Trabalho Prático 1

Wilson Moreira Tavares

October 23, 2015

1 Introdução

O cenário do problema a ser resolvido por este trabalho prático é o seguinte:

Existe uma cidade conhecida por seu alto número de incêndios. Um bombeiro foi contratado para trabalhar nessa cidade. Ao ser contratado, o bombeiro recebeu um mapa com as características estratégicas da cidade, contendo no mesmo todos os quarteirões, localização de todos os corpos de bombeiros na cidade e a probabilidade de acontecer um incêndio em cada uma das ruas da cidade.

Só que este é um bombeiro preguiçoso que usará essas informações para evitar ao máximo ter que atender um chamado de incêndio. Se por acaso acontecer de ele passar por um local onde esteja acontecendo um incêndio, ele deve estar a uma distância máxima de k quarteirões de algum corpo de bombeiros, pois nessa circunstância um outro bombeiro poderá atender àquele chamado em seu lugar.

Com as características acima, foi proposta a implementação de um algoritmo que, dado um quarteirão inicial e um final, trace uma rota entre esses dois quarteirões. Essa rota deve assegurar que o bombeiro não passará em nenhum quarteirão que não possua suporte de uma base de bombeiros e que possua a menor probabilidade de haver incêndio. Sendo que em caso de empate, deve ser considerado o caminho de menor ordem lexicográfica. A modelagem e a solução serão alcançadas através de grafos.

2 Solução do problema

Para representar o grafo foi escolhida a forma de lista de adjacências, onde foi utilizada a seguinte estrutura:

O algoritmo lê quantas serão as iterações e entra num laço do tamanho das iterações que foram previamente definidas.

Dentro desse laço um vetor de listas (a representação do grafo) é alocado e populado de acordo com as diretrizes do programa, isso é feito com a ajuda das funções $alocaLista$, que aloca o número de posições necessária no vetor e $insereRuas$, que pega os dados no formato da entrada e chama a função $insere$, que insere efetivamente as ruas como arestas no grafo.

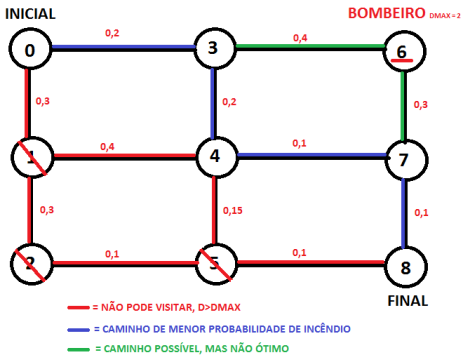

Figure 1: A algoritmo deve descobrir qual é o caminho ótimo.

```
estrutura cabeca{
   inteiro tamanho;
   inteiro elegivel;
   apontador nos primeiro;
}

estrutura nos{
   inteiro destino;
   real probabilidadeIncendio;
   apontador nos proximo;
}
```

Figure 2: Os vértices representam os quarteirões e as arestas representam as ruas, cada quarteirão é representado pelo tipo cabeca, que se liga a outros (nos).

Após inserir todas as arestas no grafo, a entrada fornece quais são os quarteirões onde existem corporações. A função que se encarrega de contabilizar isso é a *insereBombeiros*, que escaneia todas as entradas que informam os quarteirões que possuem corpo de bombeiros, chamando posteriormente a função $defineRotas$ a cada inserção de um corpo de bombeiros.

Como o bombeiro não pode passar por um quarteirão onde a distância para uma corporação é maior do que k, é necessário fazer uma poda no grafo para eliminarmos essa possibilidade. Essa poda consiste em eliminar todos os quarteirões que excedem a distância máxima k permitida até um bombeiro. Essa poda é feita através da função $defineRotas$, que é uma adaptação do algoritmo de Dijkstra.

A função *insereBombeiros* marca todos os quarteirões como inelegíveis (inalcançáveis), pois até então não há nenhum bombeiro. Em cada chamada da função $defineRotas$ é conhecida a menor distância de cada vértice para o bombeiro em questão, se a distância for menor ou igual à permitida, o vértice

é marcado como elegível.

No fim, temos marcados como elegíveis todos os vértices que podem ser alcançados, pois obedecem à menor distãncia. Essa característica é necessária, tendo em vista que, para realizar o cálculo de menor probabilidade de incêndio entre dois quarteirões é necessário saber em quais quarteirões é possível passar.

```
Algoritmo defineRotas
=====================================================================
Início
    Faça distância[inicial] = 0
    Enquanto 1 faça
        menorValor = -1;
        Para todo vértice faça
            Se vértice não visitado e (distancia[vértice] menor
            ou menorValor=-1)
                menorValor = distancia[vértice]
            Fim Se
        Fim Para
        Se menorValor=-1 faça
            Fim 'Enquanto'
        Fim Se
        Senão
            Vértice de menorValor é visitado
            Para todo vizinho de vértice de menorValor faça
                Se distancia[menorValor]=-1 ou
                menor que distancia[vizinho]
                    distancia[menorValor] = distancia[vizinho]+1
            Fim Para
        Fim Senão
    Continua Enquanto
    Para toda distância faça
        Se distancia[i] menor ou igual que distanciaMax faça
            quarteirao i é elegível
        Fim Se
    Fim Para
Fim
=====================================================================
```

A função descrita acima termina, e logo após a função que calcula o caminho de menor probabilidade de ocorrer um incêndio é chamada.

A função que calcula a menor probabilidade de incêndio é o algoritmo de Dijkstra, com algumas adaptações. Essas adaptações dizem respeito ao cálculo da probabilidade de não haver incêndio no local. A função calcula o caminho com a maior probabilidade de não ocorrer incêndio, havendo algumas mudanças simples na comparação, sendo a que os caminhos não são somados, e sim multiplicados.

A função calcula a menor probabilidade de encontrar um incêndio entre os vértices que ficaram após a poda, é verificado também se o vértice final definido na entrada é alcançável, considerando a ordem lexicográfica em caso de empate. Se o vértice final for alcançavel, o algoritmo imprime a propabilidade de haver incêndio juntamente com o caminho a ser percorrido para alcançar essa probabilidade mínima, caso contrário o algoritmo imprime -1.

```
Algoritmo maiorProbNaoIncendio
=====================================================================
Início
    Faça probNIncêndio[inicial] = 1
    Enquanto 1 faça
        maiorProb = 0;
        Para todo vértice faça
            Se vértice não visitado e probNIncêndio[vértice] maior
            e vértice elegível
                maiorProb = probNIncêndio[vértice]
            Fim Se
        Fim Para
        Se maiorProb=0 faça
            Fim 'Enquanto'
        Fim Se
        Senão
            Vértice de maiorProb é visitado
            Para todo vizinho de vértice de maiorProb elegível faça
                Se probNIncêndio[vizinho]=-1 ou menor que
                probNIncêndio[vizinho]*probNIncêndio[vértice maiorProb]
                    probNIncêndio[vizinho] = probNIncêndio[vizinho] *
                    probNIncêndio[vértice maiorProb]
                Fim Se
                Senão Se probNIncêndio[vizinho] igual a
                probNIncêndio[vizinho]*probNIncêndio[vértice maiorProb]
                    Se vizinho menor que vértice demaiorProb
                        probNIncêndio[vizinho] = probNIncêndio[vizinho]
                        * probNIncêndio[vértice maiorProb]
                    Fim Se
                Fim Se
            Fim Para
        Fim Senão
    Continua Enquanto

    Imprime 1-probNIncêndio[final]
    Imprime caminho inicial-final
Fim
=====================================================================
```

3 Análise teórica de custo assintótico

3.1 Espaço

- A função *alocaLista* aloca a lista de adjacências de acordo com seu tamanho na entrada, sendo portanto de tamanho TxC[1].
- A função *insereRuas* insere uma rua na lista de adjacências, ou seja, aloca dois itens na lista, tendo em vista que as ruas são de mão dupla. Portando possui complexidade de 2R[2].
- A função *insereBombeiros* possui complexidade T, já que aloca um vetor de tamanho T que armazena as menores distâncias de todos os quarteirões para um bombeiro.
- A função *insereBombeiros* chama a função *imprimeRotas*, que por sua vez também cria dois vetores de tamanho T, tendo complexidade de espaço 2T.
- A função *imprimeRotas* define se há ou não uma rota com suporte de bombeiros que ligue os quarteirões solicitados. Se houver tal rota, é chamada a função *maiorProbNaoIncendio*, que por sua vez ocupa 4T de espaço, pois se utiliza de quatro vetores de tamanho T.

  Dados os itens acima, a complexidade de espaço do algoritmo completo é:

$$\theta(TxC + 2R + T + 2T + 4T)$$

3.2 Tempo

- Do ponto de vista de tempo, as funções *alocaLista* e *insereRuas* são lineares, tendo em vista que alocam uma lista do tamanho T da entrada e inserem um número de vértices do tamanho R da entrada.
- A função *insereBombeiros* é a função que exerce maior impacto na complexidade do algoritmo, tendo em vista que, para cada corporação presente no mapa há uma execução da função $defineRotas$, que é uma adaptação do algoritmo de Dijkstra, ou seja, a cada inserção de um bombeiro, uma nova chamada é feita. A função $defineRotas$ possui complexidade da ordem $O((T+R)logT)$. Portanto, a função *insereBombeiros* possui complexidade $O(Bx(T+R)logT)$[3].
- A função *maiorProbNaoIncendio* é chamada apenas uma vez, portanto, não influencia na execução do algoritmo de acordo com o crescimento do número do corpo de bombeiros.

[1] Temos que o tamanho da entrada é T e o espaço ocupado por cada célula cabeça da lista de adjacências é C.

[2] Consideramos que uma ligação lista de adjacências possua tamanho R

[3] Onde B é o número de corporações de bombeiros existentes.

Dados os itens acima, a complexidade de tempo do algoritmo é de:

$$O(Bx(T+R)logT)$$

4 Execução e análise de experimentos

4.1 Diretrizes

Para a realização da entrada, foram criados vários tipos de casos de testes, todos começando de tamanhos pequenos para tamanhos maiores. O que diferencia cada tipo, é que cada um apresenta uma característica de crescimento específica. Cada grupo será caracterizado e posteriormente terá os resultados apresentados.

4.2 Testes

- A entrada 1 possui como caracteŕistica o crescimento de quarteirões na cidade, ou seja, a cada entrada, somente o número de quarteirões foi acrescido, com o número de ruas e de bombeiros permanecendo estático, os resultados obtidos foram:

  1. 100 quarteirões: 0,015s
  2. 200 quarteirões: 0,038s
  3. 400 quarteirões: 0,04s
  4. 800 quarteirões: 0,038s
  5. 1600 quarteirões: 0,014s
  6. 3200 quarteirões: 0,042s
  7. 6400 quarteirões: 0,082s
  8. 12800 quarteirões: 0,074s
  9. 25600 quarteirões: 0,094s
  10. 51200 quarteirões: 0,115s
  11. 102400 quarteirões: 0,262s
  12. 204800 quarteirões: 0,453s
  13. 409600 quarteirões: 1,483s
  14. 819200 quarteirões: 1,85s
  15. 1638400 quarteirões: 3,634s

- A entrada 2 possui como caracteŕistica o crescimento de ruas na cidade, ou seja, a cada entrada houve acréscimo apenas no número de ruas, sendo o número de quarteirões e de bombeiros estático, os resultados obtidos foram:

  1. 100 ruas: 0,219s

2. 200 ruas: 0,221s
3. 400 ruas: 0,243s
4. 800 ruas: 0,214s
5. 1600 ruas: 0,217s
6. 3200 ruas: 0,212s
7. 6400 ruas: 0,226s
8. 12800 ruas: 0,214s
9. 25600 ruas: 0,224s
10. 51200 ruas: 0,218s
11. 102400 ruas: 0,224s
12. 204800 ruas: 0,232s
13. 409600 ruas: 0,25s
14. 819200 ruas: 0,306s
15. 1638400 ruas: 0,408s

- A entrada 3 possui como caracteŕistica o crescimento de bombeiros na cidade, ou seja, a cada entrada houve acréscimo apenas no número de bombeiros, sendo o número de ruas e quarteirões constante, os resultados obtidos foram:

1. 1 bombeiro: 0,747s
2. 2 bombeiros: 0,957s
3. 4 bombeiros: 1,235s
4. 8 bombeiros: 5,313s
5. 16 bombeiros: 3,892s
6. 32 bombeiros: 5,62s
7. 64 bombeiros: 22,66s
8. 128 bombeiros: 36,989s
9. 256 bombeiros: 89,054s
10. 512 bombeiros: 148,271s
11. 1024 bombeiros: 397,585s
12. 2048 bombeiros: 992,315s
13. 4096 bombeiros: 1281,186s
14. 8192 bombeiros: 3327,93s
15. 16384 bombeiros: 10028,833s

- A entrada 4 possui como caracteŕistica o crescimento de todos os itens (número de quarteirões, ruas e bombeiros). Esse tipo de entrada foi escolhido tendo como objetivo ver como o algoritmo se comporta quando todos os parâmetros crescem, fornecendo uma visão "final" do ponto de vista de complexidade. Os resultados obtidos foram:

  1. 10 quarteirões, 10 ruas e 1 bombeiro:0,003s
  2. 20 quarteirões, 20 ruas e 2 bombeiros: 0,004s
  3. 40 quarteirões, 40 ruas e 4 bombeiros: 0,005s
  4. 80 quarteirões, 80 ruas e 8 bombeiros: 0,011s
  5. 160 quarteirões, 160 ruas e 16 bombeiros: 0,033s
  6. 320 quarteirões, 320 ruas e 32 bombeiros: 0,192s
  7. 640 quarteirões, 640 ruas e 64 bombeiros: 1,461s
  8. 1280 quarteirões, 1280 ruas e 128 bombeiros: 11,764s
  9. 2560 quarteirões, 2560 ruas e 256 bombeiros: 92,464s
  10. 5120 quarteirões, 5120 ruas e 512 bombeiros: 768,85s
  11. 10240 quarteirões, 10240 ruas e 1024 bombeiros: 7530,547s

4.3 Gráficos

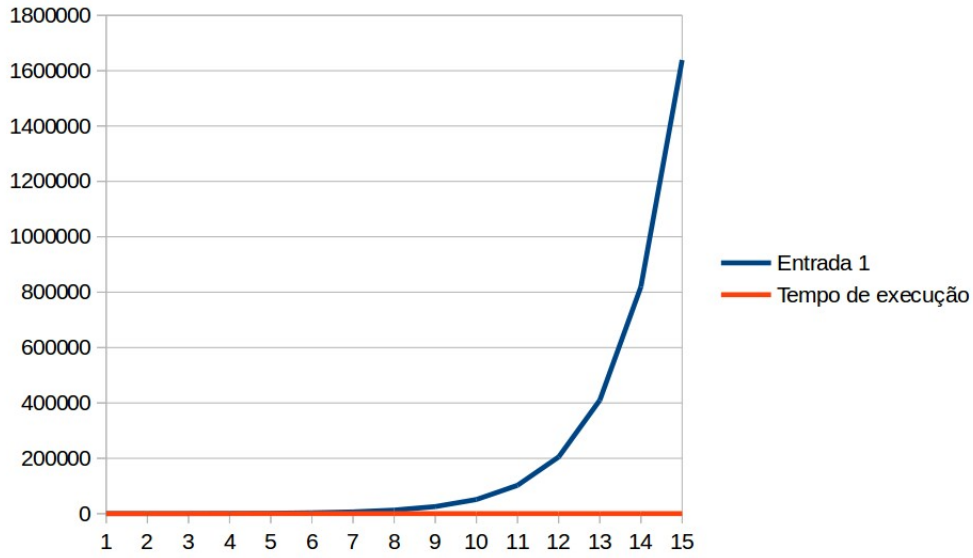

Figure 3: Testes da entrada 1, não houve impacto perceptível do crescimento do número de quarteirões no tempo.

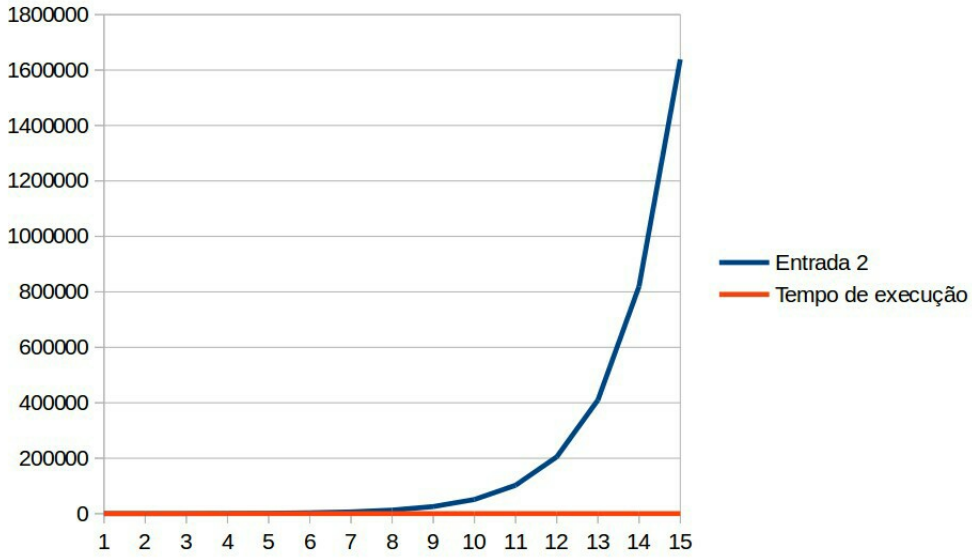

Figure 4: Testes da entrada 2, também não houveram impactos perceptíveis do crescimento do número de ruas no tempo.

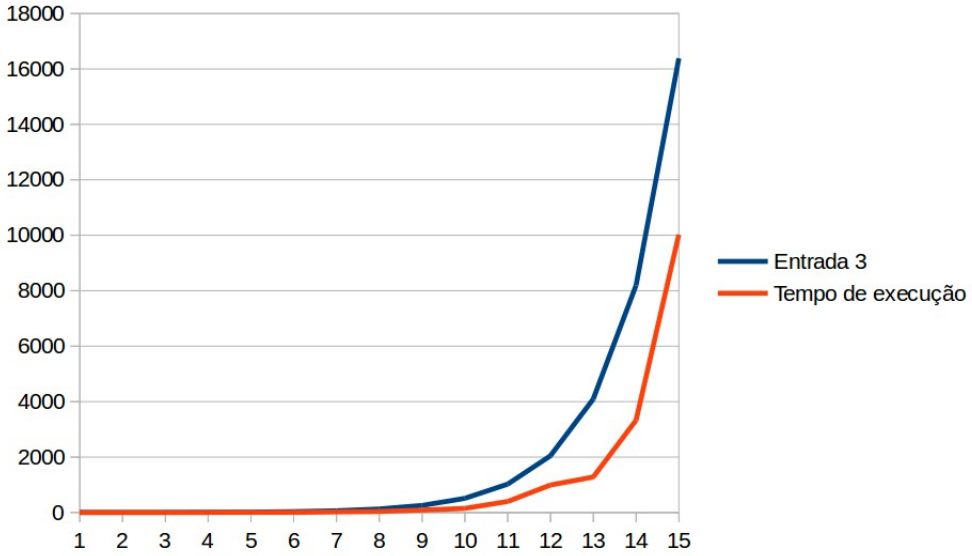

Figure 5: Testes da entrada 3, nesse caso houveram impactos perceptíveis, tendo em vista que o tamanho dessa função reflete automaticamente no número de execuções do Dikstra.

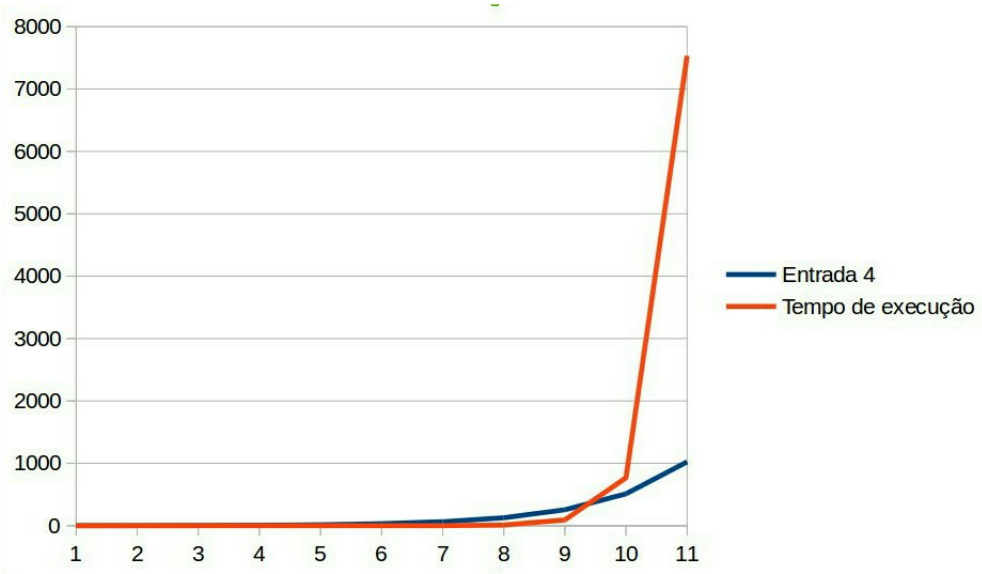

Figure 6: Testes da entrada 4, todos os parâmetros crescendo.

4.4 Conclusão

Tendo em vista os variados testes executados, é possível perceber que com o crescimento de ruas o tempo de execução cresce em ritmo lento e com o crescimento do número de quarteirões o tempo de execução fica praticamente inalterado. Foi perceptível também que quando os testes foram feitos usando o crescimento de somente um dos atributos o único que apresentou crescimento expressivo foi o de número de bombeiros. Somente tomando esses dados por si só, é possível se enganar e afirmar que somente pelo número de bombeiros é definida a complexidade, quando na verdade o crescimento é definido pelo aumento de todas as funções, sendo que a equação $O(Bx(T+R)logT)$ satisfaz os resultados que foram alcançados através da análise experimental.