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
Gabriel Vicente Rodrigues
gabriel_vr@usp.br ou telegram:@ga_vr
Lucas de Medeiros Franca Romero
lucasromero@usp.br ou telegram: @lucasromero

Voluntário
João Paulo Clarindo
jpcsantos@usp.br

Segundo 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 direcionado ponderado a partir deste e realizar investigações interessantes dentro do contexto de estações e linhas do metrô e da CPTM (Companhia Paulista de Trens Metropolitanos) da região metropolitana da cidade de São Paulo (SP).

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:

[7] 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 pode conter (ou não) registros removidos. O grafo deve ser um grafo direcionado ponderado e deve representar as estações e as linhas que ligam essas estações com as suas respectivas distâncias.

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 estação**. Um exemplo de nome de estação é “Luz”. Os vértices do vetor devem ser ordenados de forma crescente de acordo com o nome da estação. Note que, segundo a especificação do primeiro trabalho prático, se duas ou mais estações têm o mesmo nome, elas são consideradas a mesma estação.

Cada elemento da lista linear representa uma aresta entre duas estações e deve armazenar: (i) o nome da próxima estação; (ii) a distância para a próxima estação; e (iii) os nomes de linha associados. Considerando o elemento do vetor com nome de estação igual a “Luz”, um exemplo de elemento da lista linear referente a essa estação é: (i) nome da próxima estação = “Sao Bento”; (ii) distância para a próxima estação = 762; e (iii) nome da linha = “Azul”. Neste primeiro exemplo, existe apenas um único registro no arquivo de dados de forma que *aresta (u,v) = (Luz, Sao Bento)*. Ainda considerando o elemento do vetor com nome de estação igual a “Luz”, outro exemplo de elemento da lista linear é: (i) nome da próxima estação = “Bras”; (ii) distância para

a próxima estação = 2310; e (iii) nomes das linhas = “Coral”, “Jade”, “Rubi”. Neste segundo exemplo, existem três registros no arquivo de dados de forma que *aresta (u,v) = (Luz, Bras)*.

Os elementos que devem ser inseridos nas listas lineares devem ser aqueles que atendem aos seguintes requisitos:

- O elemento representa uma aresta entre um par (codEstacao, codProxEstacao), sendo que o nome da estação referente a codProxEstacao deve ser obtido fazendo-se uma pesquisa no arquivo de dados. Neste caso, a distância entre as estações e o nome da linha encontram-se no registro do arquivo de dados referente a codEstacao.
- O elemento representa uma aresta entre um par (codEstacao, codEstacaoIntegra), sendo que o nome da estação referente a codEstacaoIntegra deve ser obtido fazendo-se uma pesquisa no arquivo de dados. A aresta somente deve ser inserida se o nome da estação referente a codEstacao for diferente do nome da estação referente a codEstacaoIntegra. Neste caso, a distância entre as estações deve armazenar o valor zero e o nome da linha deve armazenar o valor “Integração”.

Os elementos de cada lista linear devem ser ordenados de forma crescente de acordo com o nome da estação (quer o nome represente uma próxima estação ou uma estação de integração). Em situações nas quais existem vários nomes de linhas armazenados em um mesmo elemento, então os nomes de linha deve ser ordenados de forma crescente.

Entrada do programa para a funcionalidade [7]:

`7 estacao.bin`

onde:

- `estacao.bin` é o arquivo de dados **estacao** no formato binário 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. 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 nome da estação. Quando necessário, os nomes das linhas devem ser exibidos de forma crescente. Deve haver uma vírgula e um espaço em branco entre cada saída mostrada na saída padrão.

Exemplo com metadados:

$nomeEstacao_1$, $nomeProxEstacao_{11}$, $distProxEstacao_{11}$, $nome(s)Linha(s)_{11}$, ..., $nomeProxEstacao_{1n}$, $distProxEstacao_{1n}$

...

$nomeEstacao_m$, $nomeProxEstacao_{m1}$, $distProxEstacao_{m1}$, $nome(s)Linha(s)_{m1}$, ..., $nomeProxEstacao_{mp}$, $distProxEstacao_{mp}$

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

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

```
./programaTrab
7 estacao.bin
Clinicas Consolacao 1063 Verde
...
Luz Bras 2310 Coral Jade Rubi Republica 1257 Amarela Sao Bento 762 Azul
...
```

[8] Problema real: Você está em uma **estação origem** e, como está com pressa, quer identificar qual o caminho mais curto para chegar em uma **estação destino**.

