



Algoritmos e Estruturas de Dados

TAD SortedMap (parte I)
2019/20

Package dataStructures: interfaces (TADs)

- **Dicionário Ordenado** (interface ***SortedMap***)
 - Dicionário, em que é necessário ordenação nas chaves.



TAD *SortedMap*

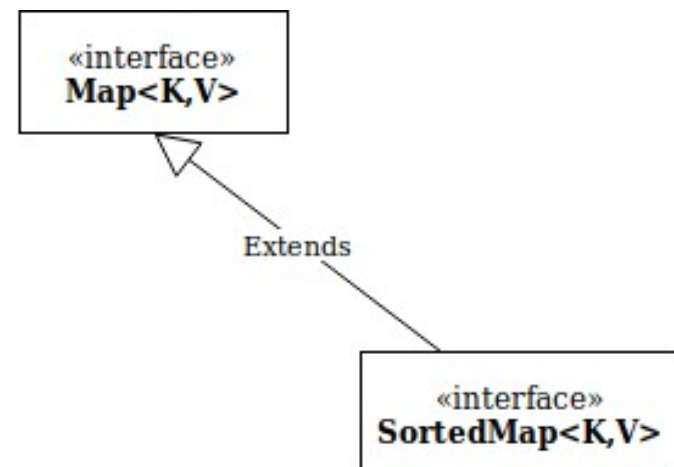
```
package dataStructures;
```

```
public interface SortedMap<K, V> extends Map<K,V>{
```

```
    // Returns the entry with the smallest key in the SortedMap.  
    // @throws NoSuchElementException if isEmpty()  
    Entry<K,V> minEntry( ) throws NoSuchElementException;
```

```
    // Returns the entry with the largest key in the SortedMap.  
    // @throws NoSuchElementException if isEmpty()  
    Entry<K,V> maxEntry( ) throws NoSuchElementException;
```

```
}
```



TAD *Map* (1)

```
package dataStructures;
```

```
public interface Map<K, V> {
```

```
// Returns true iff the map contains no entries
```

```
boolean isEmpty( );
```

```
// Returns the number of entries in the map.
```

```
int size( );
```

```
// Returns an iterator of the keys in the map.
```

```
Iterator<K> keys( ) throws NoSuchElementException;
```

```
// Returns an iterator of the values in the map.
```

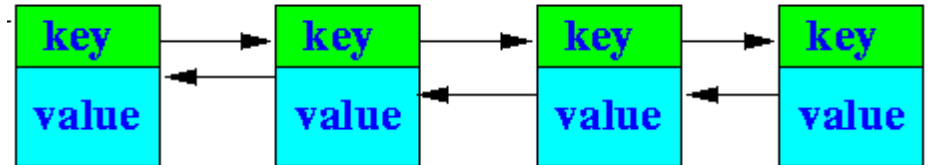
```
Iterator<V> values( ) throws NoSuchElementException;
```

```
// Returns an iterator of the entries in the map.
```

```
Iterator<Entry<K,V>> iterator() throws NoSuchElementException;
```

```
...
```

```
}
```



Estes iteradores dão
os elementos ordenados por chave.

TAD *Map* (2)

```
package dataStructures;
```

```
public interface Map<K, V> {
```

```
...
```

```
// If there is an entry in the map whose key is the specified key,  
// returns its value; otherwise, returns null.
```

```
V find( K key );
```

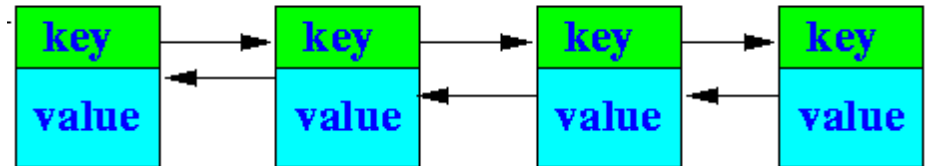
```
// If there is an entry in the map whose key is the specified key,  
// replaces its value by the specified value and returns the old value;  
// otherwise, inserts the entry (key, value) and returns null.
```

```
V insert( K key, V value );
```

```
// If there is an entry in the map whose key is the specified key,  
// removes it from the map and returns its value; otherwise, returns null.
```

```
V remove( K key );
```

```
}
```



Interface Entry

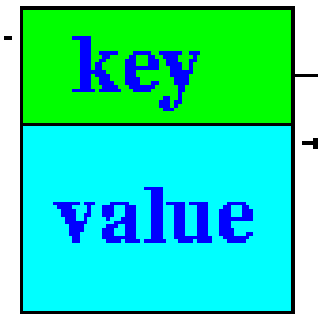
```
package dataStructures;

public interface Entry<K, V> {

    // Returns the key in the entry.
    K getKey( );

    // Returns the value in the entry.
    V getValue( );

}
```





TAD **SortMap**<K,V>

Opções de implementação

- A implementação pode ser realizada usando uma das seguintes estruturas de dados:
 - Vector ordenado;
 - Lista ligada (simples ou dupla) ordenada;
 - Tabela de dispersão mais lista ligada ordenada;
 - Tabela de dispersão mais Ordenação

TAD **SortedMap**<K,V> com **Vector ordenado** (1)

- Será uma boa implementação?

Key,value	Key,value	Key,value	Key,value	Key,value			
-----------	-----------	-----------	-----------	-----------	--	--	--

- Inserção ordenada; pesquisa binária ...

TAD **SortedMap**<K,V>

com **Vector ordenado** (2)

- Será uma boa implementação?

Key,value	Key,value	Key,value	Key,value	Key,value			
-----------	-----------	-----------	-----------	-----------	--	--	--

- Inserção ordenada; pesquisa binária ...

Operação		Caso Médio	Pior Caso
Inserção		$O(n)$	$O(n)$
Remoção		$O(n)$	$O(n)$
Pesquisa		$O(\log n)$	$O(\log n)$
Máximo e Mínimo		$O(1)$	$O(1)$
Percurso		$O(n)$	$O(n)$

TAD **SortedMap**<K,V>

com **Lista ligada (simples ou dupla) ordenada**

Nestas implementações é necessário implementar as classes **OrderedSinglyLL**<E> ou **OrderedDoublyLL**<E>. Estas classes seriam usadas para implementar a interface **SortedMap**<K,V>, onde o elemento **E** seria uma **Entry**<K,V>.

- Será uma boa implementação?



Classe OrderedDoublyLL<E> (1)

Interface do java.lang
- método "compareTo"

```
package dataStructures;
```

```
public class OrderedDoublyLL<E extends Comparable<E>>{
```

```
// Node at the head of the list.
```

```
protected DListNode<E> head;
```

```
// Node at the tail of the list.
```

```
protected DListNode<E> tail;
```

```
// Number of elements in the list.
```

```
protected int currentSize;
```

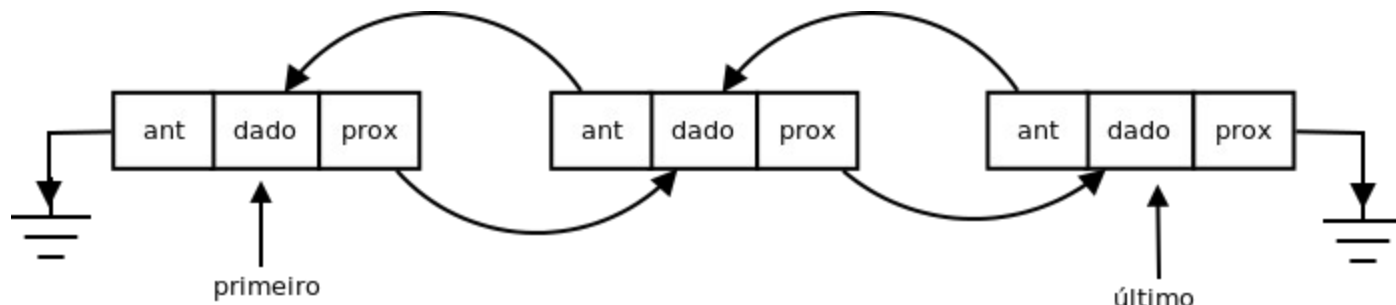
```
public OrderedDoublyLL() {
```

```
    head=null;
```

```
    tail=null;
```

```
    currentSize=0;
```

```
}
```



Classe OrderedDoublyLL<E> (2)

```
public boolean isEmpty() {  
    return currentSize==0;  
}
```

```
public int size() {  
    return currentSize;  
}
```

```
public Iterator<E> iterator() throws NoSuchElementException{  
    if (currentSize==0)  
        throw new NoSuchElementException("OrderedDoublyLL is empty.");  
    return new DoublyLLIterator<E>(head,tail);  
}  
...
```

Classe OrderedDoublyLL<E> (3)

```
public E minEntry() throws NoSuchElementException {  
    if (isEmpty())  
        throw new NoSuchElementException("OrderedDoublyLL is empty.");  
    return head.getElement();  
}
```

```
public E maxEntry() throws NoSuchElementException {  
    if (isEmpty())  
        throw new NoSuchElementException("OrderedDoublyLL is empty.");  
    return tail.getElement();  
}
```

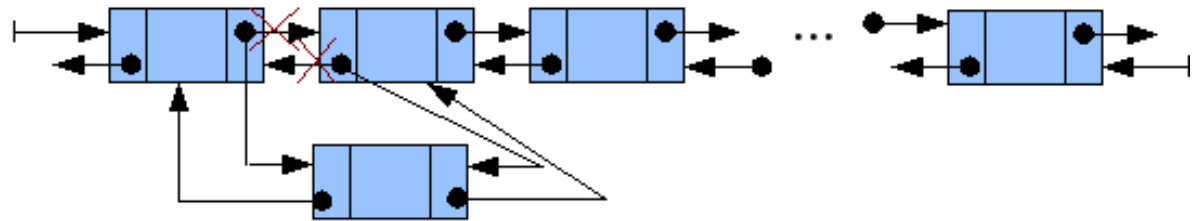
Classe OrderedDoublyLL<E> (4)

```
protected DListNode<E> findNode (E elem){
    DListNode<E> node=blue;
    while (node!=null && node.getElement().compareTo(elem)< 0){
        node=node.getNext();
    }
    return node;
}
```

