AED - PROJETO 2

Dept. de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro

André Vasconcelos, Tiago Mendes
(118827) vasconcelos13@ua.pt, (119378) tfdmendes@ua.pt

2 de janeiro de 2025

Capítulo 1

Introdução

Este relatório apresenta o desenvolvimento e análise de algoritmos para o processamento de grafos no âmbito do TAD GRAPH, com foco na implementação eficiente e na avaliação da complexidade computacional de duas soluções: o algoritmo de Bellman-Ford e o algoritmo de construção do fecho transitivo de um grafo.

Os algoritmos implementados foram testados em grafos orientados de diferentes complexidades, e os resultados experimentais foram utilizados para caracterizar o comportamento das soluções em termos de tempo de execução e desempenho.

Capítulo 2

Análise das Funções

2.1 GraphBellmanFordAlgExecute

Na função `GraphBellmanFordAlgExecute`, implementámos o algoritmo de Bellman-Ford para grafos não ponderados, com o objetivo de encontrar os caminhos mais curtos a partir de um vértice de origem:

Sendo `E` o conjunto de arestas e `V` e o conjunto dos vértices, a complexidade do algoritmo é $O(V \times E)$. O algoritmo realiza $V - 1$ iterações, processando todas as arestas em cada uma delas. A melhoria de Moore, que interrompe o *loop* quando nenhuma distância é atualizada, pode reduzir o número de iterações em grafos esparsos.

2.1.1 Testes e Casos Experimentais

Worst Case

No pior caso, o algoritmo Bellman-Ford precisa executar $V - 1$ iterações, onde `V` é o número de vértices no grafo. Durante cada iteração, ele percorre todos os vértices `u` do grafo. Para cada vértice `u`, o algoritmo verifica seus vizinhos, que são obtidos pela função `GraphGetAdjacentsTo(g, u)`. Portanto, no pior caso, para `V` vértices e `E` arestas, a complexidade é:

$$O((V - 1) \times E) = O(V \times E)$$

Best Case

No melhor caso, o algoritmo se beneficia da parada antecipada, que ocorre se nenhuma atualização de distância for feita em uma iteração. Isso significa que, se a solução for encontrada antes de completar as $V - 1$ iterações, o algoritmo pode terminar mais rapidamente.

Considerando o cenário em que o grafo é uma árvore (ou seja, não há ciclos), o número de arestas E será $V - 1$. A distância de cada vértice será atualizada rapidamente, e a parada antecipada ocorrerá logo no início. Nesse caso, o número de iterações realizadas será muito menor do que $V - 1$, sendo limitado pelo número de vértices alcançados.

No melhor caso, onde o algoritmo para logo após percorrer todas as arestas em uma única iteração, a complexidade será:

$$O(V)$$

Testes:

N	**Best Time (s)**	**Worst Time (s)**	**Ratio (Worst/Best)**
1	4.0×10^{-6}	1.0×10^{-6}	0.25
2	2.0×10^{-6}	2.0×10^{-6}	1.00
4	3.0×10^{-6}	6.0×10^{-6}	2.00
8	4.0×10^{-6}	9.0×10^{-6}	2.25
16	7.0×10^{-6}	2.7×10^{-5}	3.86
32	1.6×10^{-5}	9.1×10^{-5}	5.69
64	2.5×10^{-5}	3.28×10^{-4}	13.12
128	6.1×10^{-5}	8.09×10^{-4}	13.26
256	4.7×10^{-5}	3.09×10^{-3}	65.79
512	7.3×10^{-5}	1.23×10^{-2}	168.49
1024	1.52×10^{-4}	5.02×10^{-2}	330.56
2048	3.07×10^{-4}	1.97×10^{-1}	642.22

Tabela 2.1: Tempos de execução do Bellman-Ford em Best Case (árvore) e Worst Case (grafo completo), além da razão entre Worst e Best.

2.2 GraphComputeTransitiveClosure

A complexidade do algoritmo é dominada por duas partes principais. Primeiro, há um laço que percorre todos os vértices do grafo, o que contribui com um fator $O(V)$. Em seguida, para cada vértice, o algoritmo de Bellman-Ford é executado, o que tem uma complexidade de $O(V \times E)$. Após isso, há um outro laço que percorre novamente todos os vértices para verificar quais são alcançáveis, o que adiciona uma complexidade de $O(V)$ por iteração do laço principal. Assim, a complexidade total do algoritmo é $O(V^2 \times E)$, o que pode ser ineficiente para grafos grandes, especialmente em grafos densos. Esse custo pode ser reduzido em cenários específicos, mas o algoritmo, como está, pode se tornar pesado em grafos grandes ou complexos.

- **Worst Case:** Um grafo denso, onde todos os vértices estão conectados entre si. Neste cenário, tanto o número de arestas`|E|` quanto o comprimento máximo l atingem seus valores máximos. A complexidade computacional torna-se $O(|V|^2 \times E)$, devido à alta densidade de conexões.
- **Best Case:** Um grafo esparso, como uma árvore, em que cada vértice está conectado ao mínimo possível de outros vértices ($|E| = |V| - 1$). Aqui, o comprimento máximo dos caminhos (l) é significativamente menor, resultando em uma complexidade reduzida de $O(|V|^2)$

N	**Best Time (s)**	**Worst Time (s)**	**Ratio (Worst/Best)**
1	1.0×10^{-5}	3.0×10^{-6}	0.30
2	5.0×10^{-6}	4.0×10^{-6}	0.80
4	1.1×10^{-5}	1.9×10^{-5}	1.73
8	6.3×10^{-5}	9.5×10^{-5}	1.51
16	1.3×10^{-4}	5.3×10^{-4}	4.11
32	6.1×10^{-4}	3.4×10^{-3}	5.60
64	2.3×10^{-3}	1.6×10^{-2}	6.99
128	7.8×10^{-3}	8.9×10^{-2}	11.40
256	5.1×10^{-2}	6.9×10^{-1}	13.40
512	3.6×10^{-1}	5.4×10^{0}	15.02
1024	2.8×10^{0}	4.6×10^{1}	16.56
2048	2.2×10^{1}	3.9×10^{2}	17.27

Tabela 2.2: Tempos de execução da função de Fecho Transitivo em Best Case (DAG em linha) e Worst Case (grafo completo), com o respectivo Ratio (Worst/Best).

Capítulo 3

Conclusão

Os algoritmos demonstraram eficiência nos testes e alinhamento com previsões teóricas. A abordagem modular reforçou a clareza e precisão da implementação, consolidando o estudo prático de algoritmos em grafos.