Trabalho Prático 1 – Entregando lanches

Algorítmos e Estrutura de Dados III
Tatiana Santos Camelo de Araujo – 2015086298

1. Introdução

O objetivo deste trabalho é treinar os conhecimentos prévios de Algorítmos e Estrutura de Dados II e implementar novos conceitos de Algoritmos e Estrutura de Dados III. O trabalho é feito em alocação dinâmica, usa conceitos de fila, matriz, grafos, além de precisar de algoritmos já conhecidos na computação, modificados para se adequar ao problema proposto.

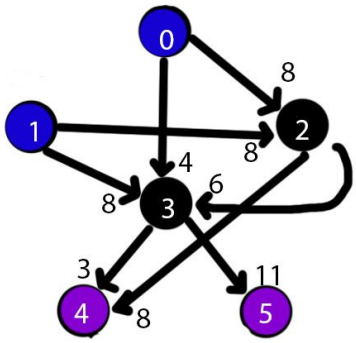
2. Modelagem

O trabalho aqui documentado se trata de um problema comum em grafos, chamado max flow. Recebemos um grafo ponderado e devemos achar todos os caminhos válidos de maior valor de todos estes caminhos.

Consideremos um serviço de entrega de lanches, que tem franquias (vértices, tipo 0), clientes (vértices, tipo 1), pontos de passagem sem ser franquia ou cliente (vértices, tipo 2), caminhos com volume máximo de circulação (peso da aresta). Devemos sair de franquias com os lanches a serem entregues e levá-los aos clientes. Devemos, também, maximizar o número de entregadores (ciclistas) que podem passar por cada caminho de destino à origem.

A explicação da solução do problema vai ser baseada na entrada:
6 8 2 2 (número de vértices, número de arestas, número de clientes, número de franquias)
0 2 8 (vértice 0 que se liga a vértice 2, com peso 8)
0 3 4 (vértice 0 que se liga a vértice 3, com peso 4)
1 2 8 (vértice 1 que se liga a vértice 2, com peso 8)
1 3 8 (vértice 1 que se liga a vértice 3, com peso 8)
2 3 6 (vértice 2 que se liga a vértice 3, com peso 6)
2 4 8
3 4 3
3 5 11
0 (vértice que é franquia)
1 (vértice que é franquia)
4 (vértice que é cliente)
5 (vértice que é cliente)

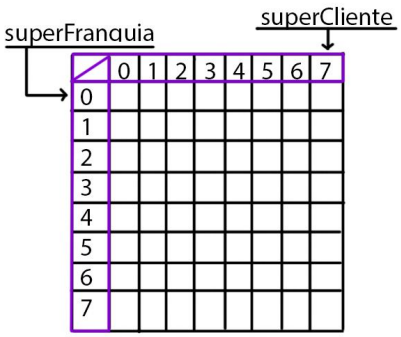
Neste caso a saída será:
22



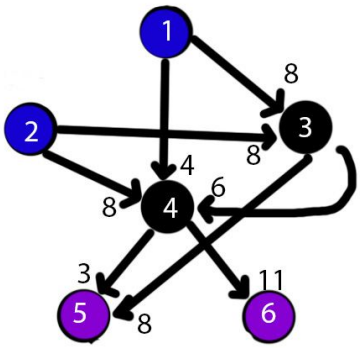
Está é a representação do grafo acima. Em azul estão os vértices de franquias e vértices roxos são clientes. Na base das setas (para onde ela aponta) estão os valores de peso de cada uma.

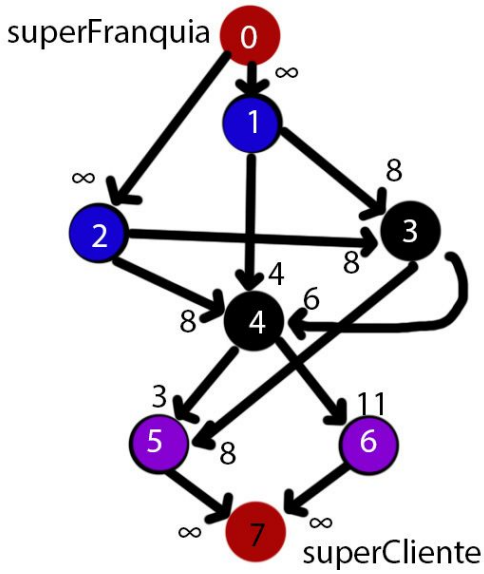
A solução deste problema foi baseada em dois conceitos: é preciso achar todos caminhos possíveis que ligam todas franquias à todos clientes; achar o menor valor de arestas neste caminho e retirar esse valor de todas as arestas. As soluções em implementação desdes dois problemas são algorítmos conhecidos, BFS (Breadth First Search) e Ford-Fulkerson.

