687674417218
Government 0.0017173461473781876
India 0.001710828165343039
Number 0.0015892187203288065
Spain 0.0015793269841214857
Japan 0.0015222624221801538
Canada 0.0015080006475424884
Day 0.0014756542534689226
People 0.0014516217304191966
Human 0.0014222495655398177
Wikimedia_Foundation_83d9 0.001386357433446705
China 0.0013721077633463375
Australia 0.0013705180450502338
Energy 0.001334458660428413
index 0.0013233200027403247
Food 0.0013163893113271275
Sun 0.0012958928216328558
Science 0.001292812838710632
Mathematics 0.0012772026957681552

Television 0.0012272634250130776
Capital_(city) 0.0012041715857429875
Russia 0.001188902187029205
State 0.0011759004785955565
Music 0.0011598837933720174
Year 0.0011381248775177892
Greece 0.0011181497484562487
Language 0.0011173171973783385
Scotland 0.0011088697040638423
Metal 0.0010823842711338541
Wikipedia 0.0010795727603524052
2004 0.001071796466505998
Greek_language 0.001067442815887234
Planet 0.0010349513811457871
Sound 0.001030561461496137
Religion 0.0010268349059565991
London 0.001022255364491143
Africa 0.00101251626070831
20th_century 9.642075915056658E-4
Poland 9.63069241656455E-4
Law 9.534793569520846E-4
Geography 9.521310002181814E-4
19th_century 9.430318897789648E-4
Liquid 9.375220366842848E-4
World 9.306962973086503E-4
Society 9.144619439862316E-4
Scientist 9.134459116223295E-4
Latin 8.836415048201389E-4
History 8.836069994973911E-4
Atom 8.772753905702734E-4
Sweden 8.762373678823626E-4
War 8.740898622054344E-4
Light 8.653924954219069E-4
Netherlands 8.636703614450521E-4
Culture 8.560385692774665E-4
Building 8.425185233457696E-4
God 8.310562719699325E-4
Turkey 8.285393320533906E-4
Plural 8.198794625148663E-4
Information 8.192533601869418E-4
Inhabitant 8.100025708538454E-4
Centuries 8.083087218192954E-4
Portugal 8.000383218562866E-4
Chemical_element 7.918822379977739E-4
Capital_city 7.91743821422428E-4
Denmark 7.85818971289876E-4

Austria 7.7886634064188E-4
Cyprus 7.658906749889934E-4
University 7.599628031572254E-4
Ocean 7.589568719492463E-4
Book 7.562792168407448E-4
North_America_e7c4 7.561594551450316E-4
Species 7.559462711972718E-4
Disease 7.53157217512551E-4
Biology 7.477529182353731E-4

Full Page Rank:
United_States_09d4 0.002947383615678752
2006 0.0026787336526111092
United_Kingdom_5ad7 0.001413125409249269
2005 0.0012370195978425999
France 9.973746722249204E-4
Biography 9.601797189543265E-4
Canada 9.250065358843792E-4
England 9.071803345436772E-4
2004 8.690733596936072E-4
Germany 8.186708220744868E-4
Geographic_coordinate_system 7.759711199291749E-4
Australia 7.45666489379422E-4
2003 6.947132565923716E-4
Japan 6.713019617707692E-4
India 6.568462933479498E-4
Italy 5.743757080967515E-4
2001 5.594819723298438E-4
2002 5.518355815794008E-4
Europe 5.426738636885612E-4
Internet_Movie_Database_7ea7 5.299089441891144E-4
2000 5.199205511475497E-4
World_War_II_d045 5.086509067659417E-4
Spain 4.90417484137664E-4
Population_density 4.801644564990879E-4
London 4.8014053083554815E-4
English_language 4.6713066379931983E-4
1999 4.620448968556777E-4
Record_label 4.442908210801266E-4
Russia 4.406597015248752E-4
Race_(United_States_Census)_a07d 4.2190118057875336E-4
Wiktionary 4.17680524496329E-4
Wikimedia_Commons_7b57 4.1419788057232105E-4
1998 3.9700752869128894E-4
Football_(soccer) 3.844997166408916E-4
1997 3.7871603293728677E-4

Sweden 3.7847360806186507E-4
Music_genre 3.7508250171666143E-4
Scotland 3.6938831045234103E-4
New_York_City_1428 3.679826660054966E-4
1996 3.541291301765962E-4
Television 3.407489737448791E-4
Square_mile 3.4027941987332277E-4
Census 3.3669938523317255E-4
1995 3.343453638881229E-4
Netherlands 3.294138491706805E-4
China 3.2610650741690575E-4
California 3.25125713358917E-4
1994 3.198965205289172E-4
New_Zealand_2311 3.1973660769353627E-4
Poland 3.1227533562711767E-4
Norway 3.0726703714075896E-4
1991 3.070106480666682E-4
Public_domain 3.0590231974477773E-4
Population 3.048682041025421E-4
1993 3.0279057663500417E-4
1990 3.0249344516484946E-4
Brazil 2.992180404556936E-4
New_York_3da4 2.9266549418636414E-4
1992 2.902754380556177E-4
United_States_Census_Bureau_2c85 2.860218043498368E-4
Ireland 2.8271880349823014E-4
Film 2.8073549823580017E-4
Actor 2.7695674740131924E-4
January_1 2.762842853908752E-4
Mexico 2.754133644047081E-4
Scientific_classification 2.7321000370569083E-4
1989 2.7269785464652973E-4
French_language 2.6675354022930866E-4
1980 2.662700621094955E-4
Latin 2.6594270156768445E-4
Switzerland 2.629776271441635E-4
Marriage 2.6195181132507506E-4
1986 2.584502168701551E-4
Politician 2.563893329619361E-4
Paris 2.555581750363345E-4
1979 2.52881456544426E-4
1982 2.52018556803077E-4
Area 2.519874327787801E-4
1985 2.5197970277902865E-4
1981 2.5170472123217647E-4
1974 2.4927987936944785E-4

```
Per_capita_income    2.4708085699782047E-4
Portugal    2.4636207745351503E-4
1984    2.461009731968013E-4
1983    2.459334718923809E-4
1987    2.4592140271412666E-4
South_Africa_1287    2.4489877090093815E-4
1970    2.4304157116965852E-4
1976    2.4055611099778473E-4
1988    2.3982153837462847E-4
Denmark    2.3864249169459033E-4
1975    2.385978989557132E-4
Album    2.3806679993070847E-4
1945    2.3780398845411055E-4
Greece    2.3734732031170438E-4
Austria    2.3676458828337035E-4
Record_producer    2.351306734936567E-4
Soviet_Union_ad1f    2.3458151221887556E-4
1969    2.3371502171618203E-4
1972    2.3264822300576E-4
```

The pageranks and values are not quite the same as that with adjacency list approach as that approach didn't handle ghost nodes very well while adjacency matrix method handles dangling and ghost nodes like a champ. The answers produced using the adjacency matrix are the most accurate.