ID2222
Lab Report 3
Philip Claesson (philipcl@kth.se) and Miguel Maricalva (miguelma@kth.se)

Streamed Graph Triangle Count with Reservoir Sampling

How to run

The assignment is handled in as a Jupyter Notebook. In the data directory, some sample data is provided.

Run the code by sequentially executing each block of the code.

Code structure

The code is an implementation of the paper *A space efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox* found at https://arxiv.org/pdf/1212.2264.pdf

The algorithm is implemented as a class. By instantiating the class, specifying path and hyperparameters, the algorithm is run.

Hyperparameters

Se and sw
The number of edges/wedges to keep in memory at any time. Higher value will increase computing time.

Results

Little Rock Lake data set:

```
se: 100, sw: 100
Estimated K to 0.21000000000000002, in fact 0.332
Estimated T to 7339.0, in fact 11292

se: 500, sw: 500
Estimated K to 0.28800000000000003, in fact 0.332
Estimated T to 9056.0, in fact 11292

se: 1000, sw: 1000
Estimated K to 0.342, in fact 0.332
Estimated T to 9304.0, in fact 11,292
```

Optional task for extra bonus:

1. **What were the challenges you have faced when implementing the algorithm?**

Time complexity was a problem since we implemented the algorithm in Python. We started with a very large data set, and eventually had to use a very small data set in order to be able to run the algorithm in reasonable time.

The main challenges, as the algorithm was given, was to implement and assure the correctness of the support methods - such as determining if a wedge and an edge form a triangle and to get all the unique wedges from a large set of edges.

2. **Can the algorithm be easily parallelized? If yes, how? If not, why? Explain.**
   Yes, the algorithm can easily be parallelized. The algorithm does not require the full graph to estimate the triangle count, but relies on just a randomly sampled fraction of the edges. For this reason, the algorithm could be parallelized into n different processes with every n:th new data point processed by each process. The results of the processes can be aggregated to get an estimate of the complete graph.

3. **Does the algorithm work for unbounded graph streams? Explain.**
   Yes, the algorithm is working online and the approximation is updated after each new edge, so it works for unbounded graph streams.

4. **Does the algorithm support edge deletions? If not, what modification would it need? Explain.**
   No, as the algorithm does not store the full graph in memory it has no capability of accounting for deletion of edges.