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 Ciferri (cdac@icmc.usp.br)
alunos PAE
João Paulo Clarindo (jpcsantos@usp.br)
monitores
Helbert Moreira Pinto [helbert.moreira@usp.br] telegram: @HelbertMP
Matheus Carvalho Raimundo [mcarvalhor@usp.br] telegram: @mcarvalhor

Terceiro Trabalho Prático

Este trabalho tem como objetivo aprofundar conceitos relacionados a grafos.

De acordo com o critério de avaliação da disciplina, 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 direcionado a partir deste e realizar investigações interessantes dentro do contexto de redes sociais.

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.

Descrição Específica. O programa deve oferecer as seguintes funcionalidades:

[9] Permita a recuperação dos dados, de todos os registros, armazenados em arquivos 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. Os arquivos de dados no formato binário devem seguir os mesmos formatos dos arquivos de dados gerados no primeiro trabalho prático e no segundo trabalho prático e pode conter (ou não) registros removidos. O grafo deve ser um grafo direcionado não ponderado e deve representar quem segue quem (ou seja, qual pessoa segue qual pessoa).

A representação do grafo deve, obrigatoriamente, 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 pessoa que segue**. Os vértices do vetor devem ser ordenados de forma crescente de acordo com o nome da pessoa. Não existem duas pessoas com o mesmo nome. Ademais, em grafos não ponderados, cada elemento da lista linear armazena o rótulo do vértice. O rótulo do vértice é um **nome de uma pessoa que é seguida**. Os elementos de cada lista linear devem ser ordenados de forma crescente de acordo com o nome da pessoa que é seguida.

Entrada do programa para a funcionalidade [9]:

`9 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin`

onde:

- `arquivoPessoa.bin` é o arquivo de dados **pessoa** no formato binário gerado conforme as especificações descritas no primeiro trabalho prático.

- `arquivoIndexaPessoa.bin` é o arquivo de índice **indexaPessoa** gerado conforme as especificações descritas no primeiro trabalho prático.

- `arquivoSegueOrdenado.bin` é o arquivo de dados **segue** no formato binário gerado pela funcionalidade [7] conforme as especificações descritas no segundo trabalho prático, o qual encontra-se ordenado de acordo o campo *idPessoaQueSegue*.

Saída caso o programa seja executado com sucesso:

A saída deve ser exibida na saída padrão da seguinte forma. Em cada linha, deve ser mostrado primeiro o elemento do vetor na posição i e depois a lista linear correspondente. Os elementos da lista linear devem ser exibidos de forma crescente de acordo com o seu rótulo. Deve haver uma vírgula e um espaço em branco entre cada saída mostrada na saída padrão.

Exemplo com metadados:

$nomePessoaQueSegue_1$, $nomePessoaQueESeguida_{11}$, $nomePessoaQueESeguida_{12}$, ..., $nomePessoaQueESeguida_{1n}$

...

$nomePessoaQueSegue_m$, $nomePessoaQueESeguida_{m1}$, $nomePessoaQueESeguida_{m2}$, ..., $nomePessoaQueESeguida_{mp}$

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

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

```
./programaTrab
9 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin
MARIA DO NASCIMENTO, CAMILA ALVES, ...
...
SAMANTHA PEREIRA SANTOS, ADRIANA PEREIRA SILVA, VITÓRIA PRADO CAMPOS,
...
```

[10] Gere o grafo transposto do grafo gerado na funcionalidade [9]. 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 deve, obrigatoriamente, 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 pessoa que é seguida**. Os vértices do vetor devem ser ordenados de forma crescente de acordo com o nome da pessoa que é seguida. Não existem duas pessoas com o mesmo nome. Ademais, em grafos não ponderados, cada elemento da lista linear armazena o rótulo do vértice. O rótulo do vértice é um **nome de uma pessoa que segue**. Os elementos de cada lista linear devem ser ordenados de forma crescente de acordo com o nome da pessoa que segue.

Entrada do programa para a funcionalidade [10]:

`10 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin`

onde:

- `arquivoPessoa.bin` é o arquivo de dados **pessoa** no formato binário gerado conforme as especificações descritas no primeiro trabalho prático.

- `arquivoIndexaPessoa.bin` é o arquivo de índice **indexaPessoa** gerado conforme as especificações descritas no primeiro trabalho prático.

- `arquivoSegueOrdenado.bin` é o arquivo de dados **segue** no formato binário gerado pela funcionalidade [7] conforme as especificações descritas no segundo trabalho prático, o qual encontra-se ordenado de acordo o campo *idPessoaQueSegue*.

Saída caso o programa seja executado com sucesso:

A saída deve ser exibida na saída padrão da seguinte forma. Em cada linha, deve ser mostrado primeiro o elemento do vetor na posição i e depois a lista linear correspondente. Os elementos da lista linear devem ser exibidos de forma crescente de acordo com o seu rótulo. Deve haver uma vírgula e um espaço em branco entre cada saída mostrada na saída padrão.

Exemplo com metadados:

nomePessoaQueÉSegueguida$_1$, nomePessoaQueSegue$_{11}$, nomePessoaQueSegue$_{12}$, ..., nomePessoaQueSegue$_{1n}$

...

nomePessoaQueÉSeguida$_m$, nomePessoaQueESegue$_{m1}$, nomePessoaQueSegue$_{m2}$, ..., nomePessoaQueSegue$_{mp}$

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

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

```
./programaTrab
10 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin
ADRIANA PEREIRA SILVA, SAMANTHA PEREIRA SANTOS, ...
CAMILA ALVES, MARIA DO NASCIMENTO, ...
...
VITÓRIA PRADO CAMPOS, SAMANTHA PEREIRA SANTOS, ...
```

[11] Determine o caminho mais curto entre cada **pessoa que segue** e uma determinada **celebridade** definida como parâmetro de entrada. Essa funcionalidade consiste em determinar os caminhos mais curtos com destino único. Para resolver este problema, ele tem que ser reduzido ao problema de determinar os caminhos mais curtos de origem única. Isso é feito invertendo a direção de cada aresta do grafo (ou seja, calculando o grafo transposto) e iniciando a busca pelo vértice destino, que passa então a ser o vértice origem no grafo transposto. Portanto, utilize o grafo transposto G^T gerado na funcionalidade [10] e faça uma busca em largura. Durante a execução do algoritmo, deve ser calculado o vetor de antecessores, para que o caminho mais curto possa ser listado na saída padrão. Considere que, durante a execução do algoritmo, quando houver mais do que um vértice para ser inserido na fila **q**, deve-se inserir esses vértices em ordem alfabética (do menor nome para o maior nome). O índice do vetor de antecessores deve ser ordenado de forma crescente de acordo com o nome da pessoa.

Entrada do programa para a funcionalidade [11]:

```
11 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin "nomeCelebridade"
```

onde:

- `arquivoPessoa.bin` é o arquivo de dados **pessoa** no formato binário gerado conforme as especificações descritas no primeiro trabalho prático.

- `arquivoIndexaPessoa.bin` é o arquivo de índice **indexaPessoa** gerado conforme as especificações descritas no primeiro trabalho prático.

- `arquivoSegueOrdenado.bin` é o arquivo de dados **segue** no formato binário gerado pela funcionalidade [7] conforme as especificações descritas no segundo trabalho prático, o qual encontra-se ordenado de acordo o campo *idPessoaQueSegue*.

- `nomeCelebridade` é o nome da celebridade passado como parâmetro. Ele é representado entre aspas duplas (") por ser do tipo *string*.

