UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Tarea 08:Algoritmos que involucran Gráficas (DFS/BFS/Topological Sorting)

ALUMNO:

Castañon Maldonado Carlos Emilio

PROFESORA

María de Luz Gasca Soto

AYUDANTES

Brenda Margarita Becerra Ruíz
Enrique Ehecatl Hernández Ferreiro (Link)

ASIGNATURA

Análisis de Algoritmos

1 El diámetro de una gráfica G, $G = (V, A)$, se define como la mayor de las longitudes de las rutas más cortas entre todo par de nodos en la gráfica, es decir:

$$diam(G) = max\{d(x, w) : x, w \in V(G)\}$$

donde $d(x, w)$ es la longitud de la ruta más corta entre los vértices x y w.

Problema A: Determinar el diámetro de un árbol.

a) Diseñar un algoritmo de orden $O(|V|)$ que solucione el **Problema A** y que presente una pareja de vértices cuya ruta más corta sea exactamente el diámetro del árbol.

Antes de empezar, recordemos que sea una gráfica G no dirigida, con $G = (V, A)$ y sean $u, v \in V$, se define a la distancia $d(u, v)$ como la mínima cantidad de aristas entre u y v, en otras palabras la ruta mas corta entre u y v.

Algoritmo para obtener el diámetro de un árbol

Pre Condiciones: La gráfica de entrada es exclusivamente la gráfica de un árbol, la gráfica es no vacía, finita y tiene como mínimo tres nodos (vértices), el nodo raíz y dos nodos hijos de este, la identidad de cada nodo y su relación con los demás nodos, es dada por el usuario.

Como parte de nuestras pre condiciones tendremos que ya disponemos de los algoritmos funcionales de **BFS**, **DFS** y el de **obtener la distancia entre dos vértices**, este ultimo es bastante trivial para este algoritmo ya que como estamos trabajando con arboles, bastara que al encontrar nuestros vértices x y w, solo deberemos pararnos en cada vértice y preguntar quien es nuestro padre, irnos a ese vértice padre y volver a preguntar quien es su padre hasta que lleguemos al vértice raíz, una vez que hayamos llegado paralelamente con cada vértice al raíz (y guardado por supuesto el recorrido hecho para llegar a el con cada vértice), solo debemos unir los caminos que han usado ambos vértices para obtener la distancia entre los dos, además, observemos como es que con esto nos evitamos comparaciones innecesarias o el uso de otros algoritmos y nos aseguramos de que sea la mas corta por que si solo estamos yendo para arriba en nuestro árbol, inevitablemente ***todos los caminos llevan a roma***, en este caso, al vértice raíz.

Post Condiciones: El resultado es el diámetro de el árbol ingresado.

Nota: El algoritmo esta hecho para funcionar en un árbol (el cual en esencia, es una gráfica con características especificas), es por esto que al referirnos a la "gráfica", en realidad nos estamos refiriendo a la gráfica del árbol con el que estamos trabajando y no a una gráfica arbitraria.

Proceso:

1 Realizamos **DFS** en nuestra gráfica G desde el nodo raíz del árbol para localizar el nodo (vértice) x mas lejano a la raíz, en caso de que mas de un nodo cumpla esta condición, x será aquel primer nodo encontrado.

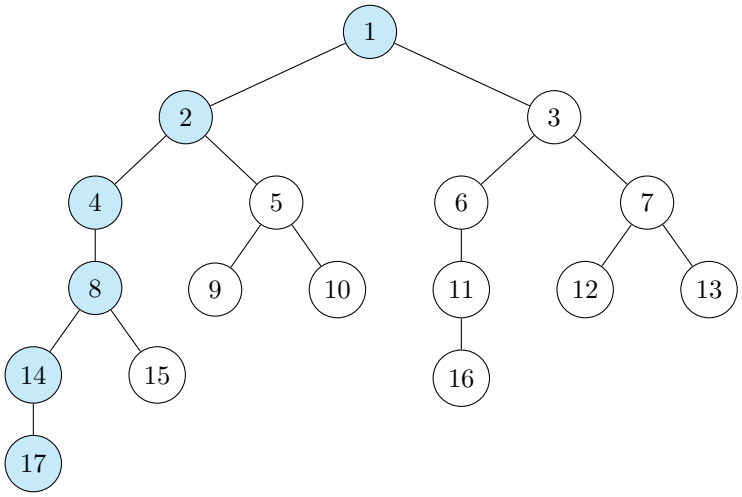
2 Ahora, realizamos **BFS** en nuestra gráfica G desde el nodo x (localizado anteriormente en el paso **1**) y buscamos el nodo w mas lejano a x, en caso de encontrarse mas de un vértice que cumpla la condición, w será aquel primer nodo encontrado en el sub arbol contrario al del vértice x, esto quiere decir que si x esta contenido en por ejemplo, en alguna de las hojas de el hijo izquierdo del nodo raíz, w será el primer nodo mas lejano a x que encontremos en alguna de las hojas del hijo extremo derecho del nodo raíz .

3 Regresamos **la distancia** entre x y w, lo cual será el diámetro de el árbol ingresado.

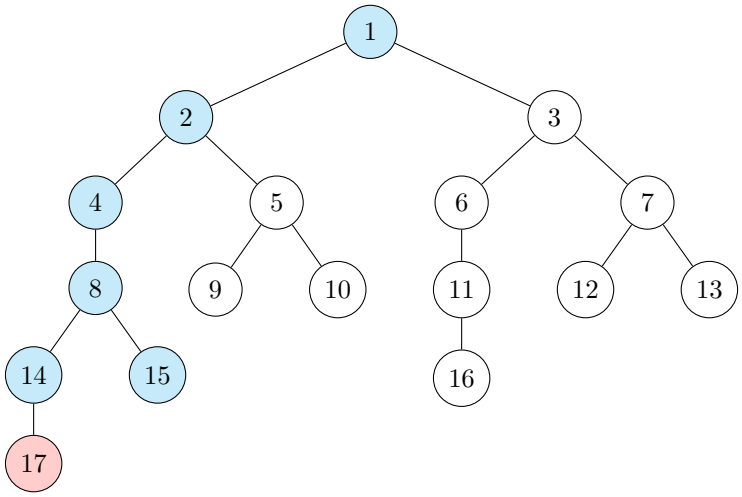
b) Presentar una gráfica G de al menos 17 vértices y aplicar, con detalle, el algoritmo diseñado en **a)**.

Ejecutamos el paso **1**.

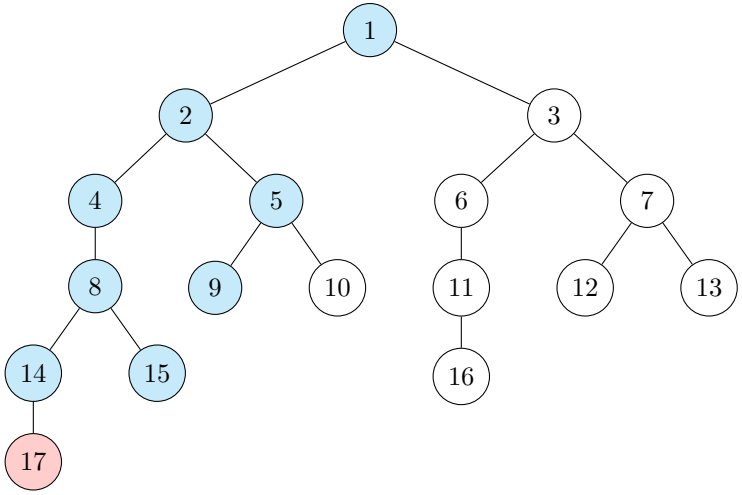
a)



