CSCI 570 – ANALYSIS OF ALGORITHMS HOMEWORK – 5

1. In the network below (See Fig. 1), the demand values are shown on vertices (supply values are negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. Please complete the following steps.

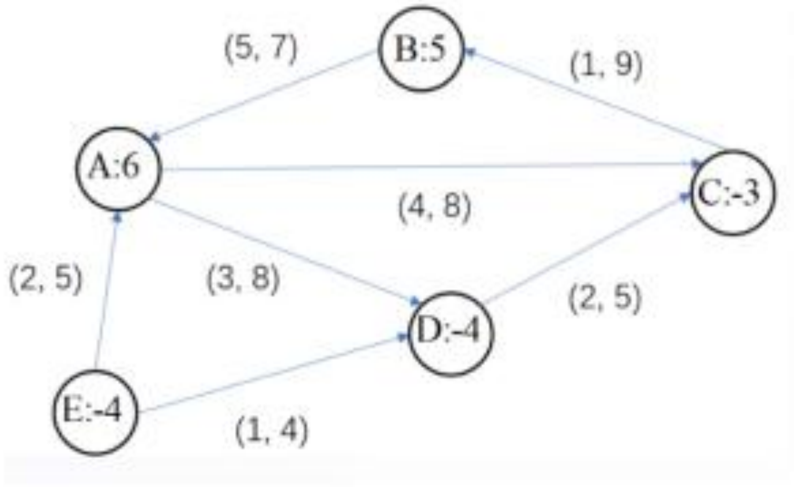


Figure 1

(a) Remove the lower bounds on each edge. Write down the new demands on each vertex A, B, C, D, E, in this order.

ANS:

(a)

New Demands after removing lower bounds i.e., we need to pass the flow through the graph with lower bound along each edge to eliminate lower bounds. After passing the lower-bound flow through the graph, the new demands are as follows

