Prova 3 - ED 18/12 (CÓDIGOS)

Criado em: December 16, 2023 8:41 PM

1. Ordenação (BubbleSort/InsertionSort/SelectionSort) CÓDIGO

BubbleSort

```
from random import randint

def buble_sort(array: list) -> list:
    for i in range(len(array) - 1, 0, -1):
        for j in range(0, i):
            if array[j] > array[j+1]:
                array[j], array[j+1] = array[j+1], array[j]

    return array
```

InsertionSort

```
from random import randint

def insertion_sort(array: list):
    for i in range(1,len(array)):
        chave = array[i]
        j = i-1
        while j>=0 and chave < array[j]:
            array[j+1] = array[j]
            j -= 1
        array[j+1] = chave
    return array
```

SelectionSort

```
from random import randint

def selectionSort(array):
    for i in range(len(array)-1):
        min = i
        for j in range(i+1, len(array)):
            if(array[j] < array[min]):
                min = j
        array[min], array[i] = array[i], array[min]
                return array
```

2. MergeSort - CÓDIGO ou ORDENAR

```
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        sub_array1 = arr[:mid]
        sub_array2 = arr[mid:]
        mergeSort(sub_array1)
        mergeSort(sub_array2)
        i = j = k = 0
        while i < len(sub_array1) and j < len(sub_array2):
            if sub_array1[i] < sub_array2[j]:
                arr[k] = sub_array1[i]
                i += 1
            else:
                arr[k] = sub_array2[j]
                j += 1
            k += 1
        while i < len(sub_array1):
            arr[k] = sub_array1[i]
            i += 1
            k += 1

        while j < len(sub_array2):
            arr[k] = sub_array2[j]
            j += 1
            k += 1
```

3. QuickSort - CÓDIGO ou ORDENAR

```
def quicksort(array):
    if len(array) <= 1:
        return array
    else:
        pivo = array[0]
        a = [x for x in array[1:] if x <= pivo]
        b = [x for x in array[1:] if x > pivo]
        return quicksort(a) + [pivo] + quicksort(b)
```

4. Busca Binária - CÓDIGO ou BUSCA

```
def buscaBinaria(array, chave):
    inicio = 0
    fim = len(array)-1
    # Enquanto houver distância entre inicio e fim
    while (inicio <= fim ):
        meio = (inicio + fim)//2
        if ( chave < array[meio] ):
            fim = meio - 1 # Ajusta a pos. final
        elif ( chave > array[meio] ):
            inicio = meio + 1 # Ajusta a pos. inicial
        else:
            return meio # elemento foi encontrado!

    # Se finalizar o laço, percorreu todo o array e
    # não encontrou
    return -1
```

5. Tabela Hash - CÓDIGO/EXEMPLO

```
class Entry:

    def __init__(self, key, value):
        self.key = key
        self.value = value

class Hash:

    def __init__(self, size=10):
        self._size = size
        self._table = [None for x in range(size)] # vetor que guarda os dados

    def _rehash(self, i):
        return (i+1) % self._size

    def _hash(self, key):
        return key % self._size

    def put(self, key, value):
        h = self._hash(key)
        if self._table[h] is None:
            self._table[h] = Entry(key, value)
        else:
            if self._table[h].key == key:
                self._table[h].value = value
            else:
                index = 2
                while index < self._size:
                    r = self._rehash(index)
                    if self._table[r] is None:
                        self._table[r] = Entry(key, value)
                    else:
                        if self._table[r].key == key:
                            self._table[r].value = value

                    index += 1


    def get(self, key):
        entry = self._table[self._hash(key)]
        if entry:
            if entry.key == key:
                return entry.value
            else:
                index = 2
                while index < self._size:
                    r = self._rehash(index)
                    if self._table[r] is None:
                        return None
                    else:
                        if self._table[r].key == key:
                            return self._table[r].value
                    index += 1
        return None # key não existe

    def delete(self, key):
        entry = self._table[self._hash(key)]
        if entry:
            if entry.key == key:
                self._table[self._hash(key)] = None
            else:
                index = 2
                while index < self._size:
                    r = self._rehash(index)
                    if self._table[r] is None:
                        return None
                    else:
                        if self._table[r].key == key:
                            self._table[self._hash(key)] = None
                    index += 1
        return None # key não existe



if __name__ == "__main__":
    h = Hash()
    h.put(1, "Rodrigo")
    assert h._table[1].value == "Rodrigo"
    assert h.get(1) == "Rodrigo"
    h.put(2, "João")
    assert h.get(2) == "João"
    assert h._table[2].value == "João"
    h.put(1, "José")
    assert h.get(1) == "José"
    assert h._table[1].value == "José"
    h.put(11, "Maria")
    assert h._table[3].value == "Maria"
    assert h.get(11) == "Maria"
    assert h.get(1) == "José"
```

```
#Chaining HashTable -> Tratamento de colisões
class AbsentKeyException(Exception):
    def __init__(self,msg):
        super().__init__(msg)

class Entry:
    """Uma classe privada utilizada para encapsular os pares chave/valor"""
    __slots__ = ( "key", "value" )
    def __init__( self, entryKey, entryValue ):
        self.key = entryKey
        self.value = entryValue

    def __str__( self ):
        return "(" + str( self.key ) + ", " + str( self.value ) + ")"

class ChainingHashTable:
    def __init__(self, size=10):
        self.size = size
        # inicializa a tabela de dispersão com todos os elementos iguais a um
        # list vazio
        self.table = list([] for i in range(self.size))

    def hash(self, key):
        ''' Método que retorna a posição na tabela hashing conforme a chave.
            Aplicação do cálculo do hash modular
        '''
        return key % self.size

    def put(self, key, data):
        '''
        Adiciona um par chave/valor à tabela hash
        '''
        slot = self.hash(key)
        print(f'key {key} mapeada ao slot {slot}')

        for entry in self.table[slot]:
            if key == entry.key:
                entry.value = data
                return slot

        self.table[slot].append(Entry(key,data))
        return slot


    def get(self, key):
        '''
        Obtem o valor armazenado na entrada referente à chave "key"
        '''
        slot = self.hash(key)
        #print(f'key {key} at slot {slot}')
        for i in range(len(self.table[slot])):
            if key == self.table[slot][i].key:
                return self.table[slot][i].value
        else:
            raise AbsentKeyException(f'Chave {key} inexistente na tabela hash')


    def __str__(self):
        info = ""
        for items in self.table:
            # examina se o slot da tabela hash tem um list com elementos
            if items == None:
                continue
            for entry in items:
                info += str(entry)
        return info

    def __len__(self):
        count = 0
        for i in self.table:
            count += len(i)
        return count



    def keys(self):
        """Retorna uma lista com as chaves armazenadas na hashTable.
        """
        result = []
        for lst in self.table:
            if lst != None:
                for entry in lst:
                    result.append( entry.key )
        return result

    def contains( self, key ):
        """Return True se somente se a tabela hash tem uma entrada com a chave passada
           como argumento.
        """
        entry = self.__locate( key )
        return isinstance( entry, Entry )

    def __locate(self, key):
        '''
        Método que verifica se uma chave está presente na tabela hash e devolve a
        entrada correspondente quando a busca é realizada com sucesso
        '''
        slot = self.hash(key)
        for i in range(len(self.table[slot])):
            if key == self.table[slot][i].key:
                return self.table[slot][i]
        else:
            return None

    def remove(self, key):
        '''
        Método que remove a entrada correspondente à chave passada como argumento
        '''
        slot = self.hash(key)
        for i in range(len(self.table[slot])):
            if key == self.table[slot][i].key:
                entry = self.table[slot][i]
                del self.table[slot][i]
                return entry
        raise AbsentKeyException(f'Chave {key} não está presente na tabela hash')

    def displayTable(self):
        entrada = -1
        for items in self.table:
            entrada += 1
            print(f'Entrada {entrada:2d}: ', end='')
            if len(items) == 0:
                print(' None')
                continue
            for entry in items:
                print(f'[ {entry.key},{entry.value} ] ',end='')
            print()
```