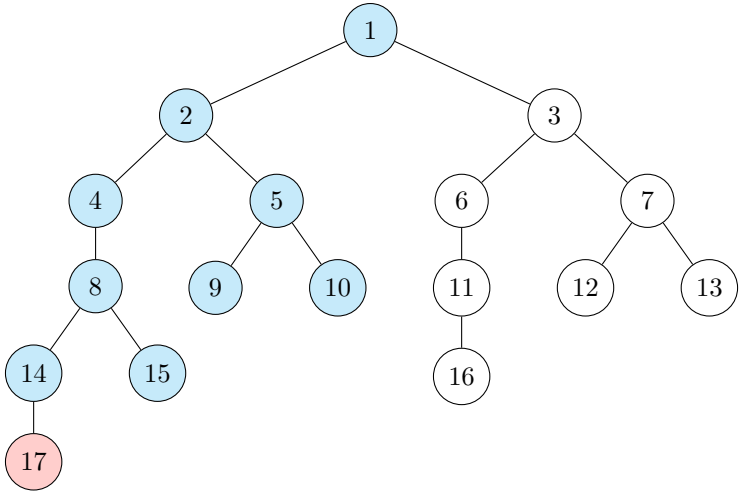
b)



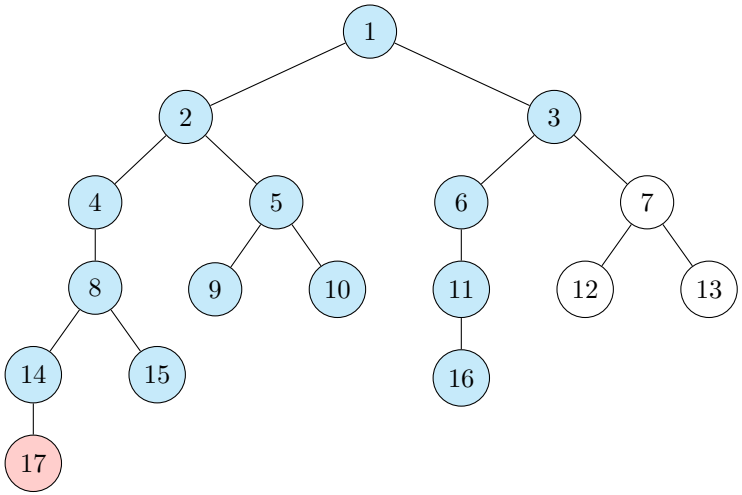
c)



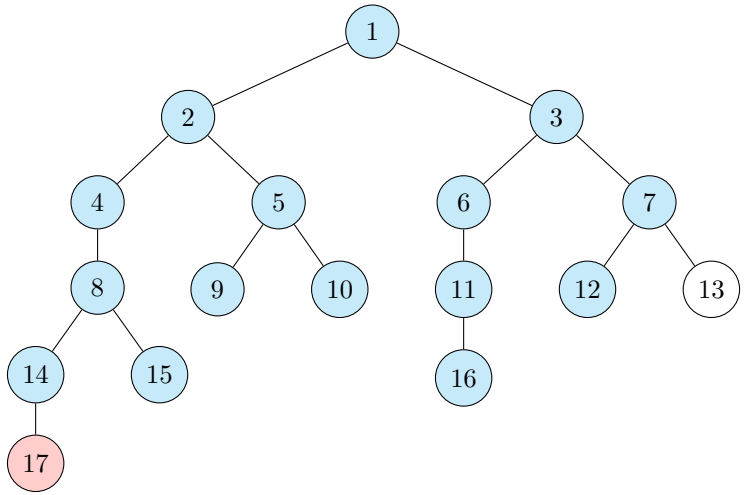
d)



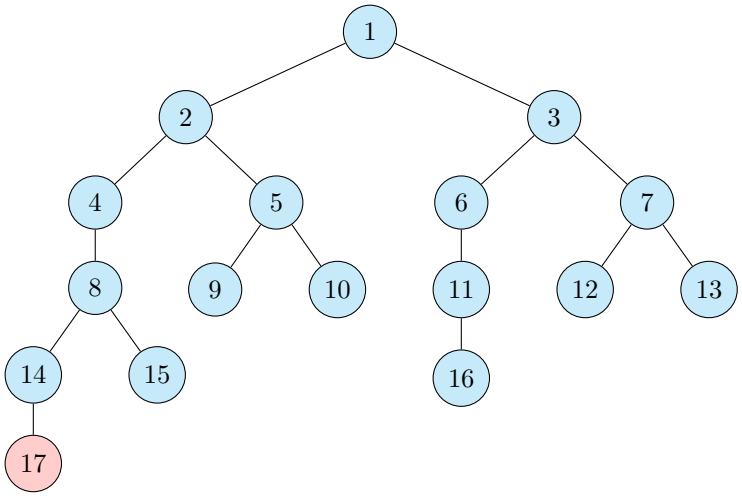
e)



f)



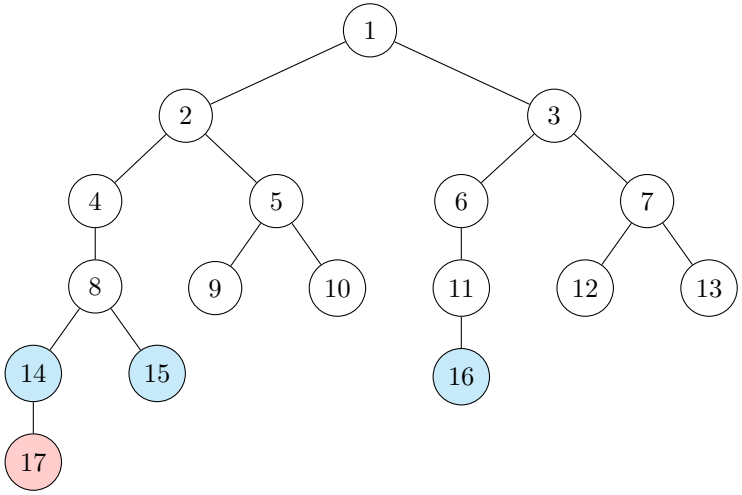
g)



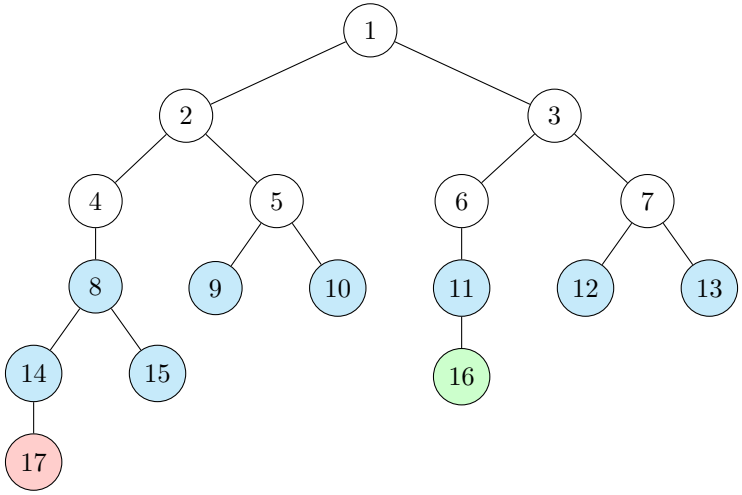
Con esto hemos finalizado el paso **1**, notemos como desde el inciso **a)** hemos encontrado al vértice x mas lejano a la raíz (el cual es el 17) y lo hemos marcado de otro color, notemos como es que en caso de haber encontrado a algun otro vértice mas lejano que el 17, a este lo habríamos marcado de rojo para poder diferenciarlo de los demás.

