

# Trabalho Prático 1: Entregando Lanches

Marcos Paulo Quintao Fernandes

Maio 2017

## 1 Introdução

Rada possui uma empresa de entrega de lanches. Gilmar, seu amigo , é responsável por coordenar a entrega de lanches em diversos pontos da cidade. A empresa funciona da seguinte maneira: para cada franquia da firma, temos ciclistas saindo para fazer entregas. Cada ciclista pode trafegar por qualquer sequência de caminhos ligados por ciclofaixas desde que ele sempre chegue em qualquer um dos clientes de Rada localizados na cidade. No entanto, devido à reformas da prefeitura, as ciclofaixas, agora possuem apenas 1 sentido. A cidade então foi mapeada em intersecções que se ligam por ciclofaixas, sendo que cada ciclofaixa comporta um numero  $w$  máximo de ciclistas trafegando por hora por questões de segurança. Tendo em mãos em quais intersecções as franquias da firma se encontram, em quais intersecções os clientes se encontram e o mapa da cidade, devemos encontrar qual o número máximo de ciclistas que podem sair das franquias de Rada de modo que a segurança do ciclista seja levada em conta.

## 2 Solução do Problema

Primeiramente , podemos modelar esse problema como um grafo. Seja  $G$  o grafo que representa o mapa da cidade. Os vértices de  $G$  representam cada intersecção da cidade e as arestas representam cada ciclovia. O peso de cada aresta simboliza o número máximo de ciclistas que podem trafegar na ciclovia em questão por hora.

Feita a modelagem, iremos simplificar um pouco o problema para vislumbrarmos uma solução para tal: Imagine que temos apenas uma franquia e um cliente. Suponha que  $f$  é um fluxo no grafo com peso  $G$  . Dizemos que  $f$  respeita as capacidade das arestas do grafo  $G$  se  $\text{fluxo}_{\text{aresta}} \leq w$  para cada aresta, sendo que  $w$  é o peso de cada aresta. Feito isso, nesse problema, queremos maximizar a quantidade de fluxo que sai da franquia e chega no cliente de forma a respeitar a capacidade das arestas. Ou seja, queremos o fluxo máximo da fonte (intersecção da franquia) e sumidouro (intersecção cliente). Resolvido esse problema, percebemos que quando inserimos muitas franquias e muitos clientes, temos o problema de fluxo máximo de várias fontes e vários sumidouros. Dessa forma,

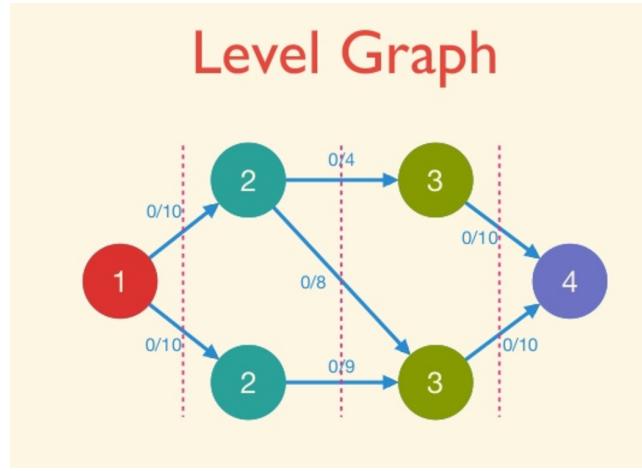
para resolvê-lo, precisamos de alguma forma transformar esse grafo em um outro com apenas 1 fonte e apenas 1 sumidouro. Seguindo esse raciocínio, iremos adicionar 2 vértices artificiais no grafo :  $s$  e  $t$ .  $s$  será a fonte do novo grafo e  $t$  será o sumidouro do novo grafo. Iremos criar 1 aresta de  $s - franquia_i$  para todas as franquias (de custo infinito ) e 1 aresta de  $cliente_i - t$  para todos os clientes ( de custo infinito). Podemos perceber, que o fluxo máximo de  $s$  para  $t$  é equivalente ao fluxo máximo de todas as franquias para todos os clientes. Para resolver o problema de fluxo máximo de 1 fonte para 1 sumidouro , iremos , por questões de eficiência, utilizar o algoritmo Dinic, que será explicado mais abaixo.

