Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Estrutura de Dados III (SCC0607)

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

Monitores
Eduardo Souza Rocha
eduardos.rocha17@usp.br ou telegram: @edwolt
Beatriz Aimee Teixeira Furtado Braga
beatriztfb@usp.br ou telegram: @bia_aimee
Heitor Tanoue de Mello
heitortanoue@usp.br ou *telegram: @heitortanoue*

Primeiro Trabalho Prático

Este trabalho tem como objetivo aprofundar conceitos relacionados a grafos.

O trabalho deve ser feito em duplas (ou seja, em 2 alunos). Os alunos devem ser os mesmos alunos do trabalho prático introdutório. Caso haja mudanças, elas devem ser informadas para a docente e os monitores. 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 direcionado ponderado a partir deste e realizar investigações interessantes dentro do contexto de ecossistemas tecnológicos e como as tecnologias estão relacionadas entre si.

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 a seguir.

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:

[8] 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 definido no trabalho introdutório. O grafo é um grafo direcionado ponderado e representa como as tecnologias estão relacionadas entre si, ou seja, a frequência que as pessoas que estão vendo uma tecnologia origem clicam para ver uma tecnologia destino a partir desta.

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 representa o nome de uma tecnologia. Os elementos devem ser ordenados de forma crescente de acordo com o nome da tecnologia. Se duas ou mais tecnologias têm o mesmo nome, elas são consideradas a mesma tecnologia. Além do nome da tecnologia, cada elemento do vetor também deve armazenar os seguintes campos: (i) *grupo*; (ii) *grau de entrada*; (iii) *grau de saída*; (iv) *grau*.

Cada elemento da lista linear representa uma aresta entre duas tecnologias. Ou seja, cada elemento da lista linear representa uma aresta entre o par (nomeTecOrigem, nomeTecDestino). A aresta é ponderada em termos do *peso*, sendo que a aresta (nomeTecOrigem, nomeTecDestino) sai do vértice nomeTecOrigem (origem) e chega no vértice nomeTecDestino (destino). Os elementos de cada lista linear devem ser ordenados de forma crescente de acordo com nomeTecDestino.

Entrada do programa para a funcionalidade [8]:

```
8 arquivoDados.bin
```

onde:

- `arquivoDados.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados do trabalho introdutório, e que contém dados desordenados e registros logicamente removidos.

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 em uma linha diferente: nomeTecnologia do elemento i, grupo do elemento i, grau de entrada do elemento i, grau de saída do elemento i, grau do elemento i, nomeTecnologia do elemento j e peso do elemento j.`

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

Exemplo de execução:

```
./programaTrab
8 tecnologias.bin
AZURE, 2, 5, 3, 8, .NET, 22
SQL-SERVER, 2, 0, 4, 4, .NET, 33
...
```

[9] Gere o grafo transposto do grafo gerado na funcionalidade [8]. Um grafo transposto $G^T = (V, A^T)$ de um grafo direcionado $G = (V, A)$ é um grafo que contém todos os vértices de G e cujas arestas são definidas da seguinte forma: $A^T = \{ (u, v) : (v, u) \in A \}$, ou seja, A^T consiste das arestas de G com suas direções invertidas.

A representação do grafo transposto 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 o nome de uma tecnologia. Os elementos ordenados de forma crescente de acordo com o nome da tecnologia. Se duas ou mais tecnologias têm o mesmo nome, elas são consideradas a mesma tecnologia. Além do nome da tecnologia, cada elemento do vetor também deve armazenar os seguintes campos: (i) *grupo*; (ii) *grau de entrada*; (iii) *grau de saída*; (iv) *grau*.

Cada elemento da lista linear representa uma aresta entre duas tecnologias. Ou seja, cada elemento da lista linear representa uma aresta entre o par (nomeTecDestino, nomeTecOrigem). A aresta é ponderada em termos do *peso*, sendo que a aresta (nomeTecDestino, nomeTecOrigem) sai do vértice nomeTecDestino (origem) e chega no vértice nomeTecOrigem (destino). Os elementos de cada lista linear devem ser ordenados de forma crescente de acordo com nomeTecOrigem.

Entrada do programa para a funcionalidade [9]:

```
9 arquivoDados.bin
```

onde:

- `arquivoDados.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados do trabalho introdutório, e que contém dados desordenados e registros logicamente removidos.

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 em uma linha diferente: nomeTecnologia do elemento i, grupo do elemento i, grau de entrada do elemento i, grau de saída do elemento i, grau do elemento i, nomeTecnologia do elemento j e peso do elemento j.`

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

Exemplo de execução:

```
./programaTrab
9 tecnologias.bin
.NET, 5, 3, 1, 4, AZURE, 22
.NET, 5, 2, 1, 1, SQL-SERVER, 33
...
```

[10] Dado o nome de uma tecnologia destino, liste o nome de todas as tecnologias que originaram o clique para essa tecnologia destino. A funcionalidade deve ser executada n vezes.

Entrada do programa para a funcionalidade [10]:

```
10 arquivoDados.bin n
nomeTecnologia1
nomeTecnologia2
...
nomeTecnologian
```

onde:

- `arquivoDados.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados do trabalho introdutório, e que contém dados desordenados e registros logicamente removidos.
- n é a quantidade de vezes que a funcionalidade deve ser realizada.
- `nomeTecnologia` é o nome de uma tecnologia, o qual é passado como parâmetro. Como esse parâmetro é do tipo *string*, deve ser aspas duplas (").