Um problema encontrado nesta implementação é o controle da busca do caminho, já que eu posso ter mais de uma franquia e mais de um cliente, como no exemplo acima. A solução implementada foi criar um vértice 0 que sempre será superFranquia, que envia infinitos ciclistas para todas franquias e superCliente (que será vértice de número [total de vértices+2]) que recebe infinitos ciclistas dos clientes. Para isso, ao receber os vértices na entrada, eles já são salvos na matriz com o incremento de 1, de forma que não conflitam com as informações das linhas e colunas de 0 e vértices+2. Isto será explicado em detalhes posteriormente.

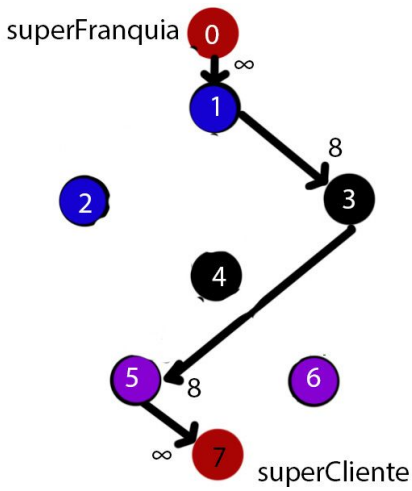


Incrementando o valor dos vértices para termos espaço para superFranquia e superCliente, será assim:

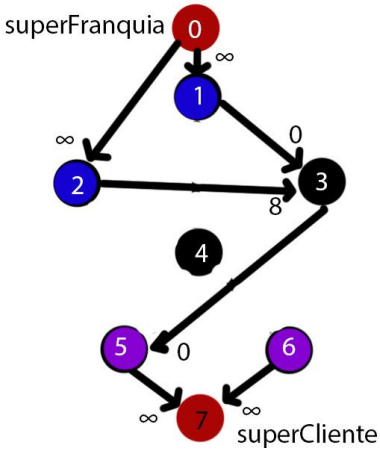




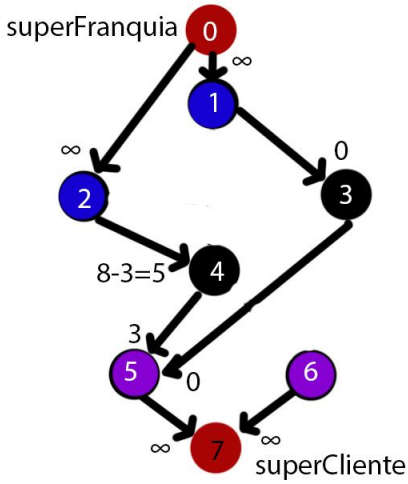
Assim, eu sempre vou sair da superFranquia e sempre vou chegar no superCliente. A resolução e implementação das arestas de tamanho infinito é o uso da variável INT_MAX, que determina o maior inteiro possível que pode ser recebido no peso de um caminho.