Ejecutamos el paso **2**

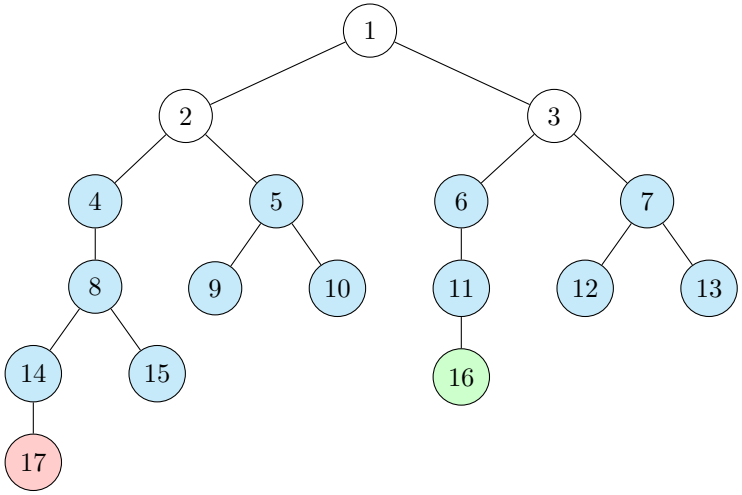
a)



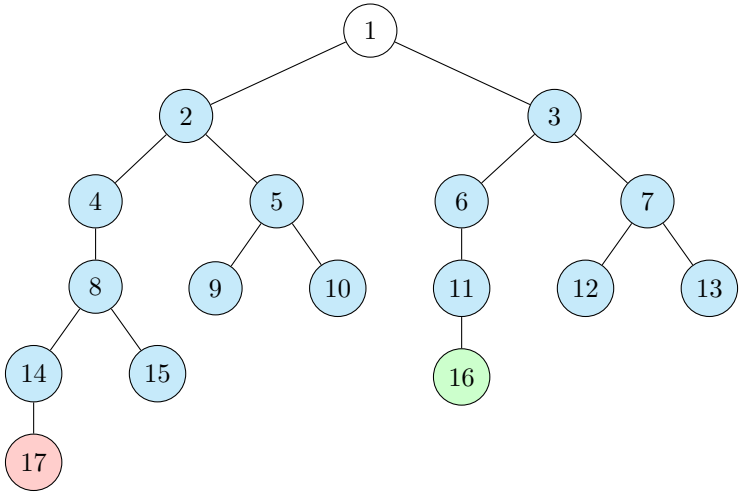
b)



c)



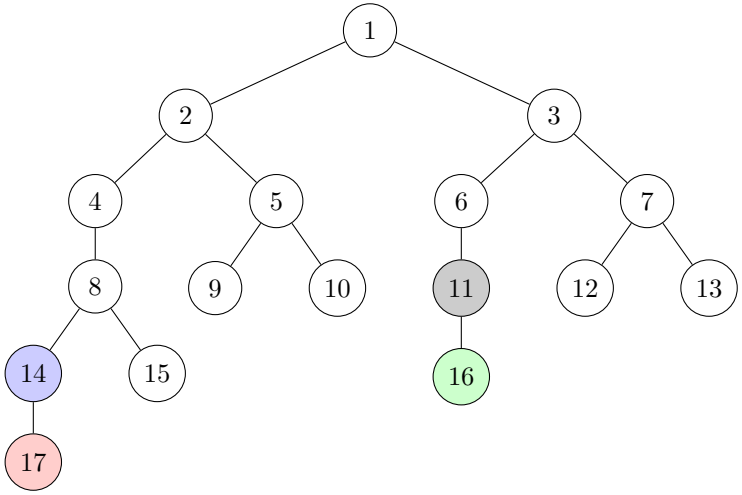
d)



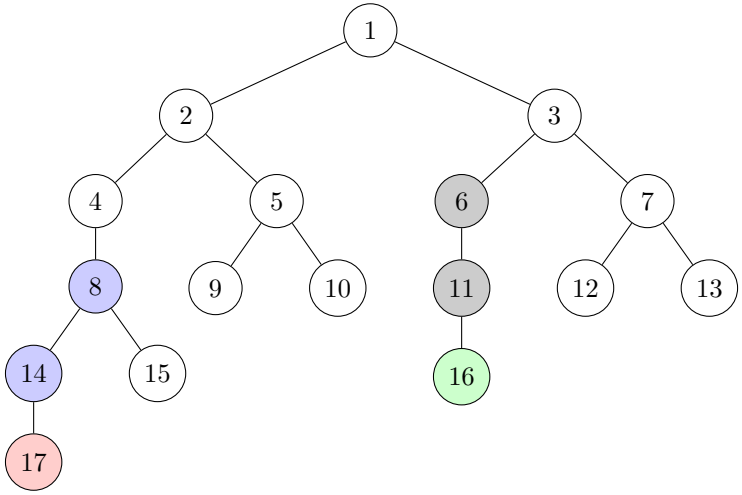
Despues de **d)** seguiríamos con **e)** pero eso es trivial ya que solo sería para colorear el vertice raíz, ahora, notemos como es que encontramos el vertice w que buscabamos, este será el vertice 16.

Ejecutamos el paso **3**

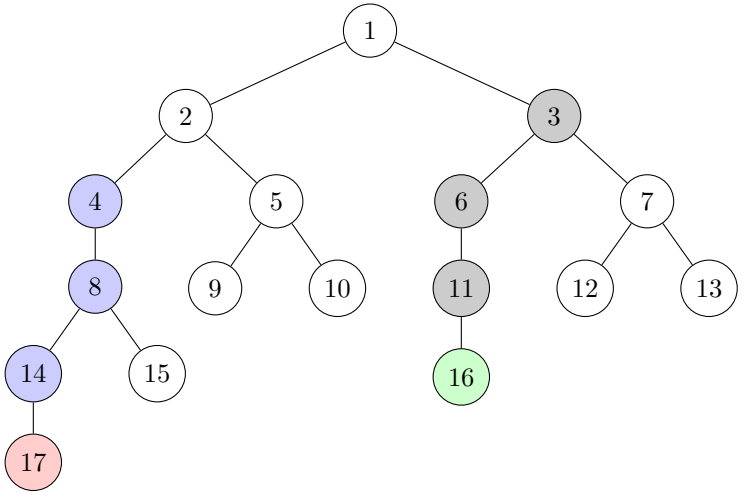
a)



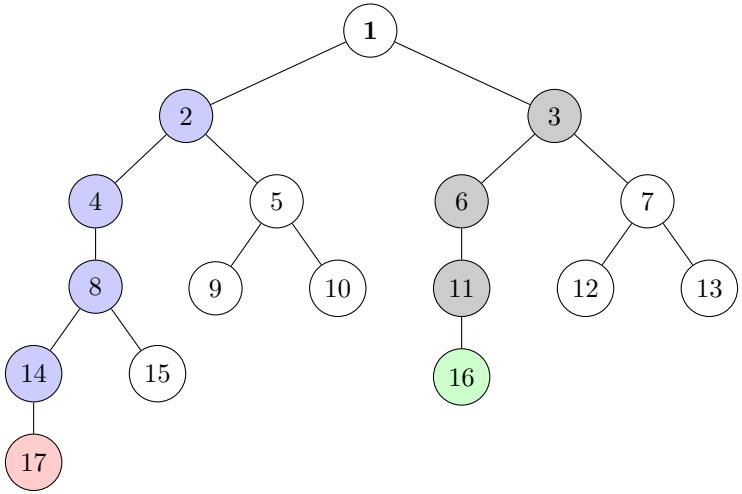
b)



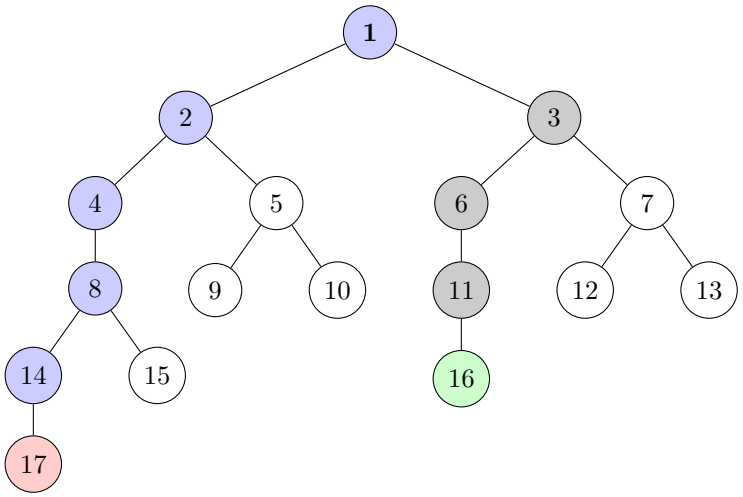
c)



d)



e)



Con esto podemos observar que hemos llegado al vértice raíz (1) desde nuestros vértices x (17) y w (16), y a su vez hemos trazado los caminos que han usado ambos vértices para poder haber llegado hasta la raíz, ahora solo debemos regresar la distancia que separa a estos vértices usando este camino, la cual será de 9.

$\therefore$ El diámetro del árbol es de 9.

2 Considerar los algoritmos **BFS** y **DFS**, modificar uno de ellos para que acepte como entrada una gráfica no conexa **D**.

a) Resolver uno de los siguientes problemas:
Problema B: Determinar el bosque generador de una gráfica disconexa D.

