Universidade Federal de Minas Gerais

Ingrid Rosselis Sant'Ana da Cunha - 2016430936

Algoritmos e Estrutura de Dados 3

Trabalho Prático 1

Belo Horizonte

2017

Introdução:

O propósito deste trabalho é o de implementar um algoritmo para calcular o fluxo máximo de um grafo dirigido e valorado que pode possuir vários vértices iniciais e vários finais. A ideia é tratar o grafo como uma rede onde deve-se passar o máximo de unidades de fluxo possível por cada aresta do início até o fim de modo que o valor total de unidades passadas pelo caminho seja máximo.

As duas condições básicas para o problema de fluxo máximo são: a capacidade de cada aresta não pode ser violada, isto é, não pode existir um fluxo maior que uma capacidade ou existir fluxo em uma aresta de capacidade nula; e o vértices não retêm fluxo, logo o fluxo total de chegada a um vértice e o de saída deste mesmo vértice devem estar em equilíbrio.

Para este fim, foi definida a utilização do algoritmo de Edmonds-Karp, uma variação do algoritmo de Ford-Fulkerson que utiliza busca por largura para encontrar o caminho mais curto entre um vértice inicial e um final.

Metodologia:

Para a atender à modelagem do problema foi utilizada a implementação de grafo por lista de adjacências, que é dividida em duas estruturas básicas: listas de adjacências e vetor de listas. O projeto feito foi dividido em 5 arquivos, cada dois para uma estrutura e um para a main. Os arquivos adjlist.h e adjlist.c possuem a estrutura e a implementação da lista de adjacências, os arquivos graph.h e graph.c possuem a estrutura e a implementação de um vetor de vértices (grafo) e o tp1.c possui a main. Todas as funções utilizadas neste trabalho se referem a um tipo de dado específico, então não foram criados arquivos para conter funções de propósito gerais.

Na implementação, as listas de adjacência representam as arestas ligadas a determinado vértice e contêm o índice do vértice adjacente e a capacidade daquela aresta. O vetor de listas representa os vértices do grafo, onde cada posição do vetor pode ou não ter uma lista de adjacências.

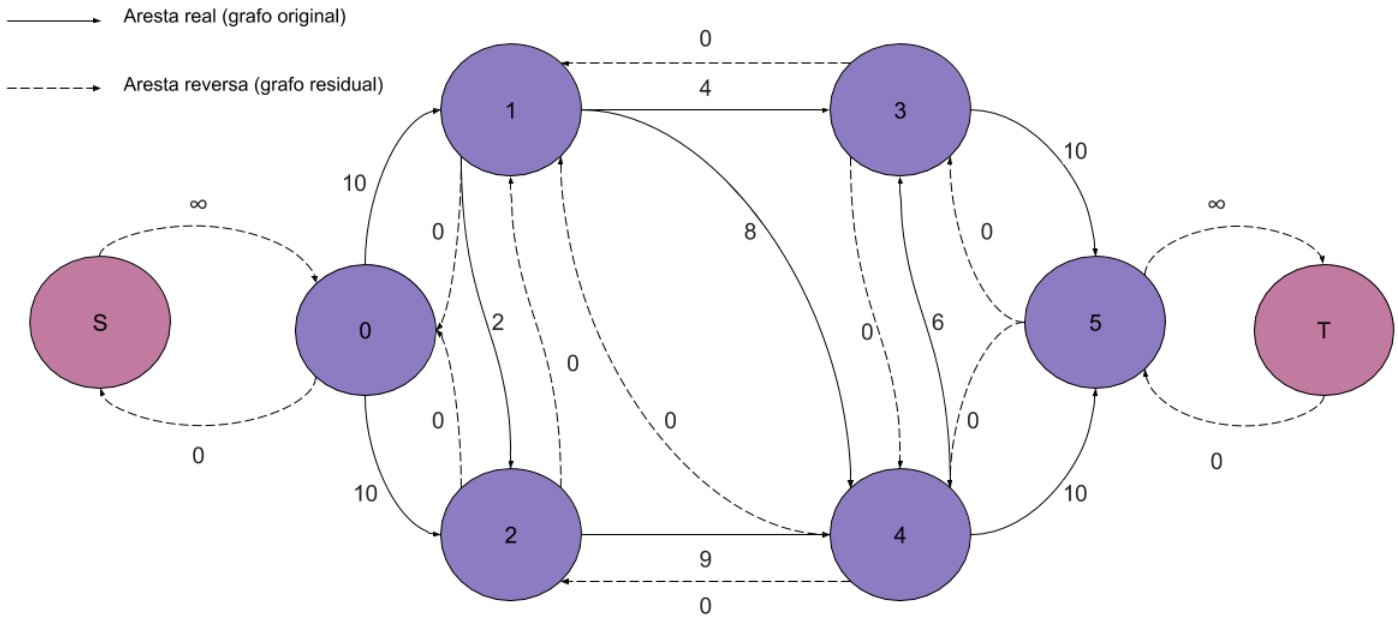
A estrutura original de grafos utiliza listas de adjacências encadeadas e o vetor de vértices contém o índice do vértice alocado, porém, para este trabalho, algumas alterações foram feitas a fim de que a estrutura atenda o que é pedido pelos algoritmos utilizados no programa e que possua um desempenho maior. As modificações são:

- Não foi considerado necessária a utilização de uma lista encadeada para o tratamento da lista de adjacências pois o grafo não é alterado durante a execução. Assim, essa estrutura foi implementada como um vetor dinamicamente alocado de acordo com o número de vértices adjacentes ao vértice em questão, isto é, baseado no número de vértices diretamente alcançáveis por cada nodo.
- O algoritmo de Edmonds-Karp necessita de um grafo residual para sua implementação, por isso foi decidido que o grafo original do problema seria utilizado como grafo residual durante a execução, sem a necessidade da criação de outro grafo e/ou de uma outra estrutura de dados.
- Como o problema pode possuir vários vértices iniciais e vários vértices finais foram criados dois vértices lógicos padrões para serem usados como o início e fim. A fim de manter o acesso ao vértice por complexidade O(1), esses vértices são inseridos como os dois últimos do grafo. Por convenção, o vértice inicial é o primeiro após os vértices do grafo e vem seguido pelo vértice final. Seus pesos são considerados infinitos (na realidade são o valor do maior inteiro possível) para não alterar a solução do problema.
- O grafo residual necessita também da criação de uma aresta lógica de peso 0 na direção contrária a toda aresta real inserida no grafo. Assim, a cada inserção de aresta em um vértice, é também inserida

a aresta reversa, trocando-se os vértices de chegada e saída e zerando a capacidade dessa aresta. Isso vale inclusive para os vértices inicial e final.

O algoritmo de Edmonds-Karp utiliza os conceitos de capacidade residual, caminho aumentador e grafo residual. A capacidade residual é a quantidade de fluxo que ainda pode ser passada pela arestas, ou seja, é a capacidade total da aresta menos o fluxo atual. O caminho aumentador é um caminho simples que não possui arestas originais de capacidade residual cheia e não possui arestas reversas de capacidade residual nula, ou seja, que ainda possui capacidade para passar fluxo. O grafo residual é uma versão do grafo original, mas possui arestas reversas de capacidade 0.

Segue imagem ilustrando um grafo dirigido e valorado, com arestas reversas nulas (situação inicial) e os vértices inicial e final:



O algoritmo de Edmonds-Karp pode ser resumido em 3 passos:

1. Deve-se encontrar o menor caminho aumentador do vértice inicial ao vértice final utilizando a busca em largura. Para que haja um caminho entre um vértice e outro (e seja o menor naquela execução) existe a condição básica da BFS de que o vértice de chegada da aresta não deve ter sido visitado ainda. Para o caso do fluxo máximo também foi adicionada a condição de que a capacidade da aresta que conecta os dois vértices deve ser positiva e não nula, pois caso contrário a condição de que o fluxo não viole a capacidade das arestas pode ser quebrada e, para piorar, o algoritmo rodaria sem fim.
2. Procura-se o gargalo da solução da BFS, ou seja, a aresta que possui menor capacidade residual entre as que compõem o caminho. Este passo é necessário para que o fluxo não ultrapasse nenhuma capacidade e para que não sejam gerados caminhos a mais ou a menos nas execuções seguintes do algoritmo.
3. Por fim, o caminho aumentador encontrado e o valor do fluxo máximo são atualizados com o valor do gargalo. Para essa etapa, o fluxo do gargalo é subtraído da capacidade residual de cada aresta e é somado à capacidade residual da aresta reversa.

