deti universidade de aveiro departamento de electrónica, telecomunicações e informática

Projeto 2 AED – O TAD *Graph*

Marta Cruz - nº 119572 ; Catarina Ribeiro – nº 119467

---

1. Introdução

No contexto da disciplina de Algoritmos e Estruturas de Dados foi desenvolvido o TAD (Tipo Abstrato de Dados) *Graph*. No decurso do desenvolvimento deste projeto foram construidos 5 módulos dos quais 2 serão analisados neste relatório: Módulos BELLMAN-FORD e TRANSITIVE-CLOSURE. Para a sua análise de complexidade computacional optamos por desenvolver mecanismos de teste simples e eficientes explorando o pior e melhor caso para cada um dos algoritmos desenvolvidos. Para testar e obter os resultados experimentais para analisar a complexidade dos módulos basta executar o *script Matlab* "main.m" presente na diretoria *GraphTester_tests*. Este *script* desenvolvido possibilita vários testes ajustáveis alterando apenas algumas constantes. O *script* começa por chamar um outro *script* shell *execute_GraphTester.sh* que , por sua vez, irá chamar o módulo de testes desenvolvido na linguagem C, *GraphTester.c*. Este ficheiro realiza testes sobre o tipo de dados *Graph*, em particular aos métodos de Bellman-Ford e Transitive Closure gerando e processando diversos grafos criados dinamicamente. Estes grafos são gerados pelo ficheiro *GraphGenerator.c*. Após a execução, o *script* Shell escreve os resultados da instrumentação utilizada num ficheiro de dados "*data.txt*". Estes resultados são, depois, lidos e processados pelo programa *Matlab* que finalmente apresenta os gráficos utilizados neste relatório. Este módulo de testes (*Figura 1*) permite uma análise robusta permitindo a visualização de variados casos e situações com diferentes números de vértices e arestas.

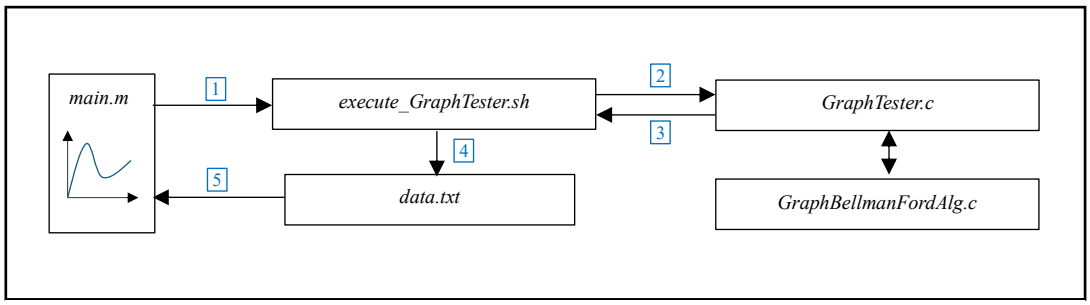

Figura 1 - Módulo de testes desenvolvido para o algoritmo de Bellman-Ford

2. Análise da complexidade do módulo Bellman-Ford

No ficheiro *GraphBellmanFordAlg.c* está presente a nossa solução para obter o caminho mais curto entre um vértice de eleição (*startVertex*) e todos os outros vértices presentes num dado grafo. A função *GraphBellmanFordAlgExecute* começa por inicializar a estrutura de dados de uma forma apropriada, isto é, coloca os *arrays marked*, *predecessor* e *distance* com os valores iniciais (0 ou -1) pedidos no enunciado. Depois de inicializada a estrutura de dados, dá-se inicio ao algoritmo propriamente dito. A função percorre todos os vértices do grafo V-1 vezes, sendo V = número de vértices presentes num dito

grafo. Para cada um dos vértices itera por todas as arestas que se iniciam nesse vértice e averigua se o caminho é melhor através do seu caminho + 1 ou através de um outro caminho já calculado previamente. Se se der o primeiro caso descrito, ou seja, se o caminho final for melhor através do seu caminho + 1, então a estrutura de dados é atualizada. De forma a otimizar o algoritmo decidimos adicionar uma *flag update* que contem o valor 1 caso exista pelo menos uma atualização feita nessa iteração. Caso não sejam feitas quaisquer atualizações à estrutura de dados, *update* fica com o valor de 0 e o ciclo é interrompido, uma vez que não existe nenhum caminho com custos melhores do que aqueles que já foram encontrados. Considerando V, E como sendo o número total de vertices e arestas presentes num dado grafo, respetivamente, podemos fazer uma análise dos casos possíveis. Apresentamos um exemplo em baixo, que representa o melhor e pior caso de grafos para este algoritmo, onde iremos analisar a sua complexidade temporal:

