Alocação de Tarefas em Grafos Bipartido: Algoritmos Ford-Fulkerson e Edmond-Karp.

Ibukun C.D ADJITCHE
Dept.de Ciencia da Computação
DCC-UFRR
Boa Vista, Brasil
adjitchedidier1@gmail.com

Larissa S. SILVA
Dept.de Ciencia da Computação
DCC-UFRR
Boa Vista, Brasil
laryysantos18@gmail.com

Victor B. ROCHA
Dept.de Ciencia da Computação
DCC-UFRR
Boa Vista, Brasil
victor.barboza.15@gmail.com

***Abstract*—The paper here presented discusses Bipartite Graphs modelled as Flow Networks to represent tasks allocation in order to calculate the maximum flow. It uses the standard Ford-Fulkerson method, with augumenting paths, minimun cuts and residual network, and its Edmond-Karp implementation, which uses Breadth-first search.**

***Index Terms*—grafos, algoritmos, Ford-Fulkerson e Edmond-Karp**

I. Introdução

Os Algoritmos, como sequências finitas de passo na resolução de tarefas, são bastante trabalhados, estudados e discutidos na área da ciência da computação e no processo de optimização de recursos computacionais, podendo serem aplicados em vários paradigmas de modelagem de problemas. Entende–se como paradigmas de modelagem a forma a estrutura que possibilita e torna compreensível um assunto ao um determinado grupo. Aqui neste caso são computadores.

Na computação vários problemas são modelados em grafos. Segundo [1] Cormen, os grafos são estruturas de dados quintessenciais na ciência da computação assim como os algoritmos que os utilizam. Esse trabalho irá focar em um método chamado Ford-Fulkerson e no algoritmo que o implementa, Edmond-Karp .

Esses Algoritmos são reconhecidamente adequados em problemas de alocação de tarefas ou na determinação do fluxo máximo em redes ou grafos. Após a explicação do funcionamento do algoritmo, ele será implementado em um problema modulado na forma de grafo bipartido, com seus resultados apresentados e discutidos posteriormente.

II. Modelagem do Problema

Para este projeto, foi nos designado o problema de Fluxo Máximo: uma empresa de prestação de serviço que deseja alocar seus filiais, contratadas a fim de solucionar uma determinada tarefa (vértices do grafo) o mais rápido possível e com mais eficiência, sabendo que cada tarefa tem uma carga(aresta do grafo) para ser resolvidas e os funcionários (vértices do grafo) tem uma capacidade adquirida ao longo do tempo para resolver determinadas tarefas. A questão colocada em pauta dessa empresa é saber qual é a melhor forma de repartir as tarefas aos seus funcionários, afim de resolver o mesmo de forma otimizada. Razão pela qual o problema foi modelado em gafos bipartido, pois não há necessidade de um relacionamento entre os funcionários, mas apenas, os funcionários poderão efetuar uma ou mais tarefas. Este grafo bipartido pôde então ser convertido em um fluxo de redes, adicionando um nodo fonte source e um de destino sink, para podermos usar o algoritmo Edmond-Karp e encontrar o fluxo máximo de tarefas que poderão ser realizadas pelos funcionário a qualquer momento. A fig 1 ilustra o grafo bipartido com a "A-origem" e o "B-destino" como demostrado.

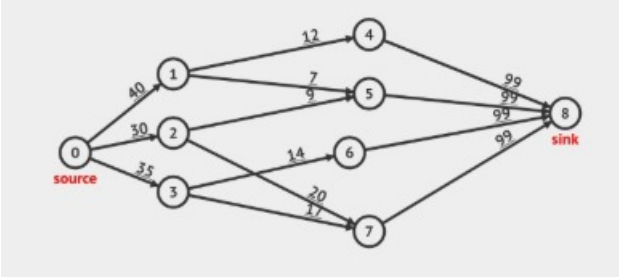

Fig. 1. Example of a figure caption.

A. Grafo bipartido

A seguir apresentamos os conceitos da teoria de grafos bipartidos que serão utilizados no decorrer do trabalho desenvolvido.

Definição: Um grafo ***G*** é dito bipartido quando suas vértices ***V(G)*** podem ser divididos em dois conjuntos disjuntos ***X*** e ***Y***, tais que toda aresta de ***G*** liga um vértice na parte ***X*** e outro na parte ***Y***. Frequentemente, utilizaremos a notação ***G = (X, Y, E)*** para nos referir ao grafo bipartido com partição ***X*** e ***Y*** e ***E(G) = E***. [13] [9]

Teorema: Um grafo bipartido é geralmente caracterizado de duas formas: [13]

- Um grafo ***G*** é bipartido se e somente se todo ciclo de G possuir comprimento par. [8]
- Um grafo é bipartido se, e somente se, seu número cromático é menor que ou igual a 22. Isso significa que podemos colorir todos os vértices usando apenas duas cores de maneira que uma aresta nunca ligue dois vértices da mesma cor. [13]

As pontes de konigsberg, este problema e bastante conhecido na área de computação, já que foi resolvido com a de teoria dos grafos. O problema consiste em encontrar o caminho que atravesse todas as pontes uma vez só e que retorne ao ponto de partida, considerando que há duas ilhas e 7 pontes no rio. [8]

Prova: abaixo iremos demostra como será aplicado o grafo bipartido para a resolução do problema da pontes de konigsberg.

Ida: Seja ***X*** e ***Y*** as duas partições de G. Todo caminho em G alterna um vértice de ***X*** com um vértice de ***Y***. Isso é a consequência da definição de grafo bipartido. Supondo que um ciclo contém um vértice v_i em uma das duas partições. Para voltar a esse vértice, é preciso ir na outra partição e voltar um número par de vezes. [8]

Volta: Seja ***G*** um grafo onde todo ciclo é de comprimento par. Seja um vértice v_i de ***G***. Colocamos num conjunto ***X*** o vértice v_i e todos os outros que são a uma distância par de v_i . Os outros vértices formam o conjunto ***Y***. Se não tivesse nenhuma aresta ligando dois vértices de ***X*** ou dois vértices de ***Y***, respeitaríamos as condições para que o grafo seja bipartido. Suponhamos agora que existe uma outra aresta entre dois vértices ***a*** e ***b*** de ***X*** (ou ***Y***). Já temos um caminho par entre ***a*** e ***b***. Acrescentando a nova aresta, obteríamos um ciclo de comprimento ímpar, o que contradiz a hipótese. Portanto, não pode existir outra aresta entre qualquer par de vértice que já está em ***X*** (igualmente par ***Y***) e o grafo é bipartido. [8] Note que essa prova indica de maneira direta qual seria o algoritmo par determinar se um grafo é bipartido ou não. [8]

A noção de grafo completo pode ser estendido aos grafos bipartidos. Um grafo bipartido completo é um grafo onde todos os vértices da partição ***X*** é ligado por umaaresta a todos os vértices da partição ***Y***. Seja ***m*** e ***n*** o número de vértices em ***X*** e ***Y***,respectivamente, o grafo completo bipartido será denotado $K_{m,n}$. [8]

B. Fluxo máximo

A seguir mostraremos alguns conceitos de fluxo máximo que serão utilizados na aplicação do trabalho desenvolvido na disciplina de análise de algoritmo.

Definição: O fluxo máximo consiste em mostrar o número máximo de unidades de fluxo que é possível enviar através da rede desde o nó origem (ou vértice origem ***S***) até o nó destino (ou vértice destino ***T***) sem violar quaisquer restrições de capacidade. [11] [13]

Teorema: primeiro caso, Seja ***X*** o conjunto de nós que pode ser atingido a partir de ***S*** através de um caminho não cheio, e Seja ***Y*** o conjunto dos restantes nós. $T \in Y$ porque senão $T \in X$ e não se encontra numa situação de fluxo máximo. [11]

