CI1065 - Algoritmos e Teoria dos Grafos - Primeiro Trabalho

GRR20206872 - Isadora Botassari
GRR20203954 - Victor Ribeiro Garcia

1. Introdução

Neste trabalho, o problema consiste em encontrar uma ordenação topológica para um dado grafo direcionado acíclico G. De acordo com o enunciado:

"Uma ordenação topológica de um grafo direcionado G é uma permutação (v1, ... , vn) de V(G) que "respeita a direção dos arcos" de G, isto é, i < j, para todo (vi , vj) ∈ A(G)."

Uma ordenação topológica em grafos é uma ordenação linear dos vértices de um grafo direcionado acíclico, que respeita às relações de precedência definidas pelas arestas direcionadas. Em outras palavras, é uma maneira de colocar os vértices em uma sequência linear de tal forma que, se existe uma aresta direcionada do vértice A para o vértice B, então o vértice A aparece antes do vértice B na ordenação.

2. Algoritmo

Para obter a ordenação topológica, foi utilizada uma busca em profundidade, seguindo o algoritmo visto e explicado em aula.

O algoritmo "Ordena(G)" começa inicializando todos os vértices do grafo com um estado 0 (não visitado). Em seguida, se o estado de v for 0, chama a função "Ordena(G, v)" para processar todos os vértices adjacentes a v.

Figura 1 - ***Algoritmo de Ordenação Topológica***

```
Ordena(G)
Para cada v ∈ V(G)
    v.estado ← 0
G.l ← lista vazia
Para cada v ∈ V(G)
    Se v.estado = 0
        Ordena(G, v)
```

```
Ordena(G, r)
r.estado ← 1
Para cada v ∈ Γ⁺_G(r)
    Se v.estado = 0
        Ordena(G, v)
acrescente r ao início de G.l
r.estado ← 2
```

Fonte: Material Professor Murilo V. G. da Silva

3. Implementação

A entrada lida é um arquivo .dot que informa os vértices e arestas de um certo grafo G utilizando a biblioteca *graphviz*. Durante a leitura, é verificado se as arestas fornecidas são direcionadas (ou seja, se são arcos); caso não sejam, uma mensagem de erro é emitida e o programa é encerrado. Se o grafo G é direcionado, a partir da entrada, é construído um vetor de vértices e um vetor de arcos relativos a esse grafo.

Código 2 - *Implementação da leitura do arquivo .dot*

```
// ler grafo do .dot
graph = agread(stdin, 0);

if (!graph) {
    fprintf(stderr, "Erro ao ler arquivo.\n");
    return 1;
}

if (agisdirected(graph) == false) {
    fprintf(stderr, "Erro: grafo fornecido não é um GDA.\n");
    return 1;
}
```

Ainda, cada vértice v possui associado a si um vetor de índices dos vértices pertencentes à sua vizinhança de saída. A busca em profundidade foi implementada com as funções ordenaEnvelope e ordenaFunc. Tal como foi visto em aula, a primeira é uma função envelope que verifica os vértices que ainda não foram processados, e para cada um deles, chama a função recursiva que executa a busca.

Código 3 - *Implementação da função que realiza a busca em profundidade*

```
void ordenaFunc(vector<Vertice>* vertices, Vertice* v) {

    v->estado = 1;
    t++;
    v->pre = t;

    vector<int>::iterator it;
    int index;

    for (int i = 0; i < (v->lista_vizinhos.size()); i++) {
        index = v->lista_vizinhos.at(i);
        //printf("%d\n ", index);

        if (vertices->at(index).estado == 0)
            ordenaFunc(vertices, &vertices->at(index));
    }
    ordenacao.push_back(v);
    v->estado = 2;
    t++;
    v->pos = t;
}
```

Ainda, durante a busca em profundidade, são calculados os índices de pré-ordem e pós-ordem de cada vértice. Esses índices são utilizados tanto para definir a ordenação topológica quanto para decidir se o grafo G é ou não um grafo direcionado cíclico. No primeiro item, de acordo com o Teorema 81:

"O reverso da pós-ordem de uma floresta direcionada resultante de uma busca em profundidade em um grafo direcionado acíclico G, é uma ordenação topológica de G."

Assim, os vértices são inseridos num vetor de ordenação topológica no momento do cálculo da pós-ordem de cada um. No segundo item, para cada vértice u do grafo G e para vértice v que pertence à vizinhança de saída de u, se existe uma aresta u→ v que é uma aresta de retorno, G possui ciclo, e portanto, não é um grafo direcionado acíclico. Essa propriedade é verificada utilizando o Teorema 78:

"F: floresta direcionada resultante de uma busca em profundidade sobre o grafo direcionado.

G. v.pre e v.pos: índices de pré-ordem e pós-ordem computados.

Arco (u, v), com relação a F é arco de retorno: se e somente se v.pre < u.pre < u.pos < v.pos."

Código 4 - *Implementação da função que verifica se o grafo G possui ciclo*

```
bool temArcoRetorno(vector<Vertice> vertices_grafo) {

    vector<int>::iterator it;

    // p/ cada vertice u do grafo G
    for (int i = 0; i < vertices_grafo.size(); i++) {

        // p/ cada vizinho de saída v de u
        for (it = vertices_grafo.at(i).lista_vizinhos.begin(); it != vertices_grafo.at(i).lista_vizinh

            // v.pre < u.pre < u.pos < v.pos.
            if (vertices_grafo.at(*it).pre < vertices_grafo.at(i).pre &&
            vertices_grafo.at(*it).pre < vertices_grafo.at(*it).pos &&
            vertices_grafo.at(i).pos < vertices_grafo.at(*it).pos) {
                return true;
            }
        }
    }

    return false;
}
```

A saída do programa, conforme código 5, caso o grafo G seja direcionado acíclico, é uma ordenação topológica do grafo G, indicada pelo nome de cada vértice. Caso contrário, uma mensagem de erro é emitida.

Código 5 - *Saída do programa*

```
if (temArcoRetorno(vertices_grafo) == false) {
    for (int i = (ordenacao.size() -1); i > 0; i--)
        printf("%s ", ordenacao.at(i)->nodo);
    printf("%s", ordenacao.at(0)->nodo);
} else
    printf("Erro: grafo fornecido não é um GDA.\n");
```