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.

2. 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*).

Para se obter a rede residual, subtraiu-se a matriz de capacidades à matriz de fluxos, e transpondo essa matriz com recurso a um nova matriz (*transposed*). De seguida, fez-se uma DFS nesta nova matriz, a partir do hipermercado, de forma a obter as partições alcançáveis e não alcançáveis, obtendo-se um *array* de vértices visitados (*visited*). Percorrendo a matriz original e vendo os pares de vértices em que um foi visitado e o outro não, chega-se ao corte mínimo.

Para cada ligação no corte mínimo, foram adicionados nos *arrays* *cutstations* e *cutedges* as estações e ligações, respetivamente, que necessitam de ser aumentadas, isto é, caso se aumentasse a capacidade destas estações ou ligações, a capacidade da rede aumentaria. Antes de se imprimir o output, estes *arrays* são ordenados atráves do algoritmo de ordenação *Quicksort* (função *qsort* presente na biblioteca *stdlib.h*).

3. Análise Teórica

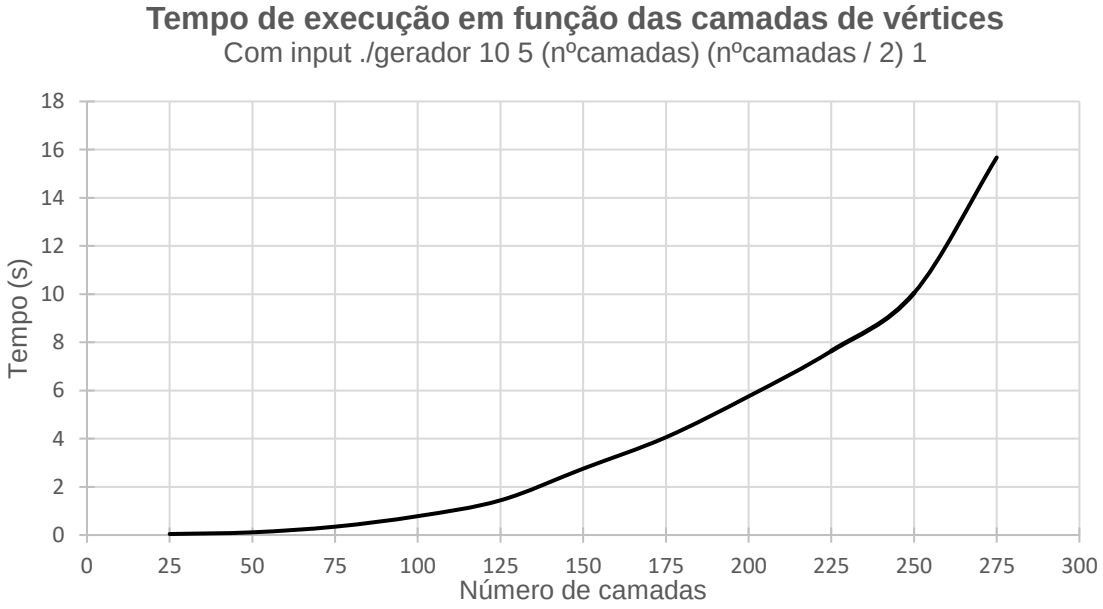
Relativamente à análise teórica do nosso algoritmo, tendo em conta as estruturas de dados e outros algoritmos utilizados, e considerando F como o número de fornecedores, E como o número de estações, L como o número de ligações, e que na rede criada existe um total de F+2E+2 vértices (fornecedores, estações e respetivos auxiliares, e os vértices correspondentes à *source* e ao hipermercado), as complexidades temporais e espaciais do projeto são:

- Algoritmo *Relabel-To-Front*: $O((F+2E+2)^3) = O(V^3)$;
- Matriz de Adjacências: $O(V^2)$ (espaço e inicialização);
- Algoritmo DFS: O(V + L);

- Algoritmo *Quicksort*: O(V log V);
  Logo, a complexidade temporal final do nosso algoritmo é de $O(V^3)$ e a espacial é de $O(V^2)$.

4. Análise Experimental dos Resultados

Para verificar o tempo de execução do algoritmo, foram testados *inputs* criados pelo gerador dado, onde se variaram as camadas e a camada de corte em dois exemplos diferentes. No primeiro gráfico, a largura de cada camada foi estabelecida a 10, a regularidade (ligações entre camadas) a 5, e o fator de peso a 1, variando o número de camadas. A camada de corte ficou estabelecida como a camada a meio da rede, isto é, "número de vértices" / 2. No segundo gráfico, a largura de cada camada foi estabelecida a 15, a regularidade a 8, e o fator de peso a 1, continuando a variar o número de camadas, e mantendo-se a camada de corte como "número de vértices" / 2. Os tempos de execução foram calculados com recurso à função *time* presente no Linux.



Em ambos os casos, consegue-se verificar que o tempo de execução aumenta com declive semelhante a uma função polinomial cúbica, o que coincide com a complexidade temporal teórica de $O(V^3)$. Aumentando a largura de cada camada e a regularidade, também se verifica que o tempo de execução aumenta. No primeiro gráfico, o programa demora 10 segundos a processar uma rede com aproximadamente 250 camadas, enquanto que no segundo gráfico apenas são precisas cerca de 175 para o mesmo tempo de execução.

5. 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
- https://www.geeksforgeeks.org/relabel-to-front-algorithm/
- https://fenix.tecnico.ulisboa.pt/downloadFile/1970943312332951/ch26a.pdf