### 3 Dinic

Seja o grafo  $G$  um grafo ponderado e direcionado. O Dinic resolve o fluxo máximo que sai de uma fonte  $s$  para um sumidouro  $t$ . Basicamente, similarmente ao algoritmo mais clássico de fluxo: Ford Fulkerson, o Dinic encontra o fluxo máximo baseado em caminhos de aumento do grafo. Um caminho de aumento é um caminho que começa em  $s$  e termina em  $t$  de tal forma que ele aumenta o fluxo resultante. Para calcular o caminho de aumento do grafo, iremos separar o algoritmo em 2 etapas:

1. Construir o grafo de níveis
2. Achar os bloqueios de fluxo

Para construir o grafo de níveis, iremos fazer uma BFS a partir de  $s$  de forma a encontrar todos os vértices que distam 1 da fonte, que distam 2 e assim por diante. Feito isso, temos o grafo de níveis, conforme a figura 1 abaixo:

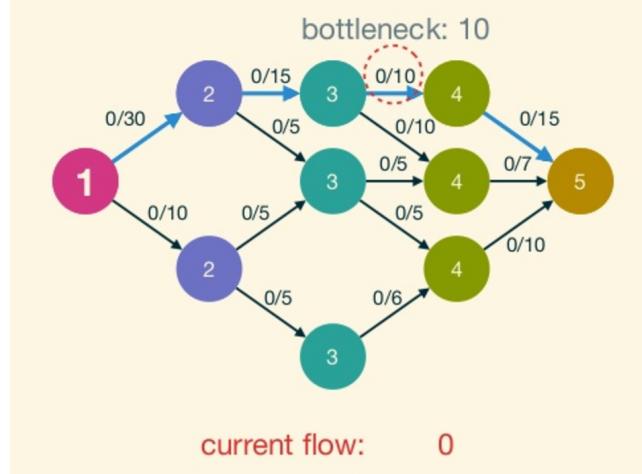


1 - Grafo de Níveis

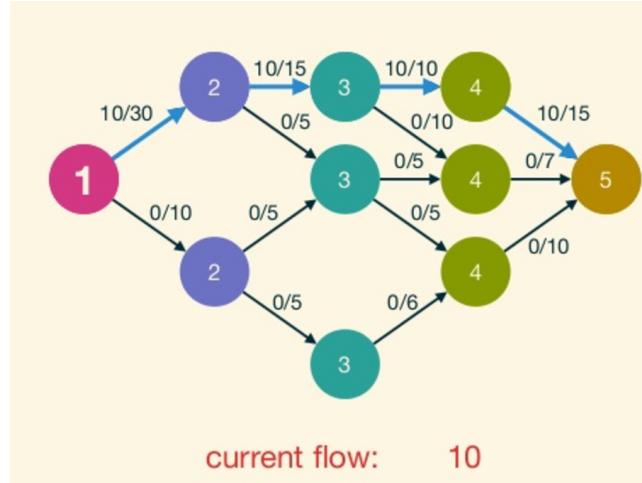
Feito o nivelamento, iremos achar um caminho de aumento de fluxo saindo de  $s$  e chegando em  $t$ . Um caminho de aumento de fluxo é um caminho em que

caminhamos somente por arestas que conseguem transportar alguma unidade de fluxo. Achado o caminho, encontramos o seu "gargalo": a aresta que menos consegue transportar alguma unidade de fluxo e que tem o par de vértices de maior nível possível. Dessa forma, a partir do vértice em que sai a aresta do "gargalo", tentaremos encontrar um outro caminho de aumento de fluxo. Iremos executar esses 2 passos até que não conseguimos mais um caminho de aumento, conforme o esquema abaixo:

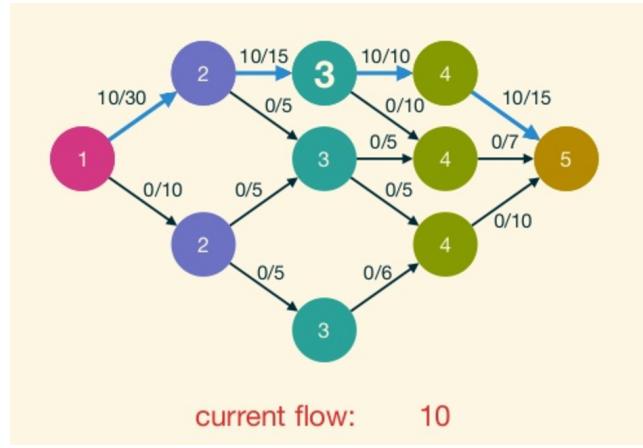
1- Encontre o caminho da fonte pro sumidouro e Encontre o gargalo



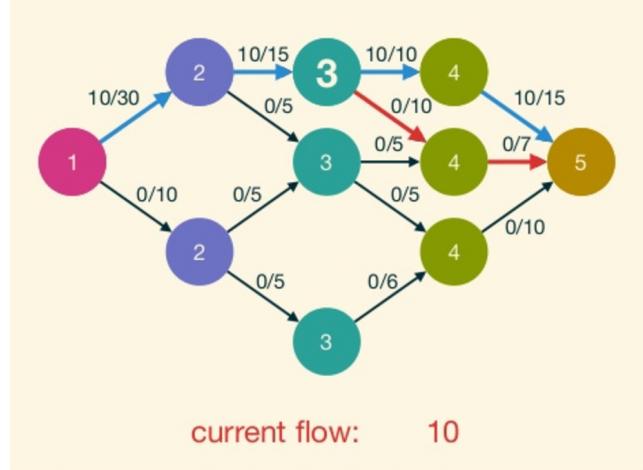
2- Atualize as capacidades da aresta e fluxo



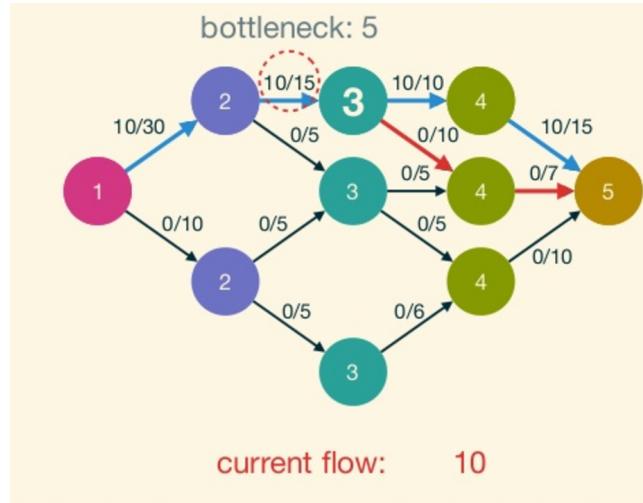
3- O vértice no nível 3 é o gargalo do caminho



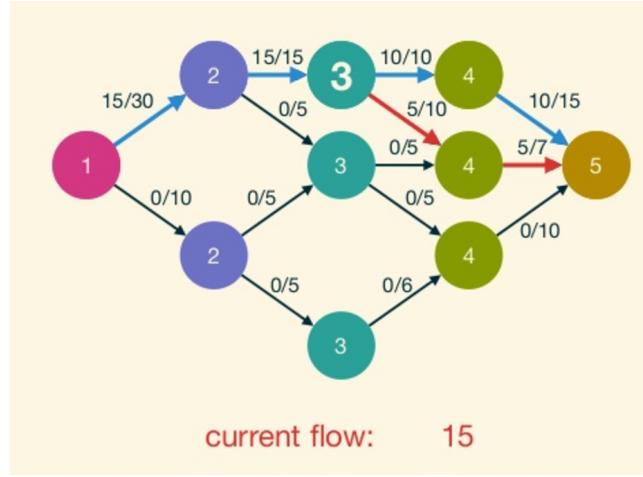
4- Encontre um novo caminho de aumento de fluxo



