Instituto Politécnico

Ordenação Topológica e Componentes Fortemente Conexas

IPRJ01-10762 Matemática Discreta II

Gustavo Barbosa Libotte

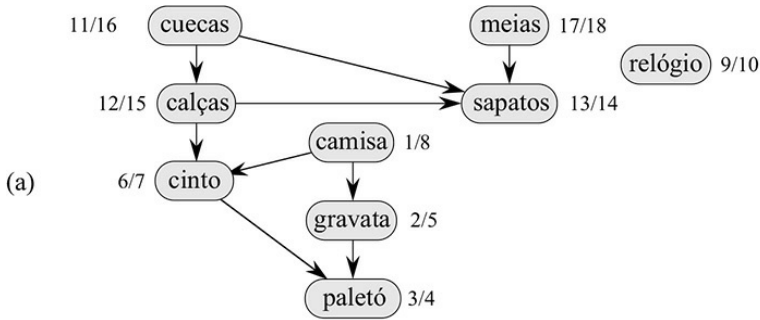
Aula 8

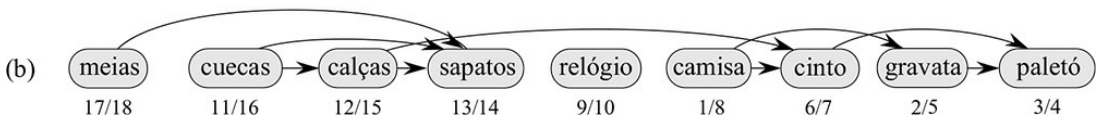
Ordenação Topológica

Instituto Politécnico

- Nesta aula vamos ver como podemos usar **busca em profundidade** para executar uma **ordenação topológica** de um **grafo acíclico dirigido**, ou "gad", como, às vezes, é chamado.
- Uma ordenação topológica de um gad $G = (V, E)$ é uma ordenação linear de todos os seus vértices, tal que se G contém uma aresta (u, v), então u aparece antes de v na ordenação.
- **Se o grafo contém um ciclo, nenhuma ordenação topológica é possível.**
- Podemos ver uma ordenação topológica de um grafo como uma ordenação de seus vértices ao longo de uma linha horizontal de modo tal que todas as arestas dirigidas vão **da esquerda para a direita**.
- Muitas aplicações usam grafos acíclicos dirigidos para indicar precedências entre eventos.
- A figura a seguir mostra um exemplo lúdico sobre como vestir certas peças de roupa antes de outras (por exemplo, meias antes de sapatos). Outros itens podem ser colocados em qualquer ordem (por exemplo, meias e calças).

Ordenação Topológica





Ordenação Topológica

- Uma aresta dirigida (u, v) no gad da figura (a) indica que a peça de roupa u deve ser vestida antes da peça v.
- Portanto, uma **ordenação topológica** desse gad dá uma ordem para o processo de se vestir.
- A figura (b) mostra o gad topologicamente ordenado como uma ordenação de vértices ao longo de uma linha horizontal tal que todas as arestas dirigidas vão da esquerda para a direita.
- O seguinte algoritmo simples ordena topologicamente um gad:

TOPOLOGICAL-SORT (G)

1: chamar DFS (G) para calcular o tempo de término $v.f$ para cada vértice v
2: à medida que cada vértice é terminado, inserir o vértice à frente de uma lista ligada
3: **return** a lista ligada de vértices

Ordenação Topológica

- Uma aplicação clássica da ordenação topológica em teoria de grafos é a resolução de dependências entre tarefas em um projeto.
- Suponha que você esteja trabalhando em um projeto que possui várias tarefas interdependentes e precisa determinar uma ordem de execução que respeite essas dependências.
- Vamos considerar um exemplo prático para ilustrar essa aplicação.
- Suponha que você esteja desenvolvendo um *software* e precisa determinar a ordem de compilação dos módulos do projeto, levando em conta as dependências entre eles.
- Cada módulo representa uma tarefa a ser executada, e as dependências entre os módulos são representadas pelas dependências entre as tarefas.

Ordenação Topológica

- Vamos supor que temos os seguintes módulos e suas dependências:
  - Módulo A: —
  - Módulo B: A
  - Módulo C: A
  - Módulo D: B, C
  - Módulo E: D
- Nesse caso, a ordem correta de compilação seria: A, B, C, D, E.
- Podemos resolver esse problema utilizando a **ordenação topológica**. Cada módulo é representado como um vértice no grafo, e as dependências são representadas como arestas direcionadas entre os vértices.

