Fluxo Máximo

Luiz Otávio Resende Vasconcelos

Maio 2017

1 Introdução

O problema a ser resolvido diz respeito à um problema clássico conhecido na Ciência da Computação como Fluxo Máximo.
Devemos modelar um algorítmo o qual, em um grafo dirigido com peso em suas arestas, encontre maior valor possível partindo de uma origem até um destino. Podemos imaginar o problema como canos de água conectados.

O presente trabalho foi desenvolvido na linguagem C.
Para executá-lo, basta utilizar os comandos em um bash/unix:
$ make
$ ./max_graph_flow

O projeto completo e as etapas do desenvolvimento podem ser encontrados no repositório:
https://github.com/Luiz0tavio/max_graph_flow

2 Implementação

2.1 Estrutura de Dados

A estrutura de dados principal é um grafo dirigido.

Cada vértice do grafo é constituida de um struct o qual contém as propriedades:

type : determina se aquele vértice é uma Franquia, um Cliente, ou nenhum dos dois.

edges : array de ponteiros para as arestas que partem do deste vértice.

edges_size: contador para o tamanho do array arestas.

E cada aresta é constituída de um struct com as propriedades:

to : ponteiro para a o vértice que a respectiva aresta está apontando.

size: valor da aresta.

Os índices descritos acima referem-se à um array que armazena os structs. O grafo é montado apenas com um elemento fazendo referência ao outro. A escolha da utilização de índices e não posições de memória é devido à utilização de **realloc** (o qual move todo o conteúdo de lugar na memória e da free na memória antiga, logo, seria necessário atualizar cada endereço após o realloc o que seria muito custoso).

2.2 Algorítmo

O método para o cálculo do fluxo máximo é conhecido como **método de Ford-Fulkerson** (descrevo como 'método' e não algorítmo porque ele abrange diversas implementações com diferentes tempos de execução). Porém, com uma melhoria: utilizando busca em largura, é denominado **algorítmo de Edmonds e Karp**, pois, a busca em profundidade permite que o algorítmo caia em loop infinito.
Métodos descritos no capítulo 26.2 do livro **Introduction to Algorithms - Thomas H. Cormen**

2.2.1 Busca em profundidade

Retorna um ponteiro para um array de índices da árvore que será o caminho a ser percorrído pelo Ford-Fulkerson (ou Edmonds e Karp). A busca em profundidade implementada é um pouco não usual. É uma busca recursiva que recebe um índice de recursão. A cada etapa de recursão ela subtrai 1 desse índice e só retorna o resultado quando chegar a 0.
Ou seja, se passarmos 1 como índice de recursão, a busca irá olhar os filhos do vértice de inicio. Se passarmos 2, irá olhar apenas os filhos dos filhos e assim por diante.
Mais à frente, na análise de complexidade, veremos que não é um método muito otimizado. Porém, a implementação de um índice de recursão foi útil em outros aspectos, por exemplo, saber o tamanho do caminho que será retornado.

2.2.2 Cálculo

Etapas do método:

1 - Através de uma busca em profundidade, encontrar um caminho aumentador do início ao fim do grafo.

2 - Encontrar o menor valor de fluxo dentre esse caminho.

3 - Somar esse menor valor à um contador total, que será o resultado final.

4 - Subtrair dos vértices do caminho esse menor valor.

5 - Adicionar uma aresta oposta à cada aresta do caminho com o menor peso.

3 Estrutura do Projeto

Apesar do código estar bem comentado, segue uma breve exposição de como o projeto está particionado.

3.1 main.c

Arquivo principal do projeto. Inicializa as varíaveis, faz a leitura das entradas, faz a chamada das principais funções e exibe o resultado.

3.2 graph.h / graph.c

Responsável por criar e realizar operações com a estrutura de dados componente de cada vértice e aresta do grafo.

Também é onde está a função recursiva irá encontrar os caminhos aumentadores.

3.3 tools.h / tools.c

Arquivo para funções auxiliares.

4 Análise Assintótica

4.1 Temporal

Ignorando as etapas de leitura das entradas, temos dois loops principais: um de tamanho **f**, onde f é o número de Franquias, e outro de 1 até o número de vétices.
A busca em largura será executada para cada uma das f*(V-1) iterações.
A busca em largura possui complexidade $\theta(\frac{V^2}{2})$, pois, ela executará, no loop de 0 até V, i operações, onde i é o índice do loop, ou seja, o total de operações será a progressão de aritimética de 0 até V.

O índice do segundo loop será nosso tamanho do caminho (também o tamanho do loop de recursão). Caso ela encontre um caminho válido, será feito o cálculo do menor caminho aumentador que custa **p*a**, onde p é tamanho do caminho retornado pela busca em largura e a é a quantidade de arestas em cada vértice do caminho, e então fará a atualização dos vértices na árvore também com o custo de **p*a**.
Logo, temos que nosso algorítmo terá, no caso médio $\theta(\text{f*(V-1)}\frac{V^2}{2}\text{(2pa)})$

4.2 Espacial

Tanto o struct de Vértices quando o de Areastas possuem 8 bytes.
Portanto, esperamos alocar (8*Arestas + 8*Vértices)bytes.
Teremos também, um array de inteiros de 0 até (sizeof(int)*path,length), ou

seja, um array de inteiros com os índices do caminho a ser percorrido pelo Ford-Fulkerson.
E por último devemos levar em conta as arestas de volta dos caminhos aumentadores.
Logo, esperamos que nosso algorítmo ocupe cerca de **(8*Arestas + 8*Vértices + (sizeof(int)*path.length) + p*8)bytes**, onde p é o número de caminhos aumentadores.

5 Conclusão

Problemas envolvendo fluxo máximo tem vasta aplicações na vida cotidiana. Desde o desejo de uma empresa maximizar sua rede de distribuição até um indivíduo que deseja uma determinada vazão em seu encamento, ou até mesmo maximizar o fluxo de veículos.
Portanto, de grande importância a compreensão e conhecimento da modelagem e dos principais algorítmos.