Relatório do Trabalho 2

Algoritmos e Estruturas de Dados 2024/2025

Gonçalo Ribau (119560) , Rodrigo Santos (119198)

Introdução

Neste relatório está descrita a Análise da Complexidade dos álgoritmos descritos nas funções *GraphBellmanFordAlgExecute()* e *GraphComputeTransitiveClosure()* do segundo projeto de AED. A função *GraphBellmanFordAlgExecute()* serve para criar a árvore dos caminhos mais curtos de um grafo dado, a partir de um vértice inicial também fornecido pelo utilizador, utilizando o algoritmo de Bellman-Ford, e a função *GraphComputeTransitiveClosure()* serve para criar um grafo que seja o fecho transitivo do grafo dado. Os vértices alcançáveis, a partir de um dado vértice de um grafo orientado, são determinados usando o módulo Bellman-Ford.

1 Função *GraphBellmanFordAlgExecute()*

1.1 Análise do Número de Iterações

- GraphGetAdjacentsTo(): Para o contador implementado dentro desta função, este é dependente de uma condição previamente estabelecida e os seus incrementos dependem dessa mesma condição ser validada. Assumimos, assim, que a seguinte condição é verdadeira:

- `Vértices Adjacentes:`1 iteração por cada vértice adjacente ao vértice dado.

$$\boxed{\text{numIterAdjs} = \text{numVerticesAdjacentes}}$$

- GraphBellmanFordAlgExecute(): Sendo V o conjunto dos vértices de um grafo g, a equação que consideramos definir o número de iterações realizadas é dada pela soma das seguintes componentes:

- `Inicializações:` 1 iteração por número de vértices do grafo g.

$$\boxed{\text{numInicializ} = \text{numVertices}}$$

- `Comparações das Arestas:` 1 iteração por cada vértice adjancente de cada vértice do grafo (numVértices - 1) vezes.

$$\boxed{\text{numComp} = (\text{numVertices -1}) \times \text{numVertices} \times \text{numVerticesAdjacentes}}$$

Assim, o valor será:

$$\boxed{\text{numIterGraphBFA} = \text{numInicializ} + \text{numVertices}}$$

$$\boxed{\text{numVertices} + (\text{numVertices} - 1) \times \text{numVertices} \times \text{numVerticesAdjacentes}}$$

1.2 Melhor Caso e Pior Caso

A complexidade do algoritmo pode ser determinada pelo número de vértices e arestas do grafo dado, como mostrámos na fórmula conseguida anteriormente. Assim, podemos concluir o melhor e pior caso a partir da mesma.

Melhor caso: O melhor caso traduz uma situação em que o algoritmo não tem de realizar nenhuma atualização, ou seja, não ocorre nenhum relaxamento das arestas. No entanto, o algoritmo ainda realiza a primeira iteração completa.

A fórmula terá a seguinte forma:

$$\boxed{\text{numVertices} + (\text{numVertices} - 1) = 2 \times \text{numVertices} - 1}$$

Pior caso: O pior caso traduz uma situação em que o algoritmo precisa de relaxar todas as arestas em todas as iterações. Isso pode ocorrer, por exemplo, em grafos densos, onde muitas arestas exigem relaxamento.

$$\boxed{\text{numVertices} + (\text{numVertices} - 1) \times \text{numVertices} \times \text{numVerticesAdjacentes}}$$

Abaixo, apresentamos uma tabela e um gráfico que ilustram o desempenho do algoritmo Bellman-Ford nos melhores e piores casos, considerando diferentes números de vértices no grafo analisado.

	Número de Iterações	
Número de Vértices	**Melhor Caso**	**Pior Caso**
5	9	45
10	19	460
20	39	3820
30	59	13080
50	99	61300
70	139	169120
100	199	495100
150	299	1676400
200	399	3980200
300	599	13455300

Tabela 1: Desempenho do Algoritmo Bellman-Ford em Diferentes Casos

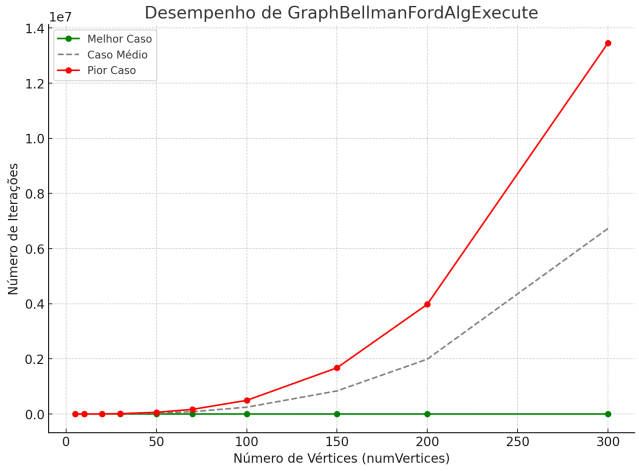

Figura 1: Gráfico de desempenho da função GraphBellmanFordAlgExecute().

