

**Solution:** To prove that SAT is **NP**, we must construct a nondeterministic algorithm that guesses an assignment for the Boolean values for the variables that exists in S and further by evaluating each of single clauses of S. If all the clauses of S turn to 1, then we can say that S is satisfied, or else it is not.

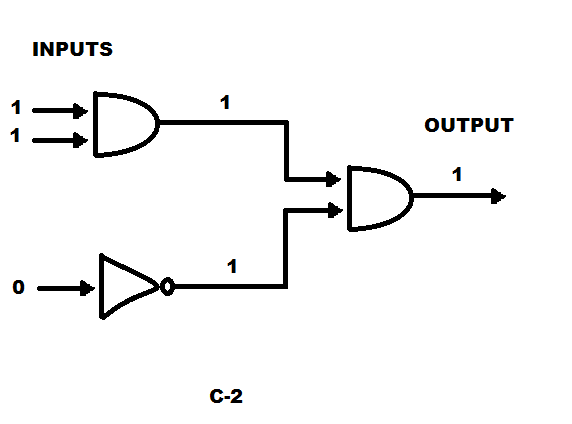
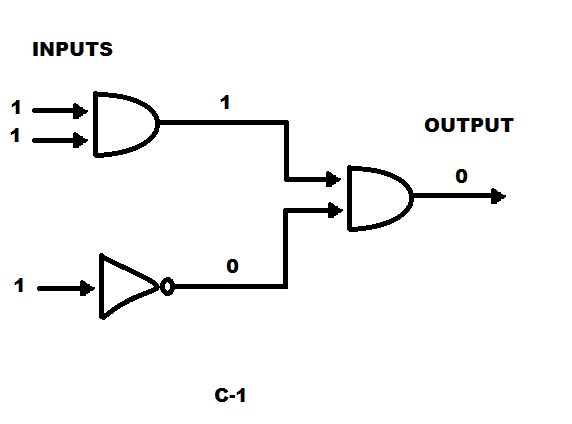
Hence, we can see that SAT is in **NP**.

To prove that Sat is **NP-hard**, we will reduce the Circuit SAT problem into polynomial time. Let us assume that we are given a Boolean Circuit B which contains a combination of the logic gates AND, OR and NOT. At each step we create a tempanswer which gives us output of each input the whole circuit B. This gives us the answer in polynomial time.

In such a Boolean Circuit, there can be only two possible cases: -

1st Case: If any check for a gate fails which means that it produces the output “0”. Then we throw the output as “No”.

2nd Case: If the checks for the gates succeed which means it produces the output “1”, then the verification algorithm outputs as “Yes”.



For example, C-1’s output is 0. Hence, the circuit is not satisfiable, whereas in C-2 the output we get is 1. Thus, the circuit is Satisfiable.

Hence, we have proved that the problem is **NP-hard**.

Since, the problem is **NP** and **NP-hard** we can say that the problem SAT is **NP**-complete.



**Solution:** Let G be a graph which has “n” number of vertices.

A clique in a graph G would be a subset S of vertices such that, for each vertex v and w in S, where v is not equal to v,(v,w) is an edge and there exists an edge between every pair of distinct vertices in S.

The CLIQUE problem takes G and an integer n as the input and tells us whether there exists a clique of size n in G.

If we have to prove that a problem is in NP. We have to satisfy these steps.

* Suppose B is a verification algorithm.
* Construct a nondeterministic Algorithm A that takes x and call the choose method to assign the value of each bit in y.
* Run B on y
* If B runs in polynomial-time, so does this non-deterministic algorithm

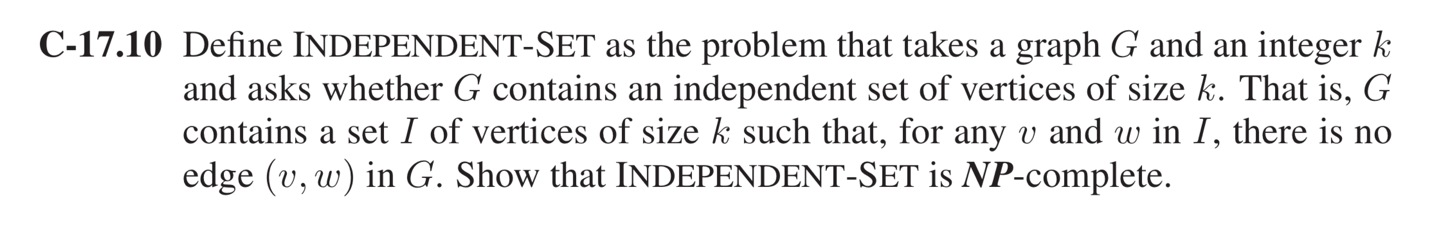
Let y be a certificate as “n” number of vertices then we can figure out if a graph has k cliques in polynomial time, which means that if there exists an edge between two vertices.

Let us create the verification algorithm that can say a “yes” in polynomial time, after that then we give it the certificate y.

Let’s suppose for a graph G we have to find the maximum clique but there exists a lot of vertices, there is no verification algorithm, or a certificate exists that can say a yes, the complexity increases exponentially. However, we aware that there happens to be n number of vertices in G, this problem would not take place.

Thus, we can verify in polynomial time they are able form a complete graph.

Therefore, CLIQUE problem is in **NP**.



**Solution:**

We can simply prove that INDEPENDENT-SET is in NP. We just must guess that a random independent set of size n and check for that set, in polynomial time.

Hence INDEPENDENT SET is in **NP.**

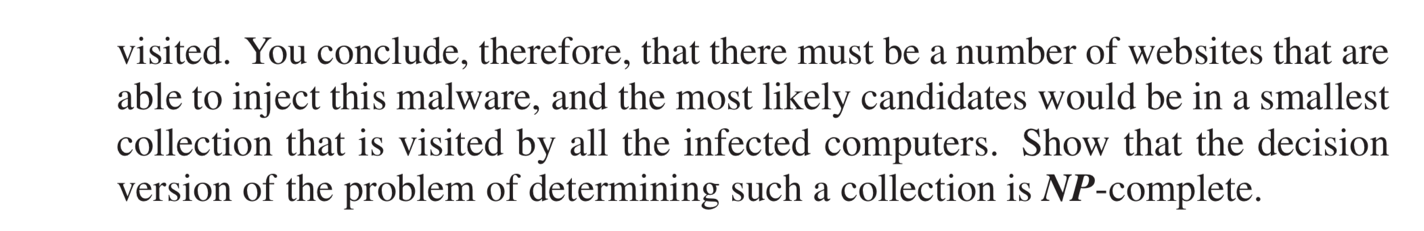
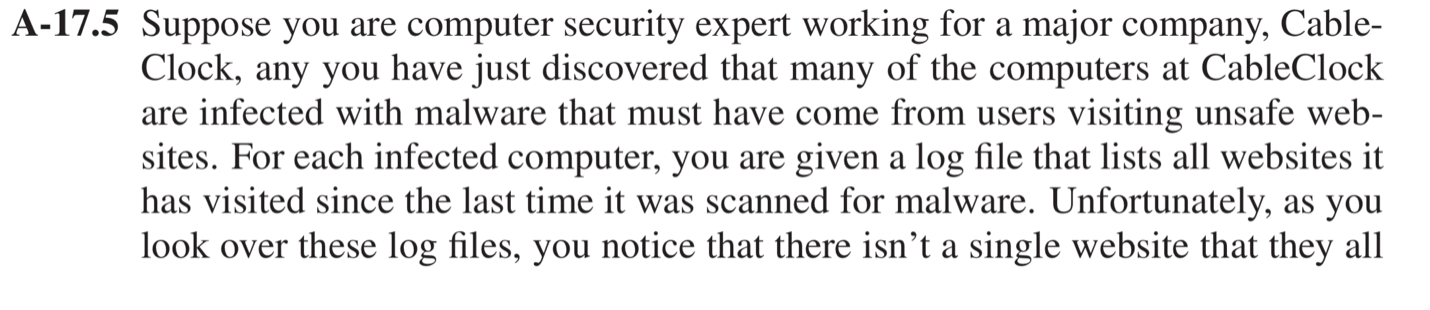
To prove that independent set in NP hard, we reduce it from Vertex Cover. Vertex Cover is a graph G with positive integer n. We say that it accepts when there exists a set of n vertices such that every edge is incident to a minimum of one of the vertices in the set. Basically, an independent set is a set where there are no edges between them.

For example, if A is a vertex cover of the graph G, then V-A is an independent set because there is a minimum of one of the vertex of every edge that’s present in A.

Thus, reducing would be able to G and n vertices, and ultimately produce the same graph G with the set a-n, which is in polynomial time wherein there exists a Vector cover of size n if there one is an independent set of size a-n.

Hence INDEPENDENT SET is also **NP-hard.**

Thus INDEPENDENT-SET is also **NP-complete**.



**Solution:** One way to get a solution is by using SET-COVER approach.

SET-COVER takes a collection of *m* sets *S*1, *S*2, *. . .*, *Sm* and an integer parameter *k* as input and asks whether there is a sub collection of *k* sets *Si*1 , *Si*2 ,*. . .*, *Sik* ,

Consider m set as in m set of computers and k input as the number of sets that needs to be checked for malware.

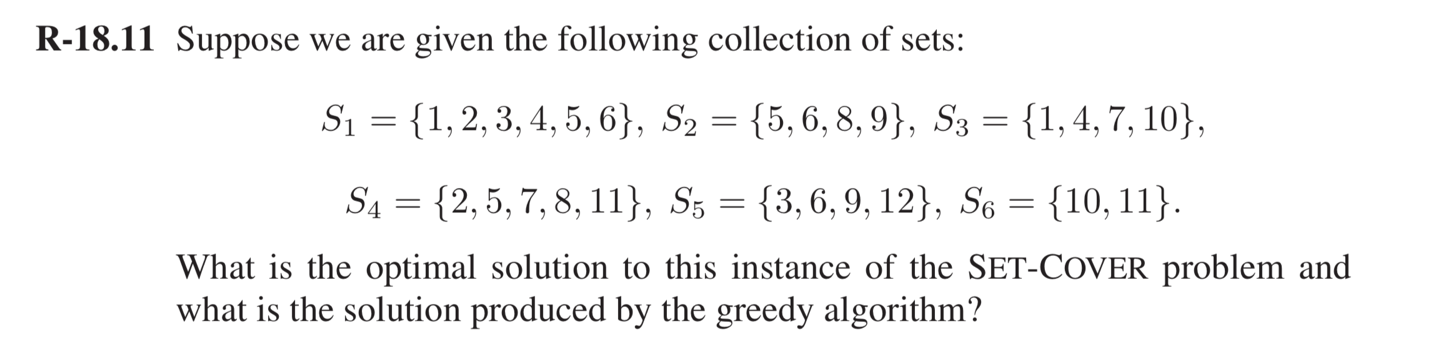
Thus, what we are trying to find is the smallest collection that’s visited by each and every m computers.

We are aware that SET-COVER is in **NP**.

Consider a reduction we can do that by defining an instance of SET-COVER from an instance G and k of vertex cover wherein for each vertex v inside Graph G, there exists a set Sv that has edges of G that is incident on V.

We can note that there exists a SET-COVER in these sets Sv of size k if there exists a vertex cover of size k in graph G. We can say that SET-COVER is NP-complete.

Thus, we have proved that the stated problem is **NP-complete**.



**Solution:** This particular algorithm selects one element at a time form the set, after every selection the set has the most uncovered elements, wherein every element in U is covered, we are finished

Consider the following Greedy Algorithm: -

**Algorithm GREEDSET(Set):**

**Input:** A Set S that has subsets S1,S2,S3,….Sn and its set union is U.

**Output:** SET-COVER V for Set S.

V 🡸 ∅ (Null set)

N 🡸 ∅ (Null set) [elements from *U* currently covered by *V*]

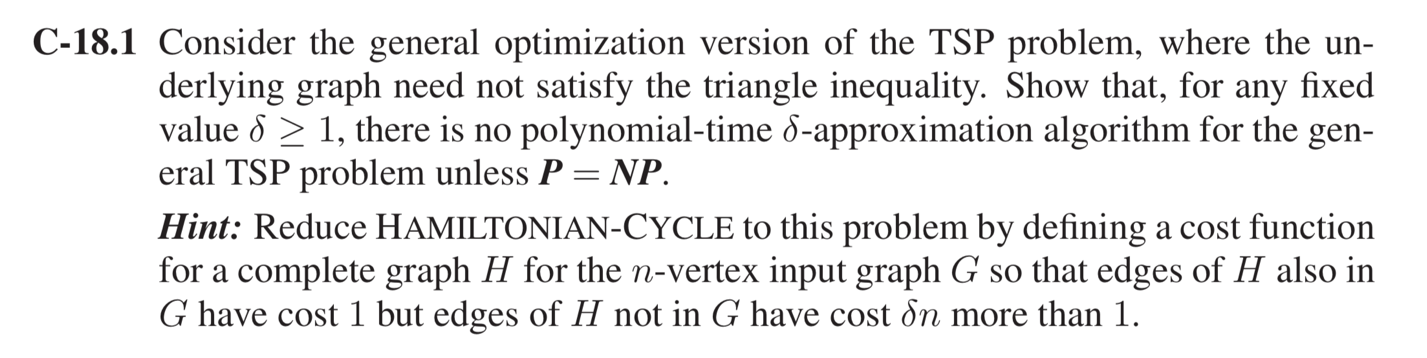
**While** (N is not equal to U) **do**

Check if there exists a set Sx wherein the numbers of undiscovered elements are maximum

Then add the set Sx to V.

N 🡸 N U Sx

**Return N**



**Solution:** We take the help of the Travelling Salesman Problem (TSP).

We know that TSP is **NP-hard**.

Therefore, to show that there is no polynomial for δ-approximation algorithm for the general TSP problem unless **P=NP**. We will approach the problem by solving the NP-complete Hamiltonian Cycle problem in polynomial time which is impossible unless P = NP

Let us assume that there is an Approximation algorithm **AP** which has δ factor as an integer. This can be solved using AP on Hamiltonian-Cycle problem.

We know that Hamiltonian-Cycle is a NP-complete problem only if P = NP. Hamiltonian-Cycle is the problem in which we take a Graph say G and asks if there exists a cycle in G that visits each Vertex V in G exactly once.

Consider a Hamilton Cycle problem(V,E) which checks if a graph has Hamiltonian cycle.

Assume K=(V,E’) to be the complete graph on V.

Now we have two graphs, a Complete Graph and a Hamiltonian Cycle Graph for a set of vertices V.

Declaring an integer to every edges, we get

**If** (u,v) belongs to E **do**

P(U,V) = 1

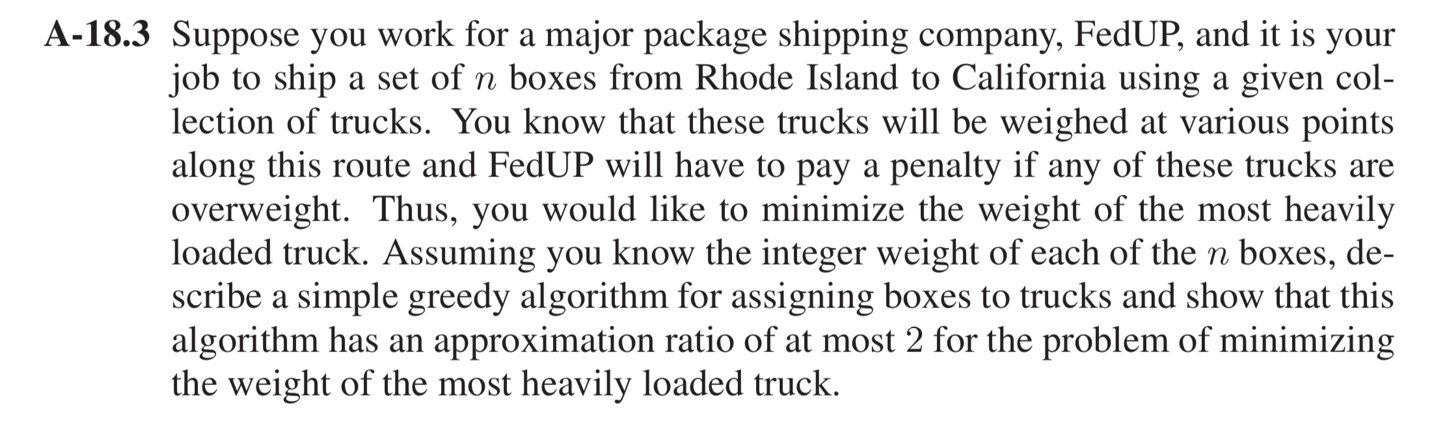
**Else do**

δn + 1

Let us assume that G(V,E) contains a Hamiltonian cycle G’. So each of the edges E would have a value 1 which we can extract from the function C. Hence, (K,T) has a cycle with a cost matrix of “V”.

If we had considered that there are no Hamiltonian Cycle for graph G then tours in K would have edges that do not exist in H. For this scenario the cost would be higher than δn + 1

As the question hints us that we can use A to solve HAMILTONIAN-CYCLE with polynomial cost. Therefore, for any fixed value δ ≥1, there is no polynomial time δ-approximation algorithm for general TSP problem unless **P= NP**.



**Solution**: Let us make an assumption that there are “n” number of items to be unloaded and their respective weights are w1,w2,w3,…. ,wn. Then let N\* be the optimal number of trucks that are required. We know that each of the single trucks would not be able to carry more than U units of load. Thus, we get the following formula.

i ≤ KN\* where N\* ≥ 1/K I 🡺①

Let us assume that “N” be the number of trucks that the greedy algorithm finds. We prove that it is within a factor two of the minimum possible number, for any set of weights and any value of K.

Let us denote Ij as the set of items that truck “j” loads and let Wj be the total weight of the items that are present in Ij, so the equation would be

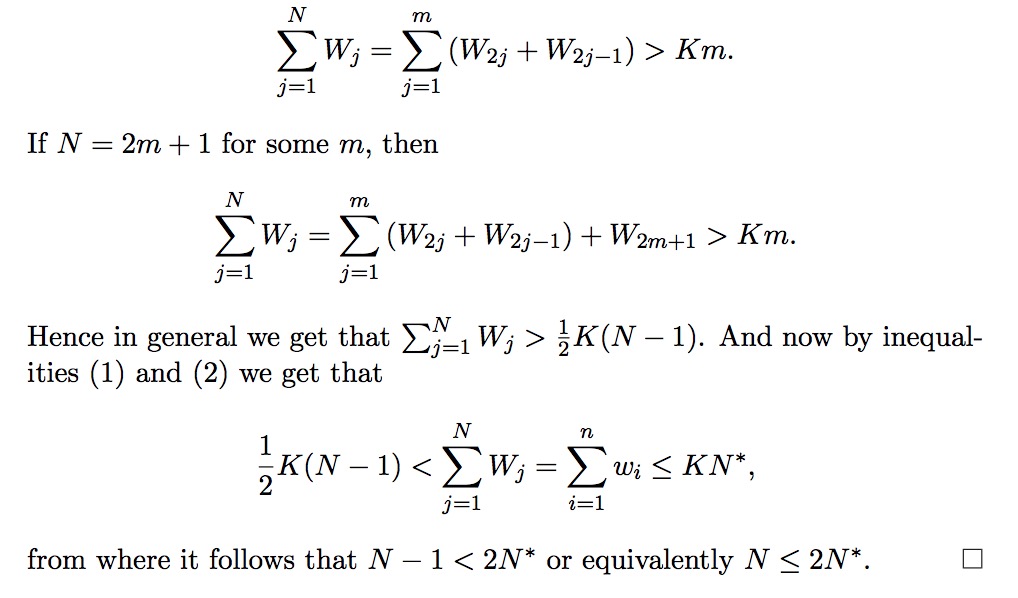
Wj := .

From the equation we can conclude that the algorithm would be true any j > 1.

Wj + Wj-1 > K

We also have the following equation

Since this is a 2-Approximation problem, consider N = 2m, Then, we get



Thus, we have proved that the given algorithm has an approximation ratio for at most 2 for the problem of minimizing the weight of the heavily loaded truck.