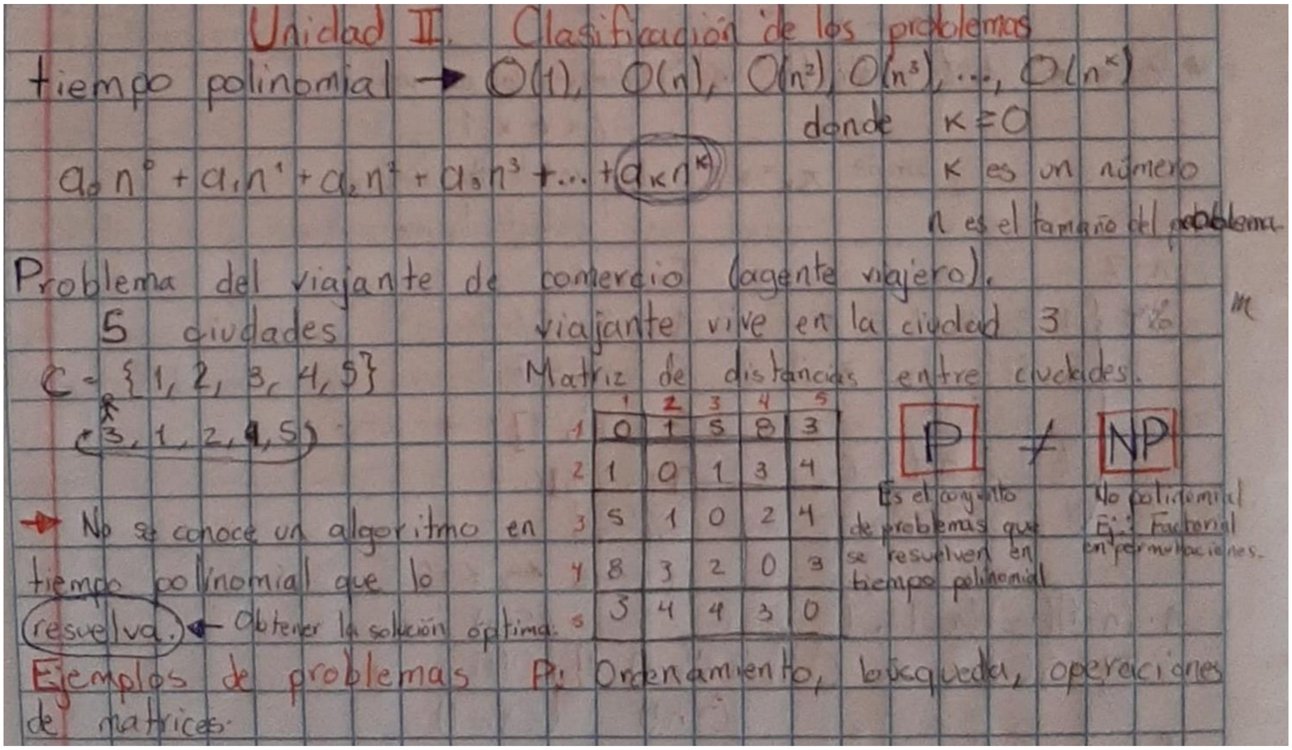
GUIA PARCIAL 2

- Clasificación de los problemas:

Tiempo polinomial: aquellos tiempos que están representados por: O(1), O(n), O(n^2), O(n^3), …, O(n^k). K es un número, n es el tamaño del problema.



Unidad II. Clasificación de los problemas

tiempo polinomial → $O(1)$, $O(n)$, $O(n^2)$, $O(n^3)$, ..., $O(n^k)$

donde $K \neq 0$

$a_0 n^0 + a_1 n^1 + a_2 n^2 + a_3 n^3 + \dots + a_k n^k$

K es un número

n es el tamaño del problema.

Problema del viajante de comercio (agente viajero).

5 ciudades

viajante vive en la ciudad 3

C = {1, 2, 3, 4, 5}

(3, 1, 2, 4, 5)

Matriz de distancias entre ciudades.

	1	2	3	4	5
1	0	1	5	8	3
2	1	0	1	3	4
3	5	1	0	2	4
4	8	3	2	0	3
5	3	4	4	3	0

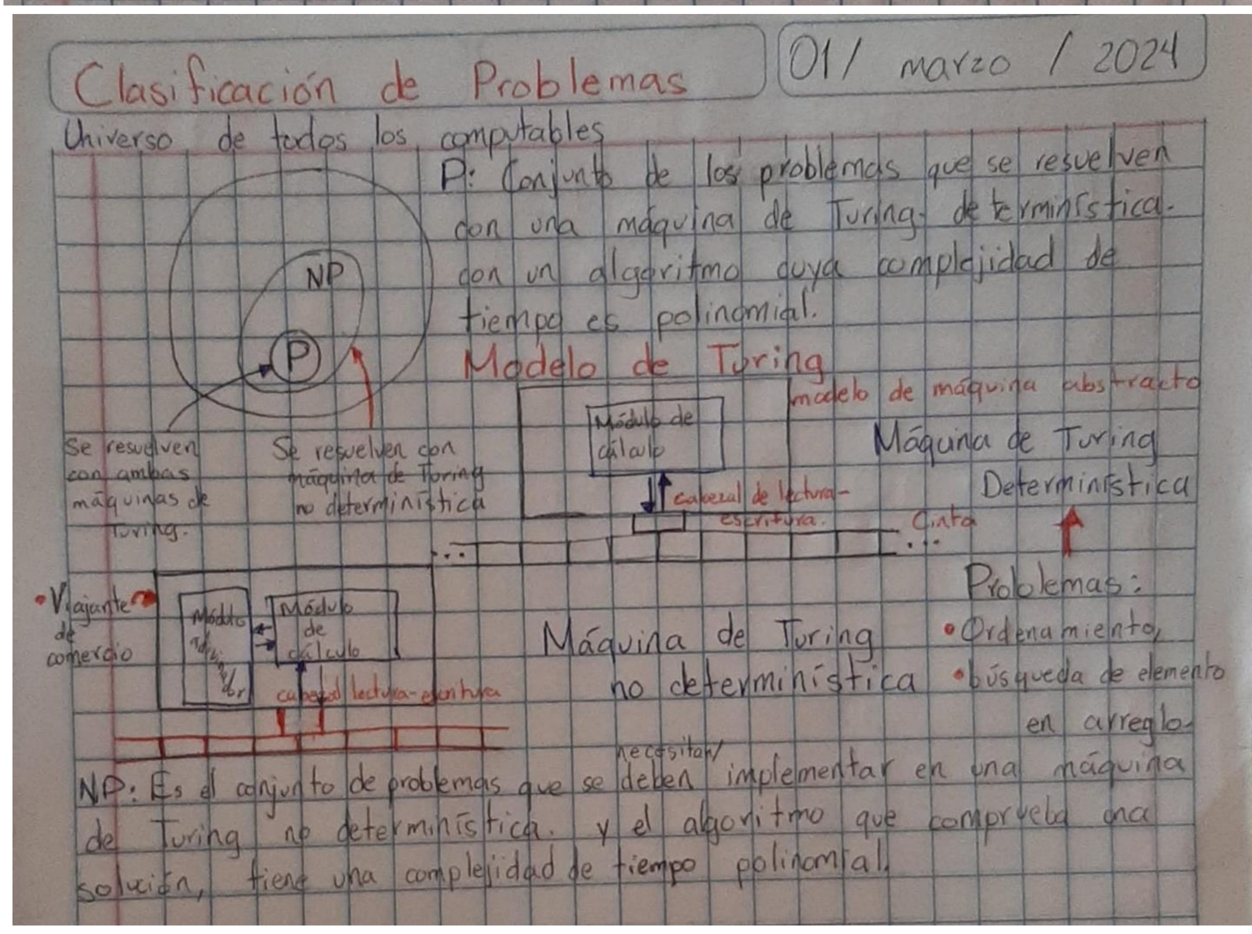
P ≠ NP

P: Es el conjunto de problemas que se resuelven en tiempo polinomial

NP: No polinomial Ej: Factorial en permutaciones.

→ No se conoce un algoritmo en tiempo polinomial que lo resuelva. ← Obtener la solución óptima.

Ejemplos de problemas P: Ordenamiento, búsqueda, operaciones de matrices.



Clasificación de Problemas

01/ marzo / 2024

Universo de todos los computables

P: Conjunto de los problemas que se resuelven con una máquina de Turing determinística. con un algoritmo cuya complejidad de tiempo es polinomial.

Modelo de Turing

Máquina de Turing Determinística

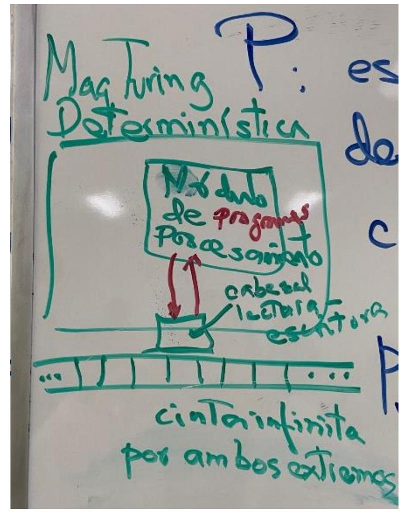
Problemas:
- Ordenamiento,
- búsqueda de elemento en arreglo.