Problema C: Determinar las componentes conexas de la gráfica disconexa D.

b) Diseñar un algoritmo que solucione el problema elegido.

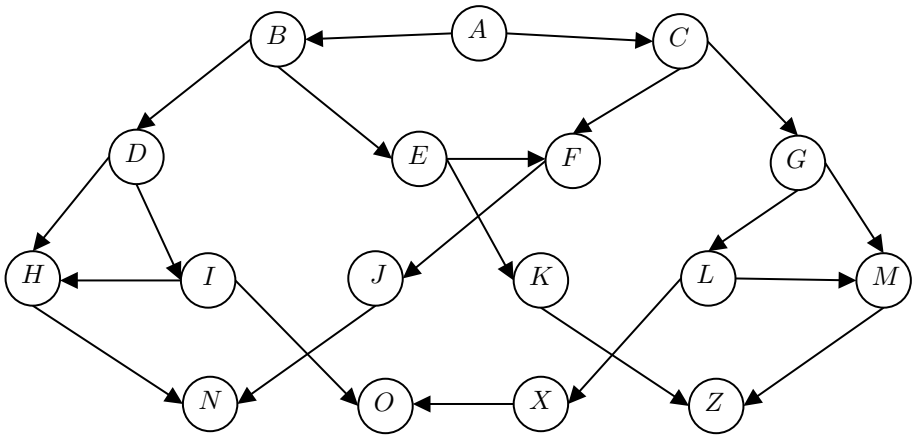
c) Determinar la complejidad del algoritmo propuesto.

d) Construir una gráfica G **disconexa** de al menos 17 vértices, 27 aristas y 4 componentes conexas y aplicar el algoritmo propuesto.

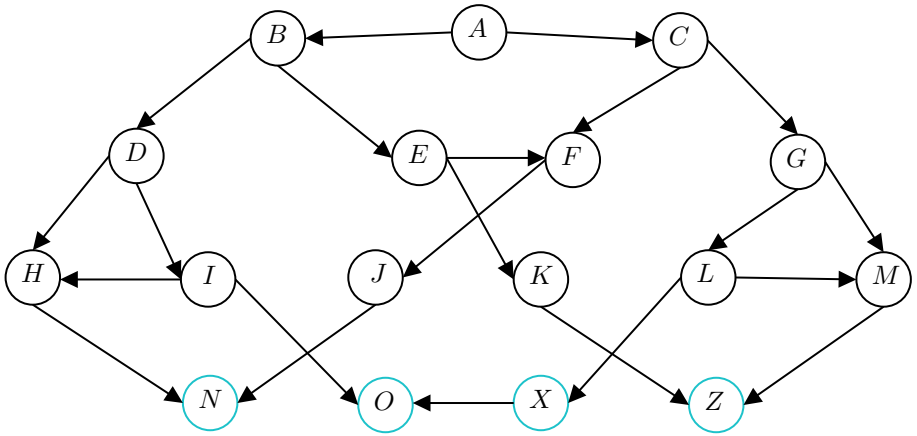
3 **Topological-Sorting (T-S)**

a) Construir una gráfica G de al menos 17 vértices y 21 aristas donde sí pueda aplicarse Topological-Sorting. Aplicar con detalle **T-S** sobre G.

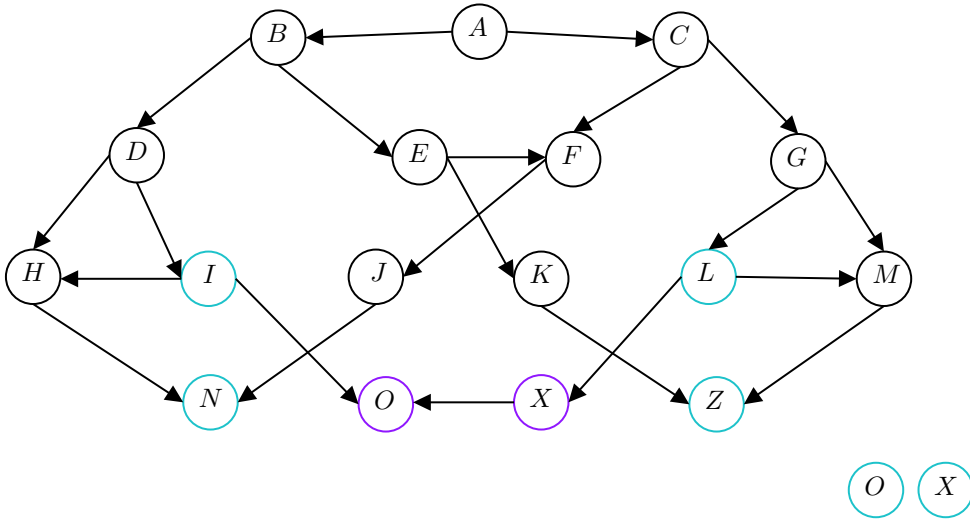
La grafica sobre la que vamos a trabajar es:



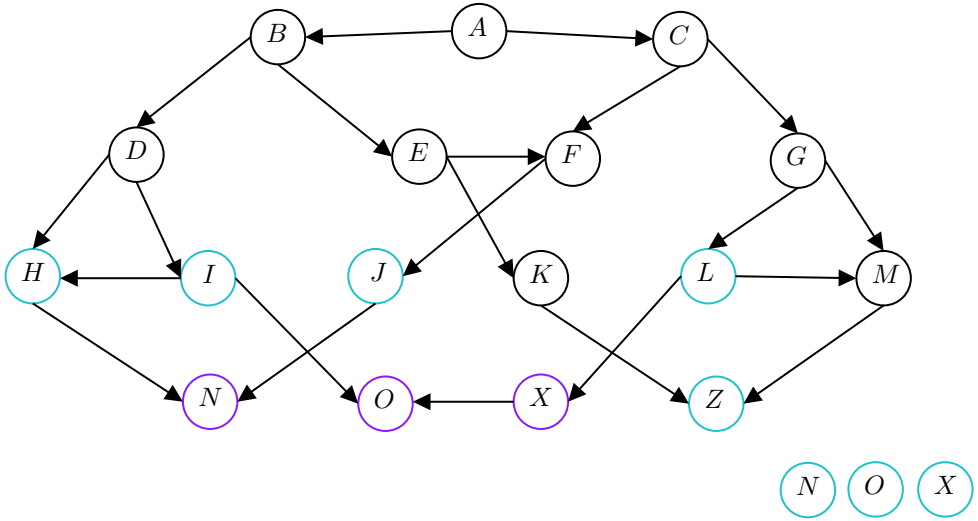
a)


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
X
Z

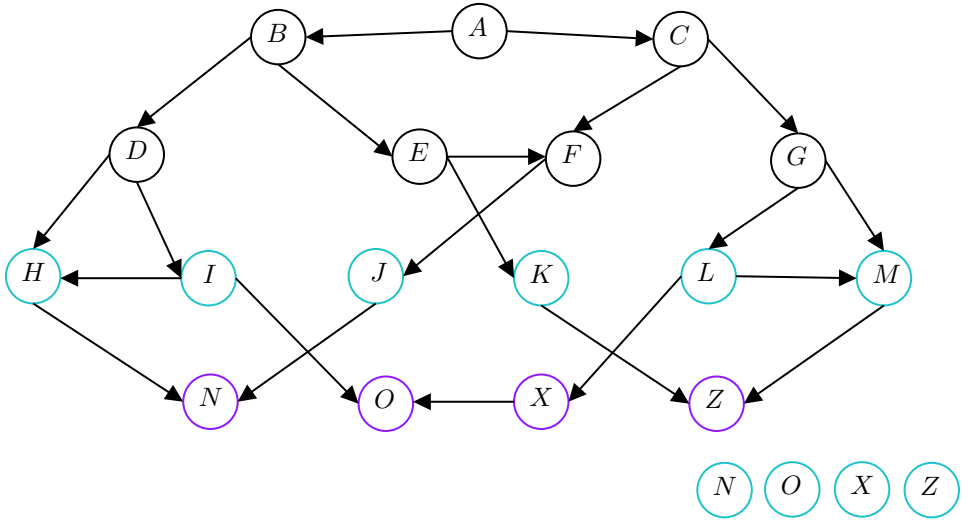
b)


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
X
Z
O
X

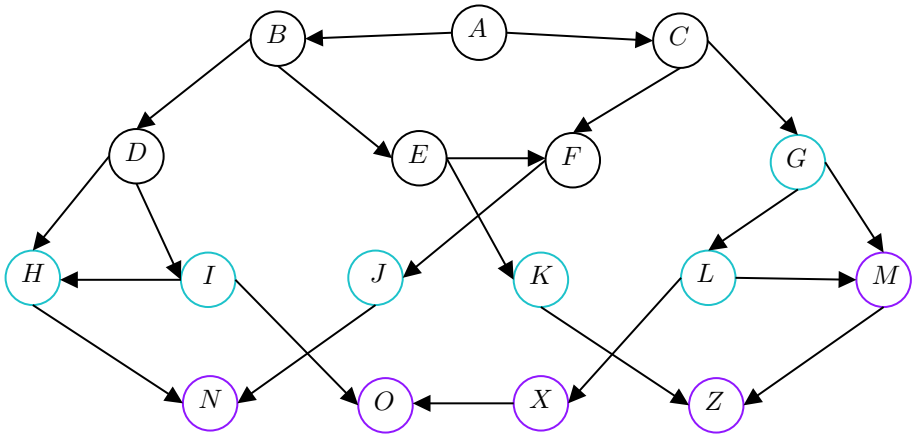
c)


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
X
Z
N
O
X

d)

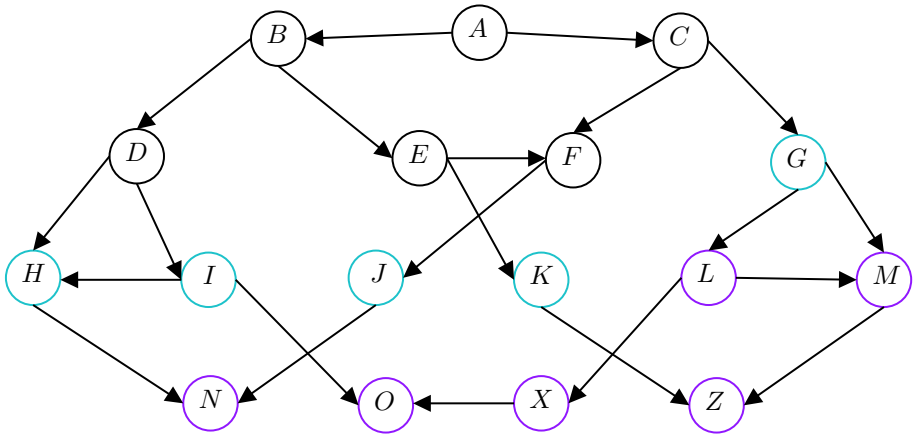

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
X
Z
N
O
X
Z

e)



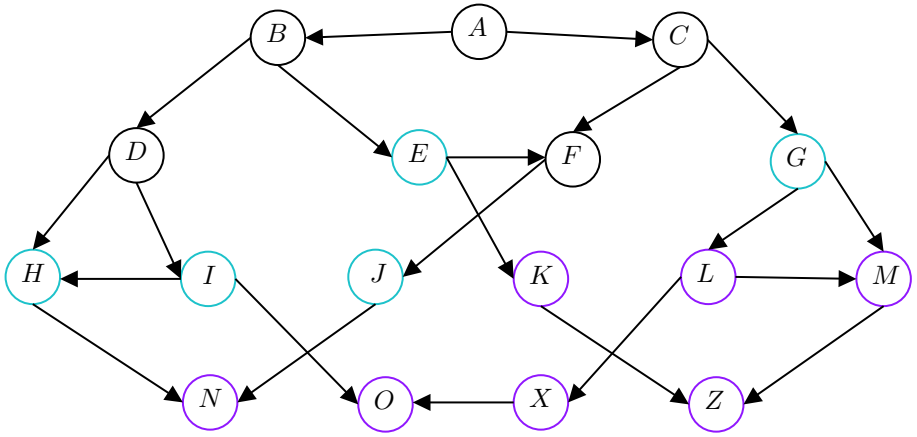
M N O X Z

f)



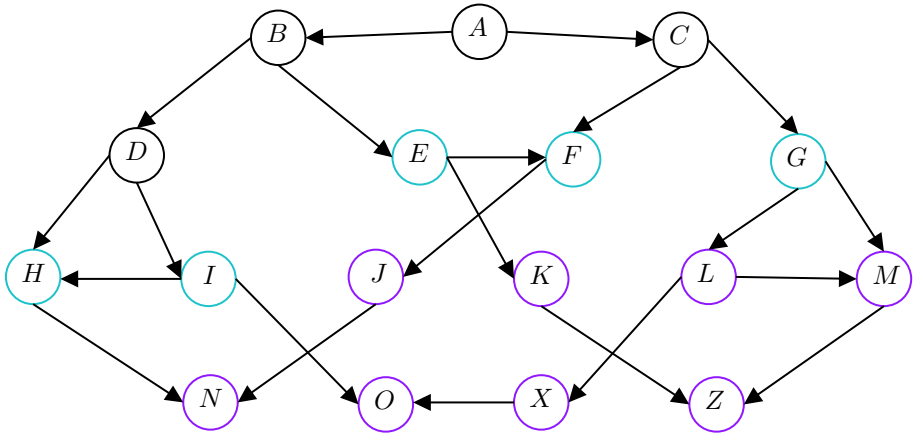
L M N O X Z

g)



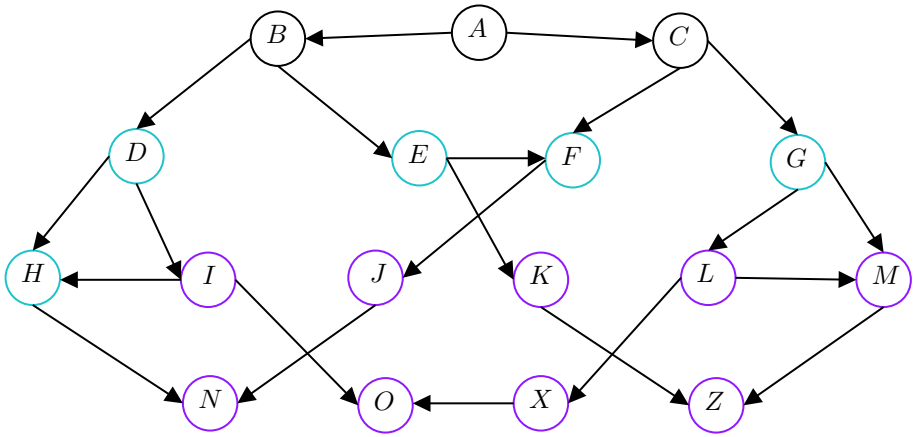
K L M N O X Z

h)



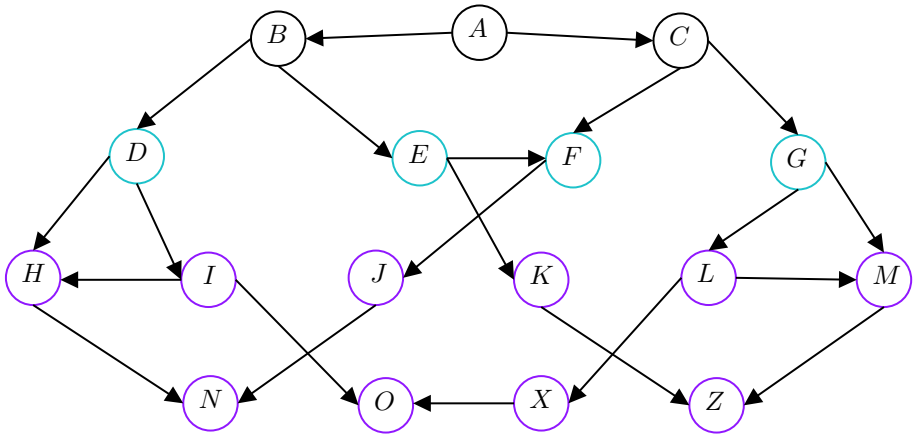
J K L M N O X Z

i)



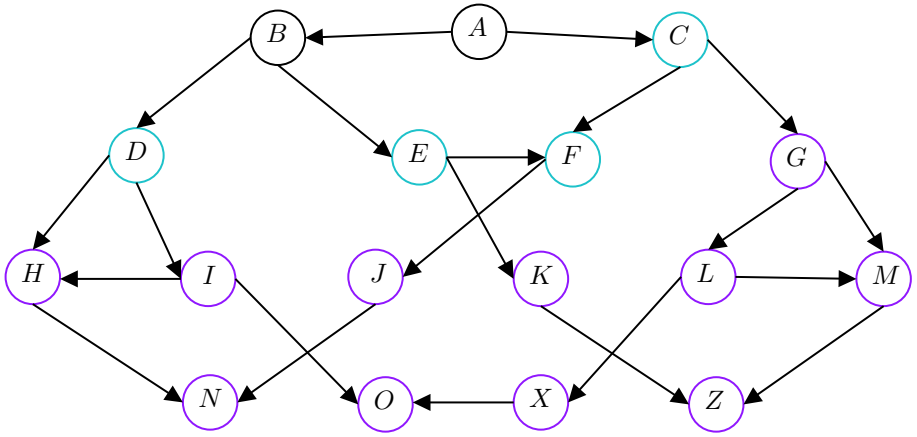
I J K L M N O X Z

j)



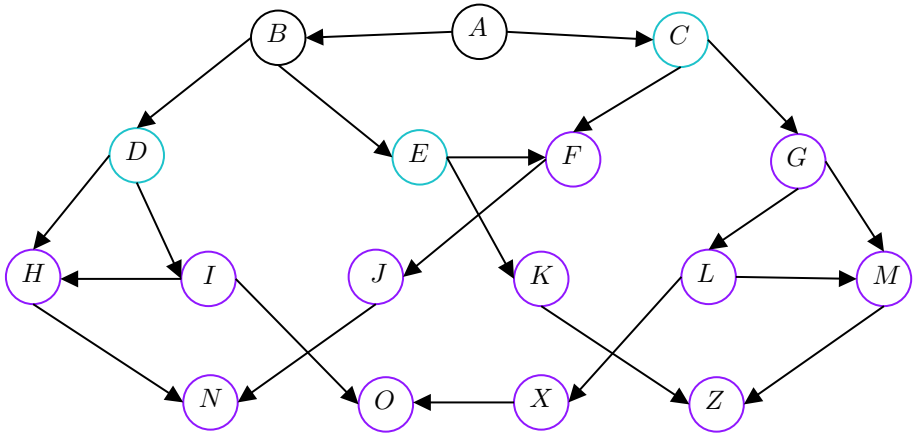
H I J K L M N O X Z

k)



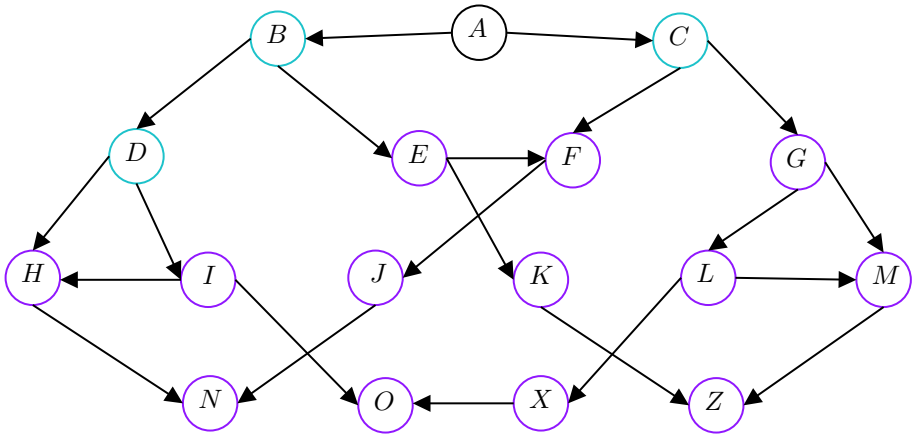
G H I J K L M N O X Z

l)



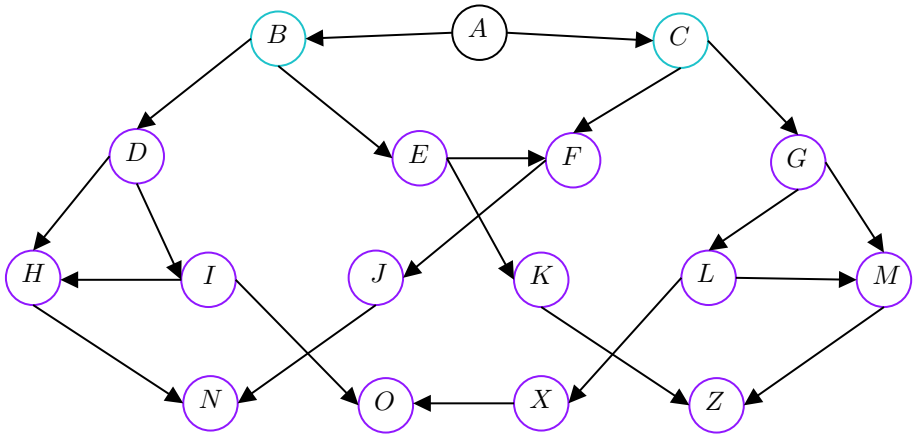
F G H I J K L M N O X Z

m)



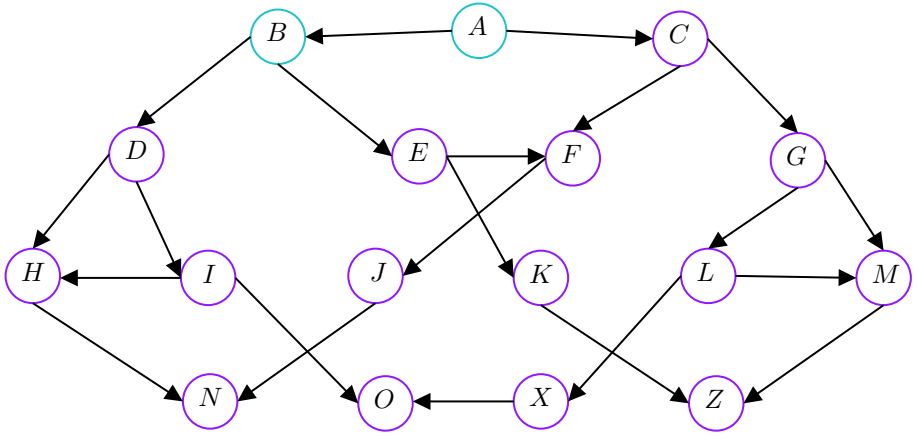
E F G H I J K L M N O X Z

n)



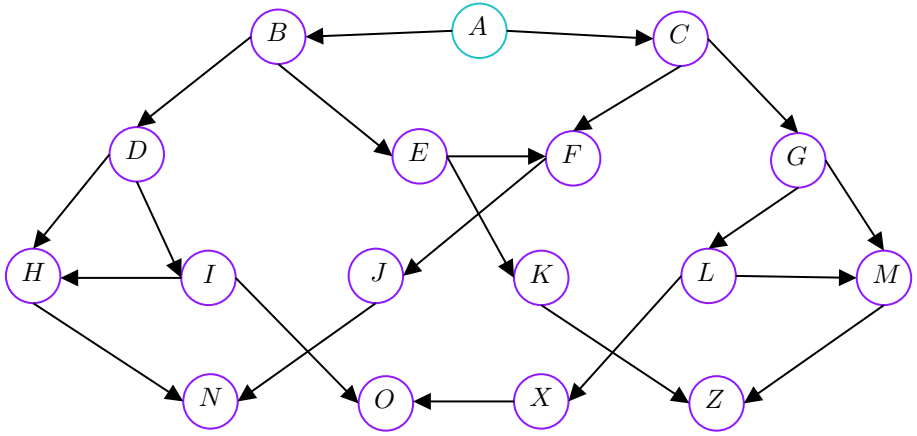
D E F G H I J K L M N O X Z

o)

