Relatório Projecto ASA

segunda Parte

TÉCNICO LISBOA

Grupo 64:

Ricardo Costa Dias Rei, 78047

Mariana Gaspar Fernandes, 76448

Breve Introdução:

Como problema foi-nos pedido para criar um software que ajuda-se o senhor Coelho a resolver o planeamento de rotas para a sua empresa.

O objetivo é estabelecer para cada localidade qual é o percurso que, com origem na sede da empresa, minimiza o valor de perda total.

Descrição da solução:

Ao olhar para o problema identificámos que poderia ser traduzido num grafo em que cada localidade representa um vértice e as arestas são as ligações entre as mesmas, sendo que um percurso será um conjunto de arestas.

Assim sendo, criámos uma função buildGraph que ao receber o input construía o grafo correspondente. Na construção do grafo é utilizada a estrutura Graph que tem o campo "_source" para guardar o vértice que representa a sede, o campo "verticesNumber" que guarda o numero de vértices, o campo "_vertices" para os vários vértices e o campo "adjacentList" que representa as arestas.

Depois aplicando o algoritmo Shortest-Path-Faster-Algorithm, ao grafo vamos conseguir saber o custo do caminho mais curto de cada vértice (localidade) ao vértice que representa a sede. Como suporte à execução do mesmo, a estrutura Vertex tem o campo "d" para guardar o custo do caminho mais curto, o campo "mark" para saber quantas vezes ele foi inserido na queue, o campo "active" para saber se ele esta na queue naquele momento e o campo cycle que vai ser usado para marcar os vertice atingíveis através da raiz de um ciclo negativo.

Inicialmente pensamos na nossa solução com um bellman-ford tradicional mas após a realização de alguns teste reparamos que a complexidade escalava linearmente com o numero de vértices o que em gratos muito grandes implicaria um tempo de execução demasiado elevado. Após algumas leituras maioritariamente na internet descobrimos que a aplicação do Shortest-Path-Faster-Algorithm seria a mais apropriada.

Assim sendo, após a utilização do algoritmo vamos fazer um ciclo extra que procura os vértices que tenham sido colocados na queue |V| vezes (ou seja as raízes dos ciclos negativos) e através desses vértices marcar no campo "cycle" que eles são acessíveis através de um ciclo negativo.

Por fim usámos uma função a que chamamos outPut percorre todos os vértices e dá o resultado pretendido, I para vértices acessíveis através de ciclos negativos, U a vértices que não conseguimos atingir através da fonte, e o custo mínimo para os restantes.

Análise Teórica de Resultados:

O nosso projecto tem como limite assimptótico superior O(V+E + V.E) em que V representa o número de vértices e E é o número de arestas. As funções mais importantes a nível de complexidade no programa são:

- buildGraph que percorre todos os vértices para os inicializar e todas as arestas para construir a lista de adjacências O(V+E).

- SPFAalgorithm vai repetidamente selecionar um vertice e aplicar a operação de relaxamento, se possível, a todos os adjacentes. Se um nó for relaxado, então provavelmente vai ser usado para relaxar outros vértices logo é colocado na queue. Quando não existirem mais vértices a considerar o algoritmo termina. No pior caso O(V.E) assim como o Bellman-Ford mas para grafo esparsos, com poucos ciclos aproxima-se do Dijkstra, portanto Ω(E+V.logV + V).

- markVertices que dado um vertice vai percorrer a lista de adjacências do grafo para aquele vertice e marcar todos os vértices atingíveis, semelhante à função DFS-visit. O(E)

- Por fim a função outPut que percorre todos os vértice e gera o output pretendido O(V).

Análise experimental:

O gráfico nº1 é o resultado de alguns testes realizados em que o pretendido era verificar a influencia do numero de arestas sobre o tempo de execução do algoritmo, ou seja verificar se o facto do grafo ser denso ou esparso era um factor de influencia. Para isso foram medidos os tempos para inputs com grafos de diferentes numero . O numero de arestas é dado por E. Nos testes realizados para a construção deste gráfico o valor de V é constante (15000 vértices). O aumento linear vai de acordo com o esperado devido a maior quantidade de arestas a serem relaxadas ao longo da execução.

O gráfico nº 2 conjuga a evolução do tempo de execução á medida que o grafo aumenta de tamanho e verifica a influencia de ciclos negativos nessa mesma relação. Podemos ver através do grafo que a presença de ciclos piora bastante a rapidez do algoritmo, o que vai de acordo com o esperado, uma vez que representa o pior caso. Uma analise posterior que consideramos que poderia ser interessante era a comparação entre o algoritmo aplicado e o Dijkstra em grafos com pesos positivos, de forma a verificar a semelhança de complexidade entre ambos.

Em suma confirmamos que a complexidade é influenciada pelo tamanho do grafo de forma linear e pela presença de ciclos negativos, sendo que a presença dos mesmo representa os piores casos.