- Viajante de comercio

Máquina de Turing no determinística

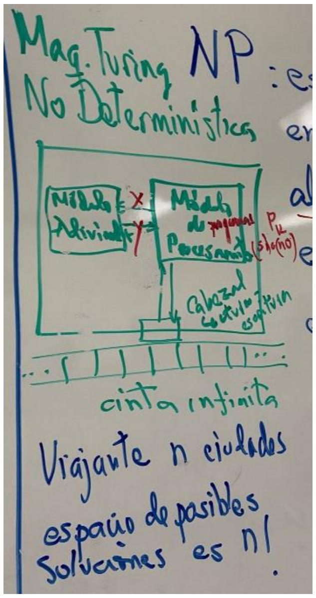
NP: Es el conjunto de problemas que se deben/necesitan implementar en una máquina de Turing no determinística. y el algoritmo que comprueba una solución, tiene una complejidad de tiempo polinomial.

- Ejemplos de problemas P y NP:

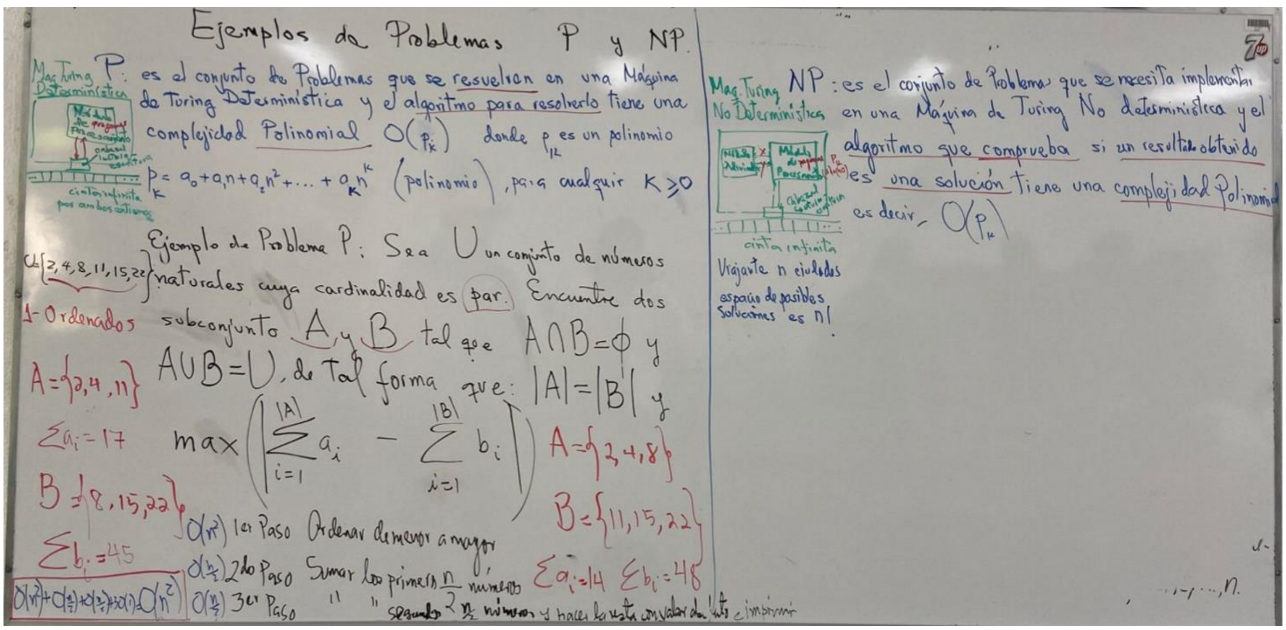


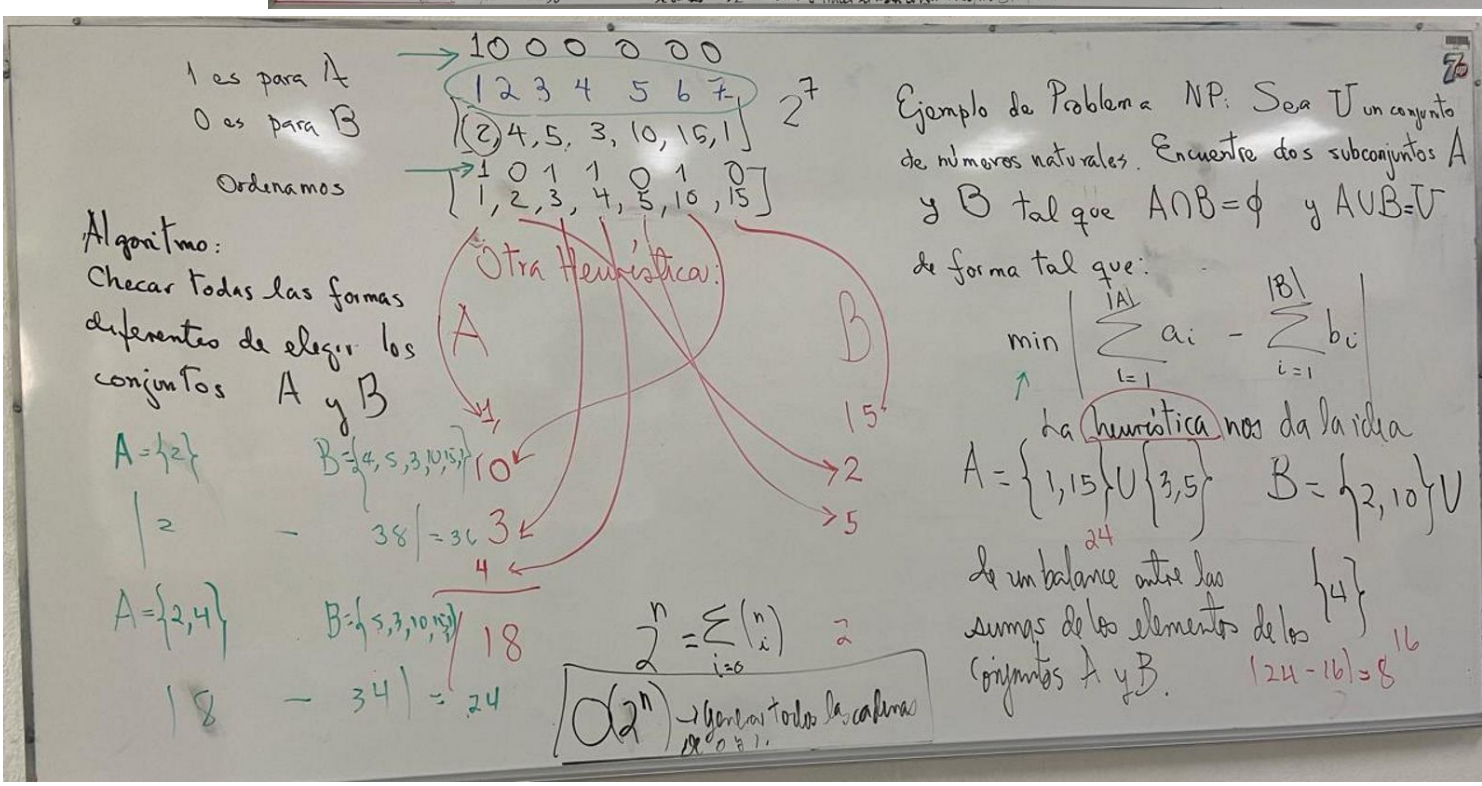
P: es el conjunto de problemas que se resuelven en una Máquina de Turing Determinística y el algoritmo para resolverlo tiene una complejidad Polinomial O(P_k) donde P_k es un polinomio.

$P_k = a_0 + a_1 n + a_2 n^2 + \cdots + a_n n^k$ $(polinomio)$, para adquirir k≥0.



NP: es el conjunto de problemas que se necesita implementar en una máquina de Turing No determinística y el algoritmo que comprueba si un resultado obtenido es una solución tiene complejidad Polinomial, es decir O(P_k).





