Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Estrutura de Dados III

Docente
Profa. Dra. Cristina Dutra de Aguiar
cdac@icmc.usp.br

Aluno PAE
João Paulo Clarindo
jpcsantos@usp.br

Monitores
Eduardo Souza Rocha
eduardos.rocha17@usp.br ou telegram: @edwolt
Maria Júlia Soares De Grandi
maju.degrandi@usp.br ou *telegram: @majudegrandi*

Terceiro Trabalho Prático

Este trabalho tem como objetivo aprofundar conceitos relacionados a grafos.

O trabalho deve ser feito por, no máximo, 2 alunos que são os mesmos alunos do primeiro e do segundo trabalhos práticos. A solução deve ser proposta exclusivamente pelo(s) aluno(s) com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.

Programa

Descrição Geral. Implemente um programa por meio do qual o usuário possa obter dados de um arquivo binário de entrada, gerar um grafo não direcionado ponderado a partir deste e realizar investigações interessantes dentro do contexto de topologia de rede da Internet.

Importante. A definição da sintaxe de cada comando bem como sua saída devem seguir estritamente as especificações definidas em cada funcionalidade. Para

especificar a sintaxe de execução, considere que o programa seja chamado de "programaTrab". Essas orientações devem ser seguidas uma vez que a correção do funcionamento do programa se dará de forma automática. De forma geral, a primeira entrada da entrada padrão é sempre o identificador de suas funcionalidades, conforme especificado na definição das funcionalidades.

Modularização. É importante modularizar o código. Trechos de programa que aparecerem várias vezes devem ser modularizados em funções e procedimentos.

Descrição Específica. O programa deve oferecer as seguintes funcionalidades:

[11] Permita a recuperação dos dados, de todos os registros, armazenados em um arquivo de dados no formato binário e a geração de um grafo contendo esses dados na forma de um conjunto de vértices V e um conjunto de arestas A. O arquivo de dados no formato binário deve seguir o mesmo formato do arquivo de dados gerado no primeiro trabalho prático e não pode conter registros removidos. O grafo deve ser um grafo não direcionado ponderado e deve representar a topologia de rede da internet.

A representação do grafo deve ser na forma de listas de adjacências. As listas de adjacências consistem tradicionalmente em um vetor de |V| elementos que são capazes de apontar, cada um, para uma lista linear, de forma que o i-ésimo elemento do vetor aponta para a lista linear de arestas que são adjacentes ao vértice i.

Cada elemento do vetor deve representar um **idConecta**. Os vértices do vetor devem ser ordenados de forma crescente de acordo com o valor de idConecta. Se dois ou mais PoPs têm o mesmo valor de idConecta, eles são considerados o mesmo PoPs. Além do idConecta, cada elemento do vetor deve armazenar também os seguintes campos correspondentes: (i) nomePoPs; (ii) nomePais; (iii) siglaPais.

Cada elemento da lista linear representa uma aresta entre dois PoPs. Ou seja, cada elemento da lista linear representa uma aresta entre idConecta e idPoPsConectado. A aresta deve armazenar a velocidade em Gbps. Velocidades indicadas em Mbps devem ser transformadas em Gbps. Considere que 1 Mbps = 0.0009765625 Gbps e que 1 Gbps = 1024 Mbps (https://www.unitconverters.net/data-transfer/megabit-second-to-gigabit-second.htm). Os elementos de cada lista linear devem ser ordenados de forma crescente de acordo com idPoPsConectado.

Entrada do programa para a funcionalidade [11]:

`11 topologiaRede.bin`

onde:

- `topologiaRede.bin` é o arquivo de dados de entrada no formato binário, o qual foi gerado conforme as especificações descritas no primeiro trabalho prático.

Saída caso o programa seja executado com sucesso:

`A saída deve ser exibida na saída padrão da seguinte forma. Para cada elemento i do vetor e para cada elemento j da lista linear correspondente, deve ser exibido: idConecta do elemento i, nomePoPs do elemento i, nomePais do elemento i, siglaPais do elemento i, idPoPsConectado do elemento j e velocidade do elemento j em Mpbs. Deve ser deixado um espaço em branco depois de cada dado. Pule uma linha em branco ao finalizar cada elemento j.`

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

Exemplo de execução (são mostrados apenas alguns elementos)**:**

```
./programaTrab
11 topologiaRede.bin
Saída
22 Teresina Brazil BR 29 3Mpbs
...
73 Frankfurt Germany DE 75 10Mpbs
73 Frankfurt Germany DE 77 5Mpbs
73 Frankfurt Germany DE 98 10Mpbs
...
```

[12] Para facilitar a manutenção dos PoPs, é necessário definir grupos de pontos interligados ciclicamente. Essa funcionalidade possui como objetivo determinar quantos ciclos simples existem no grafo. Um ciclo simples é um ciclo em que nenhum vértice se repete, com exceção do primeiro e do último (ou seja, o primeiro vértice, chamado de vértice de origem, é o mesmo que o último vértice, chamado de vértice de destino).

