Aula 3 - Desafio 1

Docupedia Export

Author:Balem Luis (CtP/ETS)
Date:30-Mar-2023 17:48

Table of Contents

1 Desafio 1.1 **3**

2 Desafio 1.2 **4**

1 Desafio 1.1

Agora que já aprendemos o básico do C# e já podemos escrever programas simples vamos ao primeiro desafio: Um compressor de dados com perdas.
A ideia é simples: Em arquivos como imagens e vídeos a perda de informação durante a compressão é aceitável e até desejado. Isso advém do fato de que é pouco perceptível aos nossos olhos.
A ideia é eliminar parte dos dados e reduzir o espaço gasto por eles. Ao ver uma cor RGB (255, 255, 255) que é branco podemos perceber que em binário temos:
11111111, 11111111, 11111111
Ao jogarmos fora os 4 bits menos significativos (da direita), poderíamos guardar apenas o 4 dígitos que mais impactam no cálculo do binário:
1111, 1111, 1111
Assim, criamos um compressor de 50%. Ao tentar voltar para 8 bits para poder rever a cor podemos preencher os bits que perdemos com valores como 0:
11110000, 11110000, 11110000
Que é o RGB (240, 240, 240). Procure no goolge por 'rgb(240, 240, 240)' e veja a pequena diferença para o branco 'rgb(255, 255, 255)'. E perceba que este é o pior caso possível, onde mais se tem perda de informação. Se o RGB já fosse (240, 240, 240), por exemplo, não teríamos perda alguma.
Seu trabalho é fazer uma função que receba um vetor de byte que será compactada desta forma. Note que precisará retornar um vetor com a metade do tamanho e pior: A cada 2 bytes você deve colocar 4 bits de cada um e um único bit. Exemplo:
Original: 11111111, 00000000, 10101111, 11111010 Compactado: 1111, 0000, 1010, 1010 Resultado: 11110000, 1010101

```
byte[] compact(byte[] originalData)
{
    // Implemente aqui
}
```

Desafio Opcional: Faça uma compactação apenas para 3 bytes, ao invés de 4.

2 Desafio 1.2

Agora implemente a função de descompactação que retorna os dados aos dados originais, preenchendo os bits que foram perdidos com 0.

```
byte[] decompact(byte[] compactedData)
{
    // Implemente aqui
}
```

Desafio Opcional: Adicione um parâmetro opcional chamado preenchimento que permite o usuário definir o modo de preenchimento, assim, caso seja 0 (que é o valor padrão), será preenchido com 0. Se for 1, os bits serão preenchidos com 1 (1111 = 15). Caso seja 2, você fará um preenchimento mais inteligente: usará a metade do valor máximo perdido (15 / 2 = 7 = 0111) para minimizar o erro da compreesão.

```
byte[] decompact(byte[] compactedData, int fillingMode = 0)
{
    // Implemente aqui
}
```