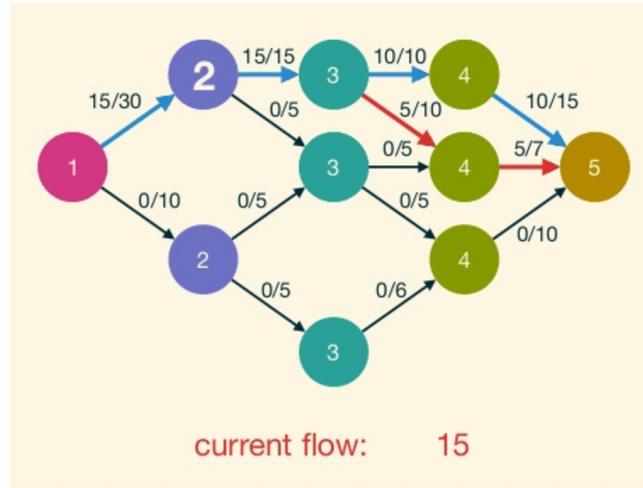
5- Encontre o novo gargalo



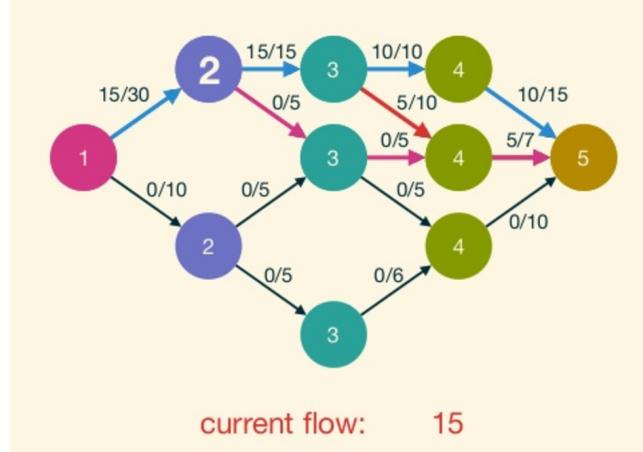
6- Atualize as capacidades das arestas e o fluxo



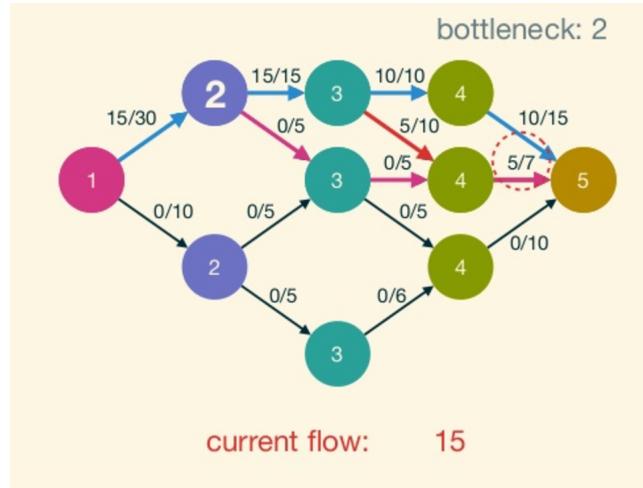
7- O vértice do nível 2 é o novo gargalo do caminho



