Análise e Síntese de Algoritmos - 2º Projeto

Grupo 16
Afonso Gonçalves - 89399
Daniel Seara - 89427

1 Introdução

O segundo projeto de ASA consiste na análise de uma rede de transportes de diversos fornecedores para estações de abastecimento e um único hipermercado. A análise consiste em calcular a capacidade máxima da rede, bem como as estações de abastecimento e ligações cuja capacidade limita a da rede. Devem ser apenas consideradas as ligações e estações de abastecimento mais próximos do hipermercado.

2 Descrição da Solução

Linguagem utilizada

Decidiu-se usar C++ mais uma vez para resolver este problema, pois é uma linguagem eficiente e tem uma boa biblioteca de estruturas de dados implementada, que seriam úteis para o projeto.

Modelação do problema

Este problema pode ser reduzido a um problema de fluxo máximo e corte mínimo, sendo a rede de transportes representada por uma rede de fluxo:

O hipermercado é representado pelo poço da rede. Cada fornecedor é representado por um vértice e as estações de abastecimento são representadas por dois vértices (*in* e *out*), ligados por um arco com capacidade igual à da respetiva estação de abastecimento. A fonte da rede liga-se a todos os fornecedores, por arcos de capacidade igual à produção desses fornecedores. Todas as ligações de transporte são representadas por arcos que ligam os respetivos pontos com a respetiva capacidade. Transportes com destino (respetivamente origem) em estações de abastecimento ligam-se aos vértices *in* (respetivamente *out*) destes.

A capacidade da rede de transportes é dada pelo fluxo máximo da rede e o conjunto de ligações/estações de abastecimento a serem aumentadas correspondem aos arcos saturados pelo fluxo máximo, ou seja, os arcos que não são respeitados pelo corte mínimo. Uma vez que se pretende calcular os arcos de aumento mais próximos do

hipermercado, considerou-se o fluxo máximo com fonte neste, usando-se, para tal, a rede transposta do problema. Use-se s (respetivamente t) para representar o hipermercado (respetivamente a fonte do grafo original).

Algoritmo usado

Para calcular o fluxo máximo, decidiu-se implementar o algoritmo Relabel-To-Front[2][3]. Este, para além de ser uma das opções com melhor complexidade[1], dá o valor do fluxo máximo em tempo constante e a função final de alturas permite calcular o corte mínimo em tempo linear.

Depois do Relabel-to-Front, o fluxo máximo é dado pelo excesso de t. O corte mínimo, por sua vez, é dado por *(S, T)*, sendo S o conjunto dos vértices de altura igual ou superior a S, e T=V-S.

Isto verifica-se uma vez que se um vértice u está a altura superior à de s, então é porque este não conseguiu descarregar todo o seu excesso para os seus outros vizinhos, tendo de enviar o seu fluxo para trás. Assim, o arco por onde esse fluxo vinha deixa de estar saturado, sendo respeitado pelo corte. Se uma adjacência v de u tem altura inferior ou igual à de s, então o arco *(u, v)* está saturado, sendo portanto um arco de aumento.

Deste modo, percorrendo todos os vértices e comparando as suas alturas com as dos respetivos vizinhos, consegue-se calcular os arcos de aumento em $O(V+E)$.

Representação do problema

Para minimizar os gastos de memória, representou-se cada arco por uma estrutura (*struct Edge*), onde se guarda a sua origem, destino, capacidade residual e o fluxo que nele passa. Deste modo, a informação não é duplicada, quando se consideram todos os vizinhos de cada vértice.

Os vizinhos de cada vértice são usados no Relabel-To-Front para efetuar a descarga de cada vértice. A lista de vizinhos é percorrida apenas num sentido e só se sabe o seu comprimento depois de se ler todo o input. Uma lista simplesmente ligada (*std::forward_list*) permite uma construção dinâmica dos vizinhos de cada vértice, guardando nela as referências para os respetivos arcos.

Durante a execução do algoritmo é necessário saber qual foi o último vizinho descarregado por cada vértice. Como se sabe, à partida, o número de vértices, guardou-se esta informação num array de iteradores da lista de vizinhos de cada vértice.

É ainda necessário manter uma ordem de descarga dos vértices. Esta ordem é mantida numa fila FIFO (*std::queue*) onde se guardam os vértices com excesso. Sempre que um vértice ganha excesso, é colocado no fim da fila. O primeiro vértice da fila é o próximo vértice a ser descarregado, sendo, nessa operação, removido dela.

As funções de altura e de excesso são arrays de inteiros (*int[V]*), uma vez que permitem um acesso constante a esta informação.

[1]A complexidade do algoritmo de Ford-Fulkerson depende do fluxo máximo. Todos os outros algoritmos são cúbicos, uma vez que se está a trabalhar com grafos esparsos

3 Análise Teórica

Em traços gerais, a solução proposta é dada pelo seguinte pseudocódigo:

Main()	***O(V ^{3})***
1. LerInput()	*O(E)*
2. InicializarDados()	*O(V+E)*
3. RelabelToFront()	*O(V ^{3})*
4. Analisar()	*O(Vlog(V))*

A leitura de input é linear e depende do número de arestas *O(E)*.