Saída caso o programa seja executado com sucesso:

Para cada execução da funcionalidade, a saída deve ser exibida na saída padrão da seguinte forma. Primeiro, deve ser escrito o nome da tecnologia passada como parâmetro, depois deve ser colocado o caractere ":". Depois, escreva cada nome de cada tecnologia que originou a tecnologia passada como parâmetro em ordem crescente, separando cada nome por vírgula e deixando um espaço em branco. Adicionalmente, pule uma linha entre cada execução da funcionalidade.

Mensagem de saída caso algum erro seja encontrado:

```
Falha na execução da funcionalidade.
```

Mensagem de saída caso não seja encontrada a tecnologia passada como parâmetro ou caso a tecnologia passada como parâmetro não tenha sido gerada por outra tecnologia:

```
Registro inexistente.
```

Exemplo de execução:

```
./programaTrab
10 tecnologias.bin 2
".NET"
"ANDROID"
.NET: AZURE, SQL-SERVER
pular uma linha em branco
ANDROID: IOS, ANDROID-STUDIO, JAVA
pular uma linha em branco
```

[11] Dado um grafo direcionado como entrada, determine se ele é fortemente conexo e quantos componentes fortemente conexos ele possui.

Entrada do programa para a funcionalidade [11]:

```
11 arquivoDados.bin
```

onde:

- `arquivoDados.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados do trabalho introdutório, e que contém dados desordenados e registros logicamente removidos.

Saída caso o programa seja executado com sucesso:

A saída deve ser exibida na saída padrão da seguinte forma. Caso o grafo seja fortemente conexo, deve ser escrita a saída "Sim, o grafo e fortemente conexo e possui 1 componente." Caso o grafo não seja fortemente conexo, deve ser escrita a saída "Nao, o grafo nao e fortemente conexo e possui x componentes.", sendo que x é o número de componentes fortemente conexos que o grafo possui.

Mensagem de saída caso algum erro seja encontrado:

Falha na execução da funcionalidade.

Exemplos de execução:

```
./programaTrab
11 tecnologias.bin
Sim, o grafo e fortemente conexo e possui 1 componente.


./programaTrab
11 tecnologias.bin
Nao, o grafo nao e fortemente conexo e possui 3 componentes.
```

[12] Dado o nome de uma tecnologia origem e uma tecnologia destino, determine o caminho mais curto entre essas duas tecnologias. A funcionalidade deve ser executada n vezes.

Entrada do programa para a funcionalidade [12]:

```
12 arquivoDados.bin n
nomeTecnologiaOrigem1 nomeTecnologiaDestino1
nomeTecnologiaOrigem2 nomeTecnologiaDestino2
...
nomeTecnologiaOrigemn nomeTecnologiaDestinon
```

onde:

- `arquivoDados.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados do trabalho introdutório, e que contém dados desordenados e registros logicamente removidos.
- n é a quantidade de vezes que a funcionalidade deve ser realizada.
- `nomeTecnologiaOrigem` é o nome de uma tecnologia, o qual é passado como parâmetro. Como esse parâmetro é do tipo *string*, deve ser aspas duplas (").
- `nomeTecnologiaDestino` é o nome de uma tecnologia, o qual é passado como parâmetro. Como esse parâmetro é do tipo *string*, deve ser aspas duplas (").

Saída caso o programa seja executado com sucesso:

Para cada execução da funcionalidade, escreva o valor de nomeTecnologiaOrigem, seguido do valor de nomeTecnologiaDestino, seguido do caractere ":", seguido pelo peso do caminho mais curto, conforme ilustrado no exemplo de execução. Quando não existir caminho entre a tecnologia origem e a tecnologia destino, a saída deve ser da seguinte forma: escreva o valor de nomeTecnologiaOrigem, seguido do valor de nomeTecnologiaDestino, seguido do caractere ":", seguido de CAMINHO INEXISTENTE.

Mensagem de saída caso algum erro seja encontrado:

Falha na execução da funcionalidade.

Exemplo de execução:

```
./programaTrab
12 tecnologias.bin 3
".NET" "DOCKER"
"ANDROID" "JAVA"
"JAVA" "XML"
.NET DOCKER: 47
ANDROID JAVA: 69
JAVA XML: CAMINHO INEXISTENTE
```

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**. O modo texto não pode ser usado.

[2] Os dados do registro descrevem os nomes dos campos, os quais não podem ser alterados. Ademais, todos os campos devem estar presentes na implementação, e nenhum campo adicional pode ser incluído. O tamanho e a ordem de cada campo deve obrigatoriamente seguir a especificação.

[3] Deve haver a manipulação de valores nulos, conforme as instruções definidas.

[4] Não é necessário realizar o tratamento de truncamento de dados.

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

[6] Os dados devem ser obrigatoriamente escritos campo a campo. Ou seja, não é possível escrever os dados registro a registro. Essa restrição refere-se à entrada/saída, ou seja, à forma como os dados são escritos no arquivo.

[7] 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.

[8] 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.

[9] A implementação deve ser realizada usando a linguagem de programação 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 e características dos diversos métodos para representar os conceitos de campo e de registro em um arquivo de dados podem ser encontrados nos *slides* de sala de aula e também no livro *File Structures* (*second edition*), de Michael J. Folk e Bill Zoellick.

Material para Entregar

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

- Código fonte do programa devidamente documentado.
- Makefile para a compilação do programa.
- 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.

O programa deve ser submetido via [run.codes]:

- página: https://runcodes.icmc.usp.br/
- Código de matrícula: **Z3BS**

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. Vídeos corrompidos ou que não puderem ser corretamente 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 do registro de cabeçalho e dos registros de dados implica que haverá uma diminuição expressiva na nota do trabalho.
- 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 realização do trabalho prático com alunos de turmas diferentes 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.