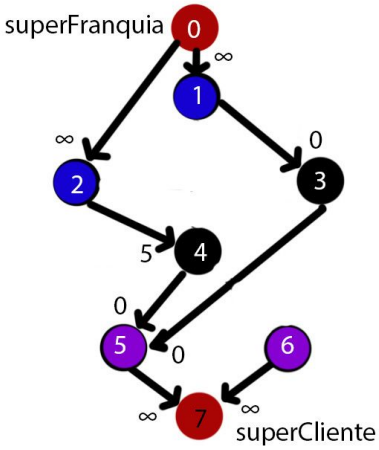
Este exemplo acima mostra um caminho em 0-1-3-5-7. Este caminho consegue enviar 8 ciclistas, mas não poderá enviar nenhum ciclista novamente pelas arestas 1-3 e 3-5, pois atingiu o máximo possível de ambas. Mesmo se o vértice 3 receber mais ciclistas, ele não poderá enviar pelo caminho 3-5. Ficará assim:



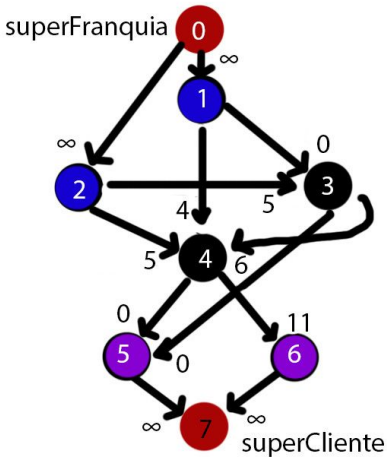
Porém, há outras possibilidades de caminho considerando o fluxo atual, por exemplo, 0-2-4-5-7 será assim:



O menor peso deste caminho é 3, pela aresta 4-5. Esta aresta será zerada então, pois já enviou o máximo possível. A aresta 2-4, por outro lado, podia enviar 8 ciclistas, mas só vai enviar 3 (afinal, não adianta mandar mais ciclistas para vértice 4 se só 3 vão poder sair de lá). Retirando do total estes 3 ciclistas da aresta 4-5, poderão ainda andar por lá 5 ciclistas, resultando em:



Este grafo está sem algumas arestas apenas para facilitar a compreensão. Após andar por estes dois caminhos exemplificados, o grafo estará assim:



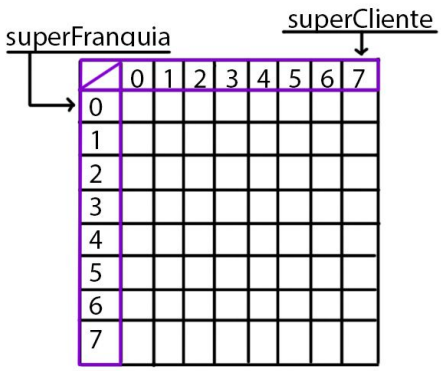
Note que o vértice 4 ainda pode receber 5+4+5 = 14 ciclistas, pelos caminhos 1-4, 2-4, e 3-4 (repare que este caminho pode enviar 6 ciclistas, mas só 5 ciclistas podem chegar no vértice 3 agora). Porém o vértice 4 só pode enviar 11 ciclistas, pelo caminho 4-6.

O resultados destes caminhos são 11+8+3 = 22, que é a saída esperada.

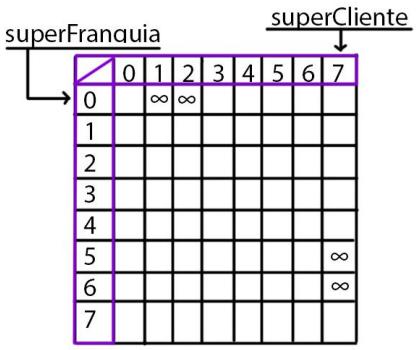
3. Solução do problema

A implementação dos conceitos de superOrigem e superDestino foi a escolha neste algoritmo devido à implementação do BFS, que precisa de um vértice inicial e um final para rodar. Outra implementação poderia ser loop que testa todos casos para cada origem e cada destino, mas a primeira foi considerada mais eficiente e menos custosa.

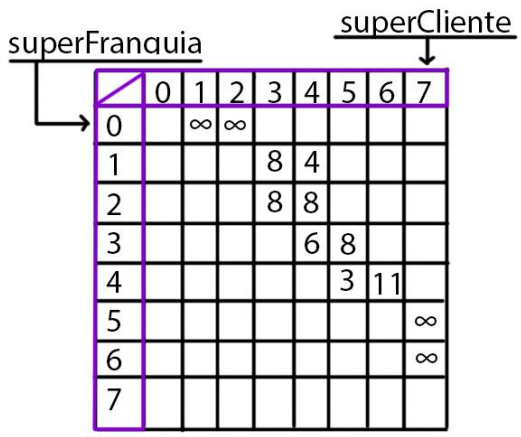
A implementação deste grafo foi feita em matriz, alocada dinamicamente com colunas e linhas de 0 à vértices totais + 2.

