CS 325 HW7 Hao Deng ID:932912420

1). Let X and Y be two decision problems. Suppose we know that X reduces to Y in polynomial time. Which of the following can we infer? Explain.

A). If Y is NP-complete then so is X.

False, it cannot be inferred because X does not necessarily in NP-Complete and it could be just in NP

B). If X is NP-complete then so is Y.

False, it cannot be inferred because Y could be in any harder NP classes.

C). If Y is NP-complete and X is in NP then X is NP-complete.

False, it cannot be inferred because X can just be in NP.

D). If X is NP-complete and Y is in NP then Y is NP-complete.

True, it can be inferred because Y is in NP and if X is NP-complete then so is Y because Y must be at least as hard as X.

E). If X is in P, then Y is in P.

False, it cannot be inferred because Y is at least as hard as X, so it can still in NP and not necessarily in P.

F). If Y is in P, then X is in P.

True, it can be inferred because knowing that Y is at least as hard as X, so if Y is in P then so does X.

2). Consider the problem COMPOSITE: given an integer *y*, does *y* have any factors other than one and itself? For this exercise, you may assume that COMPOSITE is in NP, and you will be comparing it to the well-known NP-complete problem SUBSET-SUM: given a set S of *n* integers and an integer target *t*, is there a subset of *S* whose sum is exactly *t*? Clearly explain whether or not each of the following statements follows from that fact that COMPOSITE is in NP and SUBSET-SUM is NP-complete:

A). SUBSET-SUM $\leq_p$ COMPOSITE.

No. Since SUBSET-SUM is NP-complete, it may only be reduced to any other NP-complete problem. We only know that COMPOSITE is in NP but we don't know if it's in NP-complete.

B). If there is an $O(n^3)$ algorithm for SUBSET-SUM, then there is a polynomial time algorithm for COMPOSITE.

Yes. If SUBSET-SUM has an polynomial time, hence we can prove that P=NP, which means that if NP-complete can be solved in polynomial time then every problem in NP, including NP-complete like COMPOSITE, can be solved in polynomial time.

C). If there is a polynomial algorithm for COMPOSITE, then P = NP.

No. This is not true because COMPOSITE is only in NP but we don't know if it is in NP-complete.

D). If P not equals to NP, then ***no*** problem in NP can be solved in polynomial time.

No. Since P is the subset of NP, hence all the problems in P are also in NP and they can be solved in polynomial time.

3). A Hamiltonian path in a graph is a simple path that visits every vertex exactly once. Prove that HAM-PATH = { (G, u, v): there is a Hamiltonian path from u to v in G } is NP-complete. You may use the fact that HAM-CYCLE is NP-complete.

To prove that HAM-PATH is NP-complete, we need to show that 1). HAM-PATH is in NP and 2). C can reduce to HAM-PATH for some C in NP-complete.

1). Showing that HAM-PATH is in NP

Given a graph G with n vertices with a path from u to v, we can verify the certificate in polynomial time that the path is a simple path with n vertices, by checking the adjacency list or matrix to verify the vertices are adjacent, and that there are n vertices.

2). Show that C can reduce to HAM-PATH for some C in NP-complete

- We can first choose C to be HAM-CYCLE because the structure between these two is similar and since we know that HAM-CYCLE is NP-complete, therefore it is in NP.
- Let's name HC = HAM-CYCLE and HP = HAM-PATH. HC (u-v) reduces to HP (u-u'). Given a graph G(u-v) with having a Hamiltonian Cycle, where (u-v) is a set of vertices, we can create a new graph G'(u-u') where we duplicate the vertex u with all it's connecting edges and name it u'. This new graph, G'(u-u') now has a Hamiltonian Path from u to u'. This reduction can happen in polynomial time by transformation: adding the list of edges for u' to the edge list of G.
    - For graph G(u-v) to contain a Hamiltonian Cycle, all the vertices connect to each other, can transform to graph G'(u-u') that contains Hamiltonian Path for all vertices connect to each other include u' since its edges are duplicated from u.
    - If graph G(u-v) has some missing edges like from u to y and y to u (let's say that the G(u-v) has vertices of u,v,x,y) in which that the Hamiltonian Cycle will not exist, through transformation, graph G'(u-u') will have missing edges just like G(u-v) but with the additional missing edges from u' (again, it's a duplicate of u from G(u-v)) in which that the Hamiltonian Path will not exist in G'(u-u').
- If G has a Hamiltonian Cycle, then G' has a Hamiltonian Path, and if G doesn't have Hamiltonian Cycle then so does G'.