Algoritmo para Generar las cadenas Binarias en orden Lexicográfico

```
000
001
010
011
100
```

Funcion_Imprimir_Menor() {
print ("Menor", Menor);

```
    3 2 1
1   0 0 0
2   0 0 1   1
3   0 1 1   2
4   0 1 0   1
5   1 1 0   3
6   1 1 1   1
7   1 0 1   2
    1 0 0   1
```

$n = 3$ $2^3 = 8$

Algoritmo Generación de código Gray reflectivo binario.

```
menor ← ∞
for j=0 to n+1 do { g_i ← 0
                    τ_j ← j+1
i ← 0
while i < n+1 { output(g_n, g_{n-1}, ..., g_1)
                funcion_Sumador_A y B
                i ← τ_0
                g_i ← ḡ_i
                τ_{i-1} ← τ_i
                τ_i ← i+1
                if i ≠ 1 then τ_0 ← 1
funcion_Imprimir_Menor();
```

```
          1 2 3
función_Sumador_A y B() {   V[5 2 6]
  sumaA=0; sumaB=0;
  for i=n to 1 {
    if g[i]==1 sumaA=sumaA+V[i];
    else sumaB=SumaB+V[i];
  }
  if Menor > abs(sumaA-sumaB){menor ← abs(SumaA-SumaB); Funcion Guarda_GMenor(); }
}
```

1 2 3
0 0 0

A={}
B={5,2,6}

Leyenda
Si 1 pongo el elemento en el sumador del conjunto A
Si 0 pongo el elemento el sumador del conjunto B

```
Pseudocódico para min(|Σa_i - Σb_i|) {
  // Generando cadenas binarias con el código Gray
  menor ← ∞
  Función_Leer(U)()
  for j=0 to n+1 do { g_j ← 0
                      τ_j ← j+1
  i ← 0
  while(i < n+1) { output(g_n, g_{n-1}, ..., g_1)
                   Funcion_Sumador A y B()
                   i ← τ_0
                   g_i ← ḡ_i
                   τ_{i-1} ← τ_i
                   τ_i ← i+1
                   if i ≠ 1 then τ_0 ← 1
  Funcion_Imprimir_menor()
}
```

1 2 3 4 5
U=[3, 4, 10, 8, 2]

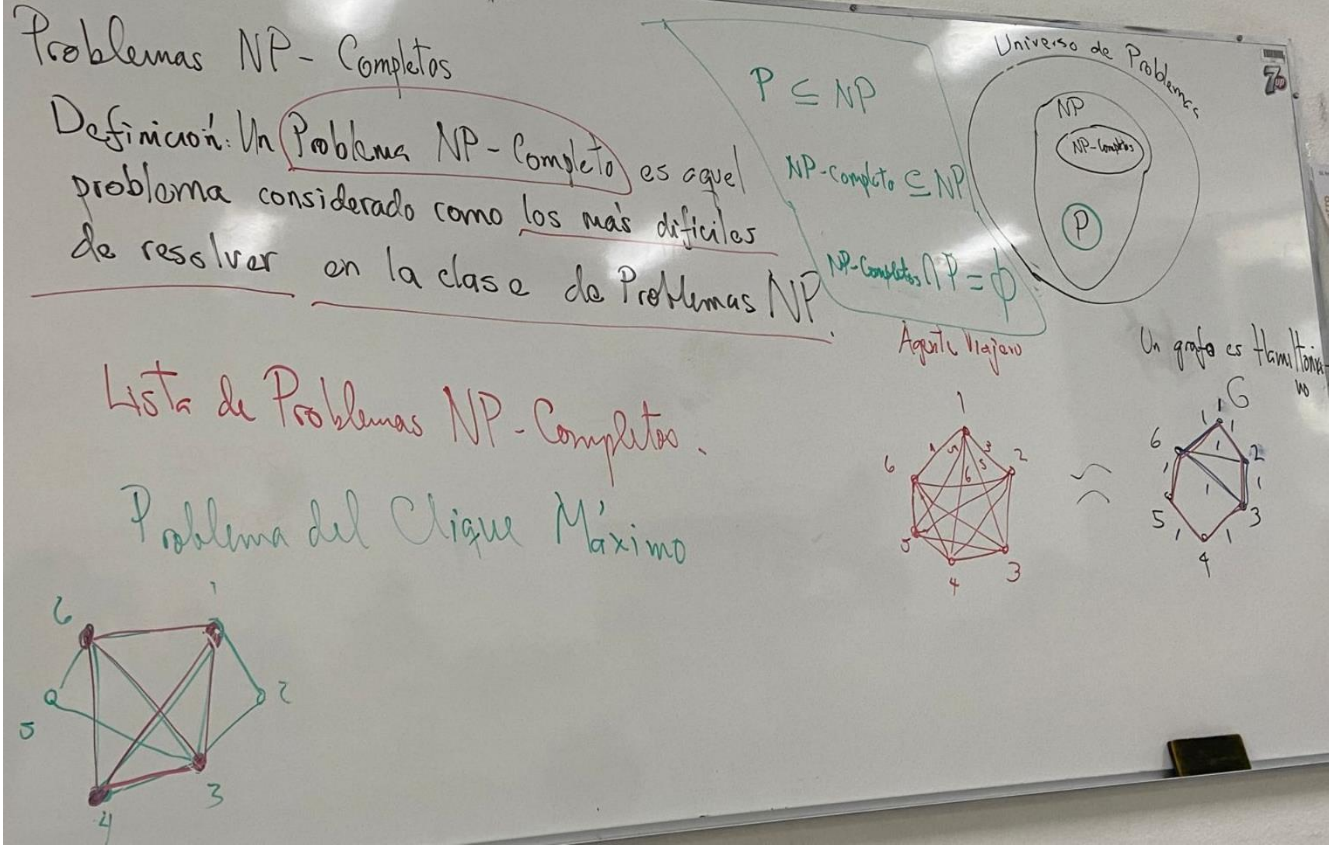
0 1 2 3 4 5 6
g[0 0 1 1 0 0 0]

A={4,10} B={3,8,2}

Leyenda si g[i]=1 U[i] ∈ A
si g[i]=0 U[i] ∈ B

```
Funcion_Sumador A y B(){        g  0 1 2 3 4
  SumaA=0;                                     n=3
  SumaB=0;
  for i=n to 1 {
    if (g[i]==1) sumaA=sumaA+U[i]
    else sumaB=sumaB+U[i]
  }
  if (menor > abs(sumaA-sumaB)) {
    menor ← abs(sumaA-sumaB)
    for i=n to 1 {              //guarda el menor hasta
      gmenor[i] ← g[i]          // el momento
    }
  } // FIN if
} // FIN Funcion_Sumador A y B()
```

```
Pseudocódigo para Max(Σ_i a_i - Σ_i b_i) {
  // Tenemos un problema de Optimización
  Leer_Arreglo_U();
  Ordenar_Quicksort(); // menor a Mayor
  sumaA=0; sumaB=0;
  for i=1 to |U|/2 do {
    sumaA=sumaA+U[i];
    sumaB=sumaB+U[i+|U|/2];
  }
  imprimir("El maximo es", |sumaA-sumaB|);
  Imprimir("se alcanza cuando A contiene a la mitad de los elementos más pequeños de U y B a la mitad de los más grandes, siendo |A|=|B|");
}
```

