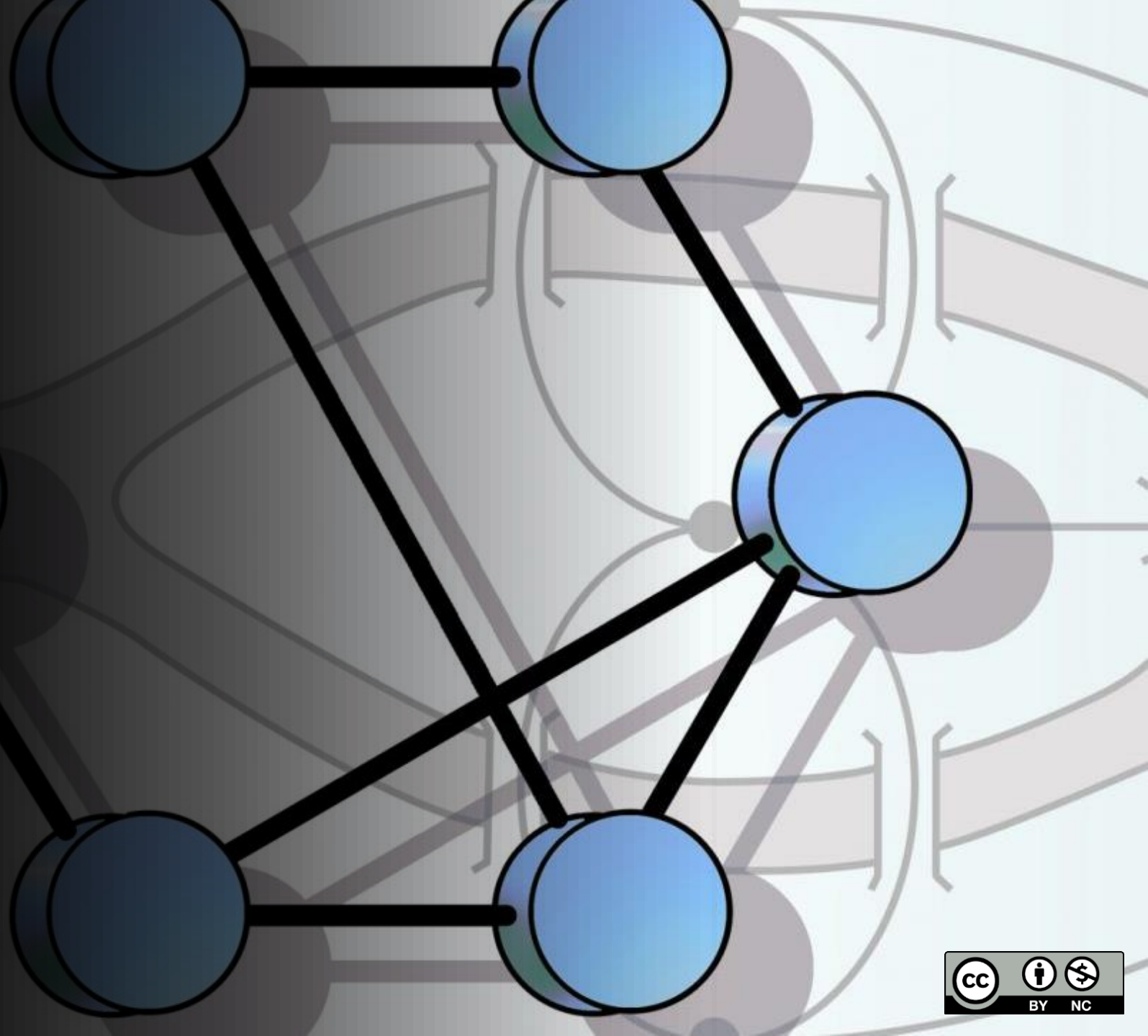
Teoria dos Grafos

Ordenação Topológica

Topological Sorting (TopSort)

Prof. André Kawamoto



- Motivação
- O que é uma Ordenação Topológica?
- Algoritmos para Ordenação Topológica
  - Ideia
  - Exemplos
  - Complexidade do Algoritmo

Motivação

- Considere as situações:
  - Um projeto é composto por diversas tarefas interdependentes. É preciso programar a execução das tarefas de forma a otimizar os recursos
  - Um aluno que precisa se matricular em várias disciplinas ao longo de um curso. As disciplinas possuem pré-requisitos, ou seja, algumas precisam ser feitas antes de outras.
- Como estabelecer essa ordem?

Uma Solução

- Converta a situação em um grafo
- Aplique uma Ordenação Topológica (Topological Sorting – TopSort)

O que é uma Ordenação Topológica?

