Trabalho 03 - Compressão de Imagens

stco01 2025s2

Data de entrega: 22/11/2025
Data de entrega atrasada sem penalidade: 29/11/2025

Importante:

- **Não** olhem códigos de outros grupos ou da internet, exceto os que são fornecidos. Não utilize LLMs.
- Os trabalhos DEVEM ser feitos com o mesmo grupo do trabalho 01.
- TODOS os membros do grupo devem participar e compreender completamente a implementação.
- Em caso de plágio, fraude ou tentativa de burlar o sistema será aplicado nota 0 na disciplina aos envolvidos.
- Alguns alunos serão solicitados para explicar com detalhes a implementação.
- Passar em todos os testes não é garantia de tirar a nota máxima. Sua nota ainda depende do cumprimento das especificações do trabalho, qualidade do código, clareza dos comentários, boas práticas de programação e entendimento da matéria demonstrada em possível reunião.
- O líder do grupo deverá submeter, até a data de entrega, o seu código na plataforma runcodes.hokama.com.br.
- As dúvidas podem ser encaminhadas primeiramente ao monitor da disciplina.
- É esperado e faz parte do aprendizado vocês terem algumas dificuldades como Problemas com Falha de Segmentação, Loops Infinitos, etc. Então não deixe para o último dia!

Este trabalho deverá ser feito em duplas (as mesmas) e implementado em linguagem C.

Neste trabalho utilizaremos imagens no formato **PBM P1**, um dos formatos mais simples da família NetPBM, usados tradicionalmente para representar imagens monocromáticas (preto e branco) de forma **texto puro (ASCII)**. O formato PBM P1 é composto por três partes principais:

Cabeçalho: A primeira linha do arquivo deve conter:

```
P1
```

Dimensões da imagem: A segunda linha contém **largura** e **altura** da imagem, em pixels:

```
4 4
```

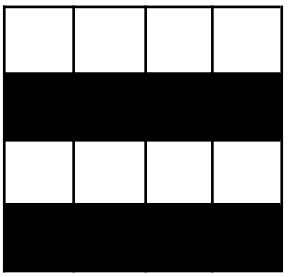
Pixels: O restante do arquivo contém os pixels, representados por valores:

- `0` → pixel branco
- `1` → pixel preto

ao final de cada linha tem um espaço depois do último caractere.

```
P1
4 4
0 0 0 0
1 1 1 1
0 0 0 0
1 1 1 1
```

Essa figura é a seguinte:



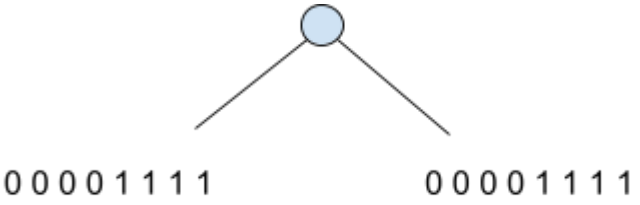
Entretanto esse formato é conhecido por não ser muito compacto. Em particular podem existir grandes sequências de pixels brancos ou pretos, mas o formato exige que cada pixel seja explicitamente declarado. Queremos então comprimir imagens nesse formato usando uma árvore binária. Para tanto os pixels devem ser organizados em uma **única sequência linear**, preservando a ordem original dos dados.

Essa etapa é chamada de **aplainamento** (ou *flattening*), e transforma uma matriz de pixels em um vetor. O exemplo acima ficaria da seguinte forma:

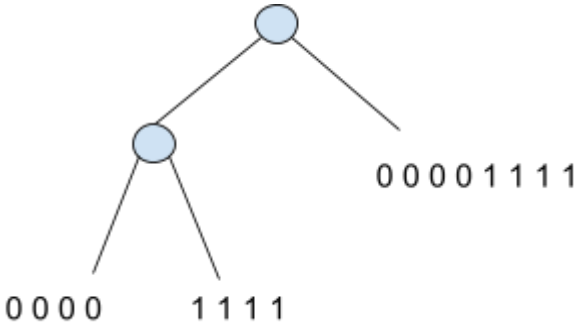
```
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
```

A ideia agora é que iremos construir uma árvore binária, da seguinte forma: Se todos os pixels de um dado trecho do vetor forem todos iguais, aquele trecho é representado por um nó folha. Se houverem zeros e um's no trecho, então dividimos aquele trecho na metade, e criamos um nó interno da árvore.

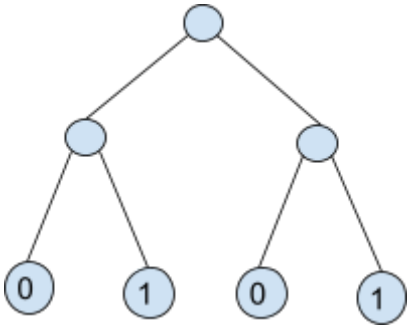
No exemplo acima, existem zeros e um's, e portanto dividimos o vetor na metade e criamos um primeiro nó interno (nesse caso a raiz) na árvore.



Como os trechos ainda não são uniformes, uma nova divisão é feita na esquerda.



Agora como o trecho “0 0 0 0” na subárvore esquerda é uniforme, ele pode ser substituido por um nó folha, que não precisa guardar 4 vezes o número zero, apenas 1 vez é suficiente. Analogamente o trecho “1 1 1 1” também pode virar uma folha. E o mesmo acontecerá para a subárvore direita. A árvore final ficará assim:



Podemos guardar essa árvore imprimindo-a em pré-order, colocando um ‘X’ quando o nó faz divisão e 0 ou 1 se for uma folha. Além disso, precisamos da informação das dimensões da imagem.

```
4 4
XX01X01
```

Esse formato tem todas as informações necessárias para reconstruirmos a imagem original, e ela normalmente gastará menos bytes que o formado bpm original.

Seu programa deverá compactar e descompactar imagens de bpm para árvore, e de árvore para bpm. Você deverá ler da entrada padrão do sistema um caractere ’c’ se for para

comprimir, ou ‘d’ se for para descomprimir, seguido do nome de um arquivo, e você deverá imprimir na saída padrão do sistema a imagem no outro formato.

Por exemplo, a imagem anterior estava salva no arquivo 01.bpm. Então seu programa deveria ler da entrada:

```
c 01.bpm
```

e deverá imprimir:

```
4 4
XX01X01
```

Depois disso o seu programa pode liberar toda a memória usada e fechar normalmente.

Já se o seu programa receber o comando:

```
d 01.arv
```

ele deverá imprimir a imagem no formato bpm (no final de cada linha dos pixels tem um espaço em branco).

```
P1
4 4
0 0 0 0
1 1 1 1
0 0 0 0
1 1 1 1
```

e seu programa pode fechar normalmente depois de liberar corretamente a memória alocada.