Analysis of Algorithms Homework 5

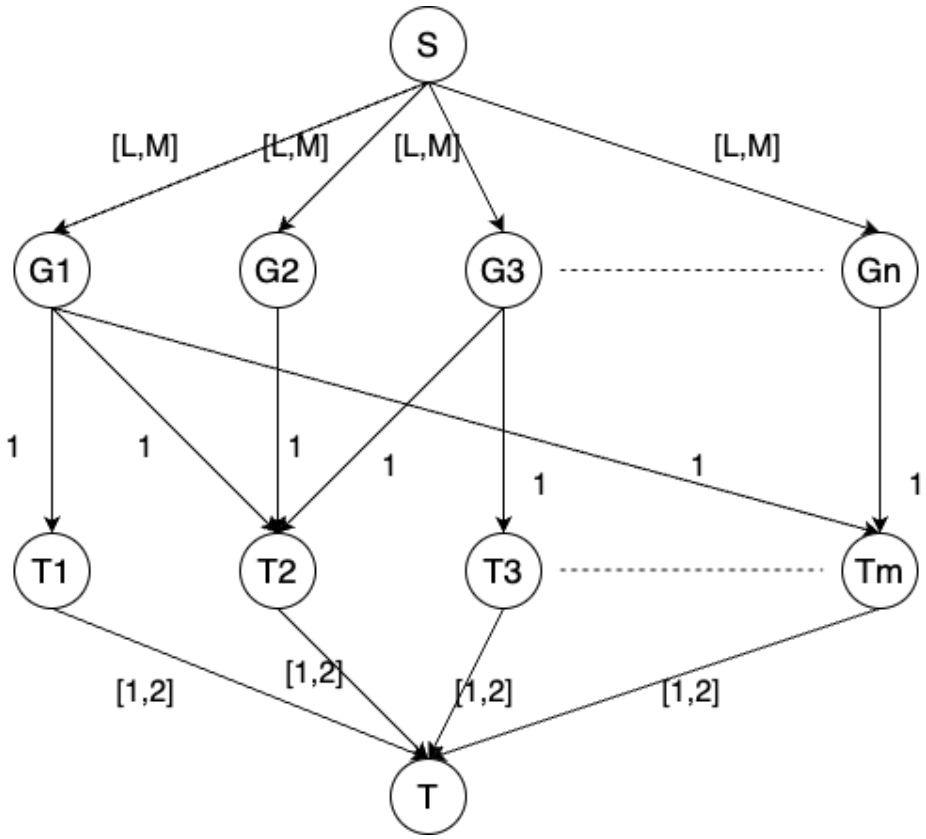
USC ID: 3725520208
Name: Anne Sai Venkata Naga Saketh
Email: annes@usc.edu

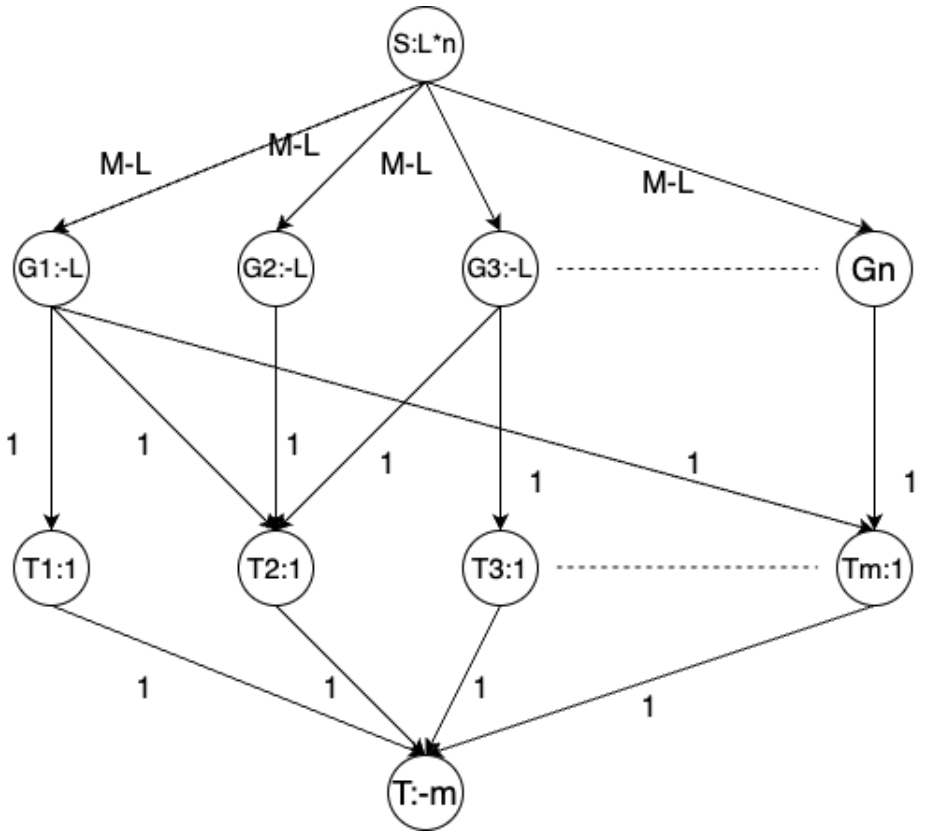
1. **There is a precious diamond that is on display in a museum at m disjoint time intervals. There are n security guards who can be deployed to protect the precious diamond. Each guard has a list of intervals for which he or she is available to be deployed. Each guard can be deployed to at most M time slots and has to be deployed to at least L time slots. Design an algorithm that decides if there is a deployment of guards to intervals such that each interval has either one or two guards deployed.**

