```
# UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO -- UFRPE
# DEPARTAMENTO DE ESTATISTICA E INFORMATICA -- DEINFO
# SISTEMAS DE INFORMAÇÃO -- BSI

# ALGORITMOS E ESTRUTURA DE DADOS -- 2018.02
# PROF. TIAGO FERREIRA
# ALUNO ALDEMAR S R FILHO
```

RELATÓRIO DO PROJETO

SISTEMA DE BIBLIOTECA

RESUMO DE FUNÇÕES, CLASSES E MÉTODOS

```
### FUNCOES ###
```

```
from libClasses import Book, User, RedWhiteTree
from time import sleep
import os
```

Importação de módulos criados e bibliotecas auxiliares (**sleep** e **os**).

```
def cleanScreen():
```

Função que realiza a limpeza da tela. **os.system** executa os comandos passados no shell do sistema operacional.
Não recebe parâmetros.
Retorna **None**

```
def validateEntry(question, start, end):
```

Função que valida a entrada verificando o tipo de dado e se o valor está entre os limites fornecidos como parâmetro. Retorna None.
Parâmetros
question : **string**
start, end : **int**

Retorna valor do **input()** dentro da função

```
def pauseForRead(text, time=1.5):
```

Função que imprime a cadeia de caracteres recebiba e pausa a execução do código pelo tempo estipulado. Pausa padrão igual a 1,50 segundos.
Parâmetros
text : **string**
time : **int** (opcional | default = 1,50s)
Retorna **None**

```
def loadBooksDatabase():
def loadUsersDatabase():
```

Funções que carregam arquivos com dados para as bases necessárias, com intuito de facilitar os testes do código.

Não recebem parâmetros.

Retornam **userBase** & **booksBase**

```
def addUser(usersBase):
```

Função de inserção de novo usuário. Colhe os dados do usuário e chama método insertNode para incluir na arvore.
Parâmetros
usersBase : **obj** (Class **RedWhiteTree**)

Retorna **None**

```
def addBook(booksBase):
```

Função de inserção de novo usuário. Colhe os dados do usuário e chama e chama método **insertNode** para incluir na arvore.
Parâmetros
booksBase : **obj** (Class **RedWhiteTree**)

Retorna **None**

```
def listBooks(booksBase):
```

Função de listagem de livros cadastrados no sistema.
Parâmetros
booksBase : **obj** (Class **RedWhiteTree**)

Retorna **None**

```
def borrowedBooks(usersBase, loggedUser=None):
```

Função de listagem de livros emprestados pelo usuário logado no sistema.
Parâmetros
usersBase : **obj** (Class **RedWhiteTree**)
loggedUser : **obj** (Class **User**)

Retorna **None**

```
def withdrawBook(usersBase, booksBase, loggedUser):
```

Função de empréstimo de livros. Verifica se o usuario esta cadastrado, caso sim, verifica se o livro está cadastrado. Se estiver, verifica o numero máximo de emprestimos (máx. 5). Estando

tudo OK, realiza a atualização das informaçõess do usuário e livro.
    Parâmetros
    booksBase : **obj** (Class **RedWhiteTree**)
    userssBase : **obj** (Class **RedWhiteTree**)
    loggedUser : **obj** (Class **User**)

    Retorna **None**

```
def returnBook(usersBase, booksBase, loggedUser):
```

    Função de devolução de livros. Verifica se o usuario esta cadastrado, caso sim, verifica se o livro está cadastrado. Se estiver tudo OK, realiza a atualiação das informações do usuário e livro.
    Parâmetros
    booksBase : **obj** (Class **RedWhiteTree**)
    userssBase : **obj** (Class **RedWhiteTree**)
    loggedUser : **obj** (Class **User**)

    Retorna **None**

```
def removeUser(usersBase):
```

    Função de remoção do usuário da base de dados. Verifica se o usuario esta cadastrado, caso sim, confirma o usuário solicitando senha. Estando tudo OK, realizada a remoção.
    Parâmetros
    userssBase : **obj** (Class **RedWhiteTree**)

    Retorna **None**

```
def removeBook(booksBase):
```

    Função de remoção de livros da base de dados. Exibe a lista de livros cadastrados (inOrder), verifica se o livro escolhido está cadastrado, caso sim, se o livro não tiver copias emprestadas, realiza a remoção.
    Parâmetros
    booksBase : **obj** (Class **RedWhiteTree**)

    Retorna **None**

```
def userLogin(usersBase):
```

Função que realiza o login de usuários. Verifica se o usuario está cadastrado, caso sim, confirma o usuário solicitando senha. Estando tudo OK, vai para o menu de usuário.

Parâmetros

usersBase : **obj** (Class **RedWhiteTree**)

Retorna **UserMenu** (function)

```
def adminLogin(usersBase):
```

Função que realiza o login de administrador. Verifica se o usuario está cadastrado, caso sim, se ele é administrador. Sendo um, confirma o usuário solicitando senha. Estando tudo OK, retorna o menu de administrador.

Parâmetros

usersBase : **obj** (Class **RedWhiteTree**)

Retorna **adminMenu** (function)

```
def MainMenu():
def userMenu(loggedUser, usersBase, booksBase):
def adminMenu(loggedUser, booksBase):
```

Funções que apresentam os menus e simulam um switch/case através de um dicionário com o nome das funções e parâmetros de cada uma.

Parâmetros

Não recebe parâmetros.

loggedUser : **obj** (Class **User**)
usersBase : **obj** (Class **RedWhiteTree**)
booksBase : **obj** (Class **RedWhiteTree**)
loggedUser : **obj** (Class **User**)
booksBase : **obj** (Class **RedWhiteTree**)

Retornam **funções**

CLASSES

```
class User:

     Classe de objetos usuário.
     Possui um contador global da classe.


def __init__(self, name, password, admin=False):

     Método construtor que faz a instanciação de cada objeto User.
Possui um contador da classe.
     Atributo books inicializado como dicionário (dict).

     Parâmetros

     self : obj (Class User)
     name : string
     password : string
     admin : boolean (optional | default = False)


def __str__(self):

     Substituição do método especial __str__ para adequar a
impressão do objeto a necessidade de apresentação.
     Parâmetros

     self : obj (Class User)


@property
def key(self):  Propriedade chave (int)
@property
def name(self):  Propriedade name
@property
def password(self):  Propriedade password
@property
def isAdmin(self):  Propriedade  isAdmin (bool)
@property
def loans(self):  Propriedade loans (int)
@property
def booksID(self): Propriedade booksID (dict.keys)
@property
def bookNames(self): Propriedade bookNames (dict.values)
@property
def books(self): Propriedade books (dict (key, value)
```

Métodos **getter** para os atributos da classe utilizando o decorator **@property** para facilitar o acesso e escrita do código.
Parâmetros

self : **obj** (Class **User**)

```
@name.setter
def name(self, name):
@password.setter
def password(self, password):
@books.setter
def books(self, bookID, title):
```

Métodos **setter** para os atributos da classe utilizando o decorator **@atributo.setter** para facilitar o acesso e escrita do código.
Parâmetros

self : **obj** (Class **User**)
atributos : **string**

Não há método setter para todos os atributos, visto que que a alguns devem permanecer imutáveis.

```
    def returnBook(self, bookID=None, title=None):
```

Método que faz as operações de devolução de livros. Verifica se o livro está com o usuário, se sim, procede com a devolução e atualização de emprestimos e limite maximo de emprestimos.
Parâmetros

self : **obj** (Class **User**)
bookID : **string**
title : **string**

```
    def withdrawBook(self, bookID, title):
```

Método que faz as operações de empréstimo de livros. Verifica se o limite maximo de emprestimos foi atingido, caso não atualiza os atributos e inclui o livro.
Parâmetros

self : **obj** (Class **User**)
bookID : **string**
title : **string**

```
class Book:

     Classe de objetos livro.
     Possui um contador global da classe.


def __init__(self, title, copies=1):

     Método construtor que faz a instanciação de cada objeto Book.
Possui um contador da classe.
     Parâmetros

     self : obj (Class Book)
     title : string
     copies : int (optional | default = 1)


def __str__(self):

     Substituição do método especial __str__ para adequar a
impressão do objeto a necessidade de apresentação.
     Parâmetros

     self : obj (Class Book)


@property
def key(self):
@property
def title(self):
@property
def copies(self):
@property
def borrowedCopies(self):
@property
def availableCopies(self):
@property
def isAvailable(self):

     Métodos getter para os atributos da classe utilizando o
decorator @property para facilitar o acesso e escrita do código.
     Parâmetros

     self : obj (Class Book)


@isAvailable.setter
def isAvailable(self, status):
@borrowedCopies.setter
def borrowedCopies(self, returned):
```

Métodos **setter** para os atributos da classe utilizando o decorator **@atributo.setter** para facilitar o acesso e escrita do código.
Parâmetros

self : **obj** (Class **Book**)
atributos : **string**

Não há método setter para todos os atributos, visto que que a alguns devem permanecer imutáveis.

```
def __updateAvailableCopies(self):
```

Método que faz a atualização do numero de cópias disponíveis.
Parâmetros

self : **obj** (Class **Nook**)

```
def returnBook(self):
```

Método que faz a atualização do numero de cópias emprestadas.
Parâmetros

self : **obj** (Class **Book**)

```
class RWTNoneNode:
class RWTNode:
```

Duas classes basicamente iguais, diferenciando-se apenas no construtor.
Classe de objetos **RWTNoneNode**.
Classe de objetos **RWTNode**.

```
def __init__(self):
```

Método construtor que faz a instaciação de cada objeto **RWTNoneNode**. Os ponteiros apontam para si mesmo.
Paramêtros

self : **obj** (Class **RWTNoneNode**)

```
def __init__(self, data, color):
```

Método construtor que faz a instanciação de cada objeto RWTNode.
Parâmetros

self : **obj** (Class **RWTNode**)
data : **string**
color : **string**

```
@property
def color(self):
@property
def data(self):
@property
def father(self):
@property
def leftSon(self):
@property
def rightSon(self):
```

Métodos **getter** para os atributos da classe utilizando o decorator **@property** para facilitar o acesso e escrita do código.
Parâmetros

self : **obj** (Class **RWTNode**)

```
@color.setter
def color(self, color):
@data.setter
def data(self, data):
@father.setter
def father(self, value):
@leftSon.setter
def leftSon(self, value):
@rightSon.setter
def rightSon(self, value):
```

Método **setter** para os atributos da classe utilizando o decorator **@atributo.setter** para facilitar o acesso e escrita do código.
Parâmetros

self : **obj** (Class **RWTNode**)
color : **string**
value : **obj** (Class **RWTNode**)
data : **obj** (Class **User || Book**)

Não há método setter para todos os atributos, visto que que a alguns devem permanecer imutáveis.

```
class RedWhiteTree():
```

Classe de objetos **RWTNode**.
Possui uma instância de **RWTNoneNode** global da classe.

```
def __init__(self):
```

Método construtor que faz a instanciação de cada objeto **RedWhiteTree**.
Parâmetros

self : **obj** (Class **User**)

```
def __str__(self):
```

Substituição do método especial **__str__** para adequar a impressão do objeto a necessidade de apresentação (**in Order**).
Parâmetros

self : **obj** (Class **RedWhiteTree**)

```
@property
def root(self):
```

Método **getter** para o atributo **root** utilizando o decorator **@property** para facilitar o acesso e escrita do código.
Parâmetros

self : **obj** (Class **RedWhiteTree**)

```
@root.setter
def root(self, value):
```

Método **setter** para o atributos **root** utilizando o decorator **@root.setter** para facilitar o acesso e escrita do código.
Parâmetros

self : **obj** (Class **RedWhiteTree**)
value : **obj** (Class **RWTNode**)

```
def isEmpty(self):
```

Método que retorna verdadeiro se a árvore estiver vazia.
Parâmetros

self : **obj** (Class **RedWhiteTree**)

```
def isOnlyRoot(self):
```

Método que retorna verdadeiro se a árvore possuir apenas a raiz.
Parâmetros

self : **obj** (Class **RedWhiteTree**)

```
def insertNode(self, value):
```

Método de inserção de nós na RWT.
Parâmetros

self : **obj** (Class **RedWhiteTree**)
value : **obj** (Class **RWTNode**)

Localiza um espaço disponível obedecendo a regra da árvore binária. No final chama a função de correção **insertFix** para que as propriedades da árvore **RedWhiteTree** se mantenham.

```
def insertFix(self, node):
```

Método de balanceamento da árvore após inserções.

Executa um loop de verificação dos casos e executa as rotações necessárias até o pai do nó ter a cor **vermelha**.
Parâmetros

self : **obj** (Class **RedWhiteTree**)
value : **obj** (Class **RWTNode**)

```
def transplantNode(self, node, node2):
```

Método auxiliar para evitar repetição de código. Essencialmente esse método realiza a troca de posição de node com node2. (Transplanta node2 para posição de node)
Parâmetros

self : **obj** (Class **RedWhiteTree**)

node : **obj** (Class **RWTNode**)
node2 : **obj** (Class **RWTNode**)

```
def removeNode(self, key):
```

Método de remoção de nós na RWT.
Parâmetros

self : **obj** (Class **RedWhiteTree**)
value : **obj** (Class **RWTNode**)

Identifica qual caso o nó se encaixa e realiza as operações de troca de ponteiros necessárias a remoção do nó. No final chama a função de correção **removesFix** para que as propriedades da árvore **RedWhiteTree** se mantenham.

```
def removesFix(self, node):
```

Método de balanceamento da árvore após remoções.

Executa um loop de verificação dos casos e executa as rotações necessárias até o nó ter a cor **branca**.
Parâmetros

self : **obj** (Class **RedWhiteTree**)
value : **obj** (Class **RWTNode**)

```
def searchValue(self, value):
```

Método de busca na árvore através de um **node** fornecido.

Executa uma busca iterativamente seguindo as regras de árvore binária até coincidir com o node fornecido ou até percorrer todo o caminho.
Parâmetros

self : **obj** (Class **RedWhiteTree**)
value : **obj** (Class **RWTNode**)

Retorna **obj** (Class **RWTNode**)

```
def searchKey(self, key):
```

Método de busca na árvore através de um **valor** fornecido.

Executa uma busca iterativamente seguindo as regras de árvore binária até coincidir com o valor fornecido ou até percorrer todo o caminho.
Parâmetros

self : **obj** (Class **RedWhiteTree**)
key : **string**

Retorna **obj** (Class **RWTNode**)

```
def order(self, node=None, method="ino"):
```

Método orquestrador de “passeios” na (sub)árvore.

Com base nos parâmetros recebidos chama o método adequado.
Parâmetros

self : **obj** (Class **RedWhiteTree**)
node : **obj** (Class **RWTNode**) (optional | default = None → **self.root**)
method : **string** (optional | default=**ino**)

Retorna **None**

```
def inOrderRecEngine(self, node):
```

Método de percorrer e exibir na tela a (sub)árvore fornecida obedecendo a sequência:
Pai/raiz,filho direito e filho esquerdo

Parâmetros

self : **obj** (Class **RedWhiteTree**)
node : **obj** (Class **RWTNode**)

Retorna **None**

```
def preOrderRecEngine(self, node):
```

Método de percorrer e exibir na tela a (sub)árvore fornecida obedecendo a sequência:
Pai/raiz, filho direito e filho esquerdo

Parâmetros

self : **obj** (Class **RedWhiteTree**)
node : **obj** (Class **RWTNode**)

Retorna **None**

```
def posOrderRecEngine(self, node):
```

Método de percorrer e exibir na tela a (sub)árvore fornecida obedecendo a sequência:
Filho direito, filho esquerdo e pai/raiz

Parâmetros

self : **obj** (Class **RedWhiteTree**)
node : **obj** (Class **RWTNode**)

Retorna **None**

```
def maximum(self, node=None):
```

Método de busca da máxima chave da (sub)árvore.

Parâmetros

self : **obj** (Class **RedWhiteTree**)
node : **obj** (Class **RWTNode**) (optional | default = None → **self.root**)

Retorna **Node** (Class **RWTNode**)

```
def minimum(self, node=None):
```

Método de busca da mínima chave da (sub)árvore.

Parâmetros

self : **obj** (Class **RedWhiteTree**)
node : **obj** (Class **RWTNode**) (optional | default = None → **self.root**)

Retorna **Node** (Class **RWTNode**)

```
def predecessor(self, value=None, node=None):
```

Método de busca do predecessor da chave na (sub)árvore.

Parâmetros

```
     self : obj (Class RedWhiteTree)
     value : string (optional | default = None → node.key)
     node : obj (Class RWTNode) (optional | default = None →
self.root)

     Retorna Node (Class RWTNode)
```

```
def successor(self, value=None, node=None):

     Método de busca do predecessor da chave na (sub)árvore.

     Parâmetros

     self : obj (Class RedWhiteTree)
     value : string (optional | default = None → node.key)
     node : obj (Class RWTNode) (optional | default = None →
self.root)

     Retorna Node (Class RWTNode)
```

```
def rotateLeft(self, node):

     Método de rotação para esquerda entre par de nodes.

     Parâmetros

     self : obj (Class RedWhiteTree)
     node : obj (Class RWTNode)

     Retorna None
```

```
def rotateRight(self, node):

     Método de rotação para esquerda entre par de nodes.

     Parâmetros

     self : obj (Class RedWhiteTree)
     node : obj (Class RWTNode)

     Retorna None
```

---------- X ----------