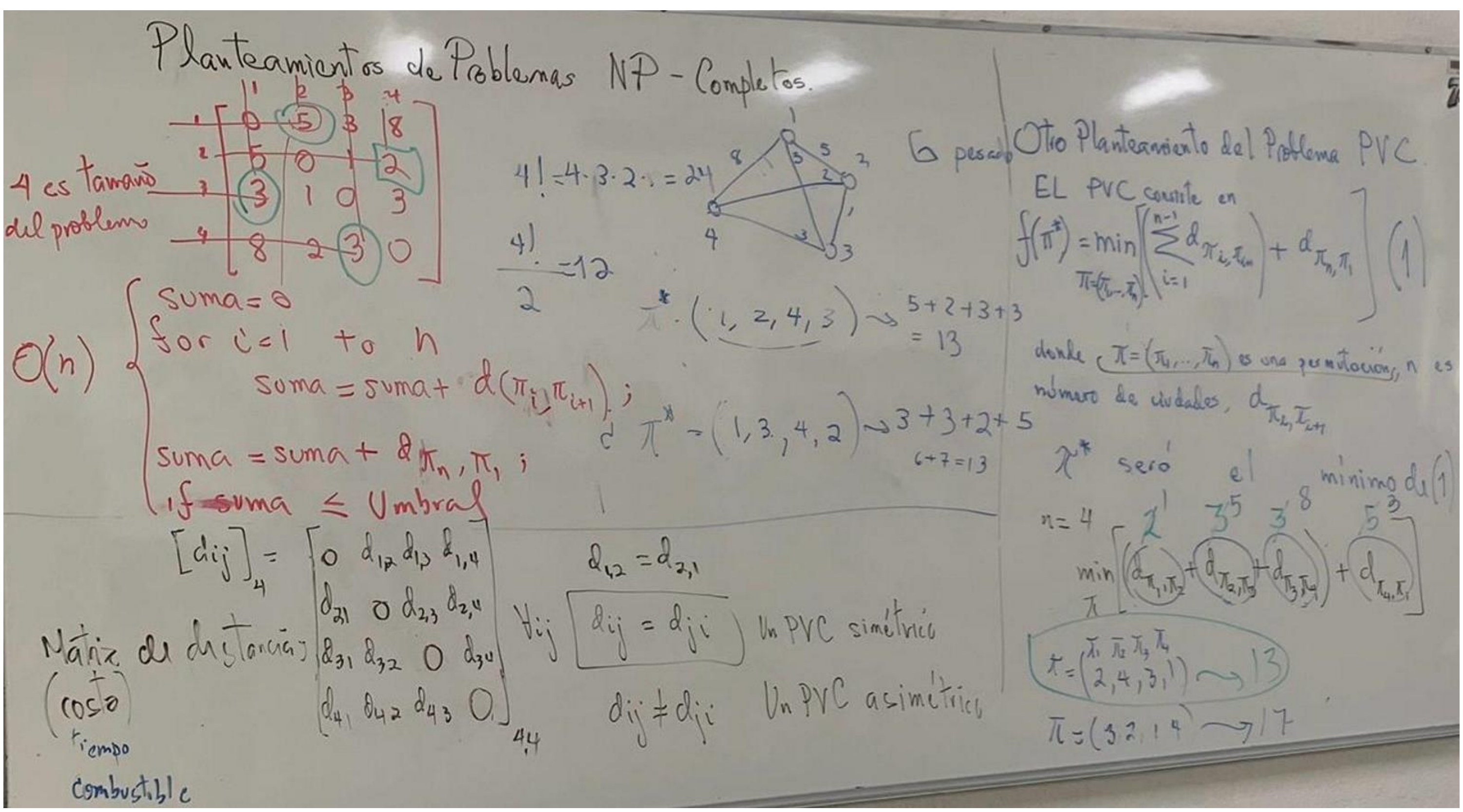


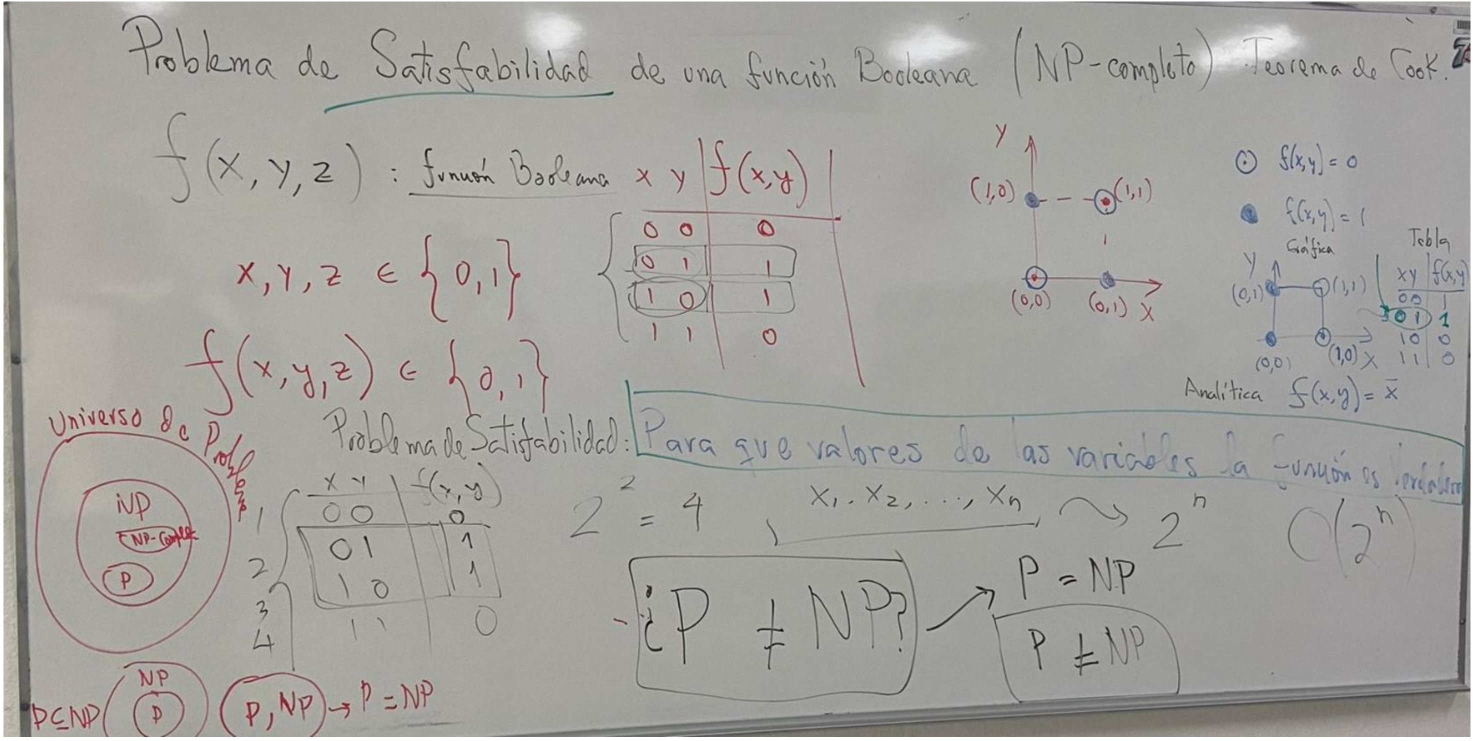
– PLANTEAMIENTOS DE PROBLEMAS NP - COMPLETOS

Problema de, viajante comercio (PVC) o agente viajero: El problema PVC consiste (TSP Traveling Saleman Problem) en encontrar un recorrido cerrado en un grafo pesado de n vértices (ciudades) tal que la suma de las distancias entre las ciudades recorridas sea la mínima. Donde un recorrido cerrado de n ciudades se puede considerar como una permutación cíclica.

- Otro planteamiento del problema PVC

El PVC consiste en:



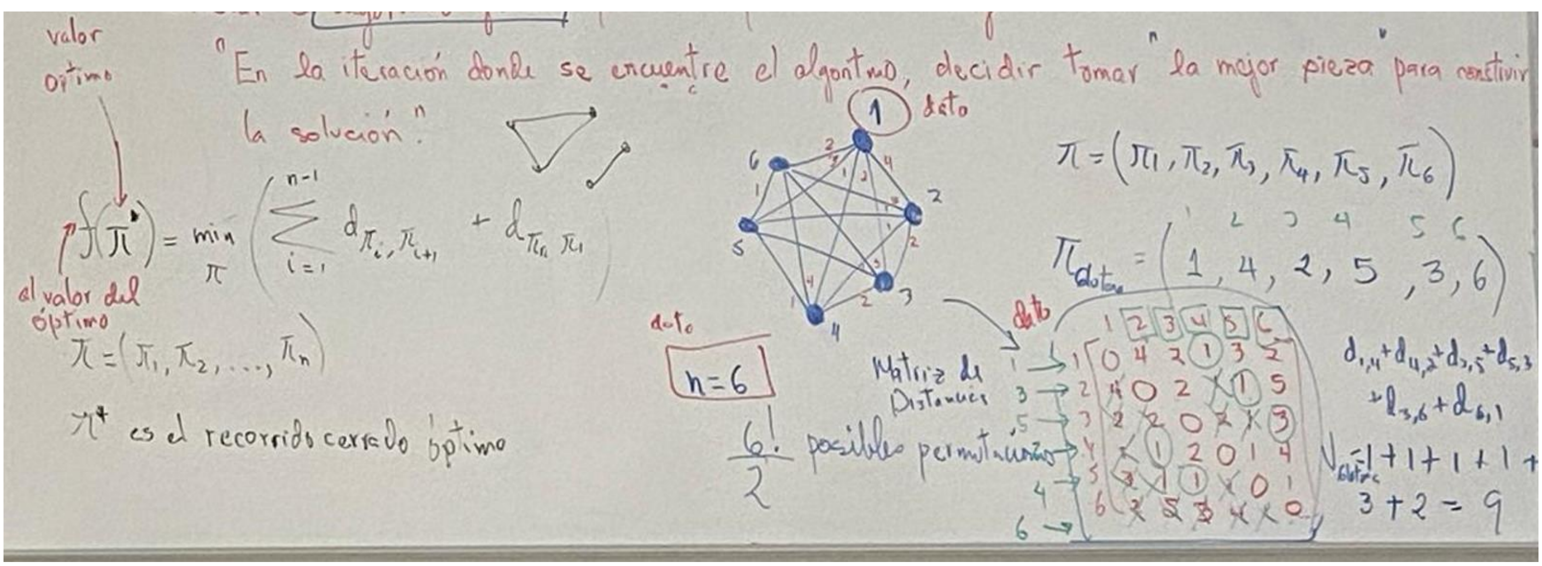


UNIDAD 3

TÉCNICAS PARA EL DISEÑO DE ALGORITMOS PARA RESOLVER UN PROBLEMA DADO

Diseño de un algoritmo glotón (ávido, primero mejor, "greedy").