Análise de Complexidade:

Ao contrário do algoritmo de Ford-Fulkerson que possui complexidade $O(E|f|)$, onde |f| é o fluxo máximo em cada iteração, o algoritmo de Edmonds-Karp possui complexidade temporal conhecida de $O(VE^2)$, onde V é o número de vértices e E é o número de arestas. Essa complexidade vem primeiramente do fato de que a cada iteração um caminho aumentador pode ser obtido em $O(E)$, a complexidade dada pela busca em largura. O que diferencia os dois algoritmos é a complexidade relacionada ao caminho aumentador.

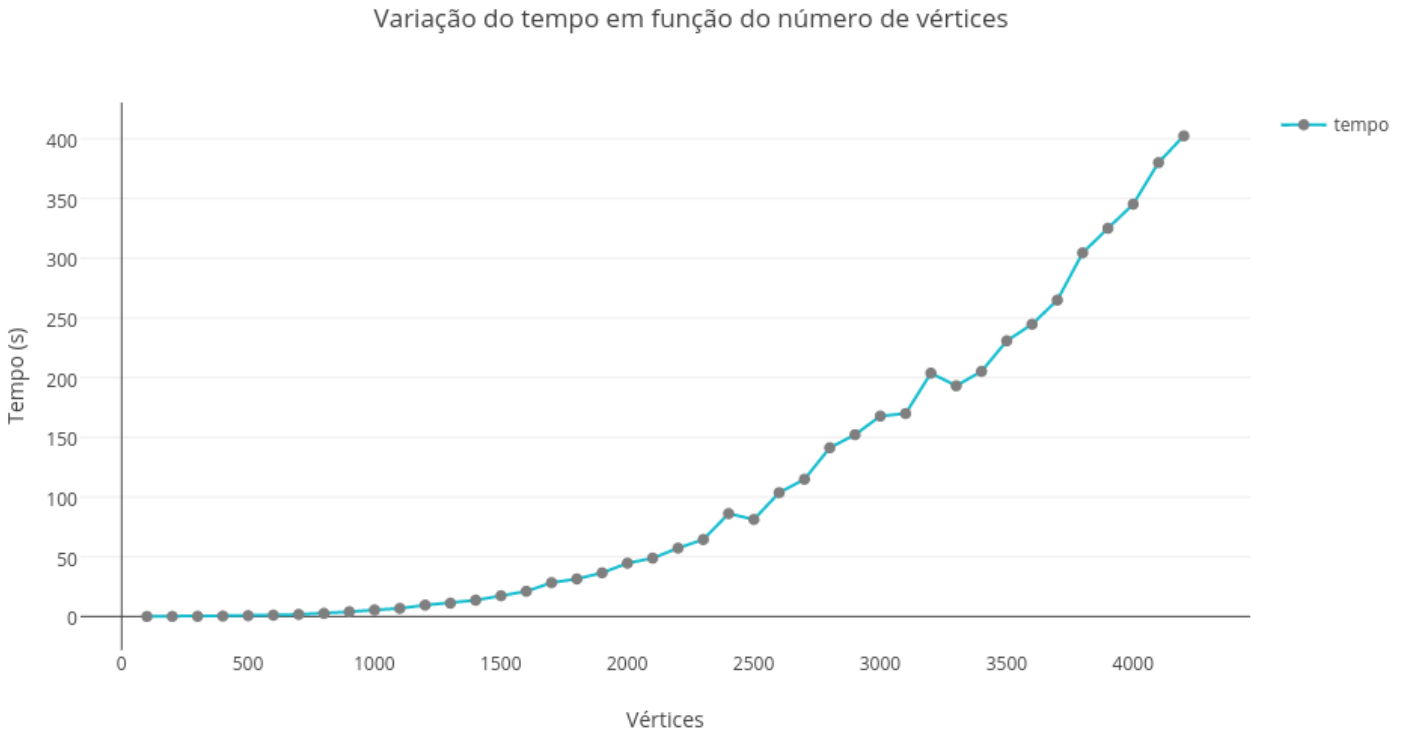
Com relação ao caminho, sabe-se que ele pode ser aumentado durante a execução, mas nunca encurtado. Para averiguar esse fato suponha dois vértices, s (inicial) e t (final). Na primeira execução do algoritmo, o caminho aumentador entre s e t é o menor possível, garantido pela busca em largura. Nesta execução, uma ou mais arestas do caminho s-t serão inviabilizadas pelo algoritmo de fluxo máximo. Então na próxima execução o caminho aumentador entre s e t deverá ser no mínimo uma unidade maior que o anterior e no máximo V - 1 unidades. Ou seja, qualquer aresta é inviabilizada no máximo $O(V)$ vezes. Supondo E arestas, o máximo de momentos onde qualquer aresta é inviabilizada é $O(VE)$.

