

# Grafos y ciclos

Clase 15

IIC 2133 - Sección 2

Prof. Mario Droguett

# Sumario

**Introducción**

Grafos

Detección de ciclos

# Viabilidad de proyectos con requisitos

Consideremos un proyecto complejo dividido en varias tareas

- Cada tarea es indivisible
- Algunas tareas tienen como requisito otras tareas
- No nos preocupamos del tiempo que toma cada tarea

## Ejemplo

Un proyecto  $G$  tiene las siguientes tareas con su especificación de requisitos

- T0: no tiene requisitos
- T1: requiere T0
- T2: requiere T0
- T3: requiere T1 y T2

¿Cómo sabemos si el proyecto especificado es viable?

# Viabilidad de proyectos con requisitos

Solo nos interesa la ejecución de tareas con requisitos

- El proyecto no será viable si hay una referencia **circular**

Para dos tareas  $A$  y  $B$ , si  $B$  tiene como requisito la tarea  $A$  entonces escribiremos

$$A \rightarrow B$$

Con esto, el proyecto es inviable si existe una secuencia de tareas

$$X \rightarrow T_1 \rightarrow \dots \rightarrow T_n \rightarrow X$$

¿Es la única condición que debemos verificar para decidir?

# Viabilidad de proyectos con requisitos

La ausencia de secuencias circulares es condición **necesaria y suficiente** para asegurar la viabilidad del proyecto

- Si hay una secuencia circular, no podemos **ordenar** las tareas
- Si no hay secuencia circular, ordenamos

El orden de las tareas nos indica en qué orden **ejecutarlas** para completar el proyecto

¿Cómo verificar la viabilidad de un proyecto de manera algorítmica?

# Representación de la dependencia de tareas

## Ejemplo

Un proyecto  $G$  tiene las siguientes tareas con su especificación de requisitos

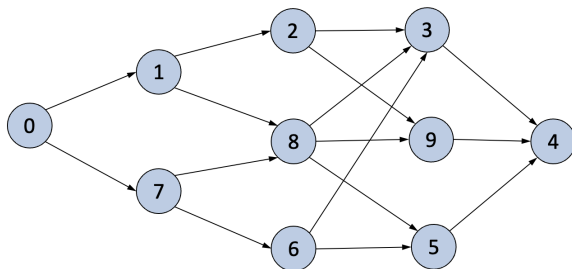
- |                            |                        |
|----------------------------|------------------------|
| ■ T0: no tiene requisitos  | ■ T5: requiere T6 y T8 |
| ■ T1: requiere T0          | ■ T6: requiere T7      |
| ■ T2: requiere T1          | ■ T7: requiere T0      |
| ■ T3: requiere T1, T6, T8  | ■ T8: requiere T1 y T7 |
| ■ T4: requiere T3, T5 y T9 | ■ T9: requiere T2 y T8 |

¿Cómo podemos representar de manera eficiente este escenario?

# Representación de la dependencia de tareas

## Ejemplo

- T0: no tiene requisitos
- T1: requiere T0
- T2: requiere T1
- T3: requiere T1, T6, T8
- T4: requiere T3, T5 y T9
- T5: requiere T6 y T8
- T6: requiere T7
- T7: requiere T0
- T8: requiere T1 y T7
- T9: requiere T2 y T8



# Sumario

Introducción

**Grafos**

Detección de ciclos



# Grafos

La representación anterior se conoce como **grafo dirigido**

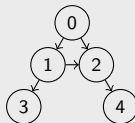
## Definición

Un **grafo dirigido** es un par  $G = (V, E)$ , donde  $V$  es el conjunto de **nodos** y  $E$  es el conjunto de **aristas** de la forma  $(u, v)$ , con  $u, v \in V$

## Ejemplo

$$V(G) = \{0, 1, 2, 3, 4\}$$

$$E(G) = \{(0, 1), (0, 2), (1, 2), (1, 3), (2, 4)\}$$



Usaremos grafos dirigidos cuando exista  
un nodo de partida y uno de llegada

# Grafos

De forma análoga definimos los grafos **no dirigidos**

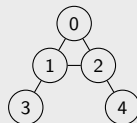
## Definición

Un **grafo no dirigido** es un par  $G = (V, E)$ , donde  $V$  es el conjunto de **nodos** y  $E$  es el conjunto de **aristas** de la forma  $\{u, v\}$ , con  $u, v \in V$  y  $u \neq v$ .