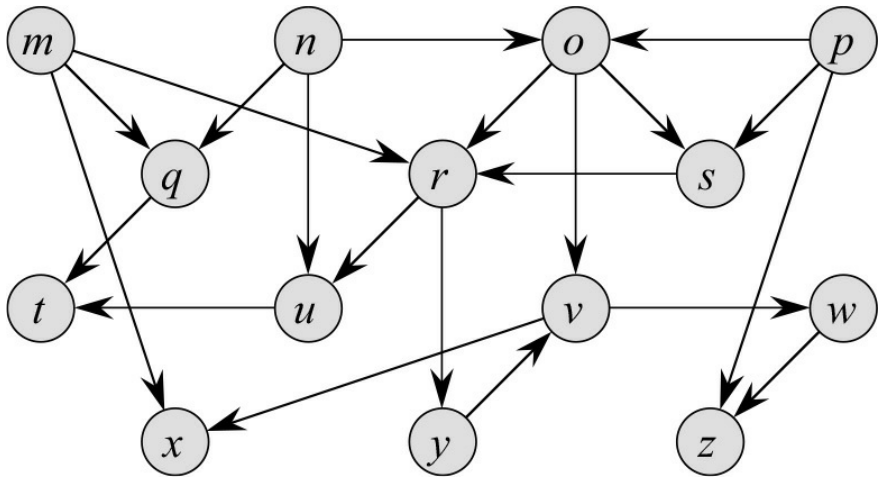
Ordenação Topológica

- Podemos executar uma ordenação topológica no tempo $\Theta(V + E)$, já que a busca em profundidade demora o tempo $\Theta(V + E)$ e que inserir cada um dos $|V|$ vértices à frente da lista ligada leva o tempo $O(1)$.

Ordenação topológica em Python

```
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adjacency_list = [[] for _ in range(vertices)]
        self.visited_order = []

    def add_edge(self, u, v):
        self.adjacency_list[u].append(v)

    def topological_sort(self):
        colors = ['WHITE'] * self.vertices
        self.visited_order = []
        result = []

        for u in range(self.vertices):
            if colors[u] == 'WHITE':
                self.DFS_visit(u, colors, result)

        return result[::-1]

    def DFS_visit(self, u, colors, result):
        colors[u] = 'GRAY'

        for v in self.adjacency_list[u]:
            if colors[v] == 'WHITE':
                self.DFS_visit(v, colors, result)

        colors[u] = 'BLACK'
        result.append(u)
```

Ordenação Topológica



Componentes Fortemente Conexas

Instituto Politécnico

- Agora, consideraremos uma aplicação clássica de busca em profundidade: a decomposição de um **grafo dirigido** em suas **componentes fortemente conexas**.
- Vamos fazer isso usando duas buscas em profundidade. Muitos algoritmos que funcionam com grafos dirigidos começam por uma decomposição desse tipo.
- Após a decomposição do grafo em componentes fortemente conexas, tais algoritmos são executados separadamente em cada uma delas e combinados em soluções de acordo com a estrutura das conexões entre componentes.

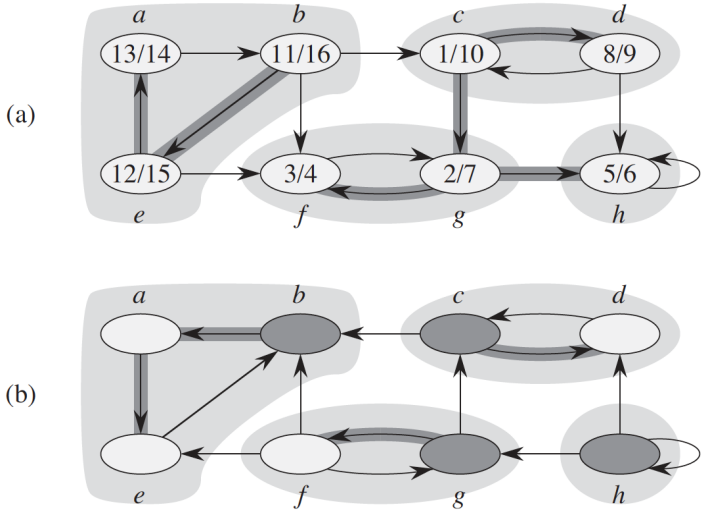
Componente Fortemente Conexa

Uma componente fortemente conexa de um grafo dirigido $G = (V, E)$ é um conjunto máximo de vértices $C \subseteq V$ tal que, para todo par de vértices u e v em C, temos $u \rightsquigarrow v$ e $v \rightsquigarrow u$; isto é, u pode ser alcançado a partir do vértice v e vice-versa.

