Documentação do Trabalho Prático 1

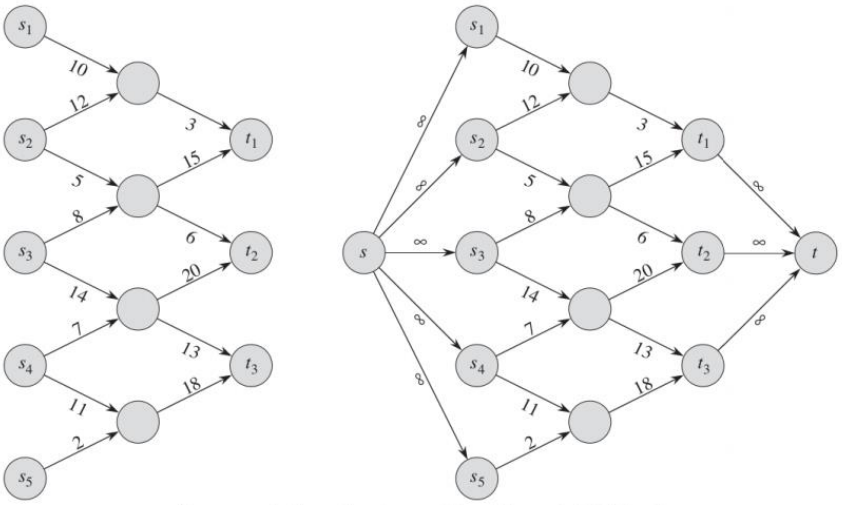
Luísa Ribeiro Bezerra

Maio, 2016

1. Introdução

O objetivo principal deste trabalho é trabalhar nossa perspectiva e aprendizado sobre Grafos e suas utilidades, estudando, assim, a implementação de algoritmos para resolver problemas de Grafos - neste caso, *direcionado*. Neste trabalho, foi-nos designado a ideia de *Fluxo Máximo*: segundo a introdução do trabalho, teríamos de construir um mapa da cidade, criando as interseções (vértices do Grafo) e ciclofaixas (ou arestas do Grafo) da mesma para analisar qual o limite de ciclistas que deveriam trafegar em cada ciclofaixa por hora. Como entrada do problema, temos então que algumas das interseções poderiam ser *franquias* da empresa de entregas de lanches operada por ciclistas, a Rada, e outras poderiam ser *casas* dos clientes que seriam o destino das encomendas da empresa Rada (nunca sendo ambas, e podendo ter interseções que não representasse nenhum dos dois). A nossa tarefa foi definir a logística desta rede de conexões, ou seja, entregar para Rada qual é a quantidade máxima de ciclistas que pode sair das franquias por hora de modo que a segurança dos ciclistas seja levada em conta – entende-se por "segurança" o limite de ciclistas que deveriam trafegar em cada ciclofaixa por hora.

A ideia que desenvolvi para tratar do problema foi baseada no algoritmo de Ford-Fulkerson. O algoritmo de *Ford-Fulkerson* é um algoritmo utilizado para resolver problemas de fluxo em rede (*network flow*). O algoritmo é empregado quando se deseja encontrar um fluxo de valor máximo que faça o melhor uso possível das capacidades disponíveis na rede em questão. Porém, como nos foi dado múltiplas entradas (franquias) e saídas (casas), decidi por também implementar a ideia da *super-origem/super-destino*: adicionei uma nova "super-origem", ligada por arestas de capacidade infinita a todas as outras origens, e um novo "super-destino", que recebe arestas de capacidade infinita vindas de todos os outros destinos. Assim, terei que somente encontrar o fluxo máximo entre a super-origem e o super-destino, o que facilita a execução e entendimento do problema. Na imagem abaixo, vemos um exemplo de um Grafo direcional implementado com a super-origem "s" e super-destino "t".