- Diseñar el algoritmo glotón para el problema del Viajante comercio:
  - "En la iteración donde se encuentre el algoritmo, decidir tomar la mejor pieza para construir la solución"
  - Su complejidad sería P, va recorriendo una matriz

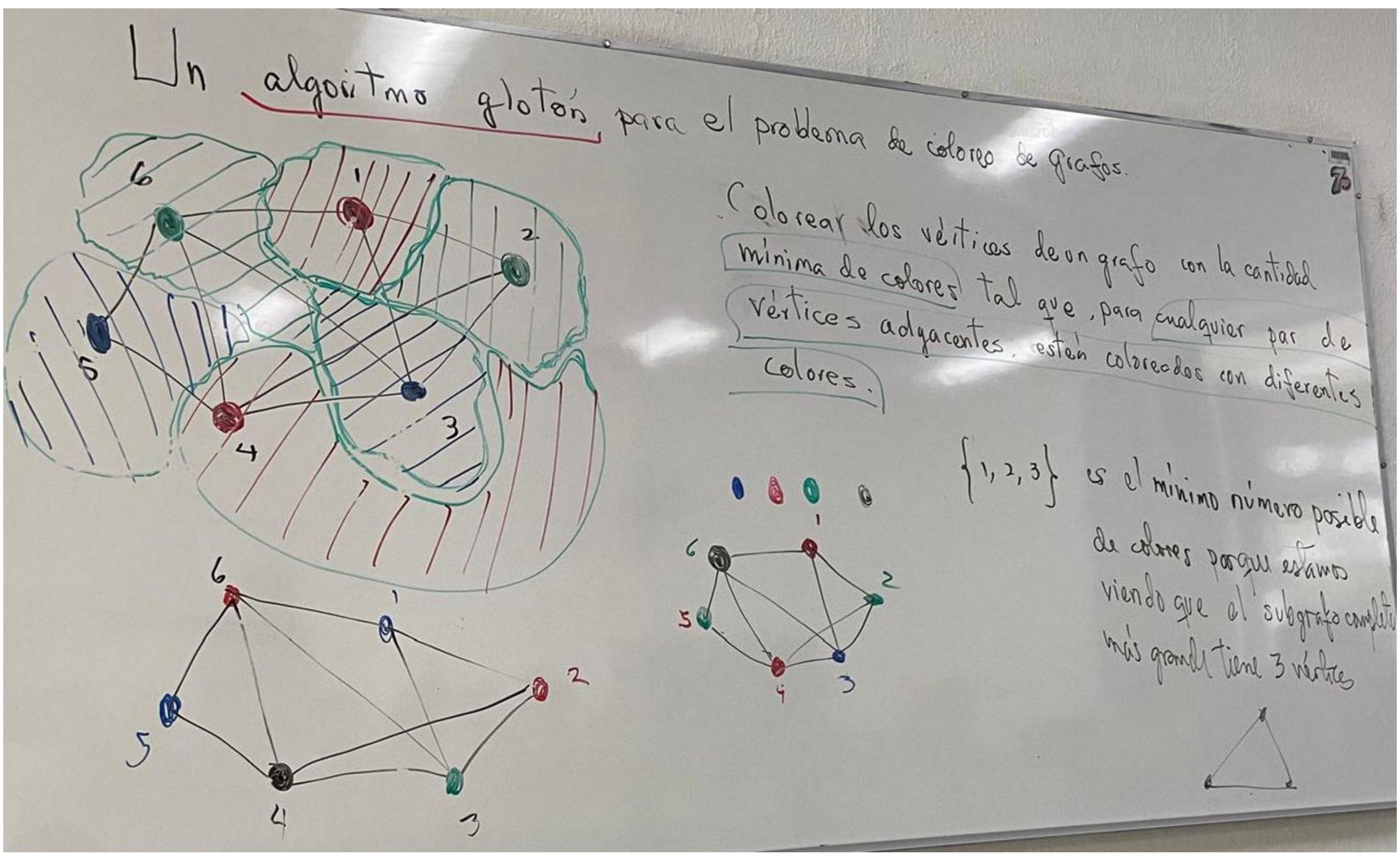


ALGORITMO GLOTÓN:

- Es determinístico
- No genera aleatorios.
- En el coloreo de grafos tratará de minimizar, de utilizar la menor cantidad de colores. En el problema del Viajante de comercio busca ir sumando ciudades de mínimo peso.
- Se usan para tener una sola solución.

Un algoritmo glotón para el problema de coloreo de grafos, tiene aplicaciones como en pintar mapas, trata de que los vértices estén coloreados y sus adyacentes tengan otro color diferente.

- Consiste en colorear los vértices de un grafo con la cantidad mínima de colores tal que para cualquier par de vértices adyacentes estén coloreados con diferentes colores.
- Para este problema no siempre sucede que obtiene la solución óptima.
- Comenzar con el que más vecinos tiene y colorear los que no están relacionados con el con otro color. Pero puede haber muchos criterios por el cual escoger. También se puede seleccionar al principio el que menor relaciones tiene y así colorear los demás no vecinos del mismo color. No sabes cual criterio es mejor para ese grafo.



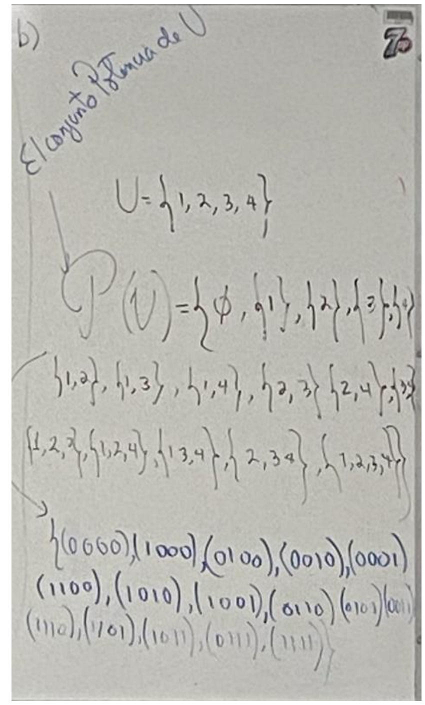
PRACTICA

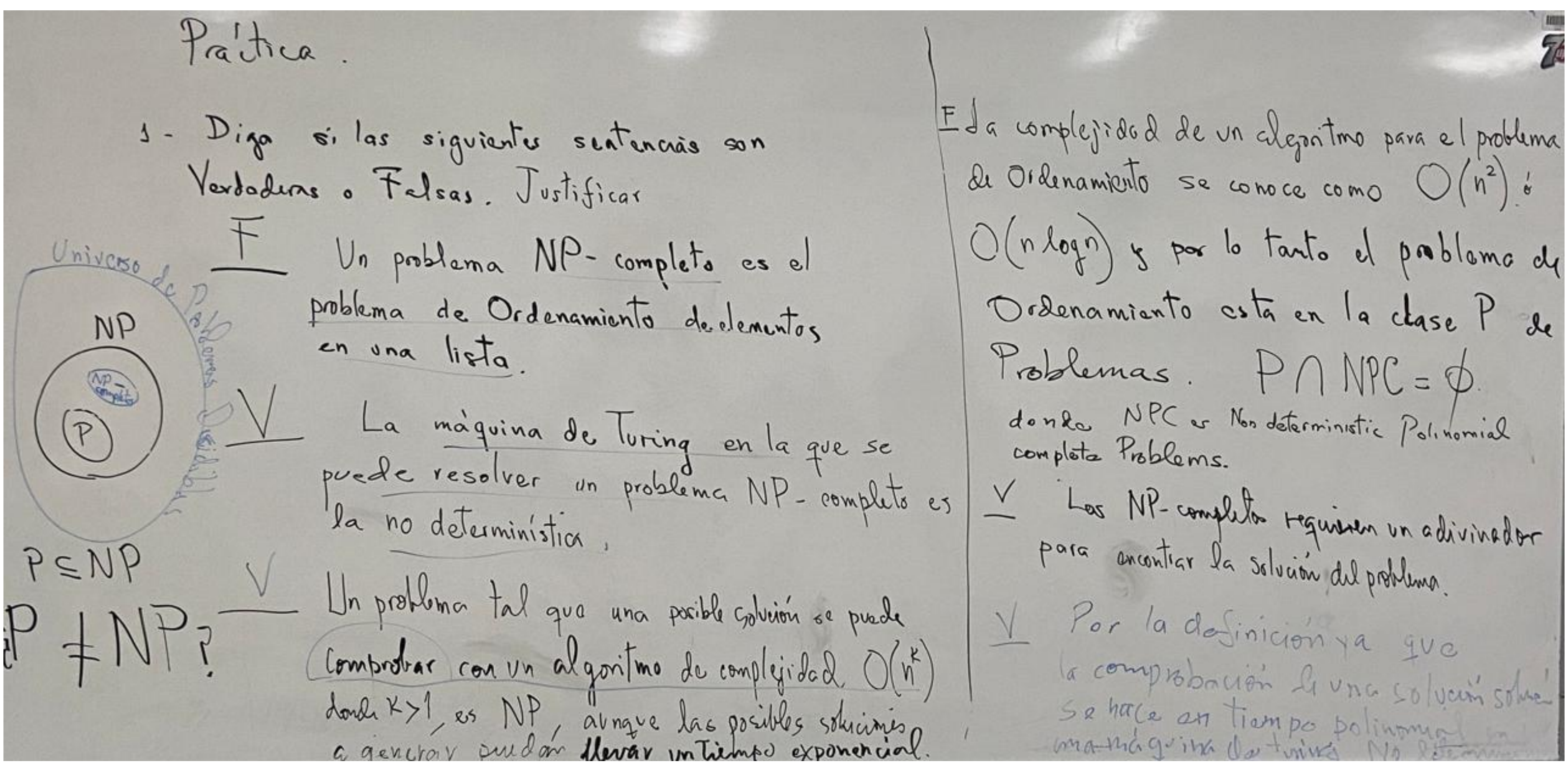
1. Diga si las siguientes sentencias son Verdaderas o Falsas:

   (F) Un problema NP-Completo es el problema de Ordenamiento de elementos en una lista.
   Justificación: la complejidad de un algoritmo para el problema de ordenamiento se conoce como O(n^2) o O(n log n), esto quiere decir que el problema de Ordenamiento está en la clase P de Problemas. En donde NPC es Non deterministic Polinomial complete Problems. Los problemas P no están contenidos en NPC

   (V) La máquina de Turing en la que puede resolver un problema NP-Completo es la no determinística.
   Justificación: Las NP-Completos requieren un adivinador para encontrar la solución del problema.