A solução para esse problema consiste em determinar o caminho mais curto de origem única usando o algoritmo de Dijkstra, considerando uma estação de origem e uma estação de destino, ambas passadas como parâmetro pelo usuário. Durante a execução do algoritmo, devem ser calculados: o vetor D de distâncias 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 cujo valor do nome da estação seja o menor; 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 estação.

Entrada do programa para a funcionalidade [8]:

`8 estacao.bin nomeEstacaoOrigem valorOrigem nomeEstacaoDestino valorDestino`

onde:

- `estacao.bin` é um arquivo binário gerado conforme as especificações descritas no primeiro trabalho prático.
- `nomeEstacaoOrigem` é o nome do campo.
- `valorOrigem` é o valor passado como entrada para o `nomeEstacaoOrigem`. Deve ser colocado aspas duplas por se tratar de uma *string*.
- `nomeEstacaoDestino` é o nome do campo.
- `valorDestino` é o valor passado como entrada para o `nomeEstacaoDestino`. Deve ser colocado aspas duplas por se tratar de uma *string*.

Saída caso o programa seja executado com sucesso:

A saída deve ser exibida na saída padrão da seguinte forma. Primeiramente, escrever em uma linha o número de estações que serão percorridas. Esse número de estações não deve considerar a estação de origem, mas deve considerar a estação de destino. Depois, escrever em outra linha a distância que será percorrida. Na sequência, escrever em outra linha os valores para os seguintes campos: o nome da estação de origem, os nomes das estações intermediárias e o nome da estação de destino. Devem ser escritos uma vírgula e um espaço em branco entre cada valor.

Exemplo com metadados:

```
Numero de estacoes que serao percorridas: x
Distancia que sera percorrida: y
nomeEstOrigem, nomeEstacao1, nomeEstacao2, ..., nomeEstDestino
```

OBS: o valor de x não inclui a estação de origem, mas inclui a estação de destino

Mensagem de saída caso não exista um caminho entre as estações solicitadas:

`Não existe caminho entre as estações solicitadas.`

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

Exemplo de execução:

```
./programaTrab
8 estacao.bin nomeEstacaoOrigem "Luz" nomeEstacaoDestino "Oscar Freire"
Numero de estacoes que serao percorridas: 4
Distancia que sera percorrida: 4790
Luz, Republica, Higienopolis-Mackenzie, Paulista, Oscar Freire
```

[9] Problema real: Você está conhecendo uma nova cidade e gostaria de saber se é possível sair de uma **estação origem** e, ao final de seu passeio, voltar para a mesma estação origem.

A solução para esse problema 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 estação de origem que foi passado como parâmetro de entrada. Durante a execução do algoritmo, devem ser calculados: o vetor D de distâncias e o vetor ANT de antecessores. Considere que, durante a execução do algoritmo de busca em profundidade: (i) em situação de empate na escolha de vértices, deve ser escolhido o vértice cujo valor do nome da estação seja o menor; (ii) em situação de empate na escolha de valores para o vetor D de distâncias, o valor já presente no vetor D deve permanecer; (iii) mesmo que existam vários ciclos relacionados à estação definida como origem, apenas a resposta relativa ao primeiro ciclo deve ser listada na saída padrão. Os índices dos vetores devem ser ordenados de forma crescente de acordo com o nome da estação.

Entrada do programa para a funcionalidade [9]:

`9 estacao.bin nomeEstacaoOrigem valorOrigem`

onde:

- `estacao.bin` é um arquivo binário gerado conforme as especificações descritas no primeiro trabalho prático.

- `nomeEstacaoOrigem` é o nome do campo.

- `valorOrigem` é o valor passado como entrada para o `nomeEstacaoOrigem`. Deve ser colocado aspas duplas por se tratar de uma *string*.

Saída caso o programa seja executado com sucesso:

A saída deve ser exibida na saída padrão da seguinte forma. Primeiramente, escrever em uma linha o número de estações que serão percorridas. Esse número de estações não deve considerar a estação de origem, mas deve considerar a estação de destino. Depois, escrever em outra linha a distância que será percorrida. Na sequência, escrever em outra linha os valores para os seguintes campos: o nome da estação de origem, os nomes das estações intermediárias e o nome da estação de destino. Devem ser escritos uma vírgula e um espaço em branco entre cada valor.