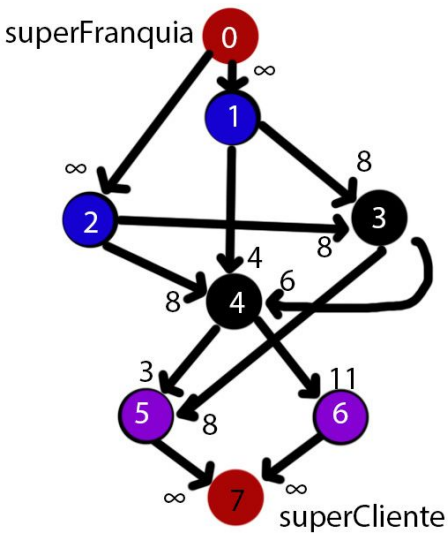


A primeira coluna, roxa, representa de qual vértice vai e a linha roxa representa qual vértice recebe os ciclistas. A linha 0, então, será todos ciclistas enviados para franquias pela superFranquia. A coluna 7 (no exemplo, 7 é o meu máximo de vértices) é o superCliente e representa todos ciclistas que podem chegar de clientes.



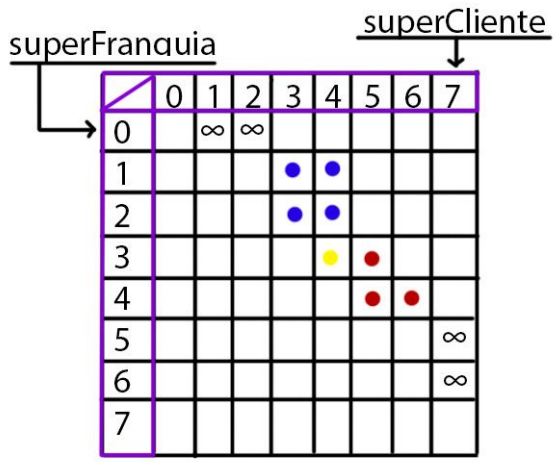
Esta é a representação com superFranquia enviando infinitos ciclistas para as franquias do exemplo, vértices 1 e 2 e superCliente recebendo infinitos ciclistas dos clientes representados pelos vértices 5 e 6.





Esta é a representação em matriz do grafo de exemplo, ao lado.

Para a devida manipulação de qual tipo são os vértices, a matriz foi feita de structs tipoVertice, que guarda a informação de qual tipo o vértice é e o peso das arestas.



Os pontos azuis representam franquias., que na implementação do struct têm valor padronizado como tipo = 0. Em vermelho estão os clientes, de valor tipo = 1. Os vértices que existem mas não são nem franquia nem cliente estão em amarelo (no exemplo, apenas o vértice 4 segue este padrão) e têm tipo = 2.

