**Instituto Superior Técnico**

Análise e Síntese de Algoritmos

**2º Projecto – Relatório**

**Introdução**

Este relatório aborda a nossa solução ao segundo projeto de Análise e Síntese de Algoritmos do 2º Semestre do ano curricular 2018/2019.

O Sr. Caracol providencia transporte de vários tipos de mercadorias provenientes de vários fornecedores até um hipermercado. Havendo pelo caminho várias estações para abastecer os meios de transporte e fazer o seu respetivo controlo. Os fornecedores produzem uma certa quantidade de produto e enviam o máximo que conseguem para o hipermercado.

Cada meio de transporte consegue transportar uma certa capacidade de produto para o hipermercado ou para uma estação de abastecimento que têm um limite máximo de produto que consegue processar.

O objectivo deste projecto é a partir de um input que indica o número de fornecedores (F), o número de estações (E) , o número de conexões (C), a produção de cada fornecedor, as capacidades das estações de abastecimento e a origem, destino e capacidade das várias conexões, obter a capacidade máxima que esta rede é capaz de transportar e quais as conexões mais perto do hipermercado que devem ser aumentadas para aumentar esta capacidade (corte mínimo).

**Descrição da Solução**

Este programa foi implementado em linguagem C.

Na nossa solução, foi usado um grafo dirigido com capacidades, em que invertemos o sentido das conexões e consideramos o hipermercado como a source para conseguirmos obter o corte mínimo. Criámos também um target ao qual todos os fornecedores estão ligados e estas conexões vão ter como capacidade a produção de cada fornecedor. As estações de abastecimento são representadas por dois vértices com id simétricos, onde a capacidade da conexão que os liga é a respetiva capacidade da estação.

Os vértices do grafo são a source, o target, os fornecedores e as estações de abastecimento (que ocupam dois vértices cada). As arestas são todas as ligações existentes entre a source, o target, os fornecedores e as estações de abastecimento.

Em termos de organização do grafo usamos diversas estruturas:

1. Estrutura geral, onde guardamos um vetor de ponteiros para os vértices, uma queue e um vetor de ponteiros para as alturas;
2. Vértice, que contem a sua altura, o seu excesso, o seu id e uma lista ligada com as suas adjacências;
3. Adjacência, que contem a sua capacidade, o seu flow e um ponteiro para o seu vértice de saída;
4. Lista ligada de nodes de vértices, com uma head (ponteiro para o início da lista) e uma tail (ponteiro para o fim da lista) (Usada para a queue);
5. Lista ligada de nodes das adjacências, com uma head e uma tail (Usada para as adjacências);
6. Nodes que contêm o vértice respetivo e um ponteiro para próximo node da lista (Usados na lista ligada da height e na lista ligada da queue);
7. Nodes que contêm a adjacência respetiva, um ponteiro para o próximo node na lista e um dual (ponteiro que aponta para o vértice com sentido contrário que é criado quando é enviado flow, por forma a este poder voltar a ser reenviado - Backedge) (lista ligada das adjacências).

Para encontrarmos o fluxo máximo e o corte mínimo baseamo-nos no algoritmo preflow-push (push relabel) modificado em que utilizamos um fifo e a gap heuristic para obter melhor desempenho.

Etapas para encontrar o fluxo máximo e o corte mínimo:

1. Inicializamos todos os vértices com o seu respetivo id, com excesso 0 e com altura 0 (exceto a source que tem altura igual ao número de vértices).
2. Adicionamos os vértices na respetiva lista ligada de alturas;
3. Inicializamos todas as adjacências com a sua respetiva capacidade e flow igual a zero e o dual a nil;
4. Função preflow, que satura a capacidade das adjacências da source e coloca essa capacidade no excesso desses vértices. De seguida iguala a flow há capacidade e cria os respetivos backedges com capacidade igual a 0 e flow simétrico ao flow do seu dual). Todos os vértices com excesso são então colocados na queue;
5. Enquanto a queue tiver elementos, ainda existem vértices com excesso, pelo que retiramos um vértice e aplicamos o discharge (6), caso contrário descobrimos o fluxo máximo e vamos imprimir o resultado (9);
6. Função discharge, enquanto existir excesso no vértice, irá verificar se alguma das suas adjacências verifica as seguintes condições: Ainda conseguir enviar flow e a altura do vértice origem da adjacência ser superior em um à altura do vértice destino dessa adjacência. Caso se verifique vai fazer push (7) para cada uma dessas adjacências, caso contrário vai fazer relabel (8) e de seguida volta a fazer este passo (6). Se já não tiver excesso voltamos para o ponto anterior (5).
7. Função push, primeiramente, verificamos o máximo de flow que ainda conseguimos enviar pela adjacência (maxf, que é mínimo entre o excesso do vértice origem e a flow que ainda pode ser enviada) e adicionamos o maxf à flow dessa adjacência, seguidamente, adicionamos o excesso ao vértice destino. Caso este ainda não tivesse excesso, adicionamo-lo na queue. Finalmente, se a adjacência não tiver ainda um dual, vamos criá-lo com o simétrico do maxf, se já tiver apenas subtraímos o maxf ao seu flow atual.
8. Função relabel, caso a altura do vértice seja menor que o número de vértices no grafo, vamos retirar o vértice da lista ligada dessa altura. Se esta lista ligada ficar vazia, então vamos aplicar a gap heuristic.

A gap heuristic consiste em colocar todos os vértices que encontramos das listas ligadas com altura superior, até encontrar uma lista ligada vazia ou ser menor que a altura igual ao número de vértices, na lista ligada da altura igual ao número de vértices (obtendo assim os vértices de um dos lados do corte nessa altura). Por fim é colocada a altura dos vértices igual ao número de vértices mais um.

No caso em que as condições para a gap heuristic não se verificam, apenas somamos mais um à altura do vértice e colocamo-lo na lista ligada de altura acima.

1. A capacidade máxima está presente no excesso do target, pelo que basta imprimir este resultado.
2. De seguida criamos dois vetores para poder guardar as estações e as conexões que podem ser aumentadas **(adicionar estruturas),** e procuramos na lista ligada de altura igual ao numero de vértices, todas as adjacências que tem a altura do vértice origem maior que a altura do vértice destino mais 1.
3. Caso essas adjacencias tenham id’s simétricos significa que são estações e são colocados no vetor respetivo, noutra situação são colocados no vetor das conexões.
4. É usado o qSort para ordenar ambos os vetores cada um usando uma função de comparação respetiva (ordenamos as estações pelo seu id e as conexões primeiro pelo id do seu vértice de origem e em caso de empate pelo seu vértice de destino) sendo posteriormente imprimidos de forma ordenada conforme pedido.

**Análise Teórica**

Em termos de tempo:

* Inicialização das estruturas e das variáveis - O(N), devido a termos de inicializar o vector de routers;
* Leitura de inputs e adicionar as adjacências à lista ligada – O(M), devido a termos de ler todas as conexões;
* 1º algoritmo – É chamado para cada router, O(N), e é chamado para cada uma das suas adjacências que, por ser um grafo não dirigido, é duas vezes o número de conexões, O(M).Assim a combinação das duas complexidades é O(N + M);
* 2ª algoritmo – O(N + M), pela mesma razão que o 1º algoritmo;
* Imprimir resultados – O(R), pelo facto de termos de imprimir o maior id de cada sub-rede;
* Fazer free – O(N + M), pela mesma razão que o 1º e 2º algoritmo;

Assim, o nosso projecto tem teoricamente complexidade temporal O(N+M).

Em termos de memória:

Temos um vector que tem N routers, O(N), e por sua vez cada um destes tem a sua própria lista de adjacências, que no total são 2M conexões, O(M). Não havendo mais nenhuma ocupação de memoria com uma escala de crescimento maior que esta podemos concluir que a complexidade espacial teórica é O(N + M).

**Avaliação Experimental**

Informações sobre a plataformas de testes - Virtual Box - Ubuntu 16.04 – Intel® Core™ i7-7700HQ – Clock Rate: 2.80GHz – 4 cores - 10Gb RAM.

Estes gráficos foram construídos com dados obtidos a partir de um script feito em bash shell que consistia em 2 ciclos que permitiu variar o N, o M e o R(100 ou 1000), de forma crescente.

Em termos de Tempo (130 testes):

Cada teste foi feito 1000 vezes e o tempo foi medido a partir do “user time” da função time.

Em termos de memória (250 testes):

Para cada teste, os picos de memória foram obtidos a partir da ferramenta massif do valgrind.

Resultados experimentais:

Os dois gráficos permitem constatar que o programa desenvolvido é, de facto, linear, dado que a memória e o tempo utilizados crescem proporcionalmente com o número de routers + número de conexões (N+M), comprovando assim os resultados obtidos teoricamente.

**Referências**

Como pontos de referência utilizamos:

* <https://www.youtube.com/watch?v=2kREIkF9UAs>;
* Introduction to Algorithms, Third Edition: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein