Informe N° 6: Ordenamiento Topológico

Gianfranco Pupiales

2024-06-13

Tabla de Contenidos

1. Objetivos

- Distinguir los conceptos fundamentales del ordenamiento topológico.
- Implementar el algoritmo de ordenamiento topológico utilizando Python.

2. Introducción

Un ordenamiento topológico toma un grafo dirigido acíclico y genera una secuencia lineal de todos sus vértices, asegurando que si el grafo G contiene una arista desde el vértice v hacia el vértice w, entonces v precede a w en la secuencia. Los grafos dirigidos acíclicos se utilizan en numerosas aplicaciones para representar la precedencia de eventos («7.17. Ordenamiento topológico — Solución de problemas con algoritmos y estructuras de datos — runestone.academy»).

3. Desarrollo

3.1 Enunciado ejercicio 1

Aplique el algoritmo de ordenamiento topológico en el siguiente ejemplo:

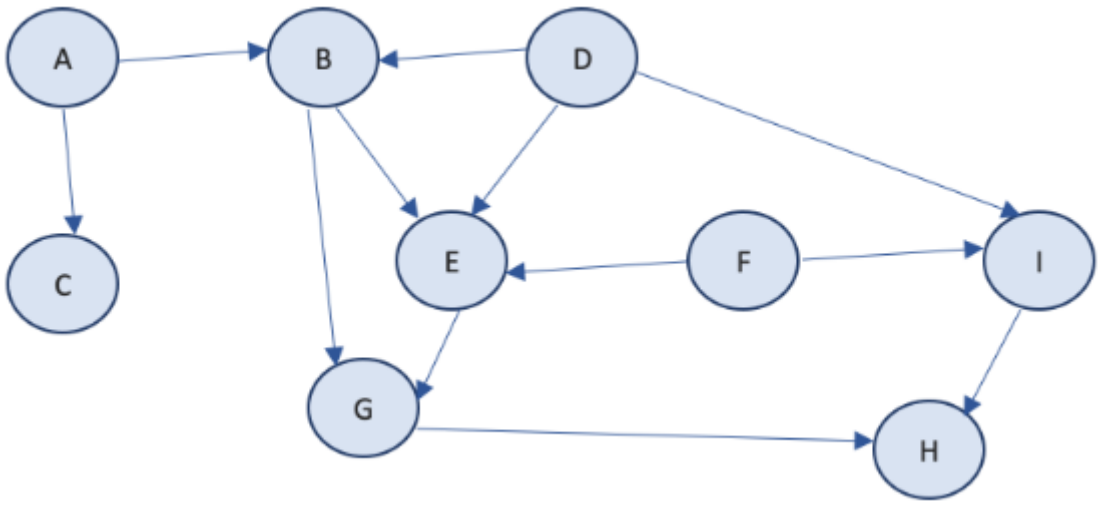

Figura 1: Grafo G

```
def indegree(nodo, grafo):
    return sum(nodo in adj for adj in grafo.values())


def topological_sort(grafo):
    top_sorted = []  # Lista para almacenar los nodos ordenados topológicamente
    ready = []  # Lista para almacenar los nodos listos para ser procesados
    incount = {}

    for nodo in grafo.keys():
        incount[nodo] = indegree(nodo, grafo)
        if incount[nodo] == 0:
            ready.append(nodo)
        print("Indegree", incount)
        print()
        print("Ready", ready)

    while len(ready) > 0:
```

```
        nodo = ready.pop()
        print("Nodo para usarse: ", nodo)
        top_sorted.append(nodo)
        for vecino in grafo[nodo]:
            print(vecino, ":", incount[vecino], " - 1")
            incount[vecino] -= 1
            if incount[vecino] == 0:
                ready.append(vecino)
        print("Ready", ready)

    print("Orden Topológico:", top_sorted)

grafo = {
    'A': ['B', 'C'],
    'B': ['E', 'G'],
    'C': [],
    'D': ['B', 'E', 'I'],
    'E': ['G'],
    'F': ['E','I'],
    'G': ['H'],
    'H': [],
    'I': ['H']
}

topological_sort(grafo)
```

```
Indegree {'A': 0}

Ready ['A']
Indegree {'A': 0, 'B': 2}

Ready ['A']
Indegree {'A': 0, 'B': 2, 'C': 1}

Ready ['A']
Indegree {'A': 0, 'B': 2, 'C': 1, 'D': 0}

Ready ['A', 'D']
Indegree {'A': 0, 'B': 2, 'C': 1, 'D': 0, 'E': 3}

Ready ['A', 'D']
Indegree {'A': 0, 'B': 2, 'C': 1, 'D': 0, 'E': 3, 'F': 0}
```

```
Ready ['A', 'D', 'F']
Indegree {'A': 0, 'B': 2, 'C': 1, 'D': 0, 'E': 3, 'F': 0, 'G': 2}

Ready ['A', 'D', 'F']
Indegree {'A': 0, 'B': 2, 'C': 1, 'D': 0, 'E': 3, 'F': 0, 'G': 2, 'H': 2}

Ready ['A', 'D', 'F']
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  - 1
I : 2  - 1
Ready ['A', 'D']
Nodo para usarse:  D
B : 2  - 1
E : 2  - 1
I : 1  - 1
Ready ['A', 'I']
Nodo para usarse:  I
H : 2  - 1
Ready ['A']
Nodo para usarse:  A
B : 1  - 1
C : 1  - 1
Ready ['B', 'C']
Nodo para usarse:  C
Ready ['B']
Nodo para usarse:  B
E : 1  - 1
G : 2  - 1
Ready ['E']
Nodo para usarse:  E
G : 1  - 1
Ready ['G']
Nodo para usarse:  G
H : 1  - 1
Ready ['H']
Nodo para usarse:  H
Ready []
Orden Topológico: ['F', 'D', 'I', 'A', 'C', 'B', 'E', 'G', 'H']
```

3.2 Enunciado ejercicio 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.

```
grafo = {
    'A': ['B'],
    'B': ['A']
}

topological_sort(grafo)
```

```
Indegree {'A': 1}

Ready []
Indegree {'A': 1, 'B': 1}

Ready []
Orden Topológico: []
```

Si un grafo dirigido tiene un ciclo, el algoritmo de ordenamiento topológico no puede completarse correctamente ni producir un ordenamiento topológico válido. En este caso, la lista resultante, `top_sorted`, queda vacía porque los nodos 'A' y 'B' tienen ambos un grado de entrada de 1 (debido a las aristas B → A y A → B, respectivamente). Ningún nodo tiene un grado de entrada de 0, por lo que la lista `ready` permanece vacía. Como la condición del bucle `while (len(ready) > 0)` no se cumple, el bucle no se ejecuta y ningún nodo se agrega a `top_sorted`.

4. Conclusiones

- El ordenamiento topológico es fundamental en la representación de relaciones de precedencia en grafos dirigidos acíclicos. Este método facilita establecer un orden secuencial de tareas o eventos, asegurando que cada tarea se realice antes de que pueda comenzar otra relacionada con ella.
-

5. Declaración del Uso de IA

En este informe se utilizó IA para mejorar la coherencia, cohesión y sintaxis de las conclusiones.

6. Referencias Bibliográficas

«7.17. Ordenamiento topológico — Solución de problemas con algoritmos y estructuras de datos — runestone.academy». https://runestone.academy/ns/books/published/pythoned/Graphs/OrdenamientoTopologico.html.