Exemplo com metadados:

```
Numero de estacoes que serao percorridas: x
Distancia que sera percorrida: y
nomeEstOrigem, nomeEstacao1, nomeEstacao2, ..., nomeEstOrigem
```

OBS: o valor de x não inclui a estação de origem de partida, mas inclui a estação de origem referente ao retorno do ciclo.

Mensagem de saída caso não exista um ciclo a partir da estação de origem:

`Nao existe um ciclo a partir da estacao de origem.`

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

Exemplo de execução:

```
./programaTrab
9 estacao.bin nomeEstacaoOrigem "Paraiso"
Numero de estacoes que serao percorridas: 2
Distancia que sera percorrida: 2419
Paraiso, Brigadeiro, Paraiso
```

[10] Problema real: Você foi contratado como um gestor de uma cidade inteligente e, como primeira atividade, ficou responsável por melhorar o projeto da malha de metrô da cidade. Nessa melhoria, foi solicitado que você gastasse a menor quantia possível para projetar novamente a malha, mas garantindo que as pessoas possam sair de qualquer estação e chegar em qualquer outra estação. Você também foi informado que as linhas podem ser de ida e volta, indicando que se existe uma linha entre uma estação 1 e uma estação 2, as pessoas podem ir da estação 1 para a estação 2 e vice-versa. Para desenvolver o seu projeto, foi passado a você a malha de metrô existente, com suas estações, linhas e distâncias existentes.

A solução para esse problema consiste em determinar a árvore geradora mínima usando o algoritmo de Prim, considerando uma estação de origem determinada como parâmetro de entrada. Para resolver esse problema, considere que o grafo que representa os dados armazenados do arquivo de dados no formato binário de entrada é um grafo não direcionado ponderado. Nesse sentido, o grafo desta funcionalidade [10] segue as mesmas especificações definidas para a funcionalidade [7] com a diferença que, para cada aresta (u,v) definida na funcionalidade [7], existe uma aresta (u,v) e uma aresta (v,u) definidas na funcionalidade [10], sendo o peso de cada aresta a distância. 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. A representação da árvore geradora mínima deve, obrigatoriamente, ser na forma de listas de adjacências, seguindo o mesmo formato definido na funcionalidade [7].

Sintaxe do comando para a funcionalidade [10]:

`10 estacao.bin nomeEstacaoOrigem "valorOrigem"`

onde:

- `estacao.bin` é um arquivo binário gerado conforme as especificações descritas no primeiro trabalho prático.

- `nomeEstacaoOrigem` é o nome do campo.

- `valorOrigem` é o valor passado como entrada para o `nomeEstacaoOrigem`. Deve ser colocado aspas duplas por se tratar de uma *string*.

Saída caso o programa seja executado com sucesso:

A saída deve ser exibida na saída padrão da seguinte forma. Percorra a árvore geradora mínima utilizando o algoritmo de busca em profundidade partir do vértice passado como entrada, o qual é representado pelo valorOrigem do atributo nomeEstacaoOrigem. Em cada chamada recursiva, escreva o nome da estação relativa ao vértice atual, chame recursivamente o algoritmo para o filho deste vértice e continue o processo até que todos vértices tenham sido analisados. O formato deve seguir a seguinte ordem: nome da estação relativa ao vértice atual, nome da estação relativa ao vértice filho, distância. Na próxima linha: nome da estação relativa ao vértice atual, nome da estação relativa ao vértice filho, distância, e assim sucessivamente. Caso um vértice tenha mais do que um filho, os filhos devem ser ordenados em forma crescente de acordo com o nome da estação.

Exemplo com metadados:

$nomeEstacao_{11}$, $nomeEstacao_{12}$, $distancia_1$
$nomeEstacao_{21}$, $nomeEstacao_{22}$, $distancia_2$
...
$nomeEstacao_{n1}$, $nomeEstacao_{n2}$, $distancia_n$

Mensagem de saída caso algum erro seja encontrado:

Falha na execução da funcionalidade.

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

```
./programaTrab
10 estacao.bin nomeEstacaoOrigem "Engenheiro Manoel Feio”
Engenheiro Manoel Feio, Itaquaquecebuta, 2270
Engenheiro Manoel Feio, Jardim Romando, 1990
Jardim Romano, Itaim Paulista, 2060
...
```

[11] Problema real: Você está em uma **estação origem** e quer chegar em uma **estação destino**. Entretanto, você quer saber quantos caminhos diferentes existem entre essa estação origem e a estação destino, bem como saber quais são esses caminhos. Baseado na resposta obtida, você vai escolher o caminho que melhor lhe agrada.