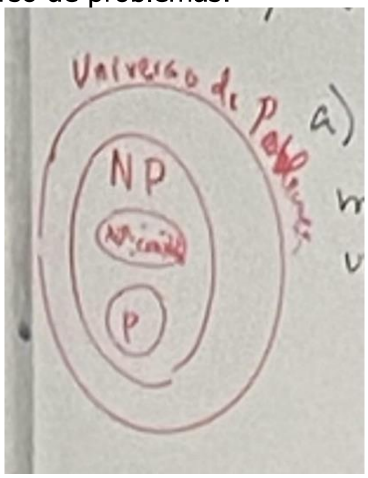
   (V) Un problema tal que una posible solución se puede comprobar con un algoritmo de complejidad O(n^k) donde k>=1, es NP aunque las posibles soluciones a generar podrán llevar un tiempo exponencial.
   Justificación: Por la definición ya que la comprobación de una solución se hace en tiempo polinomial en una máquina de Turing no determinística.





EJERCICIOS

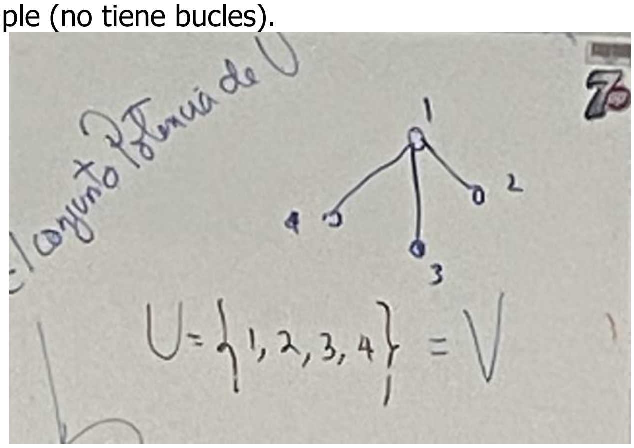
EJERCICIOS. Sobre problemas P, NP y NP-Completos.
Dado los siguientes problemas clasifíquelos según su complejidad y justifique su respuesta.
Tenemos el siguiente universo de problemas:



a. <u>Problema de cubrimiento de aristas por vértices.</u> El problema de encontrar el subconjunto de vértices de un grafo G=(V,E) más pequeño tal que toda arista del conjunto E de G, tenga un extremo en el subconjunto de vértices más pequeño.

Generar todos los subconjuntos de un conjunto dado, en este caso el conjunto potencia de U.

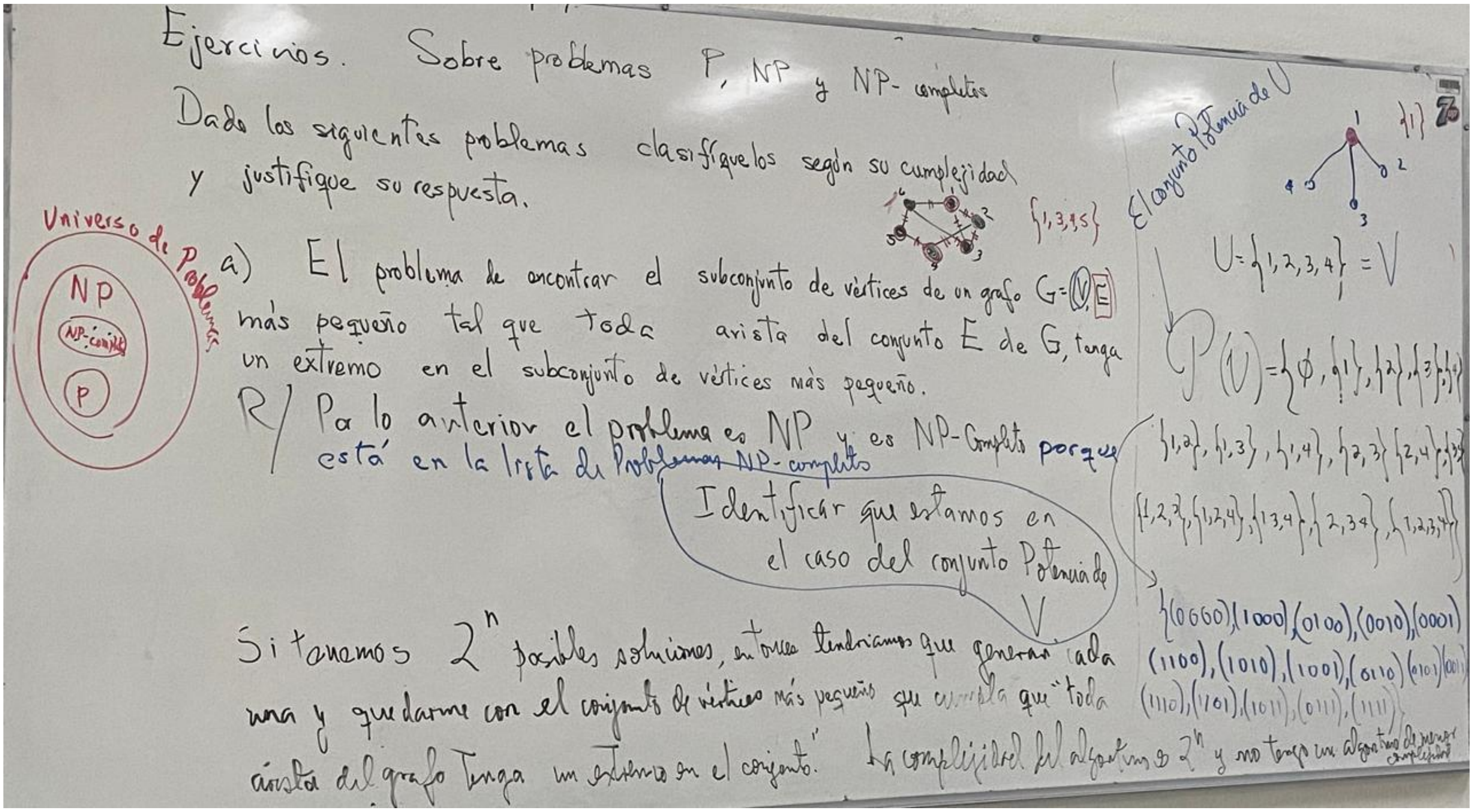
- Si te dicen que el subconjunto más pequeño del grafo es (1) o (1000) entonces todas las aristas conectan con 1 [si no te dicen cuál es el más pequeño siempre será el vacío] y que es un grafo simple (no tiene bucles).

