# Trabalho de Estrutura de Dados

## Ordenação Topológica

### Matheus Correia, Laura Beintemuller, e Mauricio Serejo

1. **Introdução**

O objetivo do trabalho é implementar dois algoritmos, um algoritmo que gere um GDA( Grafo Direcionado Acíclico), e outro que realize a ordenação topológica do grafo gerado. Após a implementação desses algoritmos iremos executar a ordenação topológica em grafos de diferentes quantidades de vértices para analisarmos o seu comportamento, assim como compararmos este com o crescimento esperado pela complexidade teórica do algoritmo.

1. **Códigos**

2.1 Classes Gerais

public class Grafo

Atributos:

*private Vertice[ ] vertices;*

*private Aresta[ ] arestas;*

*private int contadorArestas;*

Classes privadas da classe Grafo:

.public class Vertice

Atributos:

*private int chave;*

*private boolean visitado;*

*private ListaDeVertices;*

*private int numA;*

.public class Aresta

Atributos:

*private Vertice origem;*

*private Vertice fim;*

.private class ListaDeVertices

Atributos:

*Vertice vertice;*

*ListaDeVertices prox;*

Métodos da classe Grafo:

*private void gerarAresta(Vertice v1, Vertice v2){  
 if(v1.chave == v2.chave) return;  
  
 boolean jaExisteArestaIgualOuParalela = false;  
 for(int i = 0; i < arestas.length; i++)  
 if(arestas[i] != null && ((arestas[i].origem == v1 && arestas[i].fim == v2)  
 || (arestas[i].origem == v2 && arestas[i].fim == v1)))  
 jaExisteArestaIgualOuParalela = true;  
  
 if(!vaiFormarCiclo(v1, v2) && !jaExisteArestaIgualOuParalela){  
 arestas[contadorArestas++] = new Aresta(v1, v2);  
 v1.numA++;  
 v2.numA++;  
 colocarNaListaDeSucessores(v1, v2);  
 return;  
 }  
 else return;  
}*

*private boolean vaiFormarCiclo(Vertice v1, Vertice v2) {  
 if (v1.chave == v2.chave) return true;  
 if (v2.listaDeVerticesSucessores == null) return false;  
  
 for (ListaDeVertices p = v2.listaDeVerticesSucessores; p != null; p = p.prox) {  
 if (!p.vertice.visitado) {  
 p.vertice.visitado = true;  
 if (p.vertice.chave == v1.chave) {  
 // Reinicia a marcação de visitados antes de retornar  
 reiniciarMarcacaoDeVisitados();  
 return true;  
 }  
 if (vaiFormarCiclo(v1, p.vertice)) {  
 // Reinicia a marcação de visitados antes de retornar  
 reiniciarMarcacaoDeVisitados();  
 return true;  
 }  
 }  
 }*

*reiniciarMarcacaoDeVisitados();  
 return false;  
 }*

*private void reiniciarMarcacaoDeVisitados() {  
 for (Vertice vertice : vertices) {  
 if (vertice != null) {  
 vertice.visitado = false;  
 }  
 }  
 }*

*public void gerarGrafoAleatorio(){  
 Random random = new Random();  
  
 for(; contadorArestas != arestas.length;)  
 gerarAresta(vertices[random.nextInt(vertices.length)], vertices[random.nextInt(vertices.length)]);  
  
 ListaDeVertices verticesIsolados = null;  
 for(int i = 0; i < vertices.length; i++){  
 if(vertices[i].numA == 0)  
 verticesIsolados = new ListaDeVertices(vertices[i], verticesIsolados);  
 }  
 if(verticesIsolados != null) {  
 for(int i = 0; verticesIsolados != null; i++){  
 if(i == vertices.length) {  
 i = 0;  
 }  
  
 if(arestas[i].fim.numA > 1 && arestas[i].origem.numA >1) {  
  
 arestas[i].origem.numA--;  
 arestas[i].fim.numA--;  
  
 arestas[i] = new Aresta(verticesIsolados.vertice, arestas[i].fim);  
 verticesIsolados.vertice.numA++;  
 arestas[i].fim.numA++;  
  
 verticesIsolados = verticesIsolados.prox;  
 }  
 }  
 }  
 }*

*public void criarTXT() throws IOException {  
 PrintWriter writer = new PrintWriter("grafo" + vertices.length + ".txt", StandardCharsets.UTF\_8);  
  
 for(int i = 0; i < arestas.length; i++)  
 writer.println(arestas[i]);  
  
 writer.close();  
 }*

*@Override  
 public String toString() {  
 String str = "";  
 for(int i = 0; i < arestas.length; i++){  
 str += arestas[i] + "\n";  
 }  
 return str;  
 }  
  
 private void colocarNaListaDeSucessores(Vertice v1, Vertice v2){  
 v1.listaDeVerticesSucessores = new ListaDeVertices(v2, v1.listaDeVerticesSucessores);  
 }  
}*

public class OrdenacaoTopologica

Atributos:

*private Elo prim;*

*private int n;*

Classes privadas da classe OrdenacaoTopologica:

.private class Elo

Atributos:

*public int chave;*

*public int contador;*

*public Elo prox;*

*public Elosuc listaSuc;*

.private class EloSuc

Atributos:

*public Elo id;*

*public EloSuc prox;*

Métodos da classe OrdenacaoTopologica:

*public void realizaLeituraTXT(String nomeEntrada) throws IOException {  
 Path path = Paths.get(nomeEntrada);  
 Scanner scan = new Scanner(path);  
  
 while(scan.hasNext()){  
  
 Elo eloX;  
 Elo eloY;  
  
 //recuperar x e y de cada linha do txt  
 String xPrecedesY = scan.nextLine().replace(" ", "");  
  
 int x = Integer.parseInt(xPrecedesY.substring(0, xPrecedesY.indexOf("<")));  
 int y = Integer.parseInt(xPrecedesY.substring(xPrecedesY.indexOf("<") + 1));  
  
 eloX = incluirOuRetornarElo(x);  
 eloY = incluirOuRetornarElo(y);  
  
 //Atualizar a lista de sucessores de x  
 eloX.listaSuc = new EloSuc(eloY, eloX.listaSuc);  
  
 //Atualizar os dados do elemento y  
 eloY.contador++;  
 }  
}*

*public void realizaLeituraObjeto(Grafo G){  
  
 Elo eloX;  
 Elo eloY;  
  
 for(int i = 0; i < G.arestas.length; i++){  
  
 int x = G.arestas[i].origem.getChave();  
 int y = G.arestas[i].fim.getChave();  
  
 eloX = incluirOuRetornarElo(x);  
 eloY = incluirOuRetornarElo(y);  
  
 //Atualizar a lista de sucessores de x  
 eloX.listaSuc = new EloSuc(eloY, eloX.listaSuc);  
  
 //Atualizar os dados do elemento y  
 eloY.contador++;  
 }  
}  
  
private Elo incluirOuRetornarElo(int a){  
 Elo p = buscar(a);  
  
 if(p == null) return incluirNoFinal(a);  
  
 return p;  
}*

*private Elo incluirNoFinal(int a){  
 Elo p, q;  
 q = new Elo(a);  
  
 if(prim != null) {  
 for (p = prim; p.prox != null; p = p.prox) ;  
  
 p.prox = q;  
 q.prox = null;  
 }  
  
 else{  
 prim = q;  
 prim.prox = null;  
 }  
 n++;  
  
 return q;  
}*

*private void incluirNoFinal(Elo novo) {//inclui elo da lista de sucessores  
 if (prim != null) {  
 Elo p = prim;  
 while (p.prox != null) {  
 p = p.prox;  
 }  
 p.prox = novo;  
 novo.prox = null;  
 } else {  
 prim = novo;  
 prim.prox = null;  
 }  
}  
  
//retorna o Elo que contem 'chave a' ou retorna null caso nao encontre  
private Elo buscar(int a) {  
 Elo p;  
 for (p = prim; p != null; p = p.prox)  
 if (p.chave == a) return p;  
  
 return null;  
}*

*private void substituicaoDeCadeia(){  
 //busca elementos sem predecessores  
 Elo p = prim;  
 prim = null;  
  
 while(p != null){  
 Elo q = p;  
 p = q.prox;  
 if(q.contador == 0){ //adiciona no inicio  
 q.prox = prim;  
 prim = q;  
 }  
 }  
}*

*private void debug()  
{  
 System.out.println("Debug");  
 Elo p;  
 for(p = prim; p != null; p = p.prox) {  
 System.out.print(p.chave + " predecessores: " + p.contador + ", ");  
 System.out.print("sucessores: ");  
 for(EloSuc s = p.listaSuc; s != null; s = s.prox)  
 System.out.print(s.id.chave + " -> ");  
 System.out.println("NULL");  
 }  
}*