$d'(A)$: 6 – 2 – 5 + 3 + 4 = 6

$d'(B)$: 5 – 1 + 5 = 9

$d'(C)$: – 3 – 4 – 2 + 1 = – 8

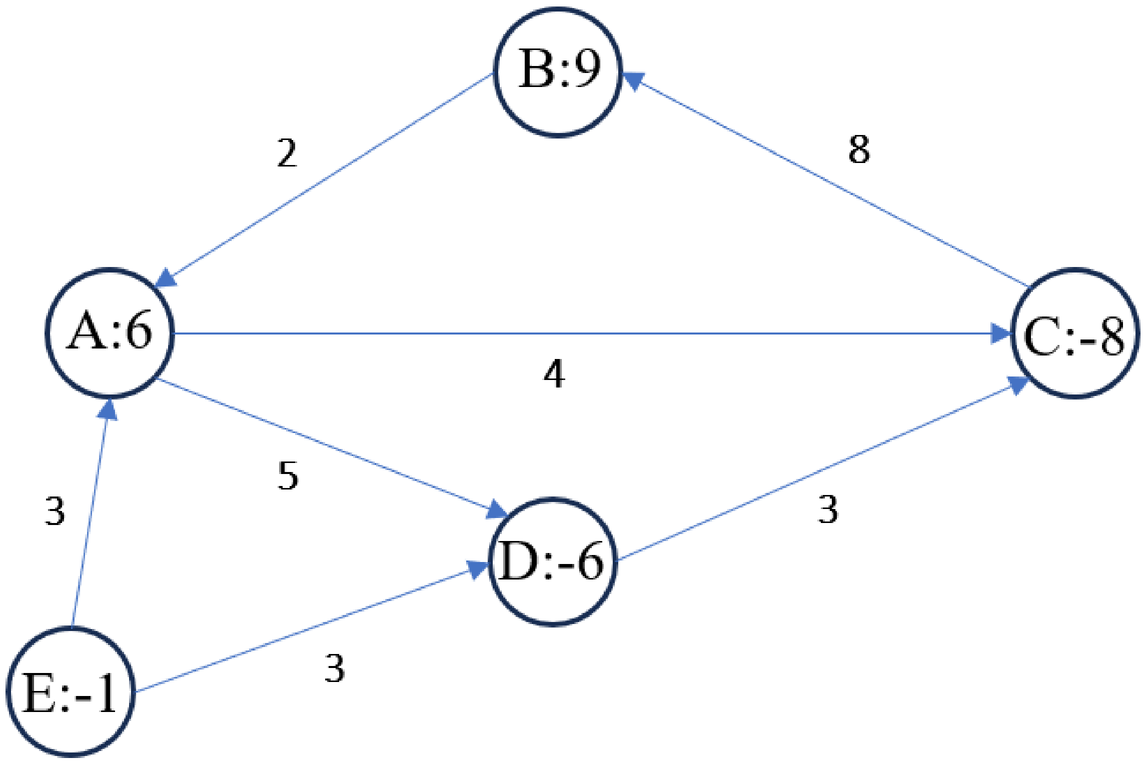
$d'(D)$: – 4 – 1 – 3 + 2 = – 6

$d'(E)$: – 4 + 1 + 2 = – 1

Sum of new demands after eliminating the lower bounds = 6 + 9 + (– 8) + (– 6) + (– 1) = 0 which satisfies the below necessary condition to have a valid circulation

For every feasible circulation $\sum_{v \in V} d(v) = 0$

We also update the Graph G to G' after updating the edge capacities after lower bound elimination.

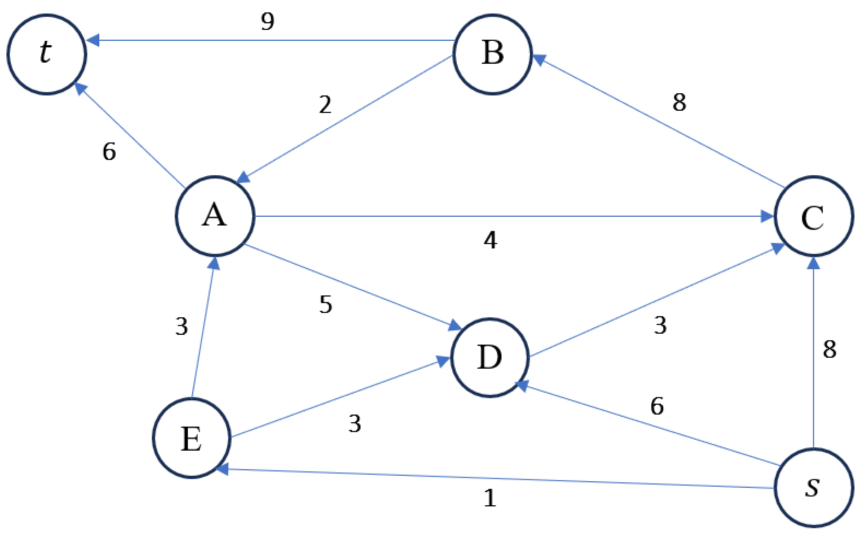


(b) Solve the circulation problem without lower bounds. Write down the max-flow value.

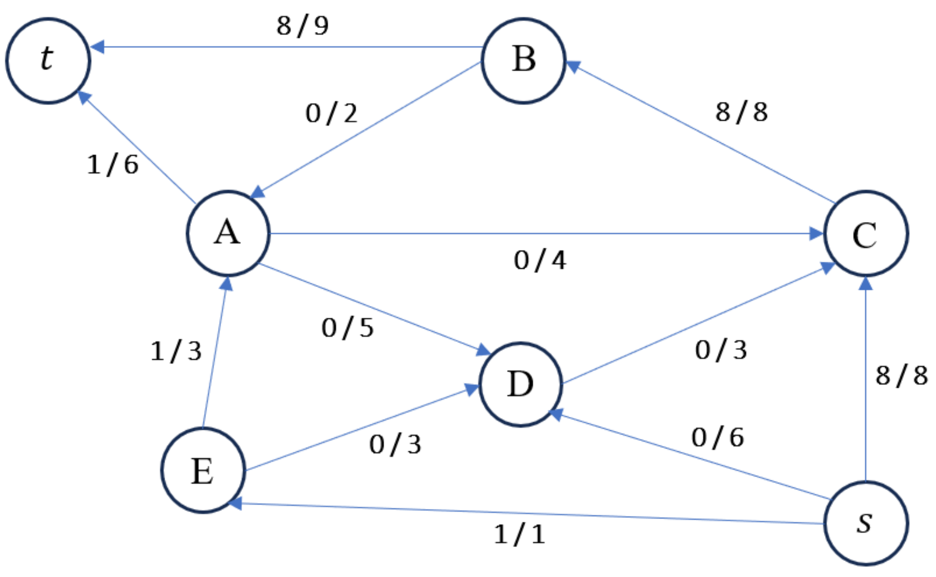
(b)

Now, we reduce the above graph G' to a Network Flow problem to find out the max-flow. We can reduce the above to a Network Flow by connecting the nodes with -ve supply to the Source s and +ve nodes demand to the Sink t

After completing the necessary updates to G', we obtain the Network Flow as follows



By computing the max-flow using Ford-Fulkerson Algorithm, we get max-flow as 9



(c) Is there is a feasible circulation in the original graph? Explain your answer.

