Brendan Jang
CS 325
Homework 7

1. a. If Y is NP-complete then so is X.
   This statement is FALSE. For NP-complete, X has to be the subset NP-complete.

   b. If X is NP-complete then so is Y.
   This statement is FALSE. The problem Y can't be NP-complete because X is reducible to Y and not the other way around.

   c. If Y is NP-complete and Y is in NP then Y is NP-complete.
   This statement is FALSE. X is reducible to Y so that means X can be solved efficiently by Y but not vice-versa for the given relation of X and Y.

   d. If X is NP-complete and Y is in NP then Y is NP-complete.
   This statement is TRUE. The problem X reduces to Y so that if there is a black box to solve Y efficiently, then it can be used to solve X efficiently as well.

   e. If X is in P, then Y is in P.
   This statement is FALSE. Problem X is reducible to Y so if Y is in P then X is in P as well.

   f. If Y is in P, then X is in P.
   This statement is TRUE. The problem X reduces to problem Y so that if there is a black box to solve Y efficiently, then it can be used to solve X efficiently as well.

   Based on the explanation shown above, we can infer options d and f.

2. a. This statement DOES NOT FOLLOW given that SUBSET-SUM problem is NP-complete. As a result, it can be reduced to another NP-complete problem. Also, the COMPOSITE problem is in NP and it is not definite that the COMPOSIITE problem is also in NP-complete. Therefore, we can conclude that it is not definite that a SUBSET-SUM problem reduces to a COMPOSITE problem.

   b. This statement FOLLOWS given that the O(n^3) algorithm is a polynomial in n. Also, given that SUBSET-SUM is NP-complete, it gives that P = NP. The problem in NP will have polynomial time algorithm. Therefore, the polynomial time algorithm exists for the COMPOSITE problem.

   c. This statement DOES NOT FOLLOW given that the COMPOSITE problem is in NP and it is not definite that the COMPOSITE problem is in NP-complete as well. Therefore, we can conclude that the argument P = NP never follows if the COMPOSITE problem has polynomial algorithm.

   d. The given statement DOES NOT FOLLOW because we know that P is the subset of NP. Also, P is not empty. If we attempt to prove P is not equal to NP, then we will get problems in NP-

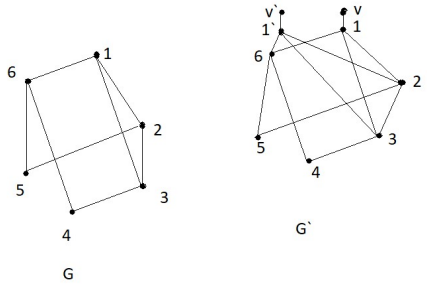
complete that can’t be solved in polynomial time. However, problems in NP-complete can have polynomial time algorithm.

3. In order to show that this problem is NP-complete, first we have to show that it belongs in NP and then find a known NP-complete problem that can be reduced to Hamiltonian path.

   For a given graph G, we can solve Hamiltonian path by choosing edges from G that are to be included in the path. We then transverse the path and confirm that we visit each vertex only once. This can be done in polynomial time and therefore belongs to NP.

   Now we find a NP-complete problem that can be reduced to a Hamiltonian Path. A closely related problem can be a problem to find whether a graph contains a Hamiltonian cycle which is basically a Hamiltonian path that starts and ends at the same vertex. We also know that Hamiltonian cycle is NP-complete so we can try to reduce this problem to Hamiltonian path problem.

   Given a graph G = <V, E> we make a graph G' such that G contains a Hamiltonian cycle if and only if G' contains a Hamiltonian path. This is done by choosing an arbitrary vertex u in G and adding a copy it, u', together with all its edges. Then, add vertices v and v' to the graph and connect v with u and v' with u'. See example graphs for reference.



