Homework 10 (due 4/20)

DS-210 @ Boston University

Spring 2022

Before you start...

Collaboration policy: You may verbally collaborate on required homework problems. However, you must write your solutions independently without showing them to other students. If you choose to collaborate on a problem, you are allowed to discuss it with at most 2 other students currently enrolled in the class.

The header of each assignment you submit must include the field "Collaborators:" with the names of the students with whom you have had discussions concerning your solutions. If you didn't collaborate with anyone, write "Collaborators: none." A failure to list collaborators may result in a credit deduction.

You may use external resources such as software documentation, textbooks, lecture notes, and videos to supplement your general understanding of the course topics. You may use references such as books and online resources for well known facts. However, you must always cite the source.

You may **not** look up answers to a homework assignment in the published literature or on the web. You may **not** share written work with anyone else.

Submitting: Solutions should be submitted via Gradescope. The entry code is 3Y85PZ.

Grading: Whenever we ask for a solution, you may receive partial credit if your solution is not sufficiently efficient or close to optimal. For instance, if we ask you to solve a specific problem that has a polynomial–time algorithm that is easy to implement, but the solution you provide is exponentially slower, you are likely to receive partial credit.

Late submission policy: No extensions, except for extraordinary circumstances. We accept submissions submitted up to one day late, but we may deduct 10% of points.

Questions

What to submit: To solve the problem in this homework, you should use Rust. Your solution to the homework should consist of one zip file, which should contain the content of your project folder. Please remove all binaries before you submit your solution. The easiest way to achieve this is to run `cargo clean`.

1. **(30 points)** In this question, the plan is to compute an approximation to *PageRank*. PageRank was developed as a method for ranking webpages. Originally, the Google search engine used PageRank.

   PageRank is related to a *memoryless* process of walking over vertices of a directed graph. We refer to this process as *memoryless* because the next vertex depends only on the vertex at which we are

currently and does not depend on previous vertices. How do we then decide on our next vertex, i.e., how do we make one step in this walk? Suppose that the current vertex is v. We select the next vertex as follows:

- If v has no outgoing edges, jump to a uniformly random vertex in the entire graph.
- If v has at least one outgoing edge:
  - With probability $9/10$, select uniformly at random one of them and follow it.
  - Otherwise—i.e., with probability $1/10$—jump to a vertex selected uniformly at random from the entire graph.

Without going into details of how exactly PageRank is defined,[1] we can compute its approximation by simulating 100 random independent walks from each vertex. Each random walk should consists of 100 random steps as described above. Our approximation of PageRank for a vertex v is then the fraction of the random walks that terminated at v (i.e., the number of random walks that terminated at v divided by $100n$, where n is the number of vertices).

Input: Your program should read a directed graph from `data.txt`. The first line of the file contains n, the number of vertices. Vertices are labelled 0 through $n - 1$. Each line of the rest of the file describes one *directed* edge and consists of two vertex labels separated by a space. The corresponding edge is a directed edge from the first vertex to the second. Multiple edges between a pair of vertices are allowed on input. During the neighbor selection process, each of them should be treated as a separate edge.

Output: Print out the labels of the five vertices with highest approximate values of PageRank and their approximate PageRank values. (If the graph has less than five vertices, output all of them.)

Sample input and output:

For the following `data.txt`:

```
5
0 3
1 3
2 3
3 4
4 3
4 0
```

your program could output:

```
vertex 3: approximate PageRank 0.426
vertex 4: approximate PageRank 0.324
vertex 0: approximate PageRank 0.214
vertex 2: approximate PageRank 0.024
vertex 1: approximate PageRank 0.012
```

For the following `data.txt`:

[1] Formally, it is the *stationary distribution* of the random walk we just described.

```
10
0 9
1 9
2 9
3 9
4 9
5 9
6 9
7 9
8 9
```

your program could output:

```
vertex 9: approximate PageRank 0.518
vertex 7: approximate PageRank 0.063
vertex 0: approximate PageRank 0.06
vertex 2: approximate PageRank 0.059
vertex 4: approximate PageRank 0.057
```

Hint: By default, floating–point types do not come with an implementation of the `Ord` trait, so they cannot be used for sorting or in a binary heap.[2] To get around this, for this problem, you can use the number of times random walks terminated in each of the vertices to determine the top five. Once you know their labels, you can compute the approximation to PageRank by dividing each of these numbers by $100n$.

2. **(10 points)** You will receive additional 10 points if your solution uses at least one module that you placed in a separate file. The partition into modules should be meaningful for the problem being solved.

[2] NaN, i.e., Not a Number, is to be blamed for this.