(c)

Claim:

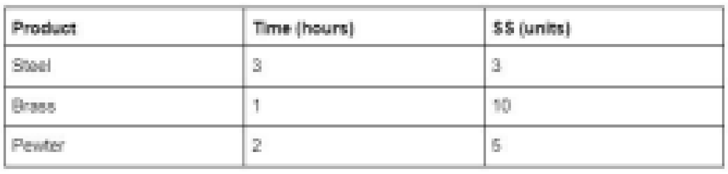
For every feasible circulation $\sum_{v \in V} d(v) = 0$

And ∃ Valid Circulation iff max-flow = 15 (All Edges going out of source should be saturated)

We should ideally get a max-flow of **15** for the circulation to be valid, but there may be a bottle-neck in between the flow. From the above observation, we have got a max-flow as 9.

Hence, circulation for the given original graph is not possible (not feasible).

3. A company is currently trying to fill a large order of steel, brass, and pewter, measured in tons. Manufacturing each ton of material requires a certain amount of time, and a certain amount of special substance (SS), both of which are limited and are given in below table. Note that it is acceptable to manufacture a fraction of a ton (e.g. 0.5t) of material. Specifically, the company currently has 8 hours of time, and 20 units of special substance (SS). Manufacturing each ton of the three products requires:



Product	Time (hours)	SS (units)
Steel	3	3
Brass	1	10
Pewter	2	5

Figure 2: Table describing the amount of time taken and special substance (SS) required for each product.

(a) Write down a linear program that determines the maximum amount of products (in tons) that the company can make.

ANS:

(a)

Let's define the decision variables for the amount of Steel (x_1), Brass (x_2), and Pewter (x_3) to be manufactured in tons.

Objective Function: The objective is to maximize the total amount of products, which can be expressed as:

$$\text{Maximize } Z = max\ (x_1 + x_2 + x_3)$$

Subject to system of inequalities,

Time constraint: The total time spent manufacturing the products should be less than or equal to 8 hours.

$$3x_1 + x_2 + 2x_3 \leq 8$$

Special substance (SS) constraint: The total amount of special substance used should be less than or equal to 20 units.

$$3x_1 + 10x_2 + 5x_3 \leq 20$$

Non-negativity constraint:

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \text{ and } x_1, x_2, x_3 \in R^+$$

This linear program represents the company's objective to maximize the production of steel, brass, and pewter while considering time and special substance constraints.

So, the linear program can be written as follows:

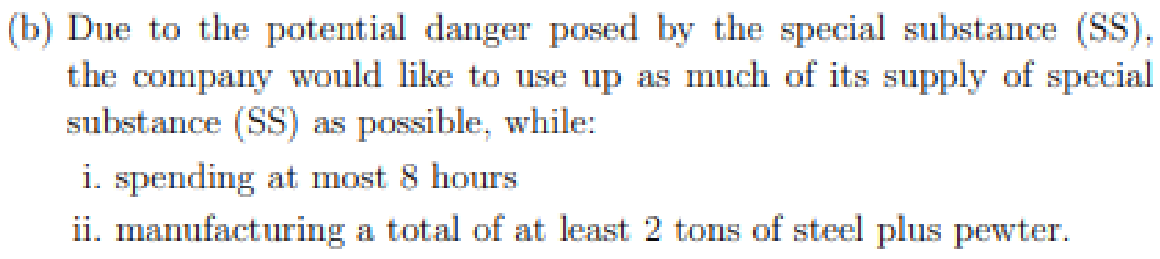
$$\text{Maximize } Z = max\,(x_1 + x_2 + x_3)$$

Subject to:

$$3x_1 + x_2 + 2x_3 \leq 8$$

$$3x_1 + 10x_2 + 5x_3 \leq 20$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0 \text{ and } x_1, x_2, x_3 \in R^+$$



(b) Due to the potential danger posed by the special substance (SS), the company would like to use up as much of its supply of special substance (SS) as possible, while:

i. spending at most 8 hours

ii. manufacturing a total of at least 2 tons of steel plus pewter.

(b)

Let's define the decision variables for the amount of Steel (x_1), Brass (x_2), and Pewter (x_3) to be manufactured in tons.