4. Análise de complexidade

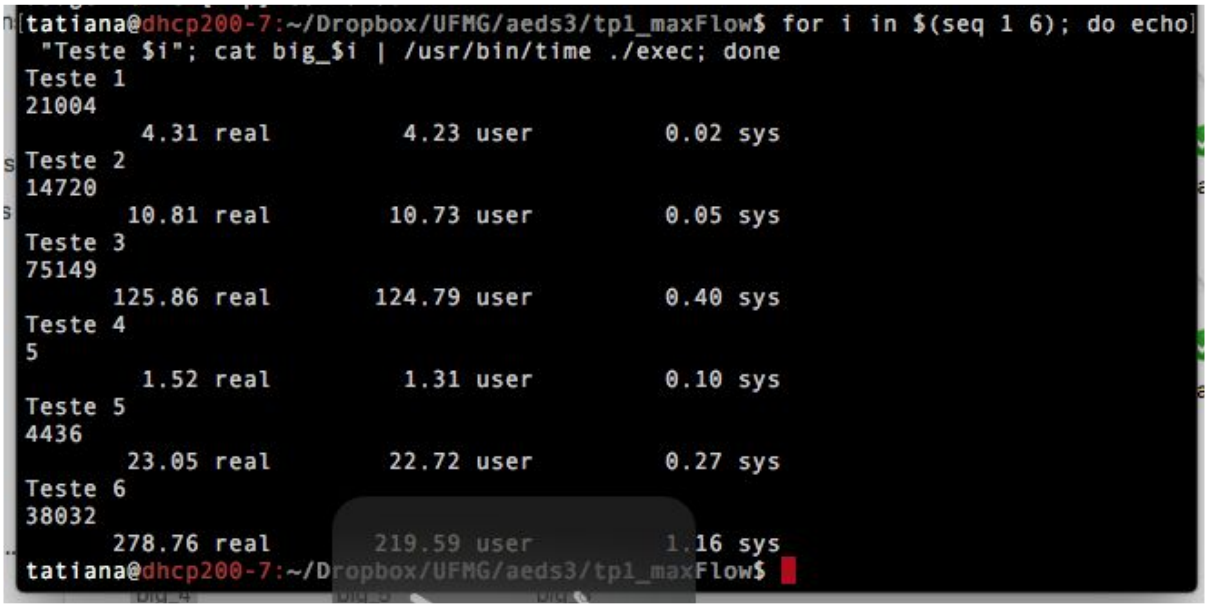
Funções principais:
Nas complexidades, A representa arestas de entrada e V os vẽrtices de entrada.

Função	**Funcionamento**	**O(n) de tempo**	**O(n) de espaço**
main	Função principal, com loops simples e alocação dinâmica de dois grafos. Chama a funcao **caminho** recursivamente incrementando o valor de máximo.	$O(VA^2)+O(V+A)$	$2O(V^2)$
bfs	Complexidade e dado em função da entrada de vértices Função que procura todos caminhos possíveis no grafo. Tem uma fila de controle e possível volta no caminho. O vetor visitado marca se ja passou no vértice, evitando loop. Vetor parent salva qual nodo e pai do nodo de número igual a posicao do pai.	O(V+A)	O(2*V)

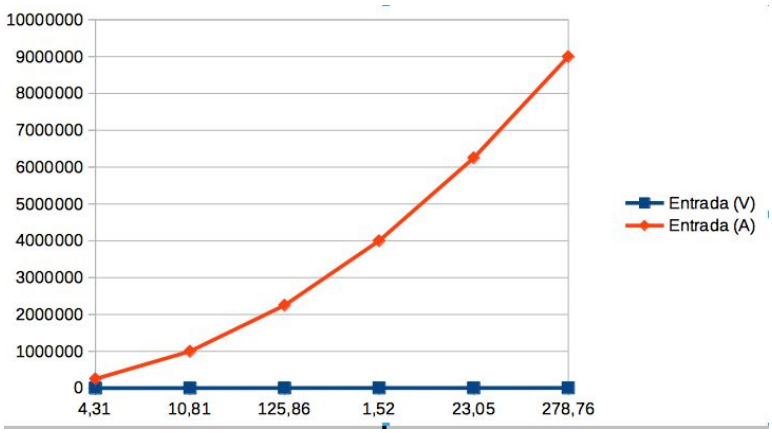
	Retorna 1 se houver caminho possível e 0 se não houver.		
caminho	Funcao que recebe BFS e o caminho achado. Enquanto houver caminho valido, procura menor aresta no caminho e retira seu valor de todas arestas de 0 a total de vertices e adiciona no vertice oposto, no grafo residual. Assim ha controle de quantos ciclistas ja passaram e podem passar pelo caminho.	$O(VA^2)$	$O(V^2)$

