Recorrência I

$$T(n) = \begin{cases} a & n = 0 \; ou \; n = 1 \\ b.T(n-1) + c & n \geq 1 \; ou \; n \geq 2 \end{cases}$$

$$T(n) = \begin{cases} O(n) & b = 1 \\ O(b^n) & b > 1 \end{cases}$$

$com\ a \geq 0 \;\; b \geq 1 \;\; c \geq 1\ constantes$

Recorrência II a)

$$T(n) = \begin{cases} a & n = 0 \; ou \; n = 1 \\ b.T\left(\frac{n}{2}\right) + O(1) & n \geq 1 \; ou \; n \geq 2 \end{cases}$$

$$T(n) = \begin{cases} O(\log n) & b = 1 \\ O(n) & b = 2 \end{cases}$$

$com\ a \geq 0 \;\; b \geq 1 \; ou \; 2\ constantes$

Recorrência II (Master Theorem)

$$T(n) = \begin{cases} a & n = 0 \\ b.T\left(\frac{n}{c}\right) + O(n^k) & n > 0 \end{cases}, k \geq 0$$

$$T(n) = \begin{cases} O(n) & b < c \\ O(n \; log_c n) & b = c \\ O(n^{log_c n}) & b > c \end{cases}$$

$com\ a \geq 0 \;\; b \geq 1 \; c > 1\ constante$

==========

Tipos Abstractos de Dados
» Operações

Stack<E>

public interface Stack<E>

- boolean isEmpty ( )
- int size ( )
- E top ( ) throws EmptyStackException;
- void push(E elementlement)
- E pop( ) throws EmptyStackException;

Iterator

public interface Iterator<E>

- boolean hasNext ( );
- E next ( ) throws NoSuchElementException;
- void rewind ( );

TwoWayIterator<E>

public interface TwoWayIterator <E> extends Iterator <E>

Todos os métodos de Iterator<E>

- boolean hasPrevious ( )
- E previous ( )
- void fullForward ( )

Entry<K,V>

- V getKey ( )
- K getValue ( )

Queue<E>

public interface Queue<E>

- boolean isEmpty( )
- int size( )
- void enqueue(E elementlement)
- E dequeue( ) throws EmptyQueueException;

List<E>

public interface List<E>

- boolean isEmpty ( )
- int size ( )
- Iterator<E> iterator ( )
- E get(int position) throws InvalidPositionException;
- E getFirst ( ) throws EmptyListException;
- E getLast ( ) throws EmptyListException;
- int find(E element)
- void add(int position, E element) throws InvalidPositionException;
- void addFirst(E element) ;
- void addLast(E element) ;
- E remove(int position) throws InvalidPositionException;
- E removeFirst ( ) throws EmptyListException;
- E removeLast ( ) throws EmptyListException;
- boolean remove(E element)

DoublyLinkedList<E>

public interface DoublyLinkedList<E>

Todos os metodos de List

- DListNode <E> getNode (int position);
- void addMiddle (int position, E element);
- void removeFirstNode();
- void removeLastNode();
- void removeMiddleNode(DListNode <E> node);
- DListNode <E> findnode (E element);
- void append (DoublyLinkedList <E>);

DListNode

public interface DlistNode <E> implements Serializable

- DlistNode (E theElement, DlistNode <E> thePrevious, DlistNode <E> theNext);
- DListNode (E theElement);
- E getElement();
- DlistNode <E> getPrevious ( );
- DlistNode <E> getNext ( );
- void setElement (E newElement)
- void setPreviousElement (DlistNode <E> newPrevious)
- void setNextElement (DlistNode <E> newNext)

Dictionary

public interface Dictionary<K,V>

- boolean isEmpty ( )
- int size ( )
- V find(K key)
- V insert(K key, V value)
- V remove(K key)
- Iterator<Entry<K,V>> iterator ( )

Ordered Dictionary

public interface OrderedDictionary <K extends Comparable<K>, V> extends Dictionary<K,V>

Todos os métodos de Dictionary.