Objective Function: The objective is to maximize the total amount of products, which can be expressed as:

$$\text{Maximize } Z = max\,(x_1 + x_2 + x_3)$$

Subject to system of inequalities,

Time constraint: The total time spent manufacturing the products should be less than or equal to 8 hours (as given in problem statement "spending at most 8 hours").

$$3x_1 + x_2 + 2x_3 \leq 8$$

Additional Special substance (SS) constraint: Use the maximum amount of special substance as possible.

$$3x_1 + 10x_2 + 5x_3 = 20$$

While manufacturing at least 2 tons of Steel + Pewter:

$$x_1 + x_3 \geq 2$$

Non-negativity constraint:

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0 \text{ and } x_1, x_2, x_3 \in R^+$$

This adjusted linear program ensures that the company maximizes the use of special substance while meeting the time constraint and manufacturing at least 2 tons of steel plus pewter. The additional special substance constraint forces the maximum usage of special substance to be equal to the available amount (20 units).

So, the linear program can be written as follows:

$$\text{Maximize } Z = max\,(x_1 + x_2 + x_3)$$

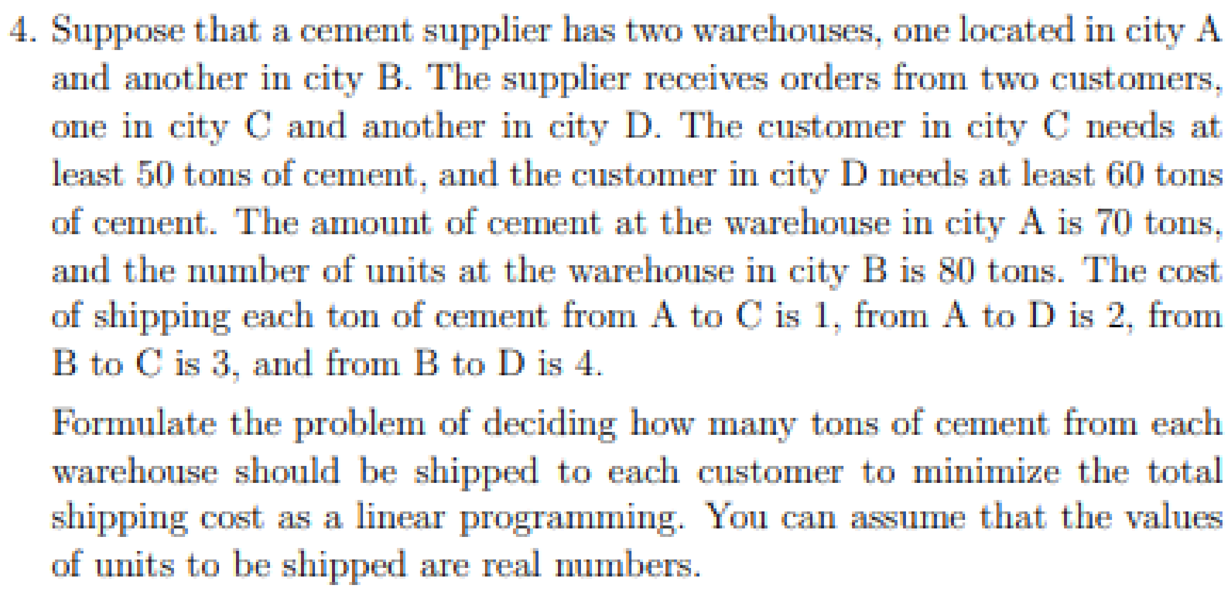
Subject to:

$$3x_1 + x_2 + 2x_3 \leq 8$$

$$3x_1 + 10x_2 + 5x_3 = 20$$

$$x_1 + x_3 \geq 2$$

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \text{ and } x_1, x_2, x_3 \in R^+$$



4. Suppose that a cement supplier has two warehouses, one located in city A and another in city B. The supplier receives orders from two customers, one in city C and another in city D. The customer in city C needs at least 50 tons of cement, and the customer in city D needs at least 60 tons of cement. The amount of cement at the warehouse in city A is 70 tons, and the number of units at the warehouse in city B is 80 tons. The cost of shipping each ton of cement from A to C is 1, from A to D is 2, from B to C is 3, and from B to D is 4.

   Formulate the problem of deciding how many tons of cement from each warehouse should be shipped to each customer to minimize the total shipping cost as a linear programming. You can assume that the values of units to be shipped are real numbers.

ANS:

Let's define the decision variables for the amount of cement to be shipped from each warehouse to each customer. Let x_{ij} represent the amount of cement (in tons) to be shipped from warehouse i to customer j.