Entrada do programa para a funcionalidade [12]:

`12 topologiaRede.bin`

onde:

- `topologiaRede.bin` é o arquivo de dados de entrada no formato binário, o qual foi gerado conforme as especificações descritas no primeiro trabalho prático.

Saída caso o programa seja executado com sucesso:

`Escreva primeiro a *string* "Quantidade de ciclos:" deixe um espaço em branco e em seguida escreva a quantidade de ciclos simples existentes no grafo. Caso o grafo seja acíclico, escreva 0.`

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

Exemplos de execução:

```
./programaTrab
12 topologiaRede.bin
Saída
Quantidade de ciclos: 3

./programaTrab
12 topologiaRede.bin
Saída
Quantidade de ciclos: 0
```

[13] Para fins de controle, é necessário saber qual o fluxo máximo de transmissão de dados entre PoPs com base em suas velocidades. Essa funcionalidade possui como objetivo determinar qual o fluxo máximo entre dois PoPs. A funcionalidade [13] deve ser executada *n* vezes seguidas.

Entrada do programa para a funcionalidade [13]:

`13 topologiaRede.bin n`

$\texttt{PoPsOrigem}_1$ $\texttt{PoPsDestino}_1$

$\texttt{PoPsOrigem}_2$ $\texttt{PoPs2Destino}_2$

`...`

$\texttt{PoPsOrigem}_n$ $\texttt{PoPs2Destino}_n$

onde:

- `topologiaRede.bin` é o arquivo de dados de entrada no formato binário, o qual foi gerado conforme as especificações descritas no primeiro trabalho prático.
- n é a quantidade de vezes que a funcionalidade deve ser realizada.
- `PoPsOrigem` é o valor de idConecta que representa um PoPs de origem.
- `PoPsDestino` é o valor de idConecta que representa um PoPs de destino.

Saída caso o programa seja executado com sucesso:

Para cada execução da funcionalidade, escreva a *string* "Fluxo máximo entre", deixe um espaço em branco, escreva o valor de PoPsOrigem, deixe um espaço em branco, escreva "e", deixe um espaço em branco, escreva o valor de PoPsDestino, escreva ":", deixe um espaço em branco, escreva o valor do fluxo máximo em Mpbs. Caso não seja possível identificar esse valor, escreva -1.

Mensagem de saída caso algum erro seja encontrado:

Falha na execução da funcionalidade.

Exemplo de execução:

```
./programaTrab
13 topologiaRede.bin 2
57 58
3 9
Saída
Fluxo máximo entre 57 e 58: 78Mpbs
Fluxo máximo entre 3 e 9: -1
```

[14] Para transmitir dados entre dois pontos, é necessário descobrir a velocidade entre esses pontos. Adicionalmente, pode ser necessário também passar por um terceiro ponto obrigatório que pertence a esse caminho, chamado de parada obrigatória. Essa funcionalidade possui como objetivo determinar o comprimento do caminho mínimo entre dois PoPs, sendo que existe a necessidade de se passar por um terceiro PoPs. A funcionalidade [14] deve ser executada *n* vezes seguidas. Nesta funcionalidade, entenda que caminho mínimo se refere à menor soma de velocidades disponíveis.

Entrada do programa para a funcionalidade [14]:

`14 topologiaRede.bin n`

$\texttt{PoPsOrigem}_1$ $\texttt{PoPsDestino}_1$ $\texttt{PoPsParada}_1$

$\texttt{PoPsOrigem}_2$ $\texttt{PoPs2Destino}_2$ $\texttt{PoPsParada}_2$

`...`

$\texttt{PoPsOrigem}_n$ $\texttt{PoPs2Destino}_n$ $\texttt{PoPsParada}_n$

onde:

- `topologiaRede.bin` é o arquivo de dados de entrada no formato binário, o qual foi gerado conforme as especificações descritas no primeiro trabalho prático.
- n é a quantidade de vezes que a funcionalidade deve ser realizada.
- `PoPsOrigem` é o valor de idConecta que representa um PoPs de origem.
- `PoPsDestino` é o valor de idConecta que representa um PoPs de destino.
- `PoPsParada` é o valor de idConecta que representa um PoPs de parada obrigatória.

Saída caso o programa seja executado com sucesso:

Para cada execução da funcionalidade, escreva a *string* "Comprimento do caminho entre", deixe um espaço em branco, escreva o valor de PoPsOrigem, deixe um espaço em branco, escreva "e", deixe um espaço em branco, escreva o valor de PoPsDestino, escreva "parando em", deixe um espaço em branco, escreva o valor de PoPsParada, escreva ":", deixe um espaço em branco, escreva o valor do caminho Mpbs. Caso não seja possível identificar esse valor, escreva -1.

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

Exemplo de execução (são mostrados apenas alguns elementos)**:**

```
./programaTrab
14 topologiaRede.bin 1
57 58 9
Saída
Comprimento do caminho entre 57 e 58 parando em 9: 78Mpbs
```

Restrições

As seguintes restrições têm que ser garantidas no desenvolvimento do trabalho.

[1] O arquivo de dados deve ser gravado em disco no **modo binário**, de acordo com as especificações da primeira parte do trabalho prático.

[2] Devem ser exibidos avisos ou mensagens de erro de acordo com a especificação de cada funcionalidade.

[3] O(s) aluno(s) que desenvolveu(desenvolveram) o trabalho prático deve(m) constar como comentário no início do código (i.e. NUSP e nome do aluno). Para trabalhos desenvolvidos por mais do que um aluno, não será atribuída nota ao aluno cujos dados não constarem no código fonte.

[4] Todo código fonte deve ser documentado. A **documentação interna** inclui, dentre outros, a documentação de procedimentos, de funções, de variáveis, de partes do código fonte que realizam tarefas específicas. Ou seja, o código fonte deve ser documentado tanto em nível de rotinas quanto em nível de variáveis e blocos funcionais.

[5] Devem ser exibidos avisos ou mensagens de erro de acordo com a especificação de cada funcionalidade.

[6] A implementação deve ser realizada usando as linguagens de programação C e C++. As funções das bibliotecas <stdio.h> devem ser utilizadas para operações relacionadas à escrita e leitura dos arquivos. A implementação não pode ser feita em qualquer outra linguagem de programação. O programa executará no [run.codes].

Fundamentação Teórica

Conceitos, características, algoritmos e implementações de grafos podem ser encontrados nos *slides* de sala de aula e também no livro *Projeto de Algoritmos*, de Nívio Ziviani.

Material para Entregar

Arquivo compactado. Deve ser preparado um arquivo .zip contendo:

- Código fonte do programa devidamente documentado.
- Makefile para a compilação do programa.
- Uma especificação da participação de cada integrante no desenvolvimento do trabalho. Isso deve ser feito logo no início do arquivo que contém a função main. Deve ser indicado, para cada integrante, seu número USP, seu nome completo e sua porcentagem de participação. A porcentagem de participação deve variar entre 0% e 100%. Por exemplo, o integrante desenvolveu 100% do que era esperado de sua parte, ou então o integrante desenvolveu 80% do que era esperado para a sua parte.
- Um vídeo gravado pelos integrantes do grupo, o qual deve ter, no máximo, 5 minutos de gravação. O vídeo deve explicar o trabalho desenvolvido. Ou seja, o grupo deve apresentar: cada funcionalidade e uma breve descrição de como a funcionalidade foi implementada. Todos os integrantes do grupo devem participar do vídeo, sendo que o tempo de apresentação dos integrantes deve ser balanceado. Ou seja, o tempo de participação de cada integrante deve ser aproximadamente o mesmo. O uso da webcam é obrigatório.

Instruções para fazer o arquivo makefile. No [run.codes] tem uma orientação para que, no makefile, a diretiva “all” contenha apenas o comando para compilar seu

programa e, na diretiva "run", apenas o comando para executá-lo. Assim, a forma mais simples de se fazer o arquivo makefile é:

```
all:
	gcc -o programaTrab *.c
run:
	./programaTrab
```

Lembrando que *.c já engloba todos os arquivos .c presentes no seu zip. Adicionalmente, no arquivo Makefile é importante se ter um *tab* nos locais colocados acima, senão ele pode não funcionar.

Instruções de entrega. A entrega deve ser feita via [run.codes]:

- página: https://run.codes/Users/login
- código de matrícula: **1YGS**

O vídeo gravado deve ser submetido por meio da página da disciplina no e-disciplinas, no qual o grupo vai informar o nome de cada integrante, o número do grupo e um link que contém o vídeo gravado. Ao submeter o link, verifique se o mesmo pode ser acessado. Vídeos cujos links não puderem ser acessados receberão nota zero.

Critério de Correção

Critério de avaliação do trabalho. Na correção do trabalho, serão ponderados os seguintes aspectos.

- Corretude da execução do programa.
- Atendimento às especificações do registro de cabeçalho e dos registros de dados.
- Atendimento às especificações da sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade.
- Qualidade da documentação entregue. A documentação interna terá um peso considerável no trabalho.

- Vídeo. Integrantes que não participarem da apresentação receberão nota 0 no trabalho correspondente.

Casos de teste no [run.codes]. Juntamente com a especificação do trabalho, serão disponibilizados 70% dos casos de teste no [run.codes], para que os alunos possam avaliar o programa sendo desenvolvido. Os 30% restantes dos casos de teste serão utilizados nas correções.

Restrições adicionais sobre o critério de correção.

- A não execução de um programa devido a erros de compilação implica que a nota final da parte do trabalho será igual a zero (0).
- O não atendimento às especificações de sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade implica que haverá uma diminuição expressiva na nota do trabalho.
- A ausência da documentação implica que haverá uma diminuição expressiva na nota do trabalho.
- A inserção de palavras ofensivas nos arquivos e em qualquer outro material entregue implica que a nota final da parte do trabalho será igual a zero (0).
- Em caso de plágio, as notas dos trabalhos envolvidos serão zero (0).

Data de Entrega do Trabalho

Na data especificada na página da disciplina.

Bom Trabalho !