## Ejemplo

$$V(G) = \{0, 1, 2, 3, 4\}$$

$$E(G) = \{\{0, 1\}, \{0, 2\}, \{1, 2\}, \{1, 3\}, \{2, 4\}\}$$



Usaremos grafos no dirigidos cuando importe agrupar pares de nodos

# Grafos

Cuando hablemos de complejidad de algoritmo sobre grafos

- usaremos  $|V|$  y  $|E|$  como parámetros de tamaño
- los abreviaremos como  $V$  y  $E$

Con esto, expresaremos la complejidad en términos de los elementos que definen el grafo

¿Es mejor un algoritmo  $\mathcal{O}(V)$  o  $\mathcal{O}(E)$ ?

Depende! Hay dos situaciones extremas

- $|E| = 0$
- Grafo completo (todas las aristas posibles)

# Representación de grafos

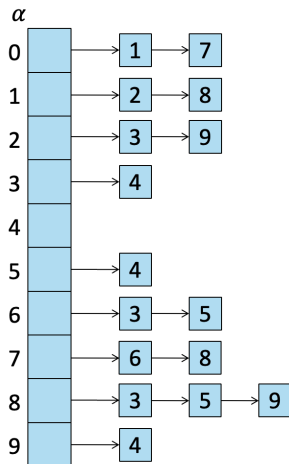
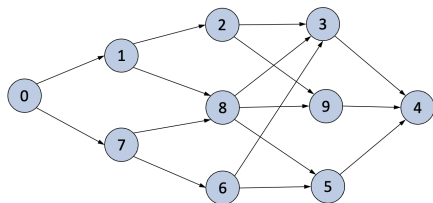
Necesitamos codificar los grafos y almacenarlos

1. **Listas de adyacencias** en que la celda  $i$  de la lista tiene una lista con los vecinos  $j$  tales que  $(i,j)$  es arista en el grafo
2. **Matriz de adyacencias** en que la celda  $[i][j]$  indica si la arista  $(i,j)$  existe en el grafo

Ambas almacenan la información de nodos en sus dimensiones, mientras que las aristas son los datos adicionales

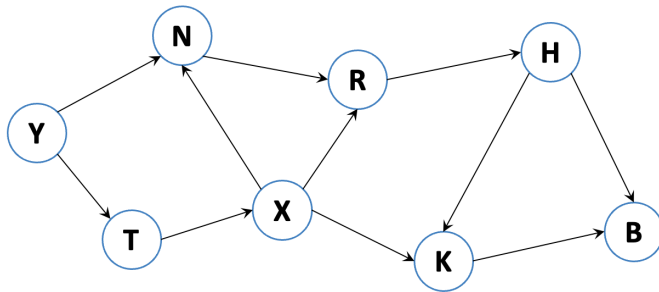
La complejidad de los algoritmos depende de qué representación se usa

# Representación de grafos



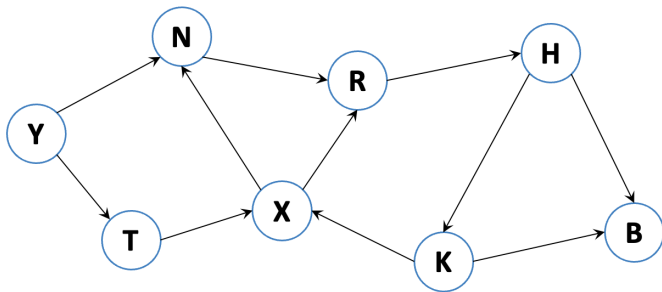
La complejidad de los algoritmos depende de qué representación se usa

## Volviendo al problema



¿Algún problema con este proyecto?

## Volviendo al problema



¿Algún problema con este proyecto?

# Ciclos

## Definición

Dado un grafo dirigido  $G = (V, E)$ , un **camino**  $\pi$  es una secuencia de nodos  $v_0, v_1, \dots, v_n$  tal que

$$\{v_i, v_{i+1}\} \in E \quad \text{para cada } i < n$$

Decimos que  $\pi := v_0, v_1, \dots, v_n$  es de largo  $n$ . Si  $n > 0$  y  $v_0 = v_n$  diremos que  $\pi$  es un **ciclo**. Un grafo que posee un ciclo se dice **cíclico**.

Para nuestro problema, determinar si hay una secuencia circular de tareas equivale a determinar si el grafo de representación contiene un ciclo

¿Podemos definir un algoritmo para detectar ciclos?