Como em cada caminho aumentador ao menos uma aresta deverá ser inviabilizada e a cada iteração do algoritmo um caminho aumentador é encontrado, o programa não terá mais que $O(VE)$ iterações, e cada uma tem um custo de $O(E)$ para encontrar o caminho. Assim, o custo total do algoritmo é $O(VE^2)$.

As funções auxiliares do projeto não possuem não possuem complexidade maior que $O(V)$, casos onde a lista de vértices deve ser verificada. Na maioria das funções a complexidade tem relação com o número de vértices adjacentes, que é menor que o número total de vértices do grafo. Assim, o maior custo temporal no trabalho possui complexidade $O(VE^2)$.

A complexidade espacial neste trabalho é definida pela estrutura de dados do grafo e busca em largura, ambas com complexidade $O(V + E)$.

Avaliação Experimental:



Para proceder a avaliação experimental foram utilizados um total de 92 casos de teste feitos por colegas de turma e disponibilizados aos demais. O motivo para o uso desses casos de teste foi o fato de que eles possuem valores de número de vértices, número de arestas e pesos extremos e permitiram uma visão mais clara dos gráficos do que os testes que foram anteriormente utilizados.

Foi notado durante o trabalho que E fica próximo a $O(V)$ quando o grafo é esparso e a $O(V^2)$ quando é denso. Dessa forma, a complexidade total pode variar, aproximadamente, de $O(V^3)$ a $O(V^5)$ dependendo do grafo.

Os tempos obtidos para estes testes foram considerados satisfatórios, principalmente dado o tamanho das entradas. Além disso, a parte mais interessante de se reparar, na opinião da autora, é que ambos os gráficos possuem forma parecida, ambos denotam uma complexidade polinomial e que não houveram muitos picos na variação de um resultado para outro. Os que aconteceram são resultado das alterações na entrada.

Conclusão:

O intuito do Trabalho Prático 1 era modelar um grafo baseado no problema proposto e calcular o fluxo máximo nesse grafo. A solução usada foi o tradicional algoritmo de Edmonds-Karp, da década de 70. Este algoritmo é uma modificação do algoritmo mais simples para cálculo de fluxo máximo, Ford-Fulkerson, e utiliza busca em largura para ter uma complexidade temporal de $O(VE^2)$, melhor que a complexidade $O(E|f|)$ do algoritmo de Ford-Fulkerson, onde |f| é o fluxo máximo no grafo a cada iteração.

As maiores dificuldades nesse trabalho foram decidir qual algoritmo usar (entre Edmonds-Karp e Dinic) e escolher a implementação de grafo a ser utilizada, para que facilitasse a implementação do algoritmo e melhorasse o desempenho do programa.

A escolha da implementação foi considerada satisfatória pois vários pontos do algoritmo foram facilitados e/ou otimizados com as oportunidades que a implementação ofereceu. E o motivo pelo qual Edmonds-Karp foi escolhido é principalmente pela facilidade de implementação e de entendimento se comparado ao algoritmo de Dinic.