Ordenação Topológica

...

História

O Algoritmo de Ordenação Topológica foi criado por Arthur B. Kahn, matemático e cientista de computação americano que trabalhou na Bell Labs(Nokia) e na IBM.

Publicou o algoritmo em 1962, no artigo intitulado “Topological sorting of large networks”

Criou-se então um algoritmo de ordenação topológica baseado em busca em profundidade(DFS), que é uma técnica comum e eficiente para encontrar a ordenação topológica de um grafo direcionado acíclico(DAG).

O que é Ordenação Topológica?

Ordenação Topológica é a linearização dos vértices de um grafo acíclico direcionado **(DAG)** de modo que cada aresta (u , v), u apareça antes que v.

Aresta u = Vértice de partida (ou origem) da aresta.

Aresta v = Vértice de chegada (ou destino) da aresta.

Exemplo: Se temos uma aresta (A, B), significa que há uma conexão direcionada do vértice A para o vértice B.

Problemas práticos

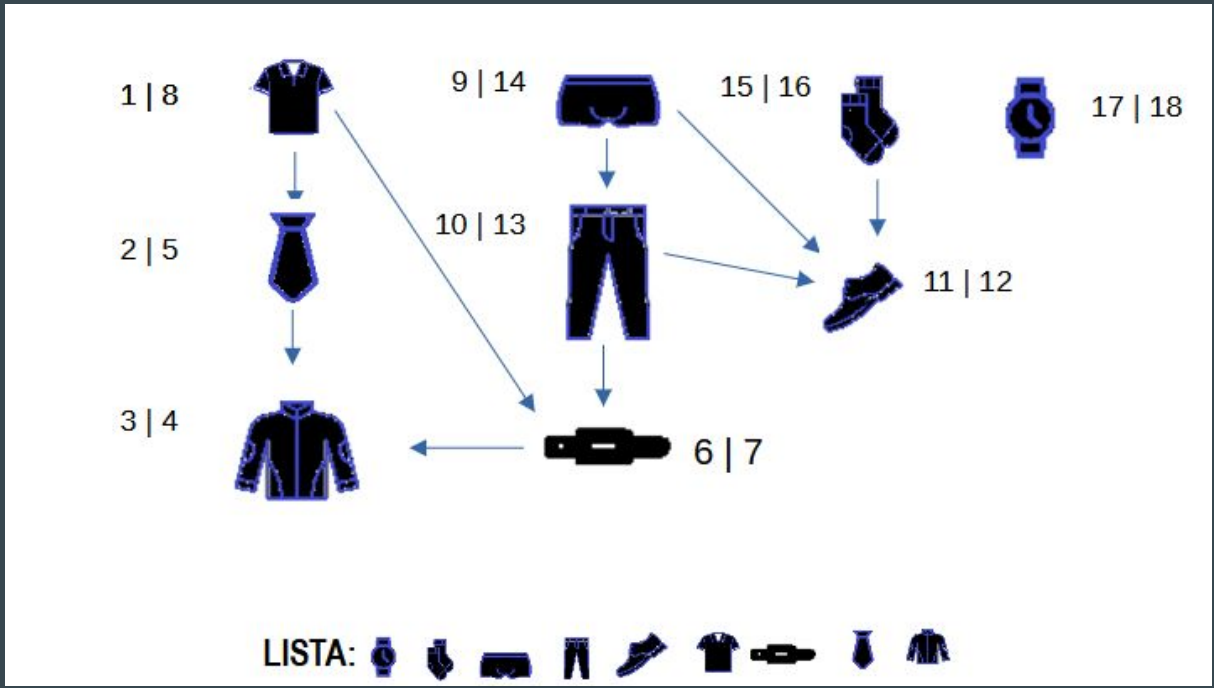
Alguns dos problemas mais comuns que foram atribuídos ao algoritmo de Kahn, são **compilação de arquivos** , onde em programas como o “make” as dependências entre arquivos são representadas por um grafo.

Em **gerenciamento de projetos** , tarefas podem depender de outras tarefas. A ordenação topológica ajuda a determinar a ordem na qual as tarefas devem ser realizadas.

Em **sistemas educacionais** , alguns cursos podem ter pré-requisitos. A ordenação topológica pode determinar uma sequência válida de cursos que um aluno pode seguir para concluir seu currículo.

Exemplo prático 1:

Exemplo envolvendo vestimentas de Roupas.



Exemplo prático 2:

Vamos supor que você está organizando um projeto de construção de uma casa, você tem uma lista de etapas para completar e algumas delas dependem da conclusão de outras.

