Theory of Computation, CSCI 438 spring 2022
Algorithms, pages 182-187, April 1

Give an algorithm that "decides" the following. If there is no algorithm that decides the language, give an algorithm which "recognizes" the language. If there is not algorithm that recognizes the language, say that.

1. L_1 = {<P> | P is a polynomial on x with an integer root}

   M= "On input <P> where P is a polynomial over variable x

   ```
   1. Evaluate P with x set successively to the values 0, 1, -1, 2, -2, …
      If at any point this evaluates to 0, accept.
   ```

   This shows that L1 is Turing recognizable, but not decidable (if P doesn't have integral roots it will run forever).

   An upper bound can be determined via a TM.
   M= "On input <P> where P is a polynomial over variable x

   ```
   1. Determine an upper bound for P.
   2. Evaluate P with x set successively to the values 0, 1, -1, 2, -2, … upper
      bound.
      If at any point this evaluates to 0, accept.
   3. reject
   ```

2. L_2 = {<P> | P is a polynomial with integer roots}
   In this case upper bounds for each variable can't be found. For many particular cases, it may be possible to determine upper bounds for the variables, however, it has been proven (Matiyasevich's Theorem concerning Diophantine equations with integer roots, Hilbert's 10th problem) that in the general case, upper bounds for each of the variables cannot be determined.

   Here is a recognizer for L_2.

   ```
   M= "On input <P> where P is a polynomial over varablex x1, … xn
           1. Evaluate P with x1, … xn set successively to the values
                   0,0, …, 0 (n values),
                   0,0, …,1,
                   0,0, …,-1,
                   0,0, …,1,0
                   0,0,…, 1,1
                   0,0, …1,-1
                   …
                   If at any point this evaluates to 0, accept.
           If P doesn't have integral roots this will run forever.
   ```

3. L_3 = {<G> | G is a connected graph}
   In discrete we discussed three ways to represent graphs (digraphs or multigraphs) in computers:
   - list of vertices and a list of edges,
   - adjacency list,
   - adjacency matrix

   Say that G is represented as an adjacency list.

   ```
   M= "On input <G> where G is a graph
     1. Mark the first vertex as reached
     2. While (number of marked vertices is increasing)
          Mark all vertices that are reachable from a reachable vertex as
          reachable
     3. If all vertices are marked as reachable, accept. Otherwise reject.
   ```

4. L_4 = {<G> | G is a tree}
   Recall that a tree is a graph which is connected and has no loops. Assume that G is not a multigraph and edges in G are not directed.

   Trees can be encoded more efficiently than an adjacency list, but I'll use an adjacency list again so that I can use the result of the previous problem.

   ```
   M= "On input <G> where G is a graph
     1. Mark any vertex as the root of the potential tree (I'll use the first
        vertex in the adjacency list)
     2. While (number of marked vertices is increasing and not all vertices are
        marked)
          Mark those vertices connected to a recently marked vertex. If any
          vertex was already marked, reject (graph contains a loop).

     3. If all vertices are marked as reachable, accept. Otherwise reject.
   ```