TRABALHO PRÁTICO I
RELATÓRIO

Luiz Eduardo Pereira

0021619

Instituto Federal de Minas Gerais, Formiga, MG.

INTRODUÇÃO

Este trabalho tem como objetivo a implementação de uma estrutura de dados que utilize um tipo abstrato de dados (TAD) de criação e utilização de grafos e a criação de um programa que encontre o menor caminho entre dois pontos. A TAD possui funções para realizar todas as operações básicas de um grafo ou dígrafo, sendo possível representar qualquer grafo que possua um tamanho pré-determinado. Além de ser possível a criação de um grafo durante a execução do programa, também é possível salvar e carregar os dados em um arquivo. O Algoritmo caminho utiliza a TAD grafo para procurar o caminho mais curto entre o ponto A e B, sendo necessária a utilização de pesos nas arestas.

IMPLEMENTAÇÃO

Este trabalho foi desenvolvido na linguagem C e foi dividido nos arquivos Grafo (grafo.c e grafo.h) e Caminho Mínimo (caminho.c e caminho.h).

Grafo

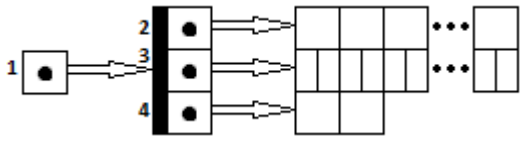

Imagem 1

A imagem 1 representa a estrutura do grafo. A única conexão que o programador do grafo possui com a implementação desta TAD é o ponteiro *Grafo (1) do tipo struct grafo. Todos os vetores do arquivo grafo.c foram indexados de 0.

Os vértices do grafo são representados em um vetor (2) do tipo inteiro. O valor 0 representa a inexistência de um vértice. O vetor vértice guarda o identificador do vértice, sendo assim, posição 0 guarda vértice 1.

As arestas também são representadas por um vetor (3) do tipo struct aresta. Esta struct possui um vértice de origem e um de destino.

Para guardar o tamanho máximo deste grafo, foi usado um vetor (4) do tipo inteiro no qual a posição 0 representa o tamanho máximo de vértices e a posição 1 o tamanho máximo de arestas.

Nas funções de busca, para encontrar o vértice ou aresta, só é necessário entrar no vetor e buscar na posição (vértice/aresta - 1).

Por se tratar apenas de números, as funções GGcarregaGrafo e GBsalvaGrafo não possuem nenhuma complexidade, por não ter a necessidade de uso de manipulação de caracteres. Basicamente, a função GGcarregaGrafo apenas verifica se o arquivo não é vazio e faz sua leitura. Primeiramente lê o número máximo de vértices e de arestas, depois as arestas do grafo.

Caminho

Para a implementação do caminho mínimo foi utilizado o algoritmo de Dijkstra, com devidas modificações para usar a estrutura de dados criada. Todo o algoritmo foi feito em apenas uma função, mas a mesma foi dividida em blocos de atividades.

Este algoritmo utiliza uma constante infinita, mas neste caso está convencionado que infinito é igual a 999.999. Para qualquer grafo com pesos absurdamente grandes, em que o custo total chega ao valor da constante, não é garantido o funcionamento do programa. Na necessidade de uma constante maior, é necessário mudar INF em caminho.h, respeitando o valor máximo de um inteiro. Todos os vetores do arquivo caminho.c são indexados de 1.

Bloco de declarações de variáveis: Este bloco declara todas as variáveis a ser usada no programa.

O vetor *pai do tipo int guarda na posição[vértice] o identificador do vértice filho, até chegar no vértice B.

O vetor *custo do tipo float armazena o menor custo para se chegar até um determinado vértice. Todos os vértices iniciam com custo infinito, exceto A, que inicia com 0.

O vetor *flag do tipo boolean marca os vértices que já foram visitados, todos iniciam com FALSE.

Bloco de custos: Este bloco irá calcular todos os custos, até chegar em B.

Primeiramente, o programa busca pelo grafo o vértice com o menor custo. Quando um vértice possui o custo igual a infinito, ou ele ainda não pode ser visitado ou ele é um vértice isolado no grafo. Para encontrar o menor custo, é usado uma variável auxiliar do tipo inteiro para armazenar o vértice com menor custo que ainda não foi visitado. Se ao final do laço, a variável auxiliar ainda armazenar o valor infinito, e o valor inicial de B (INF) não tiver sido alterado, significa que B possui isolação em relação a A.

Depois de definido qual é o vértice com menor custo, ele é marcado como visitado e é verificada cada uma de suas arestas. Se o custo do vértice mais o peso da aresta são menores que o custo do vértice da outra extremidade da aresta, então o custo do outro

vértice é atualizado com o valor da soma. Após todas as arestas serem verificadas, um novo vértice com menor custo é escolhido, é segue assim até que B seja visitado.

Bloco caminho mínimo: Com todos os menores custos colocados em cada vértice, este bloco traça uma rota de B até A, gravando o vértice pai de cada um. A rota é encontrada de traz para a frente (B - A), pois assim é mais fácil achar quem é o pai de cada vértice. Iniciando em B, a estratégia usada foi verificar, pegando cada uma das arestas do vértice atual, “se o custo do meu vizinho mais o peso entre nossa aresta for menor ou igual a meu custo, ele é meu pai”.

Em seguida é impresso o menor caminho.

VALIDAÇÃO

Os testes em sua maioria foram feitos usando o main.c fornecido pelo professor e os testes prontos do run.codes. Outros pequenos testes foram feitos para testar cada função separadamente, sem nenhuma metodologia especifica.

CONCLUSÃO

Com este trabalho foi possível concluir na pratica a utilidade de uma TAD. Mesmo que ela não esteja implementada, é possível criar códigos que a usem, sendo necessário apenas possuir os protótipos das funções e saber o que elas resolvem. O caminho mínimo é um ótimo exemplo para isso, sem possuir a estrutura de dados grafos seria muito difícil a implementação do programa.

A maior dificuldade deste trabalho foi a implementação do caminho mínimo, pois não havíamos visto os algoritmos para resolvê-los ainda. Mas com a estratégia de resolução do algoritmo de Dijkstra, foi possível a implementação do trabalho.

O trabalho contribuiu muito para o entendimento de AED e no modo com a linguagem C funciona.

REFERÊNCIAS

Caminho Mínimo - Prof. Diego Mello da Silva:
https://docs.google.com/viewer?a=v&pid=sites&srcid=aWZtZy5lZHUuYnJ8ZGllZ29zaWx2YXxneDozZDRlY2FhODlkMGQ5MWEy - (Acesso em 19/06/2016).