Homework 15

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

5 October 2016

Please provide written or oral feedback on this and all future homeworks. :)

Problem 3:

As given by the problem, the coefficients of each polynomial can be represented with a single word (either as floating point numbers or integers). As such, we represent each polynomial as an array of words. For a polynomial function parameterized by the variable x, the ith element in the array is the coefficient of x_i. This representation should be fairly obvious and is probably uniform across all students turning in this assignment.

Now, we need to show that Multiplication of Polynomials reduces to Squaring of Polynomials. That is, $Mult \leq Sq$ where $Mult$ is the problem multiplication of polynomials and Sq is the problem squaring of polynomials. To do this, we need to give an algorithm for $Mult$ that occurs in poly time and makes use of the algorithm Sq.

Our key observation is that for two polynomials A, B,

$$(A+B)^2 = A^2 + 2AB + B^2$$

Solving for AB, you get

$$AB = \frac{(A+B)^2 - A^2 - B^2}{2}$$

As such, our algorithm is very simple: given A, B, return $\frac{(A+B)^2-A^2-B^2}{2}$.

Problem 6:

Let *undirectedISO*(G, H) be the algorithm which solves the undirected graph problem (if it exists).

Let *directedISO*(G, H) be the algorithm which solves the directed graph problem (if it exists).

Let *undirectedISOEdge*(G, H) be the algorithm which solves the undirected graph problem with all vertices of degree k (if it exists).

These problems are defined as in the homework. We will show that if any of these algorithms has a polynomial time algorithm, then they all do. We will prove this proving cyclic reduction:

$$\textit{undirectedISO} \leq \textit{directedISO} \leq \textit{undirectedISOEdge} \leq \textit{undirectedISO}$$

Proof. *undirectedISO* $\leq$ *directedISO*

First, we transform the inputs into *undirectedISO*, G, H into inputs into *directedISO* in poly time. When we transform the graphs, we ensure that the result of *directedISO* is true *iff* the result of *undirectedISO* is true.

This is our transformation: Copy the graph G into a new graph G', except where there is an undirected edge from n_1 to n_2 (where n_1 and n_2 are vertices, replace it with two directed edges, one from n_1 to n_2 and another one from n_2 to n_1. This transformation preserves the correctness of both functions because conceptually, an undirected edge is simply a representation of travel in either direction; this is obvious. □

Proof. *directedISO* $\leq$ *undirectedISOEdge*

This is the most interesting of the three reductions. Maintaining correctness proved tricky.

We have two phrases of the algorithm. The first phase is a preprocessing step. The second phase simply calls *undirectedISOEdge*.

First, prove that there exists a bijection between nodes in G and H such that each node in G is mapped to another node in H with the name tuple of in and out degrees. We do this with the following procedure.

Create two sets, S_G and S_H. For each vertex n in G, add the tuple (x, y) to S_G where x is the number of edges directed into n and y is the number of edges directed out of n. Repeat this step for S_H and H. Now, for each element in S_G, search in S_H for a tuple equal to that element. If it exists, remove both elements from their respective sets. If it's match does not exist, return value, because there does not exist a bijection between the graphs since no node has the same in-out relation. After exhausting S_G assert that there are S_H is empty. If it is not empty, return false, since there are more vertices in H than in G, and thus no mapping can be surjective.

Now, we know that when we start adding vertices during our transformation, we will not add vertices to our graphs that could change the iso-

morphism of our input graphs, because for every vertex we add to one graph during our transformation, we also add to the other graph, because there is a bijection between both graphs. Finally we discuss our transformation:

Transform the graph G as follows, and then transform H in the same manner. First, determine the maximum degree between G and H. Let k be that degree. Create a new, empty graph G'. For each vertex v in G, make a vertex v' in G'. Add undirected edges to v' for each edge in v. If the degree of v' is less than k, add new vertices $n_{v',1}, \ldots, n_{v',d-k}$ to v' where d is the degree of v'. This is to make sure that all vertices in G' have the same degree, while preserving isomorphism We know that no edge will be added during the transformation that would not also be added to an isomorphic copy because there exists a bijection between G and H mapping a vertex in G to another vertex in H with the same in/out degrees. Additionally, we know that this transformation successfully distinguishes between vertices connected with a directed edge in either direction and another vertex in a second graph with directed edge in one direction, because the in/out degrees must, again, match. This allows us to represent both cases in the same manner, because in the case where the graphs are not isomorphic, either both vertices will not have the same in/out degree, or the G' will not be isomorphic to H' because of the added vertices accounted for my the difference in in/out degree.

Finally, call *undirectedISOEdge*, passing in G', H', and k.

□

Proof. *undirectedISOEdge* $\leq$ *undirectedISO*

This reduction relies on a very simple preprocessing step. Iterate through all of the vertices of G, and count the edges on each of the vertices. If all of

the vertices do not individually has k edges, return false. Repeat the process for H. Finally, return $undirectedISO(G, H)$.

□

Problem 11:

Let *Clique(G, k* be the clique problem as described in the homework.

Let *2Clique(G', k* be the two-clique problem as described in the homework.

Clique is NP-Hard iff an NP-Complete problem reduces to it in poly time. Thus, we will show that $Clique \leq 2Clique$.

We found two different methods for transforming G to G' while preserving correctness between the two algorithms. We will present them sequentially.

Our first approach is as follows:

Given G, create empty graphs G'. For each vertex v in G, add two vertices in G', v_1 and v_2. For each edge on v, add an edge to v_1 and another edge to $v2$ such that the edge on v_1 connects to the corresponding edge in G', disjoint from all edges connected to v_2, and vice verse. The informal description of this process is to make two copies of G in G'. That way, 2Clique will always find one clique in the first subgraph of G' and the mirrored clique in the second, disconnected subgraph. Return *2CliqueG', k*.

Our second approach is as follows:

Given G, create empty graphs G'. For each vertex v in G, add a vertices in G', v'. Finally, add to G' a complete graph of of degree k disconnected to the rest of G'. That way *2Clique* will find the clique in the disconnected

complete subgraph which will count as the first of the two cliques. Finally, call *2Clique(G', k)* and return its result.