*public boolean executa() {  
 debug();  
 substituicaoDeCadeia();  
 System.out.println("\nOrdenacao topologica");  
 gerarSaida();  
  
 return n == 0;  
}*

*private void gerarSaida(){  
 Elo q;  
  
 for(q = prim; q != null; q = q.prox){  
 System.out.print(q.chave + " ");  
 n--;  
  
 for(EloSuc t = q.listaSuc; t != null; t = t.prox){  
 t.id.contador--;  
 if(t.id.contador == 0)  
 incluirNoFinal(t.id);  
 q.listaSuc = q.listaSuc.prox; //remove da lista de sucessores  
 }  
  
 prim = prim.prox;  
 }  
  
 System.out.println();  
}*

*2.2 Classe Grafo*

. Executa o método: public void gerarGrafoAleatorio()

.O método irá utilizar o método: private void gerarAresta(Vertice v1, Vertice v2). Que segue esse passo a passo:

1. cria uma aresta e a coloca no vetor de arestas se possível.  
2. caso os vértices recebidos sejam iguais: não forma aresta.  
3. caso haja aresta igual ou paralela no vetor de arestas: não forma aresta.  
4. caso ocorra um ciclo ao formar essa aresta: não forma aresta  
5. se nenhuma dessas situações ocorrer: uma nova aresta é construída e inserida.  
6. no vetor de arestas, os contadores numA de cada vértice

serão incrementados  
 e a lista de sucessores de v1 receberá v2.

. O método utiliza o método auxiliar private boolean vaiFormarCiclo( ) para garantir que ciclos não sejam formados. Este método segue esse passo a passo:

1. realiza uma busca a partir do vértice v2 ate encontrar ou não o vértice v1  
2. para isso, visita o vértice v2 e seus sucessores e realiza uma chamada recursiva para cada vértice da lista de sucessores de v2.

3. Utiliza o método auxiliar private void reiniciarMarcacaoDeVisitados( ) que reinicia a marcação dos visitados.

. O método gerarGrafoAleatorio utiliza esses demais métodos para fazer a geração do grafo seguindo o passo a passo:

1. Cria todas as arestas que cabem no vetor de arestas, conectando vértices aleatórios, mas ao realizar isso pode ocorrer a presença de vértices isolados(que não formam aresta com ninguém).

2. Para resolver esse problema os vértices isolados são localizados por seu numA = 0 e colocados numa lista.

3. Por fim, substitui as arestas ja formadas por arestas novas com vértices isolados.

*2.3 Classe OrdenacaoTopologica*

. Primeriamente executa o método public void realizaLeituraTXT( ), ou o método public void realizaLeituraObjeto( ). Dessa maneira, é possível escolher se o grafo será lido através de um arquivo .txt (caso tenha sido criado dessa maneira) ou através de um objeto da classe Grafo.

Ambos os métodos pegam os dados contidos e os transformam para a estrutura de Elos.

. Com a leitura feita é executado o método private boolean executa( ), responsável por fazer a ordenação Topologica dos elementos através dos métodos private void debug( ), private void substituicaoDeCadeia ( ) e private void gerarSaida( ).

. O método debug irá demonstrar os elementos existentes e as suas respectivas relações de predecessores e sucessores, seguindo o passo a passo:

1. Utilizar um looping for para gerar a lista de predecessores.

2. Dentro desde looping utiliza outro lopping for para gerar os sucessores.

3. Dessa maneira, cada vez que o looping é executado é mostrado o elemento, sua quantidade de predecessores, e os seus elementos predecessores.

. O método substituicaoDeCadeia irá procurar os elementos que não possuem predecessores e adicioná-los no inicio para que apareçam primeiro durante a ordenação.

. Por fim, o método gerarSaida mostra o resultado da ordenação seguindo o passo a passo:

1. Tal qual o debug, utiliza um looping duplo para percorrer os Elos.

2. Os elementos serão impressos respeitando a ordem de predecessores e sucessores previamente estabelecida.

3. Ao final do gerarSaida teremos uma sequencia de elementos parcialmente ordenados.

1. **Complexidades**

A complexidade estimada do algoritmo foi

1. **Gráficos dos algoritmos**

Computador utilizado para os testes:

Processador: 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 3.11 GHz

RAM instalada: 8,00 GB (utilizável: 7,79 GB)

Tipo de sistema : Sistema operacional Windowns 11 de 64 bits, processador baseado em x64.

Tempo levado de acordo com o tamanho da entrada: