Universidade Federal do Espírito Santo
Departamento de Informática

2ª Trabalho Prático

Programação II - 2023/1
15 de Junho de 2023

Objetivo

O objetivo deste trabalho é colocar em prática as habilidades de programação na linguagem C adquiridas ao longo do curso. O trabalho foi elaborado para que você exercite diversos conceitos principalmente alocação e manipulação de memória de maneira dinâmica. Como consta no plano de ensino da disciplina, **o trabalho equivale a 40% da nota final do curso**.

Introdução

Manipulação de imagens é uma área de extrema relevância na computação moderna. Existem inúmeras aplicações que vão desde jogos digitais até reconhecimento de padrões através de um sistema de visão computacional. Obviamente, quanto mais complexa a aplicação, mais conhecimento é necessário para desenvolvê-la. Porém, aprender como uma imagem é representada e aplicar algoritmos básicos de manipulação já é possível mesmo para iniciantes na área.

A intenção deste trabalho é apresentar alguns conceitos básicos de representação de imagens bem como introduzir o uso de algoritmos para este tipo de dado. Para isso, você deverá implementar um programa na linguagem C que seja capaz de ler imagens a partir de arquivos de texto, aplicar um algoritmo de agrupamento, calcular histograma e reconstruir uma imagem a partir de um array. Para atingir esses objetivos serão necessários os conhecimentos de alocação dinâmica abordados ao longo desta disciplina.

Representação de imagens

Uma imagem é uma representação visual de um objeto ou uma cena. Em computação, uma imagem é representada por uma matriz de pixels, na qual cada pixel é um elemento da matriz que é representado por um valor numérico que indica a cor do pixel. A cor de um pixel pode ser representada por um valor inteiro de 8 bits, ou seja, entre 0 e 255, sendo 0 a cor preta e 255 a cor branca.

A dimensão da matriz que representa a imagem depende principalmente de dois fatores: a resolução da imagem e a quantidade de canais de cores. A resolução de uma imagem nada mais é do que a quantidade de pixels que a mesma possui. Já os canais de cores são os componentes que representam a cor de um pixel. Uma imagem colorida padrão possui três canais de cores: vermelho, verde e azul, que é o padrão RGB (*red*, *green*, *blue*). Portanto, uma matriz de pixels de uma imagem colorida possui três dimensões: altura, largura e profundidade. A altura e a largura são as dimensões da imagem em pixels, enquanto a profundidade é a quantidade de canais de cores. Cada pixel da imagem é o resultado da combinação da intensidade de cada um dos canais. Essa representação é ilustrada na Figura 1. Observe que o último pixel da primeira linha da matriz é a combinação das intensidades 119R + 121G + 82B, que resulta em um tom amarronzado.

Figura 1: ilustração de uma imagem colorida. Perceba que existem três matrizes de duas dimensões, uma para cada canal de cor.

Uma outra opção para representar uma imagem é utilizar tons de cinza. Neste caso, a imagem possui apenas um canal de cor, ou seja, a matriz que representa a imagem possui apenas duas dimensões: altura e largura. O pixel continua sendo representado

por um valor inteiro de 8 bits, porém, neste caso, o valor 0 representa a cor preta e o valor 255 representa a cor branca. A Figura 2 ilustra a representação de uma imagem em tons de cinza. A ideia de resolução continua a mesma.

