Análise e Síntese de Algoritmos
Relatório 2º Projeto (2015/2016)
Grupo 50
Rafael Belchior (80970), João Calisto (81329)
22 de Abril de 2016

Introdução

O problema proposto é referente ao cálculo de o melhor ponto de encontro entre os empregados de várias filiais de uma empresa, de maneira a minimizar custos de transporte até esse ponto de encontro. Este problema pode resumir-se a um problema de caminhos mais curtos, ou, neste caso, ao problema de encontrar o caminho de menor custo entre todos os vértices.

Descrição da Solução

Para resolver este problema, recorreu-se à estrutura grafo e a uma árvore binária. O grafo é dirigido e pesado, pois uma localidade pode ter um caminho para outra localidade, mas o recíproco pode não acontecer. Representou-se cada localidade como um vértice. Os caminhos entre as localidades (arcos) têm associado um valor de perda (peso), que corresponde ao custo de atravessar a localidade em questão menos a receita gerada por essa travessia. A um menor valor de perda corresponde uma maior receita. É, portanto, preferível ter valores de perda negativos.

Consideremos que o número de filiais é significativamente menor que o número de localidades e o número de pares de localidades entre os quais há um valor de perda é muito menor que o número de todos os pares possíveis. Deste modo, temos um grafo esparso. Podemos, então, resolver este problema recorrendo a uma versão do algoritmo de Johnson**[1]**. Neste caso, o algoritmo de Johnson é mais eficiente que o de Floyd-Warshall, porque se opera sobre um grafo esparso. Além disso, podemos fazer aperfeiçoamentos ao algoritmo de Johnson, tendo em conta as especificidades do problema (descritos a seguir).

O algoritmo proposto é o algoritmo de Johnson e foi baseado nas aulas teóricas de ASA 15/16 e no pseudo-códig2o da página web Chegg[**2**].

Foram utilizadas as convenções dadas nas aulas teóricas para algumas propriedades dos vértices, no âmbito da execução do algoritmo de Dijkstra e Bellman-Ford, nomeadamente na inicialização dos vértices. O pai de cada vértice é NILL e a distância do vértice fonte a cada outro é infinito (excepto para o próprio vértice fonte, em que a distância é 0).

A estrutura Grafo contém variáveis com o número de vértices e arcos. Além disso, possui um vetor de *nodelinks* (ponteiro para *node),* sendo que cada índice representa um vértice. Cada *nodelink* tem uma estrutura *Edge,* cuja função é representar os arcos entre o próprio vértice e os restantes. Cada node contém o menor caminho do próprio vértice para o seguinte e a sua altura (relativo a Bellman-Ford). Na estrutura Edge, guarda-se o peso do arco em questão, assim como o seu peso repesado e o vértice para onde aponta.

Temos também uma estrutura minHeap, que contém uma variável com o número de elementos da heap e um vetor de vértices.

Descrição informal da solução. Seguidamente será feita uma explicação detalhada.

1. Ler e tratar input.

2. Juntar ao grafo G um vértice S.

3. Correr Bellman-Ford sobre G, com início em S.

4. Para cada vértice, atribuir um peso e fazer a re-pesagem dos arcos. Para cada vértice guarda-se o novo peso**.**

5. Para cada filial, aplicar Dijkstra, calculando-se o resultado final. Obtida a localidade, aplicar Dijkstra, de modo a saber a distância de menor custo a todas as filiais. Desfazer a re-pesagem, obtendo-se a solução.

Primeiramente, constrói-se um grafo tendo em conta o input. O vértice S que é adicionado ao grafo tem, inicialmente, para cada arco seu, peso 0.

Seguidamente aplica-se Bellman-Ford sobre o grafo. Foram feitas várias otimizações ao algoritmo: Não há que verificar se existem ciclos negativos, na última iteração. se não ocorrer nenhuma relaxação durante uma iteração, o algoritmo termina. Se um arco não sofrer relaxação durante uma iteração, não sofrerá mais nenhuma e, portanto, é excluído das próximas iterações.

Após a aplicação de Bellman-Ford, temos o caminho mais curto entre o vértice S e todos os outros. Seguidamente, faz-se a re-pesagem dos arcos, de modo a não se ter valores negativos para os pesos. A re-pesagem é feita de acordo com a expressão: $w'(u,v) = w(u,v) + h(u) - h(v)$, em que w' é o novo peso do arco (u,v). Aplica-se, então, Dijkstra para cada vértice do grafo, com os arcos re-pesados. Para cada vértice u, obtem-se $\delta(u,v)$, para todos os vértices. Desfazemos a re-pesagem dos arcos, imediatamente antes de calcular a solução final, de acordo com a expressão: $\delta(u,v) = \delta'(u,v) + h(v) - h(u)$, em que $\delta'(u,v)$ é o caminho mais curto entre u e v.

Para determinar a localidade que deverá ser utilizada como ponto de encontro, basta somar os valores de perda associados aos arcos à medida que Dijkstra é executado. Os valores de menor custo para cada filial obtêm-se aplicando Dijkstra a cada uma das filiais.

Análise Teórica

O grafo é construído em O (1), pois inserir um elemento numa lista de adjecências é linear.

Para facilitar a análise da solução, procede-se à sua divisão em duas partes.

O algoritmo de Bellman-Ford é executado em tempo O (VE).

O algoritmo de Dijkstra é executado em O ((V+E)log(V)), F vezes. Ainda se aplica Dijkstra à localidade de encontro, para as filiais todas. Isto tem um custo de O(E+log(F)*F) em que F é o número de filiais, V é o número de vértices e E é o número de arcos.