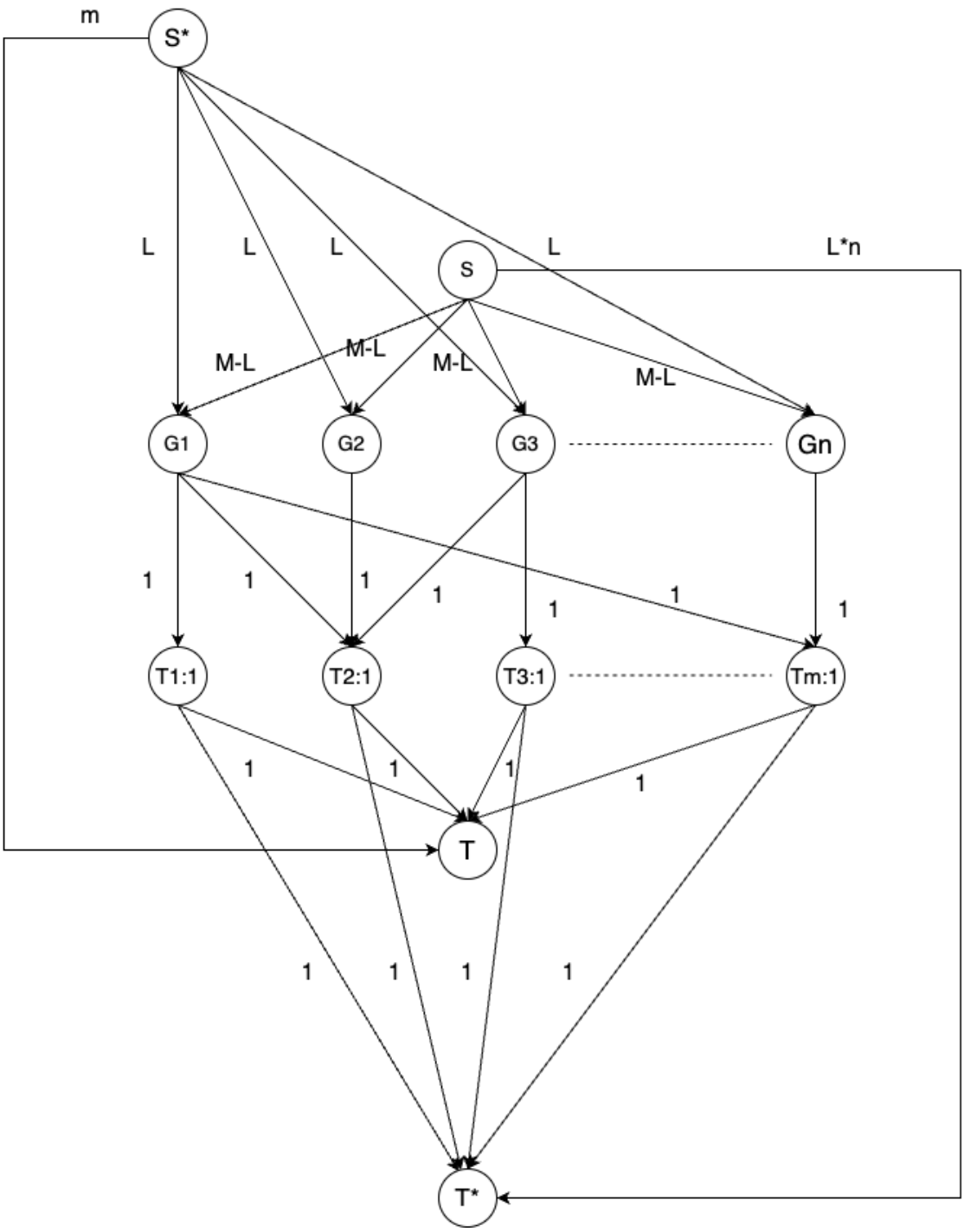
   Solution:

   1. **Construction of the Network Flow graph:**

      - Let us assume that G1, G2, G3 …. Gn is the list of Security Guards that needs to be deployed at different time intervals.
      - It is given that every guard needs to be deployed at least 'L' times and at most 'M' times. So, the capacity from the source to the Security guards' nodes is [L, M].
      - Let T1,T2, T3 …… Tm be the different time intervals that the security guards need to be deployed.
      - It is given that every time interval should have at least 1 and at most 2 security guards deployed to guard the diamond.
      - So, the capacity of the edges from the Time interval nodes to the target node is [1,2].
      - Each Security guard has a subset of time intervals that are feasible for him for the deployment.
      - At any given moment of time, each security guard can be assigned to only 1-time interval, so the capacity of the edge weight from Security guards to the Time interval nodes is 1.
      - A security guard can have more than one choice for a given time interval but can be only assigned to one at a given point in time.
      - The below network graph needs to be reduced to remove the lower bounds, which create the demand and supply on the respective nodes.
      - Then the resulting network flow graph shall be reduced to remove the demand and supply on the nodes, by creating a super source and a super target.


S
[L,M]
[L,M]
[L,M]
[L,M]
G1
G2
G3
Gn
1
1
1
1
1
1
1
T1
T2
T3
Tm
[1,2]
[1,2]
[1,2]
[1,2]
T


S:L*n
M-L
M-L
M-L
M-L
G1:-L
G2:-L
G3:-L
Gn
1
1
1
1
1
1
1
T1:1
T2:1
T3:1
Tm:1
1
1
1
1
T:-m



2. **<u>Claim for the max-flow:</u>**

   The max-flow of the given network flow graph is $\boldsymbol{m + nL}$ such that the assignment such that every guard is deployed at least **L** and at most **M** times, and every interval has at least **1** guard and at most **2** guards is possible.

3. **Proof for the Max-Flow:**
    1. **Proof in Forward direction (=>):**

        Here we need to assume that we are given an assignment of guards to the respective time intervals following all the constraints, and we need to prove the max flow of the given network flow graph.

        We have constructed the above network flow in such a way that every guard is assigned at least 'L' times and at most 'M' times and also such that every time interval has at least 1 guard and at most 2 guards. The demand on the Guards nodes is -L, and the demand on the Time interval nodes is '1'.

        Since all the guards are assigned and also all the time intervals have the guards assigned, the max flow of the above-reduced network flow graph shall be $\boldsymbol{m + nL}$.

    2. **Proof in the backward direction (<=):**

        Here we assume that the max-flow in the above graph is $\boldsymbol{m + nL}$., and we need to prove that we can get an assignment for the guards to the time intervals such that each guard is assigned at least L and at most M times is possible. As well, every time interval has at least 1 guard and at most 2 guards assigned.

        The maximum flow coming to the Sink/T vertex shall be $\boldsymbol{m + nL}$. and which is received from the Time interval nodes and the S node(previous Source node before reduction).

        So, it means that all the 'n' guards are assigned to any one of the time intervals of their choice and every guard is assigned at least 'L' times and at most 'M' times also such that every time interval has at least 1 guard and at most of 2 guards.

        Hence, we have proved our claim in both directions, which means that our claim and the network flow graph align with each other.

Algorithm to find if an assignment of Security guards to the time intervals will be possible

Step 1: Construct the network flow graph, with the lower and upper bounds.

Step 2: Reduce the network flow graph to a graph with demand and supply.

Step 3: Then Reduce the resulting network flow to remove the supply and demand from the nodes by adding an extra source(super source) and extra sink node(super sink).

Step 4: Use Edmond Karp's algorithm to find the max flow in the network flow graph.

Polynomial time algorithm to find the max-flow in the above network graph:

Here, we need to use Edmond Karp's algorithm to find the residual graph in the network flow to find the max-flow value.

The above network graph that we have constructed has n+m+4 nodes, i.e., 'n' guards, 'm' time intervals, a source node, a super source node, a sink node, and a super sink node.

Edmond Karp Algorithm Given(G, S, T, C):

Step 1: Start with |f| = 0, so f(**e**)=0

Step 2: Find the shortest augmenting path in G_f.