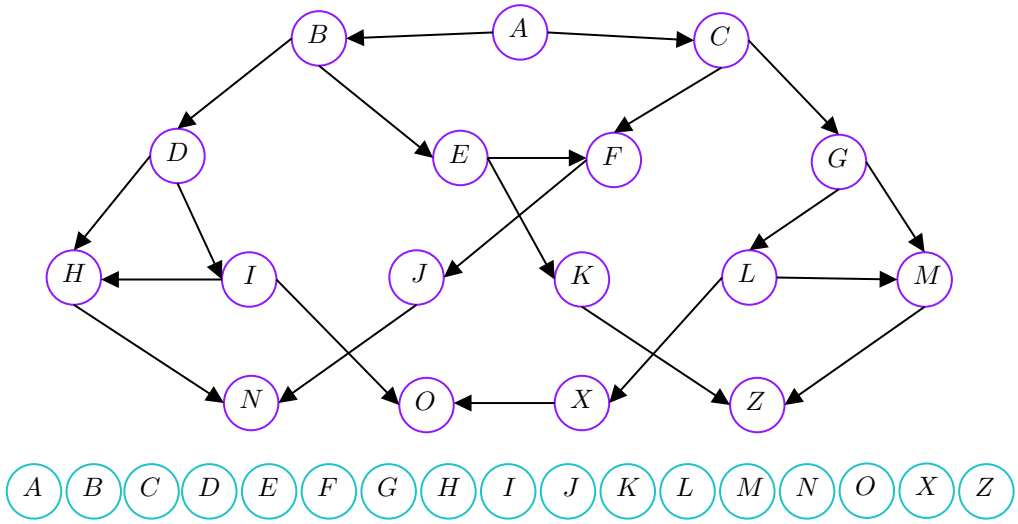


C D E F G H I J K L M N O X Z

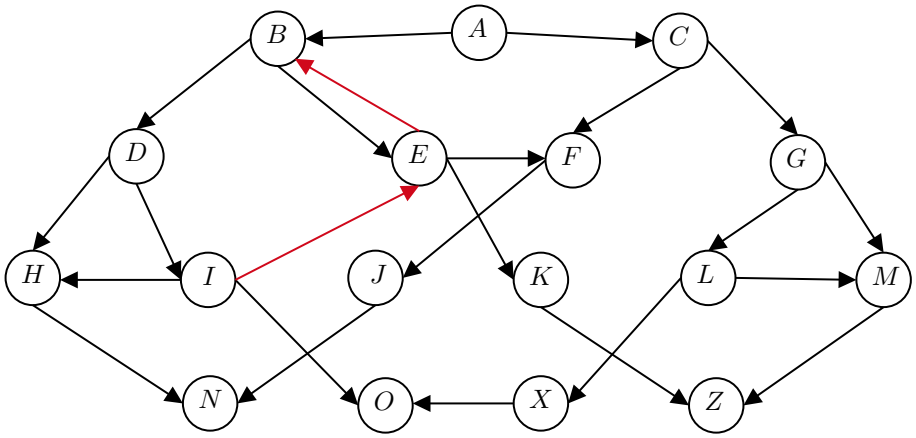
p)



B C D E F G H I J K L M N O X Z



b) Construir una gráfica G de al menos 17 vértices y 23 aristas donde no pueda aplicarse Topological-Sorting, indicar las razones por qué no podría aplicarse **T-S**.



En la presente, no es posible ejecutar **T-S** ya que tenemos un **ciclo** en la grafica y uno de los requisitos para poder ejecutar **T-S**, es que la grafica en cuestion sea **aciclica**.

4 **[Opcional] Resolver el siguiente problema:**

Problema D:
Se le llama **"Bipartita"** a una gráfica no dirigida si sus **vértices** pueden ser **coloreados con a lo más dos colores**, de tal forma que las **aristas conectan con vértices de diferentes colores**.
Encontrar un algoritmo que verifique si una gráfica G es bipartita en tiempo $C * (|V| + |E|)$, donde C es el número de componentes conexas de G.

a) **Diseñar un algoritmo** que solucione el problema elegido.
Pre Condiciones: La entrada recibida es unica y exclusivamente una grafica disconexa con 2 o mas componentes conexas no vacias y con minimo dos vertices, a su vez, todos y cada uno de los vertices de cada componente conexa debe de estar unido por minimo una arista a otro vertice.

Como parte de nuestras pre condiciones, tendremos que solo pueden existir dos colores en toda la coloracion que haremos, estos colores no son fijos ya que al ejecutar este algoritmo, podriamos repetir el proceso con por ejemplo blanco-negro, o con verde-azul, etc ... , es por esto que al primer color lo conoceremos como color1 y al segundo como color2, tambien debemos decir que se debe cumplir que $color1 \neq color2$, pero a su vez todos los vertices coloreados con color1 deben ser exclusivamente coloreados con color1, lo mismo aplica para color2.

Post Condiciones: Se devuelve una respuesta, sobre si es bipartita la gráfica recibida o en su defecto, devuelve que la gráfica recibida no es bipartita.

Procedimiento:

1) Se identifican cuales son todas las componentes conexas de la grafica recibida.

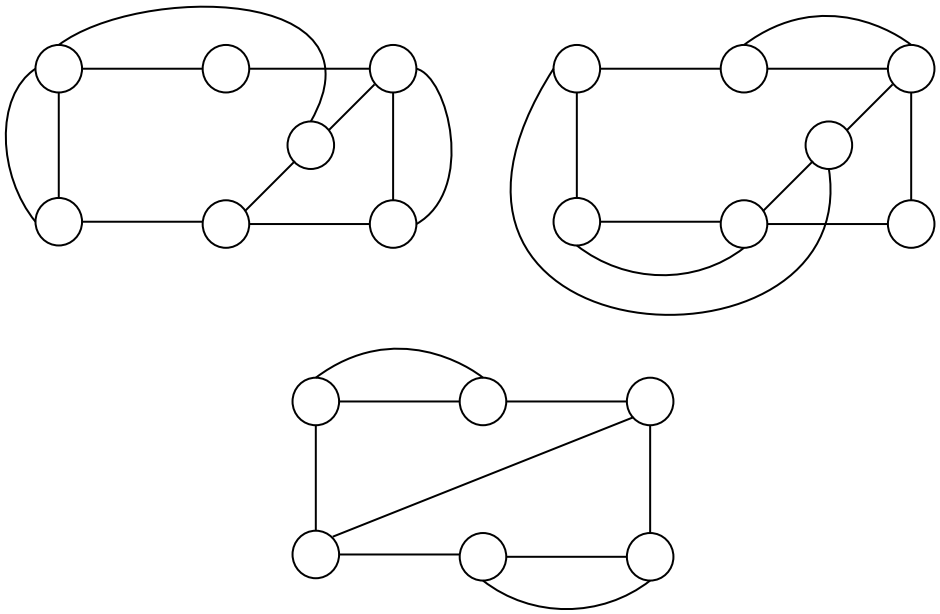
2) Para cada una de las componentes conexas identificadas, coloreamos un vertice arbitrario en cada componente con el color1.

3) Ahora, todas las componentes conexas comienzan a realizar el siguiente procedimiento paralelamente (al mismo tiempo):
En el paso **2)** coloreamos un vertice en cada componente, ahora a partir de ese vertice vamos a colorear a todos sus vecinos con el color2 (es decir, vamos a colorear a todos los vecinos con el color contrario al inicial), ahora, para todos los vecinos que hemos coloreado, vamos a visitar a todos sus vecinos y los vamos a colorear con el color contrario al del vecino original, en caso de encontrar que un vertice ya esta coloreado, no se visita y por ende no se colorea, este proceso lo vamos a repetir hasta que todos los vertices de la componente conexa en la que estemos terminen coloreados o nos encontremos en conflicto en alguna componente y no podamos colorear algun(os) vertices, lo primero nos dira que hemos tenido exito y la grafica que hemos operado es bipartita, lo segundo nos dira que la grafica no es bipartita.