Objective Function: The objective is to minimize the total shipping cost, which can be expressed as:

$$\text{Minimize } Z = min\,(1x_{AC} + 2x_{AD} + 3x_{BC} + 4x_{BD})$$

Subject to system of inequalities,

Customer demand constraints:

- Customer C needs at least 50 tons of cement:

$$x_{AC} + x_{BC} \geq 50$$

- Customer D needs at least 60 tons of cement:

$$x_{AD} + x_{BD} \geq 60$$

Warehouse availability constraints:

- Warehouse A has 70 tons of cement:

$$x_{AC} + x_{AD} \leq 70$$

- Warehouse B has 80 tons of cement:

$$x_{BC} + x_{BD} \leq 80$$

Non-negativity constraint:

$$x_{ij} \geq 0 \; where\ i\ is\ any\ Warehouse\ (A\ or\ B) and\ j\ is\ any\ customer\ (C\ or\ D)$$

So, the linear program can be written as follows:

$$\text{Minimize } Z = min\,(1x_{AC} + 2x_{AD} + 3x_{BC} + 4x_{BD})$$

Subject to:

$$x_{AC} + x_{BC} \geq 50$$

$$x_{AD} + x_{BD} \geq 60$$

$$x_{AC} + x_{AD} \leq 70$$

$$x_{BC} + x_{BD} \leq 80$$

$$x_{ij} \geq 0 \;\forall\, i \in \{A\ or\ B\}, j \in \{C\ or\ D\} \text{ i.e., } x_{AC} \geq 0,\ x_{AD} \geq 0,\ x_{BC} \geq 0 \text{ and } x_{BD} \geq 0$$

5. 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

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

ANS:

Primal:

Objective Function:

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

Subject to:

$$x_1 - x_2 - 3x_3 \leq\ -1$$

$$x_2 + 3x_3 \leq\ 5$$

$$x_3\ \leq\ 1$$

$$x_1 \geq 0,\ x_2 \geq 0, x_3 \geq 0, x_4 \geq 0$$

The above is in standard form (Let's reduce it into Matrix format)

$$\boldsymbol{A} = \begin{bmatrix} 1 & -1 & -3 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad \boldsymbol{C} = \begin{bmatrix} 1 \\ -3 \\ 4 \\ -1 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} -1 \\ 5 \\ 1 \end{bmatrix}$$

$$max\ (C^T\ X), AX \leq B, X \geq 0$$

Dual:

$$min\ (B^T\ Y)\ , A^T Y \geq C, Y \geq 0$$

Objective Function:

$$\text{Minimize } Z = min\ (-y_1 + 5y_2 + y_3)$$

Subject to:

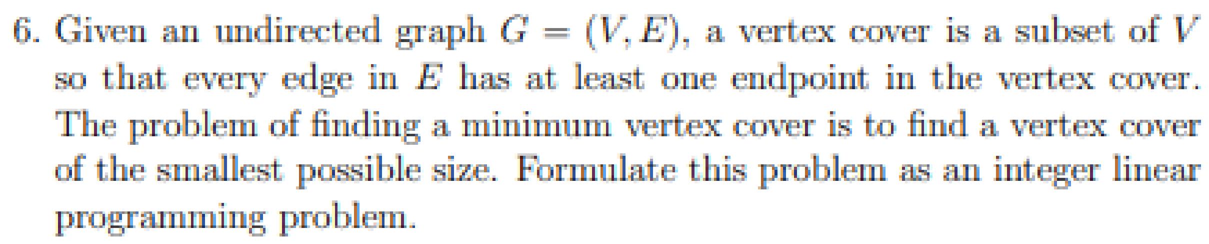
$$A^T = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -3 & 3 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \qquad C = \begin{bmatrix} 1 \\ -3 \\ 4 \\ -1 \end{bmatrix} \qquad B = \begin{bmatrix} -1 \\ 5 \\ 1 \end{bmatrix}$$

$$y_1 \geq 1$$

$$-y_1 + y_2 \geq\ -3$$

$$-3y_1 + 3y_2 + y_3 \geq 4$$

$$y_1 \geq 0,\ y_2 \geq 0,\ y_3 \geq 0$$



6. 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.

ANS:

Given an undirected graph $G = (V, E)$ with V be the set of vertices and E be the set of edges. We can use binary variables to represent whether a vertex is in the vertex cover or not. Let x_i be a binary variable that is equal to 1 if vertex v_i is in the vertex cover, and 0 otherwise.

The objective is to minimize the total number of vertices in the cover. The constraints ensure that every edge is covered by at least one endpoint in the vertex cover.