# Sumario

Introducción

Grafos

Detección de ciclos

# Ciclos

Dadas dos tareas  $X$  e  $Y$ , diremos que  $Y$  es **posterior** a  $X$  si

- $X \rightarrow Y$  o
- existe una tarea  $Z$  tal que  $X \rightarrow Z$  e  $Y$  es posterior a  $Z$

En esencia:  $Y$  es posterior a  $X$  si  $X$  debe realizarse antes que  $Y$

Esta propiedad es clave para definir un algoritmo de detección de ciclos

# Ciclos

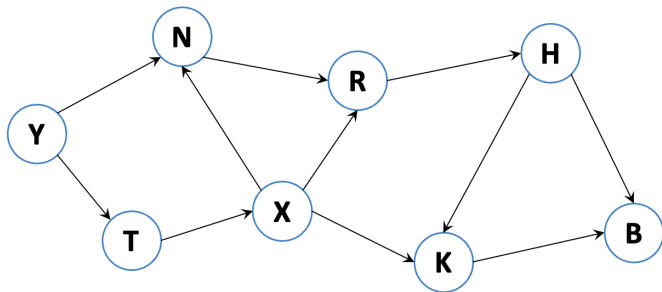
**input** : Grafo  $G$ , nodo  $X \in V(G)$

Posteriors( $G, X$ ):

```
1   $P \leftarrow \emptyset$ 
2  for  $Y$  tal que  $X \rightarrow Y$  :
3       $P \leftarrow P \cup \{Y\}$ 
4       $P \leftarrow P \cup \text{Posteriors}(G, Y)$ 
5  return  $P$ 
```

¿Qué complejidad tiene este algoritmo?

# Ciclos

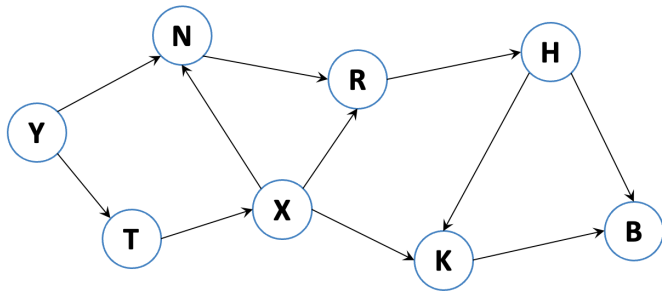


## Ejercicio

Para el grafo anterior, determine los nodos posteriores a  $X$  usando el algoritmo Posteriors

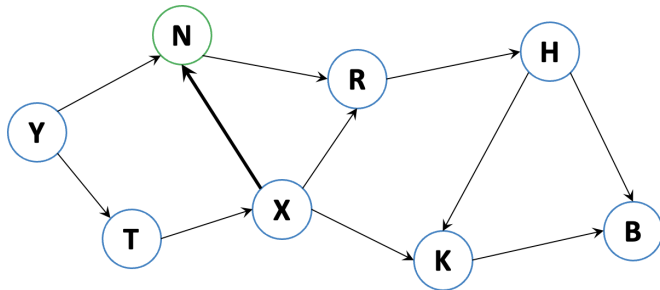
# Ciclos

Exploramos cada arista de salida desde  $X$



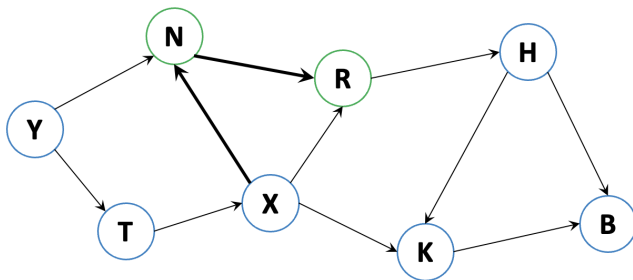
# Ciclos

Encontramos primero a *N*



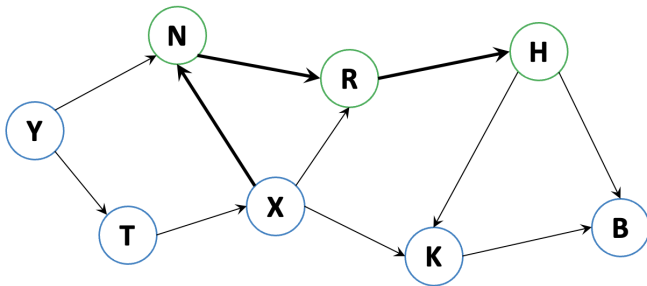
# Ciclos

Ojo, ahora corresponde revisar las aristas de  $N$  recursivamente:  
encontramos a  $R$



# Ciclos

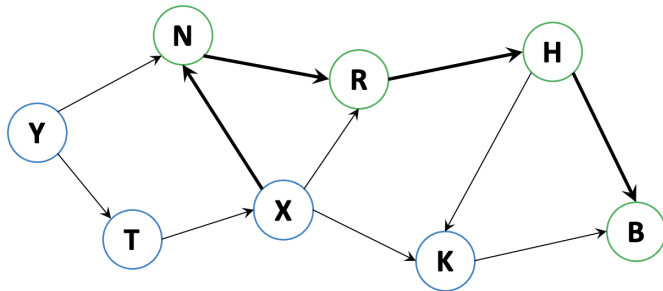
Seguimos, comenzando en  $R$  y encontrando  $H$





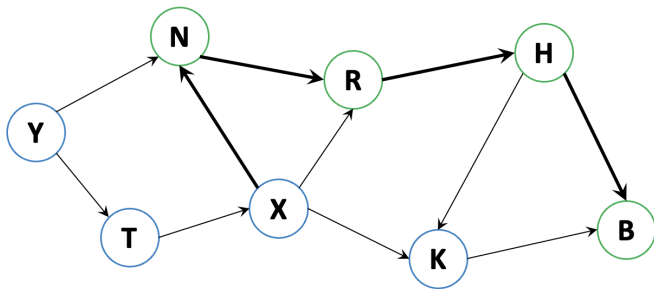
# Ciclos

Seguimos, comenzando en  $H$  y encontrando  $B$



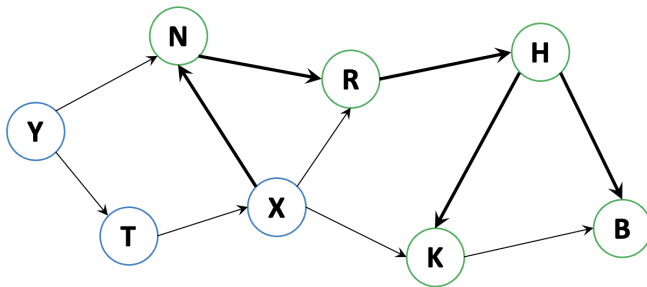
# Ciclos

Comenzando en *B* no tenemos aristas de salida. **Retornamos** hacia atrás



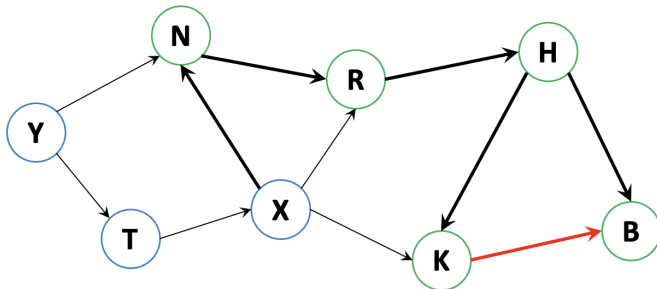
# Ciclos

Comenzando en  $H$ , exploramos la otra arista de salida, encontrando  $K$



# Ciclos

Desde  $K$  volvemos a encontrarnos con  $B$  y retornamos hacia atrás



¿Qué problema tiene el algoritmo actual?

# Ciclos

El algoritmo `Posterioris` no evita nodos ya visitados

- esto no solo impacta en la complejidad práctica
- si hay un ciclo en el grafo, el algoritmo **no termina**

Necesitamos poder distinguir a los ya visitados

Agregamos un atributo a los nodos

# Ciclos

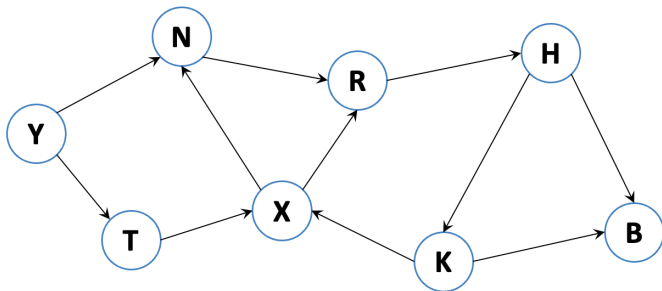
**input** : Grafo  $G$ , nodo  $X \in V(G)$

Posteriors( $G, X$ ):

```
1  if  $X.visited = 1$  : return  $\emptyset$ 
2   $X.visited \leftarrow 1$ 
3   $P \leftarrow \emptyset$ 
4  for  $Y$  tal que  $X \rightarrow Y$  :
5       $P \leftarrow P \cup \{Y\}$ 
6       $P \leftarrow P \cup \text{Posteriors}(G, Y)$ 
7  return  $P$ 
```

Esta versión termina siempre

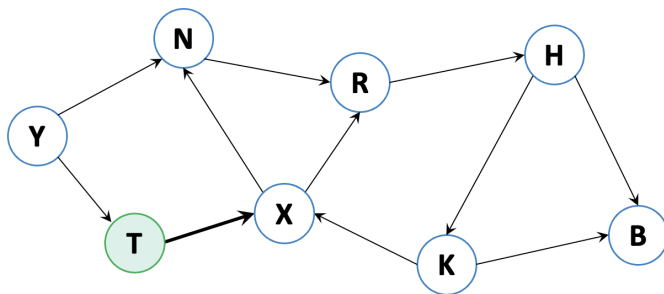
# Ciclos



## Ejercicio

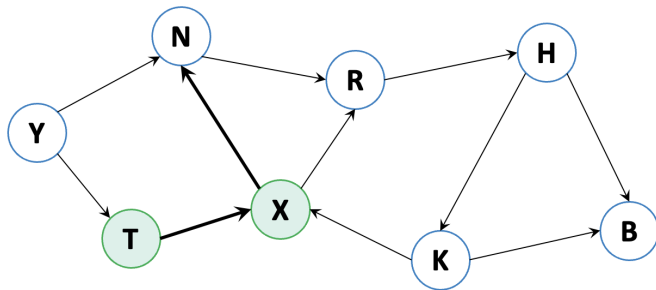
Para el grafo anterior, determine los nodos posteriores a  $T$  usando el algoritmo `Posteriors` que detecta visitados

# Ciclos

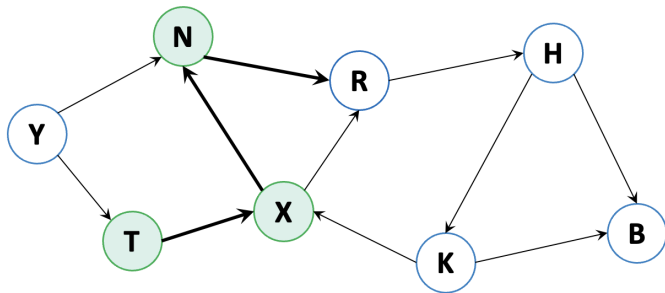




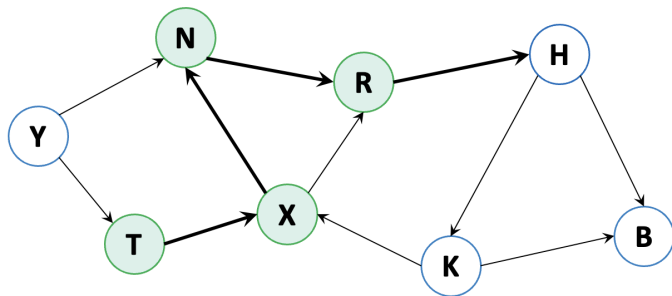
# Ciclos



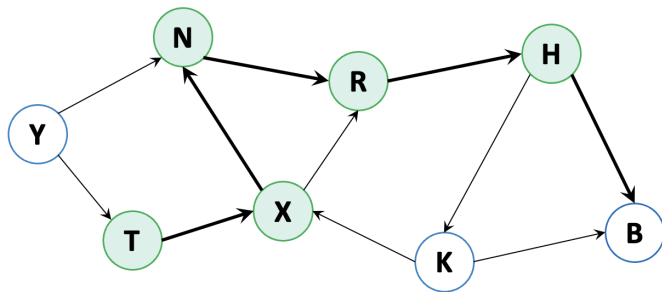
# Ciclos



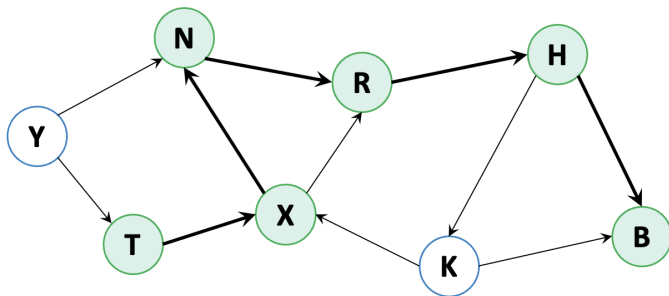
# Ciclos



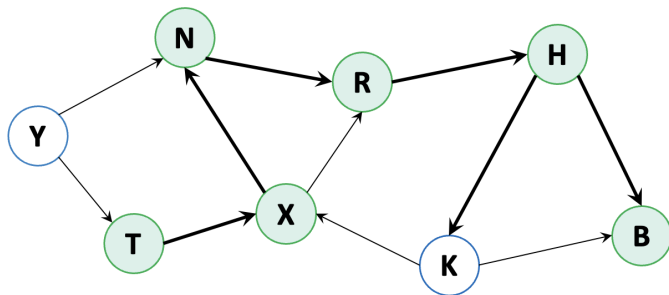
# Ciclos



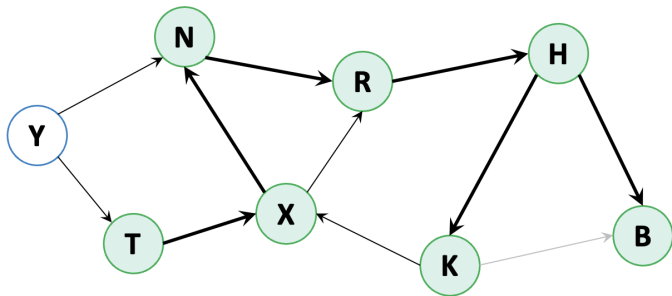
# Ciclos



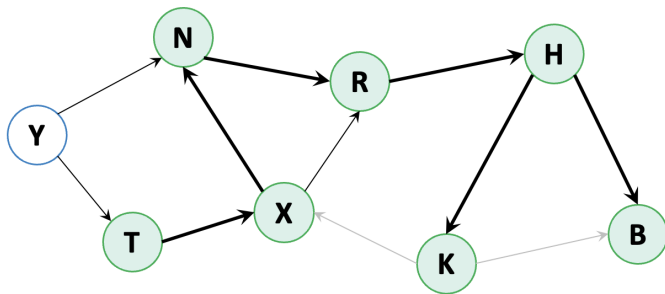
# Ciclos



# Ciclos

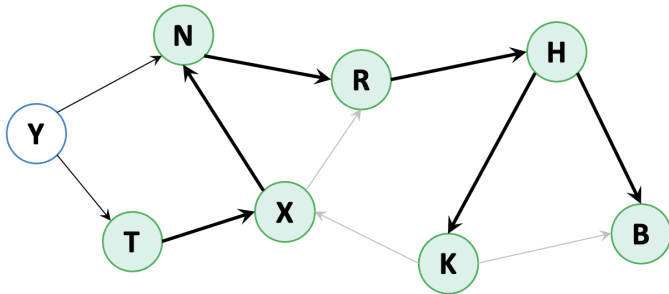


# Ciclos





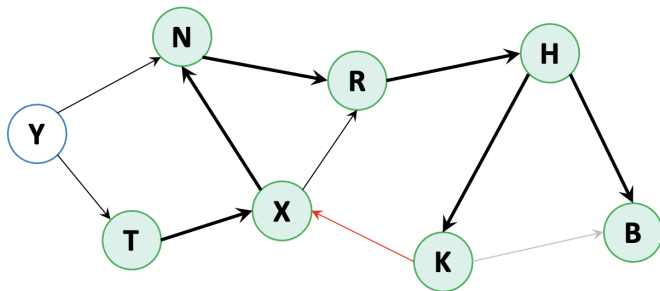
# Ciclos



¿El algoritmo *se dio cuenta* que habían ciclos?

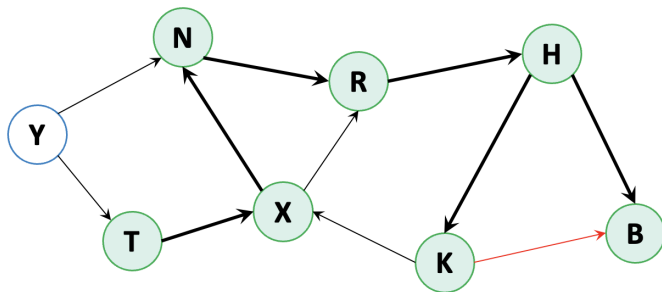
¿Basta decir que hay ciclo si se llama  $\text{Posteriors}(G, W)$  para  $W$  visitado?

# Ciclos



Queremos que esto sea detectado como ciclo

# Ciclos



Esto **NO DEBE** ser considerado un ciclo

# Detección de ciclos

Proponemos la siguiente regla para detección de ciclos usando Posteriors

- Si el nodo  $Y$  que vamos a visitar ya fue visitado
  1. Si estamos en un nodo posterior a  $Y$ , **hay ciclo**
  2. Si no, entonces **esta arista no forma ciclo** (pueden haber en otras zonas)

Importante: hasta que  $\text{Posteriors}(G, X)$  retorne, todos los visitados que se marcan son posteriores a  $X$

Agregamos colores para distinguir el **tipo de visitado**:  
blanco=no visitado, gris=visitado posterior, negro=visitado no posterior

# Detección de ciclos

**input** : Grafo  $G$ , nodo  $X \in V(G)$

**cycleAfter**( $G, X$ ):

```
1  if  $X.color = gris$  : return true
2  if  $X.color = negro$  : return false
3   $X.color \leftarrow gris$ 
4  for  $Y$  tal que  $X \rightarrow Y$  :
5      if cycleAfter( $G, Y$ ) : return true
6   $X.color \leftarrow negro$ 
7  return false
```

Este algoritmo decide si hay un ciclo **partiendo desde  $X$**

# Detección de ciclos

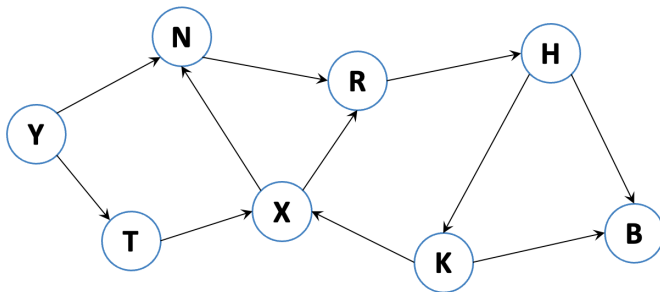
**input** : Grafo  $G$

**isCyclic**( $G$ ):

```
1  for  $X \in V(G)$  :  
2      if  $X.color \neq blanco$  :  
3          continue  
4      if CycleAfter( $G, X$ ) :  
5          return true  
6  return false
```

Este algoritmo decide si hay un ciclo **en el grafo**

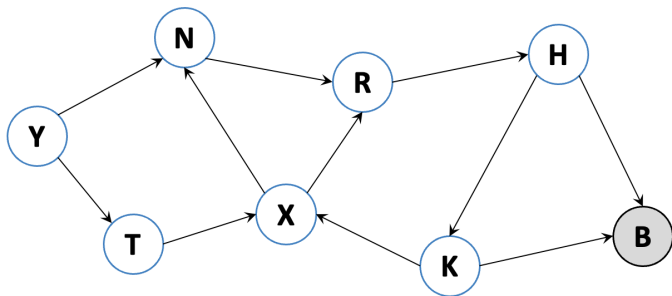
## Detección de ciclos



### Ejercicio

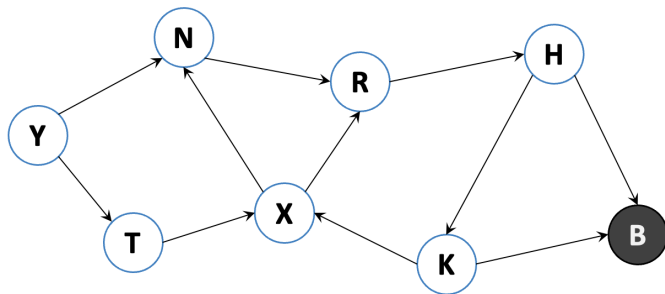
Para el grafo anterior, determine si posee algún ciclo usando el algoritmo `isCyclic`. Asuma que el **for** de la línea 1 de `isCyclic` escoge los nodos en orden alfabético

