Relatório do trabalho final de TEG - 2019/2

Matheus Cercena

1. Contextualização

O trabalho consiste em resolver o problema que um grupo de ciclistas que residem em diferentes cidades possui ao desejar chegar em uma determinada cidade de encontro para um evento, com todos os ciclistas chegando a cidade em um mesmo horário.

Pede-se que seja disponibilizada 2 opções ao usuário: dada uma cidade de origem, explicitar todas as rotas possíveis dessa cidade até a cidade encontro; e a geração de uma tabela contendo os diferentes horários de partida de cada cidade, a fim dos ciclistas das diferentes cidades poderem percorrer o trajeto juntos até a cidade de destino.

O problema possui como empecilho o fato de que as cidades possuem vários caminhos com diferentes distâncias até a cidade de encontro. O trabalho busca garantir quais rotas associadas a um horário de saída permite os ciclistas a chegarem juntos em um único horário de chegada a cidade de encontro.

2. Implementação

O trabalho foi feito na linguagem C++, onde o uso das bibliotecas list, queue e iterator foram de grande ajuda para armazenar e manipular os diferentes dados do grafo com as funções auxiliares. A leitura do arquivo de entrada contendo os dados do grafo é feita com as bibliotecas string e stdlib, com a utilização de tokens para demarcar o que está sendo lido.

A resolução do problema descrito é feita com a implementação do algoritmo de Dijkstra, para entre dos vários caminhos disponíveis da cidade de partida até a cidade de encontro, o algoritmo eleger um único caminho de menor custo e com base neste caminho, uma outra função define os horários de partida de cidade subtraindo o horário de chegada definido com o tempo gasto para percorrer o caminho (este que é feito dividindo a distância pela velocidade média previamente definida).

Para a verificação de todas as rotas disponíveis da cidade partida até a cidade de encontro, a ideia foi de se utilizar o algoritmo DFS com algumas pequenas modificações e assim imprimir cada vértice do caminho conforme a função anda no grafo.

2.1 Estrutura do grafo

Utilizando o recurso de orientação a objeto do C++, o Grafo é estruturado em formato de classe. Seus atributos privados são a quantidade de vértices totais como inteiro e um ponteiro que aponta para as listas de adjacência de cada vértice, onde cada espaço da adjacência é formado por um par (utilizando pair<x,y>) em que o primeiro elemento corresponde o vértice adjacente e o segundo elemento corresponde o custo para se chegar nele. Já os atributos públicos incluem o método construtor que define a quantidade de vértices e cria as listas de adjacências, o método que adiciona as arestas para as listas, e a definição das funções que são utilizadas para a resolução dos problemas.

2.2 Padronização e leitura da entrada de dados

Cada linha do texto já é subentendido no algoritmo como um vértice do grafo de forma sequencial (1ª linha - vértice 1, 2ª linha - vértice 2…), assim não sendo necessário a identificação explícita de qual linha corresponde a qual vértice. Sendo assim, em cada linha (vértice) cada ocorrência de adjacência é separado por um “;”, que por sua vez contém 2 elementos separados por um “-”: o primeiro elemento é o vértice de adjacência e o segundo elemento é o custo para se chegar até ele.

A leitura é iniciada com uma função que conta as linhas do arquivo e retorna a quantidade para a função main instanciar o grafo. Então parte-se para a definição dos espaços das listas de adjacência: com tokens, a função consegue cortar a linha do arquivo delimitado por “;” e “-”, armazenando-os em variáveis temporárias para quando o loop identificar a 2ª leitura (o dado do custo da aresta) chamar a função que adiciona a aresta/define o par de adjacência do vértice atual.

2.3 Dijkstra e função horários para geração da tabela de partida

O algoritmo de Dijkstra escolhe um único caminho que possui o menor custo até o destino para servir de parâmetro para a montagem da tabela de horários. O algoritmo trabalha com um vetor de distâncias que armazena o custo até os outros vértices, um vetor de visitados para verificar se determinado vértice já foi analisado, e uma fila de prioridade que ordena a prioridade dos vértices para o caminho de acordo com a menor distância disponível.

O loop do algoritmo percorre todos os elementos da fila até ser esvaziada, de forma com que os vértices não visitados possam ser checados de acordo com sua ordem de prioridade. No condicional de checagem, a lista de adjacência do vértice analisada é percorrida com o iterator para verificar se a distância armazenada no vetor de distância na posição do iterator é maior que a soma do custo no vértice adjacente com a distância contida no vetor na posição do vértice analisado. Caso verdadeiro, o vetor de distâncias na posição do iterator é atualizado e se é adicionado à fila de prioridade.

Quando a função do Dijkstra retorna o valor do caminho mínimo, ele é usado como parâmetro para a chamada de função dos horários, dividindo seu valor pela velocidade média para se obter o tempo do trajeto, junto com um horário de chegada informado pelo usuário. A função de horários define o horário de partida pela subtração do horário de chegada com a duração do trajeto, e então verifica se o resultado é menor que 0 para decidir qual dia se trata a partida e adicionar 24 ao valor de partida para corrigir. O resultado é o print da cidade que se trata, seu dia e horário de partida.

2.4 DFS para verificação de rotas disponíveis

O algoritmo para imprimir os caminhos baseado em DFS utiliza-se da recursividade para andar no grafo e uma função auxiliar inicial que inicia um vetor de vértices visitados como falso e chama a função principal. A ideia do algoritmo é imprimir o caminho a partir de um vetor que guarda os vértices de um único caminho quando os dois vértices passado por parâmetros sejam iguais, isto é, quando o vértice inicial que começa na origem for igual ao vértice final de destino.

Para se utilizar da recursividade, os 2 vértices do parâmetro precisam ser diferentes. Assim, um iterator é usado para percorrer a lista de adjacência do vértice analisado e verifica

se o vértice adjacente contido no iterator não foi visitado (pelo vetor de vértices visitados). O vértice presente na lista de adjacência é passado como parâmetro na recursão, e então este vértice é marcado como visitado e armazenado no vetor de caminho. Este processo é repetido até se chegar no vértice de destino.

2.5 Função Main

A função main apenas tem o trabalho de instanciar o grafo com as informações recebidas, de chamar as funções que fazem a parte mais densa trabalho e oferecer as 2 operações possíveis dentro do algoritmo em formato de switch case. Para a definição das tabelas dos horários de partida, a main chama a função do Dijkstra e dos horários repetidas vezes para todos os vértices individualmente.