Uma imagem com texto, cartão de visita, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

**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.

Completar com a parte a explicar o script matlab

1

*main.m*

*.sh*

*.c*

*.c*

*.txt*

2

3

4

5

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

1. **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:

A diagram of a diagram

Description automatically generated

Figura 3- Bellman-Ford melhor caso

Figura 2- Bellman-Ford pior caso

* 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 .

Sejam k o número de iterações necessárias antes da condição (!*update*) ser satisfeita, com e *outdegree(v)* o número de arestas que tenham inicio no vertice v. No melhor dos casos teremos k = 1, e , portanto, a complexidade da nossa solução pode ser avaliada em termos de grandeza *Big-Oh* da seguinte forma:

Nota: é possível deduzir que , 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.

* 1. **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. Em termos práticos, traduz se num grafo “em-linha”, 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. Foi utilizada, mais uma vez a expressão base .

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

INSERIR GRÁFICOS EXPERIMENTAIS

1. **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:

A diagram of a basket

Description automatically generated



Figura 5 – Melhor caso (grafo esparso)

Figura 4- Pior caso (grafo denso/ completo)

* 1. **Caso Geral**

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 atingínveis através desse vértice. Deste modo apresentamos a dedução da expressão para a complexidade temporal do algoritmo desenvolvido. Durante a dedução foi utilizada a expressão base .

Deste modo defenimos a complexidade do nosso algoritmo como estando na gama uma vez que este se apresenta como termo dominante na equação. Podemos então fazer uma adaptação desta equação para o melhor e pior caso, cuja dedução apresentamos abaixo.

* 1. **Melhor caso**

Em termos práticos o melhor caso ocorre em grafos que se classifiquem como sendo esparsos. Nestes casos esta condição pode ser transcrita como: e, portanto, .

* 1. **Pior caso**

O pior caso dá-se em grafos densos, sendo o pior o grafo completo, ou seja, um grafo cujos vértices se encontrem ligados por uma aresta a cada outro vértice, o que significa que o algoritmo de Bellman-Ford irá processar o número máximo de arestas para cada vértice. Para além disso o processo de adição de arestas irá ocorrer o número máximo de vezes. Esta condição pode ser descrita através da seguinte equação:

E, portanto, substituindo na expressão da complexidade obtemos o seguinte:

Em que para V muito grande termo domina, sendo esta a complexidade do algoritmo 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.

INSERIR GRÁFICOS EXPERIMENTAIS

1. **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 , 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 , 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.