If there are n vertices in the graph i.e., n vertices of vertices set V, we can formulate the required Integer Linear Program (ILP) as follows:

Objective Function: The objective function seeks to minimize the total number of vertices in the vertex cover.

$$\text{Minimize Z} = min\ (x_1 + x_2 + \ldots\ldots + x_i \ldots. + x_n)$$

$$1 \leq i \leq n$$

Subject to:

For each edge $(a, b) \in E$:

$$x_a + x_b \geq 1$$

The above constraint ensures that at least one of the vertices $a\ or\ b$ is in the vertex cover.

For each $i \in V\ (1 \leq i \leq n)$:

$$x_i \in \{0, 1\}$$

$$x_i = \begin{cases} 0 & if\ the\ vertex\ is\ not\ a\ vertex\ cover \\ 1 & if\ the\ vertex\ is\ a\ vertex\ cover \end{cases}$$

The binary variable x_i takes values 0 or 1, indicating whether vertex i is in the vertex cover or not.

7. 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.

ANS:

The language for the formula satisfiability problem is 3-SAT= $\{\varphi_i$: φ is a satisfiable Boolean formula$\}$. Let φ be a Boolean formula and $\boldsymbol{A}$ be the polynomial-time algorithm to decide the 3-SAT. φ has n variables, denoted as $\{x_1, x_2, .., x_{i,} \ldots, x_n\}$. Denote φ (x_i = 0) and φ (x_i = 1) as the formulas when a variable x_i, $i \in \{1, 2, 3\}$ is assigned 0 or 1, respectively.

The following procedure can find a satisfying assignment:

1. Apply $'A'$ on φ to decide whether φ is satisfiable. If NO, return "No satisfying assignment can be found"; else continue.

2. Initialize i = 1.

3. Let x_i = 0, then φ (x_i = 0) is a Boolean formula with $n - i$ variables. Apply A to check the satisfiability of φ (x_i = 0).

- If it returns YES, record x_i = 0.
- Else record x_i = 1. Since φ is satisfiable, there must be an assignment of x_i that satisfies φ.

4. Replace x_i in φ by the recorded value from step (3), then increase i by 1, repeat step (3) until $i = n$

5. Return the recorded assignments for all variables $\{x_1, x_2, .., x_{i,} \ldots, x_n\}$.

Input: A CNF formula ϕ with n variables $x_1, x_2,.., x_{i,}... , x_n$.

Output: A truth assignment to the variables that satisfies ϕ, or None if there is no satisfying assignment.

Claim: The CNF formula $\phi_{x_i=1}$ is satisfiable if and only if ϕ has a satisfying assignment where x_i = True.

First, if $\phi_{x_i=1}$ has a satisfying assignment, then we can augment that satisfying assignment by setting x_i = True and this will satisfy ϕ (note that the only clauses we removed from ϕ to obtain $\phi_{x_i=1}$ have x_i in them, and hence setting x_i = True will satisfy all those clauses). On the other hand, if ϕ has a satisfying assignment where x_i = True, then that assignment restricted to the variables other than x_i will satisfy $\phi_{x_i=1}$.

Time Complexity:

The algorithm runs in polynomial time. Specifically, suppose $A(\phi)$ runs in $O(N^c)$ time, where N the total size of ϕ (sum of the clause sizes). Then SatAssignment(ϕ) runs in time $O(n.N^c)$ since the formula size is only decreasing in each iteration and there are at most n iterations.

8. The graph five-coloring problem is stated as follows: Determine if the vertices of G can be colored using 5 colors such that no two adjacent vertices share the same color. Prove that the five-coloring problem is NP-complete.
   Hint: You can assume that graph 3-coloring is NP-complete

ANS:

In order to prove that the given Five-Coloring problem is NP-Complete, we need to show that

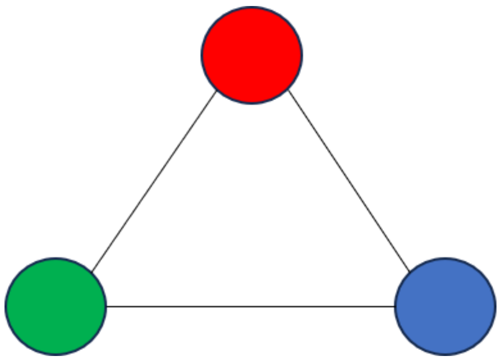
1. Five-Coloring Problem is NP
2. Five-Coloring Problem is NP-Hard
3. Three-Coloring Problem (Known NP-Complete Problem) $\leq_p$ Five-Coloring Problem

