CS 21 Decidability and Tractability **Winter 2014**

Solution Set 4

Posted: February 19

If you have not yet turned in the Problem Set, you should not consult these solutions.

1. We will reduce 2-COLORABLE to 2-SAT, which we showed to be in P. Given a graph G, our reduction produces the following set of clauses: label the vertices of G with variables $x_1, x_2, \ldots, x_n$; for every edge between vertices labelled x_i and x_j, produce the clauses $(x_i \vee x_j)$ and $(\overline{x_i} \vee \overline{x_j})$.

   Clearly this reduction runs in polynomial time. We now argue that "yes maps to yes." If G is 2-colorable, then pick one of the two colors in a 2-coloring of G and assign the associated variables TRUE. Every one of the clauses is satisfied, because the only way to fail to satisfy a clause is if the two endpoints of some edges were the same color.

   We argue that "no maps to no." Suppose we have a satisfying assignment to the set of clauses produced by the reduction. Then the two variables associated with a given edge must have different truth assignments. Therefore, if we color the vertices of G associated with TRUE variables red, and the other vertices of G green, we will have produced a valid 2-coloring of G, and hence G is 2-colorable.

2. The key here is that H is fixed. Suppose it has k vertices. Given an input G, we can simply exhaustively try all potential subgraph isomorphisms. Specifically, we try every subset of k vertices of G, and for each subset we try every one of the $k!$ possible ways of mapping it to the vertices of H. We can easily check, whether H equals the subgraph induced by a given subset of G's vertices (permuted according to a given mapping) – in fact this can be done in time polynomial in k.

   Thus the overall running time of the algorithm is

   $$\binom{|G|}{|H|} \cdot k! \cdot k^{O(1)}$$

   Note that $\binom{n}{k} \leq n^k$, and $k! \leq k^k$, so we have an overall running time of $|G|^{O(k)}$. Since k is fixed, this is polynomial in the size of the input, G.

3. Following the hint, we will build up a table T which has a TRUE/FALSE entry for each $0 \leq B' \leq B$ telling us whether some multiset of the x_i sum to exactly B'. Specifically, let T be an array with $B+1$ entries. Initialize $T[i]$ to be FALSE for all i. For $B' = 0, 1, 2, \ldots B$ do:

   - if some $x_i = B'$ then $T[B'] = $ TRUE.
   - else if for some x_i we have $T[B' - x_i] = $ TRUE, then set $T[B'] = $ TRUE (otherwise leave $T[B']$ set to FALSE).

At the end, we accept iff $T[B]$ is TRUE.

In each step we need to check all of the x_i, so the running time for one iteration of the loop is $O(n)$. The overall running time is $O(Bn)$ which is polynomial in the size of the input (here we use crucially that B is presented in unary).

We should also argue that this algorithm is correct. We claim that after the B'-th iteration, all entries of T up to and including B' are correct (i.e. they are TRUE if there is a multiset of the x_i summing to that value, and FALSE otherwise). The base case (when $B' = 0$) is clearly satisfied. Now assume T is correct up to and including $B' - 1$. There is a non-empty multiset summing to exactly B' iff for some i, there is a non-empty multiset summing to $B' - x_i$. By induction $T[B' - x_i]$ is correct, and so we correctly fill in $T[B']$ in the B'-th iteration of the main loop.