Primeiro Trabalho Prático

Estruturas de Dados 1

Prof. Paulo Henrique Ribeiro Gabriel

O objetivo deste trabalho é implementar uma aplicação do TAD Pilha. Para isso, é necessário adaptar o TAD desenvolvido em aula e usá-lo em conjunto com um algoritmo. Implementar a atividade sem compartilhar código com colegas e nem pesquisar na internet.

Aplicação do TAD Pilha

Um *grafo* é um modelo matemático formado por dois conjuntos: um conjunto de *vértices* e um conjunto de *arestas*. Cada aresta está associada a dois vértices: o primeiro é a origem do aresta e o segundo é o destino. Grafos descrevem relações entre objetos. Diversos problemas do mundo-real podem ser modelados por meio de grafos. Por exemplo, podemos representar um mapa rodoviário, onde os vértices são cidades e as arestas são as estradas.

Na Figura 1 temos um exemplo de grafo com seis vértices $\{a, b, c, d, e, f\}$ e oito arestas $\{ab, ac, ad, bd, cd, cf, de, fe\}$. Note que, nesse exemplo, as arestas são *orientadas*, ou seja, elas possuem uma direção. Assim, por exemplo, há uma aresta saindo de a e chegando a b, mas não temos uma aresta saindo de b e chegando até a. (Grafos assim são, muitas vezes, usados para representar ruas de mão única.). Além disso, as arestas possuem um peso (valor numérico) que indica seu custo.

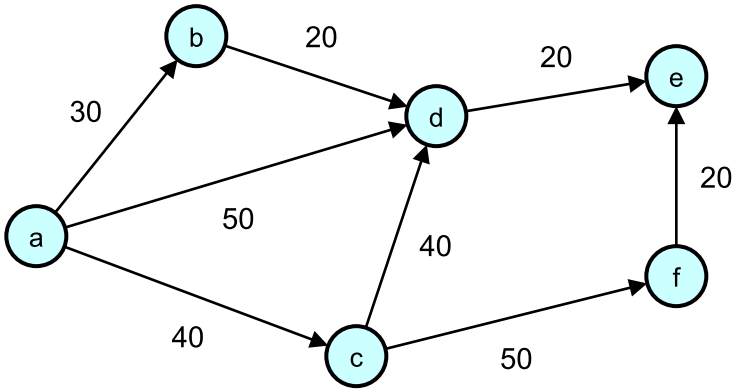

Figura 1: Exemplo de grafo.

Dizemos que um vértice Y é *alcançável* a partir de outro vértice X se é possível construir um *caminho* saindo de X e chegando a Y. No caso da Figura 1, f é alcançável a partir de a, pois existe um caminho entre ambos (no caso, a sequência: ac, cf). Nesse caso, o *custo total* desse único caminho é $40 + 50 = 90$. Obviamente, pode existir mais de um caminho entre dois vértices (entre a e d, por exemplo, existem três, cada um com um custo diferente). Finalmente, existem vértices que **não são** alcançáveis a partir de outros; é o caso, por exemplo, do vértice f que não é alcançável a partir de b (não existe um caminho respeitando a orientação das arestas). Analogamente, a não é alcançável por nenhum outro vértice.

Percebe-se, portanto, que para descobrir se um vértice é alcançável a partir de outro, basta procurar por *ao menos um caminho entre eles*. Isso pode ser feito por meio do seguinte algoritmo: a partir de um vértice inicial (origem), insira todos os vértices vizinhos em uma pilha. Em seguida, pegue (remova) o vértice do topo da pilha e repita o processo, ou seja, insira os vizinhos desse vértice na pilha. Esse algoritmo de busca segue até que o alvo (ou seja, o vértice destino) seja encontrado ou até que a pilha esteja vazia. Note que, como as arestas são orientadas, podemos dizer que b é vizinho de a, mas a **não é vizinho** de b, pois não existe a aresta ba, ou seja, saindo de b e chegando a a.

