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)
aluno PAE
João Paulo Clarindo (jpcsantos@usp.br)
colaborador
Matheus Carvalho Raimundo (mcarvalhor@usp.br)

Segunda Parte do Trabalho Prático
valor: 40%

Este trabalho tem como objetivo aprofundar conceitos relacionados a grafos.

O trabalho deve ser feito em, no máximo, 3 alunos. Os alunos devem ser os mesmos da primeira parte do trabalho prático. Caso haja a necessidade de mudança de grupo, a docente responsável deve ser consultada. 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 e possa aplicar o algoritmo de Dijkstra e o algoritmo de Prim.

Linguagem de Programação. Qualquer linguagem de programação aceita pelo [run.codes] pode ser usada para desenvolver o trabalho. Entretanto, é fortemente recomendado o uso das linguagens C e C++.

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 "programaTrab2". 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 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 na primeira parte do trabalho prático, e pode conter (ou não) registros removidos. 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 representa um nome da cidade e seu estado. Os vértices do vetor devem ser ordenados de forma crescente de acordo com o nome da cidade. Ademais, em grafos ponderados, cada elemento da lista linear armazena o rótulo do vértice e o peso da aresta correspondente. O rótulo do vértice representa um nome da cidade e seu estado e cada peso da aresta representa uma distância e um tempo de viagem. Os elementos de cada lista linear devem ser ordenados de forma crescente de acordo com o nome da cidade.

Entrada do programa para a funcionalidade [9]:

`9 arquivoBinarioEntrada.bin`

onde:

- `arquivoBinarioEntrada.bin` é um arquivo binário gerado conforme as especificações descritas na primeira parte do trabalho prático.

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 um espaço em branco entre cada saída mostrada na saída padrão.

Exemplo com metadados:

cidadeOrigem estadoOrigem cidadeDestino estadoDestino distancia tempoViagem cidadeDestino estadoDestino distancia tempoViagem ... cidadeDestino estadoDestino distancia tempoViagem

...

cidadeOrigem estadoOrigem cidadeDestino estadoDestino distancia tempoViagem cidadeDestino estadoDestino distancia tempoViagem ... cidadeDestino estadoDestino distancia tempoViagem

Mensagem de saída caso algum erro seja encontrado:

Falha na execução da funcionalidade.

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

```
./programaTrab2
9 arquivoBinarioEntrada.bin
ARARAQUARA SP LIMEIRA SP 128 1h 27min SAO PAULO SP 288 2h 53min ...
...
BONITO MS NITEROI RJ 1617 17h 52min VITORIA ES 2115 24h 0min ...
```

[10] Determine o caminho mais curto de origem única usando o algoritmo de Dijkstra, considerando uma cidade de origem determinada pelo usuário. Durante a execução do algoritmo, devem ser calculados: o vetor D de distâncias, o vetor T de tempos de viagem e o vetor ANT de antecessores. Considere que, durante a execução do algoritmo: (i) em situação de empate na escolha de vértices, deve ser escolhido o vértice de menor valor; e (ii) em situação de empate na escolha de valores para o vetor D de distâncias, o valor já presente no vetor D deve permanecer. Os índices dos vetores devem ser ordenados de forma crescente de acordo com o nome da cidade. Lembre-se também de armazenar os estados das cidades.

Entrada do programa para a funcionalidade [10]:

```
10 cidadeOrigem "valorCampo"
```

onde:

- cidadeOrigem é o nome do campo
- valorCampo é o valor do campo. 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. Considere que i representa o índice dos vetores (ou seja, os nomes das cidades e seus estados). Para cada índice i dos vetores, devem ser exibidos em uma linha: a cidadeOrigem, o estadoOrigem, i (ou seja, a cidadeDestino e o estadoDestino), o valor de D[i], o valor de T[i] e o valor de ANT[i]. Deve haver um espaço em branco entre cada saída mostrada na saída padrão.

Exemplo com metadados:

```
cidadeOrigem estadoOrigem cidadeDestino estadoDestino distanciaCalculada tempoViagemCalculado cidadeAnterior estadoAnterior
...
cidadeOrigem estadoOrigem cidadeDestino estadoDestino distanciaCalculada tempoViagemCalculado cidadeAnterior estadoAnterior
```

Mensagem de saída caso não exista a cidade de origem solicitada:

```
Cidade inexistente.
```

Mensagem de saída caso algum erro seja encontrado:

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

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

```
./programaTrab2
10 cidadeOrigem "Araraquara"
ARARAQUARA SP LIMEIRA SP 128 1h 28min ARARAQUARA SP
...
ARARAQUARA SP SAO PAULO SP 257 2h 13min LIMEIRA SP
```

[11] Determine a árvore geradora mínima usando o algoritmo de Prim, considerando uma cidade de origem determinada pelo usuário. Considere que, em situação de empate entre o peso de duas arestas (u_1,v_1) e (u_2,v_2), deve ser escolhida a aresta cujo valor de u seja o menor. Caso haja situação de empate entre u_1 e u_2, deve ser escolhida a aresta cujo valor de v seja o menor. 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 um nome da cidade e seu estado. Os vértices do vetor devem ser ordenados de forma crescente de acordo com o nome da cidade. Ademais, em grafos ponderados, cada elemento da lista linear armazena o rótulo do vértice e o peso da aresta correspondente. O rótulo do vértice representa um nome da cidade e seu estado e cada peso da aresta representa uma distância e um tempo de viagem. Os elementos de cada lista linear devem ser ordenados de forma crescente de acordo com o nome da cidade.

Sintaxe do comando para a funcionalidade [11]:

```
11 cidadeOrigem "valorCampo"
```

onde:

- cidadeOrigem é o nome do campo
- valorCampo é o valor do campo. 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. 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 um espaço em branco entre cada saída mostrada na saída padrão.

Exemplo com metadados:

```
cidadeOrigem estadoOrigem cidadeDestino estadoDestino distancia tempoViagem cidadeDestino estadoDestino distancia tempoViagem ... cidadeDestino estadoDestino distancia tempoViagem
...
cidadeOrigem estadoOrigem cidadeDestino estadoDestino distancia tempoViagem cidadeDestino estadoDestino distancia tempoViagem ... cidadeDestino estadoDestino distancia tempoViagem
```

Mensagem de saída caso não exista a cidade de origem solicitada:

```
Cidade inexistente.
```

Mensagem de saída caso algum erro seja encontrado:

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

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

```
./programaTrab2
11 cidadeOrigem "Araraquara"
ARARAQUARA SP LIMEIRA SP 128 1h 27min SAO PAULO SP 288 2h 53min ...
...
BONITO MS NITEROI RJ 1617 17h 52min VITORIA ES 2115 24h 0min ...
```

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 uma linguagem de programação aceita pelo [run.codes]. Entretanto, é fortemente recomendado o uso das linguagens C ou C++. 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 programaTrab1 *.c
run:
./programaTrab2
```

Lembrando que *.c já engloba todos os arquivos .c presentes no seu zip.

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

- página: https://run.codes/Users/login
- código de matrícula: **L31R**

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.

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 !