SCC0216 – Modelagem Computacional em Grafos

Prof. Ricardo J. G. B. Campello

Estagiário PAE: Francisco A. R. Anjos

1º semestre 2016

Projeto da Disciplina

Introdução

O problema do caminho mínimo consiste em encontrar o caminho de menor custo que começa em um vértice **v** e termina em um vértice **w**. Uma vez definido tal problema, o presente projeto consiste em:

1. Implementar uma **Estrutura de Dados Dinâmica** para grafos não-direcionados na forma de uma **lista de adjacências**.
2. Usar a estrutura de dados implementada para implementar um **TAD Grafo**. Seu TAD deve possuir necessariamente o seguinte conjunto mínimo de operações (funções):
   - **endVertices(G, e, v, w):** retorna referências, **v** e **w**, para os dois vértices finais da aresta **e**;
   - **opposite(G, v, e):** retorna a referência do vértice oposto a **v** na aresta **e**;
   - **areAdjacent(G, v, w):** retorna **1 (um)** se os vértices **v** e **w** forem adjacentes ou **0 (zero)** caso contrário;
   - **replaceVertex(G, v, o):** substitui o elemento armazenado no vértice **v** por **o**;
   - **replaceEdge(G, e, o):** substitui o elemento armazenado na aresta **e** por **o**;
   - **insertVertex(G, o):** insere um novo vértice isolado, armazenando nele o elemento **o**, e retorna uma referência para o vértice inserido;
   - **insertEdge(G, v, w, o):** insere uma nova aresta entre os vértices **v** e **w**, armazenando nela o elemento **o**, e retorna uma referência para a aresta inserida;
   - **removeVertex(G, v):** remove o vértice **v** (e suas arestas) e retorna o elemento armazenado nele;
   - **removeEdge(G, e):** remove a aresta e e retorna o elemento armazenado nela;
   - **vertexValue(G, v):** retorna o elemento armazenado no vértice **v**;
   - **edgeValue(G, e):** retorna o elemento armazenado na aresta **e**;
   - **numVertices(G):** retorna a quantidade de vértices do grafo **G**;
   - **numEdges(G):** retorna a quantidade de arestas do grafo **G**;
3. Implementar uma função chamada **FloydWarshall(G, D, P)** que determina a **distância (custo do menor caminho)** entre todos os pares de vértices do grafo **G** bem como a matriz de parentescos, armazenando os respectivos resultados em **D** e **P**.

4. Implementar uma função chamada **printDistance(G, D, v, w)** que imprime na tela a **distância (custo)** entre dois os vértices do grafo **G** conforme o formato descrito na seção **Entrada e Saída**.
5. Implementar uma função chamada **printShortestPath(G, P, v, w)** que imprime na tela o menor caminho entre os vértices **v** e **w** conforme o formato descrito na seção **Entrada e Saída**.
6. Implementar uma função chamada **printGraph(G)** que imprime na tela o grafo **G** conforme o formato descrito na seção **Entrada e Saída**.

Observações:

1. Os vértices devem armazenar elementos com valores inteiros positivos enquanto as arestas devem armazenar elementos com valores reais positivos.
2. As "referências" a vértices e arestas são ponteiros para os respectivos nós das listas de vértices e arestas. Utilize o encapsulamento descrito no capítulo 9 do livro de Cerqueira, Celes e Rangel para impedir que o usuário consiga manipular diretamente as estruturas de dados do grafo.
3. Note que as funções **endVertices(G, e, v, w)** e **FloydWarshall(G, D, P)** possuem mais do que uma saída e por isso utilizam parte dos parâmetros como retorno, isto é, **v**, **w**, **D** e **P** são parâmetros de saída e não de entrada.
4. Faça comentários ao longo do código afim de deixá-lo o mais claro possível.

Entrada e Saída de Dados

O grafo a ser manipulado será especificado ao seu programa a partir da entrada de dados padrão (teclado). Para isso, implemente um procedimento que leia dados a partir do teclado e utilize as funções do seu TAD Grafo para construir e armazenar o grafo correspondente. O número máximo de vértices que seu programa deve tratar é igual a 100 e não haverá casos de arestas paralelas ou laços. Caso não exista um caminho entre dois vértices, defina sua distância igual a 1000000 (um milhão).

