ASA 2018/2019 - Relatório Projeto 2 - Grupo al132

Francisco Serralheiro, nº 89445 Rafael Alexandre, nº 89528

Introdução

O presente relatório visa descrever e avaliar a solução desenvolvida para o segundo projeto da disciplina Análise e Síntese de Algoritmos.

O objetivo do programa desenvolvido é desenvolver uma aplicação informática que determina a capacidade máxima que uma rede de transporte de mercadoria de um hipermercado é capaz de transportar. Há 3 tipos de pontos de interesse nesta rede: fornecedores de mercadoria, estações de abastecimento e controlo, e o destino final - o hipermercado. Sabe-se o peso do material produzido pelos fornecedores a transportar e cada meio de transporte tem um limite máximo que consegue movimentar e cada estação de abastecimento tem uma capacidade máxima.

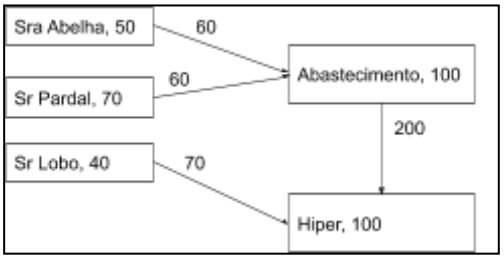
A primeira funcionalidade é o cálculo da capacidade máxima da rede de transporte.

A segunda e terceira funcionalidades calculam as estações de abastecimento da rede de transporte e os caminhos da rede de transporte cuja capacidade máxima deve ser aumentada para aumentar a capacidade máxima da rede.

Assim, este relatório descreve a solução desenvolvida, elabora a sua análise teórica e avalia os resultados experimentais.

Descrição da solução

Para desenvolver a solução deste problema foi implementado um grafo dirigido e pesado que representa o problema. Todos os vértices do grafo representam os pontos de interesse e todas as arestas os caminhos desta rede de transporte. Inicialmente foi implementado um grafo que representa a rede de transporte dada como *input*. Assim, o hipermercado foi colocado como destino (inicialmente), com o número 1, tal como é dado no *input*. O vértice 0 foi um vértice adicional, criado para servir como origem. As capacidades dos fornecedores foram convertidas para arestas com origem no vértice 0 e destino no fornecedor e com peso igual à capacidade máxima do fornecedor. As estações de abastecimento foram convertidas em 2 vértices, ligados por uma aresta dirigida e com peso igual à capacidade máxima da estação de abastecimento. Esta aresta tem como origem o vértice inicial, dado como *input*, e como destino o vértice adicional, cuja aresta aponta para o hipermercado.



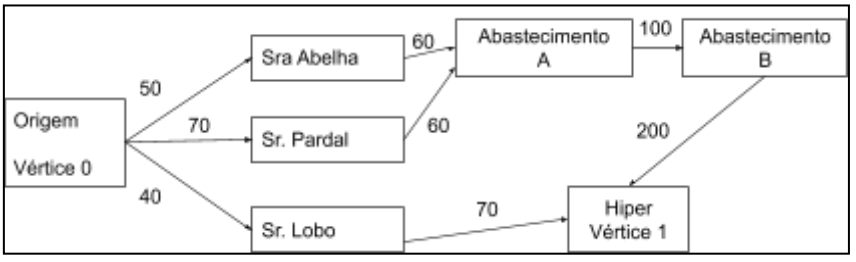


Figura 1. À esquerda, a rede dada como *input* e à direita a primeira representação da mesma rede na solução desenvolvida

Finalmente, sabendo que é necessário calcular as arestas mais próximas do Hiper para aumentar a capacidade, é necessário efetuar o corte mínimo do grafo transposto, para que o corte mínimo obtido seja o mais próximo do Hiper. Assim, o grafo apresentado na figura 2 foi transposto, e esse foi o grafo representado internamente utilizado para solucionar o problema.

No programa desenvolvido, há uma classe principal - a classe *Network* que modela a rede de distribuição de uma cadeia de hipermercados. Esta classe guarda as seguintes propriedades relativas à rede: número de pontos de interesse na rede de distribuição, número de fornecedores, número de estações de abastecimento e número de caminhos entre cada ponto de interesse: os fornecedores, as estações de abastecimento e o hipermercado, que é o destino final. Nesta classe estão também guardados num array de listas de objetos da classe *Edge* e dois vetores de inteiros que guardam o valor do *flow* em excesso e o valor da altura para cada vértice. O índice onde os valores estão guardados corresponde ao número do vértice. Por exemplo, o valor guardado no vector das alturas na posição 3, corresponde ao valor da altura do vértice 3.

Foi criada uma classe *Edge* que representa as arestas da rede de do grafo, ou seja, os caminhos que existem na rede de distribuição do hipermercado. Esta classe guarda os valores do *flow* que está a passar no caminho, a capacidade total do caminho, o valor do vértice de origem e valor do vértice de destino.

Para representar internamente as estações de abastecimento B, ou seja, as estações que cujo número não é dado como *input*, foram utilizados também números que começavam no primeiro número após o último número das estações de abastecimento. Por exemplo, se fosse dado 3 estações de abastecimento, o vértice 6 é o Abastecimento B do vértice 3, e assim sucessivamente.

Para obter todos os *outputs* pedidos foi executado um algoritmo *push-relabel* otimizado com uma *queue* que guarda apenas os vértices com *flow* em excesso. Ao executar o algoritmo, imediatamente é devolvido o fluxo máximo, o primeiro *output*.

Depois, para obter as estações que são necessárias aumentar de capacidade e os caminhos, foram consultadas as alturas dos vértices para averiguar qual o lado do corte mínimo onde o vértice se encontrava.

Para calcular as estações de abastecimento cuja capacidade máxima é necessário aumentar, foi criado um vetor local “critical_stores” de inteiros que guardava as estações de abastecimento a aumentar a capacidade. Sabe-se que as estações a aumentar são as quais o corte mínimo corta a aresta que liga Abastecimento A e B. Deste modo, por cada vértice, se o vértice fosse uma estação de abastecimento A e não estivesse do lado do hipermercado no corte mínimo, era averiguado se o seu abastecimento B estava do lado contrário, ou

seja, do lado do hipermercado (origem do *push-relabel*) e se se confirmasse, a estação A era adicionado ao vetor “critical_stores” para, mais tarde, ser devolvido no *standard output*.

Para calcular os caminhos cuja capacidade é necesśario aumentar, o objetivo foi averiguar se a aresta era “cortada” pelo corte mínimo, exceto as arestas que ligava as estações de abastecimento A e B. Primeiramente, foi também criado um vetor “to_upgrade”, neste caso, de pares de inteiros para registar os vértices origem e destino da ligação a aumentar. Deste modo, por cada vértice, se pertencesse ao lado do hipermercado, ou seja, se tivesse altura igual ou maior ao número de vértices, a sua lista de arestas é percorrida e os vértices destino das mesmas são visitados. Por cada aresta, se o vértice a ser consultado e se o vértice visitado forem ambos estações de abastecimento, o ciclo continua. Se o vértice a ser visitado não está no mesmo lado do corte mínimo, quer seja um vértice normal quer seja um vértice Abastecimento B, os índices são convertidos do valor do grafo transposto para o valor do grafo dado como *input* e adicionados ao vetor “to_upgrade”.

Finalmente, ambos os vetores ordenados com uma função de *sort* e, posteriormente são percorridos e os seus elementos são enviados para o *standard output*.

Análise Teórica

Após uma análise minuciosa ao código, tornou-se possível precisar o tempo de execução de cada ciclo, e, por conseguinte, a complexidade do programa. Tomando-se V como o número de fornecedores + estações de abastecimento + hiper (vértices do grafo) e E como as conexões entre fornecedores/ estações de abastecimento/ hiper (arestas do grafo):

1. Inicialização dos pontos da rede de distribuição da cadeia de hipermercados (vértices): O(V);
2. Leitura das conexões entre dois vértices: O(E);
3. Aplicação do algoritmo de obtenção do fluxo máximo (*Push Relabel* com uma *queue* de vértices com excesso) : $O(V^3)$;
4. Aplicação do algoritmo de obtenção do corte mínimo : O(VE);
5. Aplicação do Algoritmo de *sort* às listas a serem de elementos a serem impressos: O(V*log(V));
6. Impressão das estações de abastecimento/ conexões a serem aUmentadas: O(V + E);

Tendo em consideração as várias complexidades apresentadas acima, constatámos que a complexidade total da nossa solução é: $V + E + V^3 + VE + V*\log(V) + V + E = O(V^3)$.

Avaliação Experimental dos Resultados

A plataforma de teste maioritariamente utilizada ao longo do desenvolvimento do programa foi um computador com processador
Intel® Core™ i7-4500U CPU 1.80GHz com 16 GB de memória DDR3 a correr o sistema operativo Ubuntu 18.04 LTS.

Corremos o programa com diversos ficheiros de input com as funcionalidades *time* e *Valgrind* do Linux para obter os tempos e memória utilizada por cada teste.

Casos de teste ilustrativos :
(criados a partir do gerador aleatório)

Pontos de interesse (Fornecedores + Abastecimentos + hiper) (V)	Pontos de interesse³ (V³)	Tempo de execução (s)	Memória alocada (Bytes)
10	1000	0.004	82296
50	125000	0.007	122456
100	1000000	0.009	209000
200	8000000	0.014	522280
300	27000000	0.024	1018376
400	64000000	0.042	1585992
500	125000000	0.06	2231784
600	216000000	0.085	3278424
700	343000000	0.115	4391320
800	512000000	0.153	5874712
900	729000000	0.191	7539352
1000	1000000000	0.243	9483192

Tabela 1. Desempenho da solução com o aumento de pontos de interesse

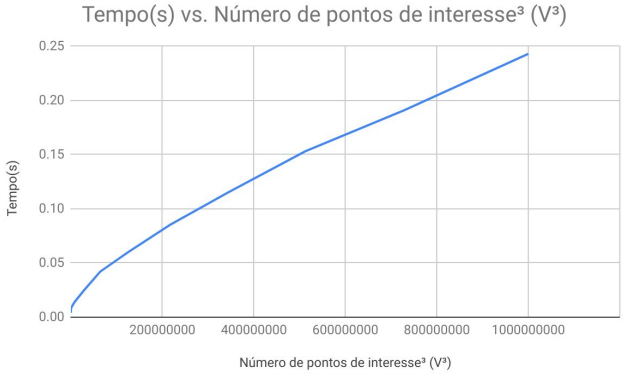


Gráfico 1.

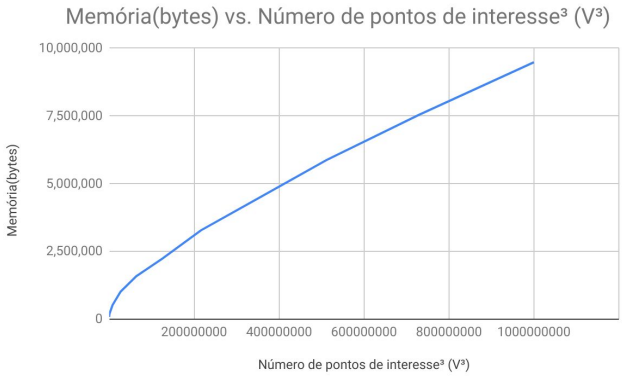


Gráfico 2.

Tal como abordado na análise teórica, a complexidade do programa é $O(V^3)$. Foram então realizados testes experimentais para comprovar esta teoria.

Após 11 testes, verificou-se que com o aumento do cubo dos pontos de interesse, tanto o tempo de execução como a memória alocada aumentavam aproximadamente na mesma proporção (linearmente), como se pode ver nos gráficos acima.

Deste modo, verifica-se que a complexidade do programa é, de facto, $O(V^3)$.

Referências

[1] https://www.geeksforgeeks.org/push-relabel-algorithm-set-2-implementation;
[2] https://en.wikipedia.org/wiki/relabel_maximum_flow_algorithm;

[3] PowerPoints facultados pelo professor (Fluxos Máximos - Pré-Fluxos (Cap.26))