Relatório - 2º Projecto
Análise e Síntese de Algoritmos
2015/2016

Grupo 58
Pedro Orvalho 81151
António Lourenço 81796
23 de Abril de 2016

1.Introdução

Este relatório visa a análise e solução da questão-problema: **como identificar da forma mais eficiente o menor caminho a precorrer entre vários vértices, ou seja, o ponto intermédio e o respectivo custo;** tal como proposto como segundo projecto de Análise e Síntese de Algoritmos do segundo semestre do ano 2015/2016.

1.1.O problema

Tendo um grafo dirigido e pesado sem ciclos negativos temos de arranjar um algoritmo, o mais eficiente possível, que descobre qual o caminho de menor custo entre todos os vértices para um vértice em comum.

Dando o exemplo do enunciado: uma empresa de distribuição que está a tentar realizar um encontro com todas as suas filiais tem de saber qual é o melhor percurso (i.e. o menos custoso) que cada filial deve realizar a fim de se poderem juntar numa localidade. Cada localidade é expressa como um vértice de um grafo e cada trajeto como uma aresta.

1.2.*Input & Output*

Partindo de um *input* onde inicialmente é fornecido o número de vértices(localidades), filiais e arestas(rotas) de um grafo em análise, segue-se uma linha com f números onde f é o número de filiais presentes no grafo e cada número é o índice da localidade onde a filial está situada. De seguida, são dadas C linhas (sendo C o número de arestas do grafo) onde são indicados dois vértices que possuem uma ligação entre si, u e v, com o respectivo custo(w) da aresta de u para v.

Daqui resulta um *output* onde é indicado o índice do ponto de encontro mais vantajoso para todas as filiais da empresa, isto é, o ponto de encontro em que a empresa perderá menos dinheiro ao realizar as deslocações. Seguido do índice da localidade vem o custo total de deslocação para a mesma. É também dado, numa segunda linha de *output,* para cada filial o custo da mesma ao deslocar-se para o ponto de encontro.

2. Descrição da Solução

2.1. Escolha de Tecnologias e Estruturas de Dados

Após uma análise do problema concluímos que seria mais vantajoso utilizar a linguagem de programação C++ não só por questões de eficiência, mas também dado a disponibilidade de estruturas de dados já existentes. Destas estruturas é de salientar a utilização das bibliotecas do std, nomeadamente std::vector. Através desta estrutura designamos um grafo como sendo um vetor de vetores de arestas (vector< vector<Aresta> >), representando deste modo uma lista de adjacências, sendo “Aresta”, uma estrutura que tem três inteiros, uma origem, um destino e um custo.

2.2. Solução e Algoritmo

Representando cada localidade como um vértice e cada rota entre duas localidades como uma aresta forma-se assim um grafo dirigido que simboliza a rede de rotas da empresa. Cada filial pertence a uma localidade e, posto isto, o grupo de filiais é representado por um vector de inteiros em que cada posição do vector tem o índice da localidade respectiva onde está situada cada uma das filiais.

De forma a resolver o problema proposto baseamo-nos no algoritmo de Johnson.

Começamos por atribuir a cada localidade um vértice do grafo e as suas respectivas arestas as “ligações”. Reservamos também espaço na memória para os seguintes vetores: o vetor de inteiros para guardar os índices das filiais e um vector de arestas para simplificar o nosso algoritmo Bellman-Ford.
A solução consiste na realização de um algoritmo Johnson no qual, de acordo com o mesmo, adicionamos uma “raiz” ao grafo e colocamos as alturas de todos os vértices a essa raiz a zero, onde de seguida é aplicado o algoritmo Bellman-Ford para encontrar o caminho mais curto da “raiz” criada a todos os vértices.

Procedemos à repesagem dos arcos do grafo de maneira a colocar todos os pesos dos arcos com um valor positivo. Desta forma, passa a ser possível aplicar o algoritmo Dijkstra para cada filial do grafo tendo um vector de inteiros auxiliar onde vamos somando, posição a posição, o vector que obtemos do algoritmo Dijkstra que tem as distâncias da filial em causa a cada um dos vértices do grafo, sempre com cuidado com as posições a infinito para não causarem overflow. Com a aplicação do Dijkstra a todas as filiais do grafo conseguimos encontrar o ponto de encontro ideal para a empresa e, ao mesmo tempo, o custo total desta para fazer todas as suas filiais lá chegarem, fazendo as repesagens para os valores dos custos reais das arestas imprimimos assim no output qual o ponto de encontro e o custo total da empresa.

Após sabermos qual o ponto de encontro voltamos a chamar o algoritmo Dijkstra para cada filial e à medida que a função retorna, imprimimos como

output o valor do custo da filial em causa para o ponto de encontro, após uma nova repesagem a fim de obtermos o custo real unitário.
Em relação ao algoritmo Bellman-Ford temos um vector de inteiros que chamamos de alturas no qual vamos guardando a altura de cada vértice iteração a iteração. Para cada vértice relaxamos todas as arestas do grafo e verificamos se a altura do vértice de destino da aresta é superior à soma da altura do vértice de origem somado ao custo da aresta em si. Caso isto se verifique, a altura do vértice de destino passa a ser a soma da altura do vértice de origem mais o custo da aresta em si.