Since both step 1 and 2 are true, in this case we can confirm that HAM-PATH is NP-complete.

4). K-COLOR. Given a graph G = (V,E), a k-coloring is a function c: V -> {1, 2, … , k} such that c(u) ≠ c(v) for every edge (u,v) ∈ E. In other words the number 1, 2, .., k represent the k colors and adjacent vertices must have different colors. The decision problem K-COLOR asks if a graph can be colored with at most K colors.

A). The 2-COLOR decision problem is in P. Describe an efficient algorithm to determine if a graph has a 2-coloring. What is the running time of your algorithm?

This is like the bipartite problem that we've done in couple assignments ago. We can use BFS or DFS to check whether a given graph is 2 colored or not.

- We can start with BFS on an arbitrary vertex
- The coloring on the vertices at the first level would be color A, then second level with color B, then third with color A, and so on.
- For each edge, check whether two ends have different color or not.
- If any edge has the same colorings, return false because the graph has no 2-coloring, else true.

```
The BFS runs from vertex a
Mark a as visited and insert a into queue.

C=1
Level[a] = 0
Color[a] = 0
While queue is not empty:
        U = queue.remove()
        C=(c%2)+1
        For all of the unvisited neighbors v of u:
        Mark v as visited
        Color[v] = c
        Level[v] = level[u]+1
        Queue.insert(v)

        For each vertex u of G:
                For each vertex v in adjacent[u]
                        If Color[u] == Color[v]
                                Return false
Return true
```

- BFS takes O(V+E) time and traversing adjacency list takes O(V+E) time
- The algorithm is O(V+E) and it's linear

B). It is known that the 3-COLOR decision problem is NP-complete by using a reduction from SAT. Use the fact that 3-COLOR is NP-complete to prove that 4-COLOR is NP-complete.

It is obvious that 4-COLOR is NP because since we know that 3-COLOR is NP-complete and 4-COLOR derived from the structure of 3-COLOR except we add an extra node that is connected with the base graph.

Proving 4-COLOR is NP-Hard (reducing 4-COLOR from 3-COLOR)
It is already given that 3-COLOR is NP-complete, hence 3-COLOR is also NP.
To prove 4-COLOR, we need to reduce this NP problem, 3-COLOR, to 4-COLOR.

Take a graph G and reduce it to a new graph G' such that if G belongs to 3-COLOR if and only if G' belongs to 4-COLOR.

Generate G' by adding a new node v to G and connecting v to all the nodes in the graph. For which graph G can be obtained if node v is removed from G'.

- Consider G belongs to 3-COLOR
- G' is obviously belongs to 4-COLOR with the added new node v with new color, since G only has 3 colors and only node v has the fourth color.

Then if G belongs to 3-COLOR then G' belongs to 4-COLOR.

- Consider G' belongs to 4-COLOR
- G is obviously 3-colorable since only G' has the fourth color with node v which makes G' 4-colorable.

Thus, if G' belongs to 4-COLOR then G belongs to 3-COLOR.

Therefore, it is proved that if G belongs to 3-COLOR if and only if G' belongs to 3-COLOR.

This also proved that 4-COLOR is NP-complete, since 4-COLOR is NP and NP-hard.