PPA-S25 hw6

Logistics

This homework consists of 100 points total, with points for individual tasks as indicated below. There are 2 coding tasks.

You should submit your work on Brightspace.

Package your solutions as follows:

```
# replace NYU_NET_ID with your ID, e.g., shw8119.tgz
$ tar czf NYU_NET_ID.tgz IntersectionCount.sml TriangleCount.sml
```

The resulting `.tgz` archive should contain only `IntersectionCount.sml` and `TriangleCount.sml`. To submit, upload this `.tgz` file.

Due Date: This submission is due at **5:00pm EST on Tuesday, Mar 11**. Course policy on late submissions is available on the course website. (https://cs.nyu.edu/~shw8119/courses/s25/3033-121-ppa/)

Setup

- Follow the instructions on the course website to access one of the `crunchy` compute servers.
- Install MaPLe. For the `crunchy` servers, you can:
  - Download https://cs.nyu.edu/~shw8119/courses/s25/3033-121-ppa/resources/mpl-v053.tgz and copy it to the server
  - Unpack by running `tar xzf mpl-v053.tgz`. This will create a directory `mpl-v053/`
  - The compiler, `mpl`, is located at `mpl-v053/bin/mpl`
  - Add `mpl-v053/bin` to your `PATH` so that you can access it easily: `export PATH="$(pwd -P)/mpl-v053/bin:$PATH"`. We recommend updating your `~/.bashrc` file or other configuration file as appropriate.

Intersection, Revisited

This homework assignment is primarily about graphs, but before we get there, we need to take a little detour.

We previously saw that, on binary search trees, we can efficiently implement functions such as *union*, *intersection*, and *difference* using only $O(m \log(1+n/m))$ work, where n and m ($n \geq m$) are the sizes of the bigger and smaller tree, respectively.

For similar functions on sorted sequences, in some cases, we can get the same bound. This will be an important primitive for the next part of the assignment.

Task 1 (40 points): In `IntersectionCount.sml`, implement the function `intersection_count cmp (s, t)` which counts the number of elements that appear in both `s` and `t`. Your implementation should have $O(m \log(1 + n/m))$ work and $O(\text{polylog}(n))$ span, where $n = \max(|s|, |t|)$ and $m = \min(|s|, |t|)$.

(To convince yourself of the cost bound, consider how your implementation relates to the code we saw in lecture for intersection.)

You may assume that the inputs are sorted. You may also assume that, within each input, there are no duplicates.

For example, on inputs $s = \langle 1, 2, 4, 5, 42 \rangle$ and $t = \langle 2, 3, 4, 5, 10 \rangle$, `intersection_count Int.compare (s,t)` should return 3, because there are 3 elements in common (2,4,5).

Triangle Counting

In a directed graph, there are two possible kinds of "triangles". We'll call them **pointed** and **cyclic**.

- In a **pointed** triangle, there are two edges pointing at the same vertex. Specifically, a pointed triangle is a group of three vertices u, v, w with three edges: $u \to v$, $v \to w$, and $u \to w$.
- In a **cyclic** triangle, every edge points at a different vertex. Specifically, a cyclic triangle is a group of three vertices u, v, w with three edges: $u \to v$, $v \to w$, and $w \to u$.

By counting these triangles (for example, in a social network graph), we can get a sense for how "tightly knit" a graph is.

Task 2 (60 points): In `TriangleCount.sml`, implement the function `triangle_count(g)` which takes a graph `g` and returns a tuple (t_p, t_c), where t_p is the number of pointed triangles, and t_c is the number of cyclic triangles. Your solution should have $O(N + M\delta)$ work and $O(\text{polylog}(N))$ span, where N is the number of vertices, M is the number of edges, and δ is the maximum degree (in-degree or out-degree) of any vertex.

See the file `lib-local/Graph.sml` for the graph interface. **You may assume that the graph has been preprocessed to store neighbors in sorted order**. That is, `Graph.out_neighbors(g,v)` always returns a sorted sequence, in $O(1)$ work and span. (Similarly for `in_neighbors(...)`.) You may also assume the neighbor sequences are free of duplicates, and that there are no self-loops (i.e., no edges where a vertex points at itself).

We recommend using the `intersection_count` function. Note that Task 2 will be graded independently of Task 1.

Hint: be careful with symmetries! How many times do you count each triangle?

Unit Testing

You can test your implementations as follows. Feel free to add more tests to the top of `test.sml`.

```
$ make test
$ ./test
```

There are test graphs inside of `test-graph/...`. Each file is a list of directed edges in human readable format. This is the same format that is commonly used in the SNAP dataset.

Download and play with some real graphs!

The SNAP dataset has a number of freely available graphs. Here are a few interesting ones in particular:

- https://snap.stanford.edu/data/cit-Patents.html A patent citation network, where each directed edge $p_1 \to p_2$ indicates that a patent p_1 cited some other patent p_2.
- https://snap.stanford.edu/data/soc-LiveJournal1.html A social network graph, taken from LiveJournal.
- https://snap.stanford.edu/data/com-Orkut.html A social network graph, taken from the short-lived Orkut social network.

You can run these through your implementation as follows:

```
# download the datasets
$ wget https://snap.stanford.edu/data/cit-Patents.txt.gz
$ wget https://snap.stanford.edu/data/bigdata/communities/com-orkut.ungraph.txt.gz
$ wget https://snap.stanford.edu/data/soc-LiveJournal1.txt.gz

# unpack
$ gunzip cit-Patents.txt.gz
$ gunzip com-orkut.ungraph.txt.gz
$ gunzip soc-LiveJournal1.txt.gz

# compile
$ make main

# run
$ ./main @mpl procs 32 -- cit-Patents.txt
Loading graph (if large, this might take a while...)
  Warning: graph contains 1 self-loops
Loaded graph in 4.3893s
num vertices 3774768
num edges    16518948
--------------------
warmup 0.0000
```

```
repeat 1
time 1.4619s

average 1.4619s
minimum 1.4619s
maximum 1.4619s
std dev 0.0000s
--------------------
num pointed triangles 7515027
num cyclic triangles  0
```

For reference, using 32 processors on `crunchy1`, Sam's code takes:

- Approximately ~1.5s on the patent citation network, to count 7515027 pointed triangles (no cyclic triangles).
- Approximately ~90s on the LiveJournal social network graph, to count 924966203 pointed triangles and 238218098 cyclic triangles. (Note that this graph has a significant number of self-loops. If your code avoids counting self-loops as triangles, then you should get the same number of triangles.)
- Approximately ~120s on the Orkut social network graph, to count 627584181 pointed triangles (no cyclic triangles)