Os comandos permitidos para operar sobre o seu TAD são representados por duas letras maiúsculas. Toda linha de entrada obrigatoriamente inicia com um comando. Só serão fornecidos como entrada comandos válidos, descritos a seguir. O símbolo ¶ denota um único espaço em branco.

- **Cria vértice**

CV ¶ *o*

CV cria um vértice contendo o valor *o* . Vértices só armazenarão números inteiros positivos.

- **Deleta vértice**

DV ¶ *v*

DV deleta o vértice identificado por v (veja abaixo a nota a respeito de identificadores). Não serão fornecidos como entrada identificadores inexistentes.

- **Cria aresta**

CA ¶ v_i ¶ v_j ¶ o

CA cria uma aresta entre os vértices de identificador v_i e v_j. O valor armazenado na aresta é um número real positivo especificado por o. Não serão fornecidos identificadores de vértices inexistentes.

- **Deleta aresta**

DA ¶ e

DA deleta a aresta de identificador e. Não serão fornecidos identificadores de arestas inexistentes.

- **Troca valor do vértice**

TV ¶ v ¶ o

TV troca o valor armazenado no vértice de identificador v pelo valor o. Não serão fornecidos identificadores de vértices inexistentes.

- **Troca valor da aresta**

TA ¶ e ¶ o

TA troca o valor armazenado na aresta de identificador e pelo valor o. Não serão fornecidos identificadores de arestas inexistentes.

- **Executa o algoritmo de Floyd-Warshall**

FW

FW executa o algoritmo de Floyd-Warshall retornando as matrizes de distância e de parentesco que deverão ser armazenadas fora do TAD.

- **Imprime grafo**

IG

IG imprime o grafo na tela. Para definir a forma que seu programa deve imprimir o grafo na tela, considere um grafo com ***n*** vértices e ***m*** arestas, em que $identificador(v_i)$ é o identificador de um vértice ***i*** e $valor(v_i)$ é o valor armazenado nele. Considere ainda que $identificador(a_i)$ é o identificador da aresta ***i***, $identificador(v_{ai})$ e $identificador(v_{bi})$ são os identificadores de seus dois vértices finais. Finalmente, $valor(a_i)$ é o valor armazenado na aresta ***i*** que deve ser impresso com exatamente três casas de precisão. Seu programa deve imprimir o grafo

exatamente na seguinte forma:

n '\n'
$identificador(v_1)$ ¶ $valor(v_1)$ '\n'
$identificador(v_2)$ ¶ $valor(v_2)$ '\n'
...
$identificador(v_n)$ ¶ $valor(v_n)$ '\n'
m '\n'
$identificador(a_1)$ ¶ $identificador(v_{a1})$ ¶ $identificador(v_{b1})$ ¶ $valor(a_1)$ '\n'
$identificador(a_2)$ ¶ $identificador(v_{a2})$ ¶ $identificador(v_{b2})$ ¶ $valor(a_2)$ '\n'
…
$identificador(a_m)$ ¶ $identificador(v_{am})$ ¶ $identificador(v_{bm})$ ¶ $valor(a_m)$ '\n'

O símbolo ¶ representa um único espaço em branco e **'\n'** é a quebra de linha a ser usada. É claro que a impressão gerada, em geral, não é única (as linhas correspondentes aos diferentes vértices e arestas podem estar em qualquer ordem). Para tornar única a impressão gerada pela função **IG**, seu programa deve proceder da seguinte forma:

1. Imprima os pares identificador de vértice – valor de vértice de forma que $identificador(v_1)$ < $identificador(v_2)$ < … < $identificador(v_n)$, ou seja, os vértices devem impressos por ordem de identificador.
2. Imprima as informações das arestas de forma que seus identificadores sejam apresentados ordenadamente, ou seja: $identificador(a_1)$ < $identificador(a_2)$ < … < $identificador(a_m)$. Além disso, imprima os identificadores de seus dois vértices de forma que o menor apareça antes.

- **Imprime a distância calculada por Floyd-Warshall**

ID ¶ v_i ¶ v_j

Após executado o comando **FW**, **ID** imprime a distância entre o vértice de identificador v_i e o vértice de identificador v_j. O valor impresso deve possuir exatamente três casas de precisão e ser seguido de uma quebra de linha (**'\n'**).

- **Imprime o caminho mínimo**

IC ¶ v_i ¶ v_j

Após executado o comando **FW**, **IC** imprime o caminho mínimo entre o vértice de identificador v_i e o vértice de identificador v_j no seguinte formato:

v_i ¶ v_a ¶ v_b ¶ ... ¶ v_c ¶ v_j '\n'

Note que v_a, v_b, …, v_c são os identificadores dos vértices intermediários. **NÃO** confundir com

os valores armazenados pelos vértices. Caso não exista um caminho mínimo imprima o caracter **0 (zero)** seguido de uma quebra de linha (**'\n'**). Não serão passados à essa função vértices que possuam mais de um caminho mínimo.

- **Termina o programa**

```
FM
```

FM termina a execução do programa. Todas as estruturas dinâmicas devem ser desalocadas e seu programa deve encerrar.

Observações:

1. Repare que não existem entradas para todas as funções especificadas no TAD, porém, essas funções serão utilizadas como operações auxiliares para as funções **FW**, **IG** e **IC**.
2. Note que o usuário não tem como inserir “referências” (ponteiros) para vértices e arestas conforme estas são especificadas no TAD. Para interagir com o usuário, o programa deverá se referir a vértices e arestas através de um **identificador numérico único**.
3. Os identificadores para vértices e arestas devem ser **gerados automaticamente** durante a entrada de dados, com o cuidado de não gerar identificadores já existentes. Para o primeiro vértice gerado o identificador 1 deve ser atribuído, para o segundo o identificador 2 e assim sucessivamente. O mesmo vale para as arestas, para a primeira aresta gerada o identificador 1 deve ser atribuido, para a segunda o identificador 2 e assim sucessivamente.
4. Os identificadores devem ser **únicos** e **NÃO** devem ser reutilizados, ou seja, caso uma aresta seja removida, seu identificador não pode ser utilizado para outra aresta.
5. Os identificadores **NÃO** devem ser armazenados na estrutura de dados do grafo, pois este será utilizado como TAD.
6. Não confunda o identificador de um vértice ou aresta com o conteúdo que o vértice ou aresta armazena.
7. Para associar os identificadores às referências dos vértices e arestas requeridos pelas operações do TAD, você deverá utilizar uma **estrutura de dados auxiliar** (um mapa), bem como as devidas operações nessa estrutura, de forma que a partir de um identificador você obtenha uma referência e vice-versa.

Exemplo de Entrada (caso1.in)

```
CV 6
CV 9
CA 1 2 20.5
CV 0
CV 0
CV 5
TV 5 8
CA 5 1 15.2
```

```
CA 2 3 10.8
CA 3 4 5.1
CA 2 16.3
IG
FM
```

Exemplo de Saída (caso1.out)

```
5
1 6
2 9
3 0
4 0
5 8
4
1 1 2 20.500
2 1 5 16.300
3 2 3 10.800
4 3 4 5.100
```

Exemplo de Entrada (caso2.in)

```
CV 3
CV 5
CA 1 2 10.3
CV 10
CA 3 1 15.2
CV 10
CA 3 4 0.8
CA 2 4 0.95
CV 5
CA 3 5 7.6
CA 5 2 19.0
FW
ID 3 2
ID 1 4
IC 2 3
IC 5 1
IC 1 5
ID 5 1
ID 1 5
FM
```

Exemplo de Saída (caso2.out)

```
1.750
11.25
2 4 3
5 3 4 2 1
1 2 4 3 5
19.650
19.650
```

Avaliação

Elabore um relatório que discuta em linhas gerais a estrutura lógica usada para desenvolver o trabalho, sem apresentar o código. O relatório deve permitir compreender o trabalho sem a necessidade de ler código fonte. Submeta o relatório compactado em conjunto com código fonte pela plataforma **Run.Codes**. A nota do projeto substituirá as duas piores notas das atividades de laboratório incluindo ausências por qualquer motivo (justificada ou não).

Considerações Finais:

1. O aluno deverá utilizar a linguagem de programação **C (padrão ANSI)**;
2. O arquivo **makefile deve estar na raiz do arquivo compactado**. Caso contrário, a plataforma run.codes não consegue encontrá-lo;
3. O projeto deve ser feito individualmente;
4. Serão anulados todos os trabalhos nos quais forem detectados qualquer tipo de cópia ou plágio, não importando a origem;
5. Os trabalhos devem ser submetidos pela plataforma run.codes **(código da disciplina: 4ZGF)**;
6. O prazo limite para a entrega do projeto é **7 de junho**;