CS5234: Combinatorial and Graph Algorithms

Problem Set 3

Due: September 1st, 6:30pm

Instructions. This problem set wraps up the first part of the class on streaming algorithms! The first question looks closely at the L0-sampler and tries to use it solve a problem for which it is not well suited! The second question asks you to develop an algorithm for testing whether a graph is bipartite. For both of these questions, an important aspect is communicating your ideas clearly. Take the time to think about the best way to explain your ideas.

- Start each problem on a separate page.
- Make sure your name is on each sheet of paper (and legible).
- Staple the pages together, and hand it in before class starts, or submit it on IVLE in the workbin. Alternatively, if you submit it late, you can put it in the envelope next to my office door (and send me an e-mail).

Remember, that when a question asks for an algorithm, you should:

- First, give an overview of your answer. Think of this as the executive summary.
- Second, describe your algorithm in English, giving pseudocode if helpful.
- Third, give an example showing how your algorithm works. Draw a picture.

You may then give a proof of correctness, or explanation, of why your algorithm is correct, an analysis of the running time, and/or an analysis of the approximation ratio, depending on what the question is asking for.

Advice. Start the problem set early—some questions take time. Come talk to me about the questions. (Different students have different questions. Some have questions about how to write a good proof. Others need pointers of designing an algorithm. Still others want to understand the material from lecture more deeply before applying it to the problem sets.) I'm here for you to talk to.

Collaboration Policy. The submitted solution must be your own unique work. You may discuss your high-level approach and strategy with others, but you must then: (i) destroy any notes; (ii) spend 30 minutes on facebook or some other non-technical activity; (iii) write up the solution on your own; (iv) list all your collaborators. Similarly, you may use the internet to learn basic material, but do not search for answers to the problem set questions. You may not use any solutions that you find elsewhere, e.g. on the internet. Any similarity to other students' submissions will be treated as cheating.

Exercises and Review (*Do not submit.*)

Exercise 1. In the lecture notes for Week 3, we describe a streaming algorithm for deciding if a graph is 2-connected. Give a streaming algorithm for deciding if a graph is k-connected. Prove that your algorithm is correct and that the probability of error is small. How much space is needed?

Exercise 2. In Week 3, we described a streaming algorithm for finding a 2-approximate minimum spanning tree which finds a spanning tree T whose weight is at most $2|T_{MST}|$, where $|T_{MST}$ is the weight of the MST. Describe how to generalize this algorithm to find a $(1+\epsilon)$-approximate minimum spanning tree, i.e., a tree T whose weight is at most $(1+\epsilon)|T_{MST}|$. How much space does this require?

Exercise 3. Imagine you have an algorithm A that estimates the number of whoozits in a graph G. (What is a whoozit? I don't know.) The algorithm guarantees that A gives the correct answer with probability at least $2/3$. Assume you know absolutely nothing else about A.

Can you use algorithm A to construct a new algorithm A' that is correct with probability $9/10$? If so, how? If not, why not?

Imagine instead that algorithm A returns a value X where $\mathrm{E}[X]$ is equal to the number of whoozits in graph G. Can you use algorithm A to construct a new algorithm A' that is correct with probability $9/10$? If so, how? If not, why not?

Standard Problems (to be submitted)

Problem 1. Perilous Proofs

Consider the following algorithm for reconstructing a graph from a stream. The algorithm should reproduce the entire graph after the stream is complete. First, we build a sketch using a single L0-sampler:

- We maintain one L0-sampler of the vector v which is indexed by pairs of nodes (as in the graph sketch).
- Whenever an element in the stream adds an edge (u, v), we increment the vector at position (u, v), updating the L0-sampler.
- Whenever an element in the stream deletes an edge (u, v), we decrement the vector at position (u, v), updating the L0-sampler.

After the stream is complete, we reconstruct the graph as follows. The goal of the reconstruction is to enumerate *all* the edges in the graph. Let E' be an initially empty set.

Repeat until the sampler returns NULL (i.e., no more edges in the graph):

- Let (u, v) be an edge sampled from the L0-sampler.
- Add (u, v) to the set E'.
- Instruct the sampler to decrement the vector at position (u, v), i.e., deleting edge (u, v).