Em relação ao algoritmo de Dijkstra criamos um vector de inteiros chamado de distâncias que começa com todas as posições a infinito, criamos também uma pilha, estrutura std::set, que contêm pares de dois inteiros, um vértice a distância até este. De seguida inserimos o vértice que contem a filial a que estamos a aplicar o algoritmo Dijkstra na pilha e visitamos todas as arestas do vértice vizinho da mesma que se encontrem à distância mais próxima, n. V

Verificamos agora se a distância actual para o vértice de destino partindo da aresta que começa em n tem uma distância maior que a distancia de n somado ao custo da aresta que sai de n, se assim acontecer então a distancia do vértice de destino passa a ser a soma entre a distancia ate n mais o custo da aresta. Caso a distancia até ao vértice de destino seja diferente de infinito significa que já foi visitado anteriormente e posto isto vamos retirar o par deste da pilha. Caso contrário, ou seja, a distância ainda seja infinito, significa que o vértice ainda não foi visitado e, deste modo, inserimo-lo na pilha e fazemos update da distância até ao vértice.

3.Análise Teórica

Analisando a complexidade temporal de cada função do nosso programa, considerando V o número de vértice, E o número de arestas do grafo e F o número de filiais, chegamos à conclusão que:

- A função main() (Johnson) é O(VE + F(V+E)lgV):
- A leitura do input é ϴ(E + F);
- A inicialização do vector das arestas tem uma complexidade de O(C).
- A inicialização do vector das filiais tem uma complexidade de O(F).
- A função dijkstra() é O(F(V+E)lgV) : pois a função é chamada 2F vezes e percorre todos os vértices recursivamente e para cada vértice percorre as suas arestas, logo verifica cada aresta E o que resulta O(V+E), mas vamos retirando os vértices da pilha de onde vem o lgV.
- A função bellmanford() é O(VE), pois só é chamada uma vez, e para cada vértice vamos verificar todas as arestas do grafo.

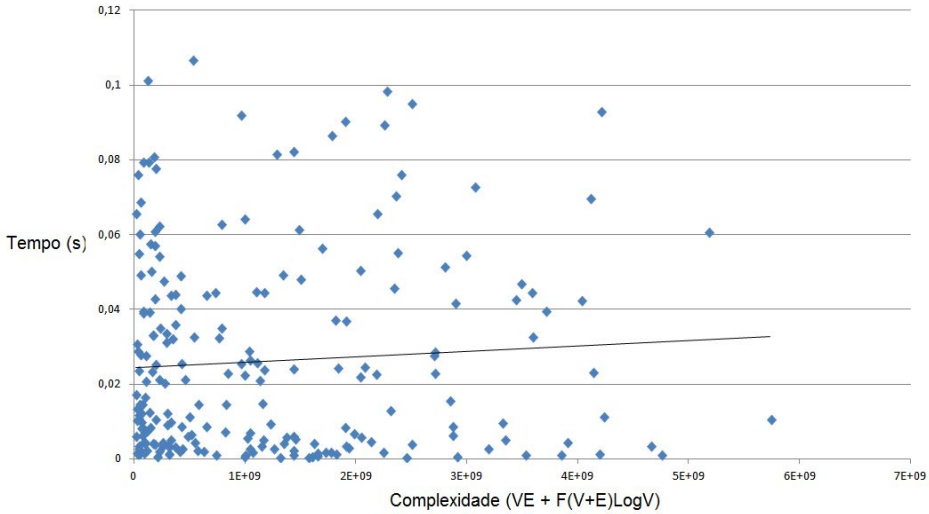
A complexidade temporal do programa é a mesma que a da função main() que é O(VE + F(V+E)lgV).

A complexidade espacial é O(V+E). Visto que criamos um grafo, V+E, que ocupa O(V+E), e 2 vetores, um de inteiros o vector filiais que é O(F) e um de Arestas o vector arestas que é O(E).

4. Avaliação Experimental dos Resultados

A fim de realizarmos uma avaliação experimental dos resultados decidimos que seria vantajoso criar um gerador de grafos. Foram então gerados cerca de 400 grafos completamente aleatórios de densidade variável. Corremos o programa sobre cada um dos grafos e, utilizando a ferramenta do linux perf, obtemos os tempos de execução.

Por fim, tratamos os dados no excel do qual conseguimos obter o seguinte gráfico:



Podemos então concluir que os resultados experimentais estão de acordo com os pressupostos teóricos. As dispersões existentes no gráfico devem-se à natureza dos grafos gerados. O nosso algoritmo tem uma complexidade O(VE + F(V+E)lgV), ou seja, o tempo de execução aumenta de forma linear numa relação com o número de vértices multiplicado ao número de arestas somado a uma relação logarítmica entre o número de filiais e o número de arestas e o número de vértices.

5. Referências utilizadas

1. https://www.youtube.com/watch?v=b6LOHvCzmkI
2. http://www.geeksforgeeks.org/johnsons-algorithm/
3. https://gist.github.com/ashleyholman/6793360
4. http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/
5. http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/
6. CLRS, Introduction to Algorithms - 3rd Edition (2009)