Step 3: Augment the flow along this path.

Step 4: Repeat the following steps until there is no remaining S-T path in the graph G_f.

Now, we can find the maximum flow in the given residual graph by summing up the capacities of all the outward edges from the Sink/T in the residual graph.

2. **A company makes three products and has 4 available manufacturing plants. The production time (in minutes) per unit produced varies from plant to plant as shown below:**

		Manufacturing Plant			
		1	2	3	4
Product	1	5	7	4	10
	2	6	12	8	15
	3	13	14	9	17

Similarly, the profit ($) contribution per unit varies from plant to plant as below:

		Manufacturing Plant			
		1	2	3	4
Product	1	10	8	6	9
	2	18	20	15	17
	3	15	16	13	17

If, one week, there are 35 working hours available at each manufacturing plant how much of each product should be produced given that we need at least 100 units of product 1, 150 units of product

2, and 100 units of product 3. Formulate this problem as a linear program. You do not have to solve the resulting LP.

Solution:

Now, we need to reduce the above problem to Linear programming by defining the variables, objective function, and the subject to constraints.

Declaration of variables:

It is given that we are given a total of 3 products, but they are produced by 4 factories. So, the total number of variables that we need to define is 12, which shall be used in the linear programming method.

X_{ij} is the variable that shall be used, Where 'i' is the product and 'j' be the factory.

So, the variables shall be as follow,

X_{11} is the number of units of product 1 produced in factory 1.

X_{12} is the number of units of product 1 produced in factory 2.

X_{13} is the number of units of product 1 produced in factory 3.

X_{14} is the number of units of product 1 produced in factory 4.

X_{21} is the number of units of product 2 produced in factory 1.

X_{22} is the number of units of product 2 produced in factory 2.

X_{23} is the number of units of product 2 produced in factory 3.

X_{24} is the number of units of product 2 produced in factory 4.

X_{31} is the number of units of product 3 produced in factory 1.

X_{32} is the number of units of product 3 produced in factory 2.

X_{33} is the number of units of product 3 produced in factory 3.

X_{34} is the number of units of product 3 produced in factory 4.

Objective Function:

In the question, it is given that we need to maximize the profit that we earn by selling the products that have been manufactured by different factories.

So, the Maximum objective function is as follows,

$MAX(10*X_{11} + 8*X_{12} + 6*X_{13} + 9*X_{14}$

$+ 18*X_{21} + 20*X_{22} + 15*X_{23} + 17*X_{24}$

$+ 15*X_{31} + 16*X_{32} + 13*X_{33} + 17*X_{34})$

In-Equality Constraints:

Now, we need to define the inequality constraints.

Here we have two in-equality constraints, one is the minimum number of products that need to be produced and the number of working hours for each factory each week.

Below are the constraints for the minimum number of products,

$X_{11} + X_{12} + X_{13} + X_{14} >= 100$

$X_{21} + X_{22} + X_{23} + X_{24} >= 150$

$X_{31} + X_{32} + X_{33} + X_{34} >= 100$

Below are the constraints for the working hours at a given factory in a week.

$5*X_{11} + 6*X_{21} + 13*X_{31} <= 35$

$7*X_{12} + 12*X_{22} + 14*X_{32} <= 35$

$4*X_{13} + 8*X_{23} + 9*X_{33} <= 35$

$10*X_{14} + 15*X_{24} + 17*X_{34} <= 35$

Since the production time for each product is given in minutes, we need to convert the working hours at each of the factories also into minutes, which shall result in the following inequalities.

$5*X_{11} + 6*X_{21} + 13*X_{31} <= 2100$

$7*X_{12} + 12*X_{22} + 14*X_{32} <= 2100$

$4*X_{13} + 8*X_{23} + 9*X_{33} <= 2100$

$10*X_{14} + 15*X_{24} + 17*X_{34} <= 2100$

The above is the Linear programming objective function that has been subject to the following constraints.

3. Solve the following linear program using the fundamental theorem. Specifically, find all vertices of the feasible region, calculate the values of the objective function at those points, and conclude the optimal solution. (*Hint: plot the feasible region in 2D*)

$$\max(-x_1 + 4x_2)$$

subject to

$$\begin{aligned} 3x_1 + x_2 &\leq 1 \\ 3x_1 + x_2 &\geq -5 \\ x_1 - x_2 &\leq 4 \\ x_1 - x_2 &\geq -2 \\ x_2 &\leq 1 \end{aligned}$$

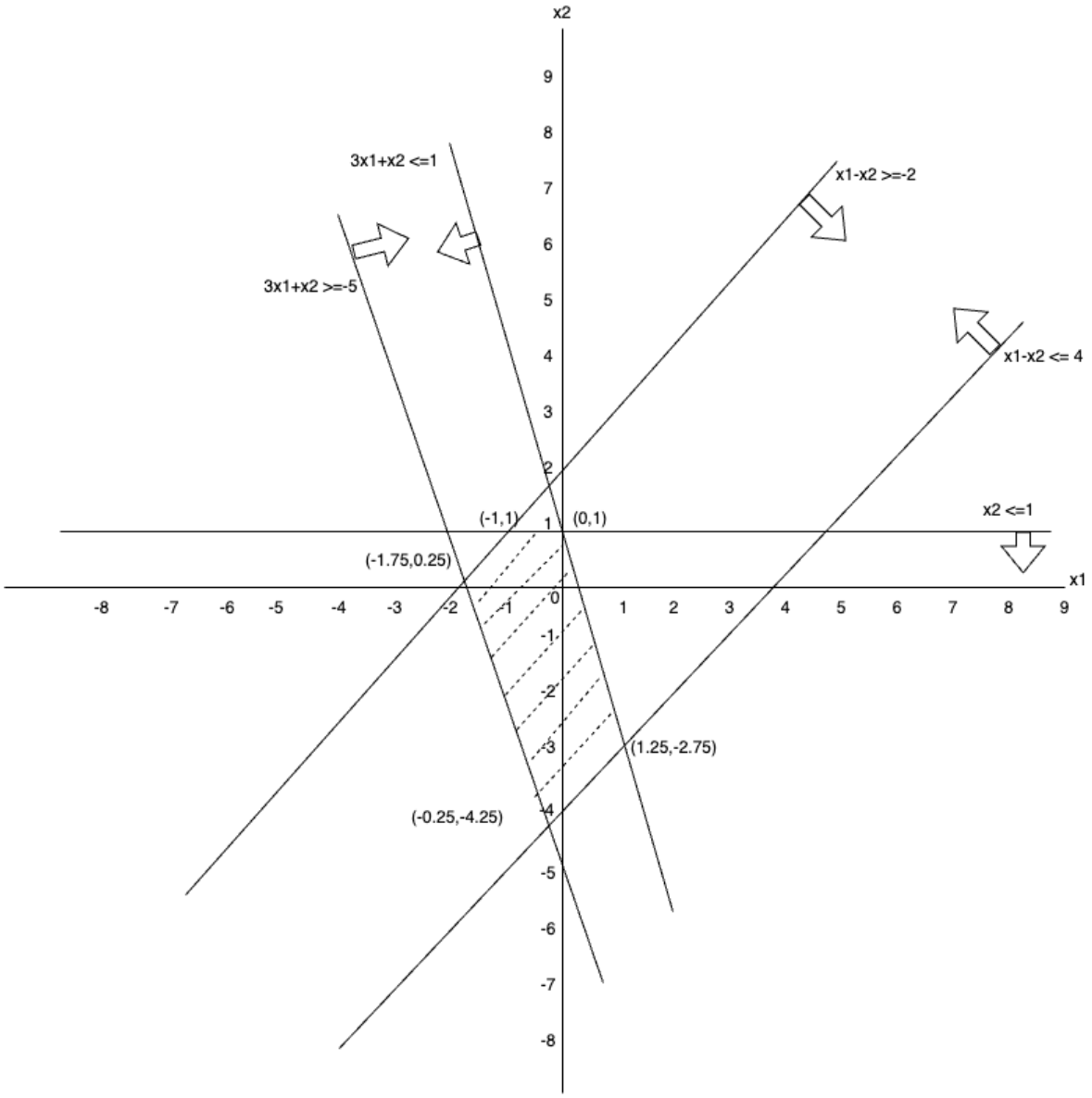
Solution:

Fundamental Theorem of Linear Programming:

As per the fundamental theorem of Linear programming, we need to plot the above-given equations on a graph to find the common area between the points, which shall help us find the feasible solution area between the given inequalities. Then we need to find a point in the feasible solution area such that the objective function has the maximum possible value.

From the fundamental theorem, we know that the maximum value of the objective function will lie along the polygon corners.

On plotting the inequations we get the below graph.



On solving each pair of equations, we get the following points,

A-> (-1,1)

B-> (0,1)

C-> (1.25,-2.75)

D-> (-0.25,-4.25)

E-> (-1.75,0.25)

Now we need to find substitute these points in the objective function that we have, and then find out which gives us the max value possible.

The objective function is as follows, Obj() = $-x_1 + 4x_2$

Obj(A) = 5

Obj(B) = 4

Obj(C) = 2.75

Obj(D) = -16.75

Obj(E) = -12.25

It is given in the question that we need to maximize the objective function, hence out of all the above points, we get the maximum possible value at point A, i.e., (-1,1).

Therefore, the maximum of the objective function can be found at **(-1,1)**.

4. **There are m basic nutritional ingredients, and Andy has to receive at least b_j units of the i-th nutrient per day to satisfy the basic minimum nutritional requirements. There are n available foods. The j-th food sells at a price c_j per unit and contains a_{ij} units of the i-th nutrient. Help Andy find the lowest cost per day to satisfy the requirement. Formulate this problem as a linear programming problem in the standard form.**

Solution:

- Let b1, b2, b3….bm be the basic 'm' nutritional ingredients that Andy must receive.
- Let c1, c2, c3….cn be the cost of the 'n' food items that she can purchase.
- Let a_{ij} be the amount of nutrients of type 'i' that the food type 'j' contains.
- We need to minimize the cost for Andy to buy the food items and still satisfy her daily nutrient requirements.

Declaration of Variables:

Let X_i be the quantity of the food 'i' that Andy purchases in order to maximize her nutrients.

X_i = 0, if the food is NOT chosen

X_i = 1, if the food is chosen

Objective Function:

$$MIN(\sum_{i=1}^{n}(Xi * Ci))$$

In-equality Constraints:

$a_{11}*X_1 + a_{12}*X_2 + a_{13}*X_3 \ldots\ldots\ldots + a_{1n}*X_n >= b_1$

$a_{21}*X_1 + a_{22}*X_2 + a_{23}*X_3 \ldots\ldots\ldots + a_{2n}*X_n >= b_2$

…… …

$a_{m1}*X_1 + a_{m2}*X_2 + a_{m3}*X_3 \ldots\ldots\ldots + a_{mn}*X_n >= b_m$

Converting into Standard Linear Programming Form:

Objective Function:

$$MAX(-\sum_{i=1}^{n}(Xi * Ci))$$

In-equality Constraints:

$-(a_{11}*X_1 + a_{12}*X_2 + a_{13}*X_3 \ldots\ldots\ldots + a_{1n}*X_n) <= -b_1$

$-(a_{21}*X_1 + a_{22}*X_2 + a_{23}*X_3 \ldots\ldots\ldots + a_{2n}*X_n) <= -b_2$

…… …

$-(a_{m1}*X_1 + a_{m2}*X_2 + a_{m3}*X_3 \ldots\ldots\ldots + a_{mn}*X_n) <= -b_m$

The above problem is an Integer Linear Program, which cannot be solved in polynomial time.

It can be considered as a NP Problem.

5. Given an undirected graph G = (V,E), a vertex cover is a subset of V so that every edge in E has at least one endpoint in the vertex cover. The problem of finding a minimum vertex cover is to find a vertex cover of the smallest possible size. Formulate this problem as an integer linear programming problem.

Solution:

Given that we have an undirected graph G =(V, E). The vertex covering problem is an NP-Hard problem.

We need to formulate the Vertex cover problem as the smallest possible integer linear programming.

Assumptions:

Let us assume that every vertex V is denoted by an integer. i.e., Vertex 1 V_1 be denoted by 1, Vertex 2 V_2 be denoted by 2, and so on.

So, if there are 'n' vertices in the set V, then the vertex V_n is denoted by '**n**'.

So for all the edges, that we have in set E, if a set of Vertices are connected, then they can be represented as a set of integers. i.e., if V_1 and V_2 are connected, then they can be represented as (1,2).

Similarly, if V_3 and V_n are connected, then they can be represented as (3,n).

Declaration of Variables:

Let X_i be the variable, such that $X_i = 0$ if a vertex 'i' V_i is NOT a part of the vertex cover, and $X_i = 1$ if a vertex 'i' V_i is a part of the vertex cover.

Objective Function:

The objective function is to find the minimum vertex cover, i.e.,

$$MIN\left(\sum_{i=1}^{n}(Xi)\right), where\ value\ of\ X\ belongs\ to\ \{0,1\}$$

In-Equality Constraints:

$Xi + Xj >= 1$, where i, j belongs to E.

And the value of X ∈ {0,1}

Converting into Standard linear Program format:

Declaration of Variables:

Let X_i be the variable, such that $X_i = 0$ if a vertex 'i' V_i is NOT a part of the vertex cover, and $X_i = 1$ if a vertex 'i' V_i is a part of the vertex cover.

