Escuela Politécnica Nacional | EDA II - Informe N° 7

Ismael Freire

Tabla de contenidos

1. Objetivos

- Explicar qué es el ordenamiento topológico y en qué tipos de grafos es aplicable.
- Señalar ejemplos concretos en los que el ordenamiento topológico es útil.
- Desarrollar y explicar el código del algoritmo de ordenamiento topológico en Python, siguiendo las recomendaciones y ejemplos dados en clase.

2. Introducción

El ordenamiento topológico es un algoritmo clave en la teoría de grafos, aplicado en grafos dirigidos y acíclicos (DAGs). Su objetivo principal es organizar los nodos en un orden lineal, de manera que si existe una arista dirigida desde el nodo A hacia el nodo B, entonces A aparece antes que B en la ordenación. Dicho de otro modo, A tiene que completarse antes de poder

pasar a B, lo que muestra que este tipo de orden es esencial cuando ciertas tareas o procesos tienen que completarse antes de que otros puedan empezar.

Este ordenamiento resulta especialmente útil en escenarios donde se deben seguir dependencias específicas, como en la planificación de tareas, la compilación de código, y la gestión de prerequisitos en procesos académicos (mallas curriculares).

Por lo que, en este informe se explorará el concepto y la implementación del ordenamiento topológico, enfatizando su aplicación práctica en los ejercicios planteados y su relevancia en la resolución de problemas.

3. Ejercicios planteados y/o programas implementados

1. Aplique el algoritmo de ordenamiento topológico en el siguiente ejemplo:

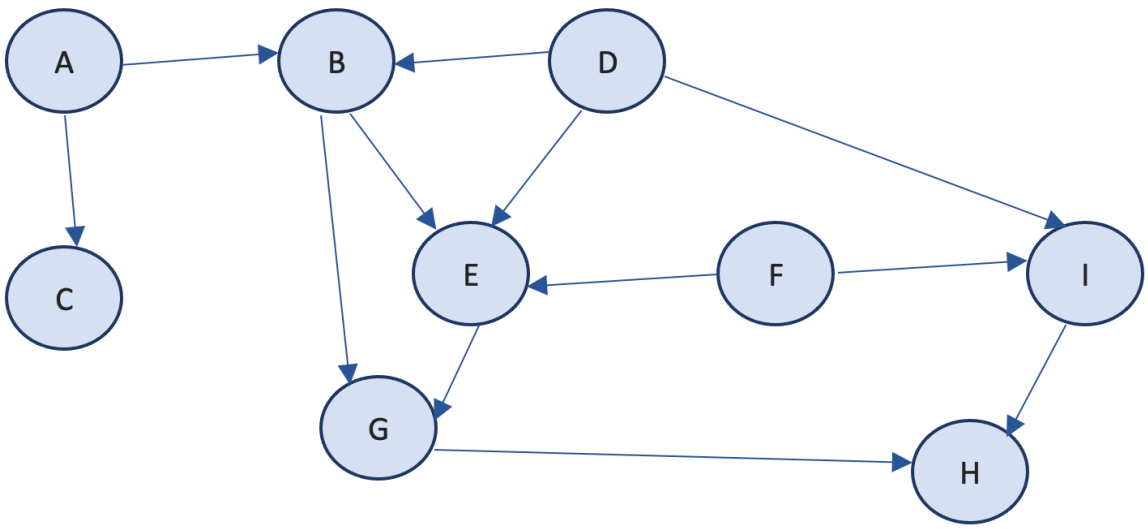

Figura 1: ejercicio1_t_o.png

Para aplicar el algoritmo de ordenamiento topological sort primero se debe pasar el grafo dado a una estructura de tipo diccionario que permita trabajar con nodos y conecciones.

```
graph = {
    "A":["B", "C"],
    "B":["E", "G"],
    "C":[],
    "D":["E", "B", "I"],
```

```
    "E":["G"],
    "F":["E", "I"],
    "G":["H"],
    "H":[],
    "I":["H"]
}
```

graph = { “A”:[“B”, “C”], “B”:[“C”, “D”], “C”:[“E”], “D”:[“E”], “E”:[], “F”:[“E”] }

Seguidamente, se define la función *indegree()* que se utiliza para calcular el número de predecesores de un nodo en un grafo dirigido. Este calculo en teoría de grafos es llamado *indegree* de un nodo y es el calculo de las aristas que entran en dicho nodo.

Dentro de la función, se define un contador *indegree_value*, que comienza en cero y se encarga de almacenar la cantidad de predecesores de un nodo. Luego, se recorre todos los nodos del grafo y se verifica si el nodo dado aparece en las listas de vecinos. Cada vez que el nodo se encuentre, se incrementa el contador mencionado en 1. Finalmente, la función devuelve el total acumulado, que corresponde al indegree del nodo.

```
def indegree(grafo, nodo):
    indegree_value = 0
    for _, vecino in grafo.items():
        if nodo in vecino:
            indegree_value += 1
    return indegree_value
```