Conclui-se, então, que a complexidade do algoritmo usado para obter a solução do problema é O(F(V+E)log(V)+VE).

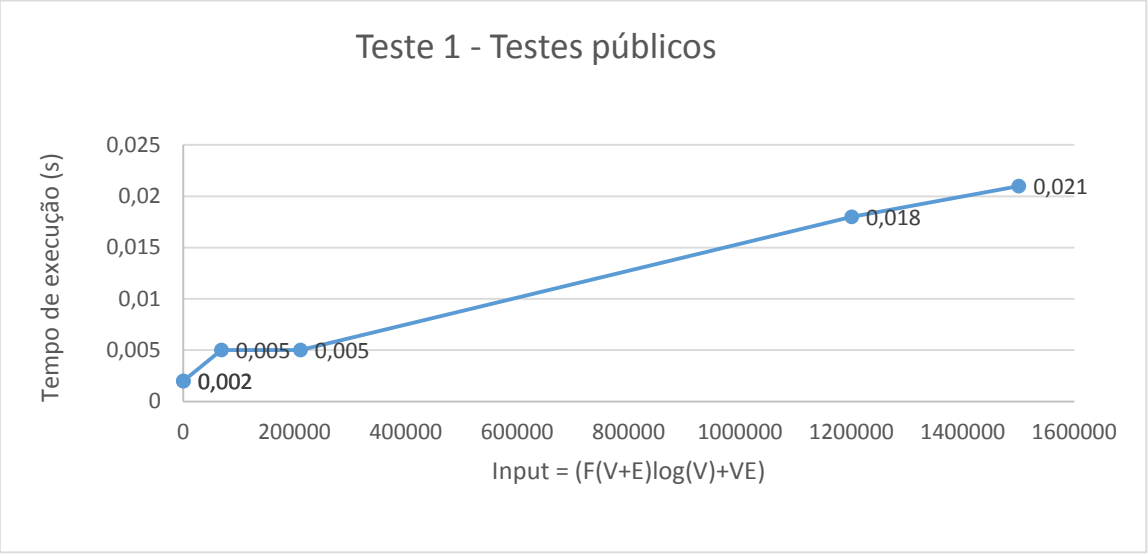
Avaliação Experimental

Para a avaliação experimental, foram feitos vários testes, sob diferentes condições. Variou-se o tamanho do input. O tempo de execução foi obtido com o comando *time*, e nos valores apresentados foi usada a média temporal de três ensaios para cada um dos inputs, para cada teste.

Teste 1 – Testes públicos fornecidos

Analisando a execução do programa com os inputs disponibilizados na página da cadeira, concluiu-se que o crescimento do tempo de execução é praticamente linear, como esperado pelo estudo realizado Análise Teórica, tendo em conta as características do problema.

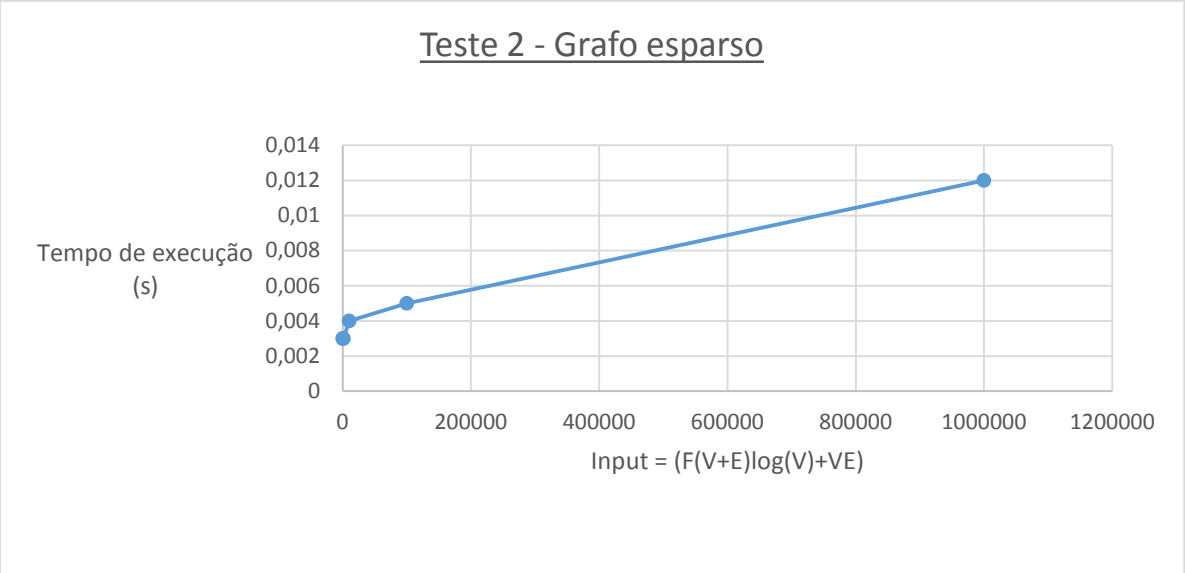
O seguinte gráfico mostra a curva que melhor se ajusta aos resultados experimentais (tempo em função do input).



Teste 2 – Grafo esparso

No 2º teste o programa foi testado com um grafo esparso. Analisando a execução do programa, concluiu-se que o crescimento do tempo de execução é praticamente linear, como esperado pelo estudo realizado Análise Teórica.

O seguinte gráfico mostra a curva que melhor se ajusta aos resultados experimentais.



Referências

[1]:https://pt.wikipedia.org/wiki/Algoritmo_de_Johnson
[2]:http://www.chegg.com/homework-help/questions-and-answers/code-johnson-s-algorithm-using-pseudo-code--input-integer-v-100-designating-number-vertice-q6283981