- Entry<K,V> minEntry ( ) throws EmptyDictionaryException;
- Entry<K,V> maxEntry ( ) throws EmptyDictionaryException;

Priority Queue

- boolean isEmpty ( )
- int size ( )
- Entry<K,V> minEntry ( )
- void insert(K key, V value)
- Entry<K,V> removeMin ( )

» Implementações

Stack
Queue
List

- Lista Ligada
- Simples
- Dupla
- Apenas com cabeça
- Com cabeça e cauda

Priority Queue

- Heap

Dictionary

- Hashtable
    - Separate Chaining (Dispersão Aberta)
    - Open Addressing (Dispersão Fechada)
    - Linear Probing (Sondagem linear)
    - Double Hashing (Dispersão dupla)

Ordered Dictionary

- Binary Search Tree
- Red-Black Tree
- AVL Tree

» Nós de EDs

BSTnode
- BSTNode(K key, V value)
- BSTNode(K key, V value, BSTNode<K,V> left, BSTNode <K,V> right)
- EntryClass<K,V> getEntry ( )
- K getKey ( )
- V getValue ( )
- BSTNode<K,V> getLeft ( )
- BSTNode<K,V> getRight ( )
- void setEntry(EntryClass<K,V> newEntry)
- void setEntry(K newKey, V newValue)
- void setKey(K newKey)
- void setValue(V newValue)
- void setLeft(BSTNode<K,V> newLeft)
- void setRight(BSTNode<K,V> newRight)
- boolean isLeaf ( )

DListNode
- DListNode(E theElement)
- DListNode(E theElement, DListNode<E> thePrevious, DListNode<E> theNext)
- E getElement ( )
- DListNode<E> getPrevious ( )
- DListNode<E> getNext ( )
- void setElement(E newElement)
- void setPrevious(DListNode<E> newPrevious)
- void setNext(DListNode<E> newNext)

AVLNode
- AVLNode(K key, V value)
- AVLNode(K key, V value, char balance, AVLNode<K,V> left, AVLNode<K,V> right)
- char getBalance ( )
- void setBalance(char newBalance)

RBNode
- RBNode(K key, V value)
- RBNode(K key, V value, boolean colour, RBNode<K,V> left, RBNode<K,V> right)
- boolean getColour ( )
- boolean isRed ( )
- boolean isBlack ( )
- void setColour(boolean newColour)
- void makeRed ( )
- void makeBlack ( )

Complexidades Temporais das Estruturas de Dados

São omitidas as seguintes estruturas de dados por terem complexidade constantE elementm todas as operações, em todos os casos: Stack com lista ligada, Queue com lista ligada. A lista ligada é omitida só porque sim!

» HashTable (Separate Chaining)

Assume-se que a resolução de colisões é feita com a OrderedDLL.

	Melhor caso	Pior Caso	Caso Esperado
Pesquisa	O(1)	O(n)	O(1 + λ)
Inserção	O(1)	O(n)	O(1 + λ)
Remoção	O(1)	O(n)	O(1 + λ)

» Binary Search Tree

	Melhor caso	Pior Caso	Caso Esperado
Pesquisa	O(1)	O(n)	O(lg(n))
Inserção	O(1)	O(n)	O(lg(n))
Remoção	O(1)	O(n)	O(lg(n))
Min/Max	O(1)	O(n)	O(lg(n))
Iterar *	O(n)	O(n)	O(n)

* Isto é o custo de iterar, não de obter o iterador!

» AVL/Red-Black Tree

	Melhor caso	Pior Caso	Caso Esperado
Pesquisa	O(1)	O(lg(n))	O(lg(n))
Inserção	O(lg(n))	O(lg(n))	O(lg(n))
Remoção	O(lg(n))	O(lg(n))	O(lg(n))
Min/Max	O(lg(n))	O(lg(n))	O(lg(n))
Iterar *$_{Text}$	O(n)	O(n)	O(n)

* Isto é o custo de iterar, não de obter o iterador!