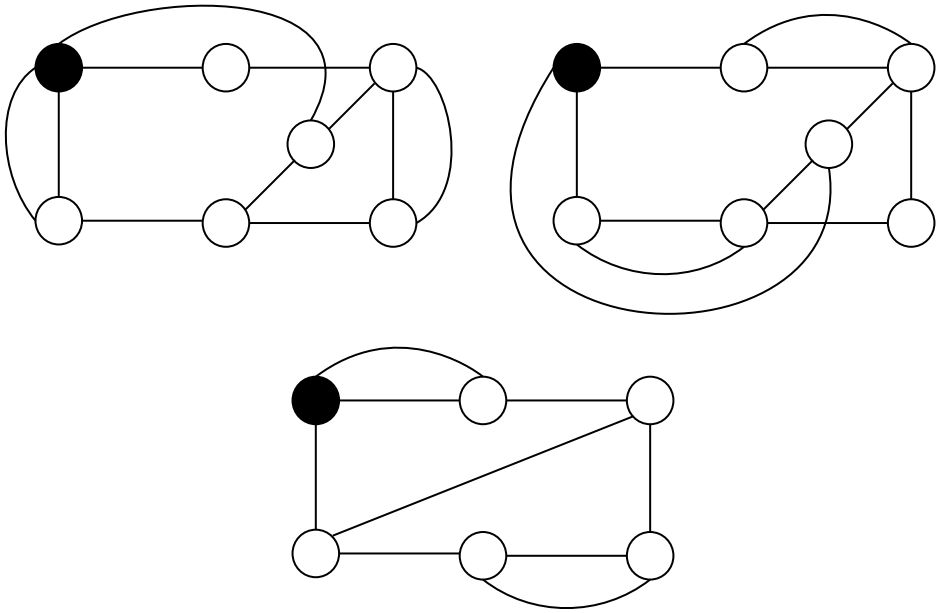
b) **Justificar** por qué se alcanza la **complejidad deseada**.
Como nuestro algoritmo se dedica a visitar todos los vertices de la grafica con el metodo de recorrerlos partiendo de un vertice inicial y visitando los vecinos de sus vecinos hasta que pase alguno de los dos escenarios anteriores (que la grafica sea bipartita o no lo sea), con esto solo estamos recorriendo los vertices de las componentes conexas de la grafica una sola vez y por ende tendremos una complejidad en tiempo (para eso) de $(|V| + |E|)$, ahora, como este proceso lo repetimos para **todas** las componentes conexas, esto nos dara que la complejidad será de $C * (|V| + |E|)$.

c) **Construir una gráfica G disconexa** de al menos 20 vértices, 31 aristas y 3 componentes conexas y **aplicar** el algoritmo propuesto.

La grafica con la que trabajaremos será:

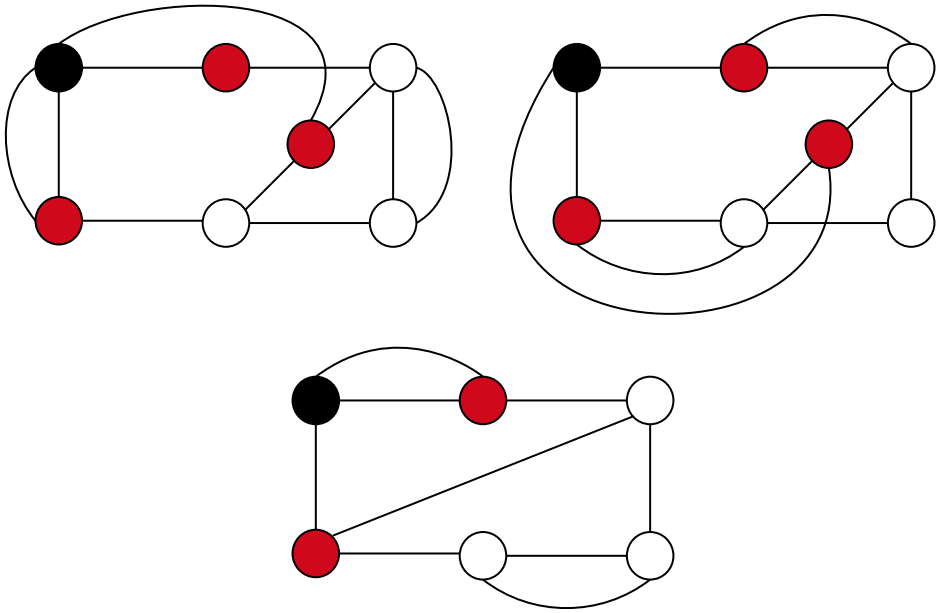


Ejecutamos el paso **1** y **2**.

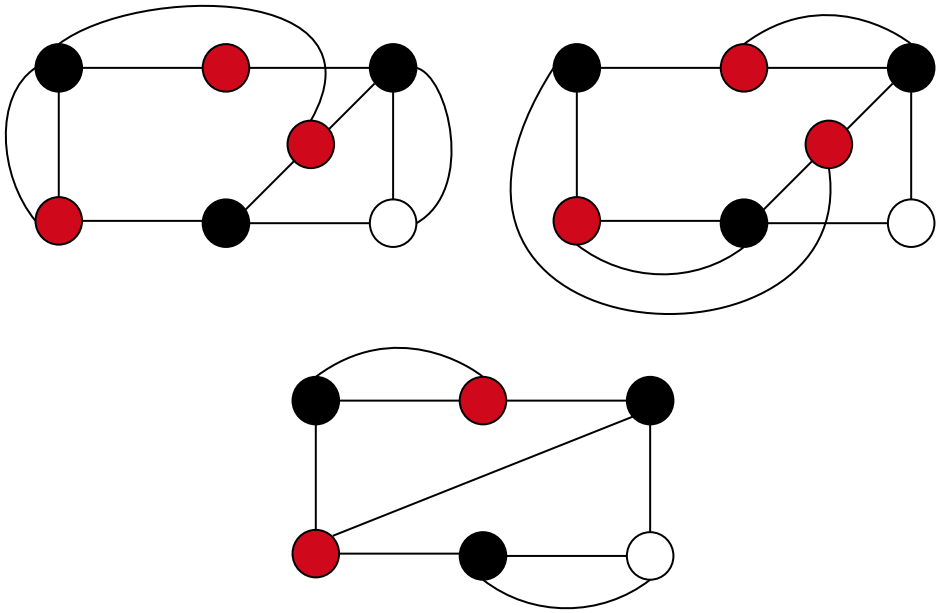


Ejecutamos el paso **3**.

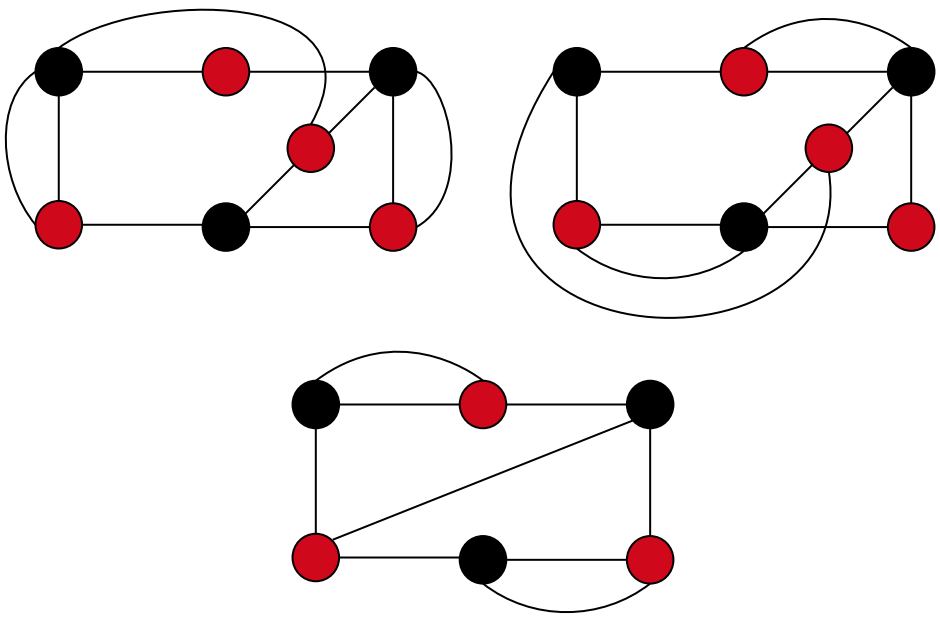
a)



b)



c)



En este caso, hemos podido colorear todas las componentes conexas de la grafica y por la definicion de bipartita podemos decir que esta grafica en particular, es bipartita.