

Análise e Síntese de Algoritmos

2º Projeto

Tomás Cunha, nº 81201, Grupo 15

1. Introdução

Este projeto tem como objetivo encontrar um ponto de encontro entre várias filiais de forma a minimizar o custo total das rotas, se existir. O problema pode ser reduzido a encontrar a menor soma dos custos dos caminhos mais curtos de todas as filiais para cada localidade, representando os caminhos como arestas de um grafo e os vértices como as localidades. Na resolução do problema utilizei a descrição do algoritmo de Johnson e da estrutura de dados Min-Heap disponíveis no livro *Introduction to Algorithms*[1].

2. Descrição da solução

A solução encontrada consiste em realizar uma variação do algoritmo Johnson tomando como vértices de fonte todas as filiais. Em vez de guardar todos os caminhos mais curtos numa matriz, é apenas guardada a soma dos caminhos até cada localidade num vetor, reduzindo o espaço ocupado. No final, este vetor é percorrido para encontrar a soma mínima. Após encontrar o ponto de encontro correto, é calculado o grafo transposto do original e realiza-se o algoritmo Dijkstra a partir do ponto de encontro, de forma a obter os custos individuais dos caminhos de cada filial até ao ponto de encontro.

O algoritmo pode ser representado em pseudocódigo da seguinte forma (os algoritmos de Dijkstra e Bellman-Ford são omitidos uma vez que não foram alterados em relação aos originais):

Algorithm 1: Função principal

```
1 Let  $G \leftarrow$  Grafo formado a partir do input;  
2 Let  $F \leftarrow$  Vértices de  $G$  correspondentes às filiais;  
3 Let  $w(u,v) \leftarrow$  Função de pesos que devolve a perda entre  $u$  e  $v$ ;  
4 meeting-place, total-loss,  $d \leftarrow \text{get-result}(G, F, w)$ ;  
5 if  $\text{meeting-place} = \emptyset$  then  
6 |   Output “N”;  
7 else  
8 |   Output meeting-place, total-loss;  
9 |   foreach  $v \in F$  do  
10 | |   Output  $d[v]$ ;
```

Algorithm 2: Descobrir o ponto de encontro, se existir, e as distâncias das filiais a este

```
1 function  $\text{get-result}(G, F, w)$   
2 |   foreach  $v \in V[G]$  do  
3 | |    $\text{reachable}[v] \leftarrow \text{true}$ ;  
4 | |    $\text{sum}[v] \leftarrow 0$ ;  
5 |    $G' \leftarrow G \cup s$ ;  
6 |    $d[v] \leftarrow 0 \forall v \in V[G]$ ;  
7 |    $\text{bellman-ford}(G', s, w)$ ;  
8 |    $h(v) = \delta(s, v)$  calculado pelo  $\text{bellman-ford} \forall v \in V[G]$ ;  
9 |    $w'(u, v) = w(u, v) + h(u) - h(v) \forall (u, v) \in E[G]$ ;  
10 |  foreach  $u \in F$  do  
11 | |    $\text{dijkstra}(G, u, w')$ ;  
12 | |   foreach  $v \in \delta'(u, v)$  calculados por  $\text{dijkstra}$  do  
13 | | |   if  $\delta'(u, v) = \infty$  then  
14 | | | |    $\text{reachable}[v] \leftarrow \text{false}$ ;  
15 | | |    $\text{sum}[v] \leftarrow \text{sum}[v] + \delta'(u, v) + h(v) - h(u)$ ;  
16 |    $\text{meeting-place} \leftarrow \emptyset$ ;  
17 |    $\text{total-loss} \leftarrow \infty$ ;  
18 |   foreach  $s \in \text{sum}$  do  
19 | |   if  $\text{reachable}[\text{index}[s]]$  and  $s < \text{total-loss}$  then  
20 | | |    $\text{meeting-place} \leftarrow \text{index}[s]$ ;  
21 | | |    $\text{total-loss} \leftarrow s$ ;  
22 |   if  $\text{meeting-place} \neq \emptyset$  then  
23 | |    $d \leftarrow \text{dijkstra}(G^T, \text{meeting-place}, w')$ ;  
24 | |   foreach  $v \in F$  do  
25 | | |    $d[v] \leftarrow d[v] + h(\text{meeting-place}) - h(v)$ ;  
26 |   return  $\text{meeting-place}, \text{total-loss}, d$ ;
```

3. Análise Teórica

A correção dos algoritmos de Johnson, Dijkstra e Bellman-Ford já foi provada. A perda total será necessariamente dada pela soma dos caminhos mais curtos a partir de cada filial até ao ponto de encontro, logo encontrar a soma mínima irá encontrar o ponto de encontro certo. A realização do algoritmo de Dijkstra a partir do ponto de encontro no grafo transposto irá dar os caminhos mais curtos a partir das filiais no grafo original até esse ponto de encontro: se em algum ponto a distância fosse diferente, teria obrigatoriamente de ser escolhido esse arco no caminho do grafo original, uma vez que os arcos são exatamente os mesmos.

Os dados do enunciado garantem que o número de filiais é sempre muito menor do que o número de localidades. Seja F o número de filiais, V o número de vértices e E o número de arestas. Analisemos o pseudocódigo da função **get-result**, correspondente ao *Algorithm 2* da secção anterior para obter a sua complexidade:

As linhas 5, 8–9, 16–17 e 22 são executadas em $\Theta(1)$. As linhas 2–4 têm complexidade $\Theta(V)$. A linha 6 é executada em $\Theta(V)$. A linha 7 executa o algoritmo de *Bellman-Ford*, que tem complexidade $\mathcal{O}(VE)$. Há um ciclo nas linhas 10–15 que é executado F vezes. Na linha 11, o algoritmo de *Dijkstra* é executado, logo a sua complexidade é $\mathcal{O}((V + E) \log V)$. Nas linhas 12–15, há um ciclo que percorre os custos dos caminhos mais curtos para todos os vértices do grafo dados pelo *Dijkstra*, tendo complexidade $\Theta(V)$.

Assim, as linhas 10–15 têm complexidade $\mathcal{O}(F((V+E)\log V + V))$, mas uma vez que $\mathcal{O}(V)$ é majorado por $\mathcal{O}((V+E) \log V)$, é simplesmente $\mathcal{O}(F(V+E)\log V)$. As linhas 18–21 têm complexidade $\Theta(V)$. A linha 23 tem complexidade $\mathcal{O}((V+E)\log V)$, na execução do algoritmo de Dijkstra, e $\mathcal{O}(V + E)$ na construção do grafo transposto. Uma vez que $\mathcal{O}(V + E)$ é majorado por $\mathcal{O}((V+E)\log V)$, tem complexidade $\mathcal{O}((V+E)\log V)$. As linhas 24–25 têm complexidade $\Theta(F)$.

Uma vez que o número de filiais é significativamente menor do que o número de localidades, a complexidade da linha 7 não será necessariamente majorada pela complexidade das linhas 10–15, o que leva a uma complexidade total de $\mathcal{O}(F(V+E)\log V + VE)$.

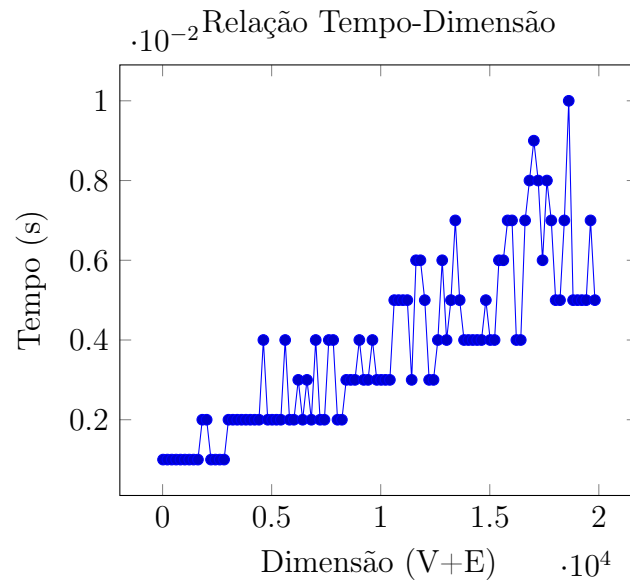
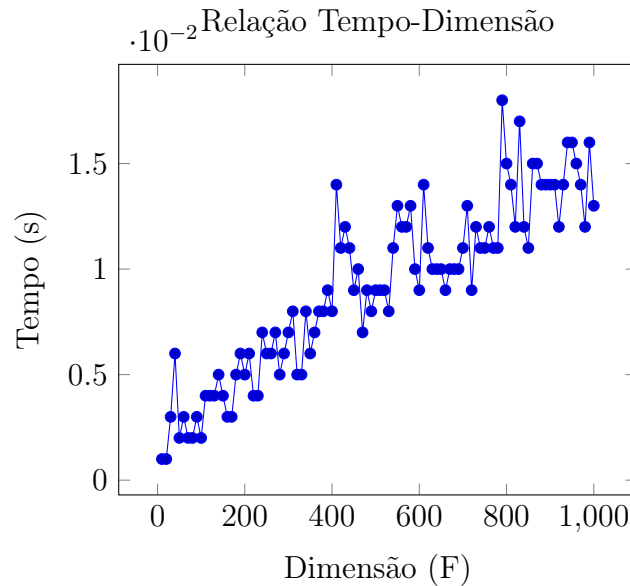
Analisando o pseudocódigo da função principal, observa-se que as linhas 1–3 são realizadas em $\mathcal{O}(V + E + F)$, a linha 4 é realizada em $\mathcal{O}(F(V+E)\log V)$ (como estudado no parágrafo anterior), as linhas 9–10 são executadas em $\Theta(F)$ e as restantes linhas são executadas em $\Theta(1)$.

Conclui-se então que a complexidade temporal do programa é majorada pela função **get-result**, logo a complexidade total é $\mathcal{O}(F(V+E)\log V + VE)$.

Analisando a complexidade espacial, nota-se que é $\mathcal{O}(V + E + F)$, uma vez que para além do grafo (e do seu transposto calculado na função **get-result**) apenas são necessários vectores com V ou F elementos (mais concretamente, o vetor de filiais inicializado na linha 2 da função principal, e os vetores das linhas 3 e 4 da função **get-result**).

4. Avaliação Experimental dos Resultados

Fazendo dois tipos de testes diferentes, um em que varia o número de filiais e outro onde varia o número de vértices e arestas, é possível ver que o primeiro tem um crescimento linear, e o segundo tem o crescimento do tipo $y = x \log x$, sendo coerente com os resultados teóricos esperados.



Referências

- [1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, 3rd Edition, September 2009