2 GraphComputeTransitiveClosure():

2.1 Análise do Número de Iterações

Sendo V o conjunto dos vértices de um grafo g, a equação que define o número de iterações realizadas na função `GraphComputeTransitiveClosure()` é dada pela soma das seguintes componentes:

- `Criação do Grafo:` A criação do grafo de fecho transitivo (`tcresult`) requer iterações proporcionais ao número de vértices do grafo original.

$$\boxed{\text{numIterCreate} = \text{numVertices}}$$

- `Chamadas ao Bellman-Ford:` O algoritmo Bellman-Ford é executado para cada vértice do grafo original, com um custo representado por `numIterGraphBFA`.

$$\boxed{\text{numBfInvocations} = \text{numVertices} \times \text{numIterGraphBFA}}$$

- `Adições de Arestas:` Após cada chamada ao Bellman-Ford, verifica-se a conectividade entre os vértices, resultando em `numVertices` iterações adicionais para cada vértice.

$$\boxed{\text{numEdgeAdditions} = \text{numVertices} \times \text{numVertices}}$$

Portanto, o número total de iterações é dado pela fórmula:

$$\boxed{\text{numIterGraphTC} = \text{numIterCreate} + \text{numVertices} \times (\text{numIterGraphBFA} + \text{numVertices})}$$

2.2 Melhor Caso e Pior Caso

Melhor caso: No melhor caso, o grafo possui poucas arestas, e o algoritmo Bellman-Ford encontra poucos caminhos entre os vértices, resultando num número mínimo de adições de arestas ao grafo de *Transitive Closure*. Assim, o número total de iterações será dominado pela criação do grafo e pelas chamadas ao Bellman-Ford, enquanto que as adições de arestas serão insignificantes. A fórmula será:

$$\boxed{\text{numIterGraphTC} = \text{numIterCreate} + \text{numVertices} \times \text{numIterGraphBFA}}$$

Pior caso: No pior caso, o grafo é denso, com muitas arestas e alta conectividade. O algoritmo Bellman-Ford realiza relaxamentos para todas as arestas, e todas as conexões possíveis entre os vértices precisam de ser verificadas e adicionadas ao grafo de *Transitive Closure*. Neste cenário, o número total de iterações inclui a criação do grafo, todas as chamadas ao Bellman-Ford, e a verificação e adição de arestas. A fórmula será a seguinte:

$$\boxed{\text{numIterGraphTC} = \text{numIterCreate} + \text{numVertices} \times (\text{numIterGraphBFA} + \text{numVertices})}$$

Abaixo, apresentamos uma tabela e um gráfico que ilustram o desempenho da função `GraphComputeTransitiveClosure()` nos melhores e piores casos, considerando diferentes números de vértices no grafo analisado.

Tabela 2: Desempenho da Função GraphComputeTransitiveClosure em Diferentes Casos

	Número de Iterações	
Número de Vértices	**Melhor Caso**	**Pior Caso**
5	55	80
10	210	310
20	820	1220
30	1830	2730
50	5050	7550
70	9870	14770
100	20100	30100
150	45150	67650
200	80200	120200
300	180300	270300

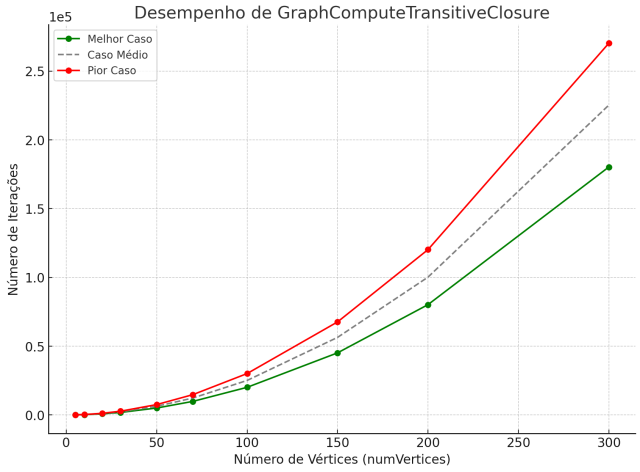

Figura 2: Gráfico de desempenho da função GraphComputeTransitiveClosure().

Conclusão

Neste projeto, implementámos e analisámos os algoritmos de Bellman-Ford e do Fecho Transitivo, mostrámos as suas complexidades e o impacto da estrutura do grafo no desempenho.