## Detección de ciclos

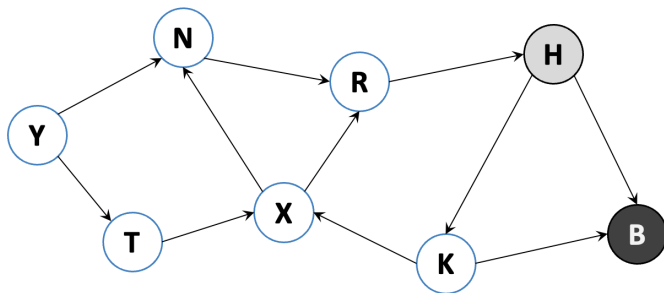




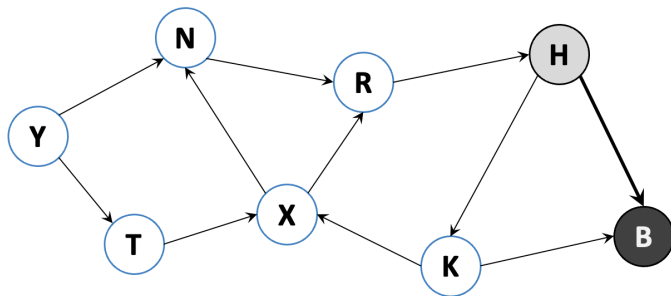
## Detección de ciclos



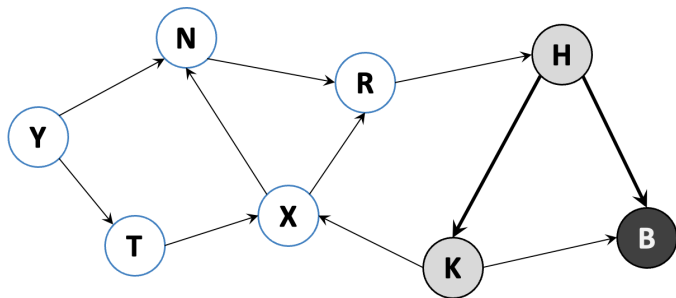
## Detección de ciclos



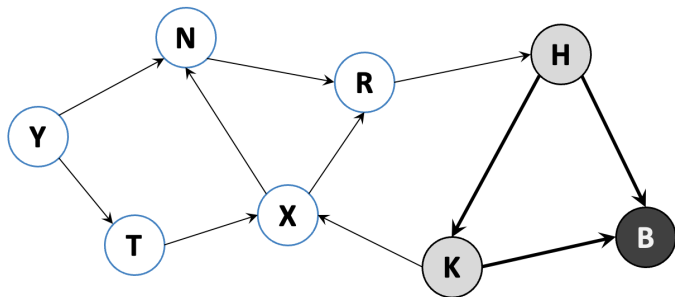
## Detección de ciclos



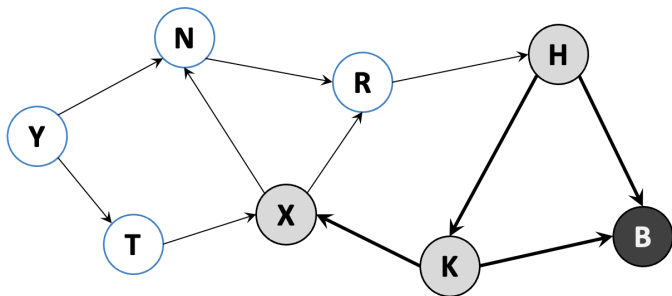
## Detección de ciclos



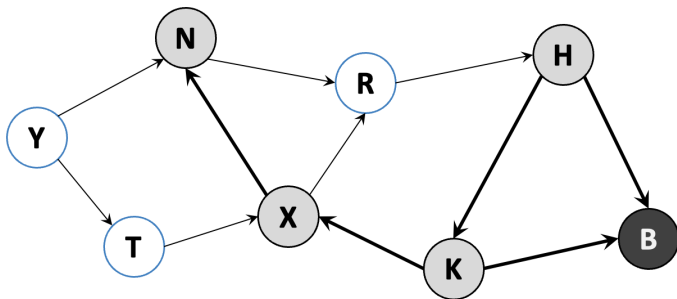
## Detección de ciclos



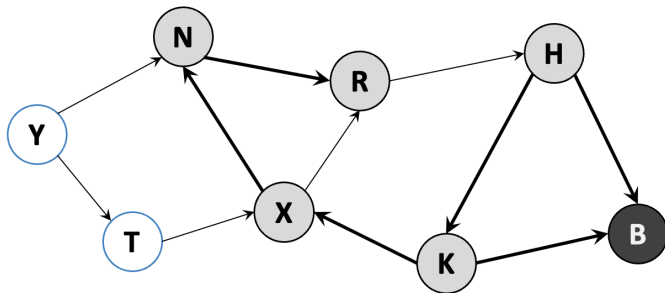
## Detección de ciclos



## Detección de ciclos

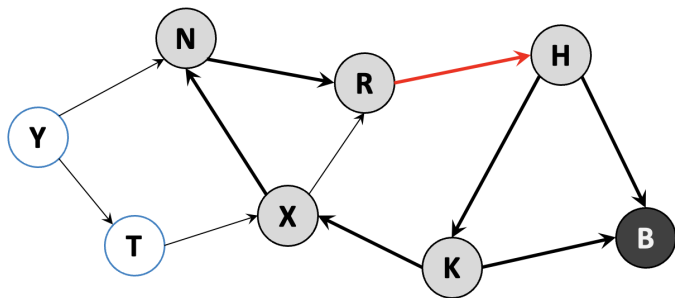


## Detección de ciclos





## Detección de ciclos



# Búsqueda en profundidad

El algoritmo `isCyclic` sigue una estrategia de **búsqueda en profundidad**

- también abreviada **DFS** por las siglas de *Depth First Search*
- la estrategia es avanzar por aristas adyacentes hasta más no poder
- luego de agotar esa ruta, se retrocede y se exploran otras
- todas las rutas se exploran **en profundidad**

¿Cuál es la complejidad de estos algoritmos?