(imagem de Introduction to Algorithms, 3rd Edition)

2. Solução do Problema

2.1. Modelagem

Para a estruturação do problema, criei algumas estruturas para apoio, como a estrutura de uma fila, que será utilizada no algoritmo de *Busca em Largura* – bem como a criação de funções de *enfileirar* e *desenfileirar*. Foi interessante, neste caso, usar a ideia de *fila,* já que o algoritmo de Busca em Largura deve garantir que nenhum vértice ou aresta será visitado mais de uma vez enquanto percorre o Grafo dado. Dessa maneira, as visitas aos vértices são realizadas através da ordem de chegada na fila e um vértice que já foi marcado não pode entrar novamente a esta estrutura. Usei, para tal, a analogia estudada em sala de aula de determinar "*cores*" para os vértices - no meu algoritmo, a cor "branca" seriam vértices ainda não marcados e não enfileirados, ou seja, seu visitado[vertice] teria valor de 0; a cor "cinza" seriam vértices que estão na estrutura da fila, ou seja, seu visitado[vertice] teria valor de 1; a cor "preta" seriam vértices que já tiveram todos os seus vértices vizinhos já visitados/enfileirados, ou seja, seu visitado[vertice] teria valor de 2. Este vetor ao qual me refiro foi criado para guardar o "estado" de cada vértice do Grafo seguindo a lógica descrita (ou seja, de tamanho vértice + 2). Com esta estrutura descrita em alto nível, teremos então uma Busca em Largura executando caminhos possíveis e os retornando para serem usados no algoritmo de Ford-Fulkerson.

O algoritmo de Ford-Fulkerson é baseado na inicialização de uma matriz que representa o fluxo. Após isso, zera-se a matriz, pois irei aumentar gradativamente o valor de fluxo acrescentando-o na fonte através da lógica do algoritmo. Para achar um caminho válido, usamos a Busca em Largura acima explicado no grafo. Com tal instrumento, temos a geração de caminhos, que vão retornando para o algoritmo de Ford-Fulkerson os caminhos válidos em um vetor de inteiros, que guarda o caminho, seguindo o índice do vértice, assim é possível de ser avaliado o fluxo. Percorre-se tal vetor com finalidade de encontrar o fluxo máximo local. Agora, é preciso atualizar o grafo de fluxos, e então subtraímos o fluxo do caminho de todas as arestas (ciclofaixas) ao longo deste caminho e adicionamos o fluxo do caminho no sentido inverso ao dado na direção do grafo. No fim, é retornado o fluxo máximo.

2.2. **Implementação**

O fluxo do programa é iniciado no ***main.c*** com a leitura da linha dada como entrada padrão para o problema, a qual contém 04 inteiros: o número de vértices (interseções), o número de arestas (ciclofaixas), o número de franquias (origens) e o número de clientes (destinos). Após esta leitura, crio uma matriz de inteiros que, no caso, é o Grafo (mapa da cidade). É notório que criei com um tamanho de 2 linhas e colunas a mais que o necessário, e o faço para justamente implementar a ideia da super-origem e super-destino. Após isso, é dado um número de linhas que corresponde ao número de arestas (assim sendo, fiz um loop para a leitura das mesmas baseada no número de vértices+2) , as quais dão 03 inteiros de entrada: um vértice origem, um vértice destino, e o peso (fluxo máximo) daquela aresta, ou seja, daquela ciclofaixa. Após isso, faço dois loops, uma iteração que percorre de 0 ao valor de franquias dadas que, em cada iteração, leio da entrada o vértice que é franquia, e o mesmo para clientes. Também crio a variável resposta, que recebe o valor de retorno da função "fordFulkerson" e, assim que recebe o valor, printa o mesmo como saída. Ao final, libero a memória do grafo que está alocado dinamicamente.

No módulo " *ff* ", temos a declaração da função fordFulkerson, que recebe como parâmetros de entrada o número de vértices e o grafo. Em sua execução, crio um grafo residual, chamado de "fluxo" e o inicializo também. Crio, em seguida, o vetor caminho, alocando-o. Entra então um while que é assumido verdadeiro quando a Busca em Largura (função buscaLargura) retorna 1, que significa que achou um caminho da super-origem ao super-destino. Dentro do while, crio um incremento máximo de valor máximo e um for onde começo do último vértice até o primeiro, olhando os fluxos e determina o incremento máximo de acordo com o fluxo do caminho. Após isso, no grafo residual, substituo as arestas existentes com o valor do fluxo, que é o valor anterior mais ou menos o incremento, de acordo com a direção que estamos seguindo. Ao final, temos, então, o fluxo máximo.