A funcionalidade [11] consiste em resolver este problema. Sua saída deve ser escrita da seguinte forma. Para cada caminho encontrado, primeiramente escrever em uma linha o número de estações que serão percorridas. Esse número de estações não deve considerar a estação de origem, mas deve considerar a estação de destino. Depois, escrever em outra linha a distância que será percorrida. Na sequência, escrever em outra linha os valores para os seguintes campos: o nome da estação de origem, os nomes das estações intermediárias e o nome da estação de destino. Devem ser escritos uma vírgula e um espaço em branco entre cada valor. Ao final da escrita de um caminho, pule uma linha em branco. Quando da escrita dos caminhos, a seguinte ordem deve ser seguida: (i) primeiramente, devem ser exibidos na saída padrão os caminhos que passam por menos vértices; (ii) em caso de empate, deve ser escolhido o caminho cujo primeiro vértice depois do vértice de origem tem menor valor; (iii) em caso de empate, deve ser escolhido o caminho cujo segundo vértice depois do vértice de origem tem o menor valor; (iv) em caso de empate, deve ser escolhido o caminho cujo terceiro vértice depois do vértice de origem tem o menor valor; (v) e assim sucessivamente.

Entrada do programa para a funcionalidade [11]:

`11 estacao.bin nomeEstacaoOrigem valorOrigem nomeEstacaoDestino valorDestino`

onde:

- `estacao.bin` é um arquivo binário gerado conforme as especificações descritas no primeiro trabalho prático.
- `nomeEstacaoOrigem` é o nome do campo.
- `valorOrigem` é o valor passado como entrada para o `nomeEstacaoOrigem`. Deve ser colocado aspas duplas por se tratar de uma *string*.
- `nomeEstacaoDestino` é o nome do campo.
- `valorDestino` é o valor passado como entrada para o `nomeEstacaoDestino`. Deve ser colocado aspas duplas por se tratar de uma *string*.

Saída caso o programa seja executado com sucesso:

A saída deve ser exibida na saída padrão da seguinte forma. Para cada caminho encontrado, primeiramente, escrever em uma linha o número de estações que serão percorridas. Esse número de estações não deve considerar a estação de origem, mas deve considerar a estação de destino. Depois, escrever em outra linha a distância que será percorrida. Na sequência, escrever em outra linha os valores para os seguintes campos: o nome da estação de origem, os nomes das estações intermediárias e o nome da estação de destino. Devem ser escritos uma vírgula e um espaço em branco entre cada valor. Ao final da escrita de um caminho, deixe uma linha em branco.

Exemplo com metadados:

```
Numero de estacoes que serao percorridas: x
Distancia que sera percorrida: y
nomeEstOrigem, nomeEstacao1, nomeEstacao2, ..., nomeEstDestino
--- pular uma linha em branco
Numero de estacoes que serao percorridas: x
Distancia que sera percorrida: y
nomeEstOrigem, nomeEstacao1, nomeEstacao2, ..., nomeEstDestino
--- pular uma linha em branco
```

OBS: o valor de x não inclui a estação de origem, mas inclui a estação de destino

Mensagem de saída caso não exista um caminho entre as estações solicitadas:

`Não existe caminho entre as estações solicitadas.`

Mensagem de saída caso algum erro seja encontrado:

`Falha na execução da funcionalidade.`

Exemplo de execução:

```
./programaTrab
11 estacao.bin nomeEstacaoOrigem "Palmeiras-Barra Funda"
nomeEstacaoDestino "Se"
Numero de estacoes que serao percorridas: 3
Distancia que sera percorrida: 5101
Palmeiras-Barra Funda, Luz, Sao Bento, Se
--- pular uma linha em branco
Numero de estacoes que serao percorridas: 4
Distancia que sera percorrida: 6115
Palmeiras-Barra Funda, Luz, Republica, Anhangabau, Se
--- pular uma linha em branco
Numero de estacoes que serao percorridas: 5
Distancia que sera percorrida: 4548
Palmeiras-Barra Funda, Marechal Deodoro, Santa Cecilia, Republica,
Anhangabau, Se
 --- pular uma linha em branco
```

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.
- 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.

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: **QLAT**

O vídeo gravado deve ser submetido por meio de um formulário no qual o grupo vai informar o nome de cada integrante, o número do grupo e um link do Google Drive que contém o vídeo gravado. Não deve ser usado o drive compartilhado da disciplina.

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

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 !