TP1 de Algoritmos e Estruturas de Dados III

Lucas O. Silvestre[1]

[1]Sistemas de Informação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, MG – Brazil

lsilvs@dcc.ufmg.br

1. Objetivo inicial

O presente trabalho visa implementar uma demanda fictícia de um projeto que busca decidir qual em qual cidade uma filial de uma empresa de distribuição que usa malha rodoviária para suas entregas deve se instalar. A empresa tem conhecimento das distâncias entre quaisquer duas cidades em que atua.

Um ponto importante desse cenário é que só um produto é entregue por viagem. Ou seja, o caminhão de transporte sai da filial, faz a entrega e retorna para a filial para buscar o próximo produto a ser entregue.

A melhor cidade para sediar a filial deve ser escolhida dado três cenários diferentes. A seguir, veremos cada um dos cenários e sua implementação.

2. Modelagem e Solução proposta

A modelagem implementada consiste na leitura de um arquivo de entrada especificado na chamada do programa. A primeira linha do arquivo de entrada representa o número de instâncias que o arquivo contém. A linha seguinte representa o número de cidades que a empresa atende. As linhas seguintes representam um matriz que indica quais cidades são interligadas e qual a distância entre elas. Por último, são apresentados o número médio de pedidos de cada cidade. Esse padrão se repete para o número de instâncias.

A solução proposta foi implementada utilizando o conceito de grafos e suas possibilidades. Assim, iniciamos a leitura do arquivo de entrada salvando os dados em variáveis. Observe que a consistência do padrão do arquivo de entrada é de extrema importância para o funcionamento do programa. Partindo do ponto que o arquivo de entrada segue o modelo especificado, o algoritmo proposto lê o número de instâncias e itera sobre ele.

Para os três cenários foi aplicado uma implementação do algoritmo de Dijkstra que retorna um vetor com o menor caminho entre cada dois vértices de um grafo. Vejamos cada cenário especificamente.

2.1. Cenário 1

No primeiro cenário, devemos retornar a cidade de modo a minimizar os custos de entrega da empresa. Por enquanto, iremos considerar que cada cidade entrega uma mesma quantidade de pedidos em um mesmo intervalo de tempo a cada cidade.

Como o objetivo é minimizar os gastos, sobretudo de combustível, devemos escolher a cidade em que a soma de suas menor distância entre a outras seja menor. Como a função de Dijkstra implementada retorna o vetor com a menor distância entre cada duas cidades, basta aplicar nossa função a cada cidade e retornar a que tiver a menor soma de caminhos mais curtos.

```
TipoValorVertice cenario1(TipoGrafo *Grafo) {

    TipoValorVertice raiz = 0;
    int i, u, dist = 0, aux = 0, result = 0;
    TipoPeso * P = Dijkstra(Grafo, &raiz);

    for (u = 0; u < Grafo->NumVertices; u++) {
        dist += P[u];
    }

    for(i = 1; i < Grafo->NumVertices; i++) {
        raiz++;
        aux = 0;

        P = Dijkstra(Grafo, &raiz);

        for (u = 0; u < Grafo->NumVertices; u++) {
            aux += P[u];
        }

        if(aux < dist){
          result = i;
          dist = aux;
        }
    }

    return result + 1;
}
```

2.2. Cenário 2

Aqui, devemos levar em consideração o número de pedidos de cada cidade. Uma vez que algumas cidades possuem muito mais pedidos que outras, isso pode influenciar bastante no consumo de combustível. O objetivo agora é descobrir qual a melhor cidade sede da empresa de modo a ter o menos gasto levando em conta o número médio de pedidos de cada cidade.

Mais uma vez utilizaremos a função Dijkstra. Porém, dessa vez iremos multiplicar o caminho mais curto entre duas cidades pelo número médio de pedidos da mesma, de modo a ter uma variável ponderada entre elas. Assim, retornamos a cidade que possuir o menor valor ponderado.

```
TipoValorVertice cenario2(TipoGrafo *Grafo) {

    TipoValorVertice raiz = 0;
    int i, u, result = 0;
    float dist = 0.0, aux = 0.0;

    TipoPeso * P = Dijkstra(Grafo, &raiz);

    for (u = 0; u < Grafo->NumVertices; u++) {
        dist += (float) P[u]*Grafo->Pedidos[u];
    }

    for(i = 1; i < Grafo->NumVertices; i++) {
        raiz++;
        aux = 0;

        P = Dijkstra(Grafo, &raiz);;

        for (u = 0; u < Grafo->NumVertices; u++) {
            aux += (float) P[u]*Grafo->Pedidos[u];
        }

        if(aux < dist){
          result = i;
          dist = aux;
        }
    }

    return result + 1;
}
```

2.3. Cenário 3

O cenário 3 propoe oferecer aos clientes um serviço que consiste em uma taxa adicional paga no momento da compra de modo a garantir que o produto seja entregue dentro de um prazo máximo. Porém, para que essa taxa não seja muito alta, desejamos escolher a cidade sede de modo que o prazo máximo seja o mínimo possível. Ou seja, temos que escolher a cidade que possui a menor distância dentre as mais distântes.