Lista de tarefas

1 - Preparação do terreno

2 - Fundações

3 - Estrutura de madeira

4 - Instalações elétricas e hidráulicas

5 - Paredes e divisórias

6 - Telhado

7 - Janelas e portas

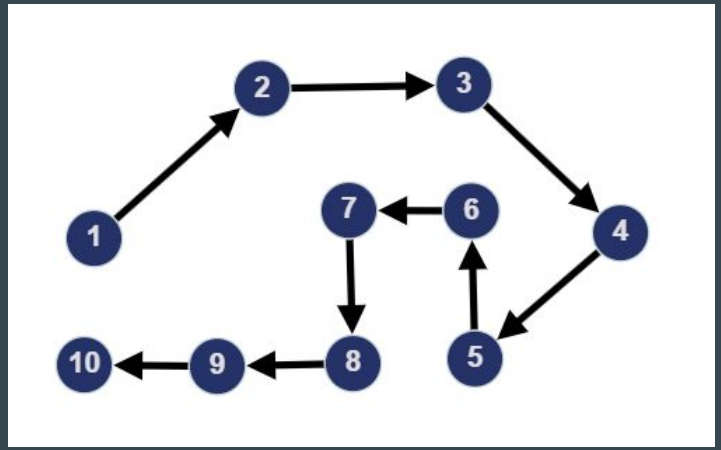
8 - Acabamentos interiores

9 - Pintura

10 - Paisagismo

O nosso objetivo é organizar as etapas de forma que cada uma seja executada após a conclusão das etapas das quais ela depende. Isso garantirá uma construção organizada e eficiente da casa.

Da mesma forma, o objetivo do algoritmo é criar uma lista na qual as tarefas devem ser executadas obedecendo uma ordem de dependência, podemos criar um grafo onde os vértices são as tarefas e as arestas são as dependências entre elas.

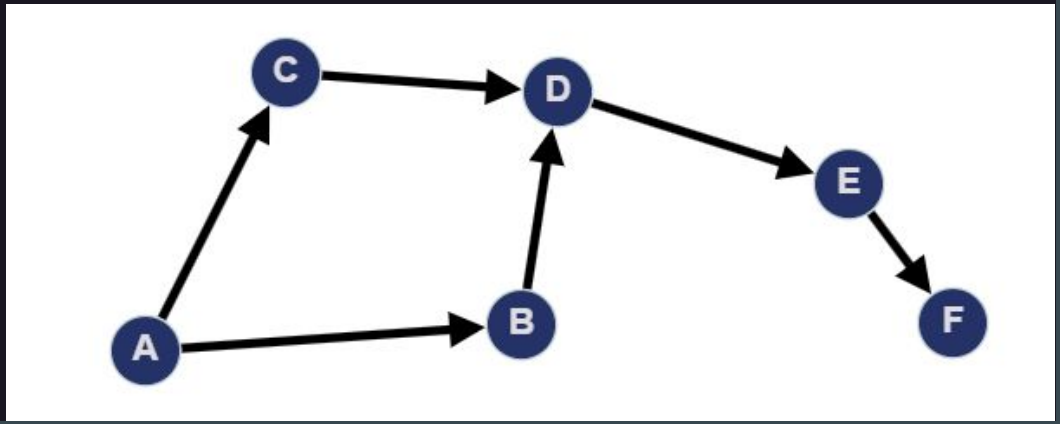


Exemplo de Algoritmo

```python
from collections import defaultdict, deque

class Graph:
    # Inicializa os vértices e a lista de adjacentes do mesmo
    def __init__(self, vertices):
        self.vertices = vertices
        self.adjList = defaultdict(list)

    # Função para adicionar um vértice
    def addVertex(self, v):
        self.adjList[v] = []

    # Função para adicionar uma aresta ao vértice
    def addEdge(self, v, w):
        self.adjList[v].append(w)
```

```python
    # Função para realizar a ordenação topológica no grafo
    def topologicalSort(self):
        inDegree = {v: 0 for v in self.adjList}  # Inicializa o grau de entrada de cada vértice como 0
        for v in self.adjList:
            for neighbor in self.adjList[v]: # Para cada vizinho do vértice, seu grau aumenta em 1
                inDegree[neighbor] += 1

        queue = deque([v for v in inDegree if inDegree[v] == 0])  # Fila de vértices com grau de entrada zero
        topologicalOrder = []

        # Enquanto ainda tiverem elementos para serem ordenados
        while queue:
            vertex = queue.popleft() # Remove e retorna o primeiro valor na fila de vértices de grau 0
            topologicalOrder.append(vertex) # Ordena o vértice na lista

            for neighbor in self.adjList[vertex]: # Diminui 1 do grau de cada vizinho do vértice
                inDegree[neighbor] -= 1
                if inDegree[neighbor] == 0: # Se o grau se tornou 0, o adiciona na fila
                    queue.append(neighbor)

        if len(topologicalOrder) != self.vertices:
            print("O grafo contém um ciclo e não pode ter uma ordenação topológica.")
        else:
            print("A ordenação topológica do grafo é:")
            print(" ".join(topologicalOrder))
```

```python
# Exemplo de uso
g = Graph(6)
vertices = ['A', 'B', 'C', 'D', 'E', 'F']

for vertex in vertices:
    g.addVertex(vertex)

g.addEdge('A', 'B')
g.addEdge('A', 'C')
g.addEdge('B', 'D')
g.addEdge('C', 'D')
g.addEdge('D', 'E')
g.addEdge('E', 'F')

g.topologicalSort()
```



Resultado da execução

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python

A B C D E F
```

Complexidade do Algoritmo

$O(n)$ onde $n = (V + A)$

V = número de vértices.

A = número de arestas.

Obrigado!