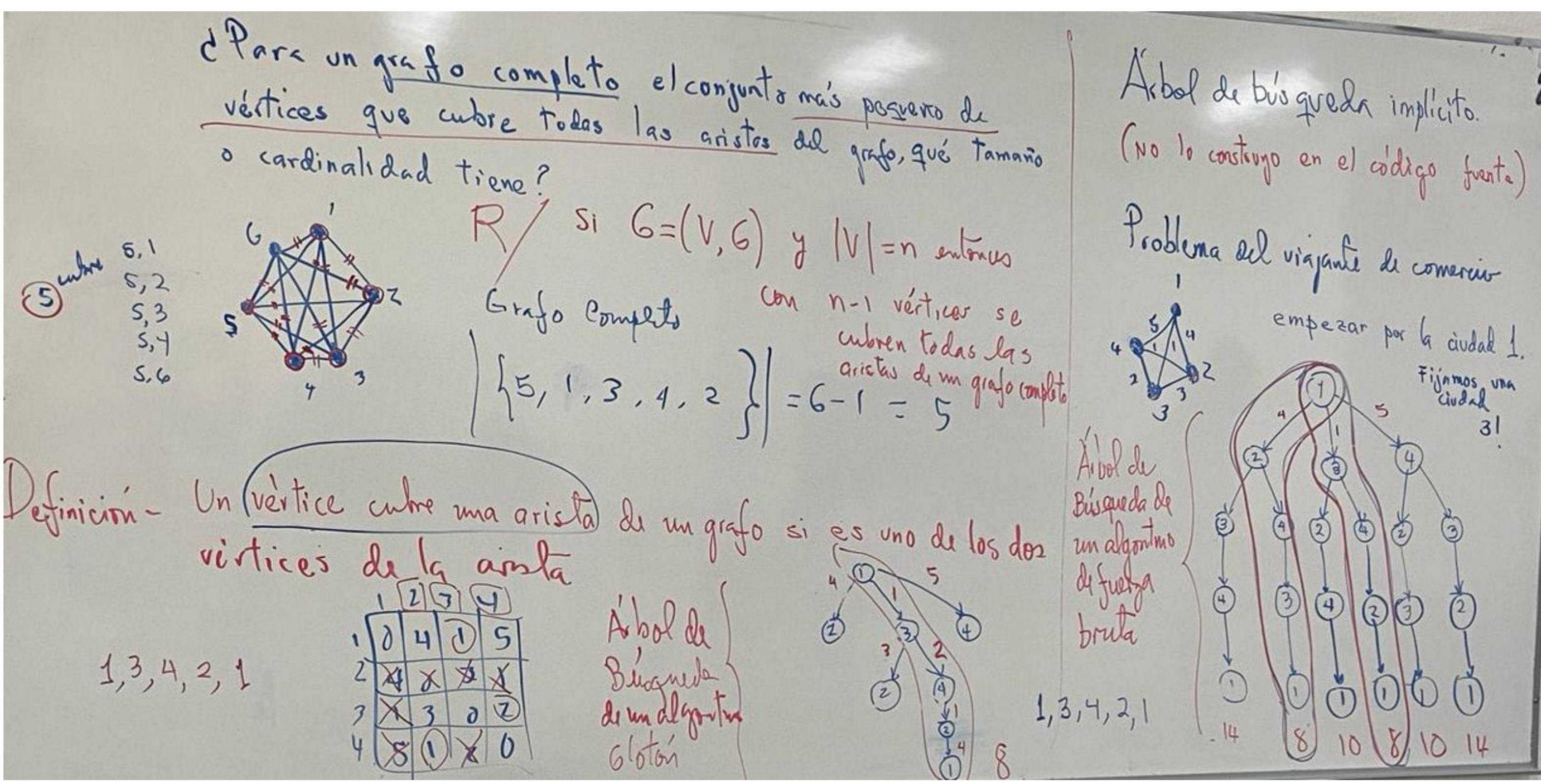


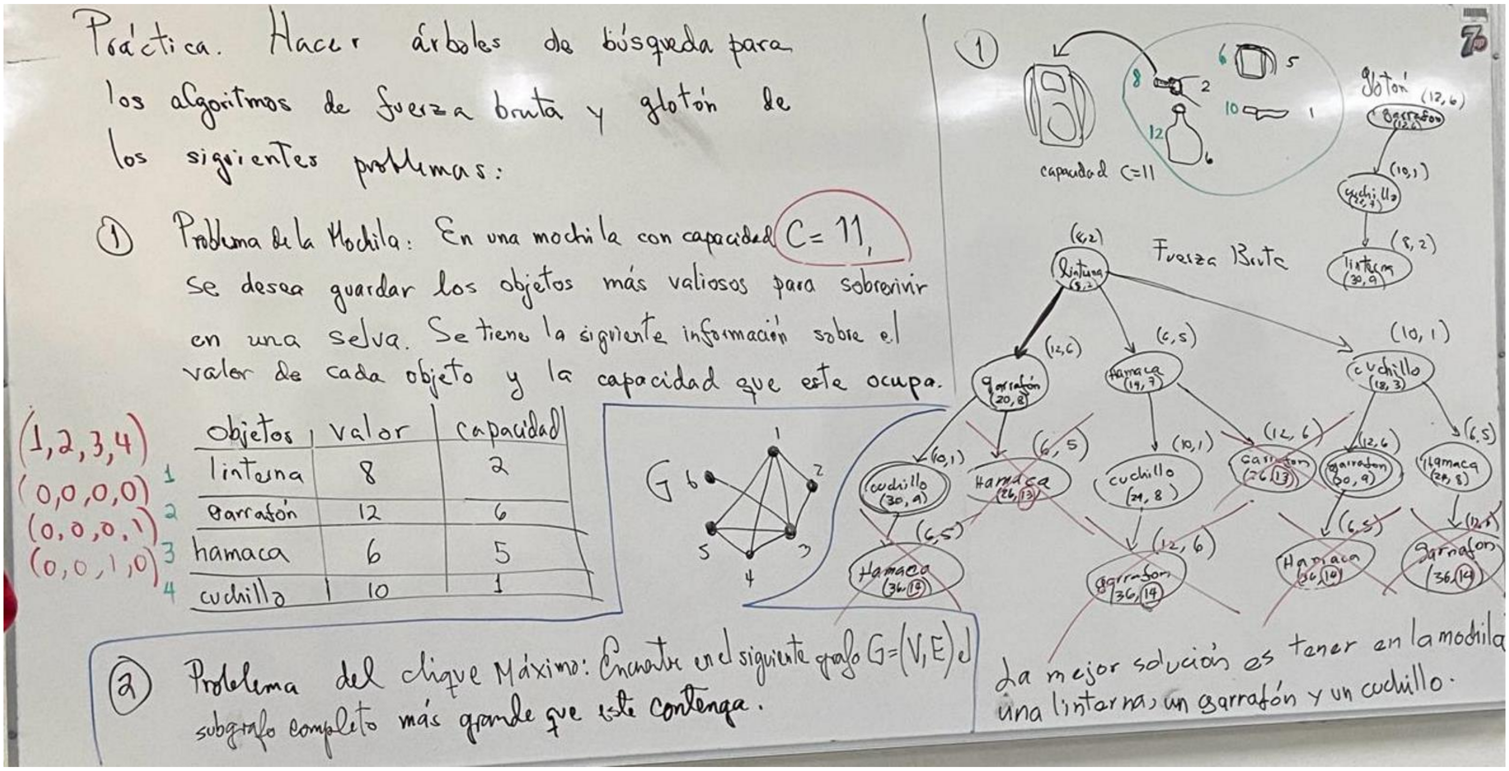
- Identificar que estamos en el caso del conjunto Potencia de V.
- Todas las cadenas posibles diferentes binarias de tamaño n.
- Si tenemos 2^n posibles soluciones entonces tendríamos que generan cada una y quedarme con el conjunto de vértices más pequeño que cumple que "toda arista del grafo tenga un extremo en el conjunto".
- Por lo tanto, la complejidad del algoritmo sería 2^n y no tengo otra forma de menor complejidad para conocer el conjunto más pequeño, o sea, no tengo un algoritmo de menor complejidad conocido.
- Respuesta: Por lo anterior, el problema sería NP y es NP-Completo porque está en la lista de Problemas NP-Completos.

Para que sea NP-Completo debería estar en la lista de problemas de NP-Completos, encontrar el juego de valores para los cuales es verdadera, se debería de pasar al problema de satisfabilidad.

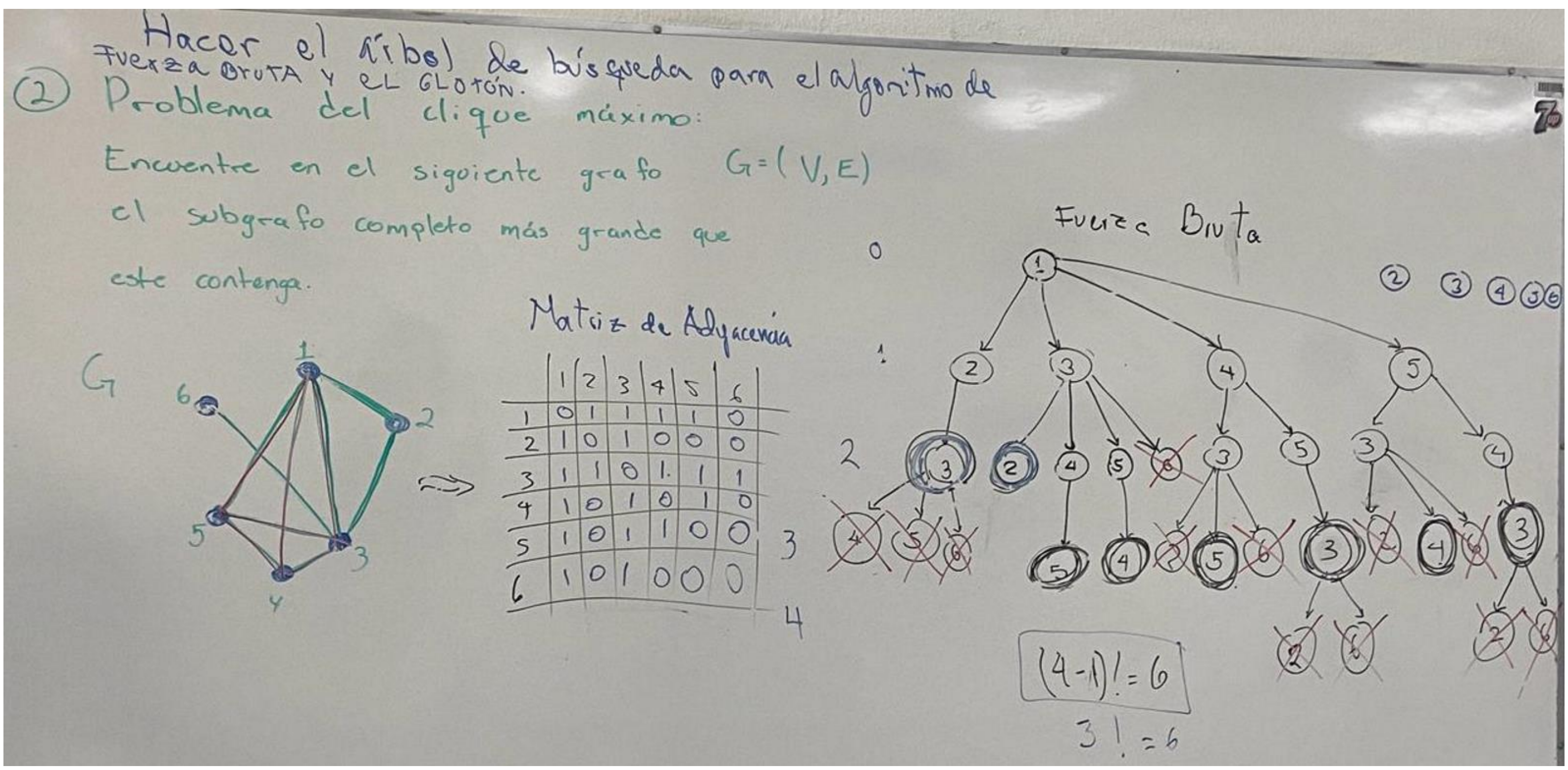
Siempre es pensar en que busca el problema, cuál es su enfoque e identificar en primera instancia en que caso estas, por ejemplo, conjunto Potencia, generando permutaciones a manera de subconjuntos y este sería básicamente NP.







