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

Docente
Prof. Anderson Canale Garcia
andersongarcia@usp.br

Monitores
Beatriz Aimée Teixeira Furtado Braga
beatrizatfb@usp.br ou telegram: @bia_aimee
Gustavo Lelli
gustavo.lelli@usp.br ou *telegram: @gustavo_lelli*
Lucas Piovani
lucaspiovani@usp.br ou telegram: @lucaspiovanii
Rafael Freitas Garcia
rafaelfreitasgarcia@usp.br ou *telegram: @rafaelfrgc*

Terceiro Trabalho Prático

Este trabalho tem como objetivo aprofundar conceitos relacionados a grafos.

O trabalho deve ser feito em duplas (ou seja, em 2 alunos). 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 biológicos como os organismos 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:

[10] 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, e não contém arquivos removidos. O grafo é um grafo direcionado ponderado e representa a cadeia alimentar.

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 um organismo. Os elementos devem ser ordenados de forma crescente de acordo com o nome. Se dois ou mais organismos têm o mesmo nome, elas são consideradas o mesmo organismo. Além do nome, cada elemento do vetor também deve armazenar os seguintes campos: *(i) espécie; (ii) habitat; (iii) dieta; (iv) tipo; (v) grau de entrada; (vi) grau de saída; (vii) grau.*

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

Entrada do programa para a funcionalidade [10]:

```
10 arquivoDados.bin
```

onde:

-arquivosDados.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: nome do elemento i, espécie do elemento i, habitat do elemento i, dieta do elemento i, tipo do elemento i, grau de entrada do elemento i, grau de saída do elemento i, grau do elemento i, alimento do elemento j e população do elemento j. 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 no processamento do arquivo.
```

Exemplo de execução:

```
./programaTrab
10 dinossauros.bin
aardonyx,  celestae, South Africa, herbivorous, sauropod, 2, 3, 5, Equisetum arvense, 6

abelisaurus, comahuensis, Argentina, carnivorous, large theropod, 1, 2, 3, abelisaurus, 4

…
```

[11] Dado o nome de uma presa, liste o nome de todos os predadores que a consome. A funcionalidade deve ser executada n vezes.

Entrada do programa para a funcionalidade [11]:

```
ll arquivoDados.bin
   presa1
   presa2
  _
   presa n
```

onde:

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

-presa é o nome do alimento, dado 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 presa passada como parâmetro, depois deve ser colocado o caractere ":". Depois, escreva cada nome de cada predador 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 no processamento do arquivo.

Mensagem de saída caso não seja encontrada a presa passada como parâmetro ou caso a presa passada como parâmetro não tenha predador:

Registro inexistente.

Exemplo de execução:

```
./programaTrab
ll dinossauros.bin 1
```

Saída

```
"Gallimimus"
Gallimimus: Velociraptor, Tiranossauro
pular uma linha em branco
```

[12] Esta funcionalidade tem como objetivo identificar quantos ciclos simples existem no grafo da cadeia alimentar. 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). Em termos da cadeia alimentar, isso pode representar situações onde uma espécie, de alguma forma, depende de si mesma, direta ou indiretamente, o que pode afetar o equilíbrio ecológico do sistema.

Entrada do programa para a funcionalidade [12]:

```
12 arquivoDados.bin
```

onde:

-arquivosDados.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 aciclico, escreva 0.

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento do arquivo.
```

Exemplo de execução:

```
./programaTrab
12 dinossauros.bin
```

Saída

```
Quantidade de ciclos: 3
```

```
./programaTrab
12 dinossauros.bin
```

Saída

```
Quantidade de ciclos: 0
```

[13] Dado um grafo direcionado que representa a cadeia alimentar, verifique se ele é fortemente conexo e quantos componentes fortemente conexos ele possui. No contexto da cadeia alimentar, isso ajuda a visualizar a resiliência do ecossistema, indicando se todas as espécies estão conectadas diretamente ou indiretamente.

Entrada do programa para a funcionalidade [13]:

```
13 arquivoDados.bin
```

onde:

-arquivosDados.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. 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 no processamento do arquivo.
```

Exemplo de execução:

```
./programaTrab
13 dinossauros.bin
```

Saida

```
Sim, o grafo e fortemente conexo e possui 1 componente.
```

```
./programaTrab
13 dinossauros.bin
```

Saida

```
Nao, o grafo nao e fortemente conexo e possui 3 componentes.
```

[14] Esta funcionalidade tem como objetivo analisar a relação entre uma presa e um predador na cadeia alimentar, determinando o caminho de menor população de predadores entre as duas espécies, representando o menor risco que a presa enfrenta ao longo da cadeia alimentar. A funcionalidade deve ser executada n vezes.

Entrada do programa para a funcionalidade [14]:

```
14 arquivoDados.bin n
nomePredador1 alimento1
nomePredador2 alimento2
 ...
nomePredador_n alimento_n
```

onde:

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

=nomePredador é o nome do dinossauro, dado como parâmetro. Como esse parâmetro é do tipo string, deve ser aspas duplas (").

=alimento é o nome do alimento, dado 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 nomePredador, seguido do valor de nomeAlimento, seguido do caractere ":", seguido pelo peso do caminho mais curto, conforme ilustrado no exemplo de execução. Quando não existir caminho entre o predador e a presa, a saída deve ser da seguinte forma: escreva o valor de nomePredador, seguido do valor de alimento, seguido do caractere ":", seguido de CAMINHO INEXISTENTE.

Mensagem de saída caso algum erro seja encontrado:

Falha no processamento do arquivo.

Exemplo de execução:

```
./programaTrab
14 dinossauro.bin 2
"Velociraptor" "Gallimimus"
"Velociraptor" "Mosassauro"
Velociraptor Gallimimus: 69
Velociraptor Mosassauro: 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 ou C++. As funções das bibliotecas <stdio.h> devem ser utilizadas para operações relacionadas à escrita e leitura dos arquivos. Outras bibliotecas podem ser utilizadas. 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.
- 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 de entrega.

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

- código de matrícula: K82G

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. Não deixem o vídeo em “Comentários”. Envie um arquivo txt.

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 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).

Bom Trabalho!