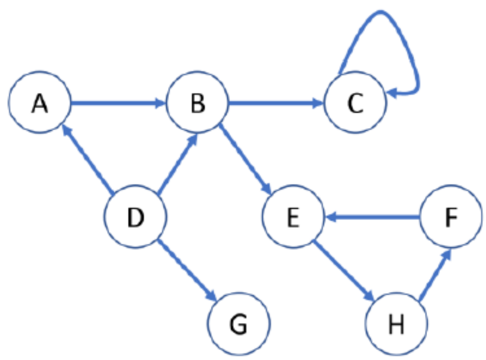
CS4225/CS5425 BIG DATA SYSTEMS FOR DATA SCIENCE

Tutorial 5: Graph and Test Practice

1. Given the following graph,

1) how many dead ends are there in the graph? For each dead end (if any), please indicate the set of vertices forming the dead end.

2) how many spider traps are there in the graph? For each spider trap (if any), please indicate the set of vertices forming the spider trap.



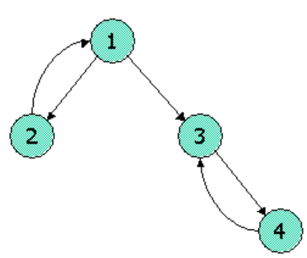
Answer:

1) Dead ends: {G},
2) Spider traps: {C}, {E, F, H}, {B, C, E, F, H}, {A, B, C, E, F, H}

2. True/False: In Topic-specific PageRank, random walker will teleport to any page with equal probability.

Answer: False. Random walker will only teleport to a topic-specific set of “relevant” pages.

3. Consider the following link topology.



Write down the Topic-Specific PageRank equations for the following link topology. Assume that pages selected for the teleport set are nodes 1 and 2 (where teleports go to either node with equal probability). Assume further that the teleport probability, (1 - β), is 0.3.

Recall the topic sensitive pagerank (TSPR) equations:

$$A_{ij} = \begin{cases} \beta M_{ij} + (1-\beta)/|S| & \textbf{if } i \in S \\ \beta M_{ij} + 0 & \textbf{otherwise} \end{cases}$$

Basically, the only difference here (compared to the non-topic sensitive case in the last 2 questions), is that the teleport terms are only distributed among the nodes in the teleport S, instead of being distributed over all nodes. Let r_1, r_2, r_3, r_3 denote the importance of the 4 nodes.

$r_1 = 0.7\ r_2 + 0.15$
$r_2 = 0.35\ r_1 + 0.15$
$r_3 = 0.35\ r_1 + 0.7\ r_4$
$r_4 = 0.7\ r_3$

4. Show pseudocode for the compute() function for the PageRank with teleport (β = 0.85) over vertices algorithm in Pregel / Giraph. Set the initial PageRank value as 1/N (N is the number of vertices), Run 30 iterations and then stop. You can (if you choose) use the functions: getValue(), setValue(), getNumVertices(), getSuperStep(), getOutEdgeIterator().

Answer:

```
Compute(v, messages):
if getSuperStep() == 0:
  v.setValue(1 / getNumVertices())
if getSuperStep() >= 1:
  sum = 0
  for m in messages:
    sum += m
  v.setValue(0.15 / getNumVertices() + 0.85 * sum)
if getSuperStep() < 30:
  sendMsgToAllEdges(v.getValue() / len(getOutEdgeIterator()))
else:
  voteToHalt()
```

Writing pseudo code is fine; no worries about the syntax. E.g. len(x) or x.size() or “size of x” are all fine.

5. On a certain large Spark cluster, Rose creates a data frame named **traceA**, and writes the shown program to process the trace. The **traceA** data frame keeps the logs, and each log entry represents the log information of one web page access, including various fields: **ip**, the IP address of the log entry and **time**, the amount of time of that access.

```
Line 0: maxSql = spark.sql("""
Line 1: SELECT ip, sum(time) as access_total
Line 2: FROM traceA
Line 3: WHERE time>0.1
Line 4: GROUP BY ip
Line 5: ORDER BY sum(time) DESC
Line 6: """)
Line 7: maxSql.collect()
```

For the above codes, what are the potential performance bottlenecks? Please identify which lines cause the bottleneck and justify your answer.

Answer:

Depending on whether the dataframe has been in the RAM, if not reading traceA (Line 1-3) may incur potential I/O cost

Depending on the size of data generated from Line 3, Line 4 & 5 can also be the bottleneck as both operations are wide transformation and thus require data shuffling through the network I/O.

If the grouped data after Line 4 is highly skewed (e.g. a super big size of data for certain ip address), Line 5 will have task straggler issue, i.e. certain tasks takes much longer time than the other tasks to complete.

Line 7 may return too many results. We can add LIMIT to the SQL query to limit the number of results.