Estudante: Maxsuel Aparecido Lima Santos
Matrícula: 202511587
Professor: Fernando S. Silva
Curso: Ciência da Computação

DOCUMENTAÇÃO

Este documento é referente ao código de operações matriciais utilizando a linguagem Python sem nenhuma importação de biblioteca externa para a manipulação matricial de um arquivo .pbm (portable bitmap) no formato ASCII P1.

Em primeira instância, é válido ressaltar que a única biblioteca utilizada neste código foi a *streamlit*, que ajuda na visualização do código no navegador, por meio do *localhost:8501*. Desse modo, a visualização fica mais fácil e intuitiva ao usuário. Além disso, o *streamlit* possibilita o deploy da aplicação, ou seja, um simples script python pode ter upload na nuvem e ser executado por um simples link de domínio na world wide web.

Para conseguir executar o código python com streamlit localmente, basta abrir o terminal do Visual Studio Code (IDE utilizada por mim) ou então o powershell do windows ou terminal do linux e executar o seguinte comando:

pip install streamlit

Assim, será possível executar o código. Para executar o código, recomendo o uso da IDE Visual Studio Code (VS Code), pois facilita a execução do runner do python. Para executar, digite o seguinte comando:

streamlit run nomedoarquivo

Após esse comando, será aberto um servidor local na porta 8501 do navegador padrão do seu dispositivo. Agora basta aproveitar e testar o software.

À luz dessa perspectiva introdutória, segue abaixo uma breve descrição dos principais pontos a serem destacados no código.

1. Leitura do arquivo pbm (ler_pbm)

O seguinte trecho é responsável pela leitura do arquivo .pbm que o usuário quiser fazer upload.

```python
def ler_pbm(conteudo):
    linhas = [linha.strip() for linha in conteudo.split('\n')
              if linha.strip() and not linha.startswith('#')]
    if not linhas or linhas[0] != 'P1':
        st.error("Formato PBM inválido. Deve começar com P1.")
        return None
    largura, altura = map(int, linhas[1].split())
    dados = ' '.join(linhas[2:]).split()
    return [[int(dados[i*largura + j]) for j in range(largura)] for i in range
    (altura)]
```

1.1. linhas = [...]

Divide o conteúdo do arquivo em linhas;
Remove espaços em branco e linhas de comentário (começando com #).

1.2. if not linhas or linhas[0] != 'P1'

Verifica se o arquivo é um PBM válido (deve começar com P1).

1.3. largura, altura = map(int, linhas[1].split())

Lê as dimensões da matriz (largura × altura).

1.4. dados = ' '.join(linhas[2:]).split()

Concatena os pixels em uma única lista.

1.5. [[int(dados[i*largura + j]) for j in range(largura)] for i in range(altura)]

Constrói a matriz bidimensional usando list comprehension.
Fórmula de indexação: i*largura + j mapeia a posição linear do pixel para a matriz 2D (2 dimensões).

2. Conversão de matriz para PBM (matriz_para_pbm)

Transforma a matriz de volta em uma string no formato PBM.

```python
def matriz_para_pbm(matriz):
    altura = len(matriz)
    largura = len(matriz[0]) if altura > 0 else 0
    return f"P1\n{largura} {altura}\n" + '\n'.join(' '.join(map(str, linha)) for
    linha in matriz)
```

2.1. altura = len(matriz)

Número de linhas da matriz.

2.2. largura = len(matriz[0]) if altura > 0 else 0

Número de colunas (verifica se a matriz não está vazia).

2.3. return f"P1\n{largura} {altura}\n" + …

Monta o cabeçalho PBM (P1, dimensões).

Converte cada linha da matriz em uma string separada por espaços.

3. Operações geométricas

3.1. Transposta (inverte linhas por colunas)

```python
def transposta(matriz):
    return [[linha[i] for linha in matriz] for i in range(len(matriz[0]))]
```

linha[i] for linha in matriz pega o i-ésimo elemento de cada linha, formando uma nova coluna.

3.2. Rotação 90º (gira a matriz 90º no sentido horário)

```python
def rotacionar_90(matriz):
    return [list(linha)[::-1] for linha in zip(*matriz)]
```

*zip(*matriz)* transposta a matriz.

reversed(linha) inverte cada linha para obter a rotação.

3.3. Rotação 180º (gira a matriz 180º no sentido horário)

```python
def rotacionar_180(matriz):
    return [linha[::-1] for linha in matriz[::-1]]
```

matriz[::-1] inverte a ordem das linhas.

linha[::-1] inverte cada linha.

3.4. Rotação 270º (Gira a matriz 270° (equivalente a 90° anti-horário))

```
def rotacionar_270(matriz):
    return [list(linha) for linha in zip(*matriz)][::-1]
```

*zip(*matriz)* transposta a matriz.

[::-1] inverte a ordem das linhas.

4. Operações de permuta (troca de linhas/colunas)

4.1. Troca de linhas (trocar_linhas)

```
def trocar_linhas(matriz, i, j):
    matriz[i], matriz[j] = matriz[j], matriz[i]
    return matriz
```

Simples troca de posições i e j usando desempacotamento de tuplas.

4.2. Troca de colunas (trocar_colunas)

```
def trocar_colunas(matriz, i, j):
    return [[linha[i] if k == j else linha[j] if k == i else linha[k]
             for k in range(len(linha))] for linha in matriz]
```

4.2.1. *linha[i] if k == j else linha[j] if k == i else linha[k]*

Se k == j, pega o valor da coluna i.

Se k == i, pega o valor da coluna j.

Caso contrário, mantém o valor original.

5. Operações matemáticas

5.1. Negativa da imagem (Inverte 0 → 1 e 1 → 0.)

```
def negativa(matriz):
    return [[1 - pixel for pixel in linha] for linha in matriz]
```

1 - pixel inverte o valor do pixel.

5.2. Multiplicação por escalar

```
def multiplicar_escalar(matriz, escalar):
    return [[pixel * escalar for pixel in linha] for linha in matriz]
```

Multiplica cada pixel por um escalar (útil para ajuste de brilho).

5.3. Soma de matrizes de mesmas dimensões

```
def somar_matrizes(m1, m2):
    return [[m1[i][j] + m2[i][j] for j in range(len(m1[0]))] for i in range(len(m1))]
```

Soma elemento a elemento (*m1[i][j] + m2[i][j]*).