Componentes Fortemente Conexas

- Nosso algoritmo para encontrar componentes fortemente conexas de um grafo $G = (V, E)$ usa o transposto de G, que é definida como o grafo $G_\top = (V, E_\top)$, onde $E_\top = \{(u, v) : (v, u) \in E\}$.
- Isto é, $E_\top$ consiste nas arestas de G com suas direções invertidas.
- Dada uma representação por lista de adjacências de G, o tempo para criar $G_\top$ é $O(V + E)$.
- É interessante observar que G e $G_\top$ têm exatamente as mesmas componentes fortemente conexas: u e v, podem ser alcançados um a partir do outro em G se e somente se puderem ser alcançados um a partir do outro em $G_\top$.
- A figura a seguir mostra o transposto do grafo na figura a seguir, com as componentes fortemente conexas sombreadas.

Componentes Fortemente Conexas



Componentes Fortemente Conexas

Instituto Politécnico

- O algoritmo de tempo linear (isto é, de tempo $\Theta(V + E)$) apresentado a seguir calcula as componentes fortemente conexas de um grafo dirigido $G = (V, E)$ usando duas buscas em profundidade, uma em G e uma em $G_\top$.

  STRONGLY-CONNECTED-COMPONENTS (G)

  1: chamar DFS(G) para calcular tempos de término $u.f$ para cada vértice u
  2: calcular $G_\top$
  3: chamar DFS $(G_\top)$ mas, no laço principal de DFS, considerar os vértices em ordem decrescente de $u.f$ (como calculado na linha 1)
  4: dar saída aos vértices de cada árvore na floresta em profundidade formada na linha 3 como uma componente fortemente conexa separada

- Considerando vértices na segunda busca em profundidade em ordem decrescente dos tempos de término que foram calculados na primeira busca em profundidade, estamos, em essência, visitando os vértices do grafo de componentes (cada um dos quais corresponde a uma componente fortemente conexa de G) em sequência ordenada topologicamente.

- Uma aplicação clássica dos componentes fortemente conexos em teoria de grafos é a identificação de comunidades ou grupos fortemente relacionados em uma rede social.
- Os componentes fortemente conexos podem representar grupos de pessoas que possuem conexões densas entre si, indicando uma forte interação e influência mútua.
- Vamos considerar um exemplo prático para ilustrar essa aplicação. Suponha que você esteja analisando uma rede social e deseja identificar grupos de amigos que interagem intensamente entre si.
- Para isso, você pode usar os componentes fortemente conexos do grafo da rede social para encontrar esses grupos.

Componentes Fortemente Conexas
Atividade prática

- Suponha que existam as seguintes ligações entre amigos:
  - (’Alice’, ’Bob’)
  - (’Bob’, ’Carol’)
  - (’Carol’, ’Alice’)
  - (’Dave’, ’Eve’)
  - (’Eve’, ’Frank’)
  - (’Frank’, ’Dave’)
  - (’Grace’, ’Heidi’)
  - (’Heidi’, ’Grace’)
  - (’Isabel’, ’Jack’)
  - (’Jack’, ’Isabel’)
- Teríamos, portanto, os seguintes grupos de amigos, identificados como componentes fortemente conexos:

  {’Carol’, ’Alice’, ’Bob’}, {’Frank’, ’Dave’}, {’Grace’, ’Heidi’} e {’Jack’, ’Isabel’}

- Cada conjunto de nós em um grupo representa um grupo de amigos que interagem fortemente entre si.
- Essa abordagem pode ser útil para descobrir comunidades ou **grupos coesos em uma rede social**, ajudando a entender melhor os relacionamentos e interações entre os indivíduos.

Componentes Fortemente Conexas
Atividade prática

Instituto Politécnico