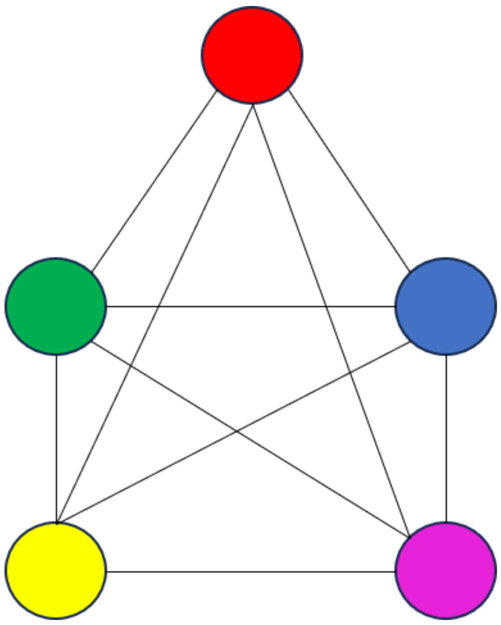
For a given problem to be NP, we need to show that, if we are given a solution to the problem, we should be able to verify if the solution satisfies all the constraints in polynomial time.

For the Five-Coloring problem, given a coloring combination for the vertices of a graph G using five colors, we can verify whether this is a valid coloring (i.e., no two adjacent vertices have the same color) in polynomial time. This verification involves checking each edge of the graph, which can be done in polynomial time relative to the size of the graph i.e., $O(E)$. Therefore, the Five-Coloring problem is in NP.

Construction:

Take an instance of the graph Three-Coloring problem, which asks whether a graph G can be colored using three colors such that no two adjacent vertices share the same color. Construct a new graph G' by adding two new vertices connected to each other and each new vertex to every existing vertex in G.


G


G'

Claim: Graph G can be Three-Colored if and only if G' can be Five-Colored.

Proof in Forward Direction:

If G can be Three-Colored, then obviously G' can be Five-Colored by using the same coloring logic for G and two additional colors for the two new vertices.

Proof in Backward Direction:

If G' can be Five-Colored, then the two new vertices must be colored with different colors (since they are connected), and they both connect to all vertices in G, so none of these two colors can be used in G. This means that the remaining part of G' (which is G) is colored with at most three colors.

Therefore, Five-Coloring problem is at least as hard as the Three-Coloring problem. Thus, the Five-Coloring problem is NP-hard.

Conclusion: Since the Five-Coloring problem is both in NP and NP-hard we can conclude that Five-Coloring problem is NP-complete.

9. Longest Path is the problem of deciding whether a graph $G = (V, E)$ has a simple path of length greater or equal to a given number k. Prove that the Longest path Problem is NP-complete by reduction from the Hamiltonian Path problem.

<u>ANS:</u>

In order to prove that the given Longest Path problem is NP-Complete, we need to show that

1. Longest Path Problem is NP
2. Longest Path Problem is NP-Hard
3. Hamiltonian Path Problem $\leq_p$ Longest Path problem

For a given problem to be NP, we need to show that, if we are given a solution to the problem, we should be able to verify if the solution satisfies all the constraints in polynomial time.

Let us assume that we are given a solution, where the length of the longest path is K in the given graph $G' = (V', E')$. We can verify the solution by performing a traversal over the graph such that each vertex is visited only once and compare the longest path obtained from the traversal with the value K. Thus, we can verify the solution in polynomial time $O(V' + E')$.

<u>Construction:</u>

Given an instance of the Hamiltonian Path Problem for a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. We construct an instance for the Longest Path problem G' as follows:

Let the length of the Hamiltonian path in the given graph $G = (V, E)$ is K ($|V| - 1$). Now, we need to create a graph G' such that $K = |V| - 1$. This graph will be enough for the Hamiltonian path problem to be reduced as the Longest Path problem.

<u>Claim:</u>

There exists a Hamiltonian Path K for G iff there exists a simple path of length K in graph G'.

<u>Proof in Forward Direction:</u>

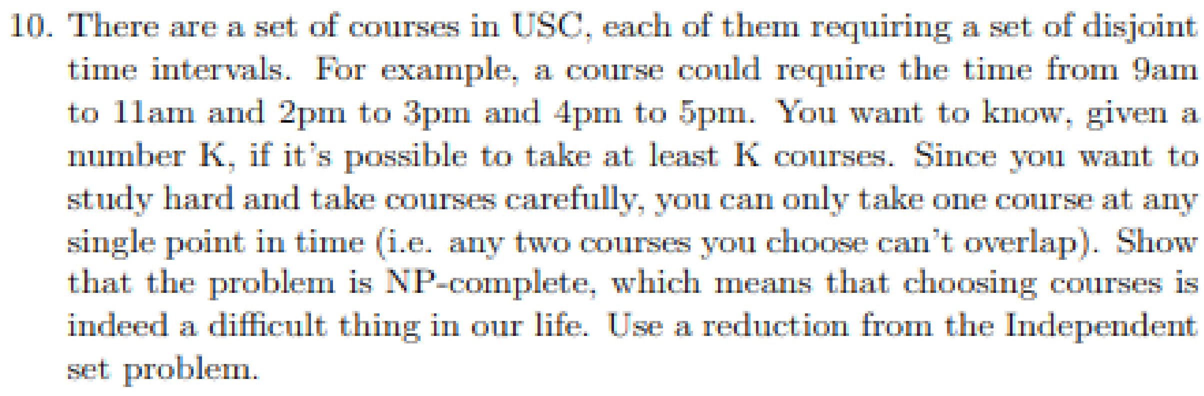
If G has a Hamiltonian Path, it includes all vertices of V, as it corresponds to a simple path that visits every vertex exactly once in G. In G', it translates to the Longest Path of length $|V| - 1$.

<u>Proof in the Backward Direction:</u>

If G' has a longest path of length $|V| - 1$, this implies that a path in G' covers every vertex exactly once. It directly translates to the Hamiltonian Path in G.

Therefore, the Longest Path problem is at least as hard as the Hamiltonian Path. Thus, the Longest Path problem is NP-Hard.

<u>Conclusion:</u> Since the Longest path problem is both NP and NP-Hard. We can conclude that the longest path problem is NP-Complete.


10. There are a set of courses in USC, each of them requiring a set of disjoint time intervals. For example, a course could require the time from 9am to 11am and 2pm to 3pm and 4pm to 5pm. You want to know, given a number K, if it's possible to take at least K courses. Since you want to study hard and take courses carefully, you can only take one course at any single point in time (i.e. any two courses you choose can't overlap). Show that the problem is NP-complete, which means that choosing courses is indeed a difficult thing in our life. Use a reduction from the Independent set problem.

<u>ANS:</u>

In order to prove that the given course scheduling problem at USC is NP-complete, we need to show that

1. Class Scheduling Problem is NP
2. Class Scheduling Problem is NP-Hard
3. Independent Set Problem $\leq_p$ Class Scheduling Problem Construction

For a given problem to be NP, we need to show that, if we are given a solution to the problem, we should be able to verify if the solution satisfies all the constraints in polynomial time.

Given a solution K indicating the number of courses to take, we can verify it efficiently. By checking if it's possible to schedule at least K courses without any overlapping time intervals, we can verify the solution in polynomial time by iterating through the courses and their time intervals.

<u>Construction:</u>

Independent set problem states that in a graph G there exists a set of K vertices where no vertices in the set K are adjacent (K $\subseteq V$).

Given an instance of the Independent Set problem, which is a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges, we construct an instance for the course scheduling problem G' as follows:

For each vertex v_i in G, create a course C_i in G' requiring the time intervals of v_i's neighbours. Specifically, if $(v_i,\ v_j)$ is an edge in G, then C_i requires the time intervals used by C_j indicating an edge between $(C_i,\ C_j)$ to ensure there are no overlapping time intervals .

<u>Claim:</u> There exists an independent set of size K in G iff it's possible to take at least K courses in the course scheduling instance such that there is no overlap in time intervals.

<u>Proof in Forward Direction:</u>

If there exists an independent set of vertices with size K in G such that no two vertices are connected by an edge. It means there are a set of K vertices (courses) in G' with nonoverlapping time intervals. Therefore, in the course scheduling instance, you can take K courses simultaneously where corresponding time intervals won't overlap.

Proof in Backward Direction:

If it's possible to take at least K courses in the course scheduling instance without overlapping time intervals in G', these courses must correspond to K vertices in G, forming an independent set.

Therefore, the Course Scheduling Problem is at least as hard as the Independent Set problem. Thus, Course Scheduling Problem is NP-Hard.

Conclusion:

The reduction from Independent Set to Course Scheduling problem is polynomial-time, and the solution for Course Scheduling problem can be verified in polynomial time (which indicates this is NP). As Course Scheduling is both NP and NP-Hard, we establish that Course Scheduling problem as NP-Complete.

******************************* **THE END** ********************************