Objective Function:

The objective function is to find the minimum vertex cover, i.e.,

$$MAX\left(-\sum_{i=1}^{n}(Xi)\right), where\ value\ of\ X\ belongs\ to\ \{0,1\}$$

In-Equality Constraints:

$-(Xi\ +\ Xj) <=\ -1$, where i, j belongs to E.

And the value of X ∈ {0,1}

The above problem is an Integer Linear Programming problem that cannot be solved in polynomial time.

6. Write down the dual program of the following linear program. There is no need to provide intermediate steps.

$$\max(x_1 - 3x_2 + 4x_3 - x_4)$$

subject to

$$\begin{aligned} x_1 - x_2 - 3x_3 &\leq -1 \\ x_2 + 3x_3 &\leq 5 \\ x_3 &\leq 1 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

Solution:

Given the following primal in the question, we need to formulate the dual for the linear program problem.

From the standard form, we know the following,

$$C = \begin{bmatrix} 1 \\ -3 \\ 4 \\ -1 \end{bmatrix}$$

$$B = \begin{bmatrix} -1 \\ 5 \\ 1 \\ 0 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & -1 & -3 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$X = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix}$$

Now, we need to determine the dual of the above primal.

We can assume that,

$$Y = \begin{bmatrix} y1 \\ y2 \\ y3 \\ y4 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -3 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Dual LP is,

$B^TY = min(-y1 + 5y2 + y3)$

Subject to the following constraints,

$A^TY >= C$

⇨ $y1 >= 1$
⇨ $-y1 + y2 >= -3$
⇨ $-3y1 + 3y2 + y3 >= 4$
⇨ $y1, y2, y3, y4 >= 0$

7. Determine whether the following linear programs are feasible bounded, feasible unbounded, or infeasible.

(a)

$$\max(x_1 + x_2)$$

subject to

$$\begin{aligned} x_1 + 2x_2 &\leq 3 \\ 3x_1 - x_2 &\leq 2 \\ -4x_1 - x_2 &\leq 2 \end{aligned}$$

(b)

$$\max(x_1 + x_2)$$

subject to

$$\begin{aligned} x_1 + 2x_2 &\geq 3 \\ 3x_1 - x_2 &\geq 2 \\ -4x_1 - x_2 &\geq 2 \end{aligned}$$

(c)

$$\max(x_1 + x_2)$$

subject to

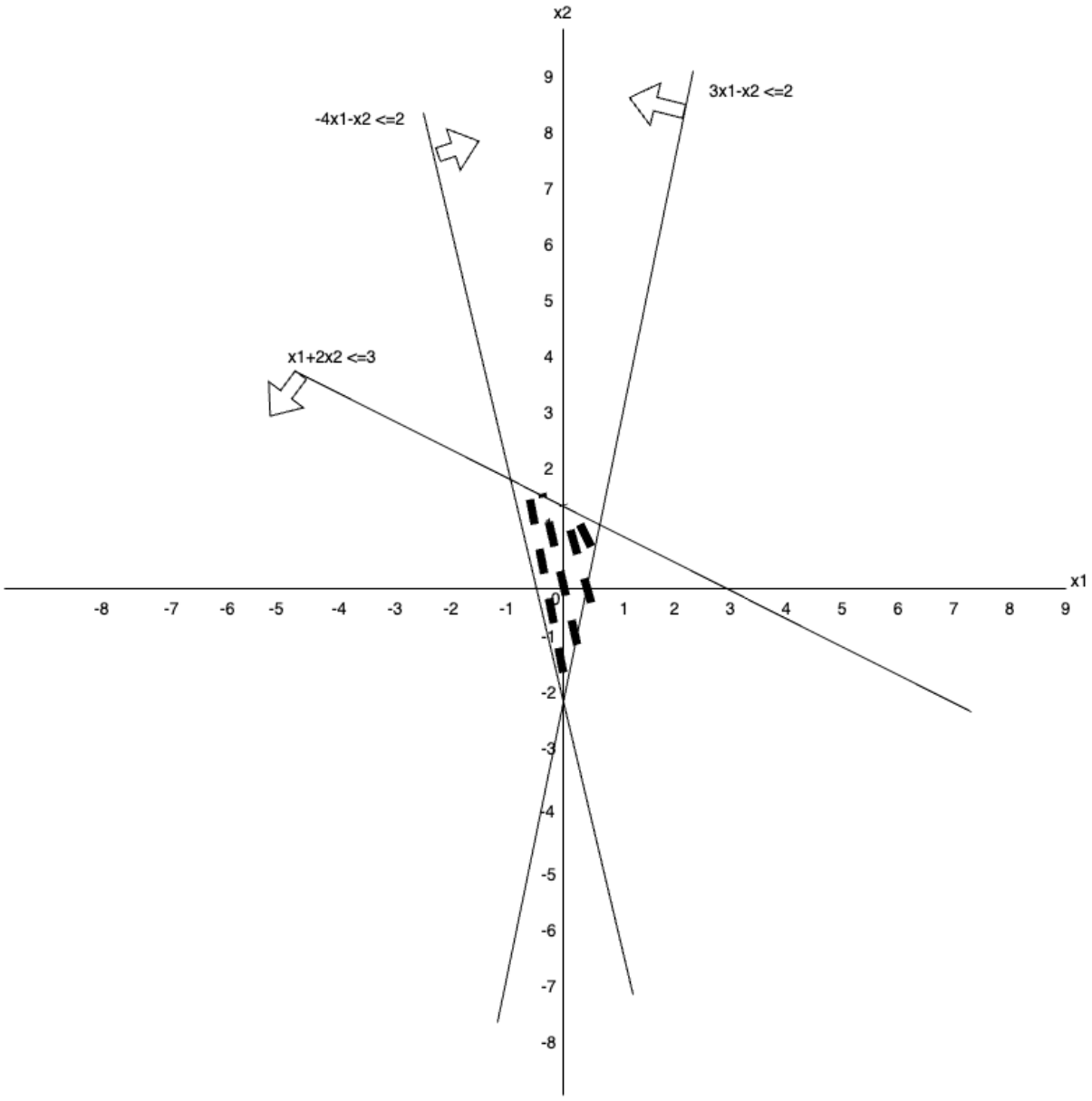
$$\begin{aligned} x_1 + 2x_2 &\geq 3 \\ 3x_1 - x_2 &\leq 2 \\ -4x_1 - x_2 &\leq 2 \end{aligned}$$

Solution:

(a).

Given that the objective function is $max(x1 + x2)$

So, for the given inequalities, we need to plot the graph and find the feasible solution area for



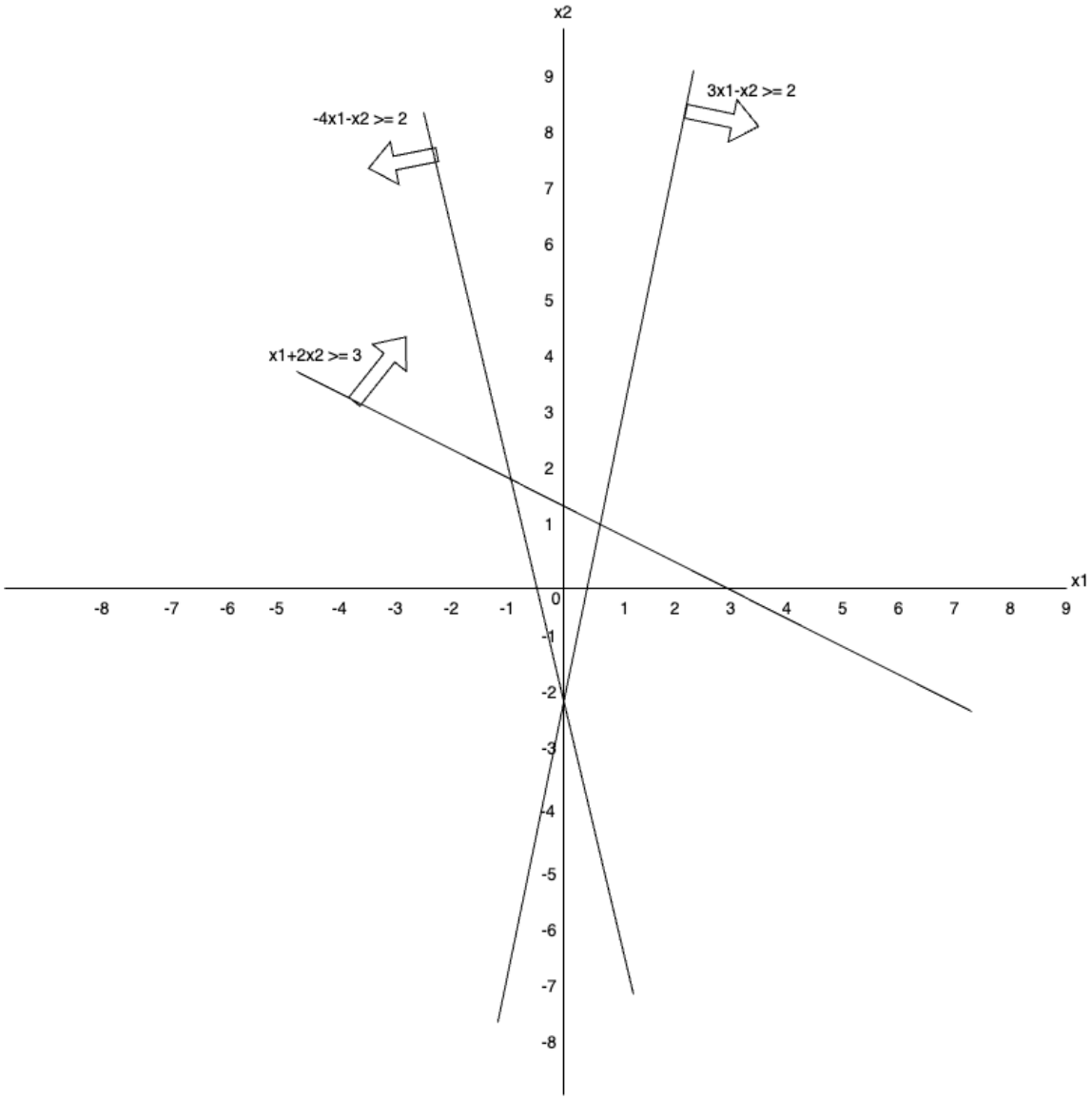
The above graph has a feasible area, the shaded area between the three lines, is the place where the solution lies.

So, it comes under **Feasible bounded category**.

(b).

Given that the objective function is $max(x1 + x2)$.

So, for the given inequalities, we need to plot the graph and find the feasible solution area for



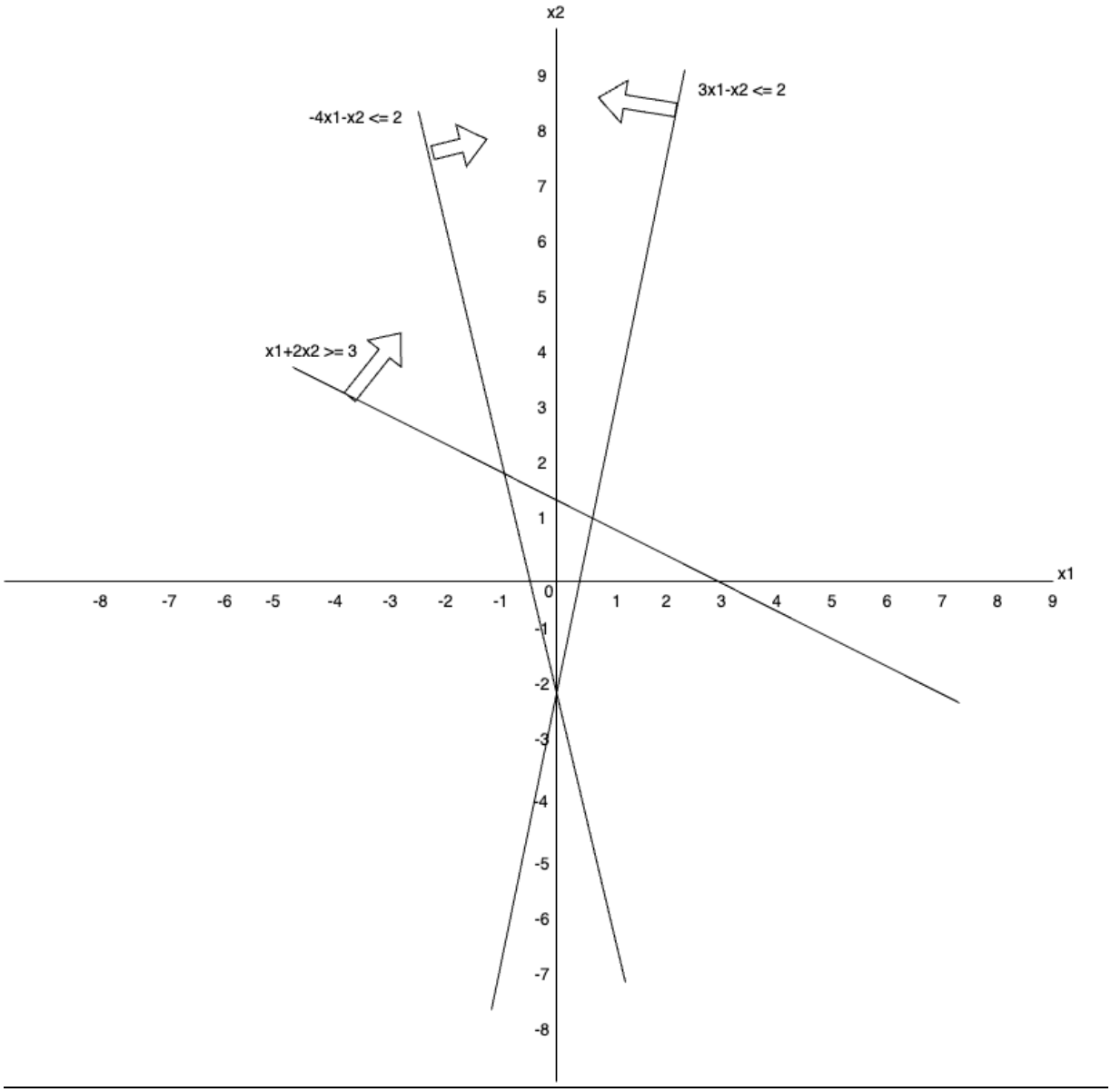
Here, we have some common area between the inequality equations, $x1 + 2x2 >= 3$ and $-4x1 - x2 >= 2$. But there is no common area between all the three inequality equations.