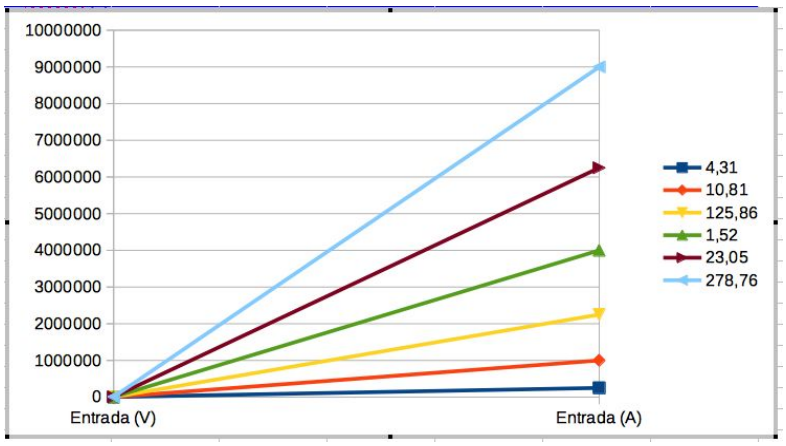
5. Avaliação experimental

A avaliação experimental do trabalho foi feita baseada nos casos *big_test* disponibilizados nos moodle. O programa passou em todos casos toy.

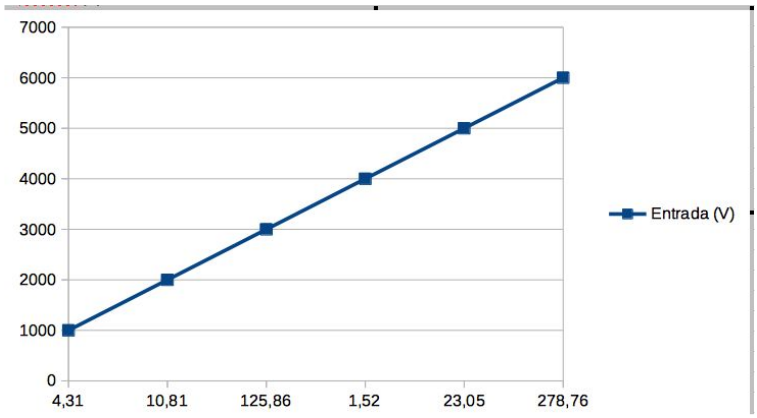


	A	B	C	D	E	F	G
1	Big Teste	1	2	3	4	5	6
2	Tempo (seg)	4,31	10,81	125,86	1,52	23,05	278,76
3	Entrada (V)	1000	2000	3000	4000	5000	6000
4	Entrada (A)	249750	999500	2249250	3999000	6248750	8998500

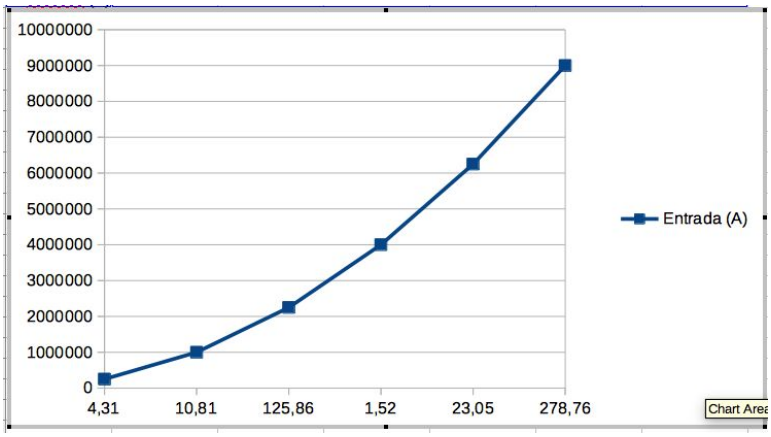




Relacao de tempo e numeros absolutos de entrada de vertice e de arestas.



Este grafico mostra que a complexidade e linear em relacao aos vertices de entrada.



Este grafico mostra que a complexidade exponencial em relacao as arestas de entrada.

6. **Conclusão e comentários**

   A implementação dos algoritmos-base de solução do problema (BFS e Ford Fulkerson) e a debugação deles foi bem delicada. O BFS achava menos caminhos que o correto nos casos que haviam mais de um vértice de cliente (portanto só passava em 5 dos teste toy). A implementação da fila foi fundamental e exatamente o que faltava para passar nos toy e nos big_tests postados no moodle.