Programação Estruturada

Yuri Malheiros

Projeto final

Introdução

O NumPy é uma das principais ferramentas para ciência de dados e base de diversas bibliotecas para linguagem Python. Um dos principais motivos para sua eficiência é que o NumPy possui diversas partes implementadas em C. O objetivo desse projeto é criar uma implementação para representação e manipulação de matrizes inspirada no NumPy.

Representação da matriz

Um dos pontos fundamentais para implementação desse projeto é a forma de representação de uma matriz. No projeto, a matriz deve ser representada por uma struct:

```
struct matrix {
    int *data;
    int n_rows;
    int n_cols;
    int stride_rows;
    int stride_cols;
    int offset;
};
```

- `data` é um array unidimensional com os dados da matriz. Nós vamos guardar todos os dados num array unidimensional, o tratamento das dimensões será feito por outros membros da struct.
- `n_rows` é o número de linhas da matriz.
- `n_cols` é o número de colunas da matriz.

Uma matriz do tipo:

```
{{1, 2, 3},
 {4, 5, 6},
 {7, 8, 9}}
```

Será guardada como:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- `stride_rows` é um número que determina quantos elementos devem ser "pulados" no array unidimensional para acessar uma próxima linha. No exemplo acima, considerando o elemento `1` (elemento da primeira linha e da primeira coluna), se quisermos acessar a próxima linha (elemento `4`), temos que andar 3 elementos no array unidimensional. Assim, `stride_rows` tem o valor `3`.
- `stride_cols` é o número que determina quantos elementos devem ser "pulados" no array unidimensional para acessar a próxima coluna. Com uma matriz criada, `stride_cols` vai ser igual a `1`, pois para acessar um elemento de uma próxima coluna basta acessar o próximo índice. Algumas funções vão modificar esse número, então é importante defini-lo mesmo que seu valor inicial seja sempre `1`.
- `offset` é o índice de onde devemos começar a considerar os dados do array unidimensional. Seu valor padrão quando uma matriz for criada é `0`, pois vamos considerar os dados desde o início. Algumas funções vão modificar esse número, então é importante defini-lo mesmo que seu valor inicial seja sempre `0`.

Funções para criação de matrizes

As funções para criação de matrizes que precisam ser implementadas são as seguintes.

`create_matrix`

```
struct matrix create_matrix(int *data, int n_rows, int n_cols);
```

`create_matrix` é a principal função para criar uma matriz.

- `*data` é um array unidimensional com os dados
- `n_rows` é o número de linhas da matriz
- `n_cols` é o número de colunas da matriz

Note que os membros `stride_rows`, `stride_cols` e `offset` da struct retornada precisam ser iniciados.

zeros_matrix

```
struct matrix zeros_matrix(int n_rows, int n_cols);
```

`zeros_matrix` cria uma matriz com todos os elementos iguais a zero.

- `n_rows` é o número de linhas da matriz
- `n_cols` é o número de colunas da matriz

random_matrix

```
struct matrix random_matrix(int n_rows, int n_cols, int b, int e);
```

`random_matrix` cria uma matriz com os elementos com valores aleatórios num intervalo entre `b` e `e`.

- `n_rows` é o número de linhas da matriz
- `n_cols` é o número de colunas da matriz
- `b` é o ínicio do intervalo dos números aleatórios
- `e` é o fim do intervalo dos números aleatórios

i_matrix

```
struct matrix i_matrix(int n);
```

`i_matrix` cria uma matriz identidade.

- `n` é o tamanho da matriz. Lembre-se que a matriz identidade tem o mesmo número de linhas e colunas.

Funções para acessar elementos

As funções para acessar os elementos de uma matriz que precisam ser implementadas são as seguintes.

get_element

```
int get_element(struct matrix a_matrix, int ri, int ci);
```

`get_element` retorna um elemento da matriz de acordo com sua posição.

- `a_matrix` é a matriz que vai ser acessada
- `ri` é o índice da linha do elemento
- `ci` é o índice da coluna do elemento

put_element

```
void put_element(struct matrix a_matrix, int ri, int ci, int elem);
```

`put_element` coloca um elemento na matriz de acordo com a posição.

- `a_matrix` é a matriz que vai ser acessada
- `ri` é o índice da linha do elemento
- `ci` é o índice da coluna do elemento
- `elem` é o elemento que vai ser colocado

print_matrix

```
void print_matrix(struct matrix a_matrix);
```

`print_matrix` exibe os dados no formato da matriz.

- `a_matrix` é a matriz que vai ser exibida

Funções para manipulação de dimensões

Os membros `n_rows`, `n_cols`, `stride_rows`, `stride_cols` e `offset` são fundamentais para implementação dessas funções. Assim, o conteúdo de `data` não deve ser modificado, isto faz com que essas funções executem com grande velocidade.