A informação é inicializada em dois momentos: Durante a leitura de input são guardadas as informações de cada arco, em tempo constante para cada. A inicialização das funções de alturas e de excesso, por sua vez, depende do número de vértices. Temos que a inicialização da informação é *O(V + E)*

O cálculo do fluxo máximo, através do algoritmo Relabel-To-Front tem complexidade *O(V ^{3})*[1], sendo a operação mais complexa da solução.

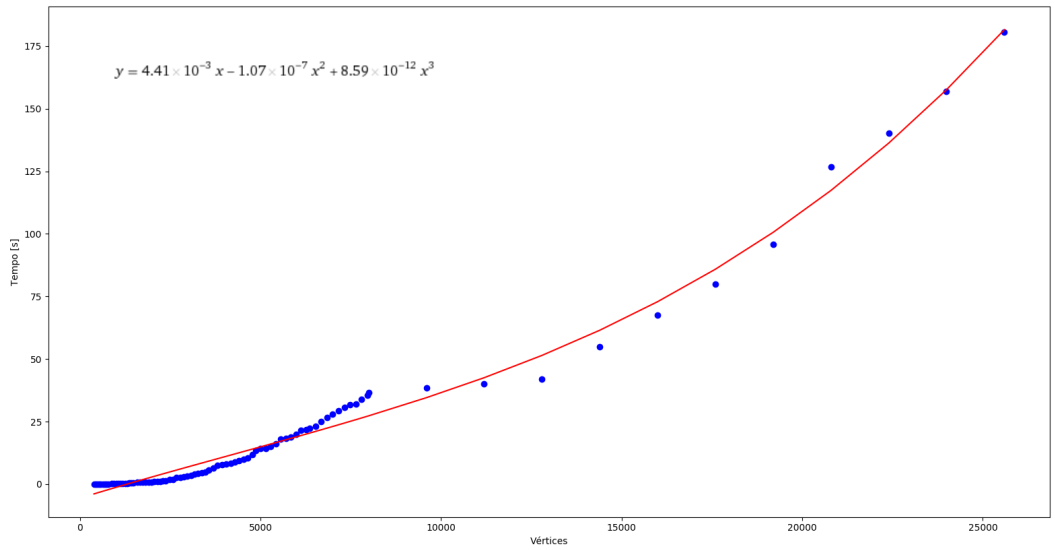
Finalmente, para exprimir a análise efetuada, são realizadas as seguintes operações: O fluxo máximo é acedido em tempo constante, à função de altura do poço da rede que modela o problema; Iterando linearmente os ID's das estações de abastecimento, ao verificar se os arcos que ligam o respetivo vértice *in* ao vértice *out* não são respeitados pelo corte, facilmente se descobre, por ordem crescente, as estações de abastecimento cuja capacidade tem de ser aumentada. Finalmente, são percorridas todas as adjacências da rede. Se estas não forem respeitadas pelo corte, serão adicionadas a uma lista simplesmente ligada em tempo constante. Depois de todas estas adjacências serem analisadas, a lista de adjacências não respeitadas pelo corte é ordenada em tempo *O(Vlog(V))*[4], sendo percorrida uma última vez para imprimir todos os arcos não respeitados pelo corte. Este último passo do algoritmo tem complexidade *O(Vlog(V))*.

Assim, o algoritmo tem, no total, complexidade *O(E) + O(V+E) + O(V ^{3}) + O(Vlog(V)) = O(V ^{3})*

4 Avaliação experimental

A solução proposta passa com sucesso os testes disponibilizados e os 16 testes do Mooshak. Apresenta-se em baixo o tempo de execução da solução em função do número de vértices da rede, variando estes entre 100 e 26 000. Uma regressão polinomial prova a complexidade cúbica da solução implementada.

É de notar que existe um amortizar do valor do coeficiente da regressão na zona dos 9000 vértices. Isso deveu-se a uma pequena variação nos parâmetros do gerador, para permitir testes com maior carga. Mesmo assim, a complexidade não foi afetada.



Vértices	Arestas	Memória[kB]	Tempo[s]
392	1960	189	0.02
1982	19820	1098	0.92
6127	104160	5248	22.30
15994	111952	6073	67.49
22394	156752	8473	140.25
25594	179152	9673	180.52

Referências

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, Clifford Stein. *Introduction to Algorithms* MIT Press, 1989

[2] The relabel-to-front algorithm:
`http://www.euroinformatica.ro/documentation/programming`

[3] Push–relabel maximum flow algorithm:
`https://en.wikipedia.org/wiki/Push%E2%80%93relabel_maximum_flow_algorithm`

[4] std::forward_list::sort
`http://www.cplusplus.com/reference/forward_list/forward_list/sort/`