**R-17.3**

An NPC problem is both a NP-hard problem and a NP problem. To prove NP-hard, we need to prove that we can reduce the problem to an NP-hard problem in Polynomial time.

According to textbook page 490, we know that CNF-SAT is SAT’s special case and belongs to NP-complete. Since we can reduce SAT to CNF-SAT, SAT is NP-hard. SAT can take an arbitrary Boolean formula S as input, so we can use them to test if S is satisfied in polynomial time, which makes SAT a NP problem. Thus, by definition, SAT is NP-complete.

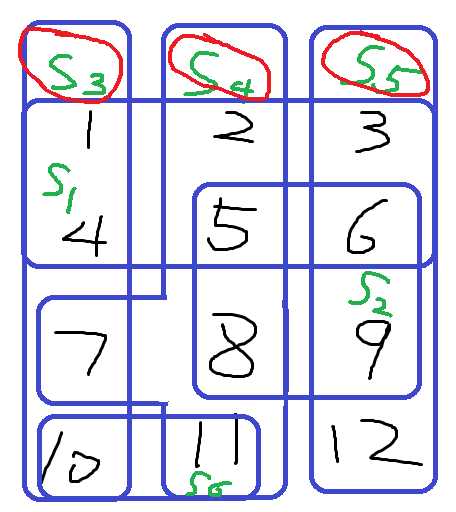
**R-17.7**

We can prove NP by giving a set of problems and finish testing them in polynomial time. In this case we can use a set of vertices with size k and use CLIQUE to test whether there is an edge between every pair of vertices and makes them adjacent to each other. This test could be finished in polynomial time and thus, making CLIQUE a NP problem.

**C-17.10**

We need to prove both INDEPENDENT-SET is NP problem and NP-hard problem. We cab use a subset of G with vertices size k and scan once to find if it’s all independent vertices, this should run in polynomial time, and makes this problem a NP problem. We can find whether there is a vertex cover of size k in G by checking if there is an independent set of size (n-k) where n is the vertices in G and k is the vertices in vertex cover. Now we have reduced the VERTEX-COVER problem to INDEPENDENT-SET problem, and thus, prove that INDEPENDENT-SET problem is NP-complete.

**A-17.5**

We can reduce SET-COVER problem to this one. Say N is the set of all infected computers, and Nw1 is the set of all infected computers that visited website w1. Then we need to find the smallest number of websites visited by all N computers, or, in other words, finding the smallest number of sets that cover the global set of infected computers N. Since SET-COVER is NP-complete, then this problem is NP-complete.

**R-18.11**

The Venn diagram is to the right. The solution is {S3,S4,S5}. With greedy approach, we grab whichever set have the most elements in the remaining set: Remain={1,…12}, Choose={S1}. => R={7,…12}, C={S1,S4}. => R={9,10,12}, C={S1,S4,S5}. => R={10}, C={S1,S4,S5,S3}. The greedy approach gives {S1,S4,S5,S3}.

**C-18.1**

We can reduce HAMILTONIAN-CYCLE to this problem by defining a cost function for a complete graph H for the n-vertex input graph G so that edges of H also in G have cost 1 but edges of H not in G have cost δn more than 1, which means if we want to include edges of H in G, it would cost at most n, but for edges in H not in G, it would cost (n-1)+(1+ δn) = n(1+ δ). Thus, to construct a polynomial-time δ-approximation of the TSP, we need to include only edges of H in G for the cycle.

**A-18.3**

If we are using a greedy algorithm to always assign the new box to the least loaded truck, then for a two truck two box example, by greedy approach, we get Truck1: 1 box, Truck2: 1 box. Before applying greedy approach, we get Truck1: 2 boxes, Truck2: 0 box. Thus, the greedy algorithm has an approximation ration of at most 2 for minimizing the weight of the most heavily loaded truck.