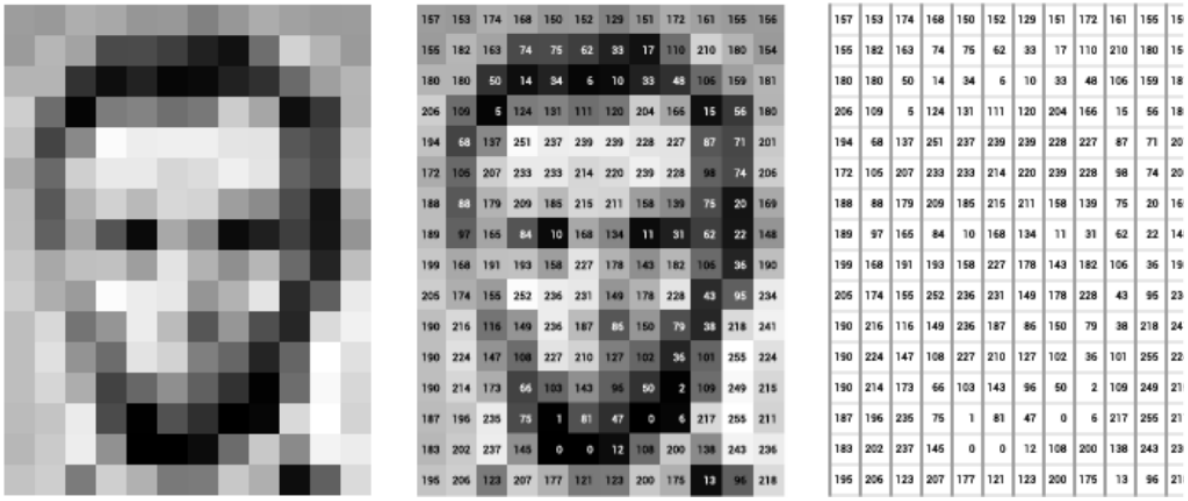

Figura 2: ilustração de uma imagem em tons de cinza. Perceba que a imagem nada mais é do que uma matriz de duas dimensões e cada pixel representa a intensidade

Manipular imagens em tons de cinza é muito mais fácil do que uma imagem colorida. Por isso, neste trabalho, vamos trabalhar apenas com imagens em tons de cinza. Além disso, vamos sempre trabalhar com resoluções bem pequenas, por exemplo, 28 x 28 pixels.

Agrupamento de dados

Agrupamento de dados (também conhecido como clusterização) é uma técnica de aprendizado de máquina não supervisionado que tem como objetivo agrupar dados similares. Existem diversas aplicações para esta técnica, como por exemplo, agrupar clientes de uma loja de acordo com o seu perfil de compra, agrupar pacientes de acordo com o seu perfil de saúde, **agrupar imagens de acordo com o seu conteúdo**, entre outros. Dado sua relevância, existem diversos algoritmos de agrupamento, sendo que cada um possui uma abordagem diferente para agrupar os dados. Neste trabalho, vamos utilizar o algoritmo K-Means (ou K-médias), que é um dos algoritmos mais simples e mais utilizados para agrupamento de dados.

O algoritmo K-means

O algoritmo K-means é um algoritmo de agrupamento baseado em uma medida de similaridade entre os dados. O método padrão recebe como entrada um conjunto de dados e o número de grupos K que devem ser formados. Uma vez conhecidos estes parâmetros, o algoritmo então funciona da seguinte forma:

1. Seleciona aleatoriamente K amostras do conjunto de dados, chamados de **centróides**. Cada centróide representa um grupo.
2. De acordo com a medida de similaridade escolhida, calcula-se o grau de similaridade entre uma dada amostra do conjunto e todas as centróides
3. O grupo da amostra é definido como sendo o grupo da centróide mais similar a ela.
4. Após todas as amostras serem atribuídas, o algoritmo calcula a média de cada grupo e atualiza os valores das centróides.
5. Os passos 2, 3 e 4 são repetidos até que algum critério de parada seja atingido.

Para esta etapa do trabalho, vamos utilizar como critério de parada o **número máximo de iterações**, ou seja, o algoritmo será executado até que o número máximo de iterações seja atingido. Além disso, a métrica de similaridade será distância Euclidiana, definida como:

$$d = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

Na qual **a** e **b** são arrays n-dimensionais.

Histograma de uma imagem

O histograma de uma imagem é uma representação gráfica da distribuição de intensidade de cores da imagem. No caso de imagens em tons de cinza, o histograma é uma representação gráfica da distribuição de intensidade de tons de cinza da imagem. O histograma é uma ferramenta muito útil para análise de imagens, pois permite identificar os padrões de cores da mesma. Usualmente ele é representado por um gráfico de barras, no qual o eixo X representa os valores de intensidade dos pixels e o eixo Y representa a quantidade de pixels que possuem determinado valor de intensidade. A Figura 3 ilustra o histograma de uma imagem em tons de cinza.

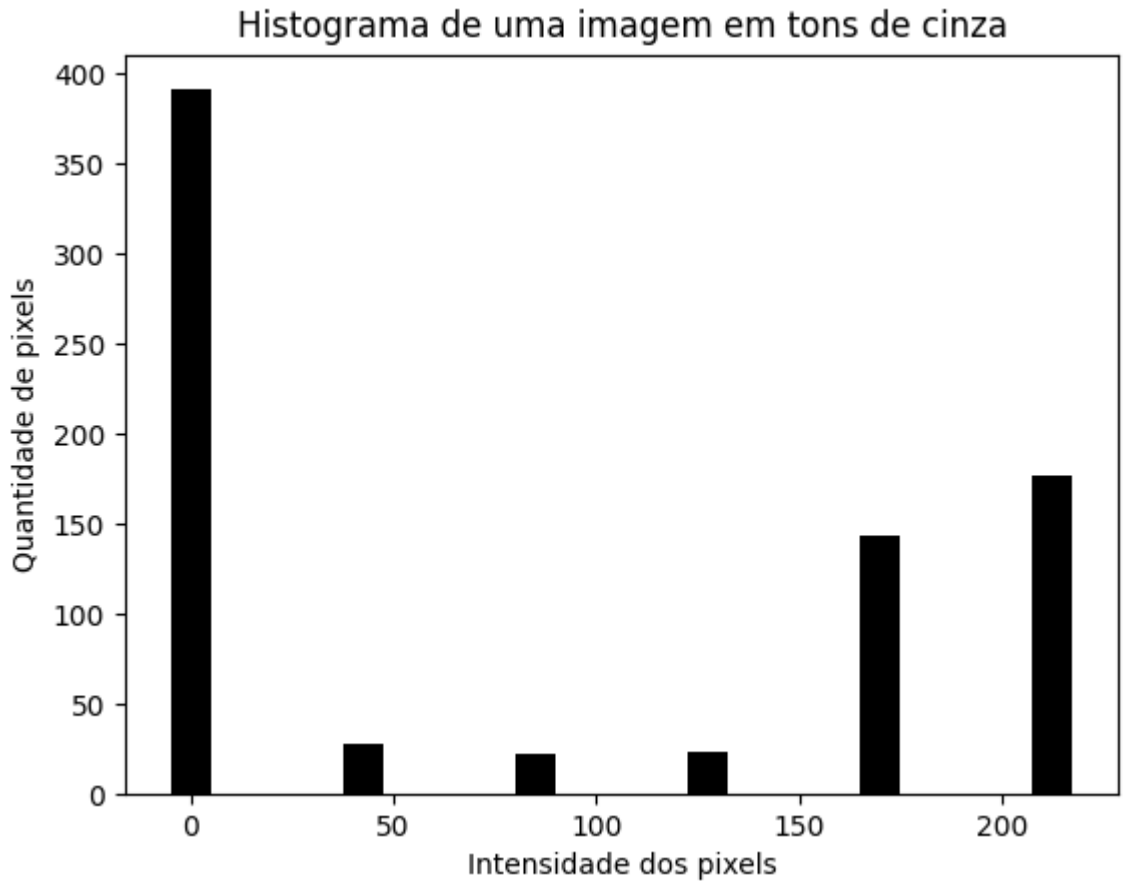

Figura 4: ilustração de um histograma de uma imagem em tons de cinza

Para construir um histograma é bem simples. Primeiro é necessário definir a quantidade de barras que o histograma terá. Usualmente, essa quantidade é conhecida como bins (ou caixotes). Um bin define um intervalo de pixel que será representado por uma barra no histograma. Como um pixel pode assumir valores de 0 a 255, para determinar o intervalo de cada bin basta dividir 255 pela quantidade de bins. Para o exemplo da Figura 4, foram utilizados 6 bins. Sendo assim, o primeiro bin será o intervalo [0, 42.5), o segundo bin será [42.5, 85), o terceiro bin será [85, 127.5), assim por diante até o último bin que será [212.5, 255]. Agora, para determinar a altura de cada barra, basta contar a quantidade de pixels que possuem intensidade dentro do intervalo de cada bin respeitando os intervalos definidos anteriormente. Por exemplo, para o primeiro bin, a altura da barra será a quantidade de pixels que possuem intensidade entre [0 e 42.5). Observe que é importante respeitar os intervalos fechados e abertos de cada bin.