As funções para manipulação de dimensões de matrizes que precisam ser implementadas são as seguintes.

transpose

```
struct matrix transpose(struct matrix a_matrix)
```

`transpose` retorna a matriz transposta.

- `a_matrix` é a matriz usada para gerar a matriz transposta

reshape

```
struct matrix reshape(struct matrix a_matrix, int new_n_rows, int new_n_cols);
```

`reshape` altera as dimensões da matriz. Note que nem todas as dimensões são válidas, pois a quantidade de elementos tem que coerente com as dimensões da matriz (`quantidade_de_elementos = n_rows * n_cols`).

- `a_matrix` é a matriz que terá suas dimensões alteradas
- `new_n_rows` é o novo número de linhas

- `new_n_colunas` é o novo número de colunas

`flatten`

```
struct matrix flatten(struct matrix a_matrix);
```

`flatten` redimensiona a matriz para que todos os dados fiquem apenas em 1 linha.

- `a_matrix` é a matriz que será transformada

`slice`

```
struct matrix slice(struct matrix a_matrix, int rs, int re, int cs, int ce);
```

`slice` retorna um "recorte" da matriz original.

- `a_matrix` é a matriz original
- `rs` é o índice da linha inicial do recorte
- `re` é o índice da linha final do recorte
- `cs` é o índice da coluna inicial do recorte
- `ce` é o índice da coluna final do recorte

Dada a matriz:

```
struct matrix my_matrix;
int data[] = {1,2,3,4,5,6,7,8,9};

my_matrix = create_matrix(data, 3, 3);
```

A chamada:

```
my_matrix = slice(my_matrix, 0, 2, 1, 3);
```

vai cortar `my_matrix` começando na linha `0` e indo até a linha de índice `1`, e começando na coluna `1` e indo até a coluna de índice `2`. É importante ressaltar que se `re` for `2` o índice final é `1`, e se `re` for `3` o índice final é `2`. Então, o índice final é sempre subtraído de `1`.

Funções de agregação

As funções de agregação que precisam ser implementadas são as seguintes.

`sum`

```
int sum(struct matrix a_matrix);
```

`sum` soma todos os elementos da matriz e retorna o resultado.

- `a_matrix` é a matriz que terá seus elementos somados

mean

```
int mean(struct matrix a_matrix);
```

`mean` calcula a média dos elementos da matriz.

- `a_matrix` é a matriz usada para calcular a média

min

```
int min(struct matrix a_matrix);
```

`min` retorna o menor elemento da matriz.

- `a_matrix` é a matriz onde o menor elemento será procurado

max

```
int max(struct matrix a_matrix);
```

`max` retorna o maior elemento da matriz.

- `a_matrix` é a matriz onde o maior elemento será procurado

Funções de operações aritméticas

As funções de operações aritméticas que precisam ser implementadas são as seguintes.

```
struct matrix add(struct matrix a_matrix, struct matrix b_matrix);
struct matrix sub(struct matrix a_matrix, struct matrix b_matrix);
struct matrix div(struct matrix a_matrix, struct matrix b_matrix);
struct matrix mul(struct matrix a_matrix, struct matrix b_matrix);
struct matrix matmul(struct matrix a_matrix, struct matrix b_matrix);
```

`add`, `sub`, `div` e `mul` são as quatro operações aritméticas. Essas funções fazem as operações elemento por elemento e retornam a matriz. `matmul` é a multiplicação de matriz, lembre que multiplicação de matriz é diferente de multiplicar elemento por elemento.

Entrega

- O trabalho pode ser em dupla ou individual;

- Será necessário criar o arquivo `matrix.h` com os protótipos das funções e a definição da struct;
- O arquivo `matrix.c` terá as implementações;
- Os arquivos devem ser compactados e enviados através do SIGAA;
- Os arquivos devem iniciar com um comentário com o nome do(s) aluno(s) e sua(s) matrícula(s);
- Se for feito em dupla, os dois alunos devem fazer upload separadamente do mesmo arquivo.

Critérios de avaliação

O projeto vai ser avaliado através de testes das funções implementadas. Eu vou incluir o arquivo `matrix.h` e executar diversos testes para avaliar se as implementações estão corretas. Por isso é fundamental que os protótipos das funções sejam exatamente os especificados nesse arquivo. Se isso não acontecer, o teste não funcionará. Também será realizada uma análise geral do código.

Observações

- Durante a implementação, vocês devem criar um arquivo `main.c` onde vocês testarão as funções implementadas;
- Os protótipos das funções precisam ser exatamente os que estão nesse arquivo;
- O projeto será acompanhado durantes as semanas para que possíveis dúvidas sejam sanadas.