El algoritmo de ordenamiento topológico procesa un grafo dirigido y acíclico (DAG) para determinar un orden lineal de sus nodos, donde cada nodo aparece antes que sus sucesores. Primero, calcula el “indegree” (número de predecesores) de cada nodo utilizando la función *indegree()*. En su lugar, los nodos sin predecesores se colocan en una lista ready para ser procesados. Luego, en un bucle, cada nodo de ready se elimina y se añade al orden topológico (top_sorted). Al procesar cada nodo, se actualiza el “indegree” de sus vecinos, y si alguno queda sin predecesores, se añade a ready. Este proceso continúa hasta que no haya más nodos listos, en *ready*, resultando en una lista que representa el ordenamiento topológico del grafo.

```
def topological_sort(grafo):
    top_sorted = []
    ready = []
    incount = {}

    for nodo in grafo.keys(): #por cada nodo que pertenece al grafo
        incount [nodo] = indegree(grafo, nodo) #extraer el indegree del nodo
```

```
        if incount [nodo] == 0: #nodo sin predecesores puede usarse
            ready.append(nodo)
    # Dividir el diccionario para facilitar su impresión
    keys_indegree = list(incount)
    mitad_indegree = len(keys_indegree)//2
    primer_indegree = {k: incount[k] for k in keys_indegree[:mitad_indegree]}
    segundo_indegree = {k: incount[k] for k in keys_indegree[mitad_indegree:]}

    print("Indegree", primer_indegree)
    print(segundo_indegree)
    #print('Indegree:', incount)
    print ('READY:', ready)

    while ready: #mientras haya nodos listos para ordenarse
        nodo = ready.pop()
        print ('Nodo para usarse', nodo)
        # nodo es insertado en orden topológico
        top_sorted.append(nodo)
        for vecino in grafo[nodo]:
            print("{} : {}".format(vecino, incount[vecino]))
            incount[vecino] -= 1
            if incount[vecino] == 0:
                ready.append(vecino)
        print("Ready: ", ready)
    return top_sorted
```

```
# Llamada a la función
topological_sort(graph)
```

```
Indegree {'A': 0, 'B': 2, 'C': 1, 'D': 0}
{'E': 3, 'F': 0, 'G': 2, 'H': 2, 'I': 2}
READY: ['A', 'D', 'F']
Nodo para usarse F
E : 3
I : 2
Ready:  ['A', 'D']
Nodo para usarse D
E : 2
B : 2
I : 1
Ready:  ['A', 'I']
```

```
Nodo para usarse I
H : 2
Ready:  ['A']
Nodo para usarse A
B : 1
C : 1
Ready:  ['B', 'C']
Nodo para usarse C
Ready:  ['B']
Nodo para usarse B
E : 1
G : 2
Ready:  ['E']
Nodo para usarse E
G : 1
Ready:  ['G']
Nodo para usarse G
H : 1
Ready:  ['H']
Nodo para usarse H
Ready:  []

['F', 'D', 'I', 'A', 'C', 'B', 'E', 'G', 'H']
```

2. Qué sucede si el grafo dirigido G dado tiene un ciclo? Pruebe el algoritmo de ordenamiento topológico con un grafo dirigido que tenga un ciclo y explique el resultado de lo que retorna.

Teóricamente, el algoritmo de ordenamiento topológico no puede procesar correctamente un grado dirigido que contenga ciclos porque está diseñado para grafos dirigidos acíclicos (DAG).

Por lo que, si el grafo tiene un ciclo, al menos uno de los nodos involucrados en el ciclo nunca llegará a un indegree de cero. Esto se debe a que siempre tendrá un predecesor en el ciclo, que impide que se agregue a la lista *ready*.

```python
grafo_ciclico = {
    'Ana': ['Benito', 'Carlos'],
    'Benito': ['Daniel'],
    'Carlos': ['Eva'],
    'Daniel': ['Ana'],
    'Eva': ['Fernando', 'Gabriela'],
```

```
    'Fernando': ['Hugo'],
    'Gabriela': ['Hugo'],
    'Hugo': ['Eva']
}
```

A continuación, se implementa el algoritmo para el grafo cíclico.

```
topological_sort(grafo_ciclico)
```

```
Indegree {'Ana': 1, 'Benito': 1, 'Carlos': 1, 'Daniel': 1}
{'Eva': 2, 'Fernando': 1, 'Gabriela': 1, 'Hugo': 2}
READY: []

[]
```

Como se observa en la ejecución del topológical sort, el programa se interrumpe con la lista de ready vacía. Esto se debe, a que el algoritmo no puede completar el proceso porque quedan nodos no procesados.

4. Conclusiones

El ordenamiento topológico es una herramienta escencial para organizar y resolver problemas en los que las tareas o procesos tienen dependencias específicas, en grafos dirigidos acíclicos. Por ello, a través de su aplicación se puede obtener un orden lineal de tareas o eventos que dependen unos de otros. Sin embargo, la presencia de ciclos en un grafo representa un grave problema, ya que el algoritmo no puede completarse si existen ciclos. Por tanto, es esencial asegurarse de que el grafo esté libre de ciclos antes de aplicar dicho método.

Finalmente, la resolución de los ejercicios demuestra la importancia del algoritmo de ordenamiento topológico para organizar tareas o procesos con dependencias. También resalta la necesidad de complementar este enfoque con técnicas adicionales para la detección de ciclos, lo que permite evitar errores y asegurar un funcionamiento correcto del programa. De esta manera, el ordenamiento topológico se convierte en una herramienta valiosa en la gestión de flujos de trabajo.

5. Referencias bibliográficas

[1] Python Software Foundation. (2023). The Python Standard Library. En Python Documentation (versión 3.11). Disponible en https://docs.python.org/3/.

6. Declaración uso de IA

En la realización de este informe, se utilizó la asistencia de herramientas de inteligencia artificial para la generación de ideas, organización del contenido, y redacción de ciertos apartados. Estas herramientas fueron empleadas para apoyar en la elaboración de texto de manera coherente y estructurada, sin reemplazar la revisión y juicio crítico del autor.