Descrição do programa

A intenção deste trabalho é o desenvolvimento de um programa capaz de manipular dados em formato de imagens em tons de cinza. Para executar as funcionalidades do programa (que serão descritas a seguir) é necessário uma base de dados de imagens. Para facilitar o trabalho, todas as imagens são fornecidas no formato de um array unidimensional na qual cada posição representa um pixel da imagem. Esse array deve ser lido de um arquivo `.csv` (*comma separated value*). Este é um arquivo padrão na área de computação na qual cada valor de interesse é separado por vírgula. Cada imagem é representada por uma arquivo com nome `imagem_<n>.csv`, na qual `<n>` é um número inteiro (exemplo: `imagem_42.csv`). Todas as imagens estarão dentro de uma pasta que será fornecida via argumento para o programa (vide seção **Funcionamento do programa**). Todas as fontes de dados de imagem de todas as funcionalidades deverão ser obtidas desta pasta. Além disso, todas as imagens possuem a mesma dimensão e todos os pixels assumem valores inteiros de 0 a 255, o que representa uma intensidade de tom de cinza.

Todas as funcionalidades do programa são descritas a seguir.

Funcionalidade 1: agrupamento dos dados

A primeira funcionalidade do seu programa é realizar o agrupamento das imagens utilizando o algoritmo K-means. Para isso, você vai receber como entrada a quantidade de grupos K, as centróides iniciais e a quantidade máxima de iteração do algoritmo. As centróides iniciais serão representadas pelos nomes das imagens (por exemplo: `imagem_42`), que deve ser lida uma por linha. Obviamente, seu programa deve utilizar a base de imagens previamente descrita. Ao final da execução do algoritmo, seu programa deverá escrever em um arquivo os valores finais das centróides obtidas.

Por exemplo, se o número de grupos for igual a 3 e as imagens possuírem apenas 10 pixels, o resultado será na forma:

```
210, 189, 30, 25, 36, 89, 78, 20, 125, 144
25, 58, 49, 55, 63, 125, 223, 255, 125, 12
63, 52, 41, 74, 85, 96, 25, 36, 14, 58
```

Na qual a primeira linha representa o grupo 0, a segunda o grupo 1 e a terceira o grupo 2. Para mais informações consulte a seção de **funcionamento do programa** e o caso de teste disponibilizado.

Funcionalidade 2: definição do grupo de uma imagem

Uma vez que a funcionalidade 1 foi executada com sucesso, a intenção desta funcionalidade é determinar o grupo de uma nova imagem (na área de aprendizado de máquina esse processo é chamado de inferência). Sendo assim, sabendo o valor das centróides já obtidas, você deve receber como entrada o nome da imagem (exemplo: `imagem_42`) e determinar qual é o grupo que essa imagem deve pertencer de acordo com as centróides obtidas na funcionalidade 1. Você deverá escrever um arquivo contendo qual o grupo a imagem pertence. Obviamente, esta funcionalidade não pode ser executada sem que a funcionalidade 1 tenha sido executada pelo menos 1 vez.

Por exemplo, se o número de grupos for igual a 10 a `imagem_42` pode pertencer a um grupo de 0 a 9. Se for o grupo 8, seu arquivo deve conter:

```
8
```

Para mais informações consulte a seção de **funcionamento do programa** e o caso de teste disponibilizado.

Funcionalidade 3: histograma

Como o próprio nome já sugere, a ideia desta funcionalidade é obter o histograma de uma dada imagem. Seu programa vai receber o nome de uma imagem (exemplo: `imagem_42`) e a quantidade de bins desejado no histograma. Na sequência, ele deve escrever um arquivo contendo a quantidade de pixels obtidas para cada bin.

Por exemplo, se o número de bins for igual 4, um histograma válido para `imagem_42` é ilustrado a seguir:

```
BIN 1, 403
BIN 2, 38
BIN 3, 70
BIN 4, 273
```

Para mais informações consulte a seção de **funcionamento do programa** e o caso de teste disponibilizado.