Saída caso o programa seja executado com sucesso:

A saída deve ser exibida na saída padrão da seguinte forma. Para cada pessoa que segue, escreva primeiro o seu nome em uma linha e depois escreva o nome de todas as pessoas que ela segue, sendo o nome da última pessoa o nome da celebridade passado como parâmetro. Deve haver uma vírgula e um espaço em branco entre cada saída mostrada na saída padrão. Caso a pessoa que segue não seguir a celebridade, então deve ser escrito "NAO SEGUE A CELEBRIDADE".

Exemplo com metadados:

$nomePessoaQueSegue_1$, $nomePessoaQueESeguida_{11}$, $nomePessoaQueESeguida_{12}$, ..., $nomePessoaQueESeguida_{1n}$, nomeCelebridade

...

$nomePessoaQueSegue_m$, $nomePessoaQueESeguida_{m1}$, $nomePessoaQueESeguida_{m2}$, ..., $nomePessoaQueESeguida_{mp}$, nomeCelebridade

Mensagem de saída caso algum erro seja encontrado:

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

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

```
./programaTrab
11 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin "MARIANA MARIANA"
MARIA DO NASCIMENTO, CAMILA ALVES, ..., MARIANA MARIANA
...
```

SAMANTHA PEREIRA SANTOS, ADRIANA PEREIRA SILVA, VITÓRIA PRADO CAMPOS,..., MARIANA MARIANA

TÂNIA MARIA ALVES, NAO SEGUE A CELEBRIDADE

[12] Determine o comprimento do caminho para que uma determinada pessoa definida como parâmetro de entrada ouça a fofoca que ela mesmo contou. Essa funcionalidade consiste em determinar se existe um ciclo no grafo, ou seja, se existe um caminho no qual o primeiro e o último vértices são iguais. No caso dessa funcionalidade, o primeiro e o último vértices devem ser correspondentes ao nome da pessoa que foi passado como parâmetro de entrada. Durante a execução do algoritmo de busca em profundidade sobre o grafo G, quando houver mais do que um vértice a ser analisado, deve ser escolhido o vértice cujo nome da pessoa seja o menor. Outro aspecto é que, mesmo que existam vários ciclos relacionados à pessoa definida como parâmetro de entrada, apenas a resposta relativa ao primeiro ciclo deve ser listada na saída padrão.

Entrada do programa para a funcionalidade [12]:

```
12 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin "nomePessoaQueGerouFofoca"
```

onde:

- `arquivoPessoa.bin` é o arquivo de dados **pessoa** no formato binário gerado conforme as especificações descritas no primeiro trabalho prático.

- `arquivoIndexaPessoa.bin` é o arquivo de índice **indexaPessoa** gerado conforme as especificações descritas no primeiro trabalho prático.

- `arquivoSegueOrdenado.bin` é o arquivo de dados **segue** no formato binário gerado pela funcionalidade [7] conforme as especificações descritas no segundo trabalho prático, o qual encontra-se ordenado de acordo o campo *idPessoaQueSegue*.

- `nomePessoaQueGerouFofoca` é o nome da pessoa que gerou a fofoca. Ele é representado entre aspas duplas (") por ser do tipo *string*.

Saída caso o programa seja executado com sucesso:

```
A saída deve ser exibida na saída padrão da seguinte forma. Deve ser exibido o comprimento do caminho caso a fofoca retorne para a pessoa ou "A FOFOCA NAO RETORNOU".
```

Mensagem de saída caso algum erro seja encontrado:

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

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

```
./programaTrab
12 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin "MARIANA MARIANA"
5

./programaTrab
12 arquivoPessoa.bin arquivoIndexaPessoa.bin arquivoSegueOrdenado.bin "MARIA DA SILVA PINHEIRO"
A FOFOCA NAO RETORNOU
```

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] 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, 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.

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: **V9DB**

Critério de Correção

Importante. Durante o desenvolvimento deste trabalho prático, serão disponibilizados no [run.codes] 70% dos casos de testes a serem considerados na nota de execução do programa. Os 30% restantes serão adicionados após a entrega dos trabalhos e serão contabilizados na nota de execução do programa.

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.

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 !