Segundo caso, considere-se o corte composto pelos ramos com uma extremidade em ***X*** e outra em ***Y***. Todos os arcos dirigidos de um nó ***V*** em ***X*** para um nó ***W*** em ***Y*** estão cheios, pois caso contrário ***W*** pertenceria a ***X*** e não a ***Y***. Qualquer arco dirigido de ***Y*** para ***X*** terá fluxo nulo pois caso contrário isso seria um retorno de fluxo de ***Y*** para ***X*** que, se fosse anulado, aumentaria o fluxo que efetivamente chega a ***T***, o que contraria a hipótese inicial de a rede estar na situação de fluxo máximo. Então, a capacidade do corte, que é igual ao fluxo de ***X*** para ***Y***, é igual ao fluxo na rede, que é máximo. [11]

Exemplo de aplicação: As aplicações mais óbvias incluem o fluxo de um líquido através de uma rede de tubos, o fluxo de mercadorias do produtor ao consumidor, o casamento de desempregados com empregos, uma lista (não exaustiva) de aplicações: [8]

- Grafo com múltiplas origens e/ou múltiplos destinos. [12]
- Encontrar o corte mínimo que separa o grafo em duas metades. [12]
- Encontrar número de caminhos que não usem as mesmas arestas. [12]
- Encontrar o maior emparelhamento num grafo bipartido. [12]

Prova: Seja ***f*** um fluxo que respeita as capacidades. Seja ***C*** um corte e ***S*** a margem superior do corte. Pela propriedade dos saldos, a intensidade de ***f*** é igual ao saldo de ***f*** em ***S***. Esse saldo é ***out(S)–in(S)*** e portanto não passa de ***out(S)***. Como ***f*** respeita as capacidades, ***out(S)*** não passa da capacidade do corte ***C***. [10]

III. Os Algoritmos

A. Ford-Fulkerson

O algoritmo de Ford-Fulkerson também conhecido como ”algoritmo de caminhos aumentantes”, funciona da seguinte maneira, enquanto existir caminho aumentante de origem até destino, envia fluxo por esse caminho, sendo nesse caso utilizado para encontrar o fluxo de valor máximo que faça o melhor uso possível da capacidade disponível do mesmo. [15]

B. Edmond-Karp

O algoritmo de Edmonds-Karp é apenas uma implementação do método Ford-Fulkerson que usa busca em largura para encontrar caminhos aumentados. [15]

IV. Implementação

O main.c é iniciado com a leitura da linha dada como entrada 04 números inteiros: o número de vértices (interseções), o número de arestas (carga), o número de tarefas (origens) e o número de funcionários (destinos). Após esta leitura é criado um Grafo com dois nodo extras, para que seja feito a implementar a ideia da A-origem e B-destino, source e sink respectivamente. Após isso, é dado um número de linhas que corresponde ao número de arestas, as quais dão 03 inteiros de entrada: um vértice origem, um vértice destino, e o peso (fluxo máximo) daquela aresta. Também é criado a variável resposta, que recebe o valor de retorno da função ”Ford-Fulkerson” e seus caminhos. [16]

No módulo ”FordFulkerson”, recebe como parâmetros de entrada o número de vértices e o grafo. Em sua execução, é criado um grafo de fluxo residual. Em seguida, o vetor caminha 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 A-origem ao B-destino. Dentro do while, e criado 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. [16]

No módulo "buscaLargura", recebe como parâmetros de entrada o número de vértices, o vetor caminho, o grafo e o grafo de fluxo (grafo residual). Nesta função, determino A-origem como o número de vértices+1 e B-destino como o número de vertice+1. Crio uma fila dinamicamente, usando o módulo " fila ", e crio um vetor de visitados (de tamanho vertice+2). Este vetor é inicializado como 0, que significa que aquele vértice não foi visitado ainda. Para inicializar, é enfileirado a A-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 desenfileirando 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. [16]

V. 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:

- Edmond-Karp: A complexidade de tempo é O(fluxo-máximo*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-máximo*numero de arestas). Neste caso, como usamos também o Busca em Largura, temos que será O(vértices * numero de arestas 2 * fluxo-maximo).
- buscaLargura : A complexidade do tempo do algoritmo da Busca em Largura pode ser expressa como O(número 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$_{m}$$\acute{a}$$ximo$).