No módulo " *buscaLargura* ", temos a função buscaLargura que recebe como parâmetros de entrada o número de vértices, o vetor caminho, o grafo e o grafo de fluxo (grafo residual). Nesta função, determino superOrigem como o número de vértices+1 (já explicado tal determinação) e superDestino como o número de vertice+1. Crio uma fila dinamicamente, usando o módulo " *fila* ", e crio um vetor de visitados (de tamanho vertice+2). Aqui, a lógica que expliquei anteriormente se aplica: este vetor é inicializado como 0, e 0 significa que aquele vértice não foi visitado ainda. Para inicializar a lógica, enfileiro a super-origem e, na função enfileirar, temos que o valor de visitado daquele vértice muda para 1, que significa que ele foi visitado, porém ainda há vizinhos a serem visitados. O valor de caminho torna –1, uma vez que ele é o ponto de partida. Temos, então, um while, que enquanto ainda houver uma fila, irá desenfileirar o primeiro elemento, e isso faz com que o valor de visitado daquele vértice muda para 2, que significa que ele e todos os seus vizinhos foram visitados. Seguindo o código, temos um for que percorre por todos os vizinhos do vértice desenfileirado e enfileira todos os vizinhos válidos, sendo que os válidos são os vértices com valor 0 em visitado e que tenham um valor de fluxo que não excedeu a capacidade ainda. Ao final, temos um check que recebe o visitado do último vértice para conferir que o algoritmo conseguiu chegar ao destino (no caso, super-destino) e que ele seja igual a 2.

3. Análise de Complexidade

Nesta seção, será apresentada a análise do custo teórico de tempo e de espaço dos principais algoritmos e estruturas de dados utilizados. Portanto, foram analisadas as funções abaixo contidas na tabela:

fordFulkerson	A complexidade de tempo é O (fluxo_maximo*numero de arestas), já que executamos um loop enquanto nos for dado um caminho existente. No pior caso, podemos adicionar 1 unidade de fluxo em cada iteração - a complexidade do tempo torna-se O (fluxo_maximo*numero de arestas). Neste caso, como usamos também o Busca em Largura com o Ford Fulkerson, temos que será O(vertices * numero de arestas ^{2} * fluxo_maximo)
buscaLargura	A complexidade do tempo do algoritmo da Busca em Largura pode ser expressa como O (numero de vértices + número de arestas), uma vez que cada vértice e cada aresta serão explorados no pior caso. A complexidade de espaço é O(numero de vertices), ou seja, O(n).
criaFila	A função somente inicializa a fila com ponteiros nulos, o que traz uma complexidade de O(1).
enfileira	A função insere o elemento no fim da fila, ou seja, não há operação nada custosa nesta situação. Como ele somente muda o endereço de onde seus ponteiros apontam, esta função é vista como O(1).
desenfileira	A função retira primeiro elemento da fila, ou seja, não há operação nada custosa nesta situação. Esta função é vista como O(1).

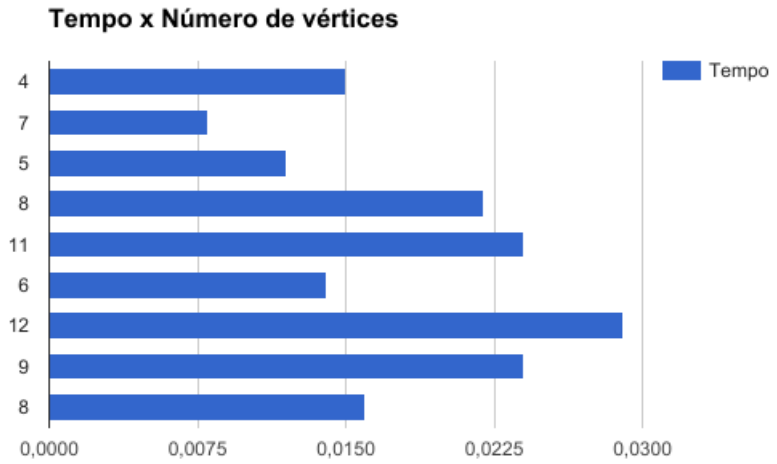
A **complexidade total do programa** é a maior complexidade dentre as citadas acima. Temos, então, ela como O(vertices * numero de arestas ^{2} * fluxo_maximo).