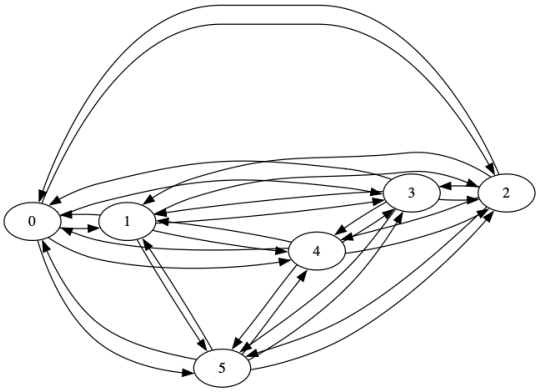

Figura 2 - Bellman-Ford pior caso

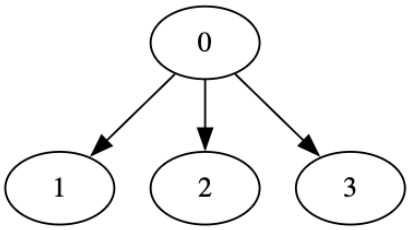

Figura 3 - Bellman-Ford melhor caso

2.1.Melhor caso

Em termos práticos o melhor caso ocorre quando o grafo no qual se vai aplicar o algoritmo se classifique como esparso, ou seja, um grafo que possua um pequeno número de arestas em relação ao número de vértices, oposto ao grafo denso, onde o número de arestas se aproxima do máximo de arestas possível, ou quando esteja suficientemente bem estruturado de forma a que todos os caminhos mais curtos sejam encontrados nas primeiras iterações, tal como no exemplo apresentado acima. A dedução da expressão para o melhor caso encontra-se abaixo. Durante a dedução foi utilizada a expressão base $\sum_{i=0}^{k} 1 = k$.

$$\sum_{i=0}^{k}\sum_{v=0}^{V}\sum_{e=0}^{outdegree(v)} 1 = \sum_{i=0}^{k}\sum_{v=0}^{V} outdegree(v) = E \cdot \sum_{i=0}^{k} 1 = k \cdot E$$

Sejam k o número de iterações necessárias antes da condição (!*update*) ser satisfeita, com $k \ll V - 1$ e *outdegree(v)* o número de arestas que tenham início no vértice v. No melhor dos casos teremos k = 2, pois as arestas serão relaxadas na primeira iteração (k=1) e será feita uma segunda iteração (k=2) para se verificar que não existem caminhos melhores do que aqueles que se obtiveram. Deste modo a complexidade da nossa solução pode ser avaliada em termos de grandeza *Big-Oh* da seguinte forma:

$$k \cdot E = 2 \cdot E \rightarrow \mathcal{O}(2 \cdot E)$$

Nota: é possível deduzir que $\sum_{v=0}^{V} outdegree(v) = E$, uma vez que, a soma de todos os graus de saída de todos os grafos será igual ao número total de arestas presentes num digrafo.

2.2.Pior caso

O pior caso ocorre quando todas as arestas do grafo precisem de passar pelo processo de "relaxamento" em cada iteração do *loop* que corre V-1 vezes. Como este tipo de grafos exige uma estrutura específica

envolvendo pesos precisamente localizados em certas arestas, decidimos optar por utilizar como pior casos grafos com um elevado número de vértices e arestas. Em termos práticos, traduz se num grafo completo e totalmente conexo, tal como representado no exemplo acima. Neste tipo de grafos não existe oportunidade para interromper o ciclo mais cedo uma vez que a *flag update* ficará sempre com o valor a 1 (existe sempre um caminho melhor a cada iteração). Abaixo apresenta-se então a dedução da expressão para o pior caso, idealmente, ou seja, quando o algoritmo é obrigado a correr o *loop* externo V-1 vezes. Foi utilizada, mais uma vez a expressão base $\sum_{i=0}^{k} 1 = k$.

$$\sum_{i=0}^{V-1}\sum_{v=0}^{V}\sum_{e=0}^{outdegree(v)} 1 = \sum_{i=0}^{V-1}\sum_{v=0}^{V} outdegree(v) = E \cdot \sum_{i=0}^{V-1} 1 = E \cdot (V-1)$$

