Trabalho Prático 1: The city is on fire!

Gabriel Henrique Souto Pires {gabrielpires@ufmg.br}

1 Introdução

Neste problema são dados um conjunto de vértices que representam os quarteirões de uma cidade e as arestas que representam as ruas que ligam os quarteirões. Você é um bombeiro preguiçoso e gostaria de trabalhar o mínimo possível, então ao sair de um corpo de bombeiros para outro seria ideal evitar passar pelos caminhos onde a probabilidade $P(u,v)$ de ter um incêndio é alta entre os quarterões u e v, sendo que $P(u,v) = P(v,u)$ e todas as ruas são de mão dupla. Durante o trajeto, também é necessário ficar a uma distância de no máximo k quarteirões de algum corpo de bombeiros, dessa forma outros bombeiros podem vir apagar o incêndio no seu lugar.

A tarefa neste TP é descobrir o caminho que respeita as restrições descritas acima, ou seja, um caminho que passe por quarteirões que tenham a menor probabilidade de incêndio possível tal que a probabilidade total de incêndio entre o quarteirão de saída e o de chegada seja mínima e o caminho passe sempre por vértices que estejam a uma distância máxima k de algum corpo de bombeiros.

2 Solução do Problema

A probabilidade $P(u,v)$ de ter um incêndio em dado trecho pode ser interpretada como o peso das arestas entre os vértices do grafo que são os quarteirões da cidade. Desta forma, o caminho mais curto é aquele em que o peso total das arestas é mínimo, ou seja, a probabilidade de incêndio é menor. Para resolver o problema, foi criado uma lista[1] de adjacência para representar o grafo (os vértices e as arestas que ligam os vértices adjacentes à eles). A lista de adjacência é basicamente um vetor com uma posição para cada vértice, cada posição do vetor contém uma lista encadeada onde são inseridos os vértices adjacentes ao vértice referente à posição atual do vetor.

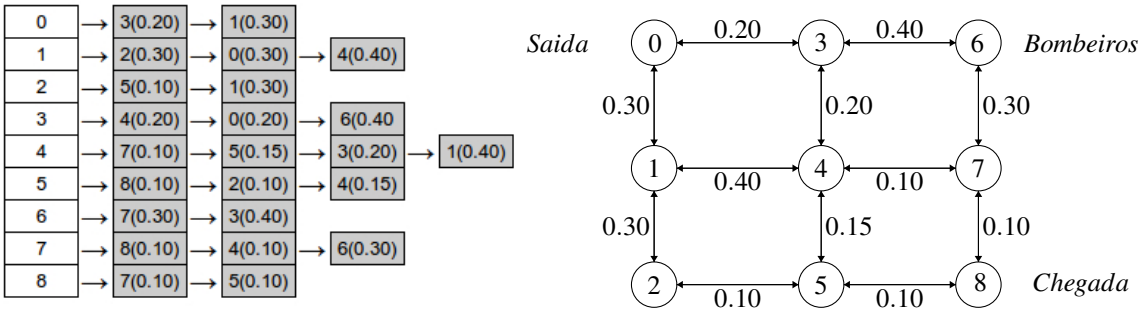

Figure 1: Representação do grafo (direita) em forma de lista de adjacências (esquerda)

Uma vez que os vértices que estão a uma distância k maior que a máxima permitida não podem ser percorridos, o menor caminho nunca passará por eles, então esses vértices devem ser desconsiderados na hora de

[1] Os TADs Lista, Fila e Heap utilizados neste TP foram feitos usando como referência o código desenvolvido pelos professores Thiago Noronha e William Schwartz durante a disciplina de AEDS 2 no semestre passado. Os aquivos podem ser baixados em `http://homepages.dcc.ufmg.br/~gabrielpires/disc/aeds3/tads_referencia/`

se procurar a solução no grafo. Isso foi feito com uma *busca em largura* onde para cada vértice com corpo de bombeiros uma busca é feita e apenas os vértices a uma distância máxima k do vértice de origem são marcados como válidos em um vetor. Dessa forma, ao se rodar o algoritmo que acha o caminho mínimo, os vértices inválidos não são nem testados, o que torna o algoritmo mais rápido e evita que caminhos mínimos que passam por vértices inválidos sejam considerados no final da execução. Como visto no pseudo código abaixo, uma fila é utilizada

Algoritmo 1 Pseudo código do loop principal da busca em largura

```
while(!filaVazia){
    if(distPai < k){
        for(Percorre a lista(vertice[i]) até o final da lista){
            enfileira(nodoAtual, distPai+1);
            vertsValidos[nodoAtual] = 1;
        }
    }
    desenfileira(fila);
    i = frenteFila(fila);
}
```

Com os vértices válidos marcados no vetor, o grafo foi submetido ao algoritmo de *Dijkstra* que acha o menor caminho entre dois vértices em um grafo onde as arestas não tem peso negativo e como as arestas no problema em questão não terão valor negativo uma vez que a probabilidade de incêndio varia de 0 a 1, isso não será um problema.

O *Dijsktra* recebe como parâmetro o grafo em forma de lista de adjacência, o vértice de origem, o número de quarteirões, o vetor de vértices válidos gerado pela busca em largura e o número do vértice de chegada. Na função os vértices válidos são percorridos a fim de se encontrar o melhor caminho entre o vértice de origem e o de chegada. Nessa função, cada vez que se chega até um vértice todas as suas arestas são inseridas em um *Heap* na forma de P_N, ou melhor, $P(s,u) * (1 - P(u,v))$, sendo que s é o vértice de saída, u é o vértice atual e v é o próximo vértice a ser percorrido. Esta probabilidade representa a chance de não haver incêndio no caminho até a origem. Desta forma é possível saber qual aresta percorrer em *O(1)* já que basta apenas retirar a maior aresta do Heap (Max-Heap) e seguir a partir dela repetindo o processo até que se ache a solução. Cada vez que uma aresta é inserida, o vértice em que ela chega recebe como antecessor o vértice de saída da aresta. Essa operação é útil na hora de se identificar o caminho que foi feito pelo algoritmo, já que basta fazer o caminho de volta nesse vetor de antecessores para saber quais vértices foram percorridos.

Durante a execução do algoritmo, a maior probabilidade de não haver incêndio de um vértice até o vértice de saída é armazenada na posição de um vetor de probabilidades referente a cada vértice. Ao final da execução, é exibida na tela $1 - prob[c]$, que representa a probabilidade de incêndio no melhor caminho possível até o vértice de chegada, assim como o caminho, obtido através do vetor de antecessores.

Caso existam dois caminhos com a mesma probabilidade final, o caminho escolhido deve ser aquele que respeita a ordem lexicográfica, ou seja, se o caminho pode passar tanto pelos vértices $1-3-4$ quanto pelos vértices $1-2-4$, o caminho escolhido será $1-2-4$ já que 2 é menor que 3. Esse problema foi solucionado ao se inserir os vértices na fila por ordem de número de vértice, ou seja, os vértices com menor número estarão sempre na frente da fila e serão testados no Dijkstra primeiro, dando preferência para os caminhos de menor ordem lexicográfica.

Se não for possível respeitas as restrições impostas na especificação ao invés de imprimir o caminho, é exibido na tela o número -1.

3 Análise Teórica do Custo Assintótico

Nesta seção serão apresentadas as análises de custo assintótico de tempo e de espaço.

3.1 Análise Teórica do Custo Assintótico de Tempo

A análise de custo assintótico de tempo pode ser feita ao se analisar os 2 algoritmos principais usados:

- O primeiro algoritmo executa uma busca em largura usando uma Fila para enfileirar as arestas. Inicialmente uma fila vazia é criada e o vértice um vértice com corpo de bombeiros é enfileirado. A partir desse vértice, todos os seus adjacentes que estão a uma distância máxima k dele são enfileirados e marcados como visitados, e então o mesmo é feito para cada vértice enfileirado. Dessa forma, todos os vértices válidos são conhecidos. Considerando que o grafo do problema foi representado em forma de lista de adjacência, o pior caso é aquele em que todos os vértices e arestas são explorados pelo algoritmo. A complexidade de tempo pode ser representada pela expressão $O(|E| * |V|)$ já que a busca é chamada para cada vértice com corpo de bombeiros, onde $|E|$ é o tempo total gasto pelas operações sobre todas as arestas do grafo onde cada operação requer um tempo constante $O(1)$ sobre uma aresta mas no pior caso $|E|$ pode ser $|V^2|$, e $|V|$ que significa o número de operações sobre todos os vértices que possui uma complexidade constante $O(1)$ para cada vértice uma vez que todo vértice é enfileirado e desenfileirado uma única vez. Então no pior caso, da Busca em Largura seria $O(|V^2| * |V|)$.

- O segundo algoritmo é o algoritmo de *Dijkstra*. A primeira coisa que é feita é o teste se o vértice de chegada é um vértice válido, se não for, então é exibido -1 já que o caminho não pode ser feito. Após isso, um *Heap* vazio é criado em $O(1)$. São criados os vetores auxiliares que guardam os antecessores e as probabilidades e seus valores iniciais são setados em $O(|V|)$. No loop principal do *Dijsktra* é testado se os vértices percorridos são válidos e se já foram visitados ou não, caso eles não tenham sido visitados eles são inseridos no *Heap*, seu antecessor é setado como o vértice anterior e sua probabilidade é guardada no vetor de probabilidades. Essa parte roda enquanto o *Heap* não estiver vazio, e como no pior caso o heap tera V elementos, essa parte roda em $O(|V|)$. Dentro desse loop, um outro loop roda para percorrer a lista e inserir as arestas no *heap,* cada vez que o algoritmo acha uma aresta válida e insere ela no heap é chamada a função *refazBaixoCima()* que roda em $O(logV)$ para manter a condição de heap no vetor, assim como quando um valor de probabilidade é atualizado. Então o loop principal do *Dijkstra* roda em $O(V^2 logV)$.

Podemos concluir então que o custo total de tempo do algoritmo é $O(|V^2| * |V|) + O(V^2 logV) = O(V^3)$

3.2 Análise Teórica do Custo Assintótico de Espaço

Para analisar o custo de espaço do programa foi levado em consideração a quantidade de memória utilizada pelos dois principais algoritmos, o *Dijkstra* e a *Busca em Largura*. Na busca em largura, quando o número de vértices no grafo é conhecido e supondo-se a representação deste em listas de adjacência, a complexidade de espaço do algoritmo pode ser representada por $O(|V|)$ onde $|V|$ representa o número total de vértices no grafo. No *Dijkstra* existe um *Heap* que é composto de um vetor em que cada posição tem 2 atributos, um *int* e um *double*, ou seja 4+8 bytes. Como o tamanho do *Heap* varia de acordo com o número de vértices, então a complexidade de espaço do *Heap* pode ser representada como $O(|V|)$ assim como todos os outros vetores que são alocados com tamanho baseado em $|V|$.

Pode-se concluir que a complexidade de espaço do algoritmo é proporcional à soma dessas complexidades, e pode ser representada como $O(|V| + |V|) = O(|V|)$.

4 Análise Experimental do Custo Assintótico

Para fazer a análise experimental do programa, foi criado um outro programa que gera arquivos de entrada compatíveis de tamanhos variados de forma que a diferença do tempo de execução com entradas diferentes possa ser medido. Para não levar em consideração o tempo que se leva para digitar a entrada no tempo de execução do programa, a entrada foi lida diretamente dos arquivos. Para se medir o tempo de execução do programa foi usado o comando *time* que ao final da execução do programa grava na saída padrão (*stdout*) estatísticas de tempo sobre o programa executado.

Os testes foram realizados em uma máquina virtual rodando Xubuntu 15.04. O computador utilizado tem um processador AMD Phenom II X4 965 3.40GHz e 8GB de memória, porém, a máquina virtual utiliza apenas um núcleo do processador e 2GB de RAM.

Foram criados arquivos de teste com 1 caso de teste cada, variando o número de vértices de 100 até 10^3, e o número de arestas variando de 4851 até 49850, e em todos os casos a busca em largura tem o pior caso que é quando todos os vértices do grafo tem corpo de bombeiros e também no Dijkstra que é testado sempre no caso em que todos os vértices podem ser visitados. O gráfico abaixo ilustra a variação de tempo em segundos com a entrada que está representada como "(vértices / arestas)".

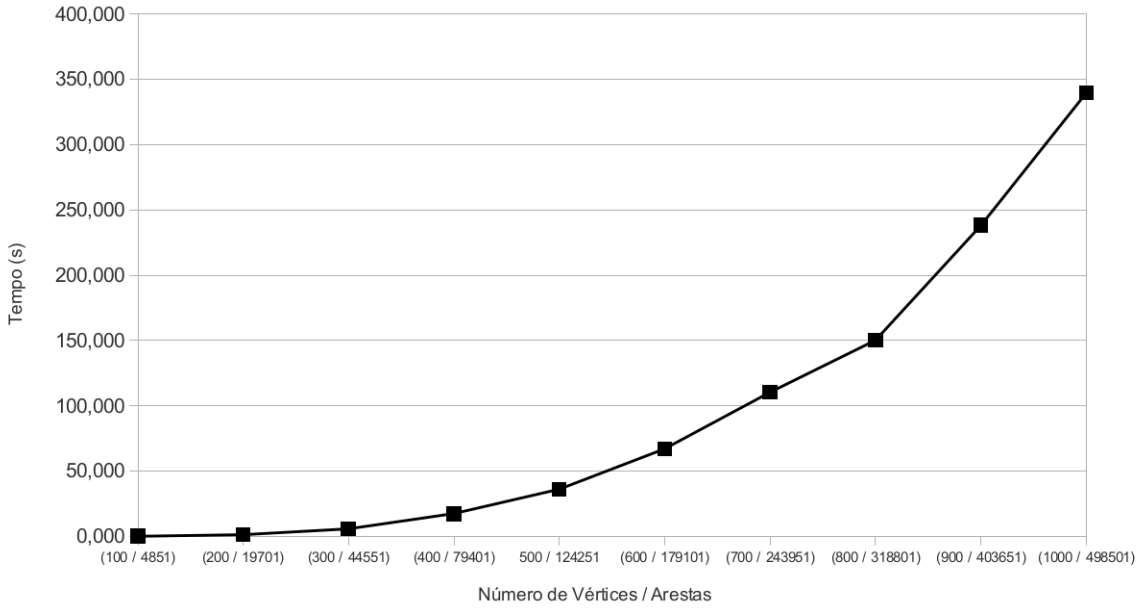

Figure 2: Gráfico representando a complexidade do programa

5 Conclusão

O problema proposto neste trabalho foi o do Bombeiro Preguiçoso que foi resolvido transformando a entrada do problema em um grafo e submetendo-o aos algoritmos de Busca em Largura e Dijkstra. Foram utilizados também os TADs Lista, para representar o grafo, Fila para auxiliar no Dijkstra e Heap para agilizar a escolha das arestas no Dijkstra. Os TADs foram desenvolvidos tomando como referência os códigos cedidos pelos professores Thiago Noronha e William Schwartz.