**Análise e Síntese de Algoritmos (2º Projeto)**

Grupo 57

Isabel Soares (89466)

Tiago Afonso (89546)

1. **Breve introdução**

Foi-nos proposto desenvolver um projeto no âmbito da cadeira de Análise e Síntese de Algoritmos que se baseia numa aplicação informática que determina a capacidade máxima que a rede de transporte do Sr. Caracol é capaz de transportar, ou seja, a maior quantidade de produtos que a rede consegue mover dos fornecedores até a um hipermercado, podendo passar por uma ou várias estações de abastecimento.

Deste modo, apresentamos o problema como um grafo dirigido com vértice fonte e destino, em que cada arco possui um valor de fluxo e capacidade. Utilizamos um algoritmo de pré-fluxos para calcular o fluxo máximo, nomeadamente *Relabel-To-Front* lecionado nas aulas.

1. **Descrição da solução**

A implementação do programa foi realizada em linguagem C.

Para a criação da rede de fluxo, foi considerado como vértice 0 uma *source* “fictícia”, que se liga a todos os fornecedores (vértices 2 a “número de fornecedores” (simbolizado por *f*) + 1), e cuja capacidade simboliza a produção de cada fornecedor. O vértice correspondente ao hipermercado é o vértice 1, e as estações representam os vértices de *f+2* a *f +* número de estações (simbolizado por *e*) *+ 1*. Para simular a capacidade de cada estação, usa-se um vértice auxiliar de *f+e+2* até *numvertices* (o número de vértices total), estando cada estação ligada ao seu auxiliar respetivo, com um arco de capacidade igual à capacidade da estação.

Como estrutura de dados, são usadas duas matrizes, uma que guarda a capacidade de cada ligação, e outra que é inicializada a 0 mas que após o algoritmo representará os fluxos da rede.

Para calcular a capacidade máxima da rede, utilizou-se o algoritmo *Relabel-to-Front*. Este cria uma lista de excessos, outra de alturas, outra de vértices vistos e outra de prioridades. Coloca-se a altura da source igual ao número de vértices, enchendo os arcos da source até aos fornecedores (*push* inicial). Enquanto houver excesso em algum vértice da rede que não seja source e o hipermercado (*market*), cada vértice da lista de prioridades vai fazer *discharge*, isto é, enquanto tiver excesso irá tentar dar o máximo possível a outro vértice de altura menor (*push*), e caso não o consiga fazer, irá aumentar a sua altura até esta ficar superior à de outro vértice a que consiga enviar (*relabel*). No caso de um vértice fizer *relabel*, a sua posição é atualizada para o primeiro lugar da lista de prioridades (*updateList*).

Implementámos também uma DFS para encontrar o corte mínimo do grafo anterior.

1. **Análise Teórica**

Em relação à análise teórica do nosso algoritmo, mais concretamente relacionado com a execução de cada ciclo, tendo em conta que V é o número de vértices de um grafo (neste caso os fornecedores(f), as estações de abastecimento(e) e o Hipermercado) e E é o número de arestas de um grafo (neste caso o número de ligações que existem na rede do Sr. Caracol), temos:

* Algoritmo Relabel-To-Front: O(V3);
* Matriz de Adjacências: O(V2);
* Algoritmo DFS: O(V + E);

Logo, a complexidade final do nosso algoritmo é de O().

1. **Análise Experimental dos Resultados**

Para verificar o uso de memória e tempo de execução do algoritmo, foram testados os inputs dados como exemplo, utilizando as funções *time* e *Valgrind* do Linux.

Como se pode ver, o gasto de memória é linear, aumentando com o número de fornecedores, estações de abastecimento e ligações, não sendo tão fácil de ver no gráfico Tempo-Routers+Ligações, devido à diferença de valores usados nos exemplos e à diferença de precisão entre o comando *time* e o *Valgrind*

1. **Referências**

* **Introduction to Algorithms, Third Edition:**Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein September 2009 ISBN-10: 0-262-53305-7; ISBN-13: 978-0-262-53305-8
* <https://stackoverflow.com/questions/4482986/how-can-i-find-the-minimum-cut-on-a-graph-using-a-maximum-flow-algorithm>