» Binary Heap

	Melhor caso	Pior Caso	Caso Esperado
Criar c/ n elementos	O(n)	O(n)	O(n)
Pesquisa	O(1)	O(lg(n))	O(lg(n))
Mínimo	O(1)	O(1)	O(1)
Remover Mínimo	O(1)	O(lg(n))	O(lg(n))

» Ordenação

	Melhor caso	Pior Caso	Caso Esperado	Estável
Insertion	O(n)	O(n2)	O(n2)	Sim
Bubble	O(n2)	O(n2)	O(n2)	Sim
Selection	O(n2)	O(n2)	O(n2)	Não
Heap	O(n)	O(n lg(n))	O(n lg(n))	Não
Merge	O(n lg(n))	O(n lg(n))	O(n lg(n))	Sim
Quick	O(n lg(n))	O(n2)	O(n lg(n))	Não
Jogo	O(1)	O(1)	O(1)	?

Generalidades

» árvores Binárias

Uma árvore com n nós tem n + 1 sub-árvores vazias. Uma árvore com n nós tem altura entre [lg(n + 1) e n.

Uma árvore diz-se perfeitamente equilibrada se para todo o nó o número de nós no seu ramo esquerdo varia no máximo em uma unidade absoluta do número de nós no seu ramo direito.

Uma árvore diz-se equilibrada se para todo o nó a altura do seu ramo esquerdo varia no máximo em uma unidade absoluta da altura do seu ramo direito.

Percursos

- Prefixo | raiz, sub-árvore esquerda, sub-árvore direita.
- Infixo | sub-árvore esquerda, raiz, sub-árvore direita.
- Sufixo | sub-árvore esquerda, sub-árvore direita, raiz.

» Red-Black Tree

Mantém as seguintes propriedades:

1. Cada nó é preto ou vermelho.

2. A raiz é preta.

3. Filhos de nós vermelhos são pretos.

4. Todos os caminhos da raiz a qualquer árvore vazia têm o mesmo número de nós pretos.

Pilha: Pilha com disciplina LIFO

Complexidades em todos os casos

	Pilha em Vector	Pilha e LinkedList
isEmpty	O(1)	O(1)
Size	O(1)	O(1)
Top	O(1)	O(1)
Push	O(1)	O(1)
pop	O(1)	O(1)

Fila: fila com disciplina FIFO

Complexidades em todos os casos

	Fila em Vector	Fila e LinkedList
isEmpty	O(1)	O(1)
Size	O(1)	O(1)
enqueue	O(1)	O(1)
dequeue	O(1)	O(1)

Lista: acesso por posição

Complexidades da lista ligada com n elementos

	Melhor	Pior	Esperado
isEmpty, size	O(1)	O(1)	O(1)
getFirst, getLast, addFirst, addLast, removeFirst, removeLast	O(1)	O(1)	O(1)
get, add, remove(p)	O(1)	O(n)	O(n)
find, remove(e)	O(1)	O(n)	O(n)
iterator	O(1)	O(1)	O(1)

Dicionário Ordenado: acesso por chave

Complexidades quando o dicionário ordenado tem n entradas

	Melhor	Pior	Esperado
Pesquisa	O(1)	O(n)	O(log n)
Remoção	O(1)	O(n)	O(log n)
Renomeação	O(1)	O(n)	O(log n)
Minimo	O(1)	O(n)	O(log n)
Máximo	O(1)	O(n)	O(log n)
Obtenção Iterator	O(1)	O(n)	O(log n)
Percurso Ordenado	O(n)	O(n)	O(n)

Iterador bidireccional

	Melhor, Pior, Esperado
hasNext, next, rewind, hasPrevious, previous, fullForward	O(1)

AVL, Árvore sem restrição e Árvore Vermelha e Preta

	AVL Pior | Esp (1,44)	Arv. s/ restrição Pior | Esp	Red Black Pior | Esp (2)
Pesquisa	log n | log n	n | log n	log n | log n
Remoção	log n | log n	n | log n	log n | log n
Renomeação	log n | log n	n | log n	log n | log n
Minimo	log n | log n	n | log n	log n | log n
Máximo	log n | log n	n | log n	log n | log n
Percurso Ordenado	n	n	n