TP1: This city is on fire!

Reuben Nascimento Morais

23 de outubro de 2015

1 Introdução

O objectivo desse trabalho prático é encontrar um caminho entre dois quarteirões de uma cidade de forma que, ao percorrer este caminho, a probabilidade de ocorrer um incêndio seja minimizada. Além disso, é preciso evitar se afastar demais de algum corpo de bombeiros, e em caso de empate entre dois caminhos, devemos desempatá-los de acordo com a ordem lexicográfica das sequências de quarteirões.

A solução apresentada consiste em modelar o mapa como um grafo e então usar um algoritmo de caminho mais curto para encontrar a solução desejada. Caso não seja possível encontrar um caminho mais curto, isso é indicado na saída.

2 Modelagem do problema

O mapa da cidade foi modelado como um grafo não-direcionado, onde os vértices correspondem aos quarteirões e as arestas correspondem às ruas, e o peso de uma aresta corresponde à probabilidade de ocorrer um incêndio ao passar por essa rua.

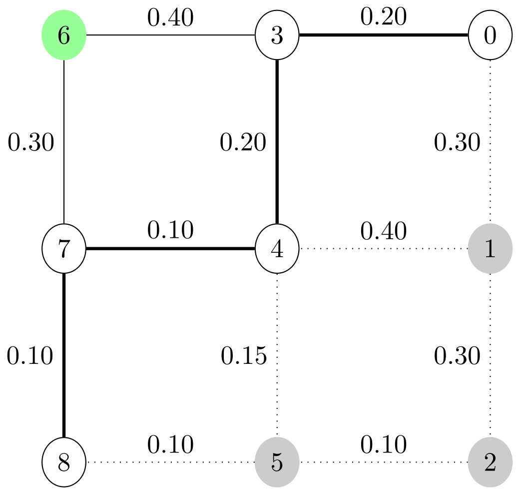

Figura 1: Exemplo de grafo. O vértice verde representa um quarteirão com um corpo de bombeiros. Os vértices cinza representam quarteirões que devem ser evitados durante o caminho, pois estão a mais que dois quarteirões de distância de um corpo de bombeiros. As arestas em negrito representam um caminho que minimiza a chance de incêndios.

Podemos dividir a solução em três passos: remover os vértices que devem ser evitados; encontrar um menor caminho; e finalmente resolver eventuais desempates. Na primeira etapa, devemos remover quaisquer vértices que estão mais distantes que K arestas de um corpo de bombeiros, onde K é um dos parâmetros de entrada. Para fazer isso, primeiro marcamos todos os vértices como inacessíveis. Depois, utilizamos o algoritmo de busca em largura a partir de cada quarteirão que tem um corpo de bombeiros, marcando todos os vértices que estão a menos de K níveis como acessíveis. Ao final desse processo, podemos remover todos os vértices que ainda estão marcados como inacessíveis.

Nesse ponto, o algoritmo de Dijkstra é utilizado para encontrar um menor caminho entre o vértice fonte e o vértice destino. No entanto, é possível existir mais que um menor caminho, e nesse caso é preciso escolher o caminho cuja sequência de vértices tem a menor ordem lexicográfica. Para facilitar esse processo de desempate, o algoritmo de Dijkstra é feito no sentido oposto, do destino para a fonte.

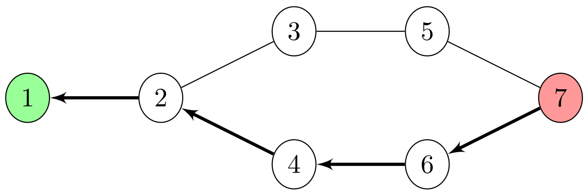

Figura 2: Exemplo de situação de empate. Todas as arestas têm peso igual. O caminho $1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 7$ foi escolhido, mas o caminho $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7$ tem chance de incêndio idêntica e ordem lexicográfica menor.

Finalmente, para resolver desempates, percorremos o menor caminho encontrado, da fonte para o destino, analisando todas as arestas na adjacência de cada vértice do caminho em busca de outro menor caminho com ordem lexicográfica menor. Quando um empate é encontrado, escolhemos o caminho de menor ordem. Caso não exista empate, o algoritmo termina ao chegar no vértice destino.

3 Análise teórica do custo assintótico

Nas análises a seguir, $\boldsymbol{V}$ é o conjunto de quarteirões na entrada e $\boldsymbol{E}$ é conjunto de ruas.

3.1 Análise teórica do custo assintótico de tempo

Podemos observar o programa de acordo com o seguinte pseudo-código de alto nível:

Algorithm 1: Pseudo-código

```
   Data: Vértices V, adjacências Adj, pesos W, corpos de bombeiro B, vértice fonte S, vértice destino D
   Result: Menor caminho entre S e D
1  for b ∈ B do
2  |  Busca-em-largura(V, Adj, b)
3  end
4  // Dijkstra no sentido contrário
5  caminho, distancia = Dijkstra(V, Adj, D)
6  // Desempate:
7  for i ← 0 to tamanho(caminho) − 1 do
8  |  u ⟵ caminho[i]
9  |  v ⟵ caminho[i + 1]
10 |  for w ∈ Adj[u] do
11 |  |  if w < v and distancia[w] + W[w, u] == distancia[u] then
12 |  |  |  caminho[i + 1] ⟵ w
13 |  |  |  para desempate
14 |  |  end
15 |  end
16 end
17 Print(distancia[S], caminho)
```

Nas linhas 1 a 3, executamos uma busca em largura a partir de cada quarteirão com corpo de bombeiros na cidade. A busca em largura é $O(V+E)$, e o número de quarteirões com corpo de bombeiros é $O(V)$. Temos então complexidade $O(V(V+E)) = O(V^2+VE) = O(V^3)$. Na linha 5, o algoritmo de Dijkstra utilizando uma heap binária tem complexidade $O(E \log V)$. Finalmente, o algoritmo de desempate nas linhas 7 a 16 percorre o caminho, analisando todas as arestas na adjacência de cada vértice, até encontrar um empate, e portanto tem complexidade $O(V+E)$. Temos a complexidade total do programa:

$$O(V^3 + E \log V + V + E) = O(V^3)$$

É importante ressaltar que esse pior caso não é comum: ele representa um grafo denso, onde o número de quarteirões com corpos de bombeiros é da ordem de V, e a distância máxima utilizada para remover quarteirões inacessíveis é maior que a distância máxima entre qualquer vértice e o vértice com corpo de bombeiros mais próximo.

3.2 Análise teórica do custo assintótico de espaço

Para cada grafo na entrada, é alocado o próprio grafo $O(V+E)$, uma fila para o algoritmo de busca em largura ($O(V)$), uma fila de prioridades para o algoritmo de Dijkstra ($O(V)$) e um arranjo para representar os menores caminhos encontrados ($O(V)$). O custo assintótico de espaço total é:

$$O(V + E + V + V + V) = O(V + E)$$

4 Análise de experimentos

Para realizar a análise experimental, foi implementado um gerador de grafos que gera problemas de acordo com a especificação, com número de vértices e arestas desejados. Para medir o tempo de execução do código, foi utilizada a API `mach_absolute_time`. Cada instância foi testada 3 vezes e o tempo de execução foi obtido retirando a média do tempo das 3 execuções. Os testes foram executados em uma máquina com processador Intel Core i7 2.6 GHz e 16GB de memória RAM.

Os gráficos a seguir comparam o tempo de execução do programa com diferentes métricas, e mostram que, apesar do pior caso de $O(V^3)$ obtido na seção 3.1, na prática o que domina o tempo de execução do programa é o algoritmo de Dijkstra, de custo $O(E \log V)$, já que raramente encontramos o pior caso onde todos os quarteirões têm corpos de bombeiros e a distância máxima é maior que o número de níveis a partir de qualquer vértice do grafo.

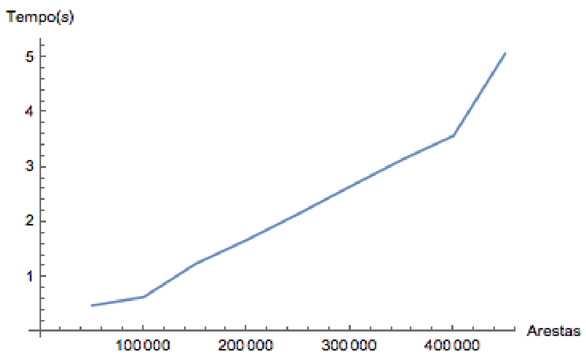

Figura 3: Tempo de execução com número de vértices fixado em 1000 e número de arestas crescendo de 50000 a 450000.

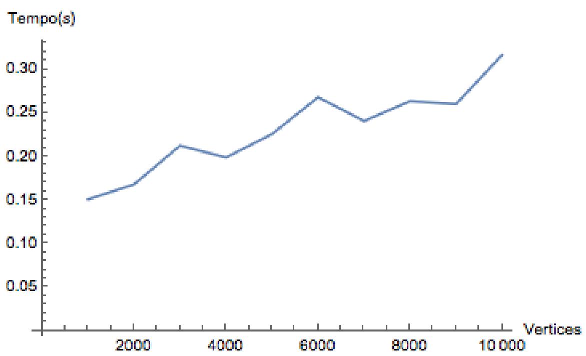

Figura 4: Tempo de execução com número de arestas fixado em 50000 e número de vértices crescendo de 1000 a 10000.

E de fato, ao observar o tempo de execução comparado com $E \log V$, podemos ver que esse fator aproxima com precisão o custo assintótico do programa:

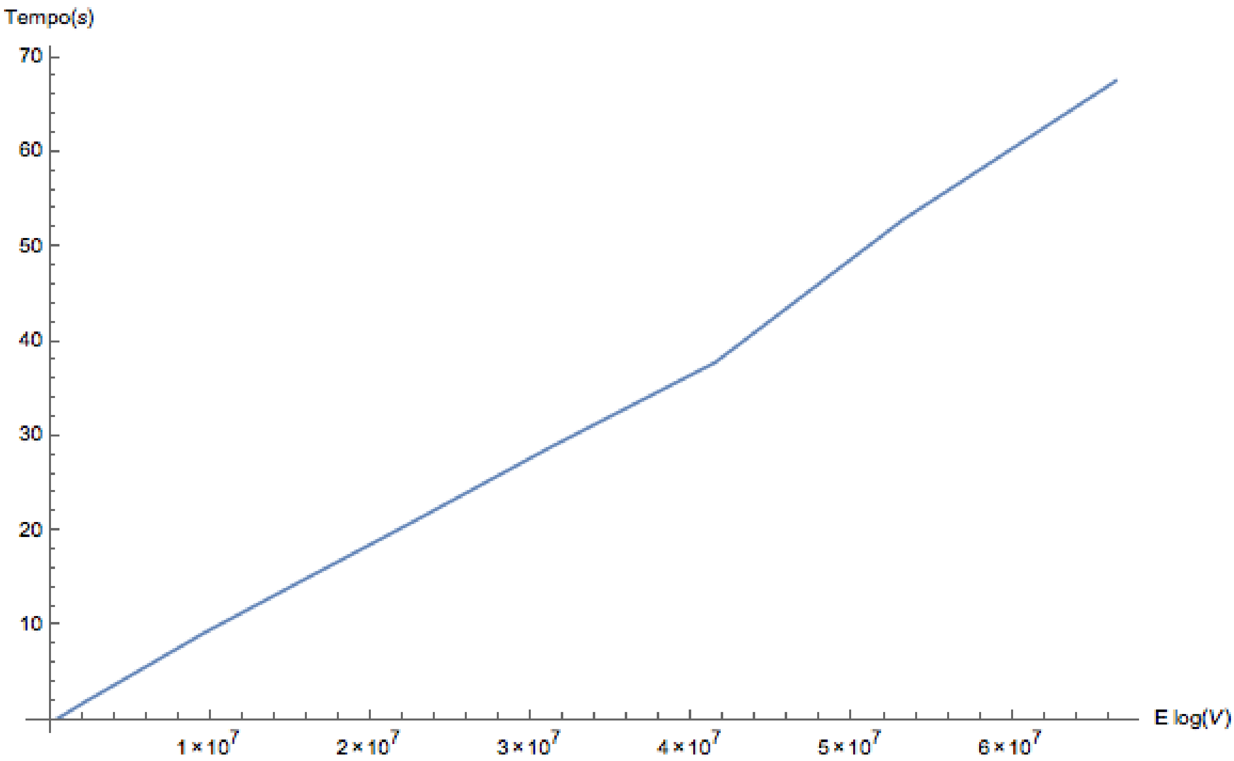

Figura 5: Tempo de execução com número de vértices crescendo de 1000 a 10000 e probabilidade de conexão igual a 5%. O número de arestas varia de 50000 a 5000000.