Homework 1 – Due Thursday, September 10 before 10am on Angel

Instructions

• Solutions written in \LaTeX are strongly preferred, but you can upload any pdf files, including scanned hand-written solutions. Template latex files are on the course webpage.

• Collaboration is allowed and encouraged. However, each of you should think about a problem before discussing it with others and write up your solution independently. You may consult books and on-line sources to get information about well-known theorems, such as the Chernoff bound. But you are not allowed to look up solutions directly in papers or any other sources. And you must list all collaborators and sources!

• Correctness, clarity, and succinctness of the solution will determine your score.

Problems

1. A tournament is a directed graph that contains exactly one edge for each pair of vertices. There are two possible orientations for each edge. (Think of vertices as representing competitors in a tournament where every pair of competitors plays exactly one match, and the direction of the edge encoding the outcome of the match.) Suppose an \( n \)-vertex tournament \( G \) is represented by a \( n \times n \) matrix in which an entry \((u,v)\) is 1 if \( G \) contains the edge \((u,v)\) and -1 otherwise (that is, if \( G \) contains the edge \((v,u)\)). A sink is a node \( u \) such that \( u \)'s row contains only 1s.

Give an algorithm to find a sink in a tournament, if it exists, with \( O(n) \) queries to the matrix.

2. This is a collection of questions with short answers (at most several sentences per question).

(a) Recall that the relative Hamming distance between two strings is the fraction of character positions on which they differ. Give an algorithm for estimating the relative Hamming distance within additive error \( \epsilon \). What’s the running time of your algorithm?

(b) We define the property as a set \( P \) of objects (intuitively, the collection of objects that satisfy the property). For example, it can be the set of monotone functions of the form \( f : \{0,1\}^d \to \{0,1\} \). Recall that an \( \epsilon \)-tester for \( P \) has to, with probability at least 2/3,

• accept objects in \( P \);
• reject objects that are \( \epsilon \)-far from \( P \).

Consider properties \( P_1 \) and \( P_2 \) such that \( P_1 \subseteq P_2 \). Let \( q(\epsilon) \) be some function that represents query complexity. E.g., \( q(\epsilon) \) could be \( 1/\epsilon \) or \( 1/\epsilon^2 \).

Prove or disprove:

i. If \( P_1 \) has an \( \epsilon \)-tester that makes \( O(q(\epsilon)) \) queries then so does \( P_2 \).

ii. If \( P_2 \) has an \( \epsilon \)-tester that makes \( O(q(\epsilon)) \) queries then so does \( P_1 \).

iii. If \( P_1 \) has an \( \epsilon \)-tester that makes \( O(q(\epsilon)) \) queries then so does \( \overline{P_1} \).

3. In class we saw an algorithm, based on spanners, for testing if a list of numbers \( x_1, \ldots, x_n \) is sorted. Now we will design another algorithm for this problem, based on binary search.
**BinarySearchSortednessTest** \((n, \epsilon)\)

1. Pick an index \(i\) from \(\{1, 2, \ldots, n\}\) uniformly at random and read the number \(x_i\).
2. Perform a binary search for \(x_i\) and reject if you find any numbers out of order.

(a) Analyze the probability that \(\text{BinarySearchSortednessTest}(n, \epsilon)\) rejects a list that is \(\epsilon\)-far from sorted. **Hint:** Call a number \(x_i\) bad if it “fails” the binary search, i.e., Step 2 performed on \(x_i\) would reject.

(b) Prove that, with enough repetitions, \(\text{BinarySearchSortednessTest}(n, \epsilon)\) is an \(\epsilon\)-tester for sortedness.

(c) How does the query complexity and running time of this test compare to those of the test we saw in class?

(d) An algorithm is called nonadaptive if it makes its queries in advance, before getting any answers. Otherwise, it is called adaptive. Was the spanner-based algorithm adaptive? Is your new algorithm adaptive? **Hint:** One of them is adaptive and the other is nonadaptive. Can you modify the adaptive algorithm to make it nonadaptive without changing its running time?

4. In class we saw a tester for connectedness that made \(O(\frac{1}{\epsilon^2})\) queries. Give a tester for connectedness that makes \(O(\frac{1}{\epsilon \, \text{polylog} \left( \frac{1}{\epsilon} \right)})\) queries, where polylog \(m\) means that there is a constant \(c\) such that the expression is \(\log^c m\).

**Hint:** In class we proved that if a graph is \(\epsilon\)-far from connected, it has many small connected components. ("Many" was \(\geq \frac{\epsilon dn}{4}\) and "small" was of size \(\leq \frac{4}{\epsilon d}\).) Try to do a more careful accounting by considering small components of different sizes separately. I.e., break components into buckets according to their size (1, 2 to 3, 4 to 7, etc.) and prove that at least one of the buckets contains "enough" components. Modify the test accordingly.