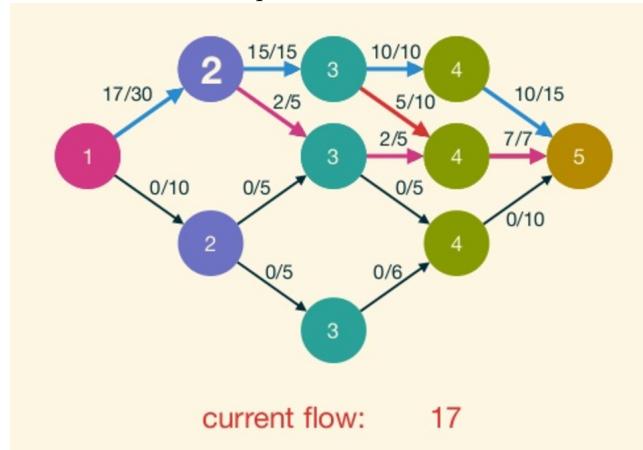
8- Encontre um novo caminho de aumento de fluxo



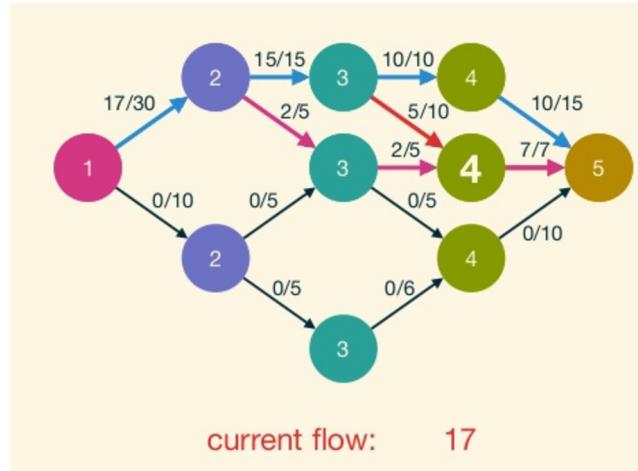
9- Encontre o novo gargalo



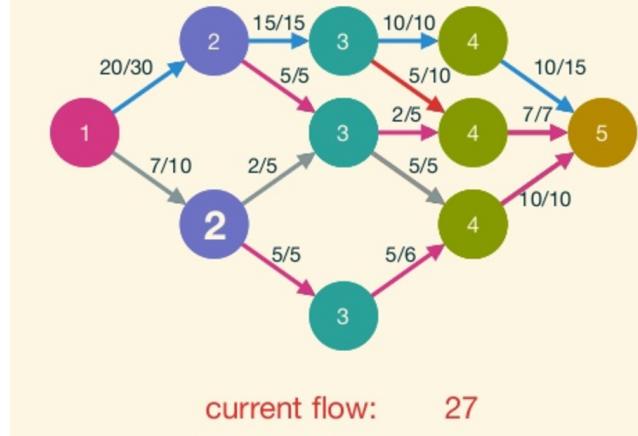
10- Atualize as capacidades das arestas e o fluxo



11- O vértice do nível 4 é o novo gargalo do caminho



12- Após repetirmos o procedimento várias vezes temos:



obs: As imagens retiradas de [www.slideshare.net/KuoE0/acmicpc-dinics-algorithm](http://www.slideshare.net/KuoE0/acmicpc-dinics-algorithm)  
Dessa forma podemos simplificar o algoritmo nos seguintes passos:

1. Construir o grafo de níveis
2. Encontrar um caminho de aumento da fonte para o sumidouro
3. Encontrar o gargalo no caminho de aumento
4. Encontrar um novo caminho de aumento, partindo do gargalo
5. Repetir os itens 3 e 4 até não encontrar um caminho de aumento
6. Repetir os itens 1,2,3,4,5 enquanto existir algum caminho de aumento da fonte para o sumidouro.