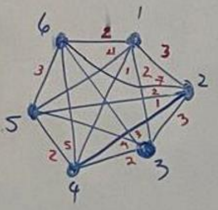
Cliqué: conjunto de vértices que están conectados con todos los demás. Su grado la cantidad de vértices menos uno



Tarea sobre programación de algoritmos de fuerza bruta y algoritmo glotón para los siguientes problemas:

a) Problema del Viajante de comercio (Fijando la primera ciudad)

b)

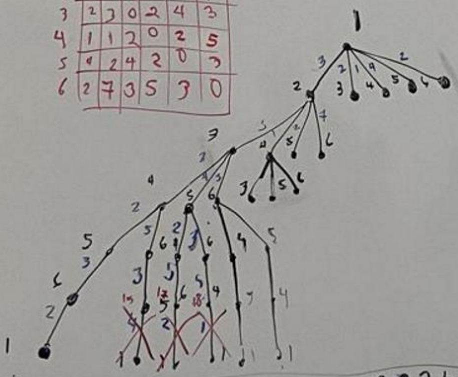


	1	2	3	4	5	6
1	0	3	2	1	4	2
2	3	0	3	1	2	7
3	2	3	0	2	4	3
4	1	1	2	0	2	5
5	4	2	4	2	0	3
6	2	7	3	5	3	0

$6! = 720$

$5! = 120$

Fuerza Bruta (Algoritmo Primero en Profundidad ó en anchura)

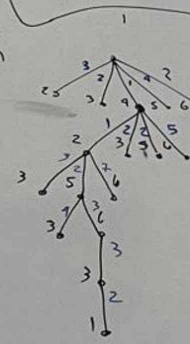


$2 \quad 3 \quad \ldots \quad n$

15

120 permutaciones

Glotón



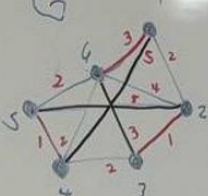
$(1, 4, 2, 5, 6, 3, 1)$

distancia $1+1+2+3+3+2 = 12$

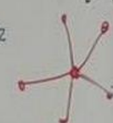
Descripción del Problema del Pareo Perfecto Máximo.

Sea $G=(V,E)$ tal que para toda arista de E se asigna un peso que es un número natural sin el cero $(\mathbb{N} \setminus \{0\})$, donde $|V|$ es un número par. Encontrar el pareo perfecto de peso máximo, entendiéndose por pareo perfecto aquel conjunto de aristas $P \subseteq E$ tal que para cualquier par de aristas e_i, e_j no tengan vértices comunes. Un pareo perfecto de peso máximo es aquel que la suma de los pesos de sus aristas es la más grande posible. Además el conjunto de arista P (Perfecto) debe cubrir todos los vértices del grafo

G

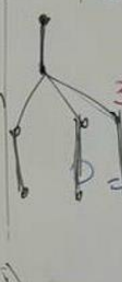


$E = \{\overset{2}{\{1,2\}}, \overset{5}{\{1,4\}}, \overset{3}{\{1,6\}}, \overset{1}{\{2,3\}}, \overset{8}{\{2,5\}}, \{2,6\}, \overset{2}{\{3,4\}}, \overset{3}{\{3,6\}}, \overset{1}{\{4,5\}}, \overset{2}{\{4,6\}}, \overset{2}{\{5,6\}}\}$



$P = \{\{1,6\}, \{2,3\}, \{4,5\}\}$

$3+1+1 = 5$



$P = \{\{2,5\}, \{1,4\}, \{3,6\}\}$

$8 + 5 + 3 = 16$

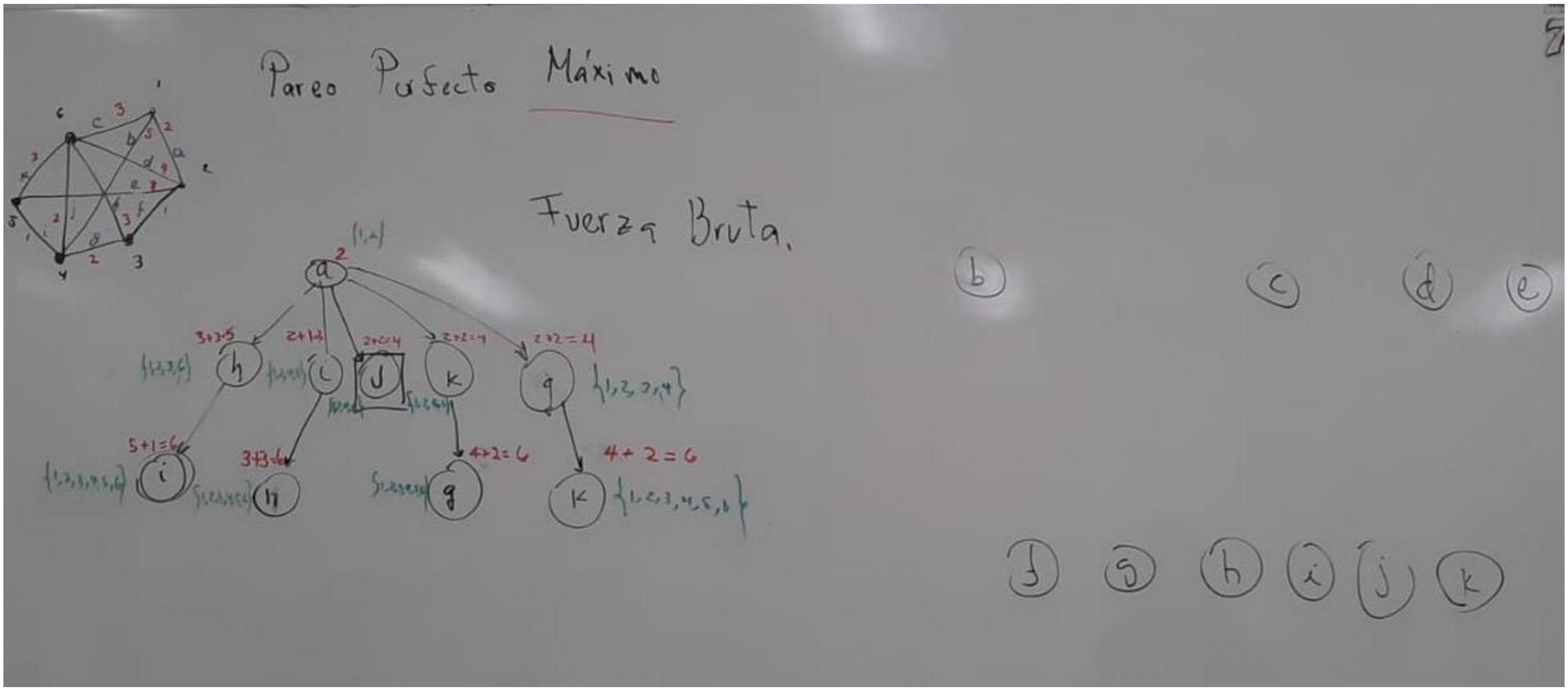
$\frac{|V|}{2}$ aristas en el Pareo.

Alg. Glotón

Paso 1. Elijo una arista con el peso mayor.

Paso 2. Elegir la siguiente arista tal que no tengan vértices en común con las anteriores y que sea la siguiente peso mayor posible.

Paso 3. Ir al Paso 2 hasta que tengamos



Pareo Perfecto Máximo
Fuerza Bruta.
2+2=4
2+2=4
5+1=6
4+2=6
4 + 2 = 6
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6}