Para isso, através da função de Dijkstra, descobrimos a distância máxima partindo de cada cidade e escolhemos a que possuir a menor distância dentre essas máximas. Assim, podemos garantir o tempo máximo de entrega e minimizar os custos dessa entrega especial.

```
TipoValorVertice cenario3(TipoGrafo *Grafo) {

    TipoValorVertice raiz = 0;
    int result = 0, maior = 0, aux = INFINITO, i, u;
    TipoPeso * P;

    for (i = 0; i < Grafo->NumVertices; i++) {
        maior = 0;

        P = Dijkstra(Grafo, &raiz);
        for (u = 0; u < Grafo->NumVertices; u++) {
            if (maior < P[u]) {
                maior = P[u];
            }
        }

        if (maior < aux) {
            result = raiz;
            aux = maior;
        }

        raiz++;
    }

    return result+1;
}
```

2.4. Cenário Prejuízo

Por fim, temos um cenário que deve retonar o prejuízo percentual acarretado caso a cidade escolhida seja a que garante o menor tempo de entrega máximo.

Para tal situação, basta fazer uma combinação do cenário 1 com o cenário 3 (uma vez que esse cenário não deve levar em consideração o número de pedidos de cada cidade.). Assim, rodamos a funcão Dijkstra para cada cidade e ao mesmo tempo em que definimos a soma das distâncias do cenário 1, também definimos a soma das distâncias do cenário 3. Após isso, basta fazer uma regra de três com os resultados encontrados.

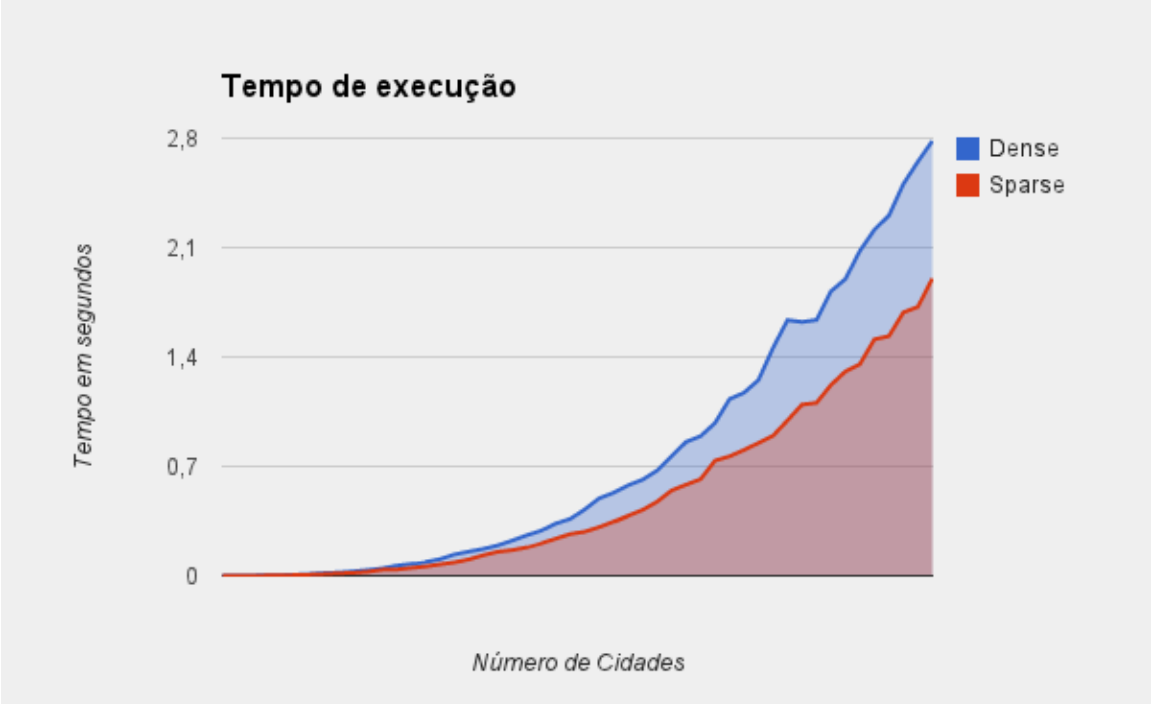
3. Análise de Complexidade

Como já é conhecido, a complexidade do algoritmo de Dijkstra é |V| log |V|. Porém, em nossa implementação executamos a função Dijkstra para cada cidade (ou vértice) de modo que a complexidade passa a ser $|V^2|$ log |V|.

4. Experimentos

Como experimentos de testes, utilizamos os arquivos disponibilizados para tal objetivo. Os arquivos são divididos em duas categorias: um dense.in, que contem grafos densos e um sparse.in, que contem grafos esparso. Ambos arquivos possuem 50 instâncias sendo cada uma incrementado de 10 em 10 cidades.

Abaixo apresentamos o gráfico de tempo de execução para cada um dos arquivos de entrada aumentando-se o número de cidades.



5. Especificação

Todo o desenvolvimento do código foi realizado utilizando um MacBook Air com as seguintes especificações:

Processador: Intel Core i5 1,7 GHz

Memoria: 4 GB 1333 MHz DDR3

Sistema Operacional: OS X 10.8.2

IDE: Sublime Test 2

GCC: i686-apple-darwin11-llvm-gcc-4.2 (GCC) 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2336.11.00)

6. Conclusão

A princípio, a especificação parece complicada. Porém, encarando as propostas como um problema de grafos, podemos utilizar suas estruturas e algoritmos para resolver facilmente as demandas.