## 4 Análise de Complexidade

Complexidade do Dinic: O custo computacional de construir o grafo de níveis é  $O(V + E)$ , que é fazer uma BFS partindo da fonte. O custo de encontrar o gargalo em um caminho de aumento é  $O(E)$ . Como cada vértice pode ser o gargalo apenas 1 vez, temos que a complexidade de encontrar todos os gargalos é:  $O(VE)$ . Como podemos construir no máximo  $V$  grafos de níveis, temos que a complexidade do dinic é:  $O(V^2E)$ . Sendo que  $V$  é o número de vértices do grafo e  $E$  o número de arestas.

Complexidade da Solução do problema de Gilmar: A complexidade da solução é a complexidade de construir o grafo <sup>1</sup> + complexidade de adicionar os vértices artificiais que representam a fonte e o sumidouro <sup>2</sup>+ complexidade de rodar o dinic no grafo resultante <sup>3</sup>.  $1 - O(V+E) 2 - O(V) 3 - O(V^2E) 1+2+3 = O(V^2E)$   
Portanto a complexidade da solução é:  $O(V^2E)$ .

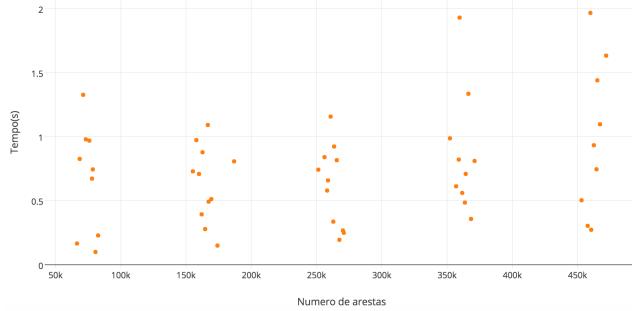
## 5 Avaliação Experimental

Para avaliarmos o desempenho desse programa utilizaremos uma máquina com as seguintes configurações:

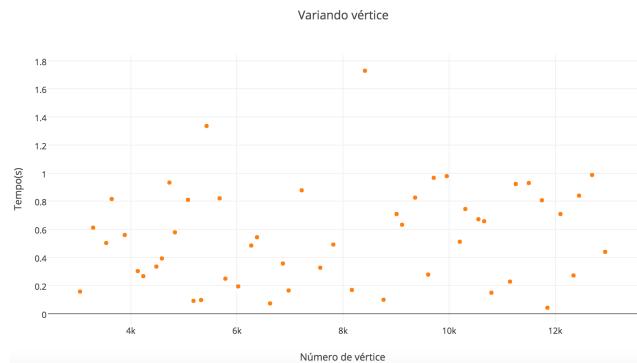


Configurações da Máquina

Para testarmos a eficiência do algoritmo , submetemos o programa à vários testes aleatórios. Foram gerados grafos com número de vértices variados e número de arestas variados. Primeiramente, fixado o número de vértices  $N=1000$  e  $f=10$  e  $c=15$ , iremos gerar um grafo com  $M$  número de arestas variados para observarmos o desempenho do programa com relação ao número de arestas:



Agora, fixado o número de arestas  $M=100000$  e  $f=10$  e  $c=15$ , iremos gerar um grafo com  $N$  número de vértices variados para observarmos o desempenho do programa com relação ao número de vértices:



Para grafos aleatórios, conseguimos evidenciar que o dinic é bem mais rápido que  $O(EV^2)$ . Dessa forma, não conseguimos evidenciar sua complexidade assintótica , uma vez que o tempo de execução dominante é de basicamente leitura e escrita de dados.

## 6 Referencias Bibliográficas

1. [www.slideshare.net/KuoE0/acmicpc-dinics-algorithm](http://www.slideshare.net/KuoE0/acmicpc-dinics-algorithm)
2. [en.wikipedia.org/wiki/Dinic%27s\\_algorithm](http://en.wikipedia.org/wiki/Dinic%27s_algorithm)
3. CORMEN, T.H. et.al. Algoritmos: teoria e prática, 3º edição. Ed. Campus, 2002.  
Sitedoautor:<http://mitpress.mit.edu/algorithms>