COT 4210 Final Exam Part D: Classes P/NP 5/4/2021 Solution

1) (8 pts) A subsequence of a string s is a subset of the letters of s in the same order as they appear in s. For example, if s = PO**L**YNOMI**A**LTI**ME**, then t = **LAME** is a subsequence of s, since we can create t using the 3rd, 9th, 13th and 14th characters of s, in that order. (Characters highlighted in red.) Define the language L as follows:

$$L = \{ (s, t) \mid s \text{ and } t \text{ are strings and } t \text{ is a subsequence of the string } s \}.$$

Prove that L belongs to the class P.

Solution

A greedy algorithm that runs in linear time of the input size correctly solves this problem. It's always better to match a letter from t as soon as possible in s because this leaves more options for future matches. For example, in the example above, matching the 'L' in LAME with the first L in "POLYNOMIALTIME" is at least as good as matching it with the second 'L' because in the former case, you must find the subsequence "AME" in "YNOMIALTIME" where as in the latter case, you must find the same subsequence in "TIME". Any subsequence you choose in the latter case is ALSO available in the former, so the former is at least as good as the latter. This type of argument, showing that a greedy choice provides at least as many options as any other choice is commonly used to prove the correctness of greedy algorithms.

Formally, our algorithm might look like this:

1. Set two counters i = 0, j = 0, where i is an index into s and j is an index into t.
2. Add one to i until t[j] equals s[i]. (Note: we only add 1 after checking, not before.)
    2a. if i equals the length of t before a match is found, reject.
3. Add one to both i and j.
    3a. if j equals the length of t, accept.
    3b. if i equals the length of s, reject.
4. Go back to step 2.

It should be fairly clear that the run time of this algorithm is O(|s|+|t|), or linear in the size of the input, since we iterate through both strings at most once.

Grading: 2 pts for setting counters/pointers at the beginning of both strings.
2 pts for greedily matching the first occurrence of t[0] in s.
1 pt for the rest of the details of the algorithm
2 pts for an informal proof of correctness
1 pt for the run-time analysis

2) (15 pts) In class it was shown that 3-SAT $\leq_P$ 3-COLOR. For this question, show that the polynomial time reduction can be done in the opposite order, namely that 3-COLOR $\leq_P$ 3-SAT. Note that looking at the proof that was shown in class is unlikely to help you on this question. (I am just trying to help you because I think people's natural instinct would be to look at the details of that proof, but those would not be helpful for this question and I want to save everyone the hassle of looking at that proof and losing time because of it.) Thus, here is what you must do:

(a) Start with an input, G, to the 3-COLOR problem, which is an arbitrary unweighted, undirected graph (**with no colors assigned to any vertices!!!)**

(b) Provide an algorithm to use G to create a Boolean formula φ, in 3-CNF form.

(c) Prove, that if the graph G is 3 colorable, then the corresponding Boolean formula, φ that your algorithm generated is satisfiable.

(d) Prove, that if the output formula φ is satisfiable, then then corresponding input G that created it must be 3 colorable.

(e) Prove that your algorithm to calculate φ from G runs in polynomial time of the input, G.

In the natural description of the reduction, quite a few of the clauses one would create would be the or of two variables, not three. Note that any clause of the form $(a \vee b)$ can easily be transformed into an equivalent clause with three terms as follows: $(a \vee b \vee b)$. There is no need to describe this in your reduction. Your algorithm may produce clauses of two or three terms.

Solution

Given an input graph, G, we must "encode" the coloring rules into a 3-SAT formula. Here is how we will do it:

For each vertex v, in G, we create 3 variables v_a, v_b, and v_c. v_a will be set to true iff vertex v is colored a, v_b will be set to true iff vertex v is colored b, and v_c will be set to true if vertex v is colored c.

We must "force" exactly one of v_a, v_b, and v_c to be set to true. We do this by adding the following clauses:

$$(v_a \vee v_b \vee v_c) \wedge (\overline{v_a} \vee \overline{v_b}) \wedge (\overline{v_a} \vee \overline{v_c}) \wedge (\overline{v_b} \vee \overline{v_c})$$

The first of these clauses makes sure that at least one color is assigned to vertex v. The other three clauses make it impossible for any pair of colors to be assigned to vertex v, limiting the number of colors assigned to vertex v to be exactly 1. Thus, we add these 4 clauses for every vertex in our input graph, G.

Next, we must make sure that no two vertices, u and v, connected by an edge in G are assigned the same color. We can ensure this by adding the following clauses:

$$(\overline{u_a} \vee \overline{v_a}) \wedge (\overline{u_b} \vee \overline{v_b}) \wedge (\overline{u_c} \vee \overline{v_c})$$

These three clauses prevent both vertices connecting an edge to be assigned color a, or both to be assigned to color b, or both to be assigned color c.

If the corresponding Boolean formula is satisfiable, then there exists a 3 coloring for the original graph. If there exists a 3 coloring for the original graph, then the Boolean formula described is satisfiable. Let's now prove these claims.

Consider a graph G with a valid 3 coloring. We can use a valid coloring to set each of the variables of the form v_a, v_b, and v_c to true and false. (Exactly 2 will be set to false and 1 to true.) This setting will satisfy the four clauses for each variable that were first described.

Next, since the coloring is valid, no two vertices connected by an edge will be assigned the same color. Thus, for all of the other clauses in groups of three clauses, one of the two vertices will not be color a, color b and color c, thereby satisfying the rest of the clauses.

Now, consider any satisfying assignment to the Boolean formula produced by the reduction. The only way to satisfy the first set of clauses is to set exactly 1 of v_a, v_b, and v_c to true, which will indicate what color to assign to each vertex. This exact truth setting by assumption satisfies all of the other groups of 3 clauses, which ensure that no two vertices colored by an edge are assigned the same color, which will mean that the colors assigned provide a valid coloring.

If G has V vertices and E edges, this reduction creates 4V + 3E clauses, which is linear in the size of the input in G. The computation of each clause takes constant time, thus, the reduction computation time is a polynomial in the input size.

Grading:

2 pts for clearly specifying an undirected, unweighted graph as input for the reduction.

2 pts for the idea of creating 3 variables for each vertex

4 pts for creating a formula that forces exactly 1 color to each vertex

3 pts for creating a formula that prevents the two vertices connecting an edge to be colored the same color

3 pts for the iff proof

1 pt for the run time analysis

3) (2 pts) What type of fruit is used to make a Pumpkin Pie?

Pumpkin, give to all