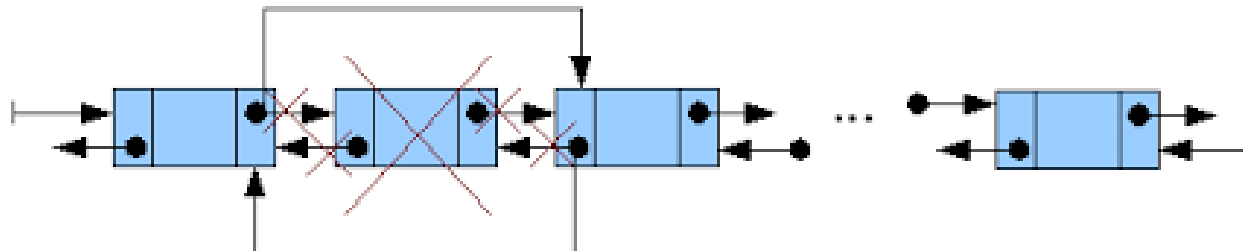
```
public E find(E elem) {
    DListNode<E> node = findNode(elem);
    if ( node == null || !node.getElement().equals(elem))
        return null;
    return node.getElement();
}
```

Classe OrderedDoublyLL<E> (5)

```
public E insert(E elem) {  
    // TODO  
    return null;  
}
```





```
public E remove(E elem) {  
    // TODO  
    return null;  
}
```



TAD **SortedMap**<K,V>

com **Lista duplamente ligada ordenada**

Operação	Caso Médio	Pior Caso
Inserção	$O(n)$	$O(n)$
Remoção	$O(n)$	$O(n)$
Pesquisa	$O(n)$	$O(n)$
Máximo e Mínimo 	$O(1)$	$O(1)$
Percurso 	$O(n)$	$O(n)$



TAD **SortedMap**<K,V>

com **Tabela de dispersão + Lista ordenada** (1)

Esta implementação pode ser realizada usando a classe **SepChainHashTable**<K,V> e a classe **OrderedDoublyLL**<K,V>.

- Será uma boa implementação?

TAD **SortedMap**<K,V>

com **Tabela de dispersão + Lista ordenada** (2)

Esta implementação pode ser realizada usando a classe **SepChainHashTable**<K,V> e a classe **OrderedDoublyLL**<K,V>.

- Será uma boa implementação?

Operação		Caso Médio	Pior Caso
Inserção		$O(n)$	$O(n)$
Remoção		$O(n)$	$O(n)$
Pesquisa		$O(1+\lambda)$	$O(n)$
Máximo e Mínimo		$O(1)$	$O(1)$
Percurso		$O(n)$	$O(n)$

TAD **SortedMap**<K,V>

com **Tabela de dispersão + Lista ordenada** (3)

Esta implementação pode ser realizada usando a classe **SepChainHashTable**<K,V> e a classe **OrderedDoublyLL**<K,V>.

- Não consigo melhorar esta implementação?
(sugestão: “ligar” ambas as estruturas com pequenas alterações – pensar na remoção)

Operação		Caso Médio	Pior Caso
Inserção		$O(n)$	$O(n)$
Remoção		$O(n)$	$O(n)$
Pesquisa	😊	$O(1+\lambda)$	$O(n)$
Máximo e Mínimo	😊	$O(1)$	$O(1)$
Iteração	😊	$O(n)$	$O(n)$



TAD **SortedMap**<K,V>

com **Tabela de dispersão + ordenação** (1)

Esta implementação pode ser realizada usando a classe **SepChainHashTable**<K,V> e algum algoritmo de ordenação quando for pedido os iteradores.




- Será uma boa implementação?

TAD **SortedMap**<K,V>

com **Tabela de dispersão + ordenação** (2)

Esta implementação pode ser realizada usando a classe **SepChainHashTable**<K,V> e algum algoritmo de ordenação quando for pedido os iteradores.

- Será uma boa implementação?

Operação		Caso Médio	Pior Caso
Inserção		$O(1+\lambda)$	$O(n)$
Remoção		$O(1+\lambda)$	$O(n)$
Pesquisa		$O(1+\lambda)$	$O(n)$
Máximo e Mínimo		$O(n)$	$O(n)$
Percurso		Depende do algoritmo de ordenação	Depende do algoritmo de ordenação



TAD SortedMap<K,V>

Implementação com as classes do Java

```
package dataStructures;
```

```
public class SortedMapWithJavaClass<K, V> implements SortedMap<K, V> {
```

```
protected java.util.SortedMap<K,V> elementos;
```

```
protected int capPrevista;
```

```
    public SortedMapWithJavaClass(int capPrevista) {
```

```
        elementos = new java.util.TreeMap<K,V>();
```

```
        this.capPrevista = capPrevista;
```

```
    }
```

```
    ...  
}
```

- O que será um **TreeMap**