Tarefa

Implemente um programa em C que verifique se dois vértices estão conectados, ou seja, se a partir de um vértice é possível alcançar o outro. Para isso, você deve implementar o TAD Pilha **completo** (com as operações de *criar*, *destruir*, *empilhar*, *desempilhar*, *cheia* e *vazia*) e todas as demais funções necessárias para armazenar os vértices e verificar a existência de caminhos.

Caso exista ao menos um caminho entre dois vértices, o programa deve indicar o *custo total* desse caminho. Caso contrário, ou seja, caso não exista caminho, o custo total será -1. Note que o programa não precisa indicar todos os possíveis caminhos, basta um deles.

Para armazenar o grafo, é possível usar uma matriz (conhecida como *matriz de adjacências*) da seguinte maneira: se existe uma aresta entre os vértices X e Y, insere-se o custo da aresta XY na célula correspondente a esses dois vértices; caso contrário, insere-se 0. No exemplo a seguir, temos a matriz correspondente ao grafo da Figura 1:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 30 & 40 & 50 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 40 & 0 & 50 \\ 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 \end{pmatrix} \end{array}$$

Note que a diagonal da matriz é sempre igual a 0 e que a matriz é simétrica. Nesta atividade, não é necessário se preocupar com ciclos (ou seja, caminhos que "voltam" à origem).

Formato da Entrada

A entrada do programa será o número de vértices seguido do número de arestas, seguidos das arestas e seus custos (uma por linha. Ao final, são fornecidos os dois vértices a serem analisados, ou seja, a origem e o destino. Todos os valores serão **inteiros**. Para o grafo da Figura 1, a entrada seria dada no seguinte formato:

```
6
8
1 2 30
1 3 40
1 4 50
2 4 20
3 4 40
3 6 50
4 5 20
6 5 20
1 6
```

Note que, nesse exemplo específico, foi necessário "mapear" as letras em números ($a = 1, b = 2, c = 3$, etc.). No caso, estamos procurando um caminho entre a e f.

Formato da Saída

Conforme dito anteriormente, a saída será composta pelo custo do caminho, se existir, ou -1, caso contrário. Para o exemplo de entrada anterior, a saída será:

```
90
```

Critérios de Avaliação

O projeto será avaliado principalmente levando em consideração:

1. Processamento correto das entradas e saídas do programa;
2. Realização das tarefas descritas;
3. Bom uso das técnicas de programação;
4. Boa endentação e uso de comentários no código;
5. Boa estruturação e modularização do código.

Observações

1. O trabalho deve ser feito em **grupos de duas ou três pessoas** e em Linguagem C. Não serão aceitos trabalhos individuais nem desenvolvidos por grupos com mais de três discentes.

2. Não deverá ser utilizada qualquer variável global nem bibliotecas com funções prontas (a não ser aquelas para entrada, saída e alocação dinâmica de memória).

3. Todas as submissões são checadas para evitar plágio; portanto, evite problemas e implemente o seu próprio código.

4. Ainda sobre plágio, A detecção de cópia de parte ou de todo código-fonte, de qualquer origem, implicará reprovação direta no trabalho. Compartilhem ideias, modos de resolver o problema, mas não o código. Qualquer dúvida entrem em contato com o professor.

5. Comente o seu código com uma explicação rápida do que cada função ou trecho importante de código faz (ou deveria fazer). Os comentários e a boa modularização do código serão checados e valem nota.

6. Todas as funções devem ser implementadas em um único arquivo `.c`; porém, deve-se ficar atendo ao bom uso de TAD.

7. Entradas e saídas devem ser lidas e escritas a partir dos **dispositivos padrão**, ou seja, use as funções `scanf` e `printf`.

8. Lembre-se de respeitar estritamente o formado de entrada e saída. Uma quebra de linha a mais ou a menos resultará em erro no caso de teste.

9. O professor pode tirar dúvidas sobre o enunciado, ou sobre a lógica por trás do programa. No entanto, ele não olhará código-fonte em busca de erros! Procure os monitores nesse caso.

10. Esse trabalho considera o uso de uma pilha estática e sequencial. Porém, caso o grupo opte por alocar dinamicamente o vetor da pilha, poderá receber uma bonificação na nota do trabalho.