COT 4210 Quiz #4 Part A: Class P 4/22/2021 Solutions

1) (10 pts) Here is an algorithm to determine if two integers, a and b, represented in binary, are relatively prime or not:

```
for each integer, x, in the range from 2 to min(a, b):
    if both a and b are divisible by x:
        return false
return true
```

Solution

(a) Explain why this algorithm does NOT run in polynomial time in the size of the input.

The input size, when both numbers are represented in binary is $\log_2 a + \log_2 b$, which is simply O(lg max(a,b)). Let n = lg max(a, b). In the worst case, the algorithm above takes min(a, b) steps.

In this worst case, the values of a and b are within a constant factor of each other, so we can effectively, in terms of order notation, for the worst case, treat max(a, b) and min(a, b) as having close to the same value. Thus, with an input size of n, the number of times the loop might run in the worst case is upto 2^n. The latter is not polynomial in the size of the input, n.

Grading: 3 pts calculation input size, 2 pts calculation run time, 2 pts proving that the latter isn't always a polynomial in the former.

(b) Explain why this algorithm is NOT proof that the language RELPRIME, (the set of pairs of positive integers that share no common factors) does NOT belong to the class P.

Just because this algorithm exists doesn't preclude the existence of a faster algorithm. A proof that shows no such algorithm exists would have to be more complicated, showing that no matter what steps one designed, those steps couldn't solve the problem fast enough. (There is a proof for example, that no comparison sort can run in faster than O(nlgn) time but it's just based on the number of possible inputs to the sorting problem and the fact that a comparison based sort can only decide between two inputs in a single comparison.)

Grading: 3 pts, probably mostly all or nothing, but feel free to give partial. The key reason is fairly clear and straightforward.

2) (10 pts) Give an algorithm to prove that the following language is in the class P:

NUMPARTS = { <G, k> | G is an undirected, unweighted graph with precisely k connected components }

You must make sure that (a) your algorithm completely solves the problem, and (b) give its run time in terms of |V| the number of vertices in the graph G and |E| the number of edges in the graph G.

Solution

This is a standard problem covered in COP 3503. Here is one efficient solution:

1. Set a counter to 0, which will keep track of each connected component.
2. Store an array visited, of size |V| the number of vertices in G, initially set to false, indicating which vertices have been marked in some component.
3. For each vertex v in V:
   a. if v is NOT visited
      i. Run DFS(G, v, visited)
      ii. Add 1 to counter.
4. Return true if counter equals k, and false otherwise.

Here is the algorithm for DFS:

1. Mark visited[v] = true.
2. For each neighbor u of v in G:
   a. if u has not been visited
      i. Run DFS(G, u, visited)

The run time of this algorithm, assuming efficient storage of the graph G is O(|V|+|E|) because each edge and vertex is explored a constant number of times. This run time is polynomial in the size of the input, which is also O(|V|+|E|).

Grading: 8 pts for the algorithm - 2 pts for counter of components, 2 pts for marking what's been visited, 3 pts for any reasonable component search, 1 pt for updating the counter and returning a result accordingly.

2 pts for the algorithm analysis - be pretty relaxed here

3) (5 pts) Prove that the following language is in the class P:

TRIANGLE: { <G> | G is an undirected, unweighted graph with a CLIQUE of size 3 or greater. }

Solution

Here is an algorithm that solves the problem in $O(|V|^3)$ time assuming that G is stored as an adjacency matrix and V is the set of vertices of G. If G is stored as an edge list, the run time might be as bad as $O(|V|^3|E|)$, depending on how edges are searched for.

1. for each vertex v in G:
   a. for each vertex u in G not equal to v:
      i. for each vertex w in G not equal to u or v:
         1. if there is an edge between (u, v), (u, w) and (v, w) return true
2. return false

Since a triangle only requires us to check 3 vertices at a time, we can simply try all combinations of 3 vertices and see if any of them contain the three necessary connections. Since this is no more than V^3 and the input size is at least V, it follows that this algorithm is polynomial in the size of the input. Even if the search for the edges always took O(E) time in the inner if, this extended run time is still a polynomial in the size of the input, since the input size must be greater or equal to O(E).

Grading: 4 pts for triple for loop idea, 1 pt for explaining why this is polynomial in the input size.