Funcionalidade 4: reconstrução da imagem

Como já mencionado seu programa deverá ler arrays unidimensionais que representam imagens bidimensionais. A ideia dessa funcionalidade é bem simples: receber um array unidimensional para reconstruir ele no formato de uma imagem bidimensional. Para isso, seu programa deve receber o nome da imagem (exemplo: `imagem_42`) e a dimensão desejada (por exemplo, 28 x 28 pixels). Feito isso, seu programa deverá escrever em um arquivo os pixels desta imagem na dimensão informada.

Por exemplo, `imagem_42` é um array com 100 posições e foi solicitado uma imagem de 10 x 10 pixels, uma saída válida seria:

```
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
250, 255, 0, 0, 0, 0, 0, 0, 0, 255
```

Para mais informações consulte a seção de **funcionamento do programa** e o caso de teste disponibilizado.

Fluxo do programa

Ao iniciar seu programa o mesmo deve oferecer um menu que solicita ao usuário qual funcionalidade ele deseja executar. Você é livre para escolher o layout deste menu, desde que respeite os caracteres que representam cada ação. Sendo assim, o primeiro menu deve apresentar as seguintes opções:

```
########################## MENU INICIAL ############################
Escolha uma opcao:
- Executar algoritmo de agrupamento (A ou a)
- Determinar o grupo de uma imagem (G ou g)
- Obter o histograma de uma imagem (H ou h)
- Reconstruir uma imagem (R ou r)
- Finalizar programa (F ou f)
```

##

Sempre que o caractere `M` ou `m` aparecer, isso significa que você deve retornar ao menu inicial. Como já mencionado, todas as funcionalidades devem utilizar a pasta de imagens fornecidas para obter suas respostas. Todas as entradas adicionais devem ser lidas via teclado, sempre uma por linha. Obviamente, ao selecionar a opção de finalização, o programa deve ser encerrado e toda memória alocada será liberada.

Funcionamento do programa

Seu programa deve ser escrito na linguagem C e **obrigatoriamente**, a função principal deve estar em um arquivo chamado `trab2.c`. Você pode criar quantos TADs e bibliotecas julgar ser necessário. Por conta disso, você deve fornecer o arquivo `makefile` do seu programa. Para mais informações sobre este arquivo, leia este tutorial. Seu `makefile` deve gerar um executável para Linux de nome `trab2`.

Após compilado, seu programa deve receber dois argumentos de entrada através da linha de comando (para mais informações, leia este tutorial). O primeiro argumento indica onde o programa deve buscar a base de imagens utilizadas para as funcionalidades. O segundo argumento indica o caminho para uma pasta de saída, local na qual seu programa deve escrever os arquivos solicitados em cada uma das funcionalidades. Exemplo de chamada do programa:

```
./trab1 /home/user/imagens /home/user/saida
```

Neste exemplo, na pasta `/home/user/imagens` estarão todas as imagens seguindo o padrão descrito na seção anterior. Além disso, todas as escritas de arquivo deverão ser realizadas dentro do diretório `/home/user/saida`. **Isso é obrigatório para que a correção funcione corretamente**. Se você não cumprir essa especificação, não é possível corrigir o seu trabalho.

A entrada de dados do menu programa será realizada utilizando a entrada padrão. Isso significa que a leitura pode ser realizada tanto via teclado quanto usando o terminal do Linux com o comando **<** . Exemplo de chamada do programa:

```
./trab1 /home/user/imagens /home/user/saida < dados_entrada
```

Saída de dados

Todos os arquivos de saída devem ser gerados dentro do diretório informado como argumento do programa. Todos arquivos deverão ter o formato .csv, com cada coluna sendo separado por vírgula.

Para **funcionalidade 1**, a saída do seu programa deve ser salvo em um arquivo chamado `centroides.csv`. Se a funcionalidade for executada mais de uma vez, esse arquivo deve ser sobrescrito. Em outras palavras, só pode existir um único arquivo do tipo na pasta de saída.

Para a funcionalidade 2, seu programa deve criar um arquivo no formato `grupo_imagem_n.csv` para cada solicitação do usuário. Por exemplo, se o usuário solicitar o grupo das imagens 10, 13 e 18, você deve criar os arquivos `grupo_imagem_10.csv`, `grupo_imagem_13.csv` e `grupo_imagem_18.csv`. Essa mesma ideia deve ser executada para as funcionalidades 3 e 4, que devem gerar arquivos nos formatos `hist_bin_imagem_n.csv` e `rec_imagem_n.csv`, respectivamente.

