Applied algorithms 2017

Approximate neigborhood function

Rasmus Pagh

August 30, 2017

This problem builds on top of the HyperLogLog counters that were introduced earlier in the course. A feature of these counters is that they can be easily *merged*: Assuming that the same hash functions are used then given HyperLogLog counters for sets A and B we can easily compute the HyperLogLog counter for $A \cup B$ by taking the component-wise maximum of each entry in the array `M`. For example, if $A = \{1, 2, 3, 5, 7\}$ and $B = \{1, 2, 4, 8\}$ we can compute the HyperLogLog counter of $A \cup B = \{1, 2, 3, 4, 5, 7, 8\}$ directly from the HyperLogLog counters of A and B, without accessing the sets A and B directly. A HyperLogLog counter can be encapsulated in a class `ApproxSet` with methods `add(int x)` and `sizeEstimate()`. A basic Java implementation of `ApproxSet` using $m = 2^{10}$ is provided with this problem.

1. Extend `ApproxSet` with a method `addSet(ApproxSet A)` that has the same effect as calling `add(x)` on every element `x` that was ever added to `A`. Use the supplied implementation in Java, or your own HyperLogLog implementation.

To test on CodeJudge: Your implementation should read two lists of integers from `stdin` (space separated, one line per list) and estimate their sizes as well as the size of their union on `stdin`. It should be able to distinguish the case where the sets are almost disjoint (at most 20% in intersection) from the case where they are almost the same (at least 80% in intersection), outputting either `almost disjoint` or `almost same`.

Next, we consider the problem of determining the *median distance* between nodes in a connected, unweighted, undirected graph (V, E). This is the smallest integer d such that at most half of all vertex pairs $v_1, v_2 \in V$ have distance $d - 1$ or less and at most half of all pairs $v_1, v_2 \in V$ have distance $d+1$ or more. For a node $v \in V$ we can compute the distance from v to every node in V in time $O(|E|)$ using breadth-first search (BFS). Doing a BFS for every node $v \in V$ (an *all pairs shortest path* computation) one can compute the number n_i of pairs $v_1, v_2 \in V$ at distance $i = 0, 1, 2, 3, \ldots, |V| - 1$. This information suffices to compute the median distance: The smallest d such that $\sum_{i=0}^{d-1} n_i \geq |V|^2/2$. A Java implementation, `AllBFS`, is provided with this problem. It reads undirected graphs from `stdin` where each line encodes an edge (as two vertex identifiers separated by a space).

2. Perform experiments on `AllBFS` to determine the *scalability* of the solution. You should create inputs that test scalability by varying $|V|$, $|E|$ and the diameter of the graph, aiming to capture edge cases. You do *not* need to test correctness. Describe your findings using descriptions of the data sets (incl. code to generate data) and tables/plots of the observed performance.

Finally, we consider the following algorithm[1] for estimating the neighborhood sizes of a graph. The idea is to maintain a HyperLogLog counter `c[v]` for each node `v` that estimates the number of nodes that can be reached in d steps, for $d = 1, 2, \ldots.$ In each step we compute a new HyperLogLog counter `m[v]` for `v` by merging the counters of its neighbors in the graph.

```
for v in V do:
   initialize HyperLogLog counter c[v]
   add v to c[v]
end
d := 0
repeat
   reach := 0
   for v in V do:
      initialize HyperLogLog counter m[v]
      m[v].addSet(c[v])
      for each w adjacent to v in E do:
         m[v].addSet(c[w])
      end
      reach := reach + size estimate from m[v]
   end
   for v in V do:
      c[v] := m[v]
   end
   d := d+1
until reach >= |V|^2 / 2
return d
```

[1] From *HyperANF: Approximating the Neighbourhood Function of Very Large Graphs on a Budget* by P. Boldi, M. Rosa and S. Vigna, 2010.

3. Make an implementation of the above algorithm using the HyperLogLog class from question 1. Discuss possibilities for vectorizing the operation of adding two HyperLogLog counters.

To test on CodeJudge: Your implementation should read a the description of a graph from `stdin`, in the format described above. All node IDs will be integers, but not necessarily consecutive integers. Your program should output the median distance according to HyperANF.