Neste caso, como os valores dos melhores caminhos estão sempre a ser atualizados o ciclo externo irá correr V-1 vezes resultando, portanto, numa complexidade

$$E \cdot (V-1) \rightarrow \mathcal{O}(E \cdot V)$$

Abaixo apresentamos os resultados empíricos:

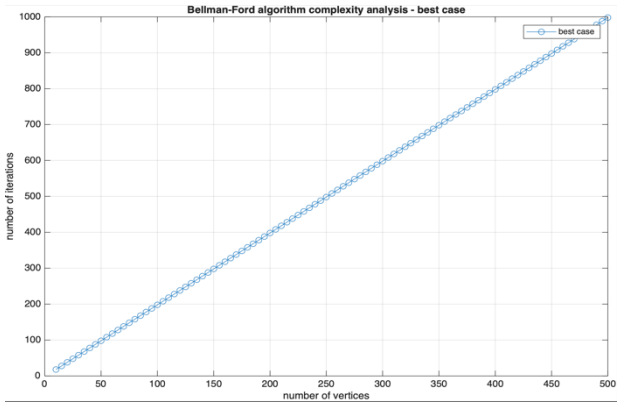

Figura 4 - Resultado experimental do módulo de Bellman-Ford (melhor caso) - $\mathcal{O}(E)$

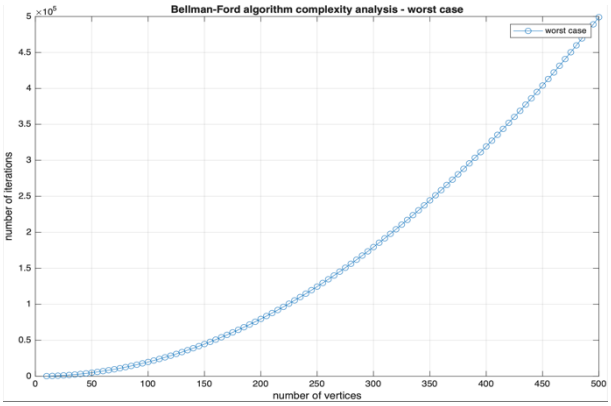

Figura 5 - Resultado experimental do módulo de Bellman-Ford (pior caso) - $\mathcal{O}(E \cdot V)$

Podemos, portanto, concluir as ordens de complexidade calculadas acima. No melhor caso temos um número de arestas que cresce linearmente de 10 até 500, como a complexidade fica na ordem $\mathcal{O}$(E) temos como resultado uma regressão linear. No pior caso temos um número de arestas aproximado ao número de vértices existentes no grafo, deste modo ficamos com algo do tipo : V^2o que resulta numa curva quadrática.

Exemples	Vertices	Edges	Iterations
Melhor Caso	10	20	18
	50	100	98
	100	250	198
	250	400	498
	500	800	998
Pior Caso	10	90	180
	50	2450	4900
	100	9900	19800
	250	62250	124500
	500	249500	499000

Figura 6 – Tabela com casos aleatórios do módulo Bellman-Ford

3. Análise da complexidade do módulo Transitive-Closure

No ficheiro *GraphTransitiveClosure.c* está presente a nossa solução para o desenvolvimento de um algoritmo que calcule o fecho-transitivo de um dado grafo. A função itera por cada um dos vertices do grafo em questão e gera a árvore de caminhos mais curtos com a raiz nesse vértice. Depois de gerada a árvore de caminhos mais curtos é adicionada uma aresta no grafo resultante se existir um caminho entre o vértice inicial (ciclo externo) e qualquer outro vertice presente no grafo (ciclo interno). Considerando V, E como sendo o número total de vertices e arestas presentes num dado grafo, respetivamente, podemos fazer uma análise dos cenários possíveis. Apresentamos ,abaixo, um exemplo que representa o melhor e pior caso de grafos para este algoritmo, onde iremos analisar a sua complexidade temporal:

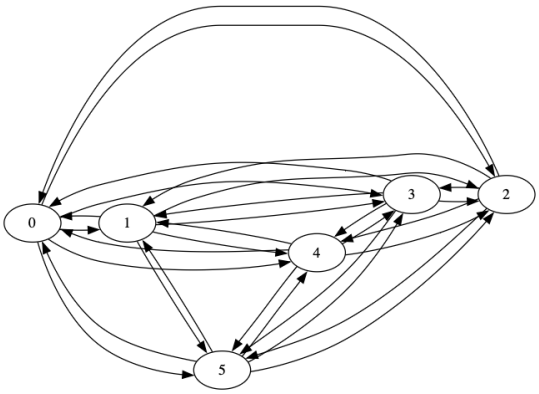

Figura 7- Pior caso (grafo denso/completo)

