Universidade Federal de Minas Gerais

Instituto de Ciências Exatas / Departamento de Ciência da Computação

Disciplina: Algoritmos e Estruturas de Dados III

Professores: Wagner Meira Junior, Marcos Augusto Menezes Vieira

Aluno: Caíque Bruno Fortunato Matrícula: 2013062731

Documentação do Trabalho Prático 1 – The city is on fire!

1. Introdução

O objetivo desse trabalho prático é descobrir qual é o caminho onde há a menor probabilidade de ocorrer um incêndio em um mapa de uma cidade que contém ruas, quarteirões e a localização dos corpos de bombeiros presentes na cidade. No entanto, algumas restrições devem ser obedecidas, entre elas manter uma distância de k quarteirões de algum corpo de bombeiro durante o trajeto e também a ordem lexicográfica. O problema deve ser modelado por grafos e implementado em linguaem C.

Após alguns estudos e avaliações, um método eficiente para a implementação do trabalho seria através do uso de listas de adjacências para representar o grafo, composta por um vetor de structs para representar cada vértice e suas informações, entre elas a lista de vértices adjacentes, por exemplo.

Através da implementação do grafo, serão realizadas funções importantes e cruciais para a solução do trabalho, entre elas a busca em largura para tratar a distância de k quarteirões de algum corpo de bombeiro e um algoritmo de caminho mais curto, denominado Dijkstra, com as modificações necessárias para o desenvolvimento e funcionamento do algoritmo.

Com a solução do problema será possível praticar conceitos de programação aprendidos em Algoritmos e Estruturas de Dados I e II, como Listas Encadeadas, Filas, Structs, ponteiros e diversas outras ferramentas presentes nas bibliotecas da linguagem C. Além disso, também será praticado o módulo de Grafo, matéria lecionada no curso de Algoritmos e Estruturas de Dados III.

Além disso, o algoritmo e utiliza recursos importantes e aplicáveis no dia-a-dia, o próprio enunciado do trabalho prático demonstra uma possível aplicação. Dijkstra, por exemplo, nos permite deslocar entre dois pontos através de uma trajetória de menor caminho e, juntamente com o algoritmo de busca em largura, podemos utilizar o menor caminho sem passar em lugares/pontos específicos e/ou indesejados. Tendo assim, inúmeras aplicações possivelmente úteis e aplicáveis no dia-a-dia.

2. Modelagem do problema

A entrada do programa é composta pela quantidade de testes a serem realizados, cada teste contêm o número de quarteirões, de ruas, o ponto de saída e de chegada, a distância máxima que você deve estar de um corpo de bombeiros e a quantidade de corpos de bombeiros presentes na cidade. Por último é fornecido o trecho entre dois querteirões e a probabilidade de ocorrer um incêndio no trecho.

Sendo assim, o problema foi modelado através de uma estrutura de dados chamada Grafo, que é composta por um vetor de structs do tipo vértice que contém importantes informações a serem armazenadas e utilizadas no decorrer da solução do programa, sendo elas: um ponteiro do tipo Lista para armazenar os vértices adjacentes a um vértice (v) e o seus respectivos pesos em uma lista encadeada, a probablilidade a ser calculada, a distância a um corpo de bombeiros, um estado do tipo char a ser utilizado na função Dijkstra e o vértice anterior, que irá auxiliar na solução.

Após o preenchimento e construção do grafo dada as informações de entrada, é lido as posições onde têm corpos de bombeiros no grafo, para cada corpo de bombeiro é zerada a sua distância a umcorpo de bombeiro e realizada uma busca em largura iniciando do local onde possui corpo de bombeiro.

A busca em largura utiliza uma estrutura de dados chamado Fila, que recebe vértices em sua inserção, mas na retirada apenas o primeiro é retornado, logo o primeiro que é armazenado é o primeiro a sair. Desse modo, a busca em largura começa pelo vértice raiz e explora todos os vértices adjacentes, sendo que para cada véritce vizinho é explorado todos os seus vizinhos e assim sucessivamente. Além disso, sempre é atualizada a distância “tem_bombeiro” (armazenada na struct vértice) de acordo com a distância do vértice a um corpo de bombeiro.

Após a execução da busca em largura para a quantidade de corpos de bombeiros presentes na cidade teremos agora, no grafo, todas as distâncias dos vértices aos corpos de bombeiros. Como exemplo, onde tem corpo de bombeiros terá distância 0, seus vértives adjacêntes 1, e assim por diante. Assim é resolvido uma das restrições: que o caminho de menor probabilidade deve estar a uma distância “k” de algum corpo de bombeiro. Assim, resta agora somente realizar o algoritmo para encontrar o caminho onde há a menor probabilidade de ocorrer um incêndio em um mapa de uma cidade.

Para encontrar o menor caminho, foi escolhido o algoritmo Dijkstra, já que o grafo é não direcionado e principalmente por conter arestas com pesos não negativos. O algoritmo, que é guloso, utiliza uma ideia bastante parecida com a função de busca em largura, calculando o custo mínimo entre os vértices de um grafo fornecendo assim o caminho mais curto, que é exatamente o que é buscado. Assim, dado um ponto inicial, ele é inserido na fila e seus vérices adjacentes são analisados, o cálculo da probabilidade é realizado para que possamos saber se compensa caminhar

pelo caminho, ou seja, se ele é o mais curto, que irá acontecer se a conta for menor que a probabilidade presente no grafo na posição do vértice adjacente. Se o vértice ainda não estiver sido inserido na fila, ele é inserido e marcado como insertido na variável estado declarado na struc vértice. Finalmente, a conta é atribuída a probabilidade do grafo no véritce adjacente que também recebe o anterior como o vértice que estamos visitando e analisando seus adjacentes.

Mas, se a conta da probabilidade for igual a probabilidade presente no grafo na posição do vértice adjacente deverá ser realizada a análise da ordem lexicográfica considerando o melhor caminho pelo menor vértice.

Como antes de iniciar todo o trabalho com o algoritmo de Dijkstra os anteriores foram marcados como -1, caso a anterior do ponto de chegada for -1, então não foi possível resolver o problema seguindo as restrições enunciadas por não ter achado nenhum caminho. Caso contrário, é imprimido, de maneira inversa, todos os vértices anteriores no vetor de Grafo na posição da chegada, solucionando assim o problema.

3. Análise de complexidade

Nesta seção, será apresentada a análise do custo teórico de tempo e de espaço.

3.1. Complexidade de tempo

Preencher o grafo usando a lista de adjacências leva o tempo de O(n), onde n é a quantidade de ruas na entrada, o mesmo é observado para a inicialização das listas, atualização da probabilidade e distância ao corpo de bombeiros, com n sendo a quantidade de quarteirões.

A busca em largura tem complexidade é O(V + A), onde A é a quantidade de arestas e V a quantidade de vértices. Como todo vértice é inserido e removido e inserido somente uma vez o tempo gasto para a fila é O(V), como percorremos os adjacentes do véritce, temos comprimento O(A). No pior caso, por exemplo, todos os vértices e arestas são explorados. No entanto, a busca em largura é chamada b vezes, onde b é a quantidade de corpos de bombeiros, tendo então complexidade b*O(V+A).

Já Dijkstra possui complexidade quadrática, $O(n^2)$ já que no pior caso o algoritmo será executado V vezes (V é a quantidade de vértices do grafo), para cada interação, haverá uma pesquisa em todos os vértices do grafo para a atualização das distâncias, dos vértices anteriores e do elemento com menor distância.

O vetor de imprimir o resultado possui tamanho n, que é o total de vértices adjacentes que possui anteriores, desde o elemento de chegada até o de saída. Como é visitado o anterior de cada vértice O(1), então a complexidade é O(n).

Por fim, a complexidade final do algoritmo de tempo será $O(n^2)$*b.

3.2. Complexidade de espaço

A complexidade de espaço será O(V+A), onde V é a quantidade de vértices e A a quantidade de arestas, já que todo o algoritmo tem memória alocada para o grafo com V vértices e A arestas, para que todas as operações sejam realizadas, onde cada vérice e arestas possuem informações que também serão usadas, logo, a alocação de memória é feita em massa em cima dessa estrutura.

4. Análise experimental

Foram realizados vários testes após a finalização do programa em uma máquina com 3GB de memória RAM, processador Intel Core 2 Duo x86 com sistema operacional Linux com distribuição Ubuntu 15.04 em seu compilador nativo, GCC.

Para a análise, foi fixado como 10 o número de casos de testes para cada arquivo testado, sendo que foi criado um programa para a geração de V vértices e A arestas aleatórias. Assim, é possível manter uma proporção entre os casos de testes.

Além disso, foi fixado o número de vértices como 100 e variado o número de arestas de 990 até 4900, assim como a quantidade de corpos de bombeiros.

Através do comando time do Linux no terminal, foram observados os seguintes tempos para o vértice:

Vértices	Arestas	Tempo
100	990	0.017s
100	1900	0.029s
100	2900	0.037s
100	3900	0.043s
100	4900	0.049s

O mesmo foi realizado para o número de arestas. De maneira análoga, foi fixado o número de arestas e variado o número de vértices:

Vértices	Arestas	Tempo
100	4900	0.049s
400	4900	0.113s
600	4900	0.256s
800	4900	0.319s
1000	4900	0.489s

Assim, observamos que o comportamento acima parece corresponder a complexidade.

5. Conclusão

A implementação do trabalho ocorreu com algumas dificuldades e algumas pesquisas. No entanto conhecimentos adquiridos durante o módulo de grafos lecionado na disciplina de Algoritomos e Estruturas de Dados III e conceitos de estruturas de dados como listas e filas que foram adquiridos em Algoritomos e Estruturas de Dados II foram cruciais para o desenvolvimento do algoritmo e programa em sua fase final.

O TAD lista construído para o Trabalho Prático anterior também foi de extrema importância já que muitas coisas puderam ser aproveitadas tanto no arquivo de lista e fila presentes no programa e outras puderam ser otimizadas com as dicas da entrevista. Logo, o maior trabalho foi realizado na construção do grafo com listas encadeadas, com o algoritmo de busca em largura e, principalmente, nas restrições impostas no Dijkstra, como a ordem lexicogŕafica.

A dificuldade com a implementação do grafo em lista encadeada foi resolvida com a construção de uma struct vértice com todas as informações necessárias, eliminando assim vetores auxiliares e trabalhos diretamentes com ponteiros da lista. Já a busca em largura ficou mais simples após muita análise e realização do algoritmo na mão para alguns casos de testes, sendo muito aproveitado no Dijkstra, realizando as modificações necessárias.

No entanto, a ordem lexicogŕafica não foi solucionada corretamente já que o raciocínio utilizado inclui somente a análise do vértice anterior e não todo o caminho, sendo que o tempo restante para a correção do detalhe foi um fator implicante para o melhoramento do algoritmo, tendo em vista a dificuldade para a implementação da correção.

Com o desenvolvimento do trabalho foi possível recordar vários conceitos da linguagem C, a manipulação de ponteiros, listas encadeadas, entre outras ferramentas relativamente simples, mas importantes e úteis na vida de qualquer programador. Treinando também um pouco de lógica e melhor compreensão para a resolução do problema proposto. Mas, principalmente foi possível compreender melhor e na prática a busca em largura e o Dijkstra, como funcionam, a razão de darem certo e a implementação.

Os conhecimentos adquiridos serão importantes ao decorrer da disciplina e na vida acadêmica já que muitos problemas podem ser modelados por grados e é bem útil achar a menor distância em diferentes situações, até mesmo para um funcionário do corpo de bombeiro que quer minimizar seu trabalho.