Ordenação topológica

Alexandre Freitas
Lewis Guilherme

22 de junho de 2023

1 Introdução

Trabalho de implementação de um programa que processa um grafo direcionado acíclico e retorna uma ordenação topológica desse grafo.

2 O problema

O problema da ordenação topológica de um grafo direcionado acíclico se resume em gerar uma permutação dos vértices do grafo tal que não haja vértice cujo índice na permutação seja menor que de qualquer de um de seus ancestrais, ou seja, o sentido dos arcos é respeitado. De maneira mais objetiva, se uma permutação de um grafo G respeita a condição: $i < j$ para todo (i, j) $\in$ A(G), então essa permutação é uma ordenação topológica de G.

Existem grafos direcionados que não admitem uma ordenação topológica, que são grafos que possuem ciclos direcionados. Na seção de implementação, será discutido como foi possível identificar quando acontece esse caso.

3 Implementação

A linguagem utilizada para implementação foi *python*, por conta de sua simplicidade e facilidade em utilizar suas estruturas de dados. Implementamos uma classe grafo genérica que utiliza uma matriz de adjacência para representar o grafo em memória. No início do programa, o arquivo *.dot* é aberto e lido linha por linha, de modo a identificar os vértices e os arcos. Tendo isso, uma instância da classe grafo é criada, passando os vértices e arcos e indicando que o grafo será direcionado.

Existem algumas maneiras de se ordenar os vértices de um grafo direcionado. A maneira escolhida neste trabalho é baseada em uma busca em profundidade. Inicialmente, são criadas 3 variáveis: um vetor para armazenar a ordenação topológica, uma pilha para realizar a busca em profundidade e uma estrutura de dados chave e valor para guardar o grau de entrada dos vértices sendo explorados, pois esses serão alterados conforme as iterações acontecem.

As variáveis são:

- zerodeg: Pilha com vértices a serem explorados
- indegree_map: Estrutura chave e valor que guarda o grau de entrada dinâmico dos vértices
- sorted_list: Guarda os vértices em ordem topológica

O primeiro passo do algoritmo itera sobre todos os vértices do grafo, associando cada vértice com seu grau de entrada original na variável *indegree_map* e coloca todos os vértices de grau de entrada zero (fontes) na pilha. Isso é possível pois todo grafo direcionado acíclico possui ao menos uma fonte, essa propriedade é importante para o funcionamento de todo o algoritmo.

A busca então é iniciada, removendo o primeiro elemento da pilha, colocando-o na última posição do vetor de ordem topológica. Depois disso, passa-se por todos os vizinhos de saída do vértice corrente, diminui seu grau de entrada em 1 na variável *indegree_map* e, caso seu grau depois do decremento seja zero, coloca-se esse vértice na pilha *zerodeg*. A busca se repete até que a pilha esteja vazia.

Caso o grafo de entrada seja um grafo com ciclo direcionado, o vetor *sorted_list* terá menos elementos do que o número de vértices do grafo. Se esse for o caso, o algoritmo devolve uma mensagem dizendo "O grafo possui ciclo direcionado", pois esses não admitem ordenação topológica.

4 Exemplos de entrada e suas saídas

4.1 Grafo direcionado acíclico

```
digraph graphname {
    1 -> 4 -> 2
    1 -> 2
}
```

Figura 1: Grafo direcionado acíclico

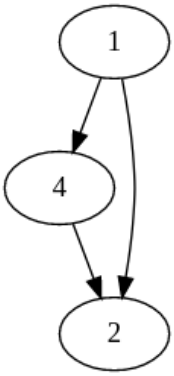

Figura 2: Desenho - Grafo direcionado acíclico

Saída:
1 4 2

4.2 Grafo direcionado acíclico desconexo

```
digraph graphname {
    a -> b -> c;
    b -> d;
    c -> d;
    b -> e;
    f -> j
}
```

Figura 3: Grafo direcionado acíclico desconexo

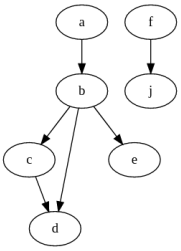

Figura 4: Desenho - Grafo direcionado acíclico desconexo

Saída:
f j a b c d e
ou
a b c d e f j

4.3 Grafo direcionado cíclico

```
digraph graphname {
    1 -> 4 -> 2 -> 1
}
```

Figura 5: Grafo direcionado cíclico

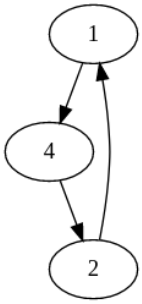

Figura 6: Desenho - Grafo direcionado cíclico

Saída:
Grafo possui ciclo direcionado