Documentação Trabalho Prático 1 da Disciplina de Algoritmos 1

Daniel Oliveira Barbosa
Matrícula: 2020006450

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

1. Introdução

O problema se baseia em ajudar o Steven Jodds a encontrar o menor caminho entre uma cidade de origem e uma cidade de destino dentre um conjunto de cidades ligadas por estradas. Porém, temos duas peculiaridades do problema:

1. O caminho não pode passar por estradas de distância ímpar, e
2. O caminho precisa passar por um número par de estradas

Assim, vamos fazer uma abstração, modelando o problema com um grafo e aplicando os conceitos aprendidos na disciplina para podermos ajudar o Steven

2. Implementação

a. Ideia Geral da Solução

Para começarmos a resolver o problema, iniciamos fazendo a modelagem em forma de grafo:

- Cidades foram modeladas em vértices
- Estradas foram modeladas em arestas
    - Essas arestas são não direcionadas pois podemos ir de uma cidade X para uma cidade Y ou vice-versa pela mesma estrada
    - Essas arestas possuem peso não negativo, pois as estradas não podem ter distância negativa

Pelo o que aprendemos na disciplina, sabemos que o algoritmo de Dijkstra é capaz de encontrar a distância mínima entre dois vértices em um grafo não direcionado de pesos não negativos. Contudo, temos duas peculiaridades do problema que precisamos tratar, sendo eles:

1. O caminho não pode passar por arestas de peso ímpar, e
2. O caminho precisa passar por um número par de arestas

Para resolvermos essas peculiaridades, podemos usar a seguinte abordagem:

Seja o conjunto de caminhos entre o vértice de origem e destino do grafo original e do novo grafo, respectivamente, P e P’. Seja também P’ um subconjunto de P. Se conseguirmos garantir que:

- Todos os vértices do grafo original estejam contidos no novo grafo
- Todas as arestas pares do grafo original estejam contidos no novo grafo (solucionando a peculiaridade 1)
- O conjunto P', do novo grafo, possua apenas os caminhos de número par de arestas do conjunto P do grafo original (solucionando a peculiaridade 2)

Poderemos também garantir que aplicar o algoritmo de Dijkstra neste novo grafo retornará a distância mínima entre o vértice de origem e destino do nosso grafo original passando apenas por caminhos de número par de arestas de peso par.

Isso ocorre pois o novo grafo foi construído de modo que, dentre todos os caminhos entre os vértices de origem e destino do grafo original, sobrem apenas caminhos de número par de arestas de peso par no novo grafo. Logo o caminho que possui a distância mínima entre o vértice de origem e destino do novo grafo é necessariamente o caminho de número par de arestas de peso par de menor distância do grafo original por construção.

Desse modo, podemos fazer a construção do novo grafo para atender aos requisitos citados anteriormente da seguinte maneira:

1. Solucionando a peculiaridade da inacessibilidade das arestas de peso ímpar:

   O nosso grafo original, que chamaremos de G1, possui um conjunto de caminhos entre o vértice de origem e o vértice de destino. Porém pode ser que alguns desses caminhos do sejam inacessíveis pois possuem pelo o menos uma aresta de peso ímpar.

   Assim, para tratar essa peculiaridade, podemos criar um novo grafo G2 onde todas as arestas ímpares são eliminadas, e consequentemente, todos os caminhos entre o vértice de origem e o vértice de destino tenham apenas arestas de peso par.

   Sabemos que o grafo original G1 e o grafo após a eliminação das arestas ímpares G2 são equivalentes pois as arestas ímpares em G1 já eram inacessíveis. Logo, eliminá-las não faria diferença em termos de encontrar a menor distância do vértice de origem até o vértice de destino passando por caminhos de número par de arestas de peso par.

2. Solucionando a peculiaridade da inacessibilidade dos caminhos de número ímpar de arestas:

   Apesar de não termos mais arestas de peso ímpar, o conjunto de caminhos entre o vértice origem e o destino do grafo G2 ainda terá caminhos que são inacessíveis pois possuem um número ímpar de arestas.

   Resolver isso não é tão simples quanto eliminar arestas, como na peculiaridade que tratamos anteriormente. Isso se dá pois uma mesma aresta pode estar contido tanto em um caminho com número par de arestas quanto em um caminho com número ímpar de arestas. Logo, precisamos encontrar outra maneira de eliminar os caminhos de número ímpar de arestas.

   A solução que usamos para resolver o problema foi construir um grafo G3, a partir do

grafo G2, que fosse bipartido de modo que:

- Uma partição do grafo, que chamaremos de “partição par”, contenha os vértices que podem ser alcançados apenas por caminhos de número par de arestas
- A outra partição do grafo, que chamaremos de “partição ímpar”, contenha os vértices que podem ser alcançados apenas por caminhos de número ímpar de arestas
- O vértice de origem e destino estejam ambos na partição par

Dado que os vértices de origem e destino estão ambos na partição par, poderemos garantir que todos os caminhos entre eles no grafo G3 sejam caminhos com um número par de arestas.

Isso se verifica pois, se tivermos que caminhar de um vértice qualquer da partição par para outro vértice vizinho qualquer, esse vértice vizinho necessariamente estará na partição ímpar pois o grafo é bipartido. Por outro lado, se tivermos que caminhar de um vértice qualquer da partição ímpar para outro vértice vizinho qualquer, esse vértice vizinho necessariamente estará na partição par pois o grafo é bipartido.

Dessa maneira, qualquer caminho que começa na partição par alterna entre vértices da partição par (que demanda um número par de arestas para chegar até ele) e vértices da partição ímpar (que demanda um número ímpar de arestas para chegar até ele). Essa ida da partição par e volta para a partição par contabiliza um número par de arestas Logo, como o grafo é bipartido e os vértices de origem e destino estão ambos na partição par, o caminho de um até o outro terá um número par de arestas.

Sabemos que o grafo original G1 e G2 são equivalentes ao grafo bipartido G3 pois os caminhos de número ímpar de arestas em G1 e G2 já eram inacessíveis. Logo, eliminá-los não faria diferença em termos de encontrar a menor distância do vértice de origem até o vértice de destino passando por caminhos de número par de arestas de peso par.

Assim, faremos a construção do grafo bipartido G3 para atender os requisitos anteriores da seguinte maneira:

1. Para cada vértice v do grafo G2, crie dois vértices v1 e v2 em G3.
2. Cada aresta com 1 no final estará na partição par e cada aresta com 2 no final estará na partição ímpar. Assim, por exemplo, v1 estará na partição par e v2 na partição ímpar.
3. Para cada aresta que conecta vértices u e v, crie duas arestas:
    i. Uma que conecta u1 com v2 (u1 na partição par para v2 na partição ímpar)
    ii. Outra que conecta u2 com v1 (u2 na partição ímpar para v1 na partição par)

    OBS.: mais a frente veremos que na verdade terão que ser criados 4 arestas (o dobro de arestas do que falamos anteriormente) para contabilizar pelo fato de que o grafo é não direcionado. Contudo, vamos deixar isso para ser explicado mais a frente.

Assim, garantimos que todos os caminhos acessíveis (caminhos de número par de arestas e de peso par) do grafo original G1 estejam no grafo bipartido G3.

Agora, basta aplicar o algoritmo de Dijkstra no grafo bipartido G3 e será garantidamente retornado a menor distância entre o vértice de origem e o vértice de destino passando apenas pelos caminhos de número par de arestas de peso par do grafo original.

b. Implementação da Solução em C++

Vamos dividir a implementação do algoritmo em código em duas partes para facilitar a compreensão:

1. Construir do grafo bipartido G3

O grafo bipartido foi construído em C++ como uma lista de adjacências (implementado como um vetor de vectors de pares de int alocado dinamicamente). A escolha dessa estrutura se deu pela sua facilidade de armazenamento e acesso.

Assim, a lista de adjacências é criada com 2**N** posições para comportar os vértices das duas partições. As primeiras **N** posições armazenam os vértices da partição par (como determinamos anteriormente ter final 1) e as próximas N posições armazenam os vértices da partição ímpar (como determinamos anteriormente ter final 2).

Para cada aresta do vértice u pra v do grafo G1 passado como entrada, são criadas 4 arestas no nosso grafo bipartido:

- Uma aresta de u1 para v2
- Uma aresta de v2 para u1

OBS.: assim garantimos a ida e a volta entre os vértices u1 e v2 dado que o grafo é não direcionado

- Uma aresta de u2 para v1
- Uma aresta de v1 para u2

OBS.: assim garantimos a ida e a volta entre os vértices u2 e v1 dado que o grafo é não direcionado

Como já foi dito, criar 4 arestas foi necessário para garantir a modelagem de um grafo não direcionado.

2. Encontrar a distância mínima do vértice 1 até **N**

Para encontrarmos a distância mínima a partir do grafo bipartido G3 que construímos anteriormente, implementamos o algoritmo de Dijkstra padrão.

No algoritmo de Dijkstra usamos a fila de prioridade da biblioteca padrão do C++ (priority_queue) e fizemos com que a prioridade da fila fosse de vértice de menor distância para vértice de maior distância através de um functor.

3. Análise de complexidade

Para fazermos a análise do algoritmo, também iremos dividi-lo em duas principais partes para facilitar a análise:

1. Construir do grafo bipartido G3

Dado que:

- A operação de adicionar uma aresta à lista de adjacências é feita em O(1)
- Para cada aresta do grafo original G1, são adicionadas 4 arestas ao grafo bipartido G3
- O número de arestas do grafo original G1 é **M**

Temos que a construção do grafo bipartido é feito em O(4***M**). Eliminando fatores constantes chegamos em uma complexidade de construção do grafo de O(**M**).

2. Encontrar a distância mínima do vértice 1 até **N**

Por outro lado, encontrar a distância mínima é feita pelo algoritmo de Dijkstra padrão. Como ele não foi modificado para atender a nenhuma peculiaridade, temos que a operação de encontrar a distância mínima a possui complexidade padrão do Dijkstra que é O(**M**+**N**).

Logo, podemos concluir que a complexidade do algoritmo como um todo (incluindo construção e encontrar a distância mínima) é de O(**M**) + O(**M**+**N**) = O(**M**+**N**).

4. Conclusão

Em suma, neste trabalho pude compreender como a construção e transformação de um grafo pode nos ajudar a entender e resolver problemas de formas mais simples e eficientes.

Isso foi muito importante dado que a minha primeira solução do problema foi manter o grafo original e tentar resolver o problema modificando o algoritmo de Dijkstra, armazenando tanto os caminhos pares quanto ímpares, gerando um retrabalho. Em um segundo momento, através de uma nova construção do grafo, pude transformar o problema em outro mais simples no qual o algoritmo de Dijkstra padrão fosse capaz de resolver de forma mais eficiente.