VI. Resultados

A. Avaliação Experimental

As tabelas representam as entradas consideradas como exemplo na implementação do trabalho juntamente com as imagens resultantes, na qual foi utilizado o programa do https://visualgo.net.

Funcionário

Numero	Tarefas	Capacidades
1	1	12
2	*1*	*7*
	2	*9*
3	*3*	*14*
4	*2*	*20*
	3	17

Tarefas

Numero	Capacidades
1	15
2	*20*
3	*9*

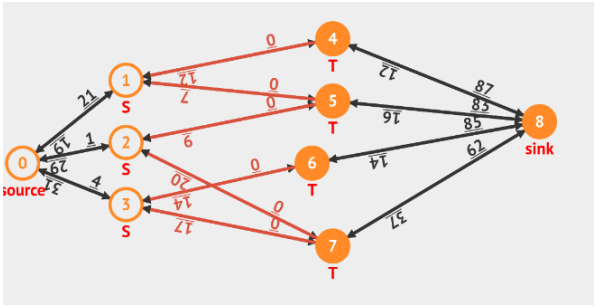

Fig. 2. Implementação com Edmond-Karp

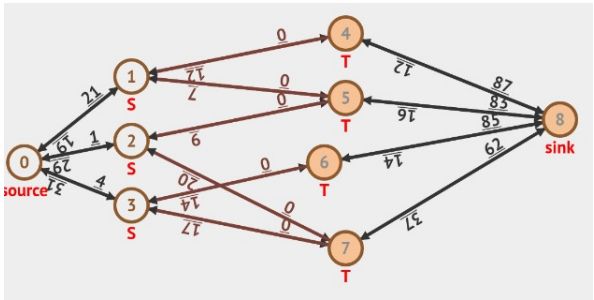

Fig. 3. Implementação com Ford-Fulkerson

B. Conclusão

Nesta abordagem foi entendido como alocação de tarefas para funcionários dividir o peso arbitário de um tarefa no maior número de funcionários que poderiam resolvê-la. Funciona como exemplo da utilização de algoritmos de

flúxo máximo, mas não é facilmente entendida como uma representação acurada de uma situação real. Um problema de Matching em grafos bipartidos talvez fosse mais pertinente à situação descrita. Dito isso, a implementação aqui proposta consegue mostrar as restrições impostas pelo funcionários, o corte mínimo, caminhos aumentantes e o grafo residual, ideias que formam a base do método Ford-Fulkerson e aqui mostradas com sucesso.

REFERENCES

[1] Thomas H. CORMEN; Charles E. LEISERSON; Ronald L. RIVEST, "Algoritmos: teoria e prática". Rio de Janeiro: Campus, 2002. 916 p.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] Borba, M. P. (s.d.). Teoria dos grafos. Fonte: https://miltonborba.org/Algf/Grafos.htm

[9] Cruz, C. B. (Outubro de 2009). Caracterizações e reconhecimento de grafos bipartidos . Fonte: https://www.cos.ufrj.br/uploadfile/1258550856.pdf

[10] Feofiloff, P. (14 de Julho de 2017). Algoritmos para Grafos. Fonte: Instituto de Matemática e Estatística da Universidade de São Paulo: https://www.ime.usp.br

[11] Oliveira, J. F. (s.d.). Problemas de Fluxo Máximo. Fonte: Faculdade de Engenharia da Universidade do Porto: https://web.fe.up.pt

[12] Ribeiro, P. (2015). Fluxo M´aximo. Fonte: http://www.dcc.fc.up.pt

[13] Siaudzionis, L. (s.d.). Grafos Bipartidos. Fonte: CodCad: http://www.codcad.com/lesson/65

[14] Valeriano A. de Oliveira, S. R. (s.d.). Teoria dos Grafos. Fonte: UNESP: https://www.ibilce.unesp.br

[15] Maximum flow - Ford-Fulkerson and Edmonds-Karp. (s.d.). Fonte: CP-Algorithms: https://cp-algorithms.com/graph

[16] Bezerra, L. (11 de Agosto de 2017). Trabalho Prático da matéria Algoritmo e Estrutura de Dados III (Universidade Federal de Minas Gerais). Fonte: https://github.com/luisarbezerra