Fundação Getulio Vargas

Estruturas de Dados e Algoritmos

Projeto Final - “Capivara”

Prof. Jorge Poco

1. **Objetivo:** Nesse projeto vocês irão implementar uma biblioteca baseada em C++ para Python chamada “Capivara”. A biblioteca teve ter a classe “DataFrame” similar a biblioteca Pandas com algumas funções básicas e também suporte para pontos espaciais como a biblioteca GeoPandas. A implementação deve ser otimizada para permitir rápido acesso dos dados.
2. **Implementação**
    a. A primeira função que deve ser implementada é para ser possível criar um *DataFrame* com colunas. Cada coluna terá um identificador do tipo *string,* todas colunas serão listas de mesmo comprimento e com o mesmo tipo de dado. Os tipos de dados possíveis serão apenas *int* e *float*. A função *DataFrame* receberá um objeto do tipo dicionário como no exemplo abaixo. Um tipo distinto de coluna que pode existir é com uma geometria de duas dimensões, isto é, uma sequência de valores “*x”* e uma sequência de valores “*y”.* Essa coluna poderá ser criada com o método de classe *points_from_xy* que receberá duas listas. Ela será útil para outras funções.

```
import capivaria as cp

# From Dict
df = cp.DataFrame({
    'City_Code': [1, 2, 3, 4, 5],
    'Latitude': [-34.58, -15.78, -33.45, 4.60, 10.48],
    'Longitude': [-58.66, -47.91, -70.66, -74.08, -66.86]
})

# 2D points
df[‘geometry’] = cp.points_from_xy(df.Longitude, df.Latitude)
```

    b. Deve ser possível criar índices em 1D e em 2D. O objetivo da indexação é construir uma estrutura interna para que seja possível fazer consultas de forma eficiente. Em 1D o comando *set_index* vai receber o identificador da coluna que será utilizada para indexação. Em 2D, o mesmo comando será utilizado (*set_index),* no entanto, a coluna que ele receberá deve ter sido criada pela função *points_from_xy.* Deve ser implementada uma árvore B+ para criar a indexação 1D e uma árvore R para a indexação 2D. Como referência para a árvore B+ indico os seguintes links 1, 2. Para a ávore R os seguintes links 1, 2.

```
df.set_index('City_Code')   # Use uma estrutura 1D para indexar os dados

df.set_index('geometry') # Use uma estrutura 2D para indexar os pontos
```

c. Deve ser possível adicionar e remover novas linhas de dados baseado nos índices. Para isso, as estruturas (B+ e R) devem ter implementadas funções para inserção e remoção de elementos.

d. Após, deve-se implementar funcionalidade de consultas no DataFrame baseado nos valores das colunas. As consultas devem ser tanto em 1D quanto em 2D. Em 1D deve ser possível fazer consultas de igualdade e de intervalo (buscar linhas do *DataFrame* cujo valor de uma coluna é igual ou está dentro de um intervalo com limite inferior e superior). Em 2D deve ser possível verificar se os pontos estão dentro de um polígono com a função *intersection,* essa função deve receber como parâmetro a descrição do polígono, uma possibilidade seria a lista de pontos que formam o "contorno". Considere também que é interessante ter suporte específico para verificar se intersecta com um retângulo, definindo apenas os dois pontos diagonais. Em 2D também deve ser possível buscar o ponto mais próximo a um ponto escolhido com a função *nearest.* As consultas devem se utilizar das vantagens da indexação sempre que possível.

```
# 1D query
df[df['City_Code']==2]
df['City_Code'].between(-0.5, 0.5, inclusive = False)

# 2D query
query_rect = [xmin, ymin, xmax, ymax] # retângulo de seleção
spatial_index = df.sindex
precise_matches =
spatial_index[spatial_index.intersection(query_rect)]
spatial_index.intersection([[0, 0], [1, 0], [1, 1], [0, 0]])
```

e. Por fim a sua função também deve ter capacidades gráficas, similar as funções *plot* do Pandas.

```
df.plot("x", "y") # deve apresentar um gráfico de dispersão das colunas "x" e 'y"
```

f. O grupo deverá fazer a ligação do código C++ para o Python. Desejamos que após clonar o repositório, seja possível instalar com *'pip install .'* . Uma biblioteca

útil na hora de fazer essa conexão entre C++ e Python que recomendamos é a Python Boost. Nas monitorias será explicado a utilização com exemplos.

```
git clone http://github.com/jpocom/capivara.git
cd capivara
pip install .
```

g. Vocês possuem liberdade para a implementação das sintaxes das funções, os exemplos acima foram ilustrativos. Na sua implementação sempre considere a praticidade que um usuário teria em utilizar da sua biblioteca. Um exemplo seria o acesso das colunas com a sintaxe df["col"] ou df.col, em que apenas uma é necessária, mas a possibilidade de utilização das duas seria de maior comodidade. Também incentivamos implementações extras que tornem a biblioteca mais robusta.

3. **Arquivos:**
   a. Código com a implementação em C++ e o código que permite que a biblioteca seja importada em Python. O código deve ser formatado de acordo com as regras do CppLint e do PyLint.
   b. Relatório contendo:
      i. A descrição da implementação, quais estruturas de dados utilizadas, quais algoritmos utilizados, etc.
      ii. A documentação de como utilizar a biblioteca em Python.
      iii. Os resultados, isto é, exemplos demonstrando as capacidades da biblioteca, testes para avaliar a velocidade da implementação.
      iv. As limitações e possíveis trabalhos futuros.
      v. A distribuição de tarefas dos integrantes do grupo.
   c. Vídeo de até no máximo 8 minutos apresentando uma visão geral do seu projeto. (Obs.: hospede o vídeo em algum site como o Youtube e adicione o link no relatório e repositório).
4. **Datas entregas:**
   a. 29/07 entrega primeira estrutura
   b. 19/08 entrega segunda estrutura
   c. 08/09 apresentação final
5. **Entrega:** Vocês devem acessar o seguinte link do Github Classroom e criar o repositório que será feito o projeto: https://classroom.github.com/a/9h3kIxCP