We implement the sampler using the L0-sampler described in class, setting $\epsilon = 1/n^3$. That is, the sampler is correct with probability at least $1 - 1/n^3$, and using space $O(\log^4 n)$. We now give a proof that this algorithm works correctly with probability at least $1 - 1/n$:

Proof Let $G = (V, E)$ be the original graph. We prove the claim by induction. Our inductive hypothesis is that before and after each iteration of the outer loop, the L0-sampler is a valid sketch of the graph $G = (V, E \setminus E')$. Therefore, when the L0-sampler contains no more edges (ie., $E \setminus E' = \emptyset$), we conclude that E' must contain all the edges.

The base case of the induction is true by construction: after the stream completes, E' is empty, and the L0-sampler is a valid sketch of the graph G.

For the inductive step, notice that we find a valid edge (u, v), delete it from the sampler and add it to E'. Thefore the invariant continues to hold.

Finally, we observe that in each iteration, the probability that the L0-sampler fails is at most $1/n^3$. There are at most n^2 edges in the graph and hence at most n^2 iteration. Thus, by a union bound, we conclude that the probability that it fails and returns a bad answer at any point during the algorithm is at most $n^2/n^3 = 1/n$. □

Problem 1.a. Explain why this immediately seems likely to be wrong, without even analyzing the details. A very simple "sanity check" should suggest to you that there must be a mistake. (Always do a sanity check on your algorithms and calculations!)

Even without knowing anything about how the L0-sampler is implemented (and hence for all possible implementations that provide the specified guarantees), it is impossible for this algorithm to work. Why?

Problem 1.b. Explan in more detail what actually goes wrong. Where, precisely, does the algorithm fail? Please give a specific answer here (i.e., do not just write what goes wrong with the proof). At what point does the algorithm give a bad answer? How does the data structure fail? What is the maximum number of edges that the algorithm can reconstruct before it fails (even if you are very, very lucky)?

Problem 1.c. What is wrong with the proof? Where exactly does the proof make a mistake? Again, be as precise as possible, pointing to the exact location in the proof where something is wrong.

Problem 1.d. Assume you are given an L0-sampler, and you know all the hash functions used inside the L0-sampler. Can you design a graph for which the L0-sampler will fail with probability $> 1/2$? Think about this, but you do *NOT* need to submit a detailed answer. Do not write more than one or two sentences.

Problem 2. Is it bipartite?

In this problem, your job is to develop a streaming algorithm for determining whether a graph is bipartite.[1] Your algorithm should process a stream of edge additions and deletions, and then when the stream is complete, it should determine whether the graph is bipartite or not. It should return the correct answer with probability at least $1 - 1/n$. And it should use as little space as possible.

Describe your algorithm for determining whether a graph is bipartite. Your goal is to give a clear and succinct description of the algorithm, including: (i) a high-level overview, (ii) a precise explanation of how it processes the elements of the stream, and (iii) a precise explanation of how it determines whether a graph is bipartite. You are encouraged to use pictures and pseudocode as necessary to clearly explain the algorithm. (You may use as building blocks any of the algorithms we have studied in this class, but your description should be sufficiently clear that someone who took the class one year ago can understand.[2])

Then, prove that your algorithm is correct, i.e., that it determines whether the graph is bipartite. (State and prove any additional lemmas that help.) Give the probability of error, and show that the algorithm achieves it. Similarly, analyze the space complexity of your algorithm.

Hint: The following construction may be helpful. Given a graph $G = (V, E)$, construct a new graph $D = (V', E')$ as follows:

- For every node $u \in V$, add two nodes u_1 and u_2 to V'.
- For every edge (x, y) in E, add two edges (x_1, y_2) and (x_2, y_1) to E'.

(Perhaps, draw a picture!) You might try to relate the number of connected components in D to whether or not G is bipartite. (Try a few examples, and count the connected components in D.)

[1] Note that the graph does not have to be connected to be bipartite.

[2] For example, referring to "that claim you made last Thursday" may not be useful for someone who took the class last year!