Sendo assim, um exemplo de saída válida para a pasta de saída do programa é exemplificado a seguir:

- `/home/user/saida`
  - `centroides.csv`
  - `grupo_imagem_42.csv`
  - `rec_imagem_42.csv`
  - `rec_imagem_10.csv`
  - `hist_5_imagem_499.csv`
  - `hist_7_imagem_5000.csv`

Regras gerais

Para todas as etapas do trabalho, considere as seguintes regras:

- Em todos os arquivos de saída você deve escrever sempre com letras maiúsculas. Por exemplo, se o paciente se chama Joao, na saída dele deve constar como JOAO.
- Todos os arquivos de entrada vão seguir sempre o mesmo padrão, incluindo o nome dos arquivos.
- Para dados de ponto flutuante, use sempre precisão simples. Para a escrita destes dados, use apenas duas casas decimais.
- **Não** é permitido o uso de bibliotecas externas além das que já foram trabalhadas em sala de aula, ou seja, `stdio.h`, `stdlib.h`, `string.h`,

`time.h`. Para este trabalho, uma dica é utilizar a biblioteca `dirent.h` para manipulação dos arquivos dos diretórios.
- Os dados devem ser alocados na memória na medida que aparecem para ser cadastrados. Isso deve ser feito com **alocação dinâmica**. Trabalhos que utilizem alocação estática para este fim, serão desconsiderados
- É obrigatório o uso de pelo menos um TAD totalmente encapsulado (opaco). Lembre-se, quanto **mais modularizado, mais fácil vai ser a modificação para inclusão da(s) nova(s) features que deverão ser implementadas em sala de aula**
- Liberar a memória alocada é de sua responsabilidade. O seu programa será testado utilizando o Valgrind. Haverá descontos significativos da nota para erros e vazamentos de memória apontados pela ferramenta
- Faça seu código de maneira eficiente. O tempo máximo de execução para todos os testes será de 60 segundos.
- **Possíveis problemas na descrição do trabalho serão solucionados e comunicados via Classroom. Don't Panic!**

Correção do trabalho

O seu trabalho será corrigido de maneira automática em um terminal Linux. Seu programa será compilado usando o `makefile` que você deve disponibilizar. Na sequência ele será executado da seguinte forma:

```
./trab1 /home/user/imagens /home/user/saida < dados_entrada
```

Após a execução, os arquivos gerados dentro do diretório de saída (informado como parâmetro) serão comparados com as respostas reais. **É muito importante que você siga rigorosamente as instruções para que o teste não falhe.**

Além do teste automático, seu código será avaliado utilizando os seguintes critérios:
- Organização
- Modularização
- Criação de TADs
- Gerenciamento e manipulação adequados da memória

Em outras palavras, não basta apenas acertar todos os testes é necessário cumprir todos os pré-requisitos desta especificação.

A correção possui duas etapas. Na primeira, seu código será testado e avaliado segundo os critérios já estabelecidos. A segunda etapa é a implementação de novas

funcionalidades que serão informadas no dia da entrega do trabalho e deverá ser realizada obrigatoriamente em sala de aula. A intenção desta etapa é verificar o seu conhecimento para com o código e forçar a implementação modularizada e organizada da solução. A nota final será a média ponderada entre as duas etapas. Além disso, todos os trabalhos serão testados em relação a plágio, que terão tolerância zero.

Se você não possui o sistema operacional Linux, **garanta que seu código funcione nos computadores do Labgrad**. Qualquer discrepância de resultado na correção por conta de Sistema Operacional, os computadores do Labgrad serão utilizados para solucionar o problema.

Prazo e submissão

O trabalho deve ser submetido até **23h:59min do dia 11 de julho de 2023** utilizando um repositório na organização do Github da disciplina. Você deve criar esse repositório **privado** seguindo o seguinte padrão: `trab-2-<seu_nome>-<seu-sobrenome>`. Exemplo: `trab-2-andre-pacheco`.

Se porventura você atrasar a entrega do trabalho, será descontado 40% da nota final a cada dia de atraso.