Ordenação Topológica de Grafos Direcionados Acíclicos usando Busca em Profundidade (DFS)

Pedro H. Kochinski
GRR20206144
phks20@inf.ufpr.br

Vinicius Mioto
GRR20203931
vm20@inf.ufpr.br

***Abstract*—A ordenação topológica é uma operação comum em teoria dos grafos, que consiste em determinar uma ordem linear dos vértices de um grafo direcionado acíclico (GDA). Neste relatório, abordamos o problema da ordenação topológica utilizando o algoritmo de busca em profundidade (DFS). Apresentamos uma implementação em Python para representar o grafo, realizar a busca em profundidade e obter a pós-ordem dos vértices.**

I. Introdução

A ordenação topológica em teoria dos grafos permite estabelecer uma ordem linear dos vértices de um GDA. Esse problema possui diversas aplicações práticas, como o planejamento de tarefas em projetos, a resolução de dependências em sistemas de software e a detecção de ciclos em grafos direcionados. Neste relatório, apresentamos uma abordagem baseada no algoritmo de busca em profundidade (DFS) para resolver o problema da ordenação topológica em um GDA.

II. Detalhes da Implementação

A implementação do algoritmo de ordenação topológica utilizando DFS foi realizada em Python e utilizamos as bibliotecas NetworkX e Pygraphviz para representar o GDA. A implementação é composta pelas seguintes etapas:

A. Representação do Grafo DOT

O grafo de entrada é lido a partir da entrada padrão utilizando a biblioteca NetworkX e Pygraphviz. O formato DOT contém a descrição dos vértices e arestas do GDA.

B. Ordenação Topológica

O algoritmo de ordenação topológica encapsula o algoritmo de busca em profundidade. Dessa forma, no início do algoritmo, são definida as estruturas de dados utilizadas um conjunto de vertices `visitados` e uma lista chamada `ordenação`. Dessa forma, para cada vértice pertencente aos vértices do grafo, é verificado se ele está ou não nos vertices do conjunto visitados - semelhante aos ”estados” dos vértices visto em aula - caso não foram visitados, então é chamado a busca em largura para esse vértice.

Após a execução recursiva da busca em profundidade para cada vertice pertencente ao grafo, a ordenação topológica é obtida no vetor `ordenacao`.

```
def ordenacao_topologica_pos_ordem(grafo):
    visitados = set()
    ordenacao = []
    for vertice in grafo.nodes():
        if vertice not in visitados:
            dfs_pos_ordem(grafo, vertice, visitados,
     ordenacao)
    return ordenacao
```

Listing 1. Algoritmo de ordenação topológica

C. Busca em Profundidade (DFS)

A função `dfs_pos_ordem` realiza a busca em profundidade no grafo. Essa função visita cada vértice não visitado, explora seus vizinhos e armazena a pós-ordem dos vértices no **início** da lista `ordenacao`. Assim, obtendo a ordenação topológica

```
def dfs_pos_ordem(grafo, vertice, visitados,
    ordenacao):
    visitados.add(vertice)
    for vizinho in grafo.successors(vertice):
        if vizinho not in visitados:
            dfs_pos_ordem(grafo, vizinho, visitados,
     ordenacao)
    ordenacao.insert(0, vertice)
```

Listing 2. Algoritmo de busca em profundidade

III. Instruções de Execução

Para executar o programa, deve ser seguido os seguintes passos:

```
$ make
$ ./toposort < entrada.txt
```

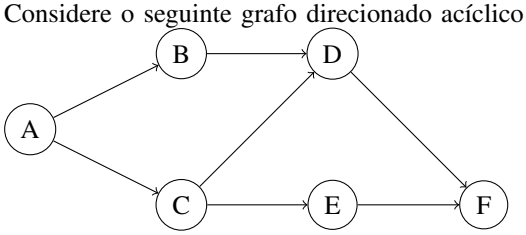
Onde `entrada.txt` é um arquivo com a entrada no formato descrito pelo enunciado do trabalho, nesse caso, um grafo no formato DOT. Por fim, o resultado da ordenação topológica será salvo no arquivo `saida.txt`.

IV. Exemplos

Nesta seção, apresentamos alguns exemplos adicionais para demonstrar a funcionalidade da nossa implementação de ordenação topológica.

A. Exemplo 1

Considere o seguinte grafo direcionado acíclico (DAG):



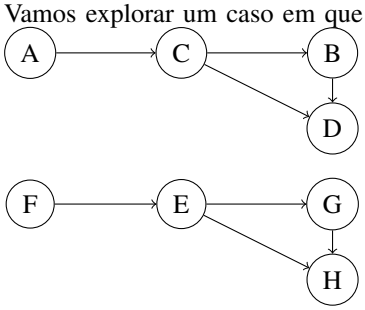
Que pode ser representado no formato DOT da seguinte forma:

```
digraph DAG {
    A -> {B, C};
    B -> D;
    C -> {D, E};
    D -> F;
    E -> F;
}
```

Após aplicar o algoritmo `dfs_pos_ordem`, obtemos a sequência de vértices: `F, D, B, E, C, A`. Agora, o algoritmo `ordenacao_topologica_pos_ordem` obtem a ordenação topológica do DAG ao inverter a sequência anterior, gerando a seguinte sequência de vértices: `A, C, E, B, D, F`.

B. Exemplo 2

Vamos explorar um caso em que o grafo não é conexo:



Agora, representado na forma DOT, temos:

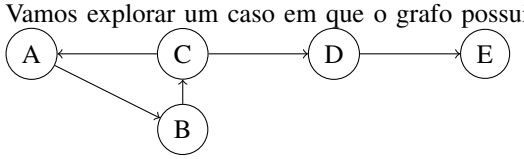
```
digraph DAG {
    A -> C;
    C -> {B, D};
    B -> D;
    F -> E;
    E -> {G, H};
    G -> H;
}
```

Neste grafo, temos duas componentes conectadas separadas: a primeira com os vértices `A, B, C, D`, e a segunda com os vértices `E, F, G, H`.

Neste caso, a ordenação é feita para cada componente do grafo. Para a primeira componente ordenação topológica é: `A, C, B, D`. Já para a segunda componente a ordenação topológica é: `F, E, G, H`.

C. Exemplo 3

Vamos explorar um caso em que o grafo possui um ciclo:



Representado por:

```
digraph DAG {
    A -> B;
    B -> C;
    C -> A;
    C -> D;
    D -> E;
}
```

Observe que é impossível estabelecer uma ordem linear entre os vértices `A, B, C` devido à presença do ciclo. Neste caso, a ordenação topológica não é possível.