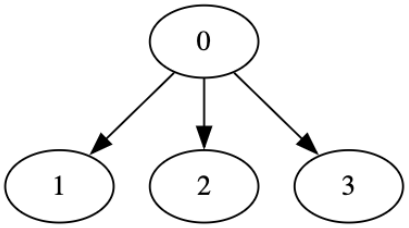

Figura 8 – Melhor caso (grafo esparso)

3.1.Pior caso

De um modo geral a complexidade da nossa solução pode ser partida em 2 partes principais: o custo de gerar a árvore de caminhos mais curtos para cada vértice + iterar por cada um dos vértices que são atingíveis através desse vértice e adicionar uma aresta. Deste modo apresentamos a dedução da expressão para a complexidade temporal do algoritmo desenvolvido. Durante a dedução foi utilizada a expressão base $\sum_{i=0}^{k} 1 = k$.

$$\sum_{v=0}^{V} [cost\ of\ generating\ spanning\ tree\ (worst\ case) + \ cost\ of\ adding\ a\ new\ edge]$$

$$= \sum_{v=0}^{V} cost\ of\ generating\ spanning\ tree\ (worst\ case) + \sum_{v=0}^{V}\sum_{i=0}^{r} 1$$

$$= \sum_{v=0}^{V} \big(E \cdot (V-1)\big) \cdot \sum_{v=0}^{V} 1 + (r \cdot V) = \sum_{v=0}^{V} E \cdot V - \sum_{v=0}^{V} E + (r \cdot V)$$

$$= E \cdot V^2 - E \cdot V + (r \cdot V)$$

Seja *r* o número de vértices atingíveis a partir do vértice *v*, podemos definir a complexidade do nosso algoritmo como estando na gama $\mathcal{O}(E \cdot V^2)$ uma vez que este se apresenta como termo dominante na equação. No pior dos casos cada vértice será atingível através de qualquer outro vértice. Deste modo *r = V-1* para cada vértice, e, portanto:

$$E \cdot V^2 - E \cdot V + (r \cdot V) = E \cdot V^2 - E \cdot V + \big((V-1) \cdot V\big) = E \cdot V^2 - E \cdot V + V^2 - V$$

Em que o termo $E \cdot V^2$ é dominante para V muito grande. Assim, podemos classificar o algoritmo como estando na ordem $\mathcal{O}(E \cdot V^2)$ no pior caso.

Podemos, portanto, concluir que o algoritmo usando o módulo de Bellman-Ford como base não é o mais eficiente, sendo a sua complexidade particularmente pesada para grafos completos com V muito

grande. Deste modo se pretendemos obter melhores resultados será mais apropriado utilizar algoritmos tais como o algoritmo de *Floyd-Warshall* ou *Depth-First Search* (DFS) para obter os vértices atingíveis.

3.2.Melhor caso

Em termos práticos o melhor caso ocorre em grafos que se classifiquem como sendo esparsos e que não tenham muitas dependências, tal como descrito no exemplo apresentado acima. Nestes casos apenas adicionaremos arestas num dos vértices, uma vez que para todos os outros vértices não existe qualquer relação. Deste modo a expressão que traduz a complexidade temporal define-se como:

$$\sum_{v=0}^{V}\left[cost\ of\ generating\ spanning\ tree\ (best\ case) + \sum_{i=0}^{r} 1\right]$$
$$= \sum_{v=0}^{V} cost\ of\ generating\ spanning\ tree\ (best\ case) + \sum_{i=0}^{V-1} 1 = \sum_{v=0}^{V} E + V - 1$$
$$= E \cdot V + V - 1$$

Podemos, portanto, concluir que a complexidade do melhor caso para este algoritmo se encontra na ordem $\mathcal{O}(E \cdot V)$. Neste caso o custo de adicionar uma nova aresta ao grafo resultante traduz-se em: $\sum_{i=0}^{V-1} 1$ para 1 dos vértices e 0 para todos os outros, deste modo apenas necessitamos de acionar este termo 1 vez para contabilizar todas as operações feitas deste tipo.