4. Avaliação Experimental

Para a avaliação experimental, utilizei os testes "toy" cedidos pelo professor, bem como também testes desenvolvidos por mim mesma e colegas da disciplina. Os experimentos foram executados em um sistema com 8 GB de memória RAM e um processador Intel Core i7-3632QM CPU @ 2.20GHz.

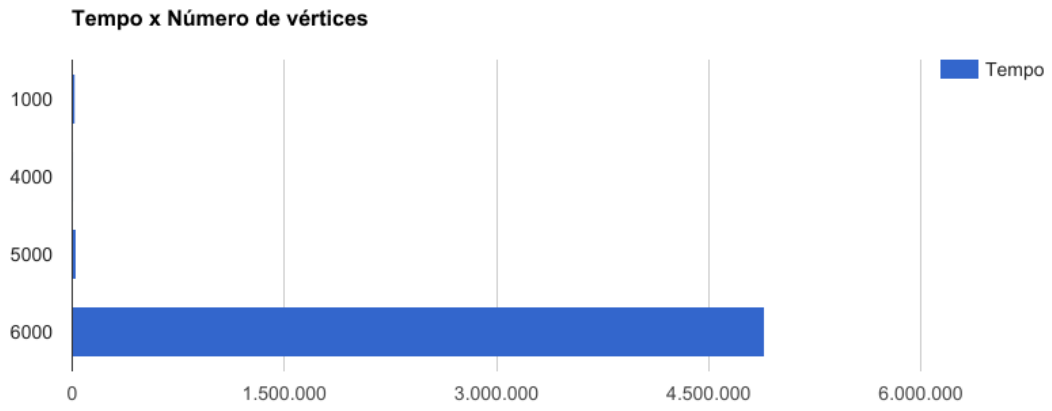
4.1. Testes Toy

Para os testes Toy fornecidos pelo professor, criei um gráfico baseado no tempo de execução versus o número vértices de entrada fornecida. Sempre que repeti o teste para a mesma entrada, o tempo de execução foi bem aproximado, o que mostra que o algoritmo é bem estável. Foram repetidas 10 vezes cada teste e feito a média deles. Segue o gráfico com a média dos tempos (em segundos) de execução:



4.2. Testes Desenvolvidos

Para os testes criados por mim e pelos colegas, criei um gráfico baseado no tempo de execução versus o número vértices de entrada fornecida. Sempre que repeti o teste para a mesma entrada, o tempo de execução também foi bem aproximado, o que mostra que o algoritmo é bem estável. Foram repetidas 10 vezes cada teste e feito a média deles. Segue o gráfico com a média dos tempos (em milisegundos) de execução:



5. Conclusão

Por mais que eu tenha tentado otimizar o código, quanto maior o número de vértices de entrada, mais custoso fica para termos um resultado em questões de tempo de execução. Mas o que também chama atenção é que, com o maior número de vértices e de arestas, maior o número de comparações - inclusive o maior número de arestas tem grande poder de custo, uma vez que ela aumenta o número de permutações possíveis, e ela que determina praticamente todo o custo do mesmo, então acaba sendo realmente custoso a entrada grande, sendo de vértices e/ou arestas - e o que também explica o crescimento exagerado das barras nos gráficos dos testes que realizei (4.2). É visível que o aumento das permutações criou um aumento exorbitantemente grande no gráfico, uma vez que aumentou o número de permutações.

O trabalho foi fundamental para a aplicabilidade de teorias que aprendi durante o curso - implementação de fila, do algoritmo de Busca em Largura e do algoritmo de Ford Fulkerson. Para ter um custo menor do algoritmo, pude trabalhar a ideia de custo em geral e tentar otimizar o tempo de execução, bem como a complexidade, do meu trabalho. Assim sendo, usei muito da criação de elementos dinâmicos para meu trabalho, como matrizes e vetores. Achei a proposta muito interessante e pude, com toda clareza, desenvolver minhas habilidades de lógica e programação.