- Uma ***ordenação topológica*** de um Grafo Acíclíco Direcionado (DAG) produz uma ordenação linear tal que, se (*u*, *v*) é uma aresta no DAG, então *u* aparece antes de *v* na ordenação linear.

O que é uma Ordenação Topológica?

- Mais Formalmente:
  - Uma relação de acessibilidade `R` é definida sobre os nós `x` e `y` de um DAG (`x R y`) **se e somente se** existe um caminho dirigido de `x` para `y`. Nesse caso, *R* é uma **ordem parcial**.
  - Uma Ordenação Topológica é uma extensão linear desta ordem parcial, isto é, **uma ordem total compatível** com a ordem parcial.

Como obter uma TopSort?

- Abordaremos duas maneiras de se obter uma Ordenação Topológica:
  - Busca em Profundidade
  - Algoritmo de Kahn

Algoritmo 1 TopSort

Usando DFS

DFS para TopSort

- Podemos modificar o algoritmo de Busca em Profundidade DFS para encontrar a Ordenação Topológica de um grafo.
- No algoritmo de DFS:
  1. Visitamos um vértice,
  2. imprimimos,
  3. chamamos recursivamente o DFS para seus vértices adjacentes.

DFS para TopSort

- Na Ordenação Topológica, usamos uma pilha.
- Em vez de imprimirmos o vértice imediatamente, primeiro chamamos recursivamente a DFS para todos os seus vértices adjacentes e o empilhamos.
- No fim, imprimimos o conteúdo da pilha.
- Nessa abordagem, um vértice é empilhado apenas quando todos os seus vértices adjacentes (e os vértices adjacentes deles, e assim por diante) já foram empilhados.

Algoritmo TopSort

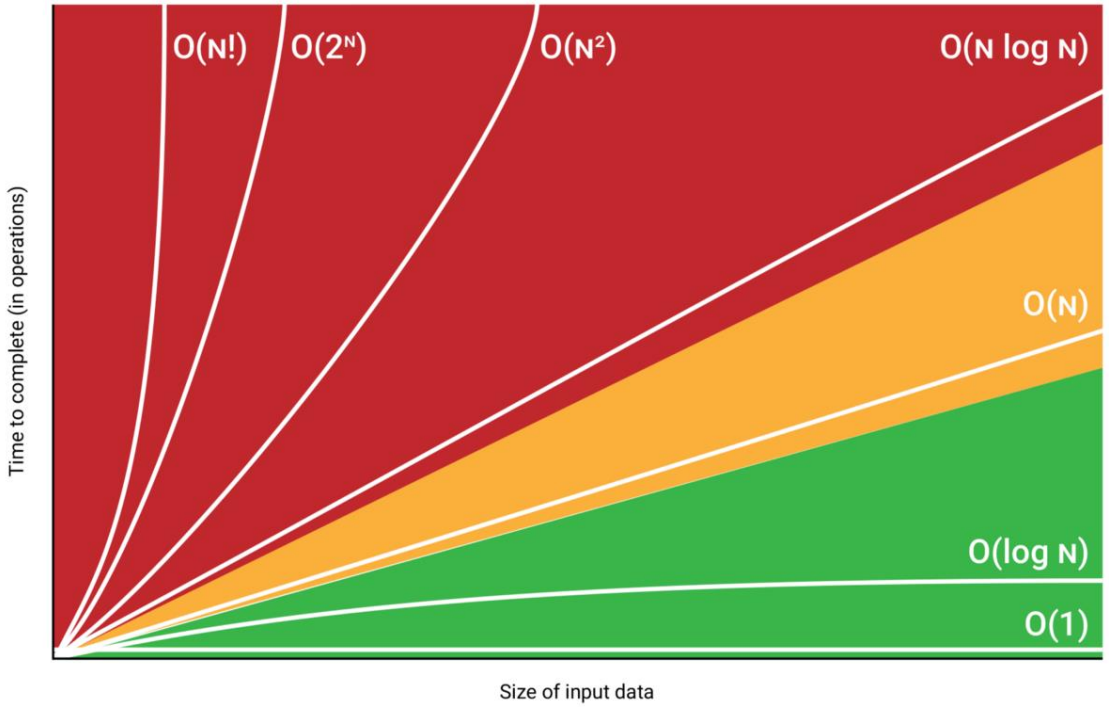
```
Algoritmo Ordenação Topológica

 function Topological-Sort(G)
   Para cada vértice v
      - chamar DFS(G) para calcular o
        tempo de finalização (v:f)
      - à medida que cada vértice é
        finalizado, inserir em uma
        pilha, inicialmente vazia
   devolver o conteúdo da pilha
```

