1 The SocialRank algorithm

SocialRank is very similar to PageRank, except that it operates on social network graphs instead of hyperlink graphs. You may wish to refer to your notes on PageRank and its Map/Reduce implementation. However, we summarize the essentials here.

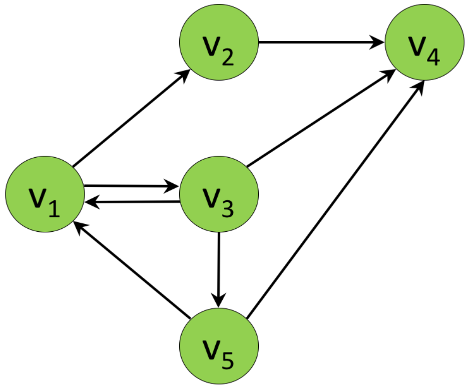
Let G := (V,E) be a social network graph, where each vertex $v \in V$ represents an individual user and each edge, or 'link', $(v_1,v_2) \in E$ represents the fact that user v_1 has listed v_2 as a friend. Note that friendship links are not necessarily reciprocal: if user A is a friend of user B, that does not necessarily mean that user B is a friend of user A. The goal of Social rank is to assign a *rank* r_i to each vertex v_i; the higher the rank value, the more 'influential' the algorithm considers the person to be.

SocialRank is an iterative algorithm, that is, it consists of several rounds. Let r_i^k be the rank of vertex i in round k. Before the first round, all ranks are initialized to one, that is, $r_i^0=1$. Given the ranks of round k, the ranks for round k+1 are calculated using the following formula:

$$r_i^{k+1} = d + (1-d)\sum_{j \in B(i)} \frac{1}{|N(j)|} r_j^k$$

Here, d is a constant (we will use d=0.15), $N(i):=\{j \mid (i,j)\in E\}$ is the set of i's friends, and $B(i):=\{j \mid (j,i)\in E\}$ is the set of users/vertices that have i as a friend. The algorithm stops when all the ranks have converged, i.e., when $|r_i^k - r_i^{k-1}|$ is small for all i.

1.1 A simple example



Your algorithm will run on a data set that contains more than 5 million users. This data set is too large for debugging because a) each run will take a long time, and b) we do not know up front what the correct answer should be. Therefore, we begin with the simple example shown on the right, which consists of just five vertices. In this example, V={1,2,3,4,5} and E={(1,2),(1,3),(2,4), (3,1),(3,4),(3,5),(5,1),(5,4)}. If we set d=0.15 and run a single round of SocialRank, we get the following values:

i	1	2	3	4	5
r_i^1	0.858	0.575	0.575	1.708	0.433

If we run the algorithm until the ranks for all the vertices change by less than 0.001 compared to the previous round, we arrive at round 10 and the following values:

i	1	2	3	4	5
r_i^{10}	0.332	0.291	0.291	0.580	0.233

Not surprisingly, the user represented by vertex 4 is returned as the most influential.

1.2 Data format

Input format: In this assignment, we will assume that the social graph is initially available as a list of edges. That is, we will have a file that contains one line for each link, and each line contains a pair of numbers that represent the vertices that are connected by the link. (We termed this the *edge* representation versus the *adjacency list* representation in our lecture notes.) For example, the graph from above would be encoded as shown on the right. In practice, the data may be spread across multiple files because the data set is very large.

```
1    2
1    3
2    4
3    1
3    4
3    5
5    1
5    4
```

Output format: The goal is to produce a file that contains the social rank of each individual in the social network. There should be one line for each vertex, and each line should contain the vertex identifier and the social rank. The file should be *sorted by rank in reverse order*. For example, the final output from the example above would be encoded as shown on the right.

```
4    0.580
1    0.332
2    0.291
3    0.291
5    0.233
```

1.3 Implementation strategy

We provide two implementation hints. First, it is very important to choose a suitable intermediate format (i.e., the format in which the data is output after each round). The input format is not particularly suitable for this because, for instance, the rank is associated with an individual vertex and not with a single link. Also, the intermediate format must clearly preserve all the information in the original input; thus, we cannot simply output a list of vertices and their ranks because the next round would no longer have the links.

Second, SocialRank cannot be implemented efficiently with a single MapReduce job. We can, however, implement each round as a separate job in an iterative process. Thus, the output of round k will be used as the input of round k+1. In addition, we will need three additional types of jobs: One for converting the input data into our intermediate format, one for computing how much the rank values have changed from one round to the next, and one for converting the intermediate format into the output format.