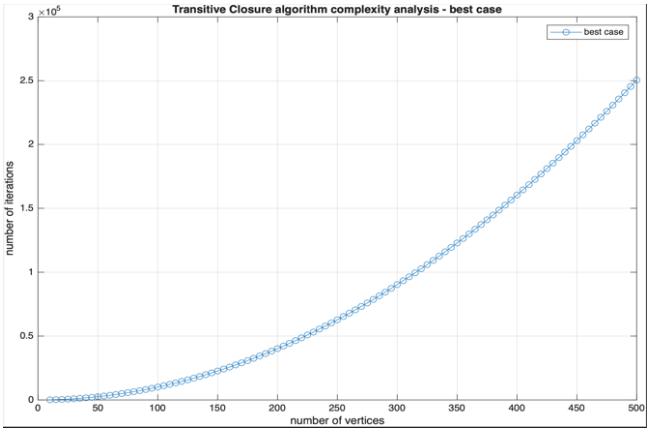

Figura 9 - Resultado experimental do módulo Transitive Closure (melhor caso) - $\mathcal{O}(E \cdot V)$

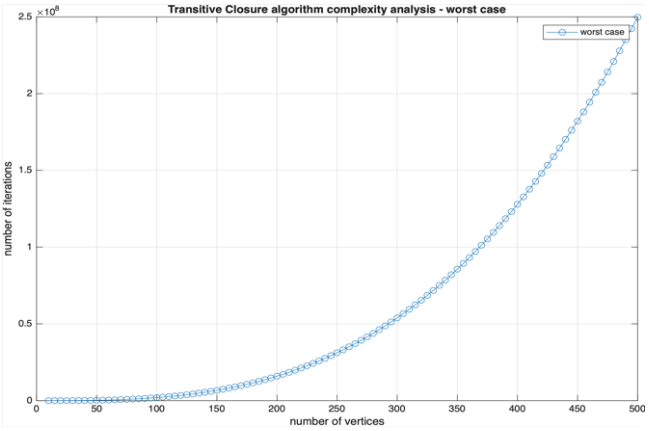

Figura 10 - Resultado experimental do módulo Transitive Closure (pior caso) - $\mathcal{O}(E \cdot V^2)$

Podemos então verificar as ordens de complexidade calculadas acima. No melhor caso temos grafos esparsos e sem muitas dependências, o que pode ser descrito como $E \approx V$. Deste modo temos uma curva que representa, aproximadamente, V^2, ou seja, uma regressão quadrática. Por outro lado, no pior caso a curva apresentada tem um declive mais evidente para valores de V muito grandes, deste modo aproxima-se a uma curva representativa de um polinómio de terceiro grau. Isto é uma consequência do tipo de grafos que utilizamos para obter estes resultados (grafos completos) em que o número de arestas se dá pela seguinte equação: $E = V \cdot (V-1)$, e, portanto, a ordem evolui de $\mathcal{O}(E \cdot V)$ para $\mathcal{O}(V \cdot (V-1) \cdot V) \rightarrow \mathcal{O}((V^2 - V) \cdot V) \rightarrow \mathcal{O}(V^3)$.

Exemples	Vertices	Edges	Iterations
Melhor Caso	10	20	108
	50	100	2548
	100	200	10098
	250	500	62748
	500	1000	250498
Pior Caso	10	90	1898
	50	2450	247450
	100	9900	1989900
	250	62250	31187200
	500	249500	249750000

Figura 11 – Tabela com casos aleatórios do módulo Transitive-Closure

4. Conclusão

O trabalho desenvolvido permitiu implementar e analisar os módulos Bellman-Ford e Fecho Transitivo, atendendo aos objetivos propostos e fornecendo soluções funcionais para problemas relacionados à análise de grafos direcionados. O módulo Bellman-Ford demonstrou ser eficaz na determinação dos caminhos mais curtos a partir de um vértice inicial, com uma complexidade de $O(E \cdot V)$, adequada para grafos esparsos, mas limitada para grafos densos, onde o elevado número de arestas aumenta o custo computacional. O módulo de Fecho Transitivo, baseado no Bellman-Ford, apresentou uma complexidade de $O(E \cdot V^2)$, sendo também mais eficiente em grafos esparsos, mas enfrentando desafios de desempenho em cenários com alta densidade de conexões. Apesar dessas limitações, os resultados obtidos confirmaram o cumprimento dos objetivos.

De forma geral, o trabalho proporcionou uma compreensão aprofundada sobre os problemas abordados e as suas soluções. Contudo, para otimizar o desempenho em grafos densos, seria interessante explorar alternativas como o algoritmo de Floyd-Warshall, mais eficiente para cálculos envolvendo todos os pares de vértices, ou o algoritmo Depth-First Search (DFS) para problemas de acessibilidade. Essas adaptações podem melhorar a eficiência e ampliar a aplicabilidade dos algoritmos desenvolvidos, tornando-os mais adequados para diferentes tipos de grafos.