So, this comes under the **Infeasible category**.

(c).

Given that the objective function is $max(x1 + x2)$

So, for the given inequalities, we need to plot the graph and find the feasible solution area for



Here, we have some common areas between the inequality equations. But that area is not bounded.

So, this comes under the **Feasible unbounded category**.

8. **Show that vertex cover remains NP - Complete even if the instances are restricted to graphs with only even degree vertices.**

   Solution:

   It is given that the vertex cover problem is to decide that for a given graph G, whether the graph G has a vertex cover of size ‘k’.

We call this as vertex cover problem '**VC'**.

The Even Vertex Cover problem is defined as the Vertex cover problem, but with more constraints on the vertex cover problem.

We call this as even vertex cover problem **'VCE'**.

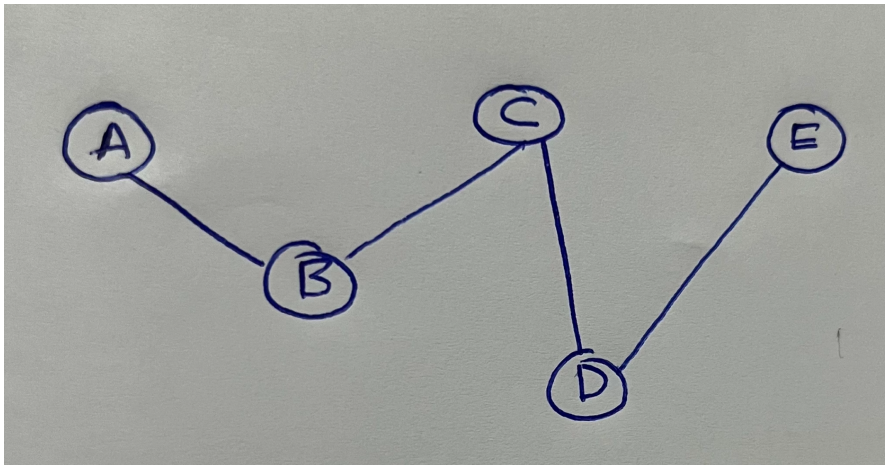
Here we need to restrict the instances of the VC to undirected graphs with only even degree vertices.

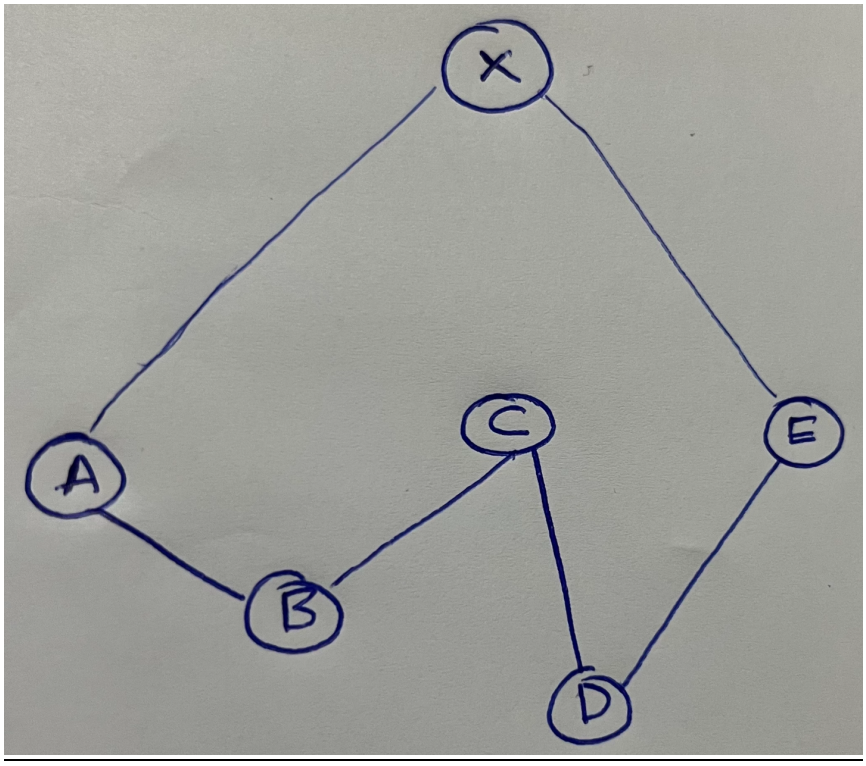
VCE is a NP Hard problem, follows the fact that the **VC** is also a NP Hard problem. As explained above, VCE is also a similar problem as VC, but only with more restrictions placed on the input.

Converting the above problem to Even Vertex Cover problem.