Código TopSort em Python (geeksforgeeks)

```
#Funcao Recursiva usada por topologicalSort
def topologicalSortUtil(self,v,visited,stack):
    # marca o nó atual como visitado.
    visited[v] = True
    # itera todos os vertices adjacentes ao v
    for i in self.graph[v]:
        if visited[i] == False:
            self.topologicalSortUtil(i,visited,stack)
    # Insere o nó atual na pilha
    stack.insert(0,v)
```

```
# Funcao de Ordenacao Topologica. Usa a funcao
# recursive topologicalSortUtil()
def topologicalSort(self):
    # Mark all the vertices as not visited
    visited = [False]*self.V
    stack =[]
    # chama a funçao auxiliar recursive para guardar
    # a orddenaçao Topologica na pilha
    for i in range(self.V):
        if visited[i] == False:
            self.topologicalSortUtil(i,visited,stack)
    # Print contents of the stack
    print(stack)
```



Complexidade do Algoritmo TopSort

- Como vimos, a DFS tem Complexidade O (V+E)
- Alterações:
  - Empilhar o vértice quando é finalizado (tempo constante)
  - Exibir o conteúdo da pilha: O (V)

- Logo, esse algoritmo de TopSort tem complexidade **O (V+E)**

Exemplo TopSort

- Uma simulação do funcionamento desse algoritmo é dada no vídeo disponível no canal Geeksforgeeks: https://www.youtube.com/watch?v=Q9PIxaNGnig

- Um simulador interativo pode ser usado no site da Universidade de San Francisco:
- https://www.cs.usfca.edu/~galles/visualization/TopoSortDFS.html

Algoritmo 1 de Kahn

Remoção de nós-fonte

Algoritmo de Kahn

- O algoritmo de Kahn foi proposto em 1962 para encontrar uma TopSort em um Grafo Direcionado Acíclico (DAG), e se baseia em um fato:
- **Todo DAG possui pelo menos 1 vértice com grau de entrada 0 (zero), e pelo menos 1 vértice com grau de saída 0 (zero)**

- A prova desse fato é bastante simples e é dada a seguir

Algoritmo de Kahn

- Em um DAG não há ciclos, logo todos os caminhos terão comprimento finito.
- Seja ‘S’ o caminho mais longo entre os nós `u` (origem) e `v` (destino).
- Uma vez que ‘S’ é o caminho mais longo, não pode haver aresta de entrada em `u`, tampouco aresta de saída de `v`; se isso acontecer, ‘S’ não seria o caminho mais longo.
- Portanto,
  - grau de entrada de `u` = 0
  - grau de saída de `v` = 0

Algoritmo de Kahn

Passo 1: Calcular o grau de entrada (número de arestas que chegam) para cada vértice do DAG e inicializar o Contador de nós visitados como 0 (zero).

Passo 2: Escolher todos os vertices com grau de entrada 0 (zero) e adicionar em uma fila

Passo 3: Remover um vértice da fila e então

- Incrementar de 1 o contador de vértices visitados
- Decrementar de 1 o grau de entrada de todos os vértices adjacentes.
- Se o grau de entrada de algum vértice adjacente foi reduzido a 0 (zero), adicioná-lo na fila.

Passo 4: Repetir o Passo 3 até que a fila esteja vazia

Passo 5: Se o contador de vértices visitados não é igual ao número de vertices do grafo, então não é possível estabelecer uma ordenação topológica nesse grafo.

Código Kahn em Python (geeksforgeeks)

```
# função para Ordenação Topologica.
def topologicalSort(self):
    # cria um vetor para guardar o grau de entrada de todos os vertices. Inicializa com zeros
    in_degree = [0]*(self.V)

    # Percorre as listas de adjacência para calcular e preencher com os graus de entrada dos vértices
    # Esse passo tem complexidade de tempo de O(V + E)
    for i in self.graph:
        for j in self.graph[i]:
            in_degree[j] += 1
    #continua . . .
```

$O(V + E)$

Código Kahn em Python (geeksforgeeks)

```
# continuação
# Cria uma fila e insere todos os vertices que possuem grau de entrada igual a zero
queue = []
for i in range(self.V):
    if in_degree[i] == 0:
        queue.append(i)
# Inicializa o Contador de vertices visitados
cnt = 0
# cria o vetor para guardar o resultado (uma ordenacao topologica dos vertices)
top_order = []

#continua...
```

$O(V)$

Código Kahn em Python (geeksforgeeks)

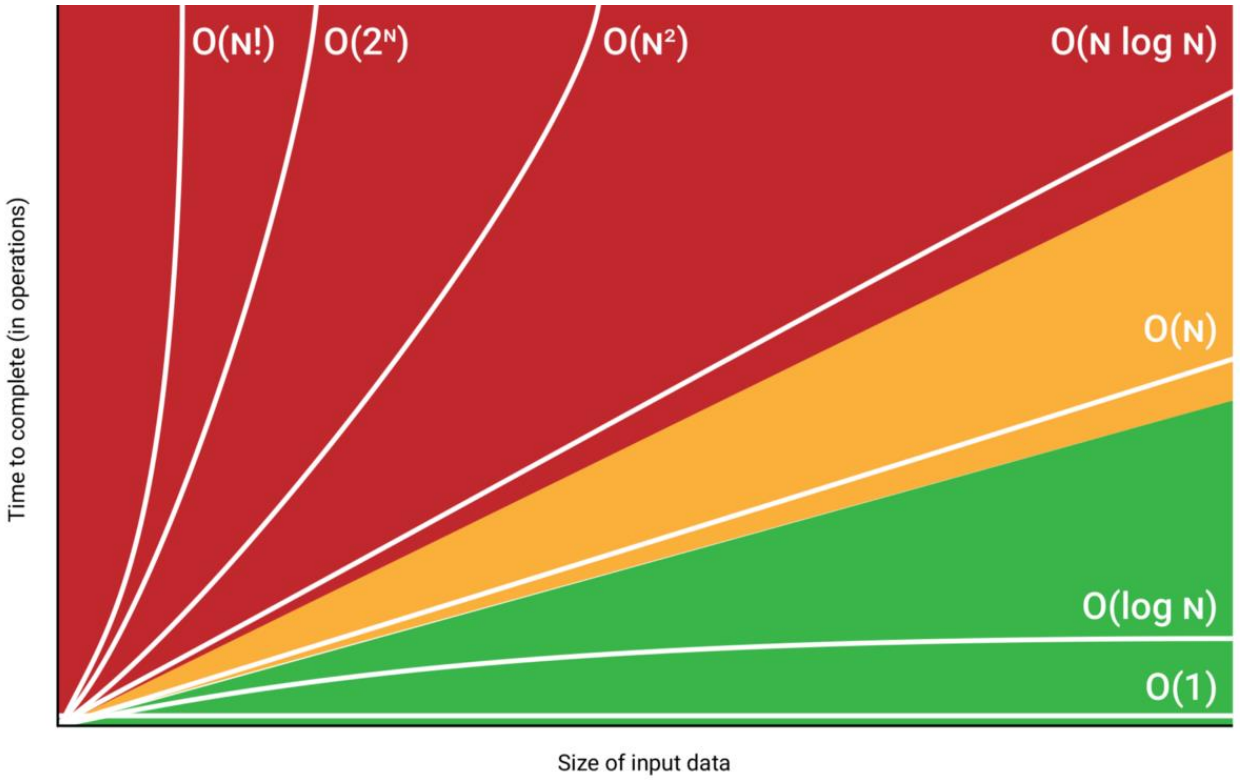
$O(V + E)$

```
# retira os vertices um de cada vez da fila, e insere os adjacentes, se o grau de entrada se tornou zero
while queue:
    # remove o primeiro da fila e adiciona na ordenacao topologica
    u = queue.pop(0)
    top_order.append(u)
    # itera em todos os adjacentes do vertice removido e decrementa o grau de entrada de 1
    for i in self.graph[u]:
        in_degree[i] -= 1
        # se o grau de entrada se tornou zero, adiciona na fila
        if in_degree[i] == 0:
            queue.append(i)
    cnt += 1  #incrementa o contador de visitados, continua...
```

Código Kahn em Python (geeksforgeeks)

```
# Verifica se havia algum ciclo
    if cnt != self.V:
        print ("There exists a cycle in the graph")
    else :
        # Imprime a ordenacao topologica
        print (top_order)
```

Complexidade do algoritmo de Kahn



- A complexidade desse algoritmo é O (V+E)
  - O laço principal (while) é executado V vezes – para todos os vértices do grafo
  - Internamente, o laço é executado E vezes

- A condição if final, se verdadeira, indica que o grafo de entrada não era sem ciclos, logo não é possível obter uma ordenação topológica

Exemplos Kahn

- Um simulador interativo pode ser usado no site da Universidade de San Francisco:
- https://www.cs.usfca.edu/~galles/visualization/TopoSortIndegree.html

Referências desse Material

- CORMEN, Thomas. **Desmistificando algoritmos**. Elsevier Brasil, 2017.
- Kahn, A., 1962. **Topological sorting of large networks**. *Communications of the ACM*, 5(11), pp.558-562.
- ROSEN, Kenneth H. **Matemática discreta e suas aplicações**. Grupo A Educação, 2009.
- Topological Sorting - GeeksforGeeks. GeeksforGeeks. Disponível em: <https://www.geeksforgeeks.org/topological-sorting/>. Acesso em: 9 jun. 2020.