   Suppose that G contains a Hamiltonian cycle. We get a Hamiltonian path in G' if we start in v, follow the cycle that we got from G back to u' instead of u and finally end in v'. For example, consider the left graph, G, in Figure 1 which contains the Hamiltonian cycle 1, 2, 5, 6, 4, 3, 1. In G' this corresponds to the path v, 1, 2, 5, 6, 4, 3, 1', v'. Conversely, suppose G' contains a Hamiltonian path. In that case, the path must necessarily have endpoints in v and v'. This path can be transformed to a cycle in G. If we disregard v and v', the path must have endpoints in u and u' and if we remove u' we get a cycle in G if we close the path back to u instead of u'.

   This will not work when G is a single edge. A single edge case must be taken care of as a special case. Hence, we can see that G contains a Hamiltonian cycle if and only if G' contains a Hamiltonian path, which concludes that Hamiltonian Path is NP-complete.

4. a. Graphs that are 2-colored are bipartite graphs. In order to check if a given graph is bipartite, we can run BFS starting from any node and then check if there is any edge between two nodes in the same BFS levels. The graph is bipartite if there is no such edge.

   - Start BFS on an arbitrarily selected vertex
   - And color the vertices at level 0 with color 1, color the vertices at level 1 with color 2, color the vertices at level 2 with color 1, and so on.
   - Now for each edge, check whether the two ends have different color or not.
   - If any edge have same colors on both the ends, return false(graph has no 2-coloring). Otherwise return true(graph has 2-coloring).

```
Algorithm twoColors(G):
// run bfs from vertex s
mark s as visited and insert s into queue
c= 1
levels[s] = 0
corlor[s] = c
while queue != empty
        u = queue.remove()
        c = (c % 2) + 1
        for all unvisited neighbors v of u
                mark v as visited
                color[v] = c
                level[v] = level[u] + 1
                queue.insert(v)
// go through the adjacency list of G
for each vertex u of G
        for each vertex v in adj[u]
                if color[u] == color[v]
                        return false
return true
```

   There are two parts in the above algorithm. The first part (BFS) colors the vertices with two colors such that the level i is colored with one color and level i + 1 with another color. We know that BFS takes O(V + E) time. It is linear in terms of input V and E. The second part (traversing adjacency list) checks whether any edge have same colors on both sides. We know that, traversing the adjacency list also takes O(V + E) time. Therefore, the running time of the algorithm above is O(V + E) and it is linear.

   b. In order to prove that 4-COLOR is NP-complete, we must prove that a language, L, is NP and we need to reduce L to L1 that belongs to NP. If L is reducible to L1, then L1 is said to be NP-hard.

First, 4-COLOR is clearly NP because it can be checked whether a graph is 4-color or not in polynomial time by doing the following. For each edge of the given graph G, check whether the colors of its both ends are the same or not. Then if no edge has the same color on both ends, then G belongs to 4-COLOR. Else, G does not belong to 4-COLOR. Therefore, we see that 4-COLOR is NP.

Next, we must prove that 4-COLOR is NP-hard. It is already given that 3-COLOR is NP-complete so 3-COLOR is also NP. Then, we reduce this NP problem (3-COLOR) to 4-COLOR in order to prove 4-COLOR. We take a graph, G, and reduce it to a new graph G1 such that if G belongs to 3-COLOR if and only if G1 belongs to 4-COLOR.

Generate G1 by adding a new node v to G and connecting v to all nodes in the graph. That is graph G can be obtained if node v is removed from the graph G1. Consider G belongs to 3-COLOR. Now color the new node v with a new color. G1 belongs to 4-COLOR since G is 3-colorable and only the node v has fourth color. Therefore, if G belongs to 3-COLOR, then G1 belongs to 4-COLOR. Consider G1 belongs to 4-COLOR. Since it is known that G1 is 4-Colorable and no other node has the color that node v has, then G is 3-colorable. Therefore, if G1 belongs to 4-COLOR, then G belongs to 3-COLOR.

We have proved that if G belongs to 3-COLOR if and only if G1 belongs to 4-COLOR. It is also proved that 4-COLOR is NP-Complete, since 4-COLOR is NP and NP-hard.