By Adding 1 Vertex:

1. Assume that the vertex cover size of graph G be **'k'**.
2. **We must note a simple fact that any undirected graph has an even number of odd degree vertices.**
3. By adding a new vertex to the undirected graph G, we get G'. and the size of the vertex cover of G' shall be **'k+1'.**





Claim: Graph G has a vertex cover of size **'k'**, if and only if, G' has a vertex cover of size **'k+1'**

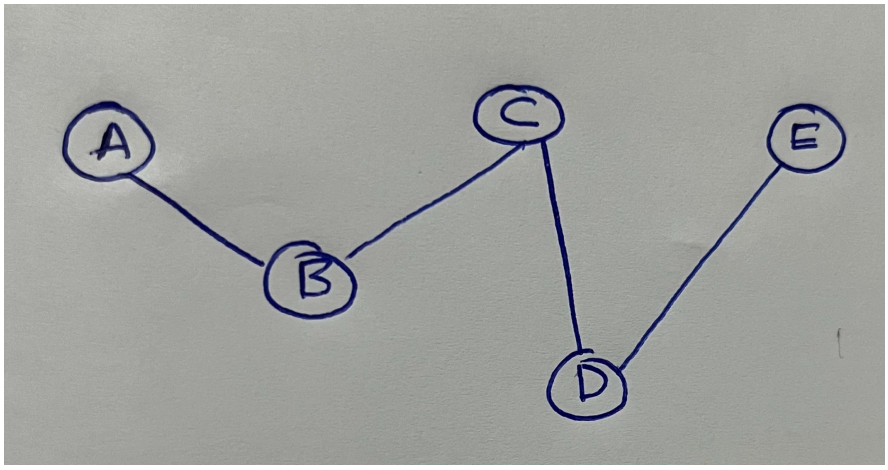
Proof for the claim:

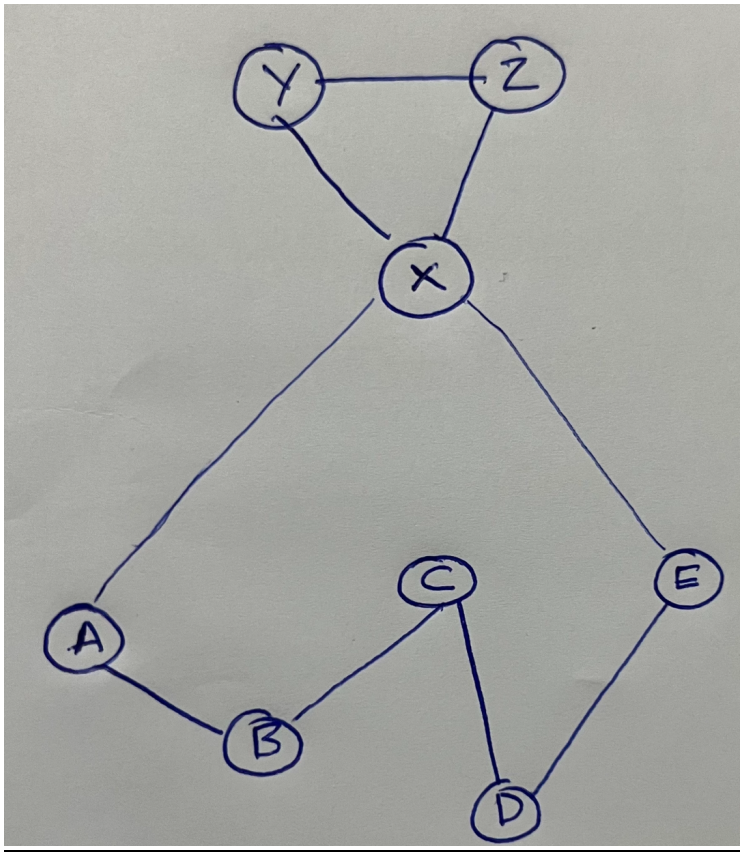
1. **Proving in the forward direction (=>):** Here we need to assume that G has a vertex cover of size **'k'.** Then the vertex cover of G' is created by adding a new vertex, thus the new vertex cover size shall be **'k+1'.**
2. **Proving in the backward direction (<=):** Here we need to assume that the vertex cover of G' is of size 'k+1', to get the vertex cover of G, we need to delete a vertex from G'. But this shall not always work as expected, since if we delete any of the vertices **other than the extra vertex that we have added**, that shall result in a completely different graph, and the vertex cover of that graph shall also be different.

Since by adding only 1 vertex to the graph G to construct G', the above reduction might not work in all cases. So, next we shall try by adding three vertices to the graph G.

By Adding 3 Vertex:

1. Assume that the vertex cover size of graph G be **'k'**.
2. **We must note a simple fact that any undirected graph has an even number of odd degree vertices.**
3. By adding three new vertices to the undirected graph G, we get G'. and the size of the vertex cover of G' shall be **'k+2'.**





Claim: Graph G has a vertex cover of size **'k'**, if and only if, G' has a vertex cover of size **'k+2'**

Proof for the claim:

1. **Proving in the forward direction (=>):** Here we need to assume that G has a vertex cover of size **'k'.** Then the vertex cover of G' is created by adding three new vertices, thus the new vertex cover size shall be **'k+2'.**

2. **Proving in the backward direction (<=):** Here we need to assume that G' has a vertex cover of size **'k+2',** to get G, we need to remove at least two vertices. These two vertices can be easily identified as they must be from the extra set of vertices that we have added. So, now the graph G has a vertex cover size of 'k' after deleting the two vertices.

9. Assume that you are given a polynomial time algorithm that given a 3-SAT instance decides in polynomial time if it has a satisfying assignment. Describe a polynomial time algorithm that finds a satisfying assignment (if it exists) to a given 3-SAT instance.

Solution:

In this case, we have a polynomial time algorithm that determines if a given 3-SAT instance has a satisfying assignment. We need to create a polynomial-time technique to locate the satisfying assignment to that 3-SAT instance.

Designing Algorithm:

We need to create a polynomial time algorithm, that finds the satisfying assignment.

As a first step, we shall try to randomly assign truth values to the 3- SAT instances. Once we have assigned then, we shall be using the polynomial time algorithm to check if we are able to get the satisfying assignment or not, but by one variable at a time.

Creating the Algorithm:

Given a 3-SAT instance,

$X = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge \ldots\ldots\ldots\ldots \wedge (x_{n-2} \vee x_{n-1} \vee x_n)$

Let 'n' be the count of variables in X:

3. Use the given algorithm, to check if X is satisfied or not by the assignment.
4. In case the assignment 'X' can't be satisfied, then there does not exist a solution. So, we can terminate the algorithm.
5. In case assignment 'X' can be satisfied, then we need to find the assignment.
6. Then we need to start by assigning the truth value 'T' to variable x_1 and then substitute back x_1 in 3-SAT X to find/create X_1.
7. Use the given algorithm, to check if 'X_1' is satisfied or not by the assignment.
8. Check, in case the assignment 'X_1' can be satisfied, then make the change x_1 = 'T' in X.
9. Check, in case the assignment 'X_1' can't be satisfied, then make the permanent change x_1 = 'F', then substitute x_1 in X to create $X_1^{|}$.
10. Now, either X_1 or $X_1^{|}$ shall be satisfiable because X is satisfiable.

11. Now, repeat the above steps for all the variables in X and make permanent changes in X.
12. The final assignment is the combination of all the variables in X.

The time complexity of the above algorithm:

For the above algorithm, we shall be repeating the given polynomial time algorithm ‘n’ times